Foundations of Machine Learning

E. Le Pennec

MAP553 - Foundations of Machine Learning - Spring 2023

Outline

Outline

Outline

Major Influences

Four major influences act today:

- The formal theories of statistics
- Accelerating developments in computers and display devices
- The challenge, in many fields, of more and ever larger bodies of data
- The emphasis on quantification in an ever wider variety of disciplines

Major Influences - Tukey (1962)

Four major influences act today:

- The formal theories of statistics
- Accelerating developments in computers and display devices
- The challenge, in many fields, of more and ever larger bodies of data
- The emphasis on quantification in an ever wider variety of disciplines

- He was talking of Data Analysis.
- Data Mining, Machine Learning, Big Data, Data Science, Artificial Intelligence...

Source: D. Donoho, 50 Years of Data Science

Large Scale ML Is (Quite) Easy


```scala
def run(params: Params) {
  val conf = new SparkConf()
    .setAppName(s"BinaryClassification with $params")
  val sc = new SparkContext(conf)

  Logger.getRootLogger.setLevel(Level.WARN)

  val examples = MLUtils.loadLibSVMFile(sc, params.input).cache()

  val splits = examples.randomSplit(Array(0.8, 0.2))
  val training = splits(0).cache()
  val test = splits(1).cache()
  val numTraining = training.count()
  val numTest = test.count()
  println(s"Training: $numTraining, test: $numTest.")
  examples.unpersist(blocking = false)

  val updater = params.regType match {
    case L1 => new L1Updater()
    case L2 => new SquaredL2Updater()
  }

  val algorithm = new LogisticRegressionWithSGD()
    algorithm.optimizer
      .setNumIterations(params.numIterations)
      .setStepSize(params.stepSize)
      .setUpdater(updater)
      .setRegParam(params.regParam)
  val model = algorithm.run(training).clearThreshold()

  val prediction = model.predict(test.map(_.features))
  val predictionAndLabel = prediction.zip(test.map(_.label))

  val metrics = new BinaryClassificationMetrics(predictionAndLabel)
  val myMetrics = new MyBinaryClassificationMetrics(predictionAndLabel)

  println(s"Empirical CrossEntropy = ${myMetrics.crossEntropy()}.")
  println(s"Test areaUnderPR = ${metrics.areaUnderPR()}.")
  println(s"Test areaUnderROC = ${metrics.areaUnderROC()}.")

  sc.stop()
}
```

Spark Lightning-fast cluster computing

stackoverflow

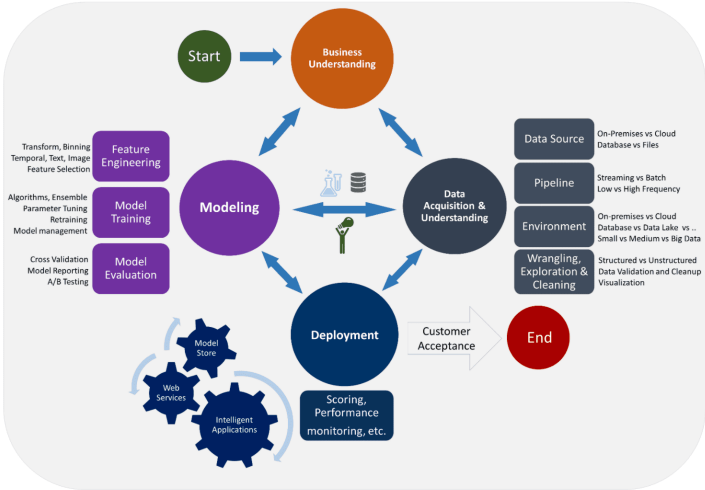
amazon web services

Example of off the shelves solution

- Algorithm implementation + copy/paste + cloud computing.
- Machine learning on an arbitrary large dataset!

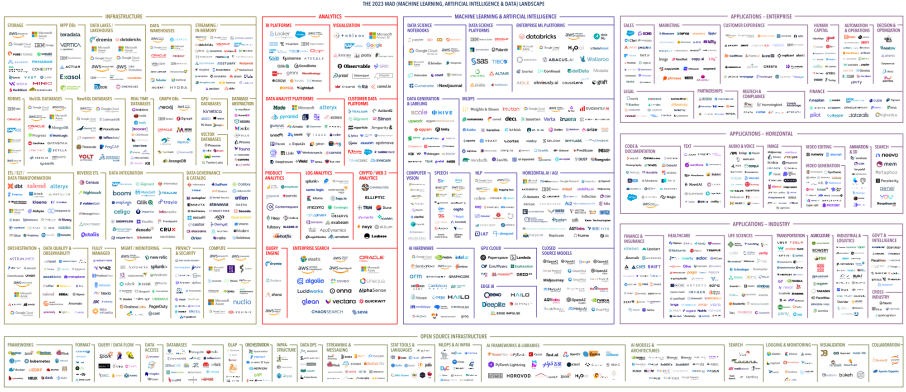
Sources: Unknown / 123rf.com / Unknown / Apache project / Stack Overflow / Amazon

Data Science Is (Quite) Complex!

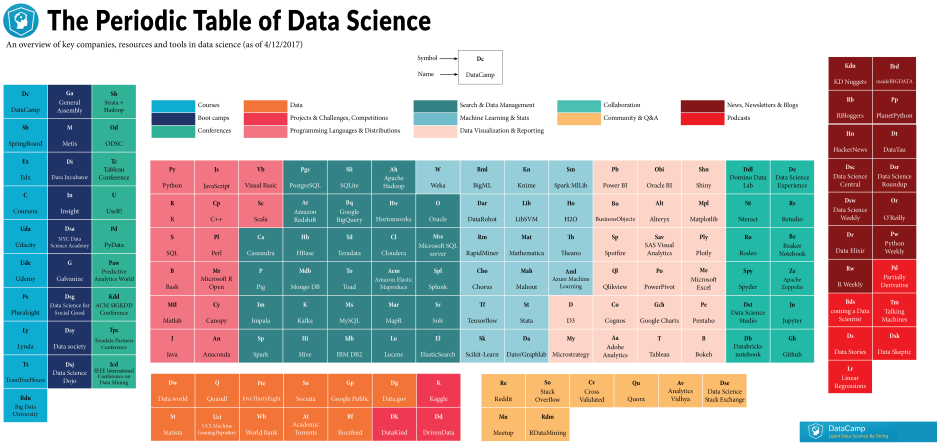


Source: Microsoft

Data Ecosystem Is (Quite) Complex!

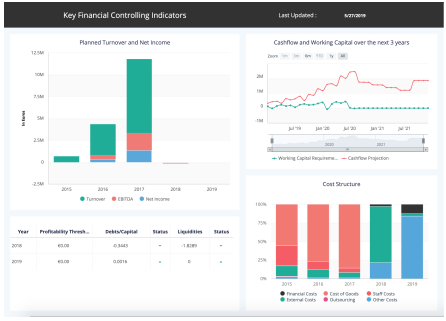


Data Ecosystem Is (Quite) Complex!



Source: DataCamp

Monthly KPI Dashboard



Year	Profitability Thresh...	Debts/Capital	Status	Liquidities	Status
2018	€0.00	0.3443	-	-1.8289	-
2019	€0.00	0.0016	+	0	+

Source: decisyon

KPI: Key Performance Index

Realtime Log Dashboard

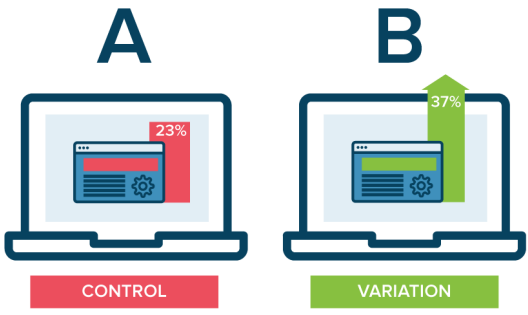
Source: edureka!

IT: Information Technology

On-demand Legal Document Generation

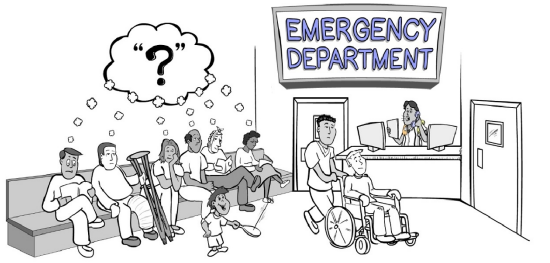
Source: Amicus Attorney

AB Testing



Source: Optimizely

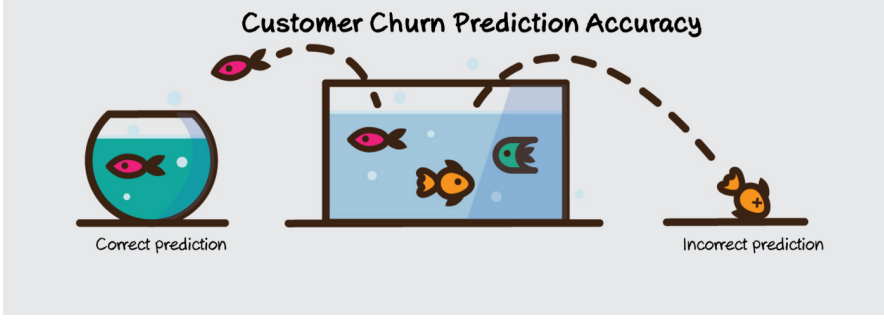
ER Waiting Time Prediction



Source: M. Evans

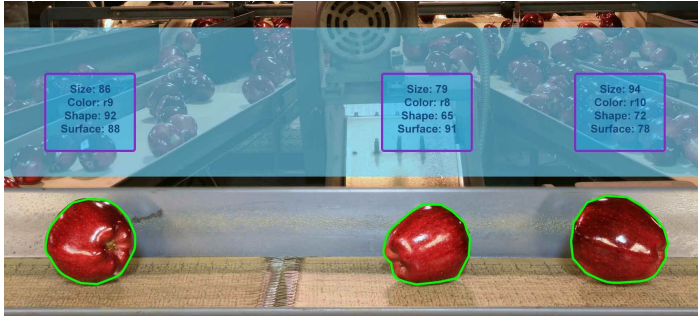
ER: Emergency Room

Weekly Churn Prediction



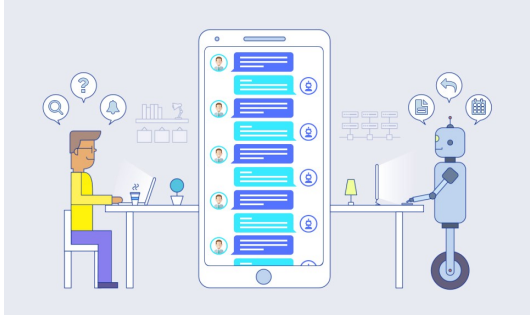
Source: Parthasarathy

Realtime Automatic Fruit Sorting



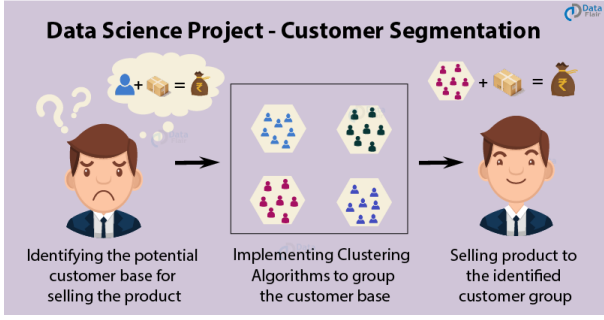
Source: BitRefine

Realtime Chatbot



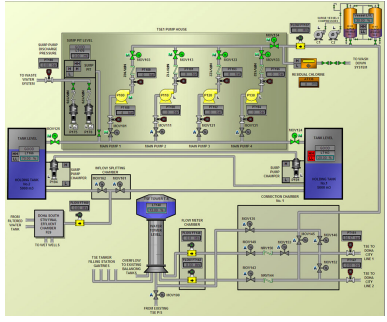
Source: M'Bufung

Customer Segmentation



Source: Data Flai

Realtime Anomaly Detection

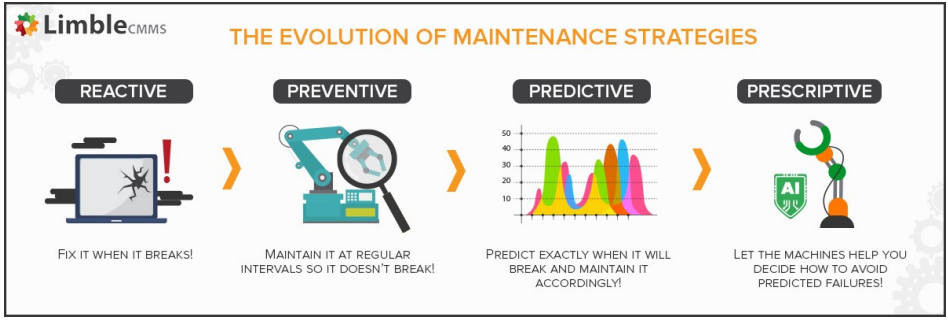


Source: Wikipedia

On-demand Fraud Detection

Source: MoneyKama

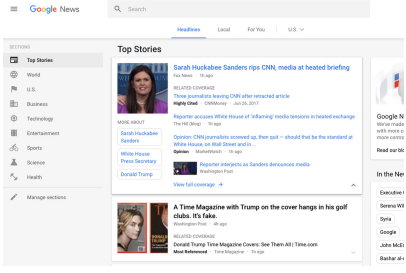
Prescriptive Maintenance

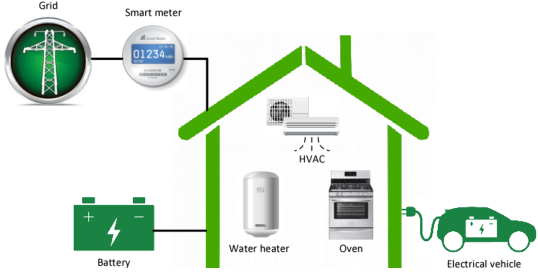


Source: Limble

Outline

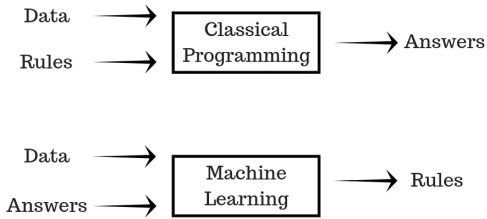
Machine Learning





Sources: MyCarDoesWhat.org/theverge.com/Zhiqiang Wan et al.

Machine Learning



A definition by Tom Mitchell

A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E.

Source: C. Barabasi

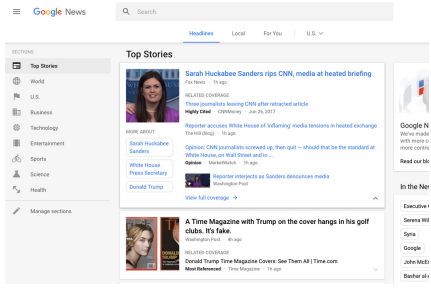
Object Detection

A detection algorithm:

- **Task**: say if an object is present or not in the image
- **Performance**: number of errors
- **Experience**: set of previously seen labeled images

Source: MyCarDoesWhat.org

Article Clustering

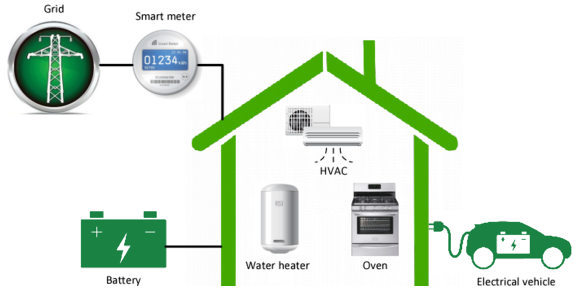


An article clustering algorithm:

- **Task**: group articles corresponding to the same news
- **Performance**: quality of the clusters
- **Experience**: set of articles

Source: theverge.com

Smart Grid Controler

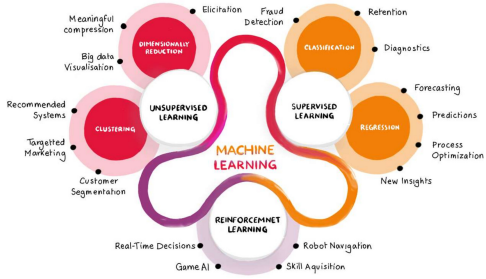


A controler in its sensors in a home smart grid:

- **Task**: control the devices
- **Performance**: energy costs
- **Experience**:
  - previous days
  - current environment and performed actions

Source: Zhiqiang Wan et al.

Three Kinds of Learning



Unsupervised Learning

- **Task:** Clustering/DR/Generative
- **Performance:** Quality
- **Experience:** Raw dataset (No Ground Truth)

Supervised Learning

- **Task:** Prediction/Classification
- **Performance:** Average error
- **Experience:** Good Predictions (Ground Truth)

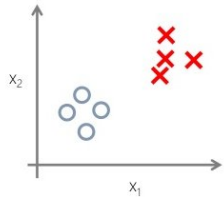
Reinforcement Learning

- **Task:** Actions
- **Performance:** Total reward
- **Experience:** Reward from env. (Interact. with env.)

Source: BCG

- **Timing:** Offline/Batch (learning from past data) vs Online (continuous learning)

Supervised and Unsupervised



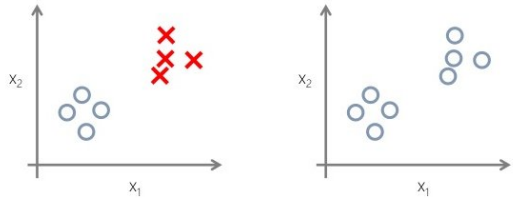
Supervised Learning (Imitation)

- **Goal:** Learn a function f predicting a variable Y from an individual $\underline{X}$.
- **Data:** Learning set with labeled examples $(\underline{X}_i, Y_i)$

- **Assumption:** Future data behaves as past data!
- **Predicting is not explaining!**

Source: KDnuggets

Supervised and Unsupervised



Supervised Learning (Imitation)

- **Goal:** Learn a function f predicting a variable Y from an individual $\underline{X}$.
- **Data:** Learning set with labeled examples $(\underline{X}_i, Y_i)$

- **Assumption:** Future data behaves as past data!
- **Predicting is not explaining!**

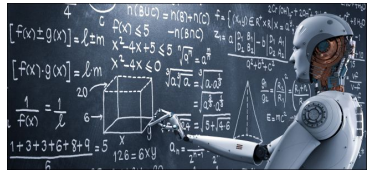
Unsupervised Learning (Structure Discovery)

- **Goal:** Discover a structure within a set of individuals $(\underline{X}_i)$.
- **Data:** Learning set with unlabeled examples $(\underline{X}_i)$

- Unsupervised learning is not a well-posed setting...

Source: KDnuggets

Machine Can and Cannot



Machine Can

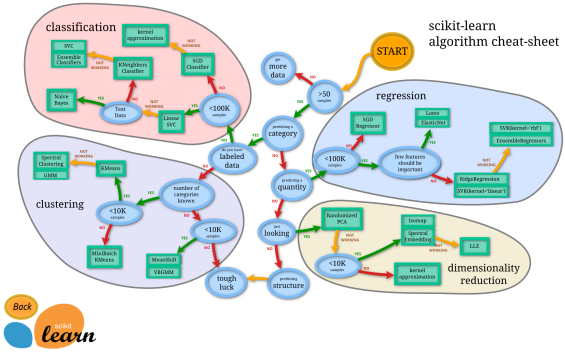
- Forecast (Prediction using the past)
- Detect expected changes
- Memorize/Reproduce
- Take a decision very quickly
- Learn from huge dataset
- Optimize a single task
- Replace/Help some humans

Machine Cannot

- Predict something never seen before
- Detect any new behaviour
- Create something brand new
- Understand the world
- Get smart really fast
- Go beyond their task
- Kill all humans

- Some progresses but still very far from the *singularity*...

Machine Learning

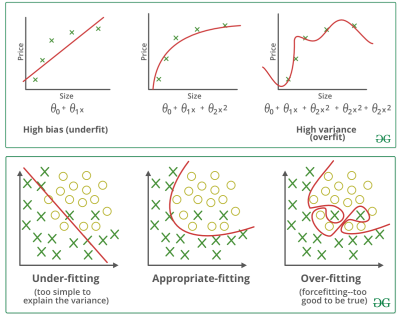


Machine Learning Methods

- Huge catalog of methods,
- Need to define the performance,
- Numerous tricks: feature design, hyperparameter selection...

Source: scikit-learn.org

Under and Over Fitting

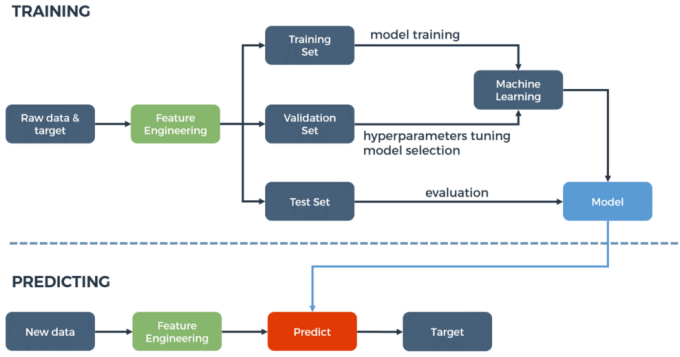


Finding the Right Complexity

- What is best?
  - A simple model that is stable but false? *(oversimplification)*
  - A very complex model that could be correct but is unstable? *(conspiracy theory)*
- Neither of them: tradeoff that depends on the dataset.

Source: geeksforgeeks.com

Machine Learning Pipeline



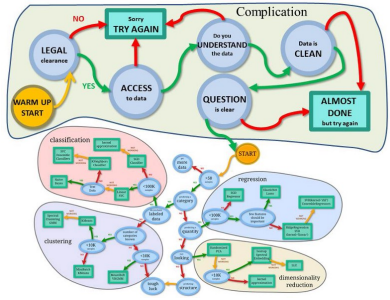
Learning pipeline

- Test and compare models.

- Deployment pipeline is different!

Source: CDiscount

Data Science $\neq$ Machine Learning

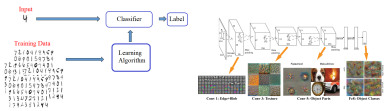


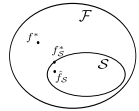
Main DS difficulties

- Figuring out the problem,
- Formalizing it,
- Storing and accessing the data,
- Deploying the solution,
- Not (always) the Machine Learning part!

Source: Ch. Bourguignat

MAP 553 - Foundation of Machine Learning - Goal





Goal

- Master the **statistical learning** framework and its challenges.
- Know the inner machinery of the most classical supervised and unsupervised **ML methods** in order to understand their strengths, limitations and connections.
- Understand some **optimization** tools used in ML as well as some **theoretical aspects** of ML.
- Not a course on practical tricks to use machine learning in a data product!

Evaluation

- A practical lab (5 pt)
- A final exam or a project (15 pt)

MAP 553 - Team

- Erwan Le Pennec

Erwan.Le-Pennec@polytechnique.edu

- Randal Douc

randal.douc@it-sudparis.eu

- El Mahdi El Mhamdi

elmahdi@elmhamdi.com

- Thierry Klein

thierry.klein@math.univ-toulouse.fr

- Edouard Oyallon

edouard.oyallon@lip6.fr

MAP 553 - Schedule

9 Lectures (8h30-10h30)

- Mon. 19/09: Statistical Learning: Introduction, Setting and Risk Estimation
- Mon. 26/09: ML Methods: Probabilistic Point of View
- Mon. 03/10: ML Methods: Optimization Point of View
- Mon. 10/10: Optimization: Gradient Descent Algorithms
- Mon. 17/10: ML Methods: Neural Networks and Deep Learning
- Mon. 24/10: ML Methods: Trees and Ensemble Methods
- Mon. 07/11: Unsupervised Learning: Dimension Reduction
- Mon. 14/11: Unsupervised Learning: Clustering
- Mon. 21/11: Statistical Learning: PAC-Bayesian Approach and Complexity Theory

- Mon. ??/12: **Exam** (?)

References

T. Hastie, R. Tibshirani, and J. Friedman.
The Elements of Statistical Learning.
Springer Series in Statistics, 2009

M. Mohri, A. Rostamizadeh, and A. Talwalkar.
Foundations of Machine Learning (2nd ed.)
MIT Press, 2018

A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd ed.)
O'Reilly, 2022

Ch. Giraud.
Introduction to High-Dimensional Statistics.
CRC Press, 2014

S. Bubeck.
Convex Optimization: Algorithms and Complexity.
Now Publisher, 2015

Outline

Supervised Learning

Supervised Learning Framework

- Input measurement $\underline{X} \in \mathcal{X}$
- Output measurement $Y \in \mathcal{Y}$.
- $(\underline{X}, Y) \sim \mathbb{P}$ with $\mathbb{P}$ unknown.
- **Training data** : $\mathcal{D}_n = \{(\underline{X}_1, Y_1), \ldots, (\underline{X}_n, Y_n)\}$ (i.i.d. $\sim \mathbb{P}$)

- Often
  - $\underline{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (classification)
  - or $\underline{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A **predictor** is a function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y} \text{ meas.}\}$

Goal

- Construct a **good** predictor $\widehat{f}$ from the training data.

- Need to specify the meaning of good.
- Classification and regression are almost the **same** problem!

Loss and Probabilistic Framework

Loss function for a generic predictor

- **Loss function**: $\ell(Y, f(\underline{X}))$ measures the goodness of the prediction of Y by $f(\underline{X})$
- Examples:
  - 0/1 loss: $\ell(Y, f(\underline{X})) = \mathbf{1}_{Y \neq f(\underline{X})}$
  - Quadratic loss: $\ell(Y, f(\underline{X})) = |Y - f(\underline{X})|^2$

Risk function

- Risk measured as the average loss for a new couple:

$$\mathcal{R}(f) = \mathbb{E}_{(\underline{X},Y)\sim\mathbb{P}}[\ell(Y, f(\underline{X}))]$$

- Examples:
  - 0/1 loss: $\mathbb{E}[\ell(Y, f(\underline{X}))] = \mathbb{P}(Y \neq f(\underline{X}))$
  - Quadratic loss: $\mathbb{E}[\ell(Y, f(\underline{X}))] = \mathbb{E}\left[|Y - f(\underline{X})|^2\right]$

- **Beware:** As $\widehat{f}$ depends on $\mathcal{D}_n$, $\mathcal{R}(\widehat{f})$ is a random variable!

Best Solution

- The best solution $f^\star$ (which is independent of $\mathcal{D}_n$) is

$$f^\star = \arg\min_{f \in \mathcal{F}} \mathcal{R}(f) = \arg\min_{f \in \mathcal{F}} \mathbb{E}[\ell(Y, f(\underline{X}))] = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{\underline{X}}\Big[\mathbb{E}_{Y|\underline{X}}[\ell(Y, f(\underline{X}))]\Big]$$

Bayes Predictor (explicit solution)

- In binary classification with $0-1$ loss:

$$f^\star(\underline{X}) = \begin{cases} +1 & \text{if } \mathbb{P}(Y=+1|\underline{X}) \geq \mathbb{P}(Y=-1|\underline{X}) \\ & \Leftrightarrow \mathbb{P}(Y=+1|\underline{X}) \geq 1/2 \\ -1 & \text{otherwise} \end{cases}$$

- In regression with the quadratic loss

$$f^\star(\underline{X}) = \mathbb{E}[Y|\underline{X}]$$

Issue: Solution requires to **know** $\mathbb{E}[Y|\underline{X}]$ for all values of $\underline{X}$!

Goal

Machine Learning

- Learn a rule to construct a **predictor** $\widehat{f} \in \mathcal{F}$ from the training data $\mathcal{D}_n$ s.t. **the risk** $\mathcal{R}(\widehat{f})$ is **small on average** or with high probability with respect to $\mathcal{D}_n$.

- In practice, the rule should be an algorithm!

Canonical example: Empirical Risk Minimizer

- One restricts f to a subset of functions $\mathcal{S} = \{f_\theta, \theta \in \Theta\}$
- One replaces the minimization of the average loss by the minimization of the empirical loss

$$\widehat{f} = f_{\widehat{\theta}} = \underset{f_\theta, \theta \in \Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f_\theta(\underline{X}_i))$$

- Examples:
  - Linear regression
  - Linear classification with

$$\mathcal{S} = \{\underline{x} \mapsto \operatorname{sign}\{\underline{x}^\top \beta + \beta^{(0)}\} \, / \beta \in \mathbb{R}^d, \beta^{(0)} \in \mathbb{R}\}$$

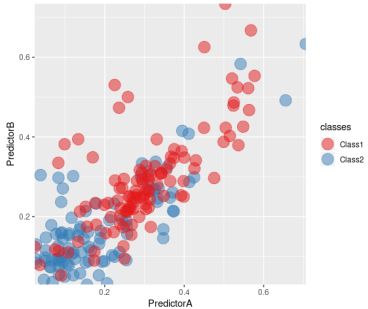
Outline

Example: TwoClass Dataset

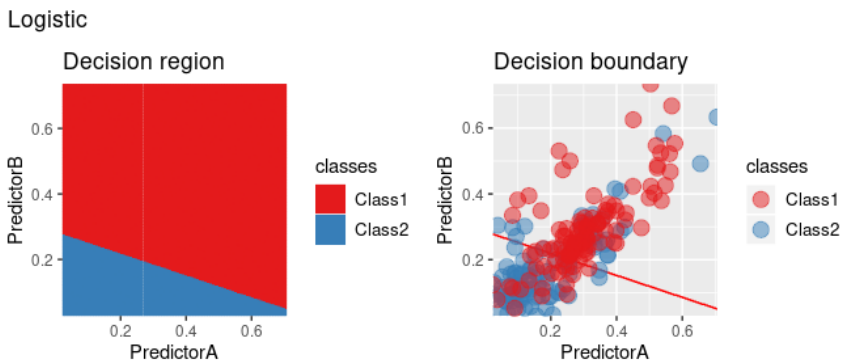
Synthetic Dataset

- Two features/covariates.
- Two classes.

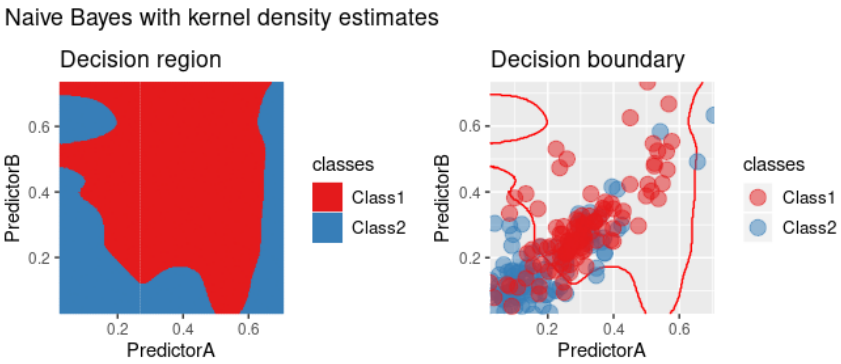
- Dataset from *Applied Predictive Modeling*, M. Kuhn and K. Johnson, Springer
- Numerical experiments with `R` and the {`caret`} package.



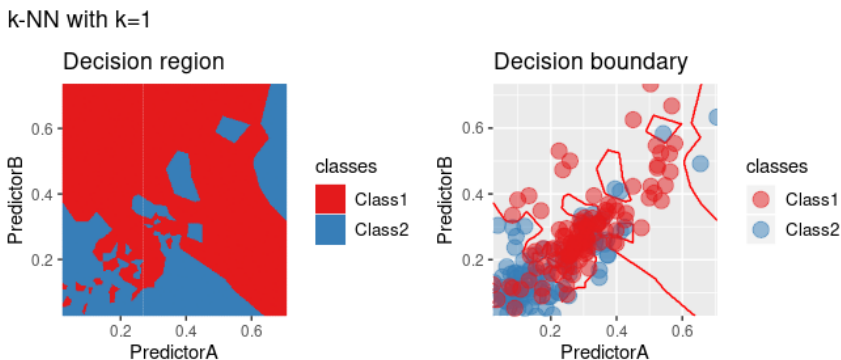
Example: Linear Discrimination



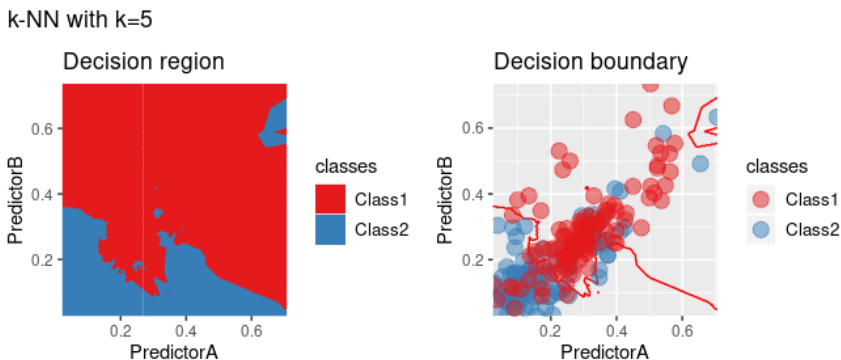
Example: More Complex Model



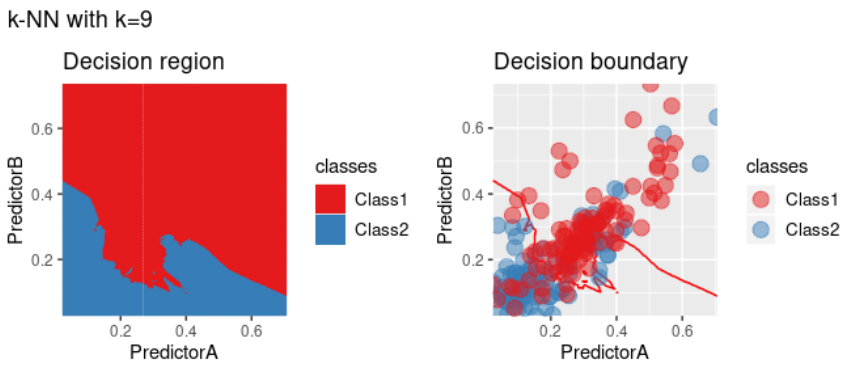
Example: KNN


k-NN with k=1
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

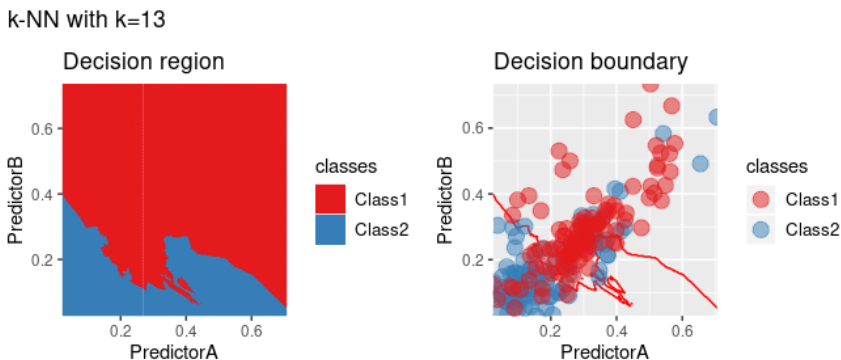
Example: KNN


k-NN with k=5
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

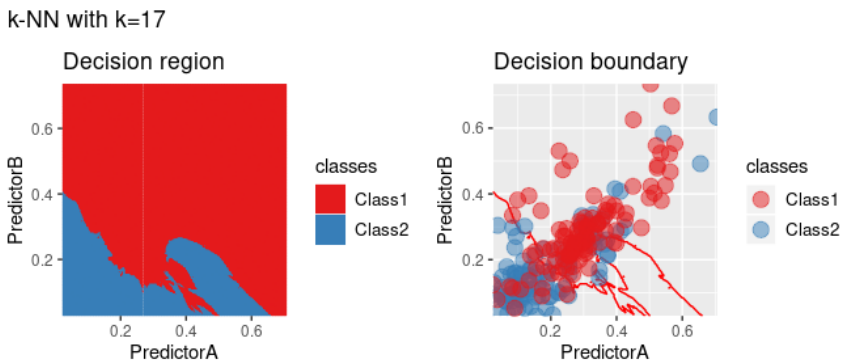
Example: KNN


k-NN with k=9
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

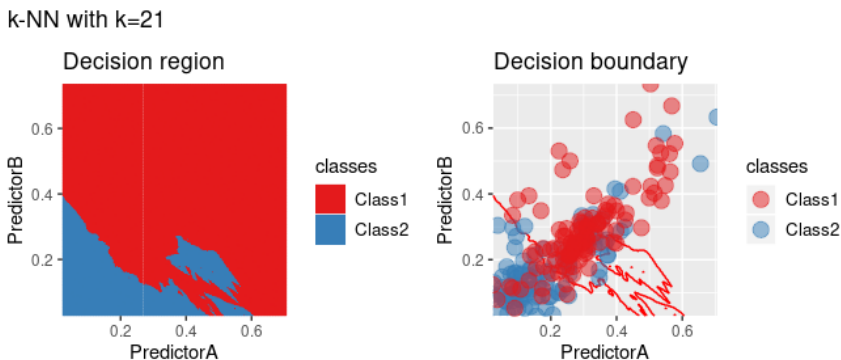
Example: KNN


k-NN with k=13
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

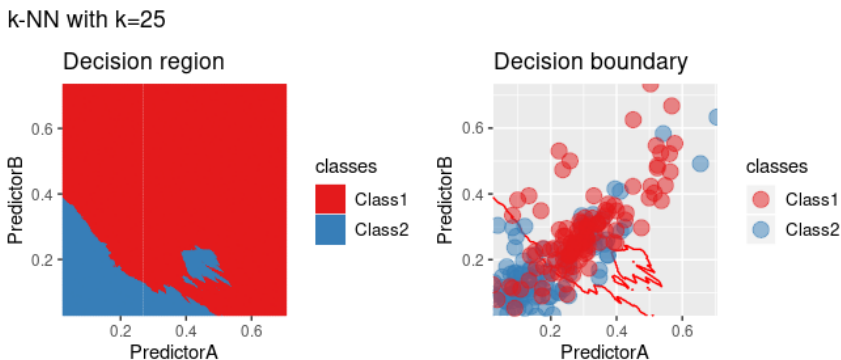
Example: KNN


k-NN with k=17
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

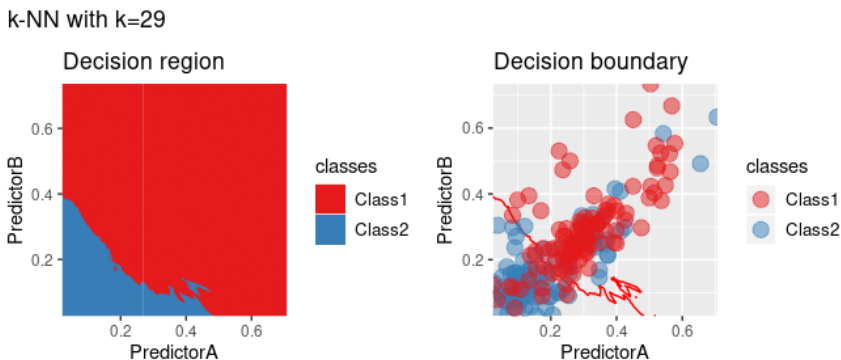
Example: KNN


k-NN with k=21
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

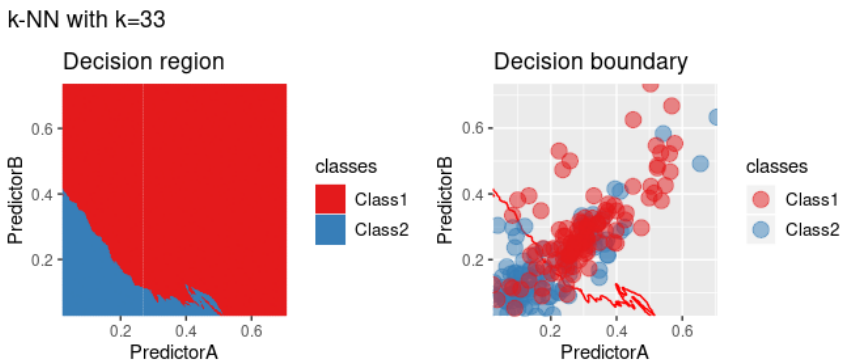
Example: KNN


k-NN with k=25
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

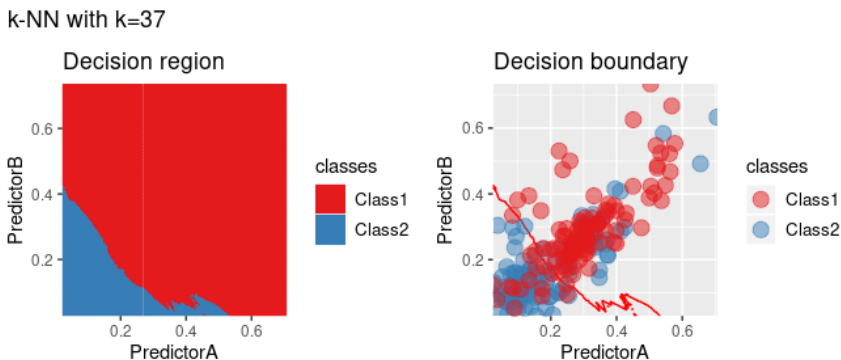
Example: KNN


k-NN with k=29
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

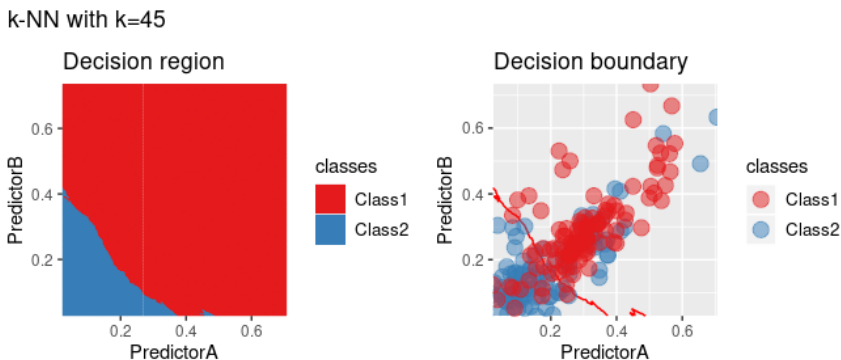
Example: KNN


k-NN with k=33
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

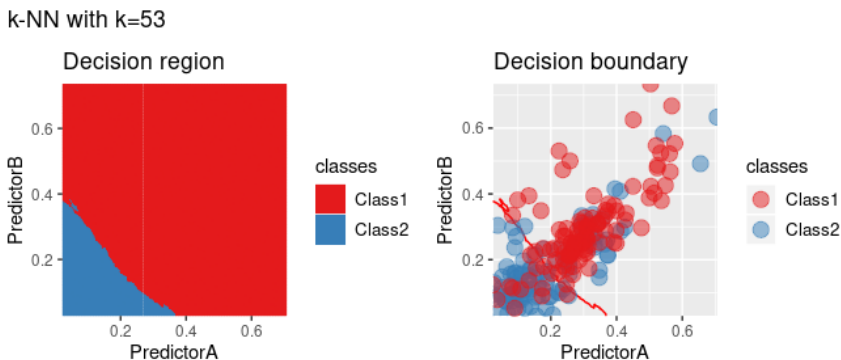
Example: KNN


k-NN with k=37
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

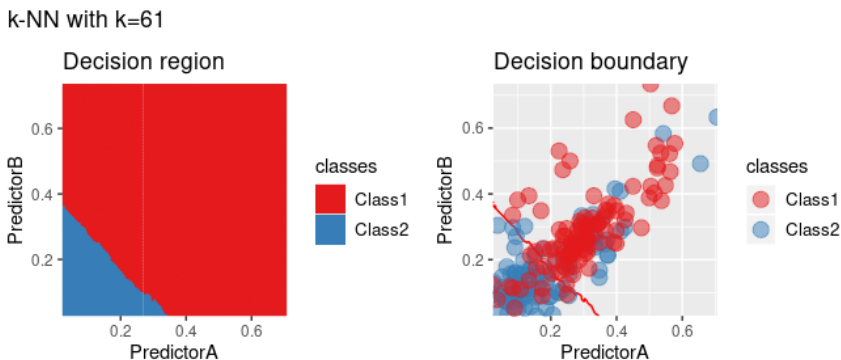
Example: KNN


k-NN with k=45
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

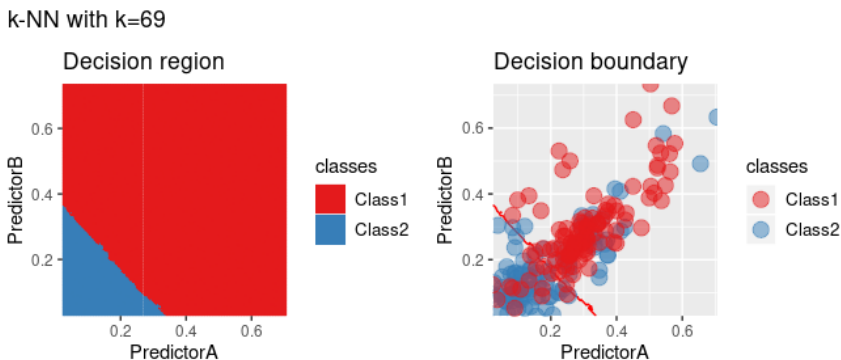
Example: KNN


k-NN with k=53
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

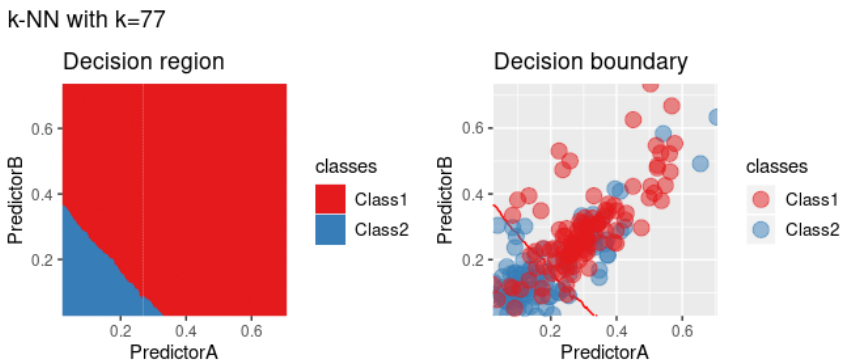
Example: KNN


k-NN with k=61
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

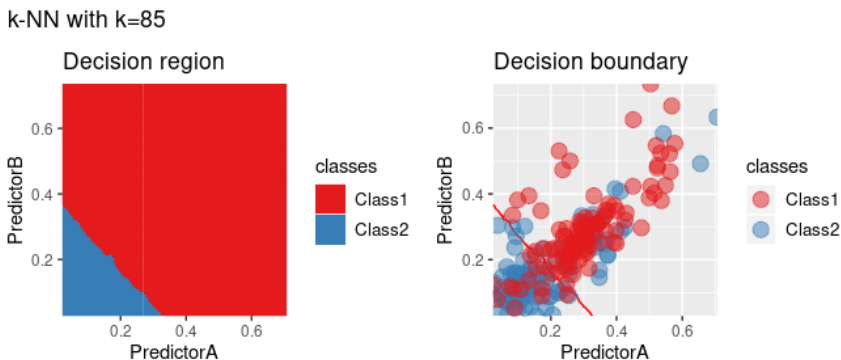
Example: KNN


k-NN with k=69
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

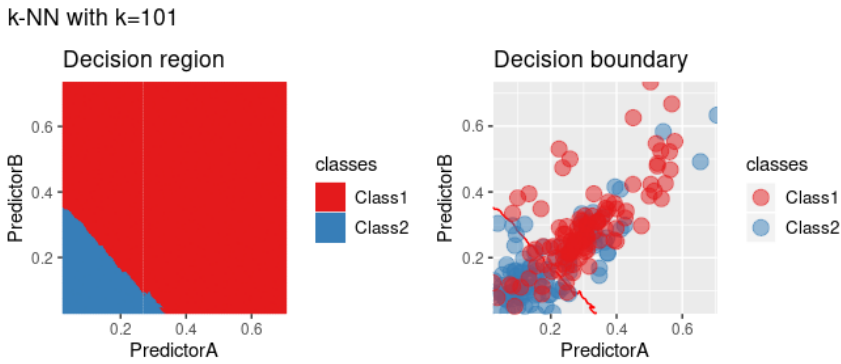
Example: KNN


k-NN with k=77
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

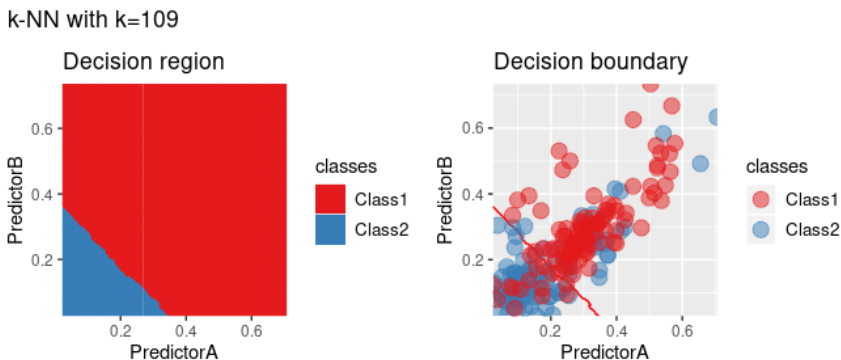
Example: KNN


k-NN with k=85
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

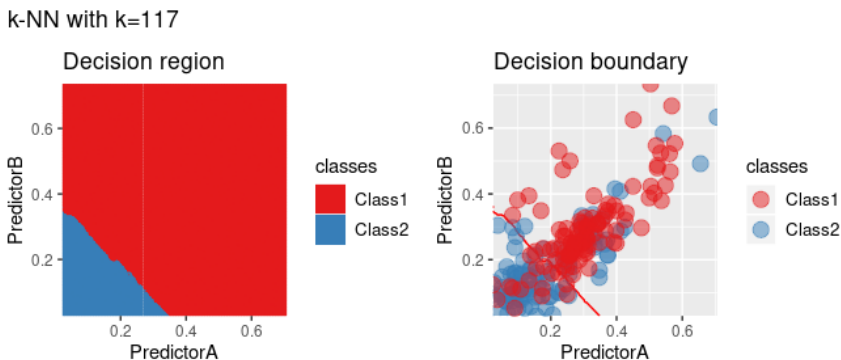
Example: KNN


k-NN with k=101
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

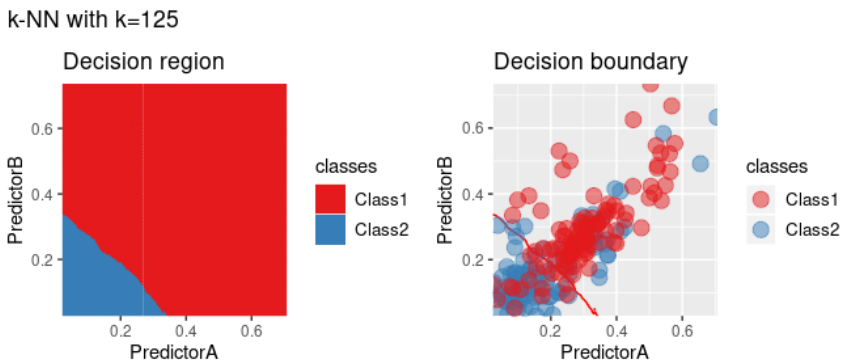
Example: KNN


k-NN with k=109
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

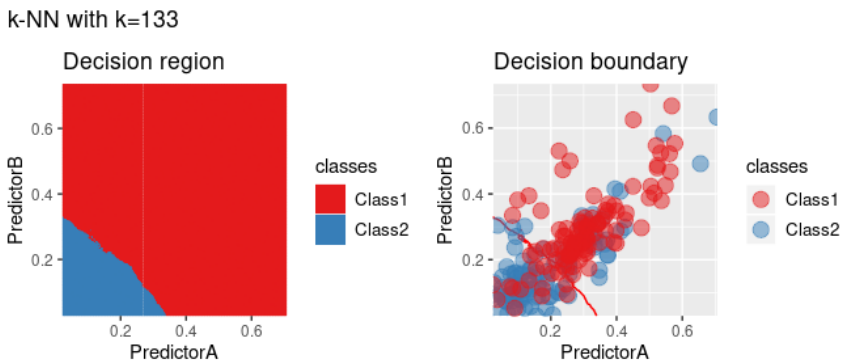
Example: KNN


k-NN with k=117
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

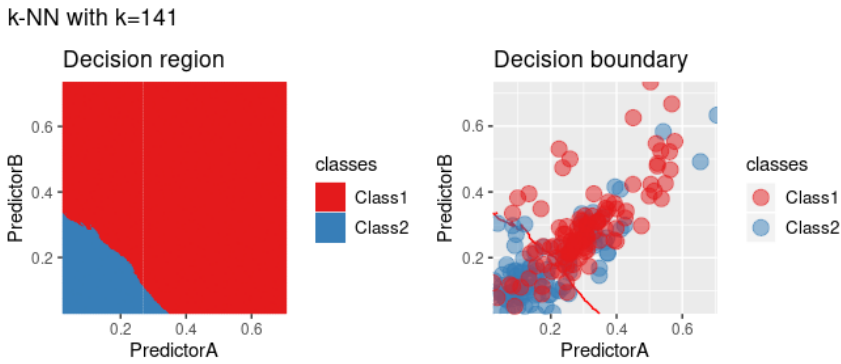
Example: KNN


k-NN with k=125
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

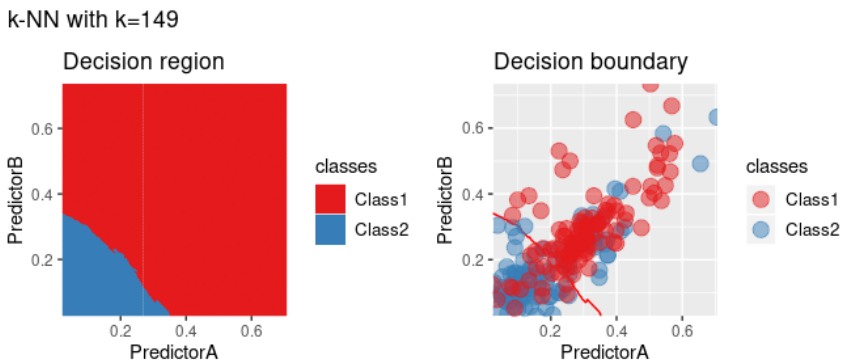
Example: KNN


k-NN with k=133
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

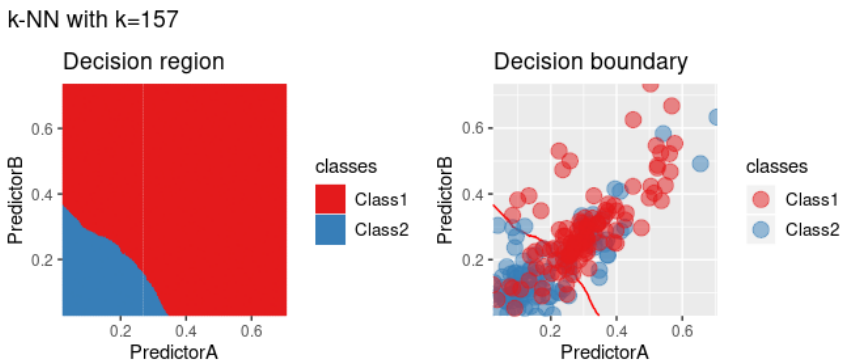
Example: KNN


k-NN with k=141
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

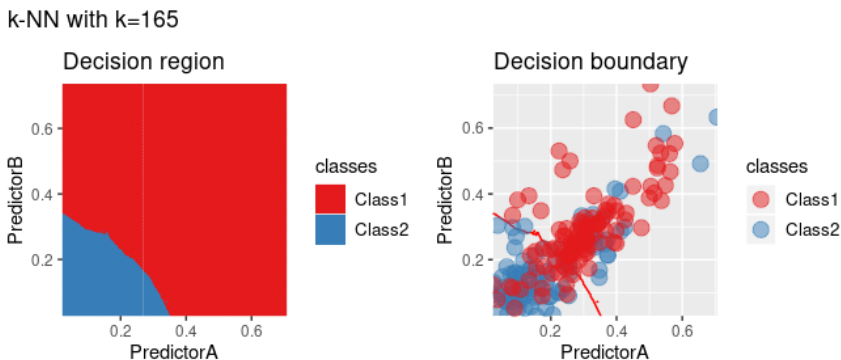
Example: KNN


k-NN with k=149
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

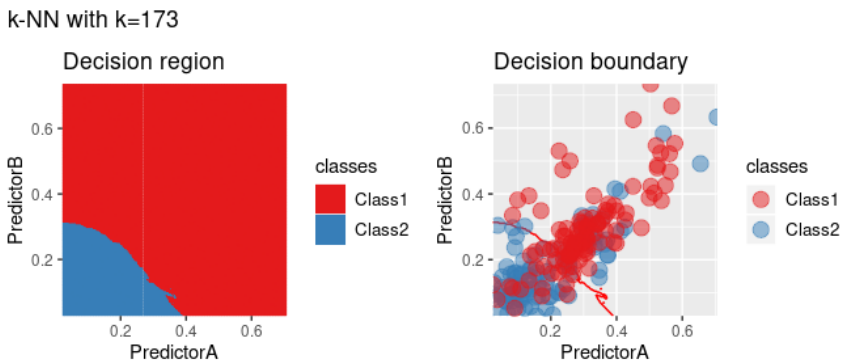
Example: KNN


k-NN with k=157
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

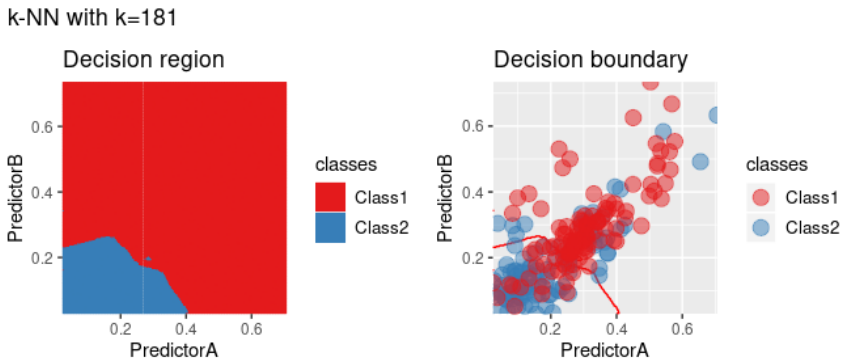
Example: KNN


k-NN with k=165
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

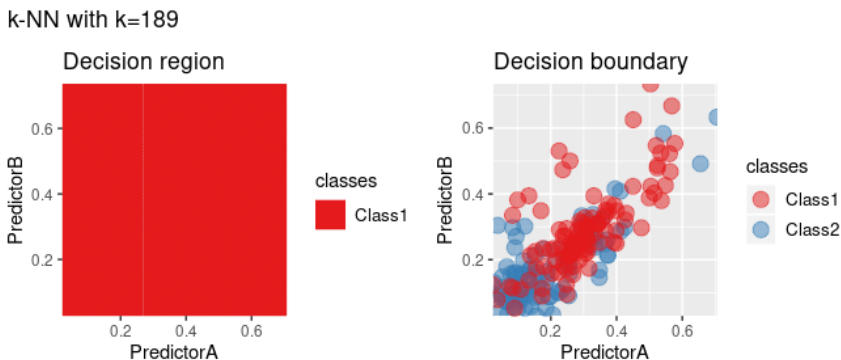
Example: KNN


k-NN with k=173
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

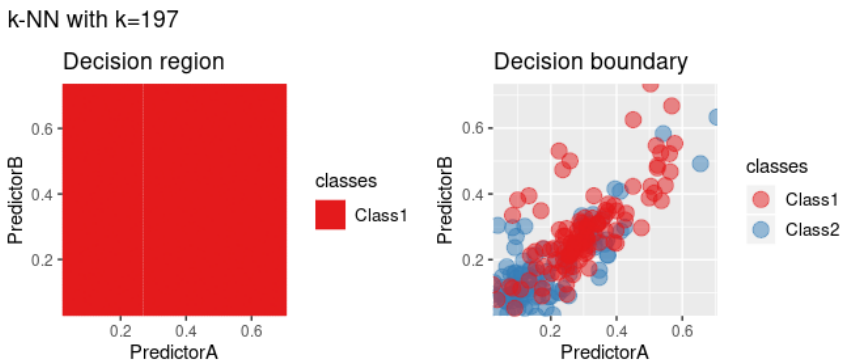
Example: KNN


k-NN with k=181
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

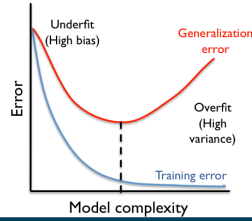
Example: KNN


k-NN with k=189
Decision region
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Decision boundary
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2

Example: KNN


k-NN with k=197
Decision region
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Decision boundary
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2

Training Risk Issue



Risk behaviour

- Learning/training risk (empirical risk on the learning/training set) decays when the complexity of the **method** increases.
- Quite different behavior when the risk is computed on new observations (generalization risk).

- Overfit for complex methods: parameters learned are too specific to the learning set!
- General situation! (Think of polynomial fit...)
- Need to use a different criterion than the training risk!

Source: JMP

Risk Estimation vs Method Selection

Predictor Risk Estimation

- **Goal:** Given a predictor f assess its quality.
- **Method:** Hold-out risk computation (/ Empirical risk correction).
- **Usage:** Compute an estimate of the risk of a selected f using a **test set** to be used to monitor it in the future.

- Basic block very well understood.

Method Selection

- **Goal:** Given a ML method assess its quality.
- **Method:** Cross Validation (/ Empirical risk correction)
- **Usage:** Compute risk estimates for several ML methods using **training/validation sets** to choose the most promising one.

- Estimates can be pointwise or better intervals.
- Multiple test issues in method selection.

Cross Validation and Empirical Risk Correction

Two Approaches

- **Cross validation:** Use empirical risk criterion but on independent data, very efficient (and almost always used in practice!) but slightly biased as its target uses only a fraction of the data.
- **Correction approach:** use empirical risk criterion but *correct* it with a term increasing with the complexity of $\mathcal{S}$

$$R_n(\widehat{f}_{\mathcal{S}}) \to R_n(\widehat{f}_{\mathcal{S}}) + \text{cor}(\mathcal{S})$$

and choose the method with the smallest corrected risk.

Which loss to use?

- The loss used in the risk: most natural!
- The loss used to estimate $\widehat{\theta}$: penalized estimation!

- Other performance measure can be used.

Cross Validation

Cross Validation

- Use $(1-\epsilon) \times n$ observations to train and $\epsilon \times n$ to verify!
- Possible issues:
  - Validation for a learning set of size $(1-\epsilon) \times n$ instead of n ?
  - Unstable risk estimate if ϵn is too small ?

- Most classical variations:
  - Hold Out,
  - Leave One Out,
  - V-fold cross validation.

Source: M. Kühn

Hold Out

Principle

- Split the dataset $\mathcal{D}$ in 2 sets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ of size $n \times (1-\epsilon)$ and $n \times \epsilon$.
- Learn $\widehat{f}^{HO}$ from the subset $\mathcal{D}_{\text{train}}$.
- Compute the empirical risk on the subset $\mathcal{D}_{\text{test}}$:

$$\mathcal{R}_n^{HO}(\widehat{f}^{HO}) = \frac{1}{n\epsilon} \sum_{(\underline{X}_i, Y_i) \in \mathcal{D}_{\text{test}}} \ell(Y_i, \widehat{f}^{HO}(\underline{X}_i))$$

Predictor Risk Estimation

- Use $\hat{f}^{HO}$ as predictor.
- Use $\mathcal{R}_n^{HO}(\widehat{f}^{HO})$ as an estimate of the risk of this estimator.

Method Selection by Cross Validation

- Compute $\mathcal{R}_n^{HO}(\widehat{f}_{\mathcal{S}}^{HO})$ for all the considered methods,
- Select the method with the smallest CV risk,
- Reestimate the $\widehat{f}_{\mathcal{S}}$ with all the data.

Hold Out

Principle

- Split the dataset $\mathcal{D}$ in 2 sets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ of size $n \times (1-\epsilon)$ and $n \times \epsilon$.
- Learn $\widehat{f}^{HO}$ from the subset $\mathcal{D}_{\text{train}}$.
- Compute the empirical risk on the subset $\mathcal{D}_{\text{test}}$:

$$\mathcal{R}_n^{HO}(\widehat{f}^{HO}) = \frac{1}{n\epsilon} \sum_{(\underline{X}_i, Y_i) \in \mathcal{D}_{\text{test}}} \ell(Y_i, \widehat{f}^{HO}(\underline{X}_i))$$

- Only possible setting for risk estimation.

Hold Out Limitation for Method Selection

- Biased toward simpler method as the estimation does not use all the data initially.
- Learning variability of $\mathcal{R}_n^{HO}(\widehat{f}^{HO})$ not taken into account.

V-fold Cross Validation

Principle

- Split the dataset $\mathcal{D}$ in V sets $\mathcal{D}_v$ of almost equals size.
- For $v \in \{1, .., V\}$:
  - Learn $\widehat{f}^{-v}$ from the dataset $\mathcal{D}$ minus the set $\mathcal{D}_v$.
  - Compute the empirical risk:

$$\mathcal{R}_n^{-v}(\widehat{f}^{-v}) = \frac{1}{n_v} \sum_{(\underline{X}_i, Y_i) \in \mathcal{D}_v} \ell(Y_i, \widehat{f}^{-v}(\underline{X}_i))$$

- Compute the average empirical risk:

$$\mathcal{R}_n^{CV}(\widehat{f}) = \frac{1}{V} \sum_{v=1}^{V} \mathcal{R}_n^{-v}(\widehat{f}^{-v})$$

- Estimation of the quality of a method not of a given predictor.
- Leave One Out : V = n.

Source: M. Kühn

V-fold Cross Validation

Analysis (when n is a multiple of V)

- The $\mathcal{R}_n^{-v}(\widehat{f}^{-v})$ are identically distributed variable but are not independent!
- Consequence:

$$\mathbb{E}\left[\mathcal{R}_n^{CV}(\widehat{f})\right] = \mathbb{E}\left[\mathcal{R}_n^{-v}(\widehat{f}^{-v})\right]$$

$$\mathbb{V}\text{ar}\left[\mathcal{R}_n^{CV}(\widehat{f})\right] = \frac{1}{V}\mathbb{V}\text{ar}\left[\mathcal{R}_n^{-v}(\widehat{f}^{-v})\right] + (1-\frac{1}{V})\,\mathbb{C}\text{ov}\left[\mathcal{R}_n^{-v}(\widehat{f}^{-v}), \mathcal{R}_n^{-v'}(\widehat{f}^{-v'})\right]$$

- Average risk for a sample of size $(1-\frac{1}{V})n$.
- Variance term much more complex to analyze!
- Fine analysis shows that the larger V the better...

- Accuracy/Speed tradeoff: $V = 5$ or $V = 10$...

Linear Regression and Leave One Out

- Leave One Out $=$ V fold for $V = n$: very expensive in general.

A fast LOO formula for the linear regression

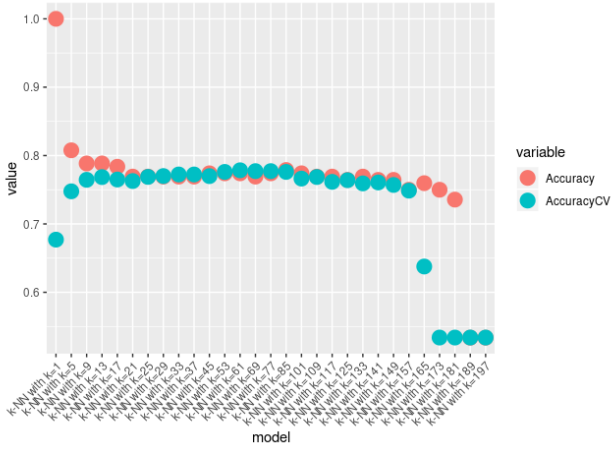
- **Prop:** for the least squares linear regression,

$$\widehat{f}^{-i}(\underline{X}_i) = \frac{\widehat{f}(\underline{X}_i) - h_{ii} Y_i}{1 - h_{ii}}$$

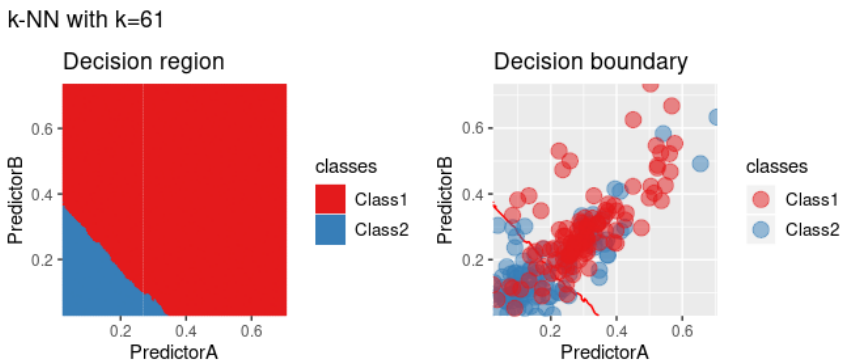
  with h_{ii} the ith diagonal coefficient of the **hat** (projection) matrix.
- Proof based on linear algebra!
- Leads to a fast formula for LOO:

$$\mathcal{R}_n^{LOO}(\widehat{f}) = \frac{1}{n} \sum_{i=1}^{n} \frac{|Y_i - \widehat{f}(\underline{X}_i)|^2}{(1 - h_{ii})^2}$$

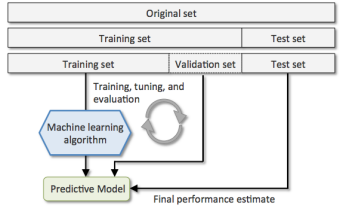
Cross Validation



Example: KNN ($\widehat{k} = 61$ using cross-validation)



Train/Validation/Test



- **Selection Bias Issue:**
  - After method selection, the cross validation is biased.
  - Furthermore, it qualifies the method and not the final predictor.
- Need to (re)estimate the risk of the final predictor.

(Train/Validation)/Test strategy

- **Split** the dataset in two a `(Train/Validation)` and `Test`.
- Use **CV** with the `(Train/Validation)` to **select a method**.
- Train this method on `(Train/Validation)` to **obtain a single predictor**.
- Estimate the **performance of this predictor** on `Test`.

- Every choice made from the data is part of the method!

Source: Shan-Hung Wu & DataLab

Risk Correction

- Empirical loss of an estimator computed on the dataset used to chose it is biased!
- Empirical loss is an optimistic estimate of the true loss.

Risk Correction Heuristic

- Estimate an upper bound of this optimism for a given family.
- Correct the empirical loss by adding this upper bound.

- **Rk:** Finding such an upper bound can be complicated!
- Correction often called a **penalty**.

Penalization

Penalized Loss

- Minimization of

$$\underset{\theta \in \Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f_\theta(\underline{X}_i)) + \text{pen}(\theta)$$

where $\text{pen}(\theta)$ is a risk correction (penalty).

Penalties

- Upper bound of the optimism of the empirical loss
- Depends on the loss and the framework!

Instantiation

- Mallows Cp: Least Squares with $\text{pen}(\theta) = 2\frac{d}{n}\sigma^2$.
- AIC Heuristics: Maximum Likelihood with $\text{pen}(\theta) = \frac{d}{n}$.
- BIC Heuristics: Maximum Likelohood with $\text{pen}(\theta) = \log(n)\frac{d}{n}$.
- Structural Risk Minimization: Pred. loss and clever penalty.

Outline

Comparison of Two Means

Means

- **Setting:** r.v. $e_i^{(l)}$ with $1 \leq i \leq n_l$ and $l \in \{1, 2\}$ and their means

$$\overline{e^{(l)}} = \frac{1}{n_l} \sum_{i=1}^{n_l} e_i^{(l)}$$

- **Question:** are the means $\overline{e^{(l)}}$ statistically different?

Classical i.i.d setting

- **Assumption:** $e_i^{(l)}$ are i.i.d. for each l.
- **Test formulation:** Can we reject the null hypothesis that $\mathbb{E}\left[e^{(1)}\right] = \mathbb{E}\left[e^{(2)}\right]$?
- **Methods:**
  - Gaussian (Student) test using asymptotic normality of a mean.
  - Non-parametric permutation test.

- Gaussian approach is linked to confidence intervals.
- The larger n_l the smaller the confidence intervals.

Comparison of Two Means

Non i.i.d. case

- **Assumption:** $e_i^{(l)}$ are i.d. for each l but not necessarily independent.
- **Test formulation:** Can we reject the null hypothesis that $\mathbb{E}\left[e^{(1)}\right] = \mathbb{E}\left[e^{(2)}\right]$?
- **Methods:**
  - Gaussian (Student) test using asymptotic normality of a mean but variance is hard to estimate.
  - Non-parametric permutation test but no confidence intervals.

- Setting for Cross Validation (other than holdout).
- Much more complicated than the i.i.d. case

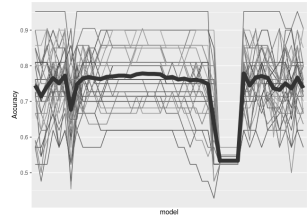
Comparison of Several Means

Several means

- **Assumption:** $e_i^{(l)}$ are i.d. for each l but not necessarily independent.
- **Tests formulation:**
  - Can we reject the null hypothesis that the $\mathbb{E}\left[e^{(l)}\right]$ are different?
  - Is the smaller mean statistically smaller than the second one?
- **Methods:**
  - Gaussian (Student) test using asymptotic normality of a mean with multiple tests correction.
  - Non-parametric permutation test but no confidence intervals.

- Setting for Cross Validation (other than holdout).
- The more models one compares:
  - the larger the confidence intervals
  - the most probable the best model is a lucky winner
- Justify the fallback to the simplest model that could be the best one.

PAC Approach



CV Risk, Methods and Predictors

- Cross-Validation risk: estimate of the average risk of a ML method.
- No risk bound on the predictor obtained in practice.

Probabibly-Approximately-Correct (PAC) Approach

- Replace the control on the average risk by a probabilistic bound

$$\mathbb{P}\Big(\mathbb{E}\Big[\ell(Y, \hat{f}(\underline{X}))\Big] > R\Big) \leq \epsilon$$

- Requires estimating quantiles of the risk.

Cross Validation and Confidence Interval

- How to replace pointwise estimation by a confidence interval?
- Can we use the variability of the CV estimates?
- **Negative result:** No unbiased estimate of the variance!

Gaussian Interval (Comparison of the means and $\sim$ indep.)

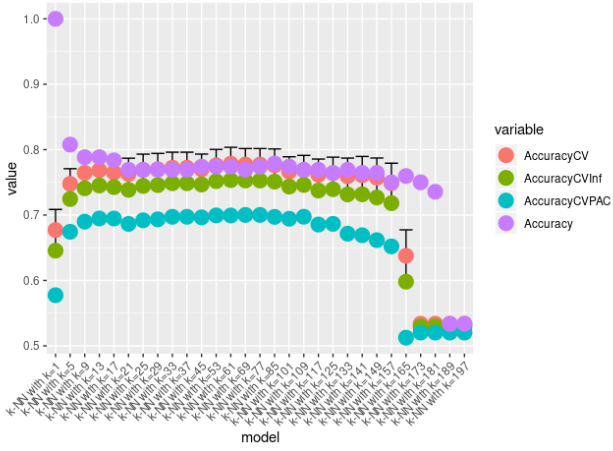
- Compute the empirical variance and divide it by the number of folds to construct an asymptotic Gaussian confidence interval,
- Select the simplest model whose value falls into the confidence interval of the model having the smallest CV risk.

PAC approach (Quantile, $\sim$ indep. and small risk estim. error)

- Compute the raw medians (or a larger raw quantiles)
- Select the model having the smallest quantiles to ensure a small risk with high probability.

- Always reestimate the chosen model with all the data.
- To obtain an unbiased risk estimate of the final predictor: hold out risk on untouched test data.

Cross Validation



Outline

References

T. Hastie, R. Tibshirani, and J. Friedman.
The Elements of Statistical Learning.
Springer Series in Statistics, 2009

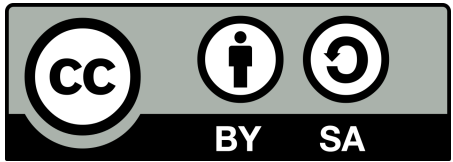
M. Mohri, A. Rostamizadeh, and A. Talwalkar.
Foundations of Machine Learning (2nd ed.)
MIT Press, 2018

A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd ed.)
O'Reilly, 2022

Ch. Giraud.
Introduction to High-Dimensional Statistics.
CRC Press, 2014

S. Bubeck.
Convex Optimization: Algorithms and Complexity.
Now Publisher, 2015

Licence and Contributors



Contributors

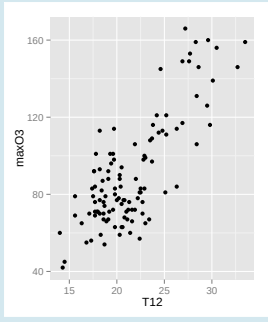
- Main contributor: E. Le Pennec
- Contributors: S. Boucheron, A. Dieuleveut, A.K. Fermin, S. Gadat, S. Gaiffas, A. Guilloux, Ch. Keribin, E. Matzner, M. Sangnier, E. Scornet.

Outline

Outline

Motivation

Ozone pollution



- **Data:** Air Breizh, Summer 2001
- **Input:** Temperature at 12h00
- **Output:** max Ozone concentration

Source: Air Breizh

Motivation

Credit Default, Credit Score, Bank Risk, Market Risk Management

- **Data:** Client profile, Client credit history...
- **Input:** Client profile
- **Output:** Credit risk

Source: Unknown

Motivation

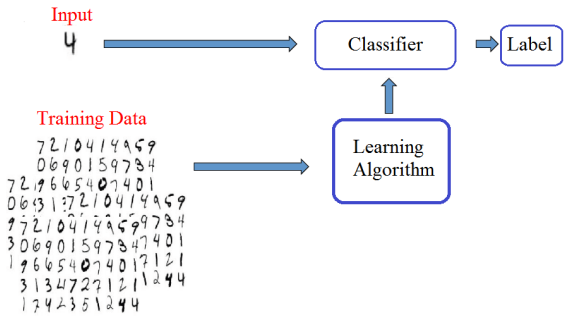
Marketing: advertisement, recommendation...

- **Data:** User profile, Web site history...
- **Input:** User profile, Current web page
- **Output:** Advertisement with price, recommendation...

Spam detection (Text classification)

Source: X. Bresson

Machine Learning



A definition by Tom Mitchell

A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E.

Source: Unknown

Outline

Supervised Learning

Supervised Learning Framework

- Input measurement $\underline{X} \in \mathcal{X}$
- Output measurement $Y \in \mathcal{Y}$.
- $(\underline{X}, Y) \sim \mathbb{P}$ with $\mathbb{P}$ unknown.
- **Training data** : $\mathcal{D}_n = \{(\underline{X}_1, Y_1), \ldots, (\underline{X}_n, Y_n)\}$ (i.i.d. $\sim \mathbb{P}$)

- Often
  - $\underline{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (classification)
  - or $\underline{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A **predictor** is a function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y} \text{ meas.}\}$

Goal

- Construct a **good** predictor $\widehat{f}$ from the training data.

- Need to specify the meaning of good.
- Classification and regression are almost the **same** problem!

Loss and Probabilistic Framework

Loss function for a generic predictor

- **Loss function**: $\ell(Y, f(\underline{X}))$ measures the goodness of the prediction of Y by $f(\underline{X})$
- Examples:
  - 0/1 loss: $\ell(Y, f(\underline{X})) = \mathbf{1}_{Y \neq f(\underline{X})}$
  - Quadratic loss: $\ell(Y, f(\underline{X})) = |Y - f(\underline{X})|^2$

Risk function

- Risk measured as the average loss for a new couple:

$$\mathcal{R}(f) = \mathbb{E}_{(\underline{X},Y)\sim\mathbb{P}}[\ell(Y, f(\underline{X}))]$$

- Examples:
  - 0/1 loss: $\mathbb{E}[\ell(Y, f(\underline{X}))] = \mathbb{P}(Y \neq f(\underline{X}))$
  - Quadratic loss: $\mathbb{E}[\ell(Y, f(\underline{X}))] = \mathbb{E}\left[|Y - f(\underline{X})|^2\right]$

- **Beware:** As $\widehat{f}$ depends on $\mathcal{D}_n$, $\mathcal{R}(\widehat{f})$ is a random variable!

Best Solution

- The best solution $f^\star$ (which is independent of $\mathcal{D}_n$) is

$$f^\star = \arg\min_{f \in \mathcal{F}} \mathcal{R}(f) = \arg\min_{f \in \mathcal{F}} \mathbb{E}[\ell(Y, f(\underline{X}))] = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{\underline{X}}\Big[\mathbb{E}_{Y|\underline{X}}[\ell(Y, f(\underline{X}))]\Big]$$

Bayes Predictor (explicit solution)

- In binary classification with $0-1$ loss:

$$f^\star(\underline{X}) = \begin{cases} +1 & \text{if } \mathbb{P}(Y=+1|\underline{X}) \geq \mathbb{P}(Y=-1|\underline{X}) \\ & \Leftrightarrow \mathbb{P}(Y=+1|\underline{X}) \geq 1/2 \\ -1 & \text{otherwise} \end{cases}$$

- In regression with the quadratic loss

$$f^\star(\underline{X}) = \mathbb{E}[Y|\underline{X}]$$

Issue: Solution requires to **know** $\mathbb{E}[Y|\underline{X}]$ for all values of $\underline{X}$!

Goal

Machine Learning

- Learn a rule to construct a **predictor** $\widehat{f} \in \mathcal{F}$ from the training data $\mathcal{D}_n$ s.t. **the risk** $\mathcal{R}(\widehat{f})$ is **small on average** or with high probability with respect to $\mathcal{D}_n$.

- In practice, the rule should be an algorithm!

Canonical example: Empirical Risk Minimizer

- One restricts f to a subset of functions $\mathcal{S} = \{f_\theta, \theta \in \Theta\}$
- One replaces the minimization of the average loss by the minimization of the empirical loss

$$\widehat{f} = f_{\widehat{\theta}} = \underset{f_\theta, \theta \in \Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f_\theta(\underline{X}_i))$$

- Examples:
  - Linear regression
  - Linear classification with

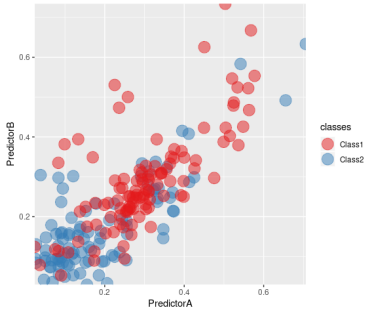
$$\mathcal{S} = \{\underline{x} \mapsto \operatorname{sign}\{\underline{x}^\top \beta + \beta^{(0)}\} \, / \beta \in \mathbb{R}^d, \beta^{(0)} \in \mathbb{R}\}$$

Example: TwoClass Dataset

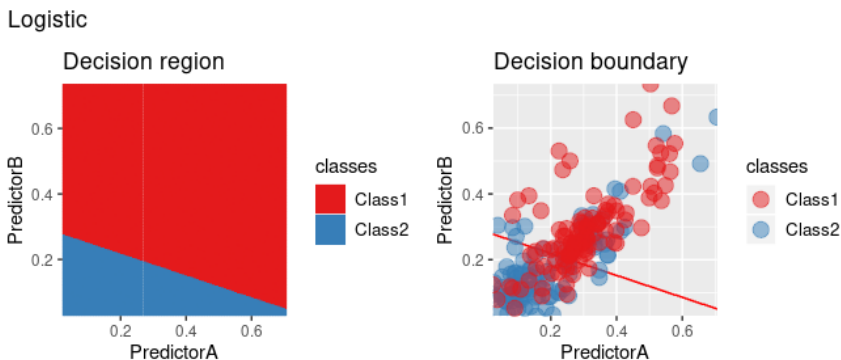
Synthetic Dataset

- Two features/covariates.
- Two classes.

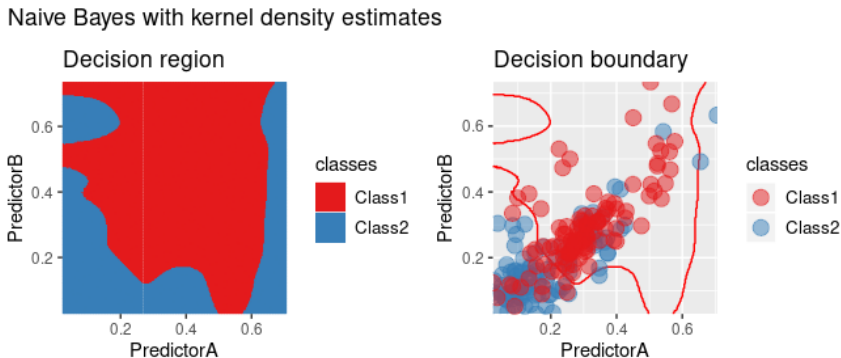
- Dataset from *Applied Predictive Modeling*, M. Kuhn and K. Johnson, Springer
- Numerical experiments with `R` and the {`caret`} package.



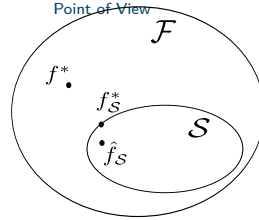
Example: Linear Discrimination



Example: More Complex Model



Bias-Variance Dilemma



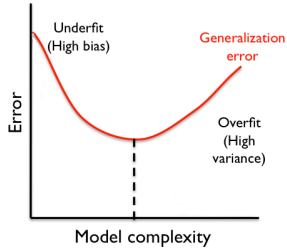
- General setting:
  - $\mathcal{F} = \{\text{measurable functions } \mathcal{X} \to \mathcal{Y}\}$
  - Best solution: $f^\star = \text{argmin}_{f \in \mathcal{F}} \mathcal{R}(f)$
  - Class $\mathcal{S} \subset \mathcal{F}$ of functions
  - Ideal target in $\mathcal{S}$: $f^\star_{\mathcal{S}} = \text{argmin}_{f \in \mathcal{S}} \mathcal{R}(f)$
  - Estimate in $\mathcal{S}$: $\widehat{f}_{\mathcal{S}}$ obtained with some procedure

Approximation error and estimation error (Bias/Variance)

$$\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star) = \underbrace{\mathcal{R}(f^\star_{\mathcal{S}}) - \mathcal{R}(f^\star)}_{\text{Approximation error}} + \underbrace{\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star_{\mathcal{S}})}_{\text{Estimation error}}$$

- Approx. error can be large if the model $\mathcal{S}$ is not suitable.
- Estimation error can be large if the model is complex.

Under-fitting / Over-fitting Issue



- Different behavior for different model complexity
- **Low complexity model** are easily learned but the approximation error (**bias**) may be large (**Under-fit**).
- **High complexity model** may contain a good ideal target but the estimation error (**variance**) can be large (**Over-fit**)

Bias-variance trade-off $\iff$ avoid **overfitting** and **underfitting**

- **Rk:** Better to think in term of method (including feature engineering and specific algorithm) rather than only of model.

Source: Unknown

Probabilistic and Optimization Framework

How to find a good function f with a *small* risk

$$\mathcal{R}(f) = \mathbb{E}[\ell(Y, f(\underline{X}))] \quad ?$$

Canonical approach: $\widehat{f}_{\mathcal{S}} = \text{argmin}_{f \in \mathcal{S}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f(\underline{X}_i))$

Problems

- How to choose $\mathcal{S}$?
- How to compute the minimization?

A Probabilistic Point of View

Solution: For $\underline{X}$, estimate $Y|\underline{X}$ plug this estimate in the Bayes classifier:
(Generalized) Linear Models, Kernel methods, k-nn, Naive Bayes, Tree, Bagging...

An Optimization Point of View

Solution: If necessary replace the loss ℓ by an upper bound $\bar{\ell}$ and minimize the empirical loss: **SVR, SVM, Neural Network,Tree, Boosting...**

Outline

Three Classical Methods in a Nutshell

Logistic Regression

- Let $f_\theta(\underline{X}) = \underline{X}^\top \beta + \beta^{(0)}$ with $\theta = (\beta, \beta^{(0)})$.
- Let $\mathbb{P}_\theta(Y = 1|\underline{X}) = e^{-f_\theta(\underline{X})}/(1 + e^{f_\theta(\underline{X})})$
- Estimate θ by $\hat{\theta}$ using a Maximum Likelihood.
- Classify using $\mathbb{P}_{\hat{\theta}}(Y = 1|\underline{X}) > 1/2$

k Nearest Neighbors

- For any $\underline{X}'$, define $\mathcal{V}_{\underline{X}'}$ as the k closest samples X_i from the dataset.
- Compute a score $g_k = \sum_{X_i \in \mathcal{V}_{\underline{X}'}} \mathbf{1}_{Y_i = k}$
- Classify using $\arg\max g_k$ (majority vote).

Three Classical Methods in a Nutshell

Quadratic Discrimant Analysis

- For each class, estimate the mean μ_k and the covariance matrix Σ_k.
- Estimate the proportion $\mathbb{P}(Y = k)$ of each class.
- Compute a score $\ln(\mathbb{P}(\underline{X}|Y = k)) + \ln(\mathbb{P}(Y = k))$

$$g_k(\underline{X}) = -\frac{1}{2}(\underline{X} - \mu_k)^\top \Sigma_k^{-1}(\underline{X} - \mu_k) - \frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln(|\Sigma_k|) + \ln(\mathbb{P}(Y = k))$$

- Classify using $\arg\max g_k$

- Those three methods rely on a similar heuristic: the probabilistic point of view!

Best Solution

- The best solution $f^\star$ (which is independent of $\mathcal{D}_n$) is

$$f^\star = \arg\min_{f \in \mathcal{F}} R(f) = \arg\min_{f \in \mathcal{F}} \mathbb{E}[\ell(Y, f(\underline{X}))] = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{\underline{X}}\Big[\mathbb{E}_{Y|\underline{X}}[\ell(Y, f(\underline{X})]\Big]$$

Bayes Predictor (explicit solution)

- In binary classification with $0-1$ loss:

$$f^\star(\underline{X}) = \begin{cases} +1 & \text{if } \mathbb{P}(Y=+1|\underline{X}) \geq \mathbb{P}(Y=-1|\underline{X}) \\ & \Leftrightarrow \mathbb{P}(Y=+1|\underline{X}) \geq 1/2 \\ -1 & \text{otherwise} \end{cases}$$

- In regression with the quadratic loss

$$f^\star(\underline{X}) = \mathbb{E}[Y|\underline{X}]$$

Issue: Explicit solution requires to **know** $Y|\underline{X}$ (or $\mathbb{E}[Y|\underline{X}]$) for all values of $\underline{X}$!

Plugin Predictor

- **Idea:** Estimate $Y|\underline{X}$ by $\widehat{Y|\underline{X}}$ and plug it the Bayes classifier.

Plugin Bayes Predictor

- In binary classification with $0-1$ loss:

$$\widehat{f}(\underline{X}) = \begin{cases} +1 & \text{if } \widehat{\mathbb{P}(Y=+1|\underline{X})} \geq \widehat{\mathbb{P}(Y=-1|\underline{X})} \\ & \Leftrightarrow \widehat{\mathbb{P}(Y=+1|\underline{X})} \geq 1/2 \\ -1 & \text{otherwise} \end{cases}$$

- In regression with the quadratic loss

$$\widehat{f}(\underline{X}) = \mathbb{E}\left[\widehat{Y|\underline{X}}\right]$$

- **Rk:** Direct estimation of $\mathbb{E}[Y|\underline{X}]$ by $\widehat{\mathbb{E}[Y|\underline{X}]}$ also possible...

Plugin Predictor

- How to estimate $Y|\underline{X}$?

Three main heuristics

- **Parametric Conditional modeling:** Estimate the law of $Y|\underline{X}$ by a **parametric** law $\mathcal{L}_\theta(\underline{X})$: *(generalized) linear regression...*
- **Non Parametric Conditional modeling:** Estimate the law of $Y|\underline{X}$ by a **non parametric** estimate: *kernel methods, loess, nearest neighbors...*
- **Fully Generative modeling:** Estimate the law of $(\underline{X}, Y)$ and use the **Bayes formula** to deduce an estimate of $Y|\underline{X}$: *LDA/QDA, Naive Bayes...*

- **Rk:** Direct estimation of $\mathbb{E}[Y|\underline{X}]$ by $\widehat{\mathbb{E}[Y|\underline{X}]}$ also possible...

Plugin Classifier

- **Input**: a data set $\mathcal{D}_n$
  Learn $Y|\underline{X}$ or equivalently $\mathbb{P}(Y = k|\underline{X})$ (using the data set) and plug this estimate in the Bayes classifier
- **Output**: a classifier $\widehat{f} : \mathbb{R}^d \to \{-1, 1\}$

$$\widehat{f}(\underline{X}) = \begin{cases} +1 & \text{if } \mathbb{P}(\widehat{Y = 1}|\underline{X}) \geq \mathbb{P}(\widehat{Y = -1}|\underline{X}) \\ -1 & \text{otherwise} \end{cases}$$

- Can we guaranty that the classifier is good if $Y|\underline{X}$ is well estimated?

Classification Risk Analysis

Theorem

- If $\widehat{f} = \text{sign}(2\widehat{p}_{+1} - 1)$ then

$$\begin{aligned}\mathbb{E}\Big[\ell^{0,1}(Y, \widehat{f}(\underline{X}))\Big] - \mathbb{E}\Big[\ell^{0,1}(Y, f^\star(\underline{X}))\Big] &\leq \mathbb{E}\Big[\|\widehat{Y|\underline{X}} - Y|\underline{X}\|_1\Big] \\ &\leq \Big(\mathbb{E}\Big[2\text{KL}(Y|\underline{X}, \widehat{Y|\underline{X}}\Big]\Big)^{1/2}\end{aligned}$$

- If one estimates $\mathbb{P}(Y = 1|\underline{X})$ well then one estimates $f^\star$ well!
- Link between a *conditional density estimation* task and a *classification* one!
- **Rk:** In general, the conditional density estimation task is more complicated as one should be good for all values of $\mathbb{P}(Y = 1|\underline{X})$ while the classification task focus on values around $1/2$ for the $0/1$ loss!
- In **regression**, (often) direct control of the quadratic loss...

Outline

Parametric Conditional Density Models

- **Idea:** Estimate directly $Y|\underline{X}$ by a parametric conditional density $\mathbb{P}_\theta(Y|\underline{X})$.

Maximum Likelihood Approach

- Classical choice for θ:

$$\widehat{\theta} = \underset{\theta}{\operatorname{argmin}} - \sum_{i=1}^{n} \log \mathbb{P}_\theta(Y_i|\underline{X}_i)$$

- **Goal:** *Minimize* the Kullback-Leibler divergence between the conditional law of $Y|\underline{X}$ and $\mathbb{P}_\theta(Y|\underline{X})$

$$\mathbb{E}[\mathrm{KL}\left(Y|\underline{X}, \mathbb{P}_\theta(Y|\underline{X})\right)]$$

- **Rk:** This is often not (exactly) the learning task!
- Large choice for the family $\{\mathbb{P}_\theta(Y|\underline{X})\}$ but depends on $\mathcal{Y}$ (and $\mathcal{X}$).
- **Regression:** One can also model directly $\mathbb{E}[Y|\underline{X}]$ by $f_\theta(\underline{X})$ and estimate it with a least-squares criterion...

Linear Conditional Density Models

Linear Models

- **Classical choice:** $\theta = (\theta', \varphi)$

$$\mathbb{P}_\theta(Y|\underline{X}) = \mathbb{P}_{\underline{X}^\top \beta, \varphi}(Y)$$

- **Very strong assumption!**

- Classical examples:
  - Binary variable: logistic, probit...
  - Discrete variable: multinomial logistic regression...
  - Integer variable: Poisson regression...
  - Continuous variable: Gaussian regression...

Binary Classifier

Plugin Linear Classification

- Model $\mathbb{P}(Y = +1|\underline{X})$ by $h(\underline{X}^\top \beta + \beta^{(0)})$ with h non decreasing.
- $h(\underline{X}^\top \beta + \beta^{(0)}) > 1/2 \Leftrightarrow \underline{X}^\top \beta + \beta^{(0)} - h^{-1}(1/2) > 0$
- Linear Classifier: $\text{sign}(\underline{X}^\top \beta + \beta^{(0)} - h^{-1}(1/2))$

Plugin Linear Classifier Estimation

- Classical choice for h:

$$h(t) = \frac{e^t}{1+e^t} \qquad \text{logit or logistic}$$

$$h(t) = F_{\mathcal{N}}(t) \qquad \text{probit}$$

$$h(t) = 1 - e^{-e^t} \qquad \text{log-log}$$

- Choice of the *best* β from the data.

Maximum Likelihood Estimate

Probabilistic Model

- By construction, $Y|\underline{X}$ follows $\mathcal{B}(\mathbb{P}(Y = +1|\underline{X}))$
- Approximation of $Y|\underline{X}$ by $\mathcal{B}(h(\underline{x}^\top \beta + \beta^{(0)}))$
- *Natural* probabilistic choice for β: maximum likelihood estimate.
- *Natural* probabilistic choice for β: β approximately minimizing a distance between $\mathcal{B}(h(\underline{x}^\top \beta))$ and $\mathcal{B}(\mathbb{P}(Y = 1|\underline{X}))$.

Maximum Likelihood Approach

- Minimization of the negative log-likelihood:

$$-\sum_{i=1}^{n} \log(\mathbb{P}(Y_i|\underline{X}_i)) = -\sum_{i=1}^{n} \left(\mathbf{1}_{Y_i=1} \log(h(\underline{X}_i{}^\top \beta)) + \mathbf{1}_{Y_i=-1} \log(1 - h(\underline{X}_i{}^\top \beta)) \right)$$

- Minimization possible if h is regular...

Maximum Likelihood Estimate

KL Distance and negative log-likelihood

- *Natural* distance: Kullback-Leibler divergence

$$\begin{aligned}
&\mathrm{KL}(\mathcal{B}(\mathbb{P}(Y=1|\underline{X})), \mathcal{B}(h(\underline{X}^\top\beta)) \\
&= \mathbb{E}_{\underline{X}}\left[\mathbb{P}(Y=1|\underline{X})\log\frac{\mathbb{P}(Y=1|\underline{X})}{h(\underline{X}^\top\beta)}\right. \\
&\qquad \left. + \mathbb{P}(Y=-1|\underline{X})\log\frac{1-\mathbb{P}(Y=1|\underline{X})}{1-h(\underline{X}^\top\beta)}\right] \\
&= \mathbb{E}_{\underline{X}}\left[-\mathbb{P}(Y=1|\underline{X})\log(h(\underline{X}^\top\beta))\right. \\
&\qquad \left. - \mathbb{P}(Y=-1|\underline{X})\log(1-h(\underline{X}^\top\beta))\right] + C_{\underline{X},Y}
\end{aligned}$$

- Empirical counterpart = negative log-likelihood (up to $1/n$ factor):

$$-\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{1}_{Y_i=1}\log(h(\underline{X}_i{}^\top\beta)) + \mathbf{1}_{Y_i=-1}\log(1-h(\underline{X}_i{}^\top\beta))\right)$$

Logistic Regression

Logistic Regression and Odd

- Logistic model: $h(t) = \frac{e^t}{1+e^t}$ (most *natural* choice...)
- The Bernoulli law $\mathcal{B}(h(t))$ satisfies then
$$\frac{\mathbb{P}(Y=1)}{\mathbb{P}(Y=-1)} = e^t \Leftrightarrow \log \frac{\mathbb{P}(Y=1)}{\mathbb{P}(Y=-1)} = t$$
- Interpretation in term of odd.
- Logistic model: linear model on the logarithm of the odd
$$\log \frac{\mathbb{P}(Y=1|\underline{X})}{\mathbb{P}(Y=-1|\underline{X})} = \underline{X}^\top \beta$$

Associated Classifier

- Plugin strategy:
$$f_\beta(\underline{X}) = \begin{cases} 1 & \text{if } \frac{e^{\underline{X}^\top \beta}}{1+e^{\underline{X}^\top \beta}} > 1/2 \Leftrightarrow \underline{X}^\top \beta > 0 \\ -1 & \text{otherwise} \end{cases}$$

Logistic Regression and Minimization

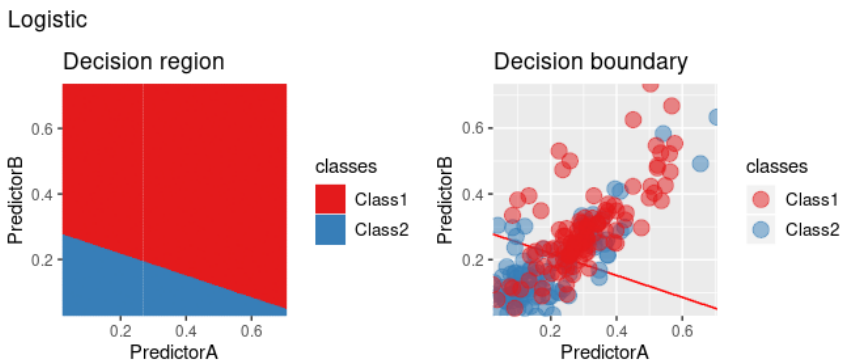
Likelihood Rewriting

- Negative log-likelihood:

$$
\begin{aligned}
&-\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{1}_{Y_i=1}\log(h(\underline{X}_i{}^\top\beta))+\mathbf{1}_{Y_i=-1}\log(1-h(\underline{X}_i{}^\top\beta))\right)\\
&=-\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{1}_{Y_i=1}\log\frac{e^{\underline{X}_i{}^\top\beta}}{1+e^{\underline{X}_i{}^\top\beta}}+\mathbf{1}_{Y_i=-1}\log\frac{1}{1+e^{\underline{X}_i{}^\top\beta}}\right)\\
&=\frac{1}{n}\sum_{i=1}^{n}\log\left(1+e^{-Y_i(\underline{X}_i{}^\top\beta)}\right)
\end{aligned}
$$

- Convex and smooth function of β
- Easy optimization.

Example: Logistic



Feature Design

Transformed Representation

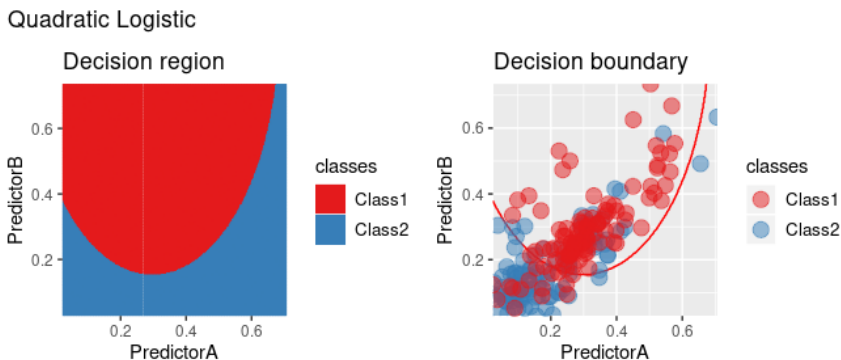
- From $\underline{X}$ to $\Phi(\underline{X})$!
- New description of $\underline{X}$ leads to a different **linear** model:

$$f_\beta(\underline{X}) = \Phi(\underline{X})^\top \beta$$

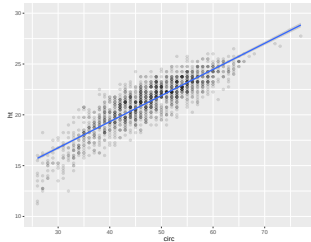
Feature Design

- Art of choosing Φ.
- Examples:
  - Renormalization, (domain specific) transform
  - Basis decomposition
  - Interaction between different variables...

Example: Quadratic Logistic



Gaussian Linear Regression



Gaussian Linear Model

- **Model:** $Y|\underline{X} \sim \mathcal{N}(\underline{X}^\top \beta, \sigma^2)$ plus independence
- Probably the most classical model of all time!
- Maximum Likelihood with explicit formulas for the two parameters.

- In regression, estimation of $\mathbb{E}[Y|\underline{X}]$ is sufficient: other/no model for the noise possible.

Extension of Gaussian Linear Regression

Generalized Linear Model

- Model entirely characterized by its mean (up to a scalar nuisance parameter) ($v(\mathbb{E}_\theta[Y]) = \theta$ with v invertible).
- Exponential family: Probability law family P_θ such that the density can be written
$$f(y, \theta, \varphi) = e^{\frac{y\theta - v(\theta)}{\varphi} + w(y,\varphi)}$$
where φ is a nuisance parameter and w a function independent of θ.
- Examples:
  - Gaussian: $f(y, \theta, \varphi) = e^{-\frac{y\theta - \theta^2/2}{\varphi} - \frac{y^2/2}{\varphi}}$
  - Bernoulli: $f(y, \theta) = e^{y\theta - \ln(1+e^\theta)}$ $(\theta = \ln p/(1-p))$
  - Poisson: $f(y, \theta) = e^{(y\theta - e^\theta) + \ln(y!)}$ $(\theta = \ln \lambda)$
- Linear Conditional model: $Y|\underline{X} \sim P_{\underline{X}^\top \beta} \ldots$

- ML fit of the parameters

Outline

Non Parametric Conditional Estimation

- **Idea:** Estimate $Y|\underline{X}$ or $\mathbb{E}[Y|\underline{X}]$ directly without resorting to an explicit parametric model.

Non Parametric Conditional Estimation

- Two heuristics:
  - $Y|\underline{X}$ (or $\mathbb{E}[Y|\underline{X}]$) is almost constant (or simple) in a neighborhood of $\underline{X}$. (Kernel methods)
  - $Y|\underline{X}$ (or $\mathbb{E}[Y|\underline{X}]$) can be approximated by a model whose dimension depends on the complexity and the number of observation. (Quite similar to parametric model plus model selection...)

- Focus on **kernel methods**!

Kernel Methods

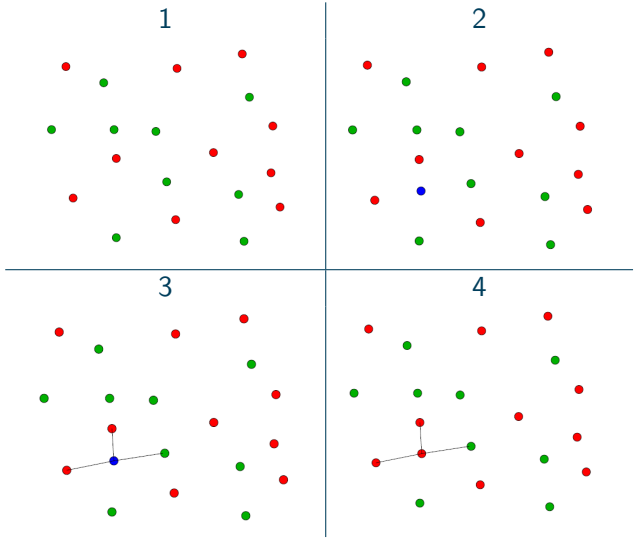
- **Idea:** The behavior of $Y|\underline{X}$ is locally *constant* or simple!

Kernel

- Choose a kernel K (think of a weighted neighborhood).
- For each $\widetilde{\underline{X}}$, compute a simple localized estimate of $Y|\underline{X}$
- Use this local estimate to take the decision

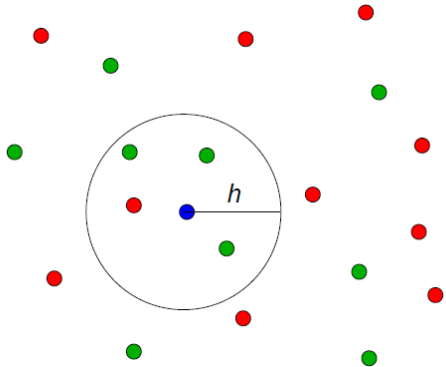
- In regression, estimation of $\mathbb{E}[Y|\underline{X}]$ is sufficient.

Example: k Nearest-Neighbors (with $k = 3$)



Source: Unknown

Example: k Nearest-Neighbors (with $k = 4$)



Source: Unknown

k Nearest-Neighbors

- Neighborhood $\mathcal{V}_{\underline{x}}$ of $\underline{x}$: k learning samples closest from $\underline{x}$.

k-NN as local conditional density estimate

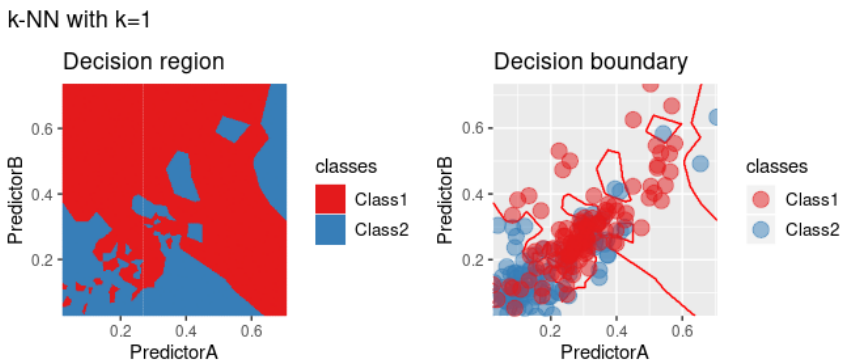
$$\widehat{\mathbb{P}(Y=1|\underline{X})} = \frac{\sum_{\underline{X}_i \in \mathcal{V}_{\underline{X}}} \mathbf{1}_{\{Y_i=+1\}}}{|\mathcal{V}_{\underline{X}}|}$$

- KNN Classifier:

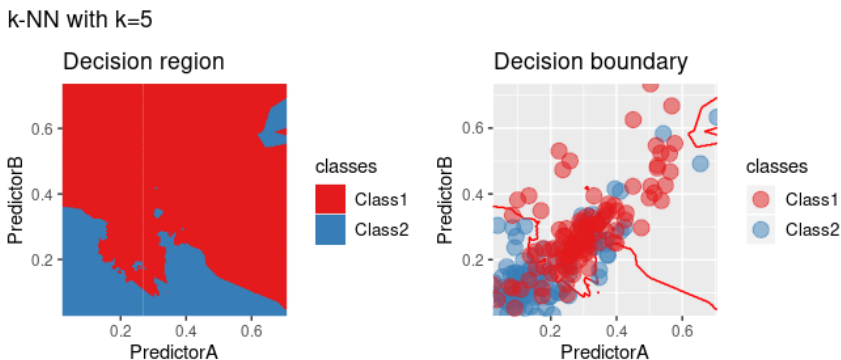
$$\widehat{f}_{KNN}(\underline{X}) = \begin{cases} +1 & \text{if } \widehat{\mathbb{P}(Y=1|\underline{X})} \geq \widehat{\mathbb{P}(Y=-1|\underline{X})} \\ -1 & \text{otherwise} \end{cases}$$

- **Lazy learning**: all the computations have to be done at prediction time.
- **Remark:** You can also use your favorite kernel estimator...

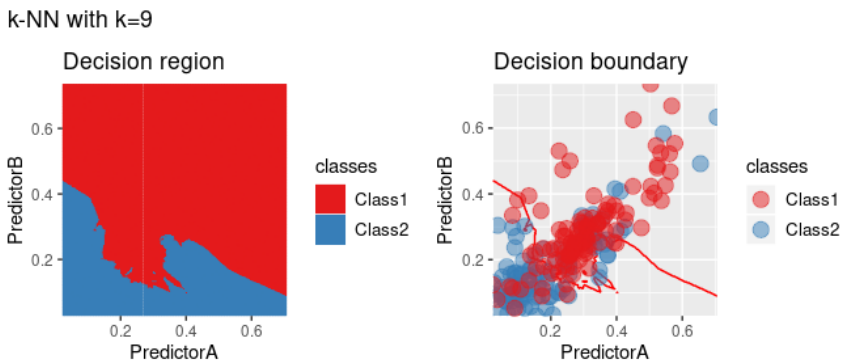
Example: KNN


k-NN with k=1
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

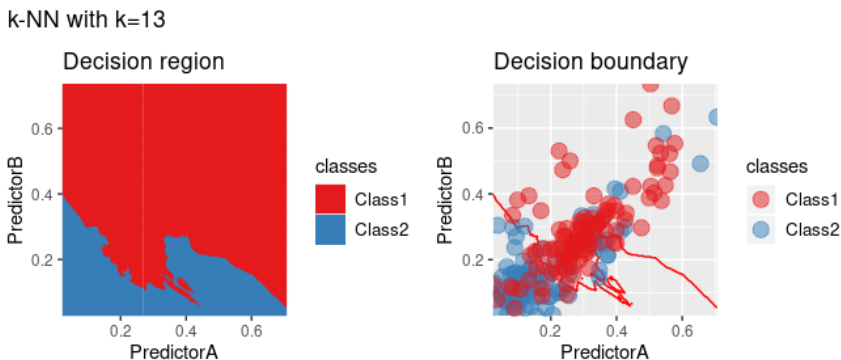
Example: KNN


k-NN with k=5
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

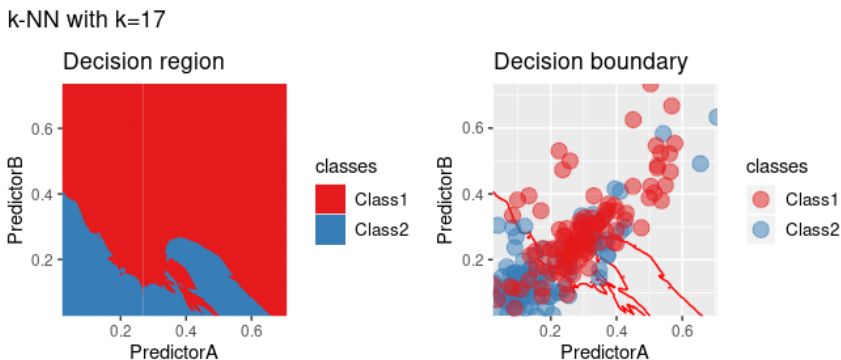
Example: KNN


k-NN with k=9
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

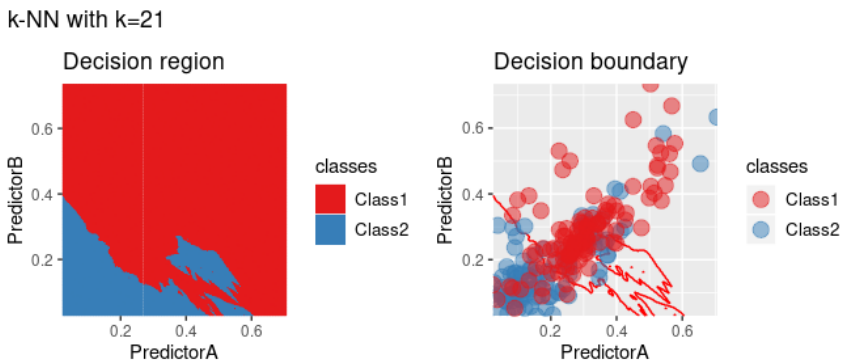
Example: KNN


k-NN with k=13
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

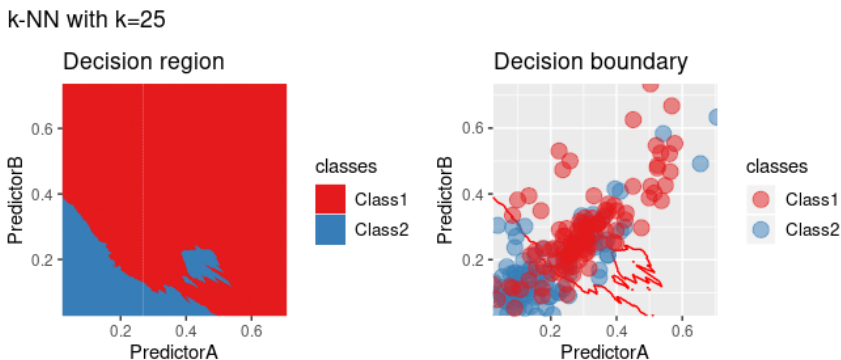
Example: KNN


k-NN with k=17
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

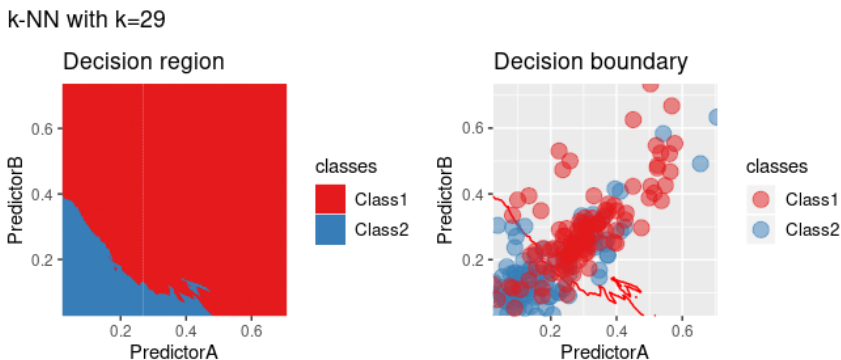
Example: KNN


k-NN with k=21
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

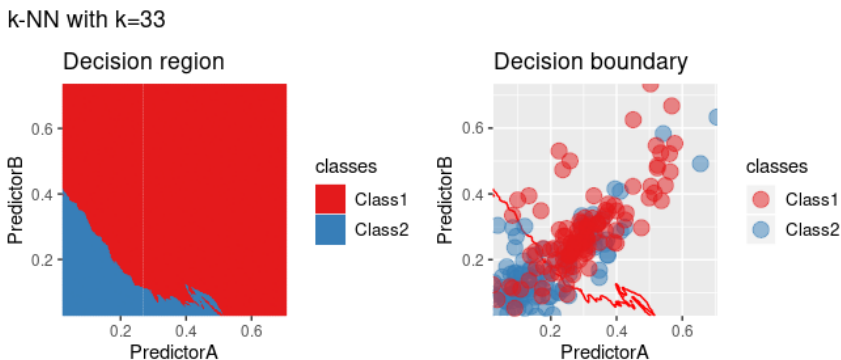
Example: KNN


k-NN with k=25
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

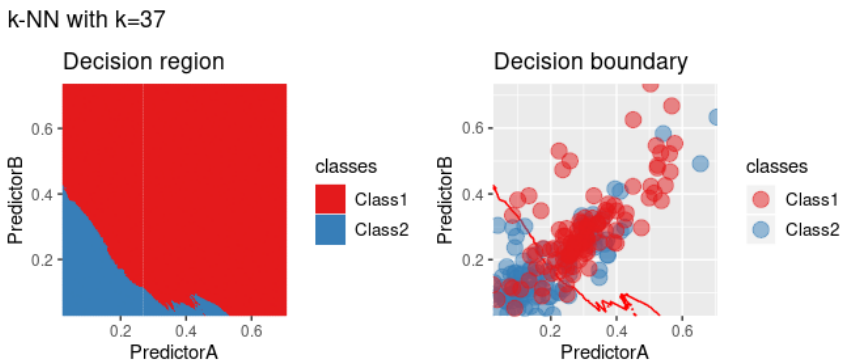
Example: KNN


k-NN with k=29
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

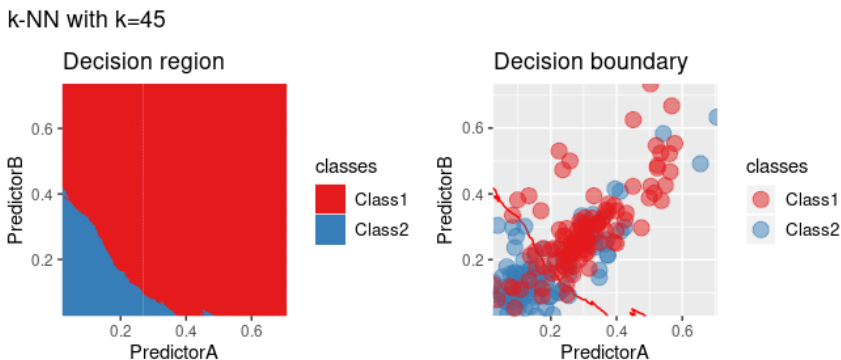
Example: KNN


k-NN with k=33
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

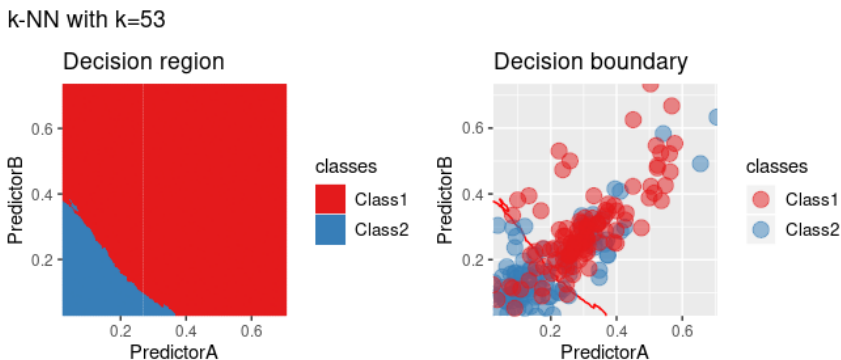
Example: KNN


k-NN with k=37
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

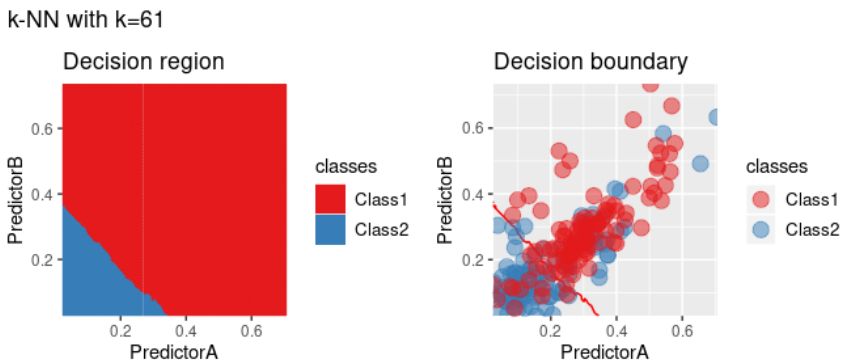
Example: KNN


k-NN with k=45
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

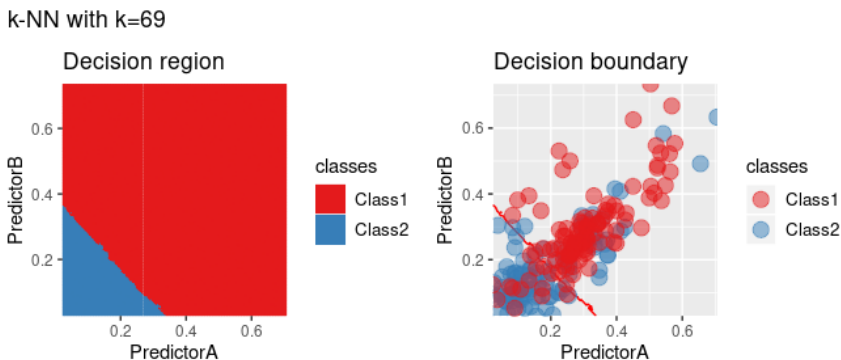
Example: KNN


k-NN with k=53
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

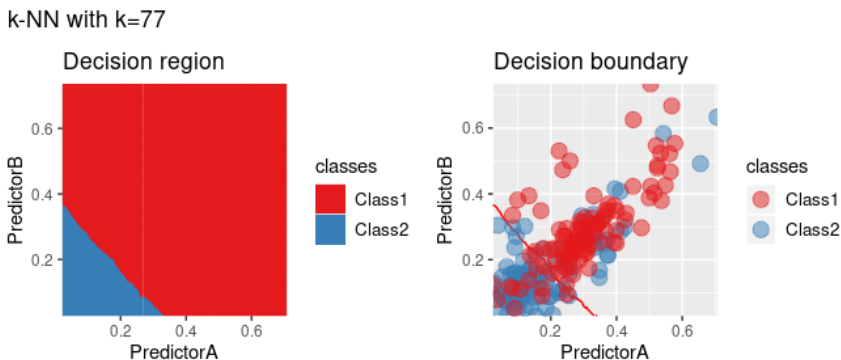
Example: KNN


k-NN with k=61
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

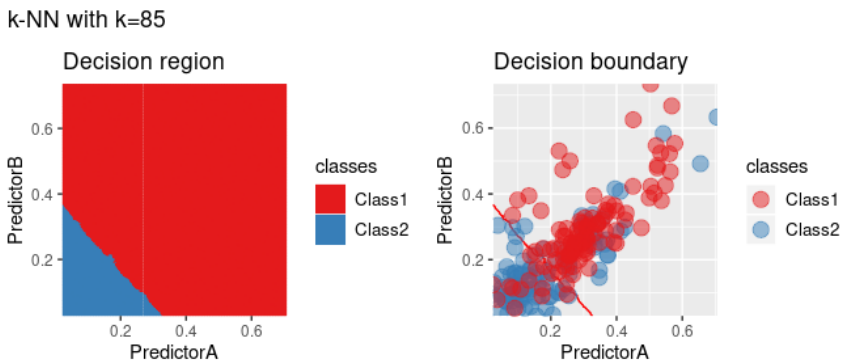
Example: KNN


k-NN with k=69
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

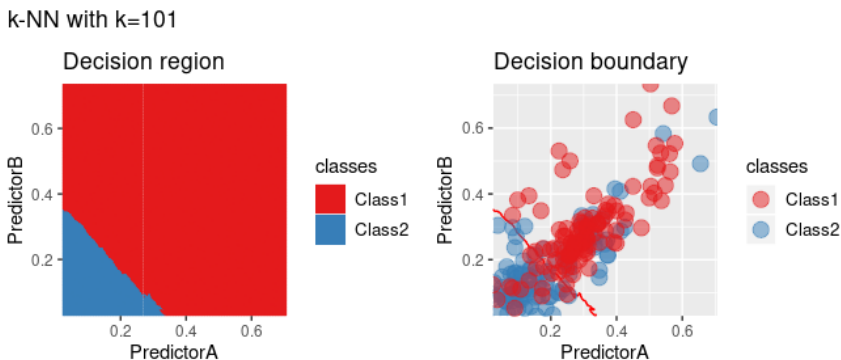
Example: KNN


k-NN with k=77
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

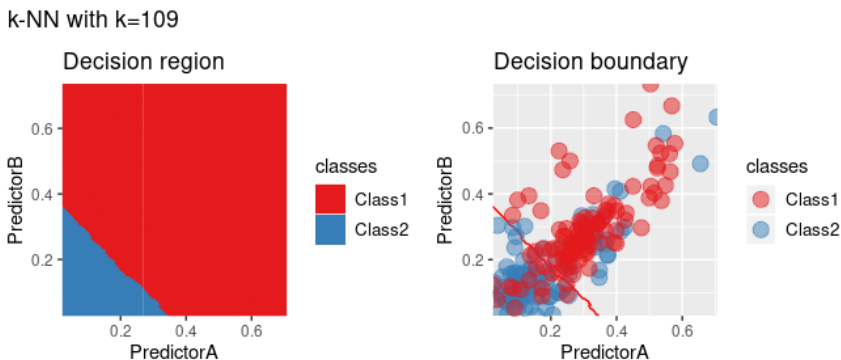
Example: KNN


k-NN with k=85
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

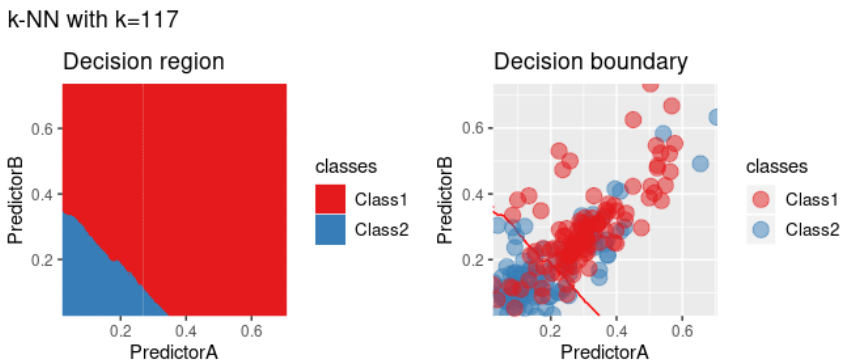
Example: KNN


k-NN with k=101
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

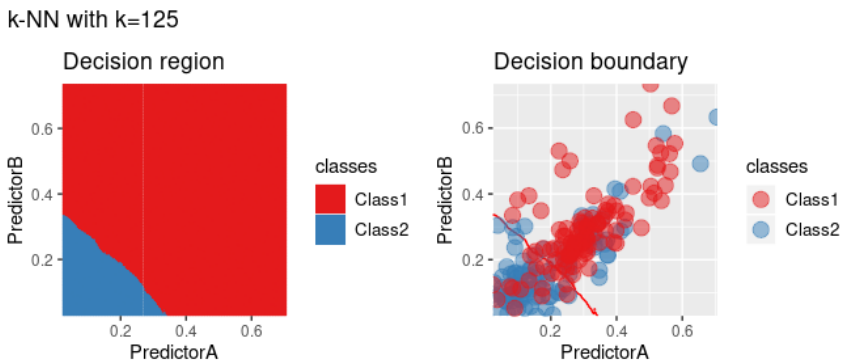
Example: KNN


k-NN with k=109
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

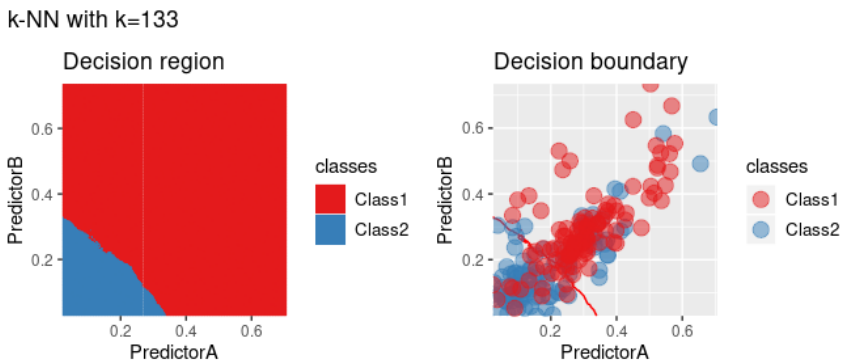
Example: KNN


k-NN with k=117
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

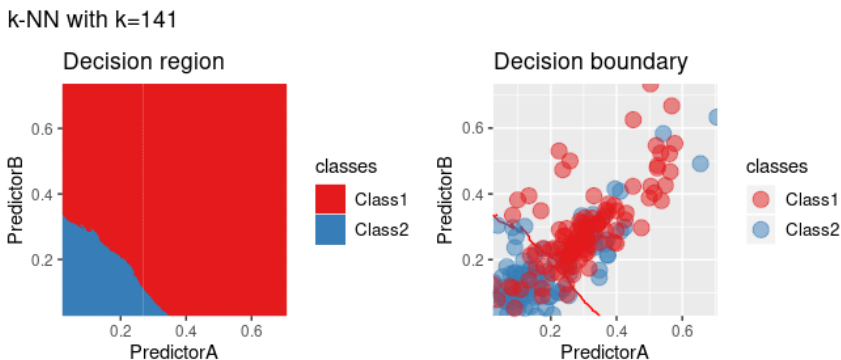
Example: KNN


k-NN with k=125
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

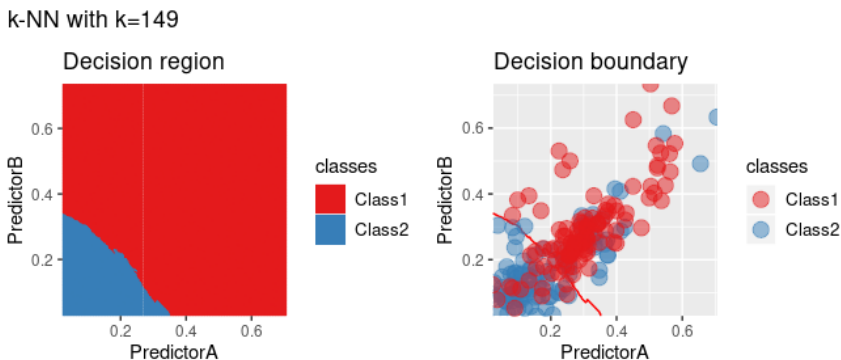
Example: KNN


k-NN with k=133
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

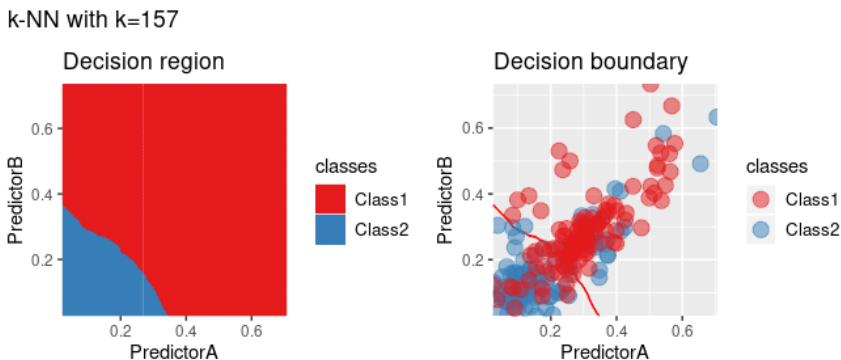
Example: KNN


k-NN with k=141
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

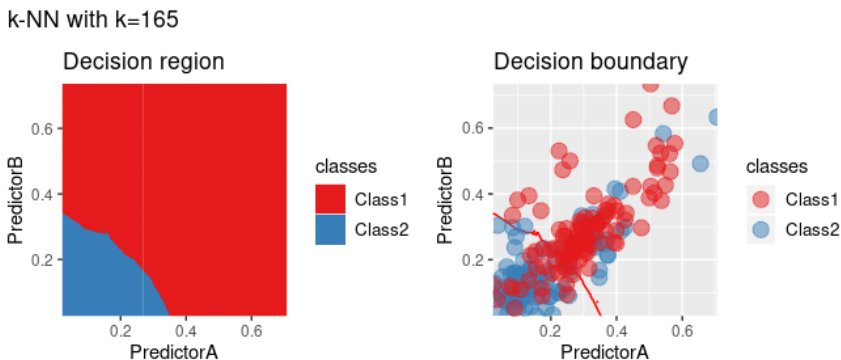
Example: KNN


k-NN with k=149
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

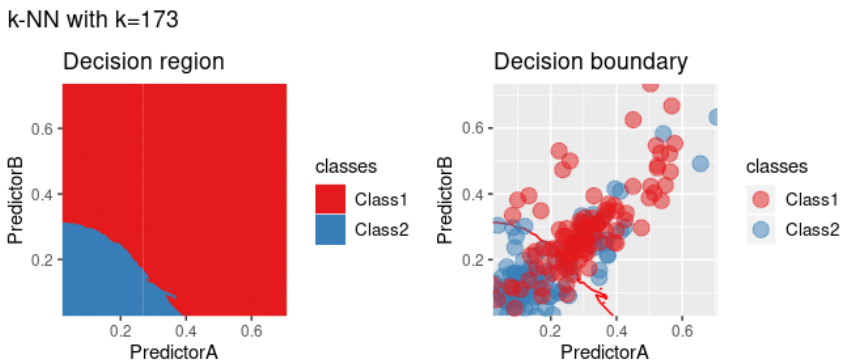
Example: KNN


k-NN with k=157
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

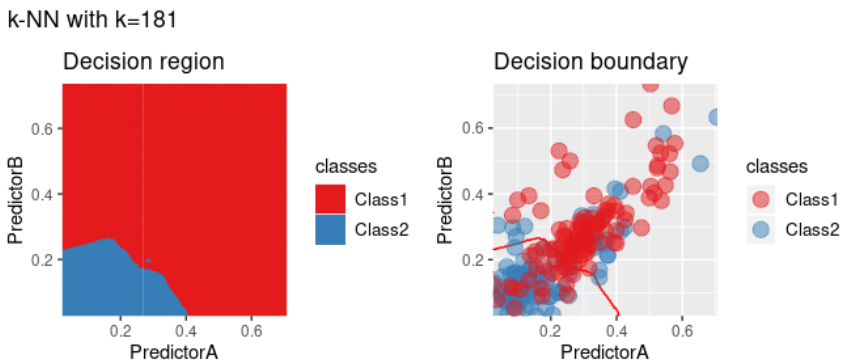
Example: KNN


k-NN with k=165
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

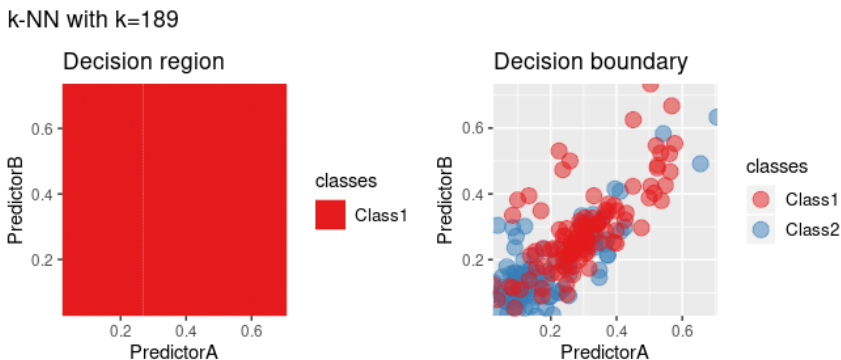
Example: KNN


k-NN with k=173
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

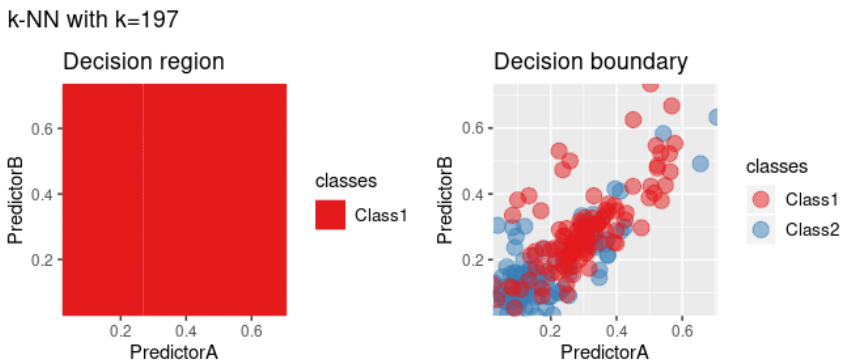
Example: KNN


k-NN with k=181
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

Example: KNN


k-NN with k=189
Decision region
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Decision boundary
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2

Example: KNN


k-NN with k=197
Decision region
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Decision boundary
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2

Regression and Local Averaging

A naive idea

- $\mathbb{E}[Y|\underline{X}]$ can be approximated by a local average:

$$\widehat{f}(\underline{X}) = \frac{1}{|\{\underline{X}_i \in \mathcal{N}(\underline{X})\}|} \sum_{\underline{X}_i \in \mathcal{N}(\underline{X})} Y_i$$

  where $\mathcal{B}(\underline{X})$ is a neighborhood of $\underline{X}$.

- **Heuristic:**
  - If $\underline{X} \to \mathbb{E}[Y|\underline{X}]$ is regular then

$$\mathbb{E}[Y|\underline{X}] \simeq \mathbb{E}\big[\mathbb{E}\big[Y|\underline{X}'\big]\,|\underline{X}' \in \mathcal{N}(\underline{X})\big] = \mathbb{E}\big[Y|\underline{X}' \in \mathcal{N}(\underline{X})\big]$$

  - Replace an expectation by an empirical average:

$$\mathbb{E}\big[Y|\underline{X}' \in \mathcal{N}(\underline{X})\big] \simeq \frac{1}{|\{\underline{X}_i \in \mathcal{N}(\underline{X})\}|} \sum_{\underline{X}_i \in \mathcal{N}(\underline{X})} Y_i$$

Regression and Local Averaging

Neighborhood and Size

- Most classical choice: $\mathcal{N}(\underline{X}) = \{\underline{X}', \|\underline{X} - \underline{X}'\| \leq h\}$ where $\|.\|$ is a (pseudo) norm and h a size (bandwidth) parameter.
- In principle, the norm and h could vary with $\underline{X}$, and the norm can be replaced by a (pseudo) distance.
- Focus here on a fixed distance with a fixed bandwidth h cased.

Bandwidth Heuristic

- A **large bandwidth** ensures that the average is taken on many samples and thus the **variance is small**...
- **A small bandwidth** is thus that the approximation $\mathbb{E}[Y|\underline{X}] \simeq \mathbb{E}[Y|\underline{X}' \in \mathcal{N}(\underline{X})]$ is more accurate **(small bias)**.

Weighted Local Averaging

Weighted Local Average

- Replace the neighborhood $\mathcal{N}(\underline{X})$ by a decaying **window function** $w(\underline{X}, \underline{X}')$.
- $\mathbb{E}[Y|\underline{X}]$ can be approximated by a **weighted local average**:

$$\widehat{f}(\underline{X}) = \frac{\sum_i w(\underline{X}, \underline{X}'_i) Y_i}{\sum_i w(\underline{X}, \underline{X}'_i)}.$$

Kernel

- Most classical choice: $w(\underline{X}, \underline{X}') = K\left(\frac{\underline{X}-\underline{X}'}{h}\right)$ where h the bandwidth is a scale parameter.
- Examples:
  - **Box kernel:** $K(t) = \mathbf{1}_{\|t\| \leq 1}$ (Neighborhood)
  - **Triangular kernel:** $K(t) = \max(1 - \|t\|, 0)$.
  - **Gaussian kernel:** $K(t) = e^{-t^2/2}$
- **Rk:** K and λK yields the same estimate.

Link with Density Estimation

Density Estimation

- How to estimate the density p of $\underline{X}$ with respect to the Lebesgue measure from an i.i.d. sample $(\underline{X}_1, \ldots, \underline{X}_n)$.
- **Parametric approach:** density has a known parameterized shape and estimate those parameters.
- **Nonparametric approach:** density has a no known parameterized shape and
  - Approximate it by a parametric one, whose parameters can be estimated
  - Estimate directly the density
- Important **nonparametric statistic topic**!
- Used in generative modeling...

Link with Density Estimation

Kernel Density Estimation (Parzen)

- Choose a positive kernel K such that $\int K(x)dx = 1$
- Use as an estimate

$$\widehat{p}(\underline{X}) = \frac{1}{n}\sum_{i=1}^{n} K(\underline{X} - \underline{X}_i)$$

- If $K = \frac{1}{Z_h}\mathbf{1}_{\|t\|\leq h}$, easy interpretation as a **local empirical density** of samples!
- General K corresponds to a **smoothed version**.
- Often $K_h(t) = \frac{1}{h^d}K(t/h)$ and let

$$\widehat{p}_h(\underline{X}) = \frac{1}{n}\sum_{i=1}^{n} K_h(\underline{X} - \underline{X}_i)$$

Link with Density Estimation

Properties

- **Error decomposition:**

$$\mathbb{E}\left[|p(\underline{X}) - \widehat{p}_h(\underline{X})|^2\right] = \mathbb{E}[p(\underline{X}) - \widehat{p}_h(\underline{X})]^2 + \mathbb{V}\text{ar}\left[p(\underline{X}) - \widehat{p}_h(\underline{X})\right]$$

- **Bias:**

$$\mathbb{E}[p(\underline{X}) - \widehat{p}_h(\underline{X})] = p(\underline{X}) - (K_h * p)(\underline{X})$$

- **Variance:** if p is upper bounded by $p_{\max}$ then

$$\mathbb{V}\text{ar}\left[p(\underline{X}) - \widehat{p}_h(\underline{X})\right] \leq \frac{p_{\max} \int K_h^2(x)dx}{nh^d}$$

Link with Density Estimation

Bandwidth choice

- A small h leads to a small bias but a large variance...
- A large h leads to a small variance but a large bias...
- Theoretical analysis possible!

From Density Estimation to Regression

Nadaraya-Watson Heuristic

- Provided all the **densities** exist

$$\mathbb{E}[Y|\underline{X}] = \frac{\int Yp(\underline{X}, Y)dY}{\int p(Y, \underline{X})dY} = \frac{\int Yp(\underline{X}, Y)dY}{p(\underline{X})}$$

- Replace the unknown densities by their **estimates**:

$$\widehat{p}(\underline{X}) = \frac{1}{n}\sum_{i=1}^{n} K(\underline{X} - \underline{X}_i)$$

$$\widehat{p}(\underline{X}, Y) = \frac{1}{n}\sum_{i=1}^{n} K(\underline{X} - \underline{X}_i)K'(Y - Y_i)$$

- Now if K' is a kernel such that $\int YK'(Y)dY = 0$ then

$$\int Y\widehat{p}(\underline{X}, Y)dY = \frac{1}{n}\sum_{i=1}^{n} K(\underline{X} - \underline{X}_i)Y_i$$

From Density Estimation to Regression

Nadaraya-Watson

- Resulting estimator of $\mathbb{E}[Y|\underline{X}]$

$$\widehat{f}(\underline{X}) = \frac{\sum_{i=1}^{n} Y_i K_h(\underline{X} - \underline{X}_i)}{\sum_{i=1}^{n} K_h(\underline{X} - \underline{X}_i)}$$

- Same **local weighted average** estimator!

Bandwidth Choice

- Bandwidth h of K allows to **balance between bias and variance**.
- Theoretical analysis of the error is possible.
- The smoother the densities the easier the estimation but the optimal bandwidth depends on the unknown regularity!

Local Linear Estimation

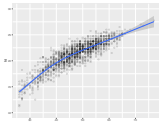
Another Point of View on Kernel

- Nadaraya-Watson estimator:
$$\widehat{f}(\underline{X}) = \frac{\sum_{i=1}^n Y_i K_h(\underline{X} - \underline{X}_i)}{\sum_{i=1}^n K_h(\underline{X} - \underline{X}_i)}$$
- Can be view as a **minimizer** of
$$\sum_{i=1}^n |Y_i - \beta|^2 K_h(\underline{X} - \underline{X}_i)$$
- **Local regression** of order 0.

Local Linear Model

- Estimate $\mathbb{E}[Y|\underline{X}]$ by $\widehat{f}(\underline{X}) = \phi(\underline{X})^\top \widehat{\beta}(\underline{X})$ where ϕ is any function of $\underline{X}$ and $\widehat{\beta}(\underline{X})$ is the minimizer of
$$\sum_{i=1}^n |Y_i - \phi(\underline{X}_i)^\top \beta|^2 K_h(\underline{X} - \underline{X}_i).$$

LOESS: LOcal polynomial regrESSion



1D Nonparametric Regression

- Assume that $\underline{X} \in \mathbb{R}$ and let $\phi(\underline{X}) = (1, \underline{X}, \ldots, \underline{X}^d)$.
- **LOESS estimate:** $\widehat{f}(\underline{X}) = \sum_{j=0}^{d} \widehat{\beta}(\underline{X}^{(j)})\underline{X}^j$ with $\widehat{\beta}(\underline{X})$ minimizing

$$\sum_{i=1}^{n} |Y_i - \sum_{j=0}^{d} \beta^{(j)} \underline{X}_i^j|^2 K_h(\underline{X} - \underline{X}_i).$$

- Most classical kernel used: Tricubic kernel

$$K(t) = \max(1 - |t|^3, 0)^3$$

- Most classical degree: 2...
- Local bandwidth choice such that a proportion of points belongs to the window.

Outline

Fully Generative Modeling

- **Idea:** If one knows the law of $(\underline{X}, Y)$ everything is easy!

Bayes formula

- With a slight abuse of notation,

$$\begin{aligned}\mathbb{P}(Y|\underline{X}) &= \frac{\mathbb{P}((\underline{X}, Y))}{\mathbb{P}(\underline{X})} \\ &= \frac{\mathbb{P}(\underline{X}|Y)\,\mathbb{P}(Y)}{\mathbb{P}(\underline{X})}\end{aligned}$$

- **Generative Modeling:**
  - Propose a model for $(\underline{X}, Y)$ (or equivalently $\underline{X}|Y$ and Y),
  - Estimate it as a density estimation problem,
  - Plug the estimate in the Bayes formula
  - Plug the conditional estimate in the Bayes *classifier*.
- **Rk:** Require to estimate $(\underline{X}, Y)$ rather than only $Y|\underline{X}$!
- Great flexibility in the model design but may lead to complex computation.

Fully Generative Modeling

- Simpler setting in classification!

Bayes formula

$$\mathbb{P}(Y = k|\underline{X}) = \frac{\mathbb{P}(\underline{X}|Y = k)\,\mathbb{P}(Y = k)}{\mathbb{P}(\underline{X})}$$

- Binary Bayes classifier (the best solution)

$$f^\star(\underline{X}) = \begin{cases} +1 & \text{if } \mathbb{P}(Y = 1|\underline{X}) \geq \mathbb{P}(Y = -1|\underline{X}) \\ -1 & \text{otherwise} \end{cases}$$

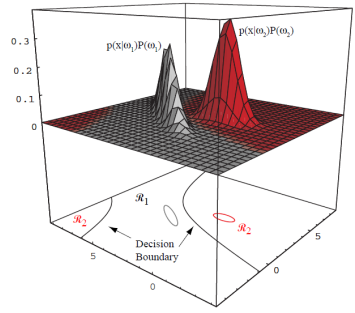
- **Heuristic**: Estimate those quantities and plug the estimations.
- By using different models/estimators for $\mathbb{P}(\underline{X}|Y)$, we get different classifiers.
- **Rk:** No need to renormalize by $\mathbb{P}(\underline{X})$ to take the decision!

Discriminant Analysis (Gaussian model)

- The densities are modeled as multivariate normal, i.e.,
$$\mathbb{P}(\underline{X}|Y=k) \sim \mathcal{N}_{\mu_k,\Sigma_k}$$
- Discriminant functions: $\boldsymbol{g_k(\underline{X}) = \ln(\mathbb{P}(\underline{X}|Y=k)) + \ln(\mathbb{P}(Y=k))}$
$$g_k(\underline{X}) = -\frac{1}{2}(\underline{X}-\mu_k)^\top \Sigma_k^{-1}(\underline{X}-\mu_k) - \frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln(|\Sigma_k|) + \ln(\mathbb{P}(Y=k))$$
- QDA (different Σ_k in each class) and LDA ($\Sigma_k = \Sigma$ for all k)

- **Beware: this model can be false but the methodology remains valid!**

Discriminant Analysis

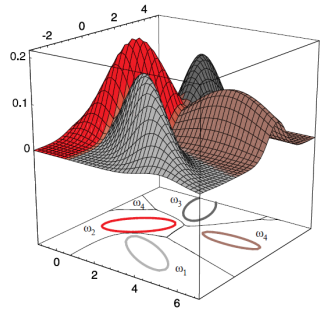


Quadratic Discriminant Analysis

- The probability densities are Gaussian
- The effect of any decision rule is to divide the feature space into some decision regions $\mathcal{R}_1, \mathcal{R}_2$
- The regions are separated by decision boundaries

Source: A. Fermin

Discriminant Analysis



Quadratic Discriminant Analysis

- The probability densities are Gaussian
- The effect of any decision rule is to divide the feature space into some decision regions $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_c$
- The regions are separated by decision boundaries

Source: A. Fermin

Estimation

In practice, we will need to estimate μ_k, Σ_k and $\mathbb{P}_k := \mathbb{P}(Y = k)$

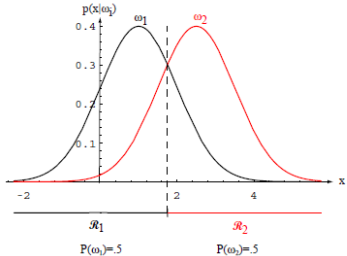
- The estimate proportion $\widehat{\mathbb{P}(Y = k)} = \frac{n_k}{n} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{Y_i = k\}}$
- Maximum likelihood estimate of $\widehat{\mu_k}$ and $\widehat{\Sigma_k}$ (explicit formulas)

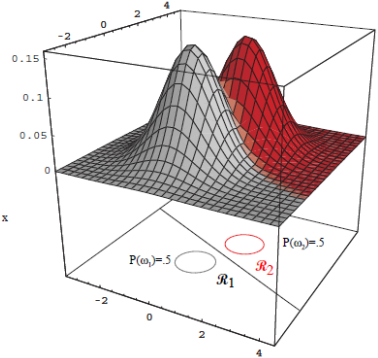
- DA classifier

$$\widehat{f}_G(\underline{X}) = \begin{cases} +1 & \text{if } \widehat{g}_{+1}(\underline{X}) \geq \widehat{g}_{-1}(\underline{X}) \\ -1 & \text{otherwise} \end{cases}$$

- Decision boundaries: quadratic = degree 2 polynomials.
- If one imposes $\Sigma_{-1} = \Sigma_1 = \Sigma$ then the decision boundaries is a linear hyperplane.

Discriminant Analysis



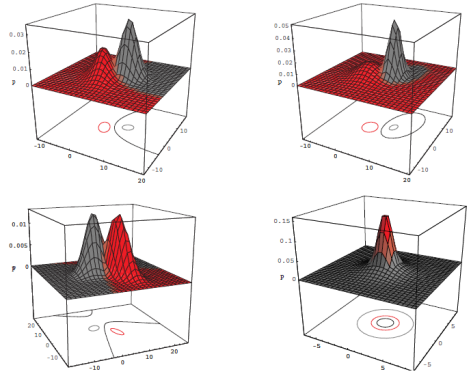


Linear Discriminant Analysis

- $\Sigma_{\omega_1} = \Sigma_{\omega_2} = \Sigma$
- The decision boundaries are linear hyperplanes

Source: A. Fermin

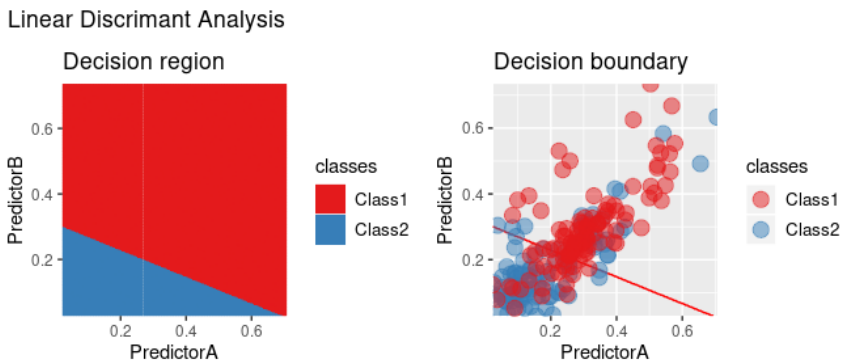
Discriminant Analysis



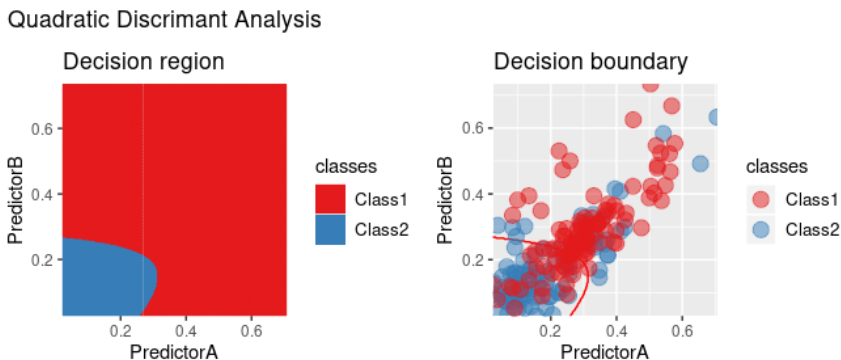
Quadratic Discriminant Analysis

- $\Sigma_{\omega_1} \neq \Sigma_{\omega_2}$
- Arbitrary Gaussian distributions lead to Bayes decision boundaries that are general quadratics.

Example: LDA


Linear Discrimant Analysis
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

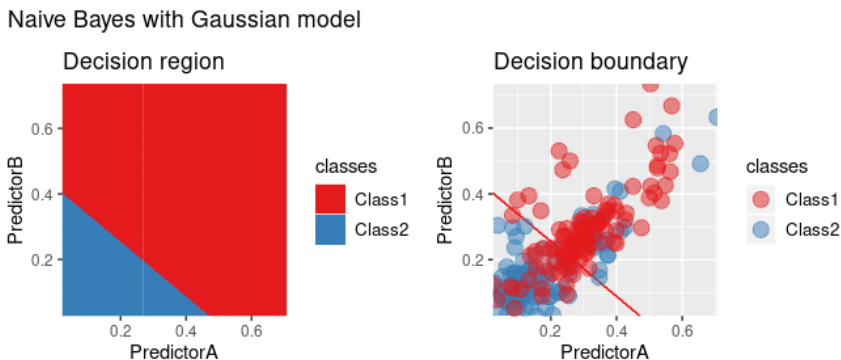
Example: QDA


Quadratic Discrimant Analysis
Decision region
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2
Decision boundary
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2

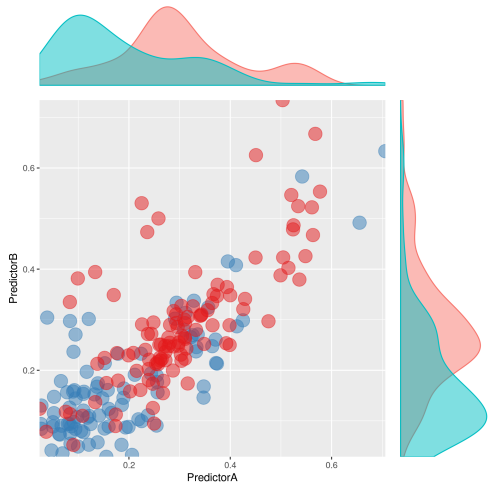
Naive Bayes

- Classical algorithm using a crude modeling for $\mathbb{P}(\underline{X}|Y)$:
  - Feature **independence** assumption:
$$\mathbb{P}(\underline{X}|Y) = \prod_{l=1}^{d} \mathbb{P}\left(\underline{X}^{(l)}\middle|Y\right)$$
  - Simple featurewise model: binomial if binary, multinomial if finite and Gaussian if continuous
- If all features are continuous, similar to the previous Gaussian but with a **diagonal covariance matrix**!
- Very simple learning even in **very high dimension!**

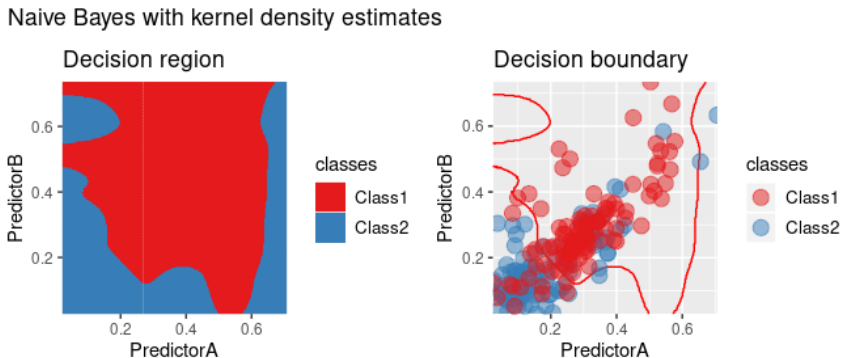
Example: Naive Bayes



Naive Bayes with Density Estimation



Example: Naive Bayes



Other Models

- Other models of the world!

Bayesian Approach

- Generative Model plus prior on the parameters
- Inference thanks to the Bayes formula

Graphical Models

- Markov type models on Graphs

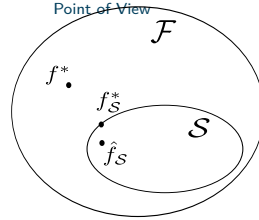
Gaussian Processes

- Multivariate Gaussian models

- ...

Outline

Bias-Variance Dilemma



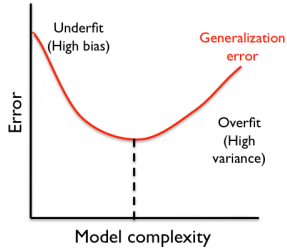
- General setting:
  - $\mathcal{F} = \{\text{measurable functions } \mathcal{X} \to \mathcal{Y}\}$
  - Best solution: $f^\star = \text{argmin}_{f \in \mathcal{F}} \mathcal{R}(f)$
  - Class $\mathcal{S} \subset \mathcal{F}$ of functions
  - Ideal target in $\mathcal{S}$: $f^\star_{\mathcal{S}} = \text{argmin}_{f \in \mathcal{S}} \mathcal{R}(f)$
  - Estimate in $\mathcal{S}$: $\widehat{f}_{\mathcal{S}}$ obtained with some procedure

Approximation error and estimation error (Bias/Variance)

$$\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star) = \underbrace{\mathcal{R}(f^\star_{\mathcal{S}}) - \mathcal{R}(f^\star)}_{\text{Approximation error}} + \underbrace{\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star_{\mathcal{S}})}_{\text{Estimation error}}$$

- Approx. error can be large if the model $\mathcal{S}$ is not suitable.
- Estimation error can be large if the model is complex.

Under-fitting / Over-fitting Issue



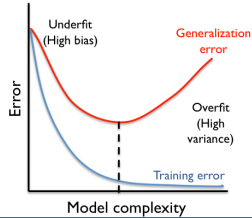
- Different behavior for different model complexity
- **Low complexity model** are easily learned but the approximation error (**bias**) may be large (**Under-fit**).
- **High complexity model** may contain a good ideal target but the estimation error (**variance**) can be large (**Over-fit**)

Bias-variance trade-off $\iff$ avoid **overfitting** and **underfitting**

- **Rk:** Better to think in term of method (including feature engineering and specific algorithm) rather than only of model.

Source: Unknown

Training Risk Issue



Risk behaviour

- Learning/training risk (empirical risk on the learning/training set) decays when the complexity of the **method** increases.
- Quite different behavior when the risk is computed on new observations (generalization risk).

- Overfit for complex methods: parameters learned are too specific to the learning set!
- General situation! (Think of polynomial fit...)
- Need to use a different criterion than the training risk!

Source: JMP

Risk Estimation vs Method Selection

Predictor Risk Estimation

- **Goal:** Given a predictor f assess its quality.
- **Method:** Hold-out risk computation (/ Empirical risk correction).
- **Usage:** Compute an estimate of the risk of a selected f using a **test set** to be used to monitor it in the future.

- Basic block very well understood.

Method Selection

- **Goal:** Given a ML method assess its quality.
- **Method:** Cross Validation (/ Empirical risk correction)
- **Usage:** Compute risk estimates for several ML methods using **training/validation sets** to choose the most promising one.

- Estimates can be pointwise or better intervals.
- Multiple test issues in method selection.

Cross Validation and Empirical Risk Correction

Two Approaches

- **Cross validation:** Use empirical risk criterion but on independent data, very efficient (and almost always used in practice!) but slightly biased as its target uses only a fraction of the data.
- **Correction approach:** use empirical risk criterion but *correct* it with a term increasing with the complexity of $\mathcal{S}$

$$R_n(\widehat{f}_{\mathcal{S}}) \to R_n(\widehat{f}_{\mathcal{S}}) + \mathrm{cor}(\mathcal{S})$$

and choose the method with the smallest corrected risk.

Which loss to use?

- The loss used in the risk: most natural!
- The loss used to estimate $\widehat{\theta}$: penalized estimation!

- Other performance measure can be used.

Cross Validation

- **Very simple idea:** use a second learning/verification set to compute a verification risk.
- Sufficient to remove the dependency issue!
- Implicit random design setting...

Cross Validation

- Use $(1-\epsilon) \times n$ observations to train and $\epsilon \times n$ to verify!
- Possible issues:
  - Validation for a learning set of size $(1-\epsilon) \times n$ instead of n ?
  - Unstable risk estimate if ϵn is too small ?

- Most classical variations:
  - Hold Out,
  - Leave One Out,
  - V-fold cross validation.

Source: M. Kühn

Hold Out

Principle

- Split the dataset $\mathcal{D}$ in 2 sets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ of size $n \times (1-\epsilon)$ and $n \times \epsilon$.
- Learn $\widehat{f}^{HO}$ from the subset $\mathcal{D}_{\text{train}}$.
- Compute the empirical risk on the subset $\mathcal{D}_{\text{test}}$:

$$\mathcal{R}_n^{HO}(\widehat{f}^{HO}) = \frac{1}{n\epsilon} \sum_{(\underline{X}_i, Y_i) \in \mathcal{D}_{\text{test}}} \ell(Y_i, \widehat{f}^{HO}(\underline{X}_i))$$

Predictor Risk Estimation

- Use $\hat{f}^{HO}$ as predictor.
- Use $\mathcal{R}_n^{HO}(\widehat{f}^{HO})$ as an estimate of the risk of this estimator.

Method Selection by Cross Validation

- Compute $\mathcal{R}_n^{HO}(\widehat{f}_{\mathcal{S}}^{HO})$ for all the considered methods,
- Select the method with the smallest CV risk,
- Reestimate the $\widehat{f}_{\mathcal{S}}$ with all the data.

Hold Out

Principle

- Split the dataset $\mathcal{D}$ in 2 sets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ of size $n \times (1-\epsilon)$ and $n \times \epsilon$.
- Learn $\widehat{f}^{HO}$ from the subset $\mathcal{D}_{\text{train}}$.
- Compute the empirical risk on the subset $\mathcal{D}_{\text{test}}$:

$$\mathcal{R}_n^{HO}(\widehat{f}^{HO}) = \frac{1}{n\epsilon} \sum_{(\underline{X}_i, Y_i) \in \mathcal{D}_{\text{test}}} \ell(Y_i, \widehat{f}^{HO}(\underline{X}_i))$$

- Only possible setting for risk estimation.

Hold Out Limitation for Method Selection

- Biased toward simpler method as the estimation does not use all the data initially.
- Learning variability of $\mathcal{R}_n^{HO}(\widehat{f}^{HO})$ not taken into account.

V-fold Cross Validation

Principle

- Split the dataset $\mathcal{D}$ in V sets $\mathcal{D}_v$ of almost equals size.
- For $v \in \{1, .., V\}$:
  - Learn $\widehat{f}^{-v}$ from the dataset $\mathcal{D}$ minus the set $\mathcal{D}_v$.
  - Compute the empirical risk:

$$\mathcal{R}_n^{-v}(\widehat{f}^{-v}) = \frac{1}{n_v} \sum_{(\underline{X}_i, Y_i) \in \mathcal{D}_v} \ell(Y_i, \widehat{f}^{-v}(\underline{X}_i))$$

- Compute the average empirical risk:

$$\mathcal{R}_n^{CV}(\widehat{f}) = \frac{1}{V} \sum_{v=1}^{V} \mathcal{R}_n^{-v}(\widehat{f}^{-v})$$

- Estimation of the quality of a method not of a given predictor.
- Leave One Out : V = n.

Source: M. Kühn

V-fold Cross Validation

Analysis (when n is a multiple of V)

- The $\mathcal{R}_n^{-v}(\widehat{f}^{-v})$ are identically distributed variable but are not independent!
- Consequence:

$$\mathbb{E}\left[\mathcal{R}_n^{CV}(\widehat{f})\right] = \mathbb{E}\left[\mathcal{R}_n^{-v}(\widehat{f}^{-v})\right]$$

$$\mathbb{V}\text{ar}\left[\mathcal{R}_n^{CV}(\widehat{f})\right] = \frac{1}{V}\mathbb{V}\text{ar}\left[\mathcal{R}_n^{-v}(\widehat{f}^{-v})\right] + (1-\frac{1}{V})\,\mathbb{C}\text{ov}\left[\mathcal{R}_n^{-v}(\widehat{f}^{-v}), \mathcal{R}_n^{-v'}(\widehat{f}^{-v'})\right]$$

- Average risk for a sample of size $(1-\frac{1}{V})n$.
- Variance term much more complex to analyze!
- Fine analysis shows that the larger V the better...

- Accuracy/Speed tradeoff: $V = 5$ or $V = 10$...

Linear Regression and Leave One Out

- Leave One Out $=$ V fold for $V = n$: very expensive in general.

A fast LOO formula for the linear regression

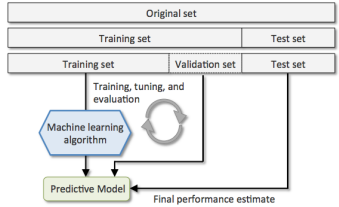
- **Prop:** for the least squares linear regression,

$$\widehat{f}^{-i}(\underline{X}_i) = \frac{\widehat{f}(\underline{X}_i) - h_{ii} Y_i}{1 - h_{ii}}$$

  with h_{ii} the ith diagonal coefficient of the **hat** (projection) matrix.
- Proof based on linear algebra!
- Leads to a fast formula for LOO:

$$\mathcal{R}_n^{LOO}(\widehat{f}) = \frac{1}{n} \sum_{i=1}^{n} \frac{|Y_i - \widehat{f}(\underline{X}_i)|^2}{(1 - h_{ii})^2}$$

Train/Validation/Test



- **Selection Bias Issue:**
  - After method selection, the cross validation is biased.
  - Furthermore, it qualifies the method and not the final predictor.
- Need to (re)estimate the risk of the final predictor.

(Train/Validation)/Test strategy

- **Split** the dataset in two a `(Train/Validation)` and `Test`.
- Use **CV** with the `(Train/Validation)` to **select a method**.
- Train this method on `(Train/Validation)` to **obtain a single predictor**.
- Estimate the **performance of this predictor** on `Test`.

- Every choice made from the data is part of the method!

Source: Shan-Hung Wu & DataLab

Risk Correction

- Empirical loss of an estimator computed on the dataset used to chose it is biased!
- Empirical loss is an optimistic estimate of the true loss.

Risk Correction Heuristic

- Estimate an upper bound of this optimism for a given family.
- Correct the empirical loss by adding this upper bound.

- **Rk:** Finding such an upper bound can be complicated!
- Correction often called a **penalty**.

Penalization

Penalized Loss

- Minimization of

$$\underset{\theta \in \Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f_\theta(\underline{X}_i)) + \text{pen}(\theta)$$

where $\text{pen}(\theta)$ is a risk correction (penalty).

Penalties

- Upper bound of the optimism of the empirical loss
- Depends on the loss and the framework!

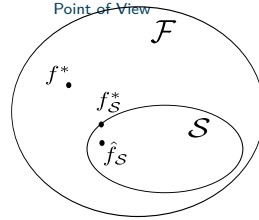
Instantiation

- Mallows Cp: Least Squares with $\text{pen}(\theta) = 2\frac{d}{n}\sigma^2$.
- AIC Heuristics: Maximum Likelihood with $\text{pen}(\theta) = \frac{d}{n}$.
- BIC Heuristics: Maximum Likelohood with $\text{pen}(\theta) = \log(n)\frac{d}{n}$.
- Structural Risk Minimization: Pred. loss and clever penalty.

Outline

Bias-Variance Dilemma

- General setting:
  - $\mathcal{F} = \{\text{measurable functions } \mathcal{X} \to \mathcal{Y}\}$
  - Best solution: $f^\star = \text{argmin}_{f \in \mathcal{F}} \mathcal{R}(f)$
  - Class $\mathcal{S} \subset \mathcal{F}$ of functions
  - Ideal target in $\mathcal{S}$: $f^\star_{\mathcal{S}} = \text{argmin}_{f \in \mathcal{S}} \mathcal{R}(f)$
  - Estimate in $\mathcal{S}$: $\widehat{f}_{\mathcal{S}}$ obtained with some procedure

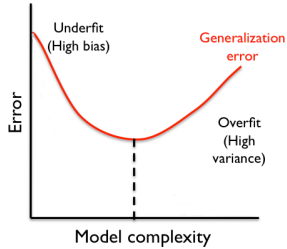


Approximation error and estimation error (Bias/Variance)

$$\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star) = \underbrace{\mathcal{R}(f^\star_{\mathcal{S}}) - \mathcal{R}(f^\star)}_{\text{Approximation error}} + \underbrace{\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star_{\mathcal{S}})}_{\text{Estimation error}}$$

- Approx. error can be large if the model $\mathcal{S}$ is not suitable.
- Estimation error can be large if the model is complex.

Under-fitting / Over-fitting Issue



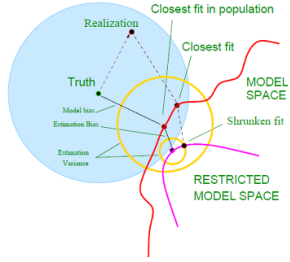
- Different behavior for different model complexity
- **Low complexity model** are easily learned but the approximation error (**bias**) may be large (**Under-fit**).
- **High complexity model** may contain a good ideal target but the estimation error (**variance**) can be large (**Over-fit**)

Bias-variance trade-off $\iff$ avoid **overfitting** and **underfitting**

- **Rk:** Better to think in term of method (including feature engineering and specific algorithm) rather than only of model.

Source: Unknown

Simplified Models



Bias-Variance Issue

- Most complex models may not be the best ones due to the variability of the estimate.

- Naive idea: can we *simplify* our model without loosing too much?
  - by using only a subset of the variables?
  - by forcing the coefficients to be small?
- Can we do better than exploring all possibilities?

Source: Tibshirani et al.

Linear Models

- **Setting**: Gen. linear model = prediction of Y by $h(\underline{x}^\top \beta)$.

Model coefficients

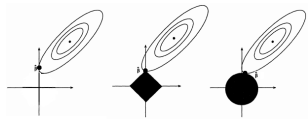
- Model entirely specified by β.
- Coefficientwise:
  - $\beta^{(i)} = 0$ means that the ith covariate is not used.
  - $\beta^{(i)} \sim 0$ means that the ith covariate as a *low* influence...

- If some covariates are useless, better use a simpler model...

Submodels

- *Simplify* the model through a constraint on β!
- Examples:
  - Support: Impose that $\beta^{(i)} = 0$ for $i \notin I$.
  - Support size: Impose that $\|\beta\|_0 = \sum_{i=1}^{d} \mathbf{1}_{\beta^{(i)} \neq 0} < C$
  - Norm: Impose that $\|\beta\|_p < C$ with $1 \leq p$ (Often $p = 2$ or $p = 1$)

Norms and Sparsity



Sparsity

- β is sparse if its number of non-zero coefficients (ℓ_0) is small...
- Easy interpretation in terms of dimension/complexity.

Norm Constraint and Sparsity

- Sparsest solution obtained by definition with the ℓ_0 norm.
- No induced sparsity with the ℓ_2 norm...
- Sparsity with the ℓ_1 norm (can even be proved to be the same as with the ℓ_0 norm under some assumptions).
- Geometric explanation.

Source: Tibshirani et al.

Constraint and Penalization

Constrained Optimization

- Choose a constant C.
- Compute β as

$$\underset{\beta \in \mathbb{R}^d, \|\beta\|_p \leq C}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \bar{\ell}(Y_i, h(\underline{x}_i{}^\top \beta))$$

Lagrangian Reformulation

- Choose λ and compute β as

$$\underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \bar{\ell}(Y_i, h(\underline{x}_i{}^\top \beta)) + \lambda \|\beta\|_p^{p'}$$

  with $p' = p$ except if $p = 0$ where $p' = 1$.
- Easier calibration... but no explicit model $\mathcal{S}$.

- **Rk:** $\|\beta\|_p$ is not scaling invariant if $p \neq 0$...
- Initial rescaling issue.

Penalization

Penalized Linear Model

- Minimization of

$$\underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \bar{\ell}(Y_i, h(\underline{x}_i^\top \beta)) + \text{pen}(\beta)$$

  where $\text{pen}(\beta)$ is a (sparsity promoting) penalty
- Variable selection if β is sparse.

Classical Penalties

- AIC: $\text{pen}(\beta) = \lambda \|\beta\|_0$ (non-convex / sparsity)
- Ridge: $\text{pen}(\beta) = \lambda \|\beta\|_2^2$ (convex / no sparsity)
- Lasso: $\text{pen}(\beta) = \lambda \|\beta\|_1$ (convex / sparsity)
- Elastic net: $\text{pen}(\beta) = \lambda_1 \|\beta\|_1 + \lambda_2 \|\beta\|_2^2$ (convex / sparsity)

- Easy optimization if pen (and the loss) is convex...
- **Need to specify λ to define a ML method!**

Penalization and Cross-Validation

Practical Selection Methodology

- Choose a penalty family pen_λ.
- Compute a CV risk for the penalty pen_λ for all $\lambda \in \Lambda$.
- Determine $\widehat{\lambda}$ the λ minimizing the CV risk.
- Compute the final model with the penalty $\text{pen}_{\widehat{\lambda}}$.

- CV allows to select a ML method, penalized estimation with a penalty $\text{pen}_{\widehat{\lambda}}$, not a single predictor hence the need of a final reestimation.

Why not using CV on a grid?

- Grid size scales exponentially with the dimension!
- **If the penalized minimization is easy**, much cheaper to compute the CV risk for all $\lambda \in \Lambda \ldots$
- CV performs best when the set of candidates is not too big (or is structured...)

References

T. Hastie, R. Tibshirani, and J. Friedman.
The Elements of Statistical Learning.
Springer Series in Statistics, 2009

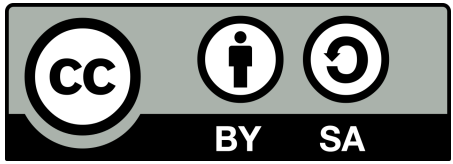
M. Mohri, A. Rostamizadeh, and A. Talwalkar.
Foundations of Machine Learning.
MIT Press, 2012

A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (2nd ed.)
O'Reilly, 2019

Ch. Giraud.
Introduction to High-Dimensional Statistics.
CRC Press, 2014

S. Bubeck.
Convex Optimization: Algorithms and Complexity.
Now Publisher, 2015

Licence and Contributors



Contributors

- Main contributor: E. Le Pennec
- Contributors: S. Boucheron, A. Dieuleveut, A.K. Fermin, S. Gadat, S. Gaiffas, A. Guilloux, Ch. Keribin, E. Matzner, M. Sangnier, E. Scornet.

Outline

Outline

Supervised Learning

Supervised Learning Framework

- Input measurement $\underline{X} \in \mathcal{X}$
- Output measurement $Y \in \mathcal{Y}$.
- $(\underline{X}, Y) \sim \mathbb{P}$ with $\mathbb{P}$ unknown.
- **Training data** : $\mathcal{D}_n = \{(\underline{X}_1, Y_1), \ldots, (\underline{X}_n, Y_n)\}$ (i.i.d. $\sim \mathbb{P}$)

- Often
  - $\underline{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (classification)
  - or $\underline{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A **predictor** is a function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y} \text{ meas.}\}$

Goal

- Construct a **good** predictor $\widehat{f}$ from the training data.

- Need to specify the meaning of good.
- Classification and regression are almost the **same** problem!

Loss and Probabilistic Framework

Loss function for a generic predictor

- **Loss function**: $\ell(Y, f(\underline{X}))$ measures the goodness of the prediction of Y by $f(\underline{X})$
- Examples:
  - 0/1 loss: $\ell(Y, f(\underline{X})) = \mathbf{1}_{Y \neq f(\underline{X})}$
  - Quadratic loss: $\ell(Y, f(\underline{X})) = |Y - f(\underline{X})|^2$

Risk function

- Risk measured as the average loss for a new couple:

$$\mathcal{R}(f) = \mathbb{E}_{(X,Y)\sim\mathbb{P}}[\ell(Y, f(\underline{X}))]$$

- Examples:
  - 0/1 loss: $\mathbb{E}[\ell(Y, f(\underline{X}))] = \mathbb{P}(Y \neq f(\underline{X}))$
  - Quadratic loss: $\mathbb{E}[\ell(Y, f(\underline{X}))] = \mathbb{E}\left[|Y - f(\underline{X})|^2\right]$

- **Beware:** As $\widehat{f}$ depends on $\mathcal{D}_n$, $\mathcal{R}(\widehat{f})$ is a random variable!

Best Solution

- The best solution $f^\star$ (which is independent of $\mathcal{D}_n$) is

$$f^\star = \arg\min_{f \in \mathcal{F}} \mathcal{R}(f) = \arg\min_{f \in \mathcal{F}} \mathbb{E}[\ell(Y, f(\underline{X}))] = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{\underline{X}}\Big[\mathbb{E}_{Y|\underline{X}}[\ell(Y, f(\underline{X}))]\Big]$$

Bayes Predictor (explicit solution)

- In binary classification with $0-1$ loss:

$$f^\star(\underline{X}) = \begin{cases} +1 & \text{if } \mathbb{P}(Y=+1|\underline{X}) \geq \mathbb{P}(Y=-1|\underline{X}) \\ & \Leftrightarrow \mathbb{P}(Y=+1|\underline{X}) \geq 1/2 \\ -1 & \text{otherwise} \end{cases}$$

- In regression with the quadratic loss

$$f^\star(\underline{X}) = \mathbb{E}[Y|\underline{X}]$$

Issue: Solution requires to **know** $\mathbb{E}[Y|\underline{X}]$ for all values of $\underline{X}$!

Goal

Machine Learning

- Learn a rule to construct a **predictor** $\widehat{f} \in \mathcal{F}$ from the training data $\mathcal{D}_n$ s.t. **the risk** $\mathcal{R}(\widehat{f})$ is **small on average** or with high probability with respect to $\mathcal{D}_n$.

- In practice, the rule should be an algorithm!

Canonical example: Empirical Risk Minimizer

- One restricts f to a subset of functions $\mathcal{S} = \{f_\theta, \theta \in \Theta\}$
- One replaces the minimization of the average loss by the minimization of the empirical loss

$$\widehat{f} = f_{\widehat{\theta}} = \underset{f_\theta, \theta \in \Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f_\theta(\underline{X}_i))$$

- Examples:
  - Linear regression
  - Linear classification with

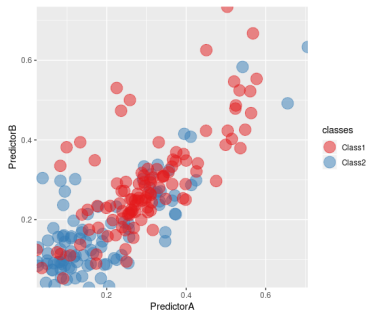
$$\mathcal{S} = \{\underline{x} \mapsto \operatorname{sign}\{\underline{x}^\top \beta + \beta^{(0)}\} \, / \beta \in \mathbb{R}^d, \beta^{(0)} \in \mathbb{R}\}$$

Example: TwoClass Dataset

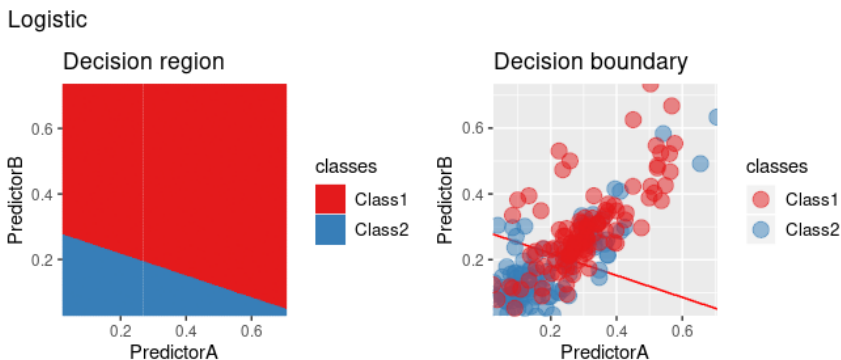
Synthetic Dataset

- Two features/covariates.
- Two classes.

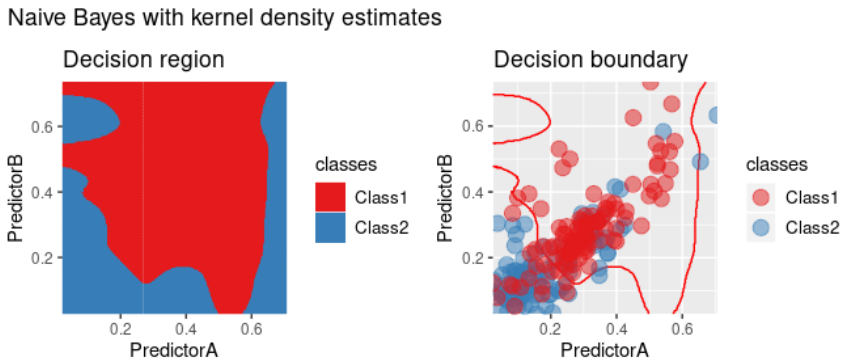
- Dataset from *Applied Predictive Modeling*, M. Kuhn and K. Johnson, Springer
- Numerical experiments with `R` and the {`caret`} package.



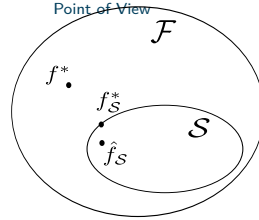
Example: Linear Discrimination



Example: More Complex Model



Bias-Variance Dilemma



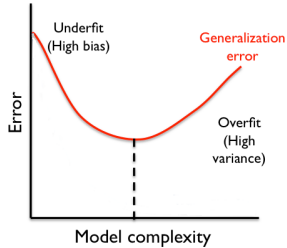
- General setting:
  - $\mathcal{F} = \{\text{measurable functions } \mathcal{X} \to \mathcal{Y}\}$
  - Best solution: $f^\star = \text{argmin}_{f \in \mathcal{F}} \mathcal{R}(f)$
  - Class $\mathcal{S} \subset \mathcal{F}$ of functions
  - Ideal target in $\mathcal{S}$: $f^\star_{\mathcal{S}} = \text{argmin}_{f \in \mathcal{S}} \mathcal{R}(f)$
  - Estimate in $\mathcal{S}$: $\widehat{f}_{\mathcal{S}}$ obtained with some procedure

Approximation error and estimation error (Bias/Variance)

$$\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star) = \underbrace{\mathcal{R}(f^\star_{\mathcal{S}}) - \mathcal{R}(f^\star)}_{\text{Approximation error}} + \underbrace{\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star_{\mathcal{S}})}_{\text{Estimation error}}$$

- Approx. error can be large if the model $\mathcal{S}$ is not suitable.
- Estimation error can be large if the model is complex.

Under-fitting / Over-fitting Issue



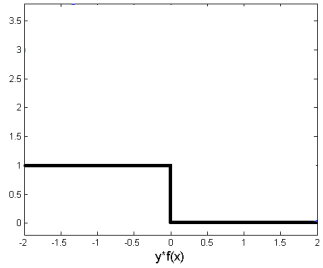
- Different behavior for different model complexity
- **Low complexity model** are easily learned but the approximation error (**bias**) may be large (**Under-fit**).
- **High complexity model** may contain a good ideal target but the estimation error (**variance**) can be large (**Over-fit**)

Bias-variance trade-off $\iff$ avoid **overfitting** and **underfitting**

- **Rk:** Better to think in term of method (including feature engineering and specific algorithm) rather than only of model.

Source: Unknown

Binary Classification Loss Issue

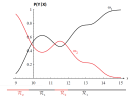


Empirical Risk Minimizer

$$\widehat{f} = \underset{f \in \mathcal{S}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell^{0/1}(Y_i, f(\underline{X}_i))$$

- Classification loss: $\ell^{0/1}(y, f(\underline{x})) = \mathbf{1}_{y \neq f(\underline{x})}$
- Not convex and not smooth!

Probabilistic Point of View
Ideal Solution and Estimation



- The best solution $f^\star$ (which is independent of $\mathcal{D}_n$) is

$$f^\star = \arg\min_{f \in \mathcal{F}} \mathcal{R}(f) = \arg\min_{f \in \mathcal{F}} \mathbb{E}[\ell(Y, f(\underline{X}))] = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{\underline{X}}\Big[\mathbb{E}_{Y|\underline{X}}[\ell(Y, f(\underline{x}))]\Big]$$

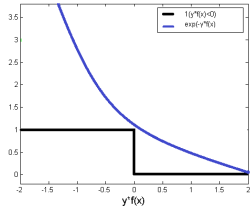
Bayes Predictor (explicit solution)

In binary classification with $0-1$ loss:

$$f^\star(\underline{X}) = \begin{cases} +1 & \text{if } \mathbb{P}(Y=+1|\underline{X}) \geq \mathbb{P}(Y=-1|\underline{X}) \\ -1 & \text{otherwise} \end{cases}$$

- **Issue:** Solution requires to **know** $\mathbb{E}[Y|\underline{X}]$ for all values of $\underline{X}$!
- **Solution:** Replace it by an estimate.

Optimization Point of View
Loss Convexification



Minimizer of the risk

$$\widehat{f} = \underset{f \in \mathcal{S}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell^{0/1}(Y_i, f(\underline{X}_i))$$

- **Issue:** Classification loss is not convex or smooth.
- **Solution:** Replace it by a convex majorant.

Probabilistic and Optimization Framework

How to find a good function f with a *small* risk

$$\mathcal{R}(f) = \mathbb{E}[\ell(Y, f(\underline{X}))] \quad ?$$

Canonical approach: $\widehat{f}_{\mathcal{S}} = \text{argmin}_{f \in \mathcal{S}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f(\underline{X}_i))$

Problems

- How to choose $\mathcal{S}$?
- How to compute the minimization?

A Probabilistic Point of View

Solution: For $\underline{X}$, estimate $Y|\underline{X}$ plug this estimate in the Bayes classifier:
(Generalized) Linear Models, Kernel methods, k-nn, Naive Bayes, Tree, Bagging...

An Optimization Point of View

Solution: If necessary replace the loss ℓ by an upper bound $\bar{\ell}$ and minimize the empirical loss: **SVR, SVM, Neural Network,Tree, Boosting...**

Outline

Probabilistic and Optimization Framework

How to find a good function f with a *small* risk

$$\mathcal{R}(f) = \mathbb{E}[\ell(Y, f(\underline{X}))] \quad ?$$

Canonical approach: $\widehat{f}_{\mathcal{S}} = \text{argmin}_{f \in \mathcal{S}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f(\underline{X}_i))$

Problems

- How to choose $\mathcal{S}$?
- How to compute the minimization?

A Probabilistic Point of View

Solution: For $\underline{X}$, estimate $Y|\underline{X}$ plug this estimate in the Bayes classifier:
(Generalized) Linear Models, Kernel methods, k-nn, Naive Bayes, Tree, Bagging...

An Optimization Point of View

Solution: If necessary replace the loss ℓ by an upper bound $\bar{\ell}$ and minimize the empirical loss: **SVR, SVM, Neural Network,Tree, Boosting...**

Three Classical Methods in a Nutshell

Penalized Logistic Regression

- Let $f_\theta(\underline{X}) = \underline{X}^\top \beta + \beta^{(0)}$ with $\theta = (\beta, \beta^{(0)})$.
- Find $\hat{\theta} = \arg\min \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-Y_i f_\theta(\underline{X}_i)}\right) + \lambda \|\beta\|_1$
- Classify using $\text{sign}(f_{\hat{\theta}})$

Deep Learning

- Let $f_\theta(\underline{X})$ with f a feed forward neural network outputing two values with a softmax layer as a last layer.
- Optimize by gradient descent the cross-entropy $-\frac{1}{n} \sum_{i=1}^{n} \log\left(f_\theta(\underline{X}_i)^{(Y_i)}\right)$
- Classify using $\text{sign}(f_{\hat{\theta}})$

Three Classical Methods in a Nutshell

Support Vector Machine

- Let $f_\theta(\underline{X}) = \underline{X}^\top \beta + \beta^{(0)}$ with $\theta = (\beta, \beta^{(0)})$.
- Find $\hat{\theta} = \arg\min \frac{1}{n} \sum_{i=1}^{n} \max(1 - Y_i f_\theta(\underline{X}_i), 0) + \lambda \|\beta\|_2^2$
- Classify using $\text{sign}(f_{\hat{\theta}})$

- Those three methods rely on a similar heuristic: the optimization point of view!

Empirical Risk Minimization

- The best solution $f^\star$ is the one minimizing

$$f^\star = \arg\min R(f) = \arg\min \mathbb{E}[\ell(Y, f(\underline{X}))]$$

Empirical Risk Minimization

- One restricts f to a subset of functions $\mathcal{S} = \{f_\theta, \theta \in \Theta\}$
- One replaces the minimization of the average loss by the minimization of the average empirical loss

$$\widehat{f} = f_{\widehat{\theta}} = \underset{f_\theta, \theta \in \Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f_\theta(\underline{X}_i))$$

- Intractable for the $\ell^{0/1}$ loss!

Convexification Strategy

Risk Convexification

- Replace the loss $\ell(Y, f_\theta(\underline{X}))$ by a convex upperbound $\bar{\ell}(Y, f_\theta(\underline{X}))$ (surrogate loss).
- Minimize the average of the surrogate empirical loss

$$\tilde{f} = f_{\widehat{\theta}} = \underset{f_\theta, \theta \in \Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \bar{\ell}(Y_i, f_\theta(\underline{X}_i))$$

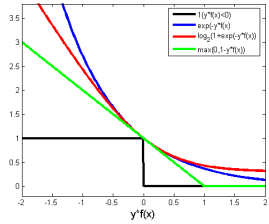
- Use $\widehat{f} = \operatorname{sign}(\tilde{f})$

- Much easier optimization.

Instantiation

- Logistic (Revisited)
- Support Vector Machine
- (Deep) Neural Network
- Boosting

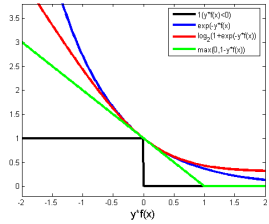
Classification Loss and Convexification



Convexification

- Replace the loss $\ell^{0/1}(Y, f(\underline{X}))$ by
$$\bar{\ell}(Y, f(\underline{X})) = l(Yf(\underline{X}))$$
with l a convex function.
- **Further mild assumption:** l is decreasing, differentiable at 0 and $l'(0) < 0$.

Classification Loss and Convexification



Classical convexification

- Logistic loss: $\bar{\ell}(Y, f(\underline{X})) = \log_2(1 + e^{-Yf(\underline{X})})$ (Logistic / NN)
- Hinge loss: $\bar{\ell}(Y, f(\underline{X})) = (1 - Yf(\underline{X}))_+$ (SVM)
- Exponential loss: $\bar{\ell}(Y, f(\underline{X})) = e^{-Yf(\underline{X})}$ (Boosting...)

Properties

The Target is the Bayes Classifier

- The minimizer of
$$\mathbb{E}\Big[\bar{\ell}(Y, f(\underline{X}))\Big] = \mathbb{E}[I(Yf(\underline{X}))]$$
is the Bayes classifier $f^\star = \text{sign}(2\eta(\underline{X}) - 1)$

Control of the Excess Risk

- It exists a convex function Ψ such that
$$\Psi\left(\mathbb{E}\Big[\ell^{0/1}(Y, \text{sign}(f(\underline{X}))\Big] - \mathbb{E}\Big[\ell^{0/1}(Y, f^\star(\underline{X})\Big]\right)$$
$$\leq \mathbb{E}\Big[\bar{\ell}(Y, f(\underline{X})\Big] - \mathbb{E}\Big[\bar{\ell}(Y, f^\star(\underline{X}))\Big]$$

- Theoretical guarantee!

Logistic Revisited

- Ideal solution:

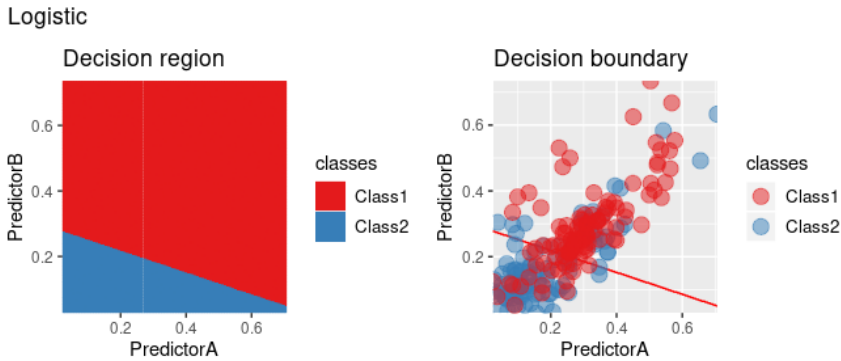
$$\widehat{f} = \underset{f \in \mathcal{S}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell^{0/1}(Y_i, f(\underline{X}_i))$$

Logistic regression

- Use $f(\underline{X}) = \underline{X}^\top \beta + \beta^{(0)}$.
- Use the logistic loss $\bar{\ell}(y, f) = \log_2(1 + e^{-yf})$, i.e. the negative log-likelihood.

- Different vision than the statistician but same algorithm!

Logistic Revisited



Outline

Support Vector Machine

$$f_\theta(\underline{X}) = \underline{X}^\top \beta + \beta^{(0)} \quad \text{with} \quad \theta = (\beta, \beta^{(0)})$$

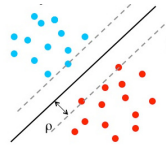
$$\hat{\theta} = \arg\min \frac{1}{n} \sum_{i=1}^{n} \max\left(1 - Y_i f_\theta(\underline{X}_i), 0\right) + \lambda \|\beta\|_2^2$$

Support Vector Machine

- Convexification of the 0/1-loss with the hinge loss:
$$\mathbf{1}_{Y_i f_\theta(\underline{X}_i) < 0} \leq \max\left(1 - Y_i f_\theta(\underline{X}_i), 0\right)$$
- Penalization by the quadratic norm (Ridge/Tikhonov).
- Solution can be approximated by gradient descent algorithms.

- **Revisit** of the original point of view.
- Original point of view leads to a different optimization algorithm and to some extensions.

Ideal Separable Case



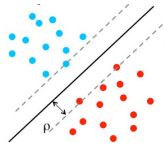
- Linear classifier: $\text{sign}(\underline{X}^\top \beta + \beta^{(0)})$
- Separable case: $\exists(\beta, \beta^{(0)}), \forall i,\ Y_i(\underline{X_i}^\top \beta + \beta^{(0)}) > 0$

How to choose $(\beta, \beta^{(0)})$ so that the separation is maximal?

- Strict separation: $\exists(\beta, \beta^{(0)}), \forall i,\ Y_i(\underline{X_i}^\top \beta + \beta^{(0)}) \geq 1$
- Distance between $\underline{X}^\top \beta + \beta^{(0)} = 1$ and $\underline{X}^\top \beta + \beta^{(0)} = -1$:
$$\frac{2}{\|\beta\|}$$
- Maximizing this distance is equivalent to minimizing $\frac{1}{2}\|\beta\|^2$.

Source: M. Mohri et al.

Ideal Separable Case



Separable SVM

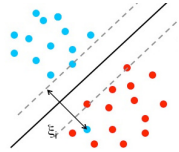
- Constrained optimization formulation:

$$\min \frac{1}{2}\|\beta\|^2 \quad \text{with} \quad \forall i, Y_i(\underline{X}_i{}^\top \beta + \beta^{(0)}) \geq 1$$

- Quadratic Programming setting.
- Efficient solver available...

Source: M. Mohri et al.

Non Separable Case



- What about the non separable case?

SVM relaxation

- Relax the assumptions

$$\forall i, Y_i(\underline{X}_i{}^\top \beta + \beta^{(0)}) \geq 1 \quad \text{to} \quad \forall i, Y_i(\underline{X}_i{}^\top \beta + \beta^{(0)}) \geq 1 - s_i$$

  with the **slack variables** $s_i \geq 0$

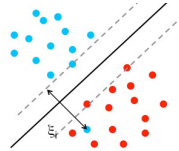
- Keep those slack variables as small as possible by minimizing

$$\frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} s_i$$

  where $C > 0$ is the **goodness-of-fit strength**

Source: M. Mohri et al.

Non Separable Case



SVM

- Constrained optimization formulation:

$$\min \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} s_i \quad \text{with} \quad \begin{cases} \forall i, Y_i(\underline{X}_i{}^\top \beta + \beta^{(0)}) \geq 1 - s_i \\ \forall i, s_i \geq 0 \end{cases}$$

- **Hinge Loss** reformulation:

$$\min \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} \underbrace{\max(0, 1 - Y_i(\underline{X}_i{}^\top \beta + \beta^{(0)}))}_{\text{Hinge Loss}}$$

- Constrained convex optimization algorithms vs gradient descent algorithms.

Source: M. Mohri et al.

SVM as a Penalized Convex Relaxation

- Convex relaxation:

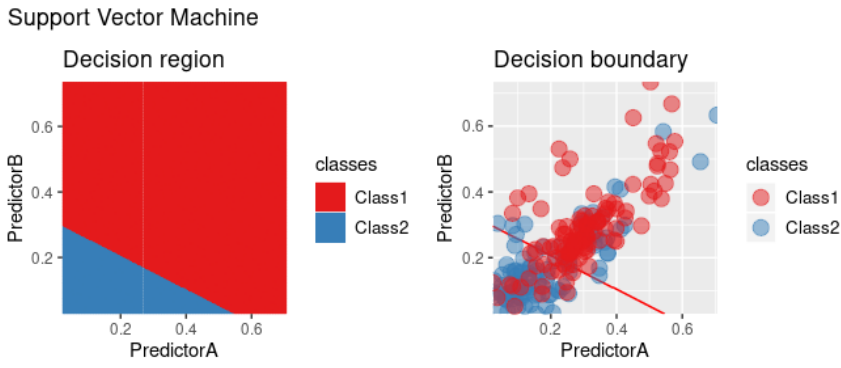
$$\begin{aligned}&\operatorname{argmin} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} \max(1 - Y_i(\underline{X}_i{}^\top\beta + \beta^{(0)}), 0)\\ &= \operatorname{argmin} \frac{1}{n}\sum_{i=1}^{n} \max(1 - Y_i(\underline{X}_i{}^\top\beta + \beta^{(0)}), 0) + \frac{1}{Cn}\frac{1}{2}\|\beta\|^2\end{aligned}$$

- **Prop:** $\ell^{0/1}(Y_i, \operatorname{sign}(\underline{X}_i{}^\top\beta + \beta^{(0)})) \leq \max(1 - Y_i(\underline{X}_i{}^\top\beta + \beta^{(0)}), 0)$

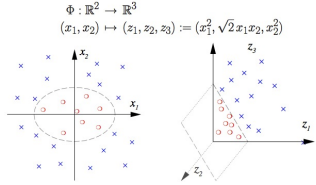
Penalized convex relaxation (Tikhonov!)

$$\begin{aligned}&\frac{1}{n}\sum_{i=1}^{n} \ell^{0/1}(Y_i, \operatorname{sign}(\underline{X}_i{}^\top\beta + \beta^{(0)})) + \frac{1}{Cn}\frac{1}{2}\|\beta\|^2\\ &\leq \frac{1}{n}\sum_{i=1}^{n} \max(1 - Y_i(\underline{X}_i{}^\top\beta + \beta^{(0)}), 0) + \frac{1}{Cn}\frac{1}{2}\|\beta\|^2\end{aligned}$$

SVM


Support Vector Machine
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

The Kernel Trick



- Non linear separation: just replace $\underline{X}$ by a non linear $\Phi(\underline{X})\ldots$
- Knowing $\phi(\underline{X}_i)^\top \phi(\underline{X}_j)$ is sufficient to compute the SVM solution.

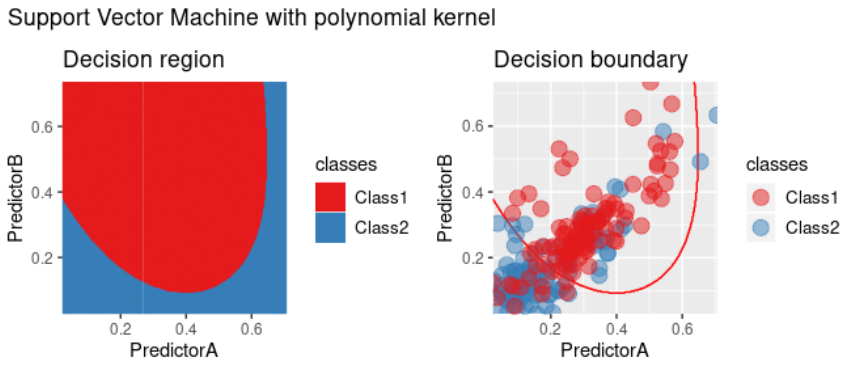
Kernel trick

- **Computing $k(\underline{X}, \underline{X}') = \phi(\underline{X})^\top \phi(\underline{X}')$ may be easier than computing $\phi(\underline{X})$, $\phi(\underline{X}')$ and then the scalar product!**
- ϕ can be specified through its definite positive kernel k.

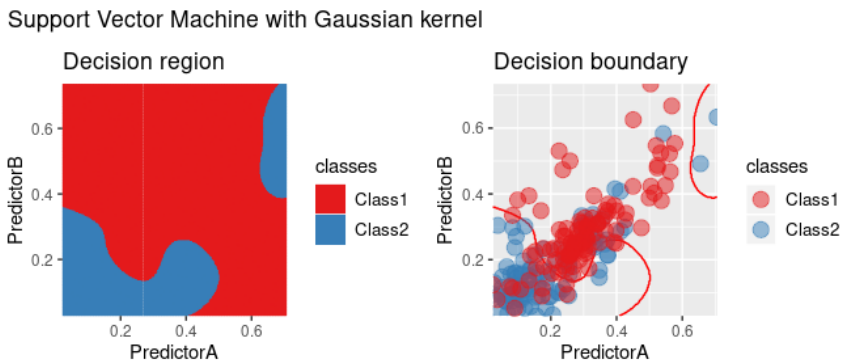
- Examples: Polynomial kernel $k(\underline{X}, \underline{X}') = (1 + \underline{X}^\top \underline{X}')^d$, Gaussian kernel $k(\underline{X}, \underline{X}') = e^{-\|\underline{X}-\underline{X}'\|^2/2}$,...
- RKHS setting!
- Can be used in (logistic) regression and more...

Source: Unknown

SVM


Support Vector Machine with polynomial kernel
Decision region
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2
Decision boundary
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2

SVM


Support Vector Machine with Gaussian kernel
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

SVM and Lagrangian

SVM

- Constrained optimization formulation:

$$\min \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} s_i \quad \text{with} \quad \begin{cases} \forall i, Y_i(\underline{X}_i^\top \beta + \beta^{(0)}) \geq 1 - s_i \\ \forall i, s_i \geq 0 \end{cases}$$

SVM Lagrangian

- Lagrangian:

$$\mathcal{L}(\beta, \beta^{(0)}, s, \alpha, \mu) = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} s_i + \sum_i \alpha_i(1 - s_i - Y_i(\underline{X}_i^\top \beta + \beta^{(0)})) - \sum_i \mu_i s_i$$

SVM and KKT

KKT Optimality Conditions

- Stationarity:

$$\nabla_\beta \mathcal{L}(\beta, \beta^{(0)}, s, \alpha, \mu) = \beta - \sum_i \alpha_i Y_i \underline{X}_i = 0$$

$$\nabla_{\beta^{(0)}} \mathcal{L}(\beta, \beta^{(0)}, s, \alpha, \mu) = -\sum_i \alpha_i = 0$$

$$\nabla_{s_i} \mathcal{L}(\beta, \beta^{(0)}, s, \alpha, \mu) = C - \alpha_i - \mu_i = 0$$

- Primal and dual admissibility:

$$(1 - s_i - Y_i(\underline{X}_i{}^\top \beta + \beta^{(0)})) \leq 0, \quad s_i \geq 0, \quad \alpha_i \geq 0, \text{ and } \mu_i \geq 0$$

- Complementary slackness:

$$\alpha_i(1 - s_i - Y_i(\underline{X}_i{}^\top \beta + \beta^{(0)})) = 0 \quad \text{and} \quad \mu_i s_i = 0$$

Consequence

- $\beta^\star = \sum_i \alpha_i Y_i \underline{X}_i$ and $0 \leq \alpha_i \leq C$.
- If $\alpha_i \neq 0$, $\underline{X}_i$ is called a **support vector** and either
  - $s_i = 0$ and $Y_i(\underline{X}_i{}^\top \beta^\star + \beta^{(0)\star}) = 1$ (margin hyperplane),
  - or $\alpha_i = C$ (outliers).
- $\beta^{(0)\star} = Y_i - \underline{X}_i{}^\top \beta^\star$ for any support vector with $0 < \alpha_i < C$.

SVM Dual

SVM Lagrangian Dual

- Lagrangian Dual:

$$Q(\alpha, \mu) = \min_{\beta, \beta^{(0)}, s} \mathcal{L}(\beta, \beta^{(0)}, s, \alpha, \mu)$$

- Prop:
  - if $\sum_i \alpha_i Y_i \neq 0$ or $\exists i, \alpha_i + \mu_i \neq C$,

$$Q(\alpha, \mu) = -\infty$$

  - if $\sum_i \alpha_i Y_i = 0$ and $\forall i, \alpha_i + \mu_i = C$,

$$Q(\alpha, \mu) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j Y_i Y_j \underline{X}_i^\top \underline{X}_j$$

SVM Dual problem

- Dual problem is a Quadratic Programming problem:

$$\max_{\alpha \geq 0, \mu \geq 0} Q(\alpha, \mu) \Leftrightarrow \max_{0 \leq \alpha \leq C} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j Y_i Y_j \underline{X}_i^\top \underline{X}_j$$

- Involves the $\underline{X}_i$ only through their scalar products.

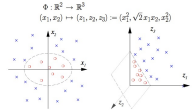
Mercer Theorem

Mercer Representation Theorem

- For any loss $\bar{\ell}$ and any increasing function Φ, the minimizer in β of
$$\sum_{i=1}^{n} \bar{\ell}(Y_i, \underline{X}_i^\top \beta + \beta^{(0)}) + \Phi(\|\beta\|_2)$$
is a linear combination of the input points $\beta^\star = \sum_{i=1}^{n} \alpha_i' \underline{X}_i$.
- Minimization problem in α':
$$\sum_{i=1}^{n} \bar{\ell}(Y_i, \sum_j \alpha_j' \underline{X}_i^\top \underline{X}_j + \beta^{(0)}) + \Phi(\|\beta\|_2)$$
involving only the scalar product of the data.

- Optimal predictor requires only to compute scalar products.
$$\hat{f}^\star(\underline{X}) = \underline{X}^\top \beta^\star + \beta^{(0),*} = \sum_i \alpha_i' \underline{X}_i^\top \underline{X}$$
- Transform a problem in dimension $\dim(\mathcal{X})$ in a problem in dimension n.
- Direct minimization in β can be more efficient...

Feature Map



Feature Engineering

- Art of creating **new features** from the existing one $\underline{X}$.
- Example: add monomials $(\underline{X}^{(j)})^2$, $\underline{X}^{(j)}\underline{X}^{(j')}\dots$
- Adding feature increases the dimension.

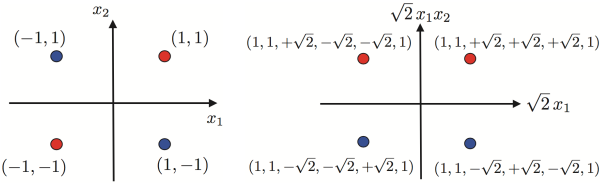
Feature Map

- Application $\phi : \mathcal{X} \to \mathbb{H}$ with $\mathbb{H}$ an Hilbert space.
- Linear decision boundary in $\mathbb{H}$: $\phi(\underline{X})^\top \beta + \beta^{(0)} = 0$ is **not an hyperplane anymore** in $\mathcal{X}$.

- **Heuristic:** Increasing dimension allows to make data almost linearly separable.

Source: Unknown

Polynomial Mapping



Polynomial Mapping of order 2

- $\phi : \mathbb{R}^2 \to \mathbb{R}^6$

$$\phi(\underline{X}) = \left((\underline{X}^{(1)})^2, (\underline{X}^{(2)})^2, \sqrt{2}\underline{X}^{(1)}\underline{X}^{(2)}, \sqrt{2}\underline{X}^{(1)}, \sqrt{2}\underline{X}^{(2)}, 1\right)$$

- Allow to solve the XOR classification problem with the *hyperplane* $\underline{X}^{(1)}\underline{X}^{(2)} = 0$.

Polynomial Mapping and Scalar Product

- **Prop:**

$$\phi(\underline{X})^\top \phi(\underline{X}') = (1 + \underline{X}^\top \underline{X}')^2$$

Source: M. Mohri et al.

SVM Primal and Dual

Primal, Lagrandian and Dual

- Primal:

$$\min \|\beta\|^2 + C \sum_{i=1}^{n} s_i \quad \text{with} \quad \begin{cases} \forall i, Y_i(\phi(\underline{X}_i)^\top \beta + \beta^{(0)}) \geq 1 - s_i \\ \forall i, s_i \geq 0 \end{cases}$$

- Lagrangian:

$$\mathcal{L}(\beta, \beta^{(0)}, s, \alpha, \mu) = \frac{1}{2}\|\beta\|^2 + C \sum_{i=1}^{n} s_i + \sum_i \alpha_i (1 - s_i - Y_i(\phi(\underline{X}_i)^\top \beta + \beta^{(0)})) - \sum_i \mu_i s_i$$

- Dual:

$$\max_{\alpha \geq 0, \mu \geq 0} Q(\alpha, \mu) \Leftrightarrow \max_{0 \leq \alpha \leq C} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j Y_i Y_j \phi(\underline{X}_i)^\top \phi(\underline{X}_j)$$

- Optimal $\phi(\underline{X})^\top \beta^\star + \beta^{(0),*} = \sum_i \alpha_i Y_i \phi(\underline{X})^\top \phi(\underline{X}_i)$

- Only need to know to compute $\phi(\underline{X})^\top \phi(\underline{X}')$ to obtain the solution.

From Map to Kernel

- Many algorithms (e.g. SVM) require only to be able to compute the scalar product $\phi(\underline{X})^\top \phi(\underline{X}')$.

Kernel

- Any application
$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$
is called a **kernel** over $\mathcal{X}$.

Kernel Trick

- Computing directly the **kernel** $k(\underline{X}, \underline{X}') = \phi(\underline{X})^\top \phi(\underline{X}')$ may be easier than computing $\phi(\underline{X})$, $\phi(\underline{X}')$ and then the scalar product.

- Here k is defined from ϕ.
- Under some assumption on k, ϕ can be implicitly *defined* from k!

Positive Definite Symmetric Kernels

- A kernel k is PDS if and only if
  - k is symmetric, i.e.
$$k(\underline{X}, \underline{X}') = k(\underline{X}', \underline{X})$$
  - for any $N \in \mathbb{N}$ and any $(\underline{X}_1, \ldots, \underline{X}_N) \in \mathcal{X}^N$,
$$\boldsymbol{K} = [k(\underline{X}_i, \underline{X}_j)]_{1 \leq i,j \leq N}$$
    is positive semi-definite, i.e. $\forall u \in \mathbb{R}^N$
$$u^\top \boldsymbol{K} u = \sum_{1 \leq i,j \leq N} u^{(i)} u^{(j)} k(\underline{X}_i, \underline{X}_j) \geq 0$$
    or equivalently all the eigenvalues of $\boldsymbol{K}$ are non-negative.

- The matrix $\boldsymbol{K}$ is called the **Gram matrix** associated to $(\underline{X}_1, \ldots, \underline{X}_N)$.

Reproducing Kernel Hilbert Space

Moore-Aronsajn Theorem

- For any PDS kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, it exists a Hilbert space $\mathbb{H} \subset \mathbb{R}^{\mathcal{X}}$ with a scalar product $\langle \cdot, \cdot \rangle_{\mathbb{H}}$ such that
  - it exists a mapping $\phi : \mathcal{X} \to \mathbb{H}$ satisfying
  $$k(\underline{X}, \underline{X}') = \left\langle \phi(\underline{X}), \phi(\underline{X}') \right\rangle_{\mathbb{H}}$$
  - the **reproducing property** holds, i.e. for any $h \in \mathbb{H}$ and any $\underline{X} \in \mathcal{X}$
  $$h(\underline{X}) = \langle h, k(\underline{X}, \cdot) \rangle_{\mathbb{H}} .$$

- By def., $\mathbb{H}$ is a **reproducing kernel Hilbert space** (RKHS).
- $\mathbb{H}$ is called the **feature space** associated to k and ϕ the **feature mapping**.
- No unicity in general.
- **Rk:** if $k(\underline{X}, \underline{X}') = \phi'(\underline{X})^\top \phi'(\underline{X}')$ with $\phi' : \mathcal{X} \to \mathbb{R}^p$ then
  - $\mathbb{H}$ can be chosen as $\{\underline{X} \mapsto \phi'(\underline{X})^\top \beta, \beta \in \mathbb{R}^p\}$ and $\|\underline{X} \mapsto \phi'(\underline{X})^\top \beta\|_{\mathbb{H}}^2 = \|\beta\|_2^2$.
  - $\phi(\underline{X}') : \underline{X} \mapsto \phi'(\underline{X})^\top \phi'(\underline{X}')$.

Kernel Construction Machinery

Separable Kernel

- For any function $\Psi : \mathcal{X} \to \mathbb{R}$, $k(\underline{X}, \underline{X}') = \Psi(\underline{X})\Psi(\underline{X}')$ is PDS.

Kernel Stability

- For any PDS kernels k_1 and k_2, $k_1 + k_2$ and $k_1 k_2$ are PDS kernels.
- For any sequence of PDS kernels k_n converging pointwise to a kernel k, k is a PDS kernel.
- For any PDS kernel k such that $|k| \leq r$ and any power series $\sum_n a_n z^n$ with $a_n \geq 0$ and a convergence radius larger than r, $\sum_n a_n k^n$ is a PDS kernel.
- For any PDS kernel k, the renormalized kernel $k'(\underline{X}, \underline{X}') = \dfrac{k(\underline{X}, \underline{X}')}{\sqrt{k(\underline{X}, \underline{X})k(\underline{X}', \underline{X}')}}$ is a PDS kernel.

- Cauchy-Schwartz for k PDS: $k(\underline{X}, \underline{X}')^2 \leq k(\underline{X}, \underline{X})k(\underline{X}', \underline{X}')$

Classical Kernels

PDS Kernels

- Vanilla kernel:
$$k(\underline{X}, \underline{X}') = \underline{X}^\top \underline{X}'$$
- Polynomial kernel:
$$k(\underline{X}, \underline{X}') = (1 + \underline{X}^\top \underline{X}')^k$$
- Gaussian RBF kernel:
$$k(\underline{X}, \underline{X}') = \exp\left(-\gamma \|\underline{X} - \underline{X}'\|^2\right)$$
- Tanh kernel:
$$k(\underline{X}, \underline{X}') = \tanh(a\underline{X}^\top \underline{X}' + b)$$

- Most classical is the Gaussian RBF kernel...
- Lots of freedom to construct kernel for non classical data.

Representer Theorem

Representer Theorem

- Let k be a PDS kernel and $\mathbb{H}$ its corresponding RKHS,
  for any increasing function Φ and any function $L : \mathbb{R}^n \to \mathbb{R}$, the optimization problem
  $$\underset{h \in \mathbb{H}}{\operatorname{argmin}}\, L(h(\underline{X}_1), \ldots, h(\underline{X}_n)) + \Phi(\|h\|)$$
  admits only solutions of the form
  $$\sum_{i=1}^{n} \alpha_i' k(\underline{X}_i, \cdot).$$

- Examples:
  - (kernelized) SVM
  - (kernelized) Penalized Logistic Regression (Ridge)
  - (kernelized) Penalized Regression (Ridge)

Kernelized SVM

Primal

- Constrained Optimization:

$$\min_{f\in\mathbb{H},\beta^{(0)},s} \|f\|_{\mathbb{H}}^2 + C\sum_{i=1}^{n} s_i \quad \text{with} \quad \begin{cases} \forall i, Y_i(f(\underline{X}_i) + \beta^{(0)}) \geq 1 - s_i \\ \forall i, s_i \geq 0 \end{cases}$$

- Hinge loss:

$$\min_{f\in\mathbb{H},\beta^{(0)}} \|f\|_{\mathbb{H}}^2 + C\sum_{i=1}^{n} \max(0, 1 - Y_i(f(\underline{X}_i) + \beta^{(0)}))$$

- Representer:

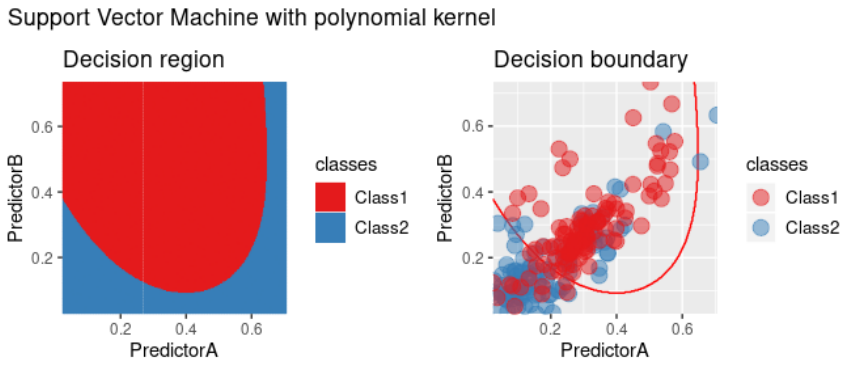
$$\min_{\alpha',\beta^{(0)}} \sum_{i,j} \alpha'_i\alpha'_j k(\underline{X}_i, \underline{X}_j) + C\sum_{i=1}^{n} \max(0, 1 - Y_i(\sum_j \alpha'_j k(\underline{X}_j, \underline{X}_i) + \beta^{(0)}))$$

Dual

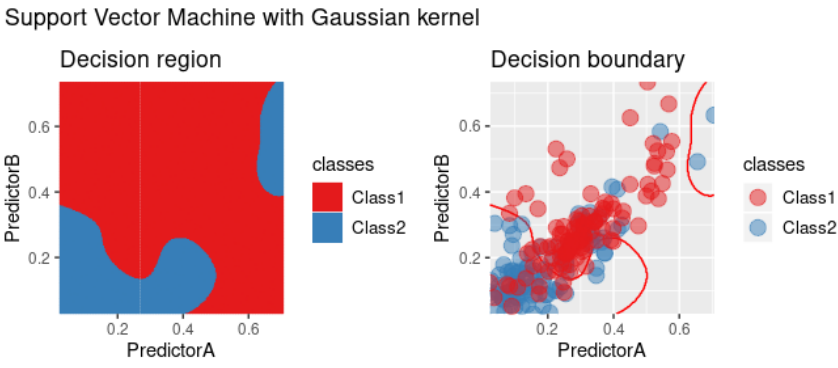
- Dual:

$$\max_{\alpha\geq 0,\mu\geq 0} Q(\alpha,\mu) \Leftrightarrow \max_{0\leq\alpha\leq C} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j Y_i Y_j k(\underline{X}_i, \underline{X}_j)$$

SVM

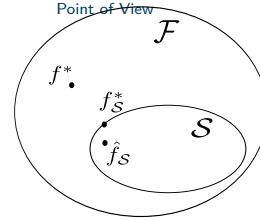

Support Vector Machine with polynomial kernel
Decision region
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2
Decision boundary
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2

SVM


Support Vector Machine with Gaussian kernel
Decision region
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2
Decision boundary
PredictorB
PredictorA
0.2
0.4
0.6
classes
Class1
Class2

Outline

Bias-Variance Dilemma



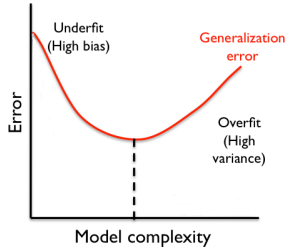
- General setting:
  - $\mathcal{F} = \{\text{measurable functions } \mathcal{X} \to \mathcal{Y}\}$
  - Best solution: $f^\star = \text{argmin}_{f \in \mathcal{F}} \mathcal{R}(f)$
  - Class $\mathcal{S} \subset \mathcal{F}$ of functions
  - Ideal target in $\mathcal{S}$: $f^\star_{\mathcal{S}} = \text{argmin}_{f \in \mathcal{S}} \mathcal{R}(f)$
  - Estimate in $\mathcal{S}$: $\widehat{f}_{\mathcal{S}}$ obtained with some procedure

Approximation error and estimation error (Bias/Variance)

$$\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star) = \underbrace{\mathcal{R}(f^\star_{\mathcal{S}}) - \mathcal{R}(f^\star)}_{\text{Approximation error}} + \underbrace{\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star_{\mathcal{S}})}_{\text{Estimation error}}$$

- Approx. error can be large if the model $\mathcal{S}$ is not suitable.
- Estimation error can be large if the model is complex.

Under-fitting / Over-fitting Issue



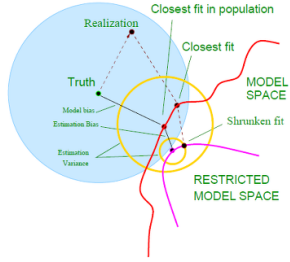
- Different behavior for different model complexity
- **Low complexity model** are easily learned but the approximation error (**bias**) may be large (**Under-fit**).
- **High complexity model** may contain a good ideal target but the estimation error (**variance**) can be large (**Over-fit**)

Bias-variance trade-off $\iff$ avoid **overfitting** and **underfitting**

- **Rk:** Better to think in term of method (including feature engineering and specific algorithm) rather than only of model.

Source: Unknown

Simplified Models



Bias-Variance Issue

- Most complex models may not be the best ones due to the variability of the estimate.

- Naive idea: can we *simplify* our model without loosing too much?
  - by using only a subset of the variables?
  - by forcing the coefficients to be small?
- Can we do better than exploring all possibilities?

Source: Tibshirani et al.

Linear Models

- **Setting**: Gen. linear model = prediction of Y by $h(\underline{x}^\top \beta)$.

Model coefficients

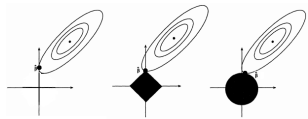
- Model entirely specified by β.
- Coefficientwise:
  - $\beta^{(i)} = 0$ means that the ith covariate is not used.
  - $\beta^{(i)} \sim 0$ means that the ith covariate as a *low* influence...

- If some covariates are useless, better use a simpler model...

Submodels

- *Simplify* the model through a constraint on β!
- Examples:
  - Support: Impose that $\beta^{(i)} = 0$ for $i \notin I$.
  - Support size: Impose that $\|\beta\|_0 = \sum_{i=1}^{d} \mathbf{1}_{\beta^{(i)} \neq 0} < C$
  - Norm: Impose that $\|\beta\|_p < C$ with $1 \leq p$ (Often $p = 2$ or $p = 1$)

Norms and Sparsity



Sparsity

- β is sparse if its number of non-zero coefficients (ℓ_0) is small...
- Easy interpretation in terms of dimension/complexity.

Norm Constraint and Sparsity

- Sparsest solution obtained by definition with the ℓ_0 norm.
- No induced sparsity with the ℓ_2 norm...
- Sparsity with the ℓ_1 norm (can even be proved to be the same as with the ℓ_0 norm under some assumptions).
- Geometric explanation.

Source: Tibshirani et al.

Constraint and Penalization

Constrained Optimization

- Choose a constant C.
- Compute β as

$$\underset{\beta \in \mathbb{R}^d, \|\beta\|_p \leq C}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \bar{\ell}(Y_i, h(\underline{x}_i{}^\top \beta))$$

Lagrangian Reformulation

- Choose λ and compute β as

$$\underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \bar{\ell}(Y_i, h(\underline{x}_i{}^\top \beta)) + \lambda \|\beta\|_p^{p'}$$

  with $p' = p$ except if $p = 0$ where $p' = 1$.
- Easier calibration... but no explicit model $\mathcal{S}$.

- **Rk:** $\|\beta\|_p$ is not scaling invariant if $p \neq 0$...
- Initial rescaling issue.

Penalization

Penalized Linear Model

- Minimization of

$$\underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \bar{\ell}(Y_i, h(\underline{x}_i^\top \beta)) + \text{pen}(\beta)$$

  where $\text{pen}(\beta)$ is a (sparsity promoting) penalty
- Variable selection if β is sparse.

Classical Penalties

- AIC: $\text{pen}(\beta) = \lambda \|\beta\|_0$ (non-convex / sparsity)
- Ridge: $\text{pen}(\beta) = \lambda \|\beta\|_2^2$ (convex / no sparsity)
- Lasso: $\text{pen}(\beta) = \lambda \|\beta\|_1$ (convex / sparsity)
- Elastic net: $\text{pen}(\beta) = \lambda_1 \|\beta\|_1 + \lambda_2 \|\beta\|_2^2$ (convex / sparsity)

- Easy optimization if pen (and the loss) is convex...
- **Need to specify λ to define a ML method!**

Penalization and Cross-Validation

Practical Selection Methodology

- Choose a penalty family pen_λ.
- Compute a CV risk for the penalty pen_λ for all $\lambda \in \Lambda$.
- Determine $\widehat{\lambda}$ the λ minimizing the CV risk.
- Compute the final model with the penalty $\text{pen}_{\widehat{\lambda}}$.

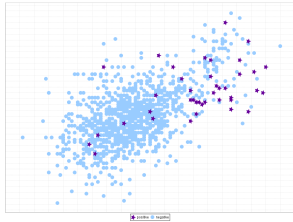
- CV allows to select a ML method, penalized estimation with a penalty $\text{pen}_{\widehat{\lambda}}$, not a single predictor hence the need of a final reestimation.

Why not using CV on a grid?

- Grid size scales exponentially with the dimension!
- **If the penalized minimization is easy**, much cheaper to compute the CV risk for all $\lambda \in \Lambda \ldots$
- CV performs best when the set of candidates is not too big (or is structured...)

Outline

Unbalanced and Rebalanced Dataset



Unbalanced Class

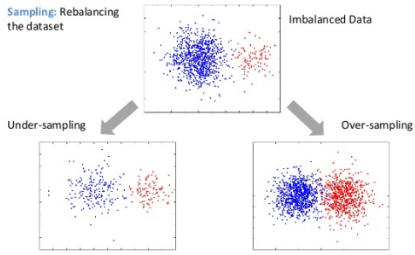
- **Setting:** One of the class is much more present than the other.
- **Issue:** Classifier *too attracted* by the majority class!

Rebalanced Dataset

- **Setting:** Class proportions are different in the training and testing set (stratified sampling)
- **Issue:** Training risks are not estimate of testing risks.

Source: University of Granada

Resampling Strategies



Resampling

- Modify the training dataset so that the classes are more balanced.
- Two flavors:
  - Sub-sampling which spoils data,
  - Over-sampling which needs to create *new* examples.
- **Issues**: Training data is not anymore representative of testing data
- **Hard to do it right!**

Source: Oracle

Resampling Effect

Testing

- Testing class prob.: $\pi_t(k)$
- Testing risk target:
$$\mathbb{E}_{\pi_t}[\ell(Y, f(\underline{X}))] = \sum_k \pi_t(k)\mathbb{E}[\ell(Y, f(\underline{X}))|Y = k]$$

Training

- Training class prob.: $\pi_{tr}(k)$
- Training risk target:
$$\mathbb{E}_{\pi_{tr}}[\ell(Y, f(\underline{X}))] = \sum_k \pi_{tr}(k)\mathbb{E}[\ell(Y, f(\underline{X}))|Y = k]$$

Implicit Testing Risk Using the Training One

- Amounts to use a weighted loss:
$$\begin{aligned}\mathbb{E}_{\pi_{tr}}[\ell(Y, f(\underline{X}))] &= \sum_k \pi_{tr}(k)\mathbb{E}[\ell(Y, f(\underline{X}))|Y = k] \\ &= \sum_k \pi_t(k)\mathbb{E}\left[\frac{\pi_{tr}(k)}{\pi_t(k)}\ell(Y, f(\underline{X}))\middle| Y = k\right] \\ &= \mathbb{E}_{\pi_t}\left[\frac{\pi_{tr}(Y)}{\pi_t(Y)}\ell(Y, f(\underline{X}))\right]\end{aligned}$$
- Put more weight on less probable classes!

Weighted Loss

- In unbalanced situation, often the **cost** of misprediction is not the same for all classes (e.g. medical diagnosis, credit lending...)
- Much better to use this explicitly than to do blind resampling!

Weighted Loss

- **Weighted loss:**

$$\ell(Y, f(\underline{X})) \to C(Y)\ell(Y, f(\underline{X}))$$

- Weighted risk target:

$$\mathbb{E}[C(Y)\ell(Y, f(\underline{X}))]$$

- **Rk:** Strong link with ℓ as C is independent of f.
- Often allow reusing algorithm constructed for ℓ.
- C may also depend on $\underline{X}$...

Weighted Loss, $\ell^{0/1}$ loss and Bayes Classifier

- The Bayes classifier is now:

$$f^\star = \operatorname{argmin} \mathbb{E}[C(Y)\ell(Y, f(\underline{X}))] = \operatorname{argmin} \mathbb{E}_{\underline{X}}\Big[\mathbb{E}_{Y|\underline{X}}[C(Y)\ell(Y, f(\underline{X}))]\Big]$$

Bayes Predictor

- For $\ell^{0/1}$ loss,

$$f^\star(\underline{X}) = \operatorname*{argmax}_k C(k)\mathbb{P}(Y = k|\underline{X})$$

- Same effect than a threshold modification for the binary setting!

- Allow putting more emphasis on some classes than others.

Linking Weights and Proportions

Cost and Proportions

- Testing risk target:
$$\mathbb{E}_{\pi_t}[C_t(Y)\ell(Y, f(\underline{X}))] = \sum_k \pi_t(k)C_t(k)\mathbb{E}[\ell(Y, f(\underline{X}))|Y = k]$$
- Training risk target
$$\mathbb{E}_{\pi_{tr}}[C_{tr}(Y)\ell(Y, f(\underline{X}))] = \sum_k \pi_{tr}(k)C_{tr}(k)\mathbb{E}[\ell(Y, f(\underline{X}))|Y = k]$$
- **Coincide if**
$$\pi_t(k)C_t(k) = \pi_{tr}(k)C_{tr}(k)$$

- Lots of flexibility in the choice of C_t, C_{tr} or π_{tr}.

Combining Weights and Resampling

Weighted Loss and Resampling

- **Weighted loss:** choice of a weight $C_t \neq 1$.
- **Resampling:** use a $\pi_{tr} \neq \pi_t$.

- Stratified sampling may be used to reduce the size of a dataset without loosing a low probability class!

Combining Weights and Resampling

- **Weighted loss:** use $C_{tr} = C_t$ as $\pi_{tr} = \pi_t$.
- **Resampling:** use an implicit $C_t(k) = \pi_{tr}(k)/\pi_t(k)$.
- **Combined:** use $C_{tr}(k) = C_t(k)\pi_t(k)/\pi_{tr}(k)$

- Most ML methods allow such weights!

References

T. Hastie, R. Tibshirani, and J. Friedman.
The Elements of Statistical Learning.
Springer Series in Statistics, 2009

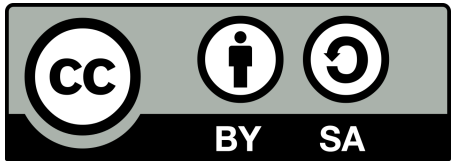
M. Mohri, A. Rostamizadeh, and A. Talwalkar.
Foundations of Machine Learning (2nd ed.)
MIT Press, 2018

A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd ed.)
O'Reilly, 2022

Ch. Giraud.
Introduction to High-Dimensional Statistics.
CRC Press, 2014

S. Bubeck.
Convex Optimization: Algorithms and Complexity.
Now Publisher, 2015

Licence and Contributors



Contributors

- Main contributor: E. Le Pennec
- Contributors: S. Boucheron, A. Dieuleveut, A.K. Fermin, S. Gadat, S. Gaiffas, A. Guilloux, Ch. Keribin, E. Matzner, M. Sangnier, E. Scornet.

Outline

Outline

ML Optimization Problem

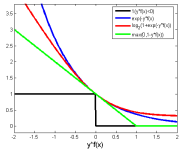
Typical Optimization Problem in ML

- ML Opt.: $\underset{\boldsymbol{w}\in\mathbb{R}^d}{\operatorname{argmin}}\, G(\boldsymbol{w})$ with $G(\boldsymbol{w}) = F(\boldsymbol{w}) + R(\boldsymbol{w})$ where
  - F a goodness-of-fit function: $F(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(Y_i, f_{\boldsymbol{w}}(\underline{X}_i))$
    where ℓ is some loss and $f_{\boldsymbol{w}}$ a parametric predictor,
  - R a regularizer: $R(\boldsymbol{w}) = \lambda\, \text{pen}(\boldsymbol{w})$
    where $\text{pen}(\cdot)$ is some penalization function.

Examples

- Predictor:
  - Linear $f_{\boldsymbol{w}}(\underline{X}_i) = \underline{X}_i{}^\top \boldsymbol{w}$
  - Neural Nets...
- Regularizer:
  - $\text{pen}(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$ (ridge)
  - $\text{pen}(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$ (Lasso)

Losses



Classification

- Logistic loss, $\ell(y, f) = \log(1 + e^{-yf})$
- Hinge loss, $\ell(y, f) = (1 - yf)_+$
- Quadratic hinge loss, $\ell(y, f) = \frac{1}{2}(1 - yf)_+^2$
- Huber loss $\ell(y, f) = -4yf\mathbf{1}_{yf<-1} + (1 - yf)_+^2\mathbf{1}_{yf\geq -1}$

- These losses are convex upper bound of the $0/1$ loss $\ell(y, y') = \mathbf{1}_{yy'\leq 0}$.

Regression

- Quadratic loss, $\ell(y, f) = \frac{1}{2}(y - f)^2$
- Absolute loss, $\ell(y, f) = \frac{1}{2}|y - f|$

ML Problem

- Minimization of

$$G(\boldsymbol{w}) = F(\boldsymbol{w}) + R(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(Y_i, \langle \underline{X}_i, \boldsymbol{w} \rangle) + \lambda \operatorname{pen}(\boldsymbol{w})$$

Gradient and Hessian of F

- Gradient:

$$\nabla F(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \bar{\ell}(Y_i, \langle \underline{X}_i, \boldsymbol{w} \rangle)\underline{X}_i$$

with $\bar{\ell}(y, f) = \dfrac{\partial \ell(y, f)}{\partial f}$

- Hessian matrix:

$$\nabla^2 F(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \bar{\ell}'(Y_i, \langle \underline{X}_i, \boldsymbol{w} \rangle)\underline{X}_i \underline{X}_i^\top$$

with $\bar{\ell}'(y, f) = \dfrac{\partial^2 \ell(y, f)}{\partial f^2}$

Outline

Zero Order Method

Zero Order Heuristic

- Zero order approximation:

$$G(\boldsymbol{w}) = G(\boldsymbol{w}') + O(\|\boldsymbol{w} - \boldsymbol{w}'\|)$$

Optimization in a Compact Set

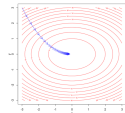
- Optimization problem:

$$\boldsymbol{w}^\star \in \operatorname*{argmin}_{\boldsymbol{w} \in [0,1]^d} G(\boldsymbol{w}).$$

Exhaustive Search

- **Algorithm:**
  - Evaluate G on a grid,
  - Approximate the minimizer by the minimizer in the grid.
- If G is C-Lipschitz, evaluating G on a grid of precision $\epsilon/(\sqrt{d}C)$ is sufficient to find a ϵ-minimizer of G.
- Required number of evaluation: $N_\epsilon = O\left((C\sqrt{d}/\epsilon)^d\right)$

First Order Method



First Order Heuristic

- First order approximation:
$$G(\boldsymbol{w}) = G(\boldsymbol{w}') + \nabla G(\boldsymbol{w}')^\top (\boldsymbol{w} - \boldsymbol{w}') + o(\|\boldsymbol{w} - \boldsymbol{w}'\|)$$
- Best descent direction: $-\nabla G(\boldsymbol{w}')$

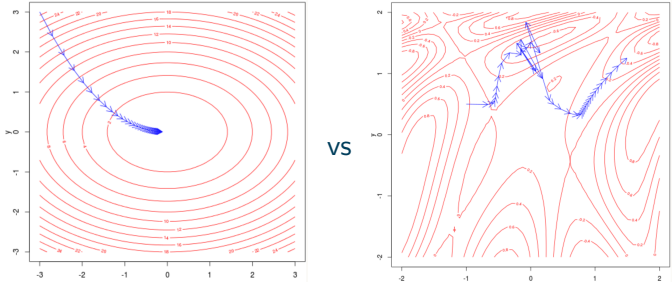
Gradient Descent Algorithm

- Start from a point $\boldsymbol{w}^{[0]}$ and let $k = 0$.
- Repeat until convergence:
  - $\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k]} - \alpha^{[k]} \nabla G(\boldsymbol{w}^{[k]})$
  - $k \to k + 1$

with $\alpha^{[k]}$ a sequence of (small) steps.

Source: Lijia Yu

Do The Gradient Descent Converge?



- **Not always!**

Assumptions

- Convexity assumption on G.
- Regularity assumption on G
- Size of the steps $\alpha^{[k]}$.

Source: Lijia Yu

Convexity and Regularity

Convex function

- A function $G : \mathbb{R}^d \to \mathbb{R}$ is **convex** on $\mathbb{R}^d$ if, for all $x, y \in \mathbb{R}^d$, for all $\lambda \in [0,1]$,
$$G(\lambda x + (1-\lambda)y) \leq \lambda G(x) + (1-\lambda) f(y).$$
- If $f_{\boldsymbol{w}}$ is linear, $F(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^n \ell(Y_i, f_{\boldsymbol{w}}(\underline{X}_i))$ is **convex** iff
$$f \mapsto \ell(y_i, f)$$
is convex for any $i = 1, \ldots, n$.

Regularity

- G is L-**smooth** if it is continuously differentiable and if
$$\left\|\nabla G(\boldsymbol{w}) - \nabla G(\boldsymbol{w}')\right\|_2 \leq L \left\|\boldsymbol{w} - \boldsymbol{w}'\right\|_2 \quad \text{for any } \boldsymbol{w}, \boldsymbol{w}' \in \mathbb{R}^d$$
- If G is twice differentiable, this is equivalent to assuming
$$\lambda_{\max}(\nabla^2 G(\boldsymbol{w})) \leq L \quad \text{for any } \boldsymbol{w} \in \mathbb{R}^d$$
(largest eigenvalue of the Hessian matrix of G smaller than L)

Least-Squares Regression

Formulation

- Minimization of

$$G(\boldsymbol{w}) = F(\boldsymbol{w}) + R(\boldsymbol{w}) = \frac{1}{2n}\sum_{i=1}^{n}(Y_i - \underline{X}_i{}^\top \boldsymbol{w})^2 + \lambda\,\mathrm{pen}(\boldsymbol{w})$$

Gradient and Hessian of F

- Gradient:

$$\nabla F(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(\underline{X}_i{}^\top \boldsymbol{w} - Y_i)\underline{X}_i$$

- Hessian:

$$\nabla^2 F(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}\underline{X}_i\underline{X}_i{}^\top$$

so that

$$L = \lambda_{\max}\Big(\frac{1}{n}\sum_{i=1}^{n}\underline{X}_i\underline{X}_i{}^\top\Big)$$

Logistic Regression

Formulation

- Minimization of

$$G(\boldsymbol{w}) = F(\boldsymbol{w}) + R(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + e^{(-Y_i(\underline{X}_i{}^\top \boldsymbol{w}))}\right) + \lambda\, \mathrm{pen}(\boldsymbol{w})$$

Gradient and Hessian of F

- Gradient:

$$\nabla F(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} Y_i(\sigma(Y_i \underline{X}_i{}^\top \boldsymbol{w}) - 1)\underline{X}_i$$

- Hessian:

$$\nabla^2 F(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \sigma(Y_i \underline{X}_i{}^\top \boldsymbol{w})(1 - \sigma(Y_i \underline{X}_i{}^\top \boldsymbol{w}))\underline{X}_i \underline{X}_i{}^\top$$

  so that F is L-smooth with

$$L \leq \frac{1}{4}\lambda_{\max}\left(\frac{1}{n}\sum_{i=1}^{n} \underline{X}_i \underline{X}_i{}^\top\right)$$

- $\sigma(t) = e^{-t}/(1 + e^{-t})$.

Convergence of GD

Theorem

- Let $G : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth convex function. Let $\boldsymbol{w}^\star$ be the minimum of f on $\mathbb{R}^d$. Then, Gradient Descent with step size $\alpha \leq 1/L$ satisfies
$$G(\boldsymbol{w}^{[k]}) - G(\boldsymbol{w}^\star) \leq \frac{\|\boldsymbol{w}^{[0]} - \boldsymbol{w}^\star\|_2^2}{2\alpha k}.$$

- In particular, for $\alpha = 1/L$,
$$N_\epsilon = O(L\|\boldsymbol{w}^{[0]} - \boldsymbol{w}^\star\|_2^2/(2\epsilon))$$
iterations are sufficient to get an ϵ-approximation of the minimal value of G.
- Bound is independent of the dimension d.
- **Weak dependency** hidden in the constant L...

Descent Lemma

A key Property: the Descent Lemma.

- If G is convex and L-smooth, then for any $\boldsymbol{w}, \boldsymbol{w}' \in \mathbb{R}^d$

$$G(\boldsymbol{w}) \leq G(\boldsymbol{w}') + \nabla G(\boldsymbol{w}')^\top (\boldsymbol{w} - \boldsymbol{w}') + \frac{L}{2} \|\boldsymbol{w} - \boldsymbol{w}'\|_2^2 .$$

Link with the Gradient Descent Algorithm

- At step k, for any $\boldsymbol{w}$,

$$\begin{aligned} G(\boldsymbol{w}) &\leq G(\boldsymbol{w}^{[k]}) + \nabla G(\boldsymbol{w}^{[k]})^\top (\boldsymbol{w} - \boldsymbol{w}^{[k]}) + \frac{L}{2} \left\|\boldsymbol{w} - \boldsymbol{w}^{[k]}\right\|_2^2 \\ &\leq G(\boldsymbol{w}^{[k]}) + \frac{L}{2} \left\|\boldsymbol{w} - (\boldsymbol{w}^{[k]} - \frac{1}{L} \nabla G(\boldsymbol{w}^{[k]}))\right\|^2 - \frac{1}{2L} \left\|\nabla G(\boldsymbol{w}^{[k]})\right\|^2 \end{aligned}$$

- Optimizing the upper bound in $\boldsymbol{w}$ leads to

$$\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k]} - \frac{1}{L} \nabla G(\boldsymbol{w}^{[k]})$$

- **Rk:** Newton iteration comes from the approximation $G(\boldsymbol{w}) = G(\boldsymbol{w}') + \nabla G(\boldsymbol{w}')^\top (\boldsymbol{w} - \boldsymbol{w}') + \frac{1}{2} (\boldsymbol{w} - \boldsymbol{w}')^\top \nabla^2 G(\boldsymbol{w}') (\boldsymbol{w} - \boldsymbol{w}') + o(\|\boldsymbol{w} - \boldsymbol{w}'\|^2)$

Faster Rate - Strongly Convex Function

Strong convexity

- A function $G : \mathbb{R}^d \to R$ is **μ-strongly convex** if $x \mapsto G(x) - \frac{\mu}{2}\|x\|_2^2$ is convex.
- If G is differentiable, this is equivalent to, for all $x, y \in \mathbb{R}^d$,
$$G(y) \geq G(x) + \nabla G(x)^\top (y - x) + \frac{\mu}{2}\|y - x\|_2^2.$$
- If G is twice differentiable, this is equivalent to
$$\lambda_{min}(\nabla^2 G(x)) \geq \mu.$$

Theorem

- Let $G : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth, μ strongly convex function. Let $\boldsymbol{w}^\star$ be the minimum of G on $\mathbb{R}^d$. Then, Gradient Descent with step size $\alpha \leq 1/L$ satisfies
$$G(\boldsymbol{w}^{[k]}) - G(\boldsymbol{w}^\star) \leq \frac{1}{2\alpha}\Big(1 - \alpha\mu\Big)^k \|G(\boldsymbol{w}^{[0]}) - G(\boldsymbol{w}^\star)\|_2^2.$$

- **Rk:** $N_\epsilon = O(-\log \epsilon/(\alpha\mu))$ iterations are sufficient to obtain an ϵ-minimizer.

Lipschitz Convex Function - Slower Rate

Convexity

- If G is a convex function then for any $\boldsymbol{w} \in \mathbb{R}^d$, there exists a (not necessarily unique) subgradient $\delta G(\boldsymbol{w})$ such that
$$G(\boldsymbol{w}') \geq G(\boldsymbol{w}) + \delta G(\boldsymbol{w})^\top (\boldsymbol{w}' - \boldsymbol{w}) \quad \text{for any } \boldsymbol{w}' \in \mathbb{R}^d$$
- If G is differentiable then it is unique and equal to the gradient.

- **Subgradient Descent Algorithm:** Gradient Descent with subgradient instead of gradient.

Regularity

- G is C-**Lipschitz** if
$$|G(\boldsymbol{w}) - G(\boldsymbol{w}')| \leq C \left\|\boldsymbol{w} - \boldsymbol{w}'\right\|_2 \quad \text{for any } \boldsymbol{w}, \boldsymbol{w}' \in \mathbb{R}^d$$
- If G is differentiable, this is equivalent to assuming
$$\|\nabla G(\boldsymbol{w})\| \leq C \quad \text{for any } \boldsymbol{w} \in \mathbb{R}^d$$

Lipschitz Convex Function - Slower Rate

Theorem

- Let $G : \mathbb{R}^d \to \mathbb{R}$ be a convex function, C-Lipschitz in $B(\boldsymbol{w}^\star, R)$ where $\boldsymbol{w}^\star$ be the minimizer of f on $\mathbb{R}^d$. Assume that

$$\alpha^{[k]} > 0, \quad \alpha^{[k]} \to 0, \quad \sum_k \alpha^{[k]} = +\infty$$

and $\left\| \boldsymbol{w}^{[0]} - \boldsymbol{w}^\star \right\| \leq R$ Then, Subgradient Descent with step size $\alpha^{[k]}$ satisfies

$$\min_{k' \leq k} G(\boldsymbol{w}^{[k']}) - G(\boldsymbol{w}^\star) \leq C \frac{R^2 + \sum_{k'=0}^{k} (\alpha^{[k']})^2}{2 \sum_{k'=0}^{k} \alpha^{[k']}}$$

- In particular, for $\alpha^{[k]} = \frac{r}{\sqrt{k+1}}$

$$\min_{k' \leq k} G(\boldsymbol{w}^{[k']}) - G(\boldsymbol{w}^\star) \leq C \frac{R^2 + r^2 \log(k+1)}{4r\sqrt{k+1}}$$

and

$$N_\epsilon = O\left((C(-\log \epsilon)/\epsilon)^2 \right)$$

In practice, how to choose α?

- **Theoretical setting of $\alpha = 1/L$ is a worst case scenario.**

Exact line search

- At each step, choose the best α by optimizing
$$\alpha^{[k]} = \underset{\alpha>0}{\operatorname{argmin}}\, G(\boldsymbol{w} - \alpha \nabla G(\boldsymbol{w})).$$
- **Too costly!**

Backtracking line search

- Fix a parameter $0 < \beta < 1$ and set $\alpha = \alpha_{init}$
- At each iteration,
  - while $G(\boldsymbol{w} - \alpha \nabla G(\boldsymbol{w})) > G(\boldsymbol{w}) - \frac{\alpha}{2}\|\nabla G(\boldsymbol{w})\|^2$, modify $\alpha \leftarrow \beta\alpha$.
  - Use the final α as the stepsize for this iteration.
- **Simple and work pretty well in practice.**
- Theoretical guarantees available.

Outline

Function Decomposition and Descent

Function Decomposition: $G = F + R$

- F convex and L-**smooth**
- R convex and **simple**

Another Descent Algorithm

- At step k, for any $\boldsymbol{w}$,

$$\begin{aligned} G(\boldsymbol{w}) &\leq F(\boldsymbol{w}) + R(\boldsymbol{w}) \\ &\leq F(\boldsymbol{w}^{[k]}) + \nabla F(\boldsymbol{w}^{[k]})^\top (\boldsymbol{w} - \boldsymbol{w}^{[k]}) + \frac{L}{2} \left\| \boldsymbol{w} - \boldsymbol{w}^{[k]} \right\|_2^2 + R(\boldsymbol{w}) \\ &\leq F(\boldsymbol{w}^{[k]}) + \frac{L}{2} \left\| \boldsymbol{w} - (\boldsymbol{w}^{[k]} - \frac{1}{L} \nabla F(\boldsymbol{w}^{[k]})) \right\|^2 - \frac{1}{2L} \left\| \nabla F(\boldsymbol{w}^{[k]}) \right\|^2 + R(\boldsymbol{w}) \end{aligned}$$

- Optimizing the upper bound in $\boldsymbol{w}$ leads to

$$\boldsymbol{w}^{[k+1]} = \operatorname*{argmin}_{\boldsymbol{w}} \frac{1}{2} \left\| \boldsymbol{w} - (\boldsymbol{w}^{[k]} - \frac{1}{L} \nabla F(\boldsymbol{w}^{[k]})) \right\|^2 + \frac{1}{L} R(\boldsymbol{w})$$

- R **simple** means that this minimization is easy.

Proximal Operator and Proximal Gradient Descent

Proximal Operator

- For any convex function R:

$$\text{prox}_\gamma R(\boldsymbol{w}') = \underset{\boldsymbol{w}}{\text{argmin}} \frac{1}{2} \|\boldsymbol{w} - \boldsymbol{w}'\|^2 + \gamma R(\boldsymbol{w})$$

- Generalization of the projection operator:
  - $R(\boldsymbol{w}) = \mathbf{1}_\Omega$: $\text{prox}_\gamma R(\boldsymbol{w}') = P_\Omega \boldsymbol{w}'$
  - $R(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$: $\text{prox}_\gamma R(\boldsymbol{w}') = \frac{1}{1+\gamma}\boldsymbol{w}$.
  - $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$: $\text{prox}_\gamma R(\boldsymbol{w}') = T_\gamma(\boldsymbol{w}')$ with $T_\gamma(\boldsymbol{w})_i = \text{sign}(\boldsymbol{w}_i)\max(0, |\boldsymbol{w}_i| - \gamma)$ (soft thresholding).

Proximal Gradient Descent Algorithm for simple R

- Start from a point $\boldsymbol{w}^{[0]}$ and let $k = 0$.
- Repeat until convergence:
  - $\boldsymbol{w}^{[k+1]} = \text{prox}_{1/L} R\left(\boldsymbol{w}^{[k]} - \alpha^{[k]} \nabla F(\boldsymbol{w}^{[k]})\right)$
  - $k \to k + 1$

with $\alpha^{[k]}$ a sequence of (small) steps.

Theoretical Guarantees

- **Same as smooth case!**

Rates

- F L-smooth and R simple:

$$G(\boldsymbol{w}^{[k]}) - G(\boldsymbol{w}^\star) \leq \frac{\|\boldsymbol{w}^{[0]} - \boldsymbol{w}^\star\|_2^2}{2\alpha k}.$$

  and $N_\epsilon = O(L\|\boldsymbol{w}^{[0]} - \boldsymbol{w}^\star\|_2^2/2\epsilon)$.

- F L-smooth and μ-convex and R simple:

$$G(\boldsymbol{w}^{[k]}) - G(\boldsymbol{w}^\star) \leq \frac{1}{2\alpha}\Big(1 - \alpha\mu\Big)^k \|G(\boldsymbol{w}^{[0]}) - G(\boldsymbol{w}^\star)\|_2^2.$$

  and $N_\epsilon = O(-\log\epsilon/(\alpha\mu))$.

- F C-Lipschitz and R is the characteristic function of a convex set:

$$G(\boldsymbol{w}^{[k]}) - G(\boldsymbol{w}^\star) \leq C\frac{R^2 + r^2\log(k+1)}{4r\sqrt{k+1}}$$

  and $N_\epsilon = O\left((C(-\log\epsilon)/\epsilon)^2\right)$.

Outline

Coordinate Descent

Heuristic

- Is $\boldsymbol{w}$ a minimizer of a G when
$$G(\boldsymbol{w} + t\boldsymbol{e}_i) \geq G(\boldsymbol{w}) \quad \forall t \in \mathbb{R} \text{ and } 1 \leq i \leq d,$$
with G convex?
- **YES** if G is smooth ($\mathcal{C}^1$), G is separable or G is a sum of a smooth ($\mathcal{C}^1$) and a separable function ($G(w) = \sum_{i=1}^{d} g_i(w^{(i)})$).
- **NO** otherwise.

Coordinate Descent Algorithm

- Start from an arbitrary $\boldsymbol{w}^{[0]}$
- Repeat until convergence:
  - Pick a coordinate i
  - Set $\boldsymbol{w}^{[k+1],(j)} = \boldsymbol{w}^{[k],(j)}$ for $j \neq i$.
  - **Optimize** only the ith coordinate to obtain $\boldsymbol{w}^{[k+1],(i)}$
  - $k \to k+1$
- State-of-the art is several ML problems!

Exact Coordinate Descent

Exact Coordinate Descent (CD)

- Start from an arbitrary $\boldsymbol{w}^{[0]}$
- Repeat until convergence:
  - Pick a coordinate i
  - Set $\boldsymbol{w}^{[k+1],(j)} = \boldsymbol{w}^{[k],(j)}$ for $j \neq i$.
  - Compute
    $$\boldsymbol{w}^{[k+1],(i)} = \underset{z \in \mathbb{R}}{\operatorname{argmin}}\, G(\boldsymbol{w}^{[k],(1)}, \dots \boldsymbol{w}^{[k],(i-1)}, z, \boldsymbol{w}^{[k],(i+1)}, \dots, \boldsymbol{w}^{[k],(d)})$$
  - $k \to k+1$

- Several way to choose the coordinate i: uniform sampling, deterministic cycle...
- Only 1D optimization problems to solve... but probably a lot of them.

Theorem - Warga (1963)

- If G is continuously differentiable and strictly convex, then exact coordinate descent converges to a minimum.

Coordinate Gradient Descent

Coordinate Gradient Descent (CGD)

- Start from an arbitrary $\boldsymbol{w}^{[0]}$
- Repeat until convergence:
  - Pick a coordinate i
  - Set $\boldsymbol{w}^{[k+1],(j)} = \boldsymbol{w}^{[k],(j)}$ for $j \neq i$.
  - Compute
$$\boldsymbol{w}^{[k+1],(i)} = \boldsymbol{w}^{[k],(i)} - \alpha^{[k]} \nabla^{(i)} G(\boldsymbol{w}^{[k]})$$
  - $k \to k+1$

- For smooth function, step-size $\alpha^{[k]}$ can be taken as $\alpha^{[k]} = 1/L_i$ where i is the coordinate chosen and L_i the Lipschitz constant of
$$G^i(z) = G(\boldsymbol{w} + z\boldsymbol{e}_i) = G(\boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(i-1)}, \boldsymbol{w}^{(i)} + z, \boldsymbol{w}^{(i+1)}, \ldots, \boldsymbol{w}^{(d)})$$
- If G is L-smooth, $L_i \leq L$.

Rate of Coordinate Gradient Descent

Theorem - Nesterov (2012)

- Assume that G is convex and smooth and that each G^i is L_i-smooth. Consider a sequence $\{\boldsymbol{w}^{[k]}\}$ given by CGD with $\alpha^{[k]} = 1/L_{i_k}$ and coordinates $i_1, i_2, \ldots$ chosen at random: i.i.d and uniform distribution in $\{1, \ldots, d\}$. Then

$$\mathbb{E}\Big[G(\boldsymbol{w}^{[k+1]}) - G(\boldsymbol{w}^\star)\Big] \leq \frac{d}{d+k}\Big((1-\frac{1}{d})(G(\boldsymbol{w}^{[0]}) - G(\boldsymbol{w}^\star)) + \frac{1}{2}\left\|\boldsymbol{w}^{[0]} - \boldsymbol{w}^\star\right\|_L^2\Big),$$

with $\|\boldsymbol{w}\|_L^2 = \sum_{j=1}^d L_j \boldsymbol{w}_j^2$.

- Bound in expectation, since coordinates are taken at random.
- For cycling coordinates $i_k = (k \mod d) + 1$ the bound is much worse.
- Similar result when $G = F + R$ with F smooth and R simple and separable.

Comparison of Gradient Descent and Coordinate Gradient Descent

Gradient Descent

- Cost of an iteration $O(d)$.
- Number of iteration to obtain an ϵ-minimizer:

$$N_\epsilon = \frac{L \left\| \boldsymbol{w}^{[0]} - \boldsymbol{w}^\star \right\|_2^2}{2\epsilon}$$

Coordinate Gradient Descent

- Cost of an iteration $O(1)$.
- Number of iteration to obtain an ϵ-minimizer:

$$N_\epsilon = \frac{d}{\epsilon}\Big((1 - \frac{1}{d})(G(\boldsymbol{w}^{[0]}) - G(\boldsymbol{w}^\star)) + \frac{1}{2}\left\| \boldsymbol{w}^{[0]} - \boldsymbol{w}^\star \right\|_L^2\Big)$$

- Same order of complexity but smaller constant for CGD as soon as $(G(\boldsymbol{w}^{[0]}) - G(\boldsymbol{w}^\star)) \ll L \left\| \boldsymbol{w}^{[0]} - \boldsymbol{w}^\star \right\|_2^2$.
- In practice, often much faster (especially when dealing with anisotropic regularity).

Outline

Gradient Descent Acceleration

- Can we improve the number of iterations $O(L/\epsilon)$ (L-smooth) and $O(\frac{L}{\mu}\log(1/\epsilon))$ (L-smooth and μ strongly-convex) ?
- **Heuristic:** add some momentum to propagate gradients.

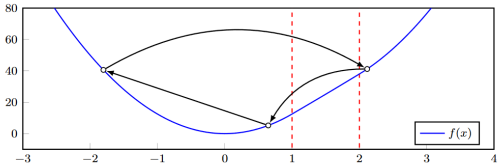
Polyak's momentum algorithm - Heavy ball method

- **Input**: starting point $\boldsymbol{w}^{[0]}$, step size $\alpha^{[k]} > 0$, momentum $\beta \in [0, 1]$
- While *not converge* do
  - $\boldsymbol{w}^{[k+1/2]} = \boldsymbol{w}^{[k]} - \alpha^{[k]} \nabla G(\boldsymbol{w}^{[k]})$
  - $\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k+1/2]} + \beta(\boldsymbol{w}^{[k]} - \boldsymbol{w}^{[k-1]})$
  - $k \leftarrow k + 1$
- **Return** last $\boldsymbol{w}^{[k+1]}$.

- Most classical value for β: 0.9
- When $\alpha^{[k]} = \alpha$, the update equation can be written

$$\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k]} - \alpha \sum_{t=1}^{k} \beta^{k-t} \nabla G(\boldsymbol{w}^{(t)}).$$

Polyak's Momentum Failure



Counter Example

- Polyak's momentum algorithm fails to converge in some specific cases, for instance:

$$\nabla G(x) = \begin{cases} 25x \text{ if } x < 1 \\ x + 24 \text{ if } 1 \leq x < 2 \\ 25x - 24 \text{ if } x \geq 2 \end{cases}$$

- In that case, G is μ strongly convex and L-smooth with $(\mu, L) = (1, 25)$. However, iterations given by Polyak's algorithm cycles.

Nesterov Accelerated Gradient Descent

Nesterov Method

- **Input**: starting point $\boldsymbol{w}^{[0]} (= \boldsymbol{w}^{[-1/2]})$, step size $\alpha^{[k]} > 0$, momentum $\beta^{[k]} \in [0, 1]$
- While *not converge* do
  - $\boldsymbol{w}^{[k+1/2]} = \boldsymbol{w}^{[k]} - \alpha^{[k]} \nabla G(\boldsymbol{w}^{[k]})$
  - $\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k+1/2]} + \beta^{[k]} (\boldsymbol{w}^{[k+1/2]} - \boldsymbol{w}^{[k-1/2]})$
  - $k \leftarrow k + 1$
- **Return** last $\boldsymbol{w}^{[k+1]}$.

- Subtle difference with momentum method.
- Not much freedom for $\beta^{[k]}$:
  - Nesterov original choice:

$$\beta^{[k]} = \frac{t^{[k]} - 1}{t^{[k+1]}} \text{ with } \begin{cases} t^{[1]} = 1 \\ t^{[k+1]} = \frac{1 + \sqrt{1 + 4(t^{[k]})^2}}{2} \end{cases}$$

  - Simpler equivalent choice $\beta^{[k]} = \dfrac{k}{k+3}$
  - Other choice possible for μ convex function.
- Proximal version possible.

Convergence Rate of Nesterov Accelerated Gradient

Theorem for L-smooth function

- Assume that G is a L-smooth, convex function whose minimum is reached at $\boldsymbol{w}^\star$. Then, if $\beta^{[k]} = k/(k+3)$,

$$G(\boldsymbol{w}^{[k]}) - G(\boldsymbol{w}^\star) \leq \frac{2\|\boldsymbol{w}^{[0]} - \boldsymbol{w}^\star\|_2^2}{\alpha(k+1)^2}.$$

Theorem for μ strongly convex function

- Assume that G is a L-smooth, μ strongly convex function whose minimum is reached at $\boldsymbol{w}^\star$. Then, if $\beta^{[k]} = \frac{1-\sqrt{\mu/L}}{1+\sqrt{\mu/L}}$,

$$G(\boldsymbol{w}^{[k]}) - G(\boldsymbol{w}^\star) \leq \frac{\|\boldsymbol{w}^{[0]} - \boldsymbol{w}^\star\|_2^2}{\alpha}\Big(1 - \sqrt{\frac{\mu}{L}}\Big)^k.$$

- Faster rates than respectively $O(1/k)$ and $O((1-\mu/L)^k)$.
- Theorems holds for the proximal algorithm with $G = F + R$ with F smooth and R simple.

Optimal Bounds

General First Order Method

- Any iterative method that generates a sequence $\{\boldsymbol{w}^{[k]}\}$ s.t.
$$\boldsymbol{w}^{[k]} \in \boldsymbol{w}^{[0]} + \mathrm{Span}(\nabla f(\boldsymbol{w}^{[0]}), \ldots, \nabla f(\boldsymbol{w}^{[k-1]})).$$

Lower Bounds

- For any $\boldsymbol{w}^{[0]} \in \mathbb{R}^d$ and any k satisfying $1 \leq k \leq (d-1)/2$, there exists a L-smooth convex function f such that for any general first order method
$$G(\boldsymbol{w}^{[k]}) - G(\boldsymbol{w}^\star) \geq \frac{3L\|\boldsymbol{w}^{[0]} - \boldsymbol{w}^\star\|_2^2}{32(k+1)^2}.$$
- For any $\boldsymbol{w}^{[0]} \in \mathbb{R}^d$ and any $k \leq (d-1)/2$, there exists a L-smooth, μ strongly convex function f such that for any general first order method
$$G(\boldsymbol{w}^{[k]}) - G(\boldsymbol{w}^\star) \geq \frac{\mu}{2}\Big(\frac{1-\sqrt{\mu/L}}{1+\sqrt{\mu/L}}\Big)^{2k}\|\boldsymbol{w}^{[0]} - \boldsymbol{w}^\star\|_2^2.$$
- Accelerated rates: best possible without further assumptions.

Outline

Goodness-of-Fit and Stochastic Gradient Heuristic

Goodness-of-Fit Optimization

- Minimizer of $G = F + R$ with
  - F smooth and R simple (as before).
  - Empirical average structure for F:

$$F(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} F_i(\boldsymbol{w}) = \mathbb{E}_n[F_i(\boldsymbol{w})]$$

    where each F_i is smooth.

- Cost of evaluating ∇F is proportional to the dataset cardinality n (and the dimension d).
- If the dataset is large, this can be an issue.
- **Stochastic Gradient Heuristic**: replace $\nabla F(\boldsymbol{w}) = \mathbb{E}_n[\nabla F_i(\boldsymbol{w})]$ by a Monte Carlo approximation.

Stochastic Gradient

- Empirical average structure

$$\nabla F(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla F_i(\boldsymbol{w}) = \mathbb{E}_n[\nabla F_i(\boldsymbol{w})]$$

- Monte Carlo approximation of an average:
  - draw uniformly $m \ll n$ indices $i_1, \ldots, i_m$
  - Replace $\nabla F(\boldsymbol{w})$ by

$$\widehat{\nabla} F(\boldsymbol{w}) = \frac{1}{m}\sum_{j=1}^{m} \nabla F_{i_j}(\boldsymbol{w})$$

- **Prop:** $\mathbb{E}\left[\widehat{\nabla} F(\boldsymbol{w})\right] = \nabla F(\boldsymbol{w})$

- Extreme case $m = 1$: unbiased but quite noisy estimate of the true gradient.
- Evaluation cost of $\widehat{\nabla} F(\boldsymbol{w})$ independent of n.
- **Question:** Can we use this approximation of the true gradient in a descent algorithm?

Stochastic Gradient Descent Algorithm

Stochastic Gradient Descent Algorithm

- Start from a point $\boldsymbol{w}^{[0]}$ and let $k = 0$.
- Repeat until convergence:
  - Draw m indices $1 \leq i_j \leq n$ uniformly at random
  - Set $\widehat{\nabla} F(\boldsymbol{w}^{[k]}) = \frac{1}{m} \sum_{j=1}^{m} \nabla F_{i_j}(\boldsymbol{w}^{[k]})$
  - $\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k]} - \alpha^{[k]} \left(\widehat{\nabla} F(\boldsymbol{w}^{[k]}) + \nabla R(\boldsymbol{w}^{[k]}) \right)$
  - $k \to k + 1$

  with $\alpha^{[k]}$ a sequence of (small) steps.

- Each iteration has complexity $O(md)$ instead of $O(nd)$ for full gradient methods
- m is called the batch size.
- **Rk:** no gain when using a batch size larger than 1, except if the sum can be parallelized.
- Proximal variant if R is simple instead of smooth.

Convergence Rate of SGD

- Theoretical analysis requires some modifications:
  - at each step $\boldsymbol{w}^{[k+1]}$ should be projected intro a ball $B(c, R)$ with $R > 0$.
  - G should be convex such that $\sup_i |\nabla F_i(\boldsymbol{w}) + \nabla R(\boldsymbol{w})| \leq b,\ \forall \boldsymbol{w} \in B(c, r)$
  - **Polyak-Ruppert** averaging: use SGD iterates w^t but return an average.

SGD Rate

- With $\alpha^{[k]} = 2R/(b\sqrt{k})$

$$\mathbb{E}\left[G\Big(\frac{1}{k}\sum_{j=1}^{k}\boldsymbol{w}^{[j]}\Big)\right] - G(\boldsymbol{w}^\star) \leq \frac{3rb}{\sqrt{k}}$$

- If G is μ-strictly convex then with $\alpha^{[k]} = 2/(\mu(k+1))$,

$$\mathbb{E}\left[G\Big(\frac{2}{k(k+1)}\sum_{j=1}^{k} j\boldsymbol{w}^{[j]}\Big)\right] - G(\boldsymbol{w}^\star) \leq \frac{2b^2}{\mu(k+1)}.$$

- Without averaging, logarithmic loss.
- Same rate for proximal algorithm if F is at least L-smooth and R simple.

Convergence Rate of SGD

Simplified Result for the (Averaged) SGD

- If G is convex and gradients are bounded ($\|\nabla F_i(\boldsymbol{w})\|_2 \leq b$) then the convergence rate is
$$O\Big(\frac{1}{\sqrt{k}}\Big) \quad \text{with} \quad \alpha^{[k]} = O\Big(\frac{1}{\sqrt{k}}\Big)$$
- G is μ-strongly convex, the rate is
$$O\Big(\frac{1}{\mu k}\Big) \quad \text{with} \quad \alpha^{[k]} = O\Big(\frac{1}{\mu k}\Big)$$

Comparison with GD

- Much slower rate:
  - for L smooth function: $O(1/\sqrt{k})$ vs $O(1/k)$
  - For μ convex function: $O(1/k)$ vs $O((1-\mu/L)^k)$
  - Hidden factor σ in SGD measuring the variance of $\hat{\nabla} F$ which decreases when m increases.
- Much cheaper cost per iteration: $O(md)$ vs $O(nd)$

SGD with Fixed Step Size

Stochastic Gradient Descent Algorithm

- Start from a point $\boldsymbol{w}^{[0]}$ and let $k = 0$.
- Repeat until convergence:
  - Draw m indices $1 \leq i_j \leq n$ uniformly at random
  - Set $\widehat{\nabla} F(\boldsymbol{w}^{[k]}) = \frac{1}{m} \sum_{k=1}^{m} \nabla F_{i_j}(\boldsymbol{w}^{[k]})$
  - $\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k]} - \alpha \left(\widehat{\nabla} F(\boldsymbol{w}^{[k]}) + \nabla R(\boldsymbol{w}^{[k]}) \right)$
  - $k \to k + 1$

  with α constant.

- No convergence but for μ-convex function:

$$\mathbb{E}\left[\left\| \boldsymbol{w}^{[k]} - \boldsymbol{w}^{\star} \right\|^2 \right] \to O(\alpha/\mu)$$

- SGD behavior in practice: very fast initially but much slower afterward.
- Justify the use of decreasing step size when the error decay becomes slow.

Stabilization

- Difference due to the variance of the stochastic gradient.

Variance reduction of the gradient

- In the iterations of SGD, replace $\nabla F_{i_k}(\boldsymbol{w}^{[k-1]})$ by
$$\nabla F_{i_k}(\boldsymbol{w}^{[k-1]}) - \nabla F_{i_k}(\tilde{\boldsymbol{w}}) + \nabla F(\tilde{\boldsymbol{w}})$$
where $\tilde{\boldsymbol{w}}$ is an **old** value of the iterate, namely use
$$\boldsymbol{w}^{[k]} \leftarrow \boldsymbol{w}^{[k-1]} - \alpha^{[k]}\big((\nabla F_{i_k}(\boldsymbol{w}^{[k-1]}) - \nabla F_{i_k}(\tilde{\boldsymbol{w}})) + \nabla F(\tilde{\boldsymbol{w}})\big)$$
- Several instantiations
  - SVRG where $\tilde{\boldsymbol{w}}$ and $\nabla F(\tilde{\boldsymbol{w}})$ is computed every few step.
  - SAGA where, instead of $\nabla F_i(\tilde{\boldsymbol{w}})$, one use $F_i(\boldsymbol{w}^{[k_i]})$, where k_i is the last time the ith coordinate has been updated, even to compute the equivalent of $\nabla F(\tilde{\boldsymbol{w}})$.

- Lead to faster rate but requires more memory ($3d$ for SVRG and nd for SAGA).

Outline

Adaptive Step Size

Stochastic Gradient Descent Algorithm

- Start from a point $\boldsymbol{w}^{(0)}$ and let $k = 0$.
- Repeat until convergence:
  - Draw m indices $1 \leq i_j \leq n$ uniformly at random
  - Set $\widehat{\nabla} F(\boldsymbol{w}^{[k]}) = \frac{1}{m} \sum_{k=1}^{m} \nabla F_{i_j}(\boldsymbol{w}^{[k]})$
  - $\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k]} - \alpha^{[k]} \left(\widehat{\nabla} F(\boldsymbol{w}^{[k]}) + \nabla R(\boldsymbol{w}^{[k]}) \right)$
  - $k \to k + 1$

  with $\alpha^{[k]}$ a sequence of (small) steps.

- Step sizes $\alpha^{[k]}$ are crucial for convergence.
- How to choose them? Decaying? Fixed? **Adaptative?**
- Several algorithms available with no clear winner.

  Determining a good learning rate becomes more of an art than science for many problems.

  M.D. Zeiler

Gradient Descent, Step Size and Reparameterization

Gradient Descent and Reparameterization

- Gradient Descent: $\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k]} - \alpha \nabla G(\boldsymbol{w}^{[k]})$.
- Reparameterization: $\boldsymbol{w} = W\mathbf{z}$.
- Gradient Descent in $\mathbf{z}$: $\mathbf{z}^{[k+1]} = \mathbf{z}^{[k]} - \alpha W^\top \nabla G(W\mathbf{z}^{[k]})$.
- Implied Gradient Descent in $\boldsymbol{w}$: $\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k]} - \alpha WW^\top \nabla G(\boldsymbol{w}^{[k]})$.
- **Different dynamics if $WW^\top \neq \mathrm{Id}$!**

Gradient Descent and Norm

- Descent lemma upperbound: $f(\boldsymbol{w}') \leq f(\boldsymbol{w}) + \nabla G(\boldsymbol{w})^\top (\boldsymbol{w}' - \boldsymbol{w}) + \frac{1}{2\alpha}\|\boldsymbol{w}' - \boldsymbol{w}\|^2$.
- Uses the classical euclidean norm.
- Using $f(\boldsymbol{w}') \leq f(\boldsymbol{w}) + \nabla G(\boldsymbol{w})^\top (\boldsymbol{w}' - \boldsymbol{w}) + \frac{1}{2\alpha}\|\boldsymbol{w}' - \boldsymbol{w}\|_\Sigma^2$ leads to $\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k]} - \alpha \Sigma^{-1} \nabla G(\boldsymbol{w}^{[k]})$.
- **Different dynamics if $\Sigma \neq \mathrm{Id}$!**

Gradient Descent

Modified Gradient Descent Algorithm

- $\boldsymbol{w}^{[k+1]} = \boldsymbol{w}^{[k]} - \alpha P \nabla G(\boldsymbol{w}^{[k]})$ where P can be interpreted as $WW^\top$ or Σ of the previous slide.
- Convergence holds if corresponding descent lemma holds.
- P can even changes from one iteration to another.

Choices for P

- Newton method:
  - $P = \left(\nabla^2 G\right)^{-1}$
  - Hard to compute and no descent lemma property.
- Diagonal approximation:
  - $P = D$
  - Easy computation but no descent lemma in general.
  - Several strategies for the choice of D...

- Coordinate Descent could be reframed in this framework!

Self Normalized Gradient Descent Algoritm

$$\boldsymbol{w}_j^{[k+1]} = \boldsymbol{w}_j^{[k]} + \frac{\alpha_b^{[k]}}{d_j^{[k]}} m_j^{[k]}$$

Self Normalized Gradient Descent

- Gradient: $m^{[}k] \simeq \nabla G(\boldsymbol{w}^{[}k])$
- Renormalization factor: $d_j^{[k]} \simeq |\nabla G(\boldsymbol{w}^{[k]})|_j$
- Base stepsize: $\alpha_b^{[k]}$ could be constant or decaying...

- Makes the algorithm invariant to a diagonal rescaling.
- Amounts to use

$$\|\boldsymbol{w}' - \boldsymbol{w}\|_\Sigma^2 \simeq \sum_j \frac{|\boldsymbol{w}'_j - \boldsymbol{w}_j|^2}{|\nabla G(\boldsymbol{w})_i|^2}$$

ADAptive GRADient

$$\boldsymbol{w}_j^{[k+1]} = \boldsymbol{w}_j^{[k]} + \frac{\alpha_b^{[k]}}{d_j^{[k]}} m_j^{[k]}$$

ADAptive GRADient update

- $m^{[k]} = \nabla G(\boldsymbol{w}^{[k]})$
- $d_j^{[k]} = \sqrt{\sum_{k'=1}^{k} (\nabla G(\boldsymbol{w}^{[k']}))_j^2}$
- $\alpha_b^{[k]} = \alpha$

- Step sizes grow as the inverse of the gradient magnitude.
- Accumulation of the gradients acts as a decreasing learning rate.
- Sensitive to initial condition and may require large initial parameter or restarts.

ADAptive GRADient

$$\boldsymbol{w}_j^{[k+1]} = \boldsymbol{w}_j^{[k]} + \frac{\alpha_b^{[k]}}{d_j^{[k]}} m_j^{[k]}$$

ADAptive GRADient update reformulation

- $m^{[k]} = \nabla G(\boldsymbol{w}^{[k]})$
- $d_j^{[k]} = \sqrt{v_j^{[k]}}$ with $v_j^{[k]} = \frac{1}{k} \sum_{k'=1}^{k} (\nabla G(\boldsymbol{w}^{[k']}))_j^2$
- $\alpha_b^{[k]} = \alpha / \sqrt{k}$

- Explicit averaging and decay.

RMSprop

$$\boldsymbol{w}_j^{[k+1]} = \boldsymbol{w}_j^{[k]} + \frac{\alpha_b^{[k]}}{d_j^{[k]}} m_j^{[k]}$$

RMSprop update

- $m^{[k]} = \nabla G(\boldsymbol{w}^{[k]})$
- $d_j^{[k]} = \sqrt{v_j^{[k]}}$ with $v_j^{[k]} = \rho v_j^{[k-1]} + (1-\rho)(\nabla G(\boldsymbol{w}^{[k]})_j^2$
- $\alpha_b^{[k]} = \alpha$

- Unpublished method!
- Exponential average instead of classical one.
- No step size decay.
- Often very efficient (especially in a non stationary setting).

ADAM: ADAptive Moment estimation

$$\boldsymbol{w}_j^{[k+1]} = \boldsymbol{w}_j^{[k]} + \frac{\alpha_b^{[k]}}{d_j^{[k]}} m_j^{[k]}$$

ADAM

- Exponential averaging estimation: let $\tilde{m}_0 = \tilde{v}_0 = 0$,
$\tilde{m}^{[k]} = \beta_m \tilde{m}^{[k-1]} + (1-\beta_m)\nabla G(\boldsymbol{w}^{[k]})$ and $\tilde{v}^{[k]} = \beta_v \tilde{v}^{[k-1]} + (1-\beta_v)(\nabla G(\boldsymbol{w}^{[k]}))^2$
- Bias correction: $m^{[k]} = \dfrac{\tilde{m}^{[k]}}{1-\beta_m^k}$ and $v^{[k]} = \dfrac{\tilde{v}^{[k]}}{1-\beta_v^k}$
- $\alpha_b^{[k]} = \alpha$.

- Use a smoothed estimation of the first two moments of the gradients.
- Often efficient.
- ADAMax: $v^{[k]} = \tilde{v}^{[k]} = \max(\beta_v v^{[k-1]}, |\nabla G(\boldsymbol{w}^{[k]})|)$.

AdaDelta

$$\boldsymbol{w}_j^{[k+1]} = \boldsymbol{w}_j^{[k]} + \frac{\alpha_b^{[k]}}{d_j^{[k]}} m_j^{[k]}$$

AdaDelta

- $m^{[k]} = \nabla G(\boldsymbol{w}^{[k]})$
- $d_j^{[k]} = \frac{\sqrt{v_j^{[k]}}}{\sqrt{\tilde{v}^{[k-1]}}}$ with $v_j^{[k]} = \rho v_j^{[k-1]} + (1-\rho)(\nabla G(\boldsymbol{w}^{[k]})_j^2$ and $\tilde{v}_j^{[k]} = \rho \tilde{v}_j^{[k-1]} + (1-\rho)\left(\frac{\alpha_j^{[}[k]]}{d_k^{[k]}} \nabla G(\boldsymbol{w}^{[k]})_j\right)^2$
- $\alpha_b^{[k]} = 1$

- Only one parameter!
- Based on a dimensional analysis... with a strange looking renormalization!

AdaDelta

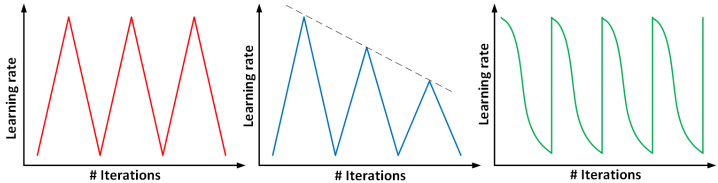
$$\boldsymbol{w}_j^{[k+1]} = \boldsymbol{w}_j^{[k]} + \frac{\alpha_b^{[k]}}{d_j^{[k]}} m_j^{[k]}$$

AdaDelta

- $m^{[k]} = \nabla G(\boldsymbol{w}^{[k]})$ and $d_j^{[k]} = 1$
- $\alpha_j^{[0]} = \alpha$ and $\alpha_j^{[k]} = \dfrac{\bar{\alpha}_j^{[k-1]}}{\sqrt{\rho + (1-\rho)(\nabla G(\boldsymbol{w}^{[k]})_j^2 / v_j^{[k-1]})}}$ with
  $v_j^{[k]} = \rho v_j^{[k-1]} + (1-\rho)(\nabla G(\boldsymbol{w}^{[k]})_j^2$, and
  $v_j^{[k]} = \rho \tilde{v}_j^{[k-1]} + (1-\rho)\left(\alpha_j^{[}[k]]\right)^2 \left(\nabla G(\boldsymbol{w}^{[k]}\right)_j^2$ and $\bar{\alpha}_j^{[k-1]} = \sqrt{\tilde{v}_j^{[k]} / v^{[k]}{}_j}$

- Better insight (and more flexible initial step)!
- Quite different from the previous schemes.

Latest Trends



Cyclic Learning Rate

- Cycle between small and large learning rate with stopping when the learning rate is small.
- One cycle strategy seems possible!

- Plain SGD with a constant step is also making a comeback!

Outline

(S)GD and Non Convex Function

- No global convergence result for non convex functions optimization (NP Hard problem)
- Nevertheless (Stochastic) Gradient Descent algorithm can be used.

Theoretical results?

- Typical weak convergence results:
  - convergence to a stationary point
  - convergence to a local minimum...
- Often in specific cases.

- In practice: convergence depends a lot on the step size α choice strategy.
- In general, more exploration than in the convex setting.

Outline

References

T. Hastie, R. Tibshirani, and J. Friedman.
The Elements of Statistical Learning.
Springer Series in Statistics, 2009

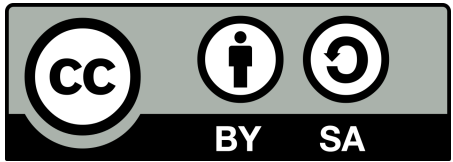
M. Mohri, A. Rostamizadeh, and A. Talwalkar.
Foundations of Machine Learning (2nd ed.)
MIT Press, 2018

A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd ed.)
O'Reilly, 2022

Ch. Giraud.
Introduction to High-Dimensional Statistics.
CRC Press, 2014

S. Bubeck.
Convex Optimization: Algorithms and Complexity.
Now Publisher, 2015

Licence and Contributors



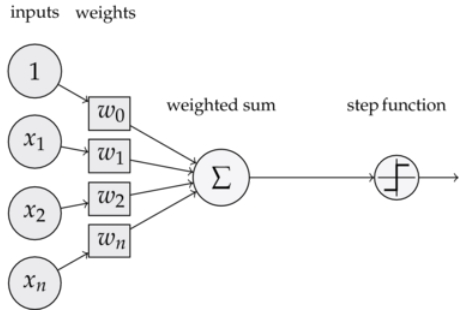
Contributors

- Main contributor: E. Le Pennec
- Contributors: S. Boucheron, A. Dieuleveut, A.K. Fermin, S. Gadat, S. Gaiffas, A. Guilloux, Ch. Keribin, E. Matzner, M. Sangnier, E. Scornet.

Outline

Outline

Perceptron

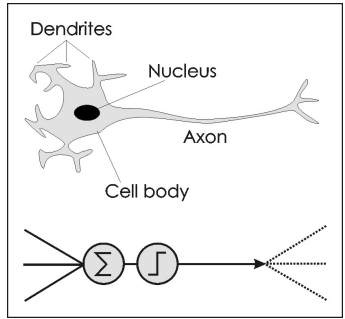


Perceptron (Rosenblatt 1957)

- Inspired from biology.
- Very simple (linear) model!
- Physical implementation and proof of concept.

Source: Tikz

Perceptron

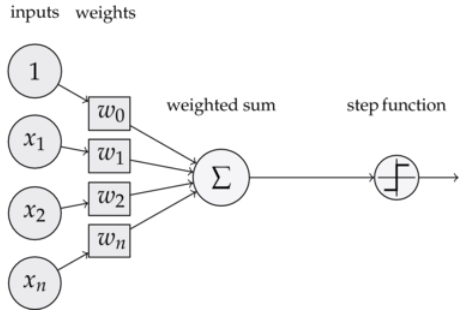


Perceptron (Rosenblatt 1957)

- Inspired from biology.
- Very simple (linear) model!
- Physical implementation and proof of concept.

Source: Tikz

Perceptron

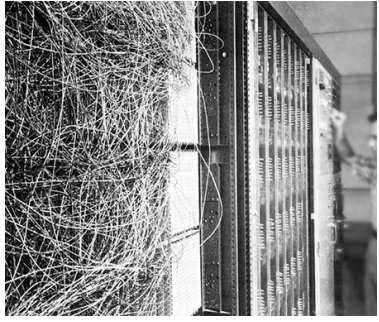


Perceptron (Rosenblatt 1957)

- Inspired from biology.
- Very simple (linear) model!
- Physical implementation and proof of concept.

Source: Tikz

Perceptron

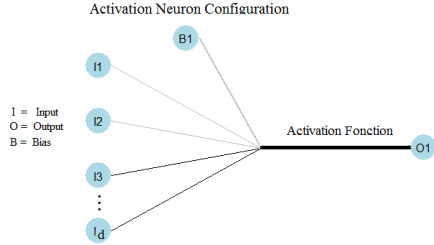


Perceptron (Rosenblatt 1957)

- Inspired from biology.
- Very simple (linear) model!
- Physical implementation and proof of concept.

Source: Avin Calspan Advanced Technology Center

Artificial Neuron and Logistic Regression



Artificial neuron

- Structure:
  - Mix inputs with a **weighted sum**,
  - Apply a (non linear) **activation function** to this sum,
  - Possibly threshold the result to make a decision.
- Weights learned by minimizing a loss function.

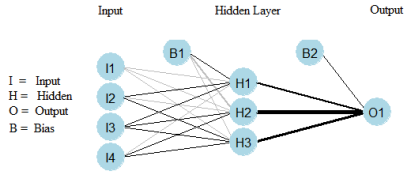
Logistic unit

- Structure:
  - Mix inputs with a **weighted sum**,
  - Apply the **logistic function** $\sigma(t) = e^t/(1+e^t)$,
  - Threshold at $1/2$ to make a decision!
- Logistic weights learned by minimizing the -log-likelihood.

- Equivalent to linear regression when using a linear activation function!

Source: Unknown

Multilayer Perceptron



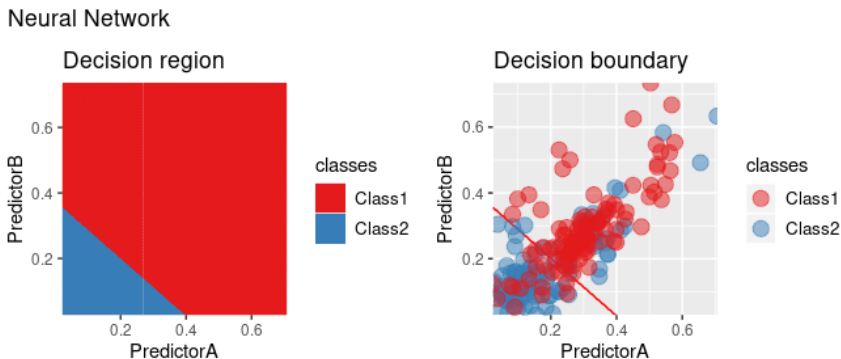
MLP (Rumelhart, McClelland, Hinton - 1986)

- Multilayer Perceptron: cascade of layers of artificial neuron units.
- Optimization through a gradient descent algorithm with a clever implementation (**Backprop**).

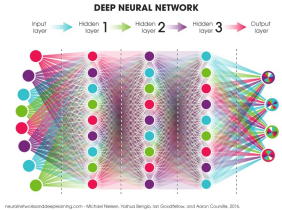
- Construction of a function by composing simple units.
- MLP corresponds to a specific direct acyclic graph structure.
- Non convex optimization problem!

Source: Unknown

Multilayer Perceptron



Deep Neural Network



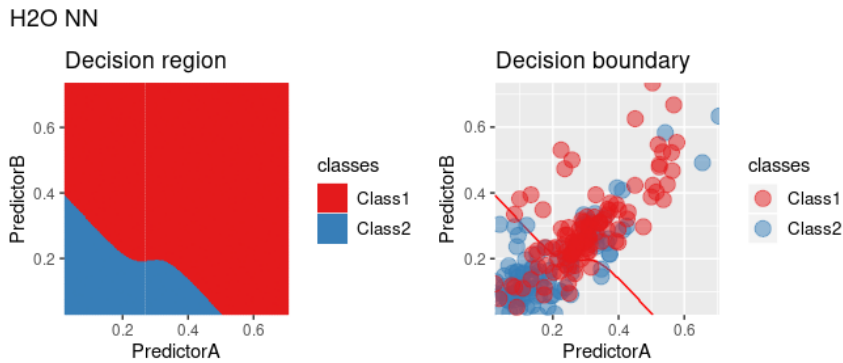
Deep Neural Network structure

- Deep cascade of layers!

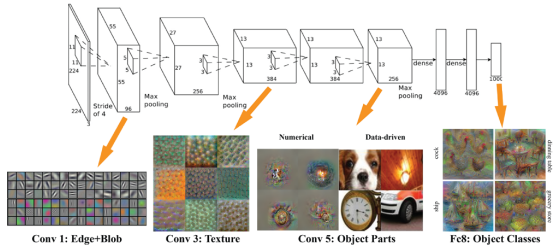
- No conceptual novelty...
- But a **lot of tricks** allowing to obtain a good solution: clever initialization, better activation function, weight regularization, accelerated stochastic gradient descent, early stopping...
- Use of GPU and a lot of data...
- Very impressive results!

Source: Nielsen, Bengio, Goodfellow and Courville

Deep Neural Network



Deep Learning



Family of Machine Learning algorithm combining:

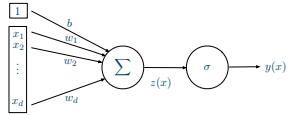
- a (deep) multilayered structure,
- a clever optimization including initialization and regularization.

- Examples: Deep NN, AutoEncoder, Recursive NN, GAN, Transformer...
- Interpretation as a **Representation Learning**.
- **Transfer learning:** use as initialization a pretrained net.
- Very efficient and still evolving!

Source: J. Hays

Outline

Logistic Regression as NN



Binary Logistic Regression

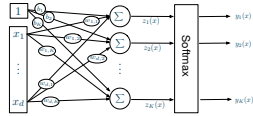
- Conditional probability model:
$$\mathbb{P}(Y = 1|\underline{X}) = \sigma(\boldsymbol{w}^\top \underline{X} + b) = \frac{e^{\boldsymbol{w}^\top \underline{X}+b}}{1 + e^{\boldsymbol{w}^\top \underline{X}+b}}$$
- Model **weights** $\boldsymbol{w} \in \mathbb{R}^d$ and intercept (or **bias**) $b \in \mathbb{R}$.

Artificial Neuron or Unit

- $\underline{X}$ is the input (single feature vector).
- $z(\underline{X}) = \boldsymbol{w}^\top \underline{X} + b$ is called **pre-activation**.
- $y(\underline{X}) = \sigma(z(\underline{X}))$ is the output (in $[0, 1]$ in this case)
- $\boldsymbol{w}$ is **weights** and b is **bias**
- σ is an **activation** function

Source: S. Gaiffas

Softmax Regression



Multinomial Logistic Model

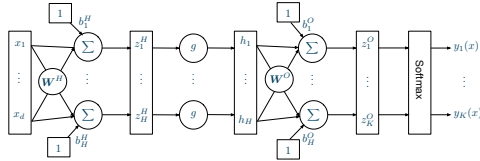
- Cond. prob. model: $\mathbb{P}(Y = k|\underline{X}) = \frac{e^{\boldsymbol{w}_k{}^\top \underline{X} + b_k}}{\sum_{k'=1}^{K} e^{\boldsymbol{w}_{k'}{}^\top \underline{X} + b_{k'}}}$
- Model weights: k vector $\boldsymbol{w}_k \in \mathbb{R}^d$ or equivalently a model weight matrix $\boldsymbol{W}$ with $\boldsymbol{W}_{\bullet,k} = \boldsymbol{w}_k$

Softmax Regression (Softmax Layer)

- $z_k(\underline{X}) = \boldsymbol{W}_{\bullet,k}{}^\top \underline{X} + b_k$ pre-activations or *logits*
- $y_k(\underline{X}) = \frac{e^{z_k(\underline{X})}}{\sum_{k'=1}^{K} e^{z_{k'}(\underline{X})}}$ coming out of the softmax activation
- Matricially:

$$y(\underline{X}) = \text{softmax}(z(\underline{X})) = \text{softmax}(\boldsymbol{W}^\top \underline{X} + b)$$

One-Hidden Layer Neural Network



One-Hidden Layer Neural Network - Neural Representation

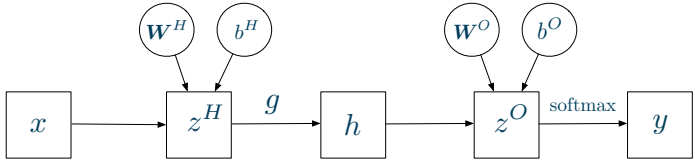
- Recursive definition:

$$\begin{aligned} y(\underline{X}) &= \text{softmax}(z^{(O)}) = \text{softmax}(\boldsymbol{W}^{(O)\top} h + b^{(O)}) \\ &= \text{softmax}(\boldsymbol{W}^{(O)\top} g(z^{(H)}) + b^{(O)}) \\ &= \text{softmax}(\boldsymbol{W}^{(O)\top} g(\boldsymbol{W}^{(H)\top} x + b^{(H)}) + b^{(O)}) \end{aligned}$$

- g is an **activation function** applied entrywise on $z_1^H, \ldots, z_H^H$

- This neural network has a width-H hidden layer

One-Hidden Layer Neural Network



One-Hidden Layer Neural Network - Vector Representation

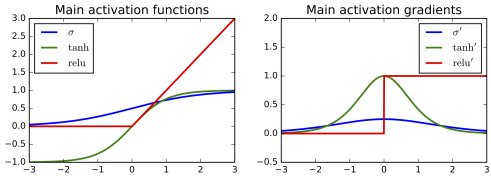
- Vector representation:

$$\begin{aligned} y(\underline{X}) &= \text{softmax}(z^{(O)}) = \text{softmax}(\boldsymbol{W}^{(O)^\top} h + b^{(O)}) \\ &= \text{softmax}(\boldsymbol{W}^{(O)^\top} g(z^{(H)}) + b^{(O)}) \\ &= \text{softmax}(\boldsymbol{W}^{(O)^\top} g(\boldsymbol{W}^{(H)^\top} \underline{X} + b^{(H)}) + b^{(O)}) \end{aligned}$$

- g is an **activation function** applied entrywise

- Equivalent representation focusing on layers.

Activation Functions



Entrywise Activation Functions

- Most classical: **sigmoid**, **tanh** and **ReLU** (rectified linear unit)

$$\sigma(z) = \frac{1}{1+e^{-z}}, \quad \tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}, \quad \mathrm{ReLU}(z) = \max(0, z)$$

- Their **derivatives** are given by

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$
$$\tanh(z) = 1 - \tanh(z)^2,$$
$$\mathrm{ReLU}'(z) = \mathbf{1}_{z>0}$$

- Note that $\tanh(z) = 2\sigma(2z) - 1$

Activation Functions

ReLU Activation

- ReLU: $z \mapsto \max(0, z)$
- Often the **best choice** for deep neural networks
- Easier to optimize, since their behavior is closer to linear
- Its gradient is not defined at $z = 0$: not a problem since, during training, it's unlikely that any input equals 0

- In contrast to ReLu, *Sigmoid* activations σ and tanh (not recommended anymore) saturate for large positive (or large negative values).

Identity Activation

- Hidden layer:
  - Amount to use $h = g(\boldsymbol{V}\boldsymbol{U}^\top z + b)$ instead of $h = g(\boldsymbol{W}^\top z + b)$
  - $\boldsymbol{W}$ is factorized as $\boldsymbol{U}\boldsymbol{V}^\top$
  - Reduces the dimension of the model
- Output layer: linear model on the last layer

Softmax

- Softmax is not an element-wise activation:

$$\text{softmax}(z) = \frac{1}{\sum_{k=1}^{K} e^{z_k}} \begin{bmatrix} e^{z_1} \\ \vdots \\ e^{z_K} \end{bmatrix}$$

- Gradient:

$$\frac{\text{softmax}(z)_k}{\partial x_{k'}} = \begin{cases} \text{softmax}(z)_k \times (1 - \text{softmax}(z)_k) & \text{if } k = k' \\ -\text{softmax}(z)_k \times \text{softmax}(z)_{k'} & \text{otherwise} \end{cases}$$

- Maps $z \in \mathbb{R}^K$ to the space of vector in $[0, 1]^K$ with entries that sum to 1 (**probabilities**)
- The inputs of the softmax are called the **logits** in deep learning

Feed-Forward Neural Network

- Heuristic: It's easy to construct a complex function by composing simple elements in some order.
- **Layerwise structure**:

$$
\begin{aligned}
h^{(1)} &= g^{(1)}(\boldsymbol{W}^{(1)\top} x + b^{(1)}) \\
h^{(2)} &= g^{(2)}(\boldsymbol{W}^{(2)\top} h^{(1)} + b^{(2)}) \\
&\vdots \\
h^{(L)} &= g^{(L)}(\boldsymbol{W}^{(L)\top} h^{(L-1)} + b^{(L)}) \\
y &= \text{softmax}(\boldsymbol{W}^{(O)\top} h^{(L)} + b^{(O)})
\end{aligned}
$$

- Need to choose the width of each layer and the depth of the network.

- The name comes from the fact that information flows through the layers.
- For regression, replace the softmax by identity.

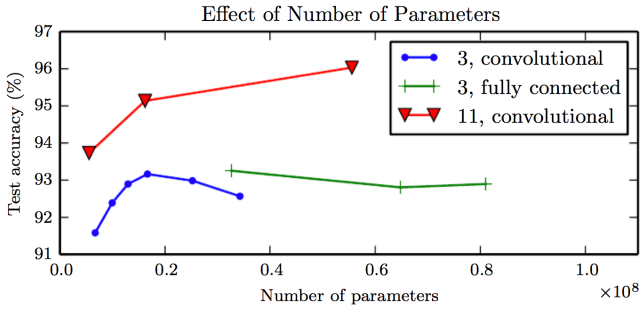
Universal Approximation Theorem

Universal Approximation Theorem (Hornik, 1991)

- A **single hidden layer neural network** with a linear output unit can **approximate** any continuous function **arbitrarily well**, given enough hidden units

- Valid for most activation functions.
- No bounds on the number of required units... (Asymptotic flavor)
- A single hidden layer is sufficient but more may require less units?

Network Architecture



- What is best?
  - One hidden layer with large width?
  - . . . or several layers with smaller width ?

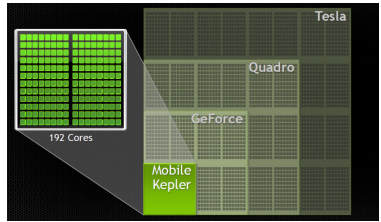
A Recipe

- For the same number of parameters, several layers with a smaller width lead to better generalization. (More abstract layers)

Source: Goodfellow, Bengio and Courville

Outline

How to Train a DNN



- Use a differentiable loss.

Loss Minimization

- Compute the prediction and the loss **(forward-propagation)**
- Compute the gradients **(back-propagation)**
- Use a Stochastic Gradient Descent algorithm

- In practice, use
  - A nice open source library (`tensorflow`, `keras`, `pyTorch`)
  - A GPU... since training a DNN can benefit from massive parallel computations

Neural Network Optimization Formulation

Feed-Forward Neural Network

- Network structure:

$$h^{(1)} = g^{(1)}(\boldsymbol{W}^{(1)\top} x + b^{(1)}) \qquad h^{(2)} = g^{(2)}(\boldsymbol{W}^{(2)\top} h^{(1)} + b^{(2)})$$

$$\dots \qquad h^{(L)} = g^{(L)}(\boldsymbol{W}^{(L)\top} h^{(L-1)} + b^{(L)})$$

$$f(x, \boldsymbol{w}) = \text{softmax}(\boldsymbol{W}^{(O)\top} h^{(L)} + b^{(O)})$$

with $\boldsymbol{w} = (\boldsymbol{W}^{(1)}, b^{(1)}, \dots, \boldsymbol{W}^{(L)}, b^{(L)}, \boldsymbol{W}^{(o)}, b^{(o)})$

Optimization Formulation

- Minimization of

$$G(\boldsymbol{w}) = \underbrace{\frac{1}{n}\sum_{i=1}^{n} \ell(Y_i, f(\underline{X}_i, \boldsymbol{w}))}_{F(\boldsymbol{w})} + \underbrace{\text{pen}(\boldsymbol{w})}_{R(\boldsymbol{w})}$$

with ℓ a loss function and pen a penalization.

- Most classical choices:
  - ℓ is the cross entropy (-log-likelihood)
  - pen is a ridge penalty on the $\boldsymbol{W}$.

Optimization Algorithm

Optimization Formulation

- Minimization of

$$G(\boldsymbol{w}) = \underbrace{\frac{1}{n}\sum_{i=1}^{n} \ell(Y_i, f(\underline{X}_i, \boldsymbol{w}))}_{F(\boldsymbol{w})} + \underbrace{\text{pen}(\boldsymbol{w})}_{R(\boldsymbol{w})}$$

with ℓ a loss function and pen a penalization.

Optimization Algorithm

- Non convex optimization problem.
- Stochastic Gradient Descent algorithm.
- Key issue: computing

$$\nabla F_i(\boldsymbol{w}) = \nabla \ell(Y_i, f(\underline{X}_i, \boldsymbol{w}))$$

Back-Propagation

- Clever algorithm to compute $\nabla F_i(\boldsymbol{w}) = \nabla \ell(Y_i, f(\underline{X}_i, \boldsymbol{w}))$.

Heuristic

- Key observation: f is obtained through a direct acyclic graph of composition.
- Key lemma (Chain Rule): $\frac{\partial}{\partial w} f(g(x, w)) = \frac{\partial}{\partial g}(g(x, w)) \frac{\partial g}{\partial w}(x, w)$.
- More generally: $\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$
- Back-propagation: use of this chain rule to compute the derivatives in $\nabla F_i(\boldsymbol{w})$ starting from the parameters of the last layer and going backward to the ones of the first one.

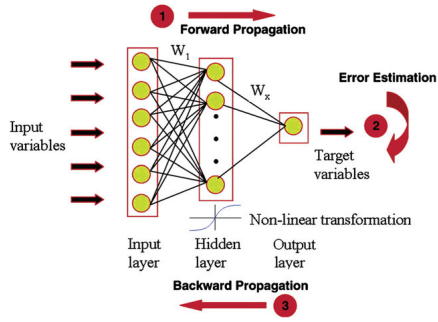
Partial Derivative and Back Propagation

Set of Partial Derivative Linked $\nabla F_i(\boldsymbol{w}) = \nabla \ell(Y_i, f(\underline{X}_i, \boldsymbol{w}))$

- Loss: $\frac{\partial F_i}{\partial f_j}(\boldsymbol{w}) = \frac{\partial \ell}{\partial f_j}(Y_i, f(\underline{X}_i, \boldsymbol{w}))$.
- Output layer:
  - Softmax: $\frac{\partial f_j}{\partial z_k^{(O)}}(\boldsymbol{w})$ is known.
  - Pre-activation at the output layer:
    - Parameters: $\frac{\partial z_j^{(O)}}{\partial \boldsymbol{W}_{j,k}^{(O)}} = h_k^L(\boldsymbol{w})$ and $\frac{\partial z_j^{(O)}}{\partial b^{(O)}} = 1$
    - Previous Layer: $\frac{\partial z_j^{(O)}}{\partial h_k^{(L)}} = \boldsymbol{W}_{j,k}^{(0)}$
- Layer l:
  - Activation: $\frac{\partial h_j^{(l)}}{\partial z_j^{(l)}} = g^{(l)\prime}(z_j^{(l)})$
  - Pre-activation:
    - Parameters: $\frac{\partial z_j^{(l)}}{\partial \boldsymbol{W}_{j,k}^{(l)}} = h_k^{l-1}(\boldsymbol{w})$ and $\frac{\partial z_j^{(l)}}{\partial b^{(l)}} = 1$
    - Previous Layer: $\frac{\partial z_j^{(l)}}{\partial h_k^{(l-1)}} = \boldsymbol{W}_{j,k}^{l)}$

- Recurs. comput. with the chain rule $\Rightarrow$ efficient comput. of $\frac{\partial F_i}{\partial \boldsymbol{W}_{j,k}^{(.)}}$ and $\frac{\partial F_i}{\partial b^{(.)}}$.

Back-Propagation Algorithm



Back-Propagation Algorithm

- This set is sufficient to compute $\nabla F_i(\boldsymbol{w}) = \nabla \ell(Y_i, f(\underline{X}_i, \boldsymbol{w}))$.
- Those partial derivatives can be computed in a backward-pass...
- provided all the intermediate values of h and z have been computed in a forward-pass

Source: Raeesi

SGD and DNN

- Nearly all of deep learning is powered by **SGD**.

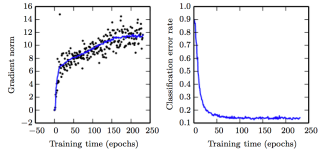
Stochastic Gradient Descent for DNN

- Regular Stochastic Gradient Algorithm with
  - adaptive step size (ADAM, RMSProp...) or fixed size with or without restarts,
  - batch size larger than 1 to use GPU parallelism bit not too large.

Initialization

- SGD does not work without a good initialization scheme.
- Key properties:
  - Break symmetry by random initialization around 0.
  - Order of magnitude at layer l given by $1/\sqrt{H_l + H_{l-1}}$ provided the input is normalized.
  - Examples:
    - Normal with $\mathcal{N}(0, 2/(H_l + H_{l-1}))$
    - Uniform with $\mathcal{U}(-\sqrt{6/(H_l + H_{l-1})}, \sqrt{6/(H_l + H_{l-1})})$

A Challenging Optimization Problem



Challenges

- Non convex optimization in very high dimension.
- SGD can be stuck around critical point and the gradient may not diminishes.
- Several local minima due to invariance by permutation and scaling.
- Some local minima may be bad.

- **Natural concerns but it seems that for sufficiently large network most of the local minima have a similar low cost value.**
- Larger networks may be easier to train than smaller ones...

Source: Goodfellow, Bengio and Courville

Outline

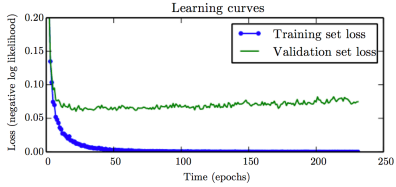
Regularization by Penalization or Projection

Penalization or Projection

- Regularization on weights and not on bias.
- Mostly ℓ^1 or ℓ^2.
- Two possibilities:
  - penalization: $\text{pen}(\boldsymbol{W}) = \lambda \|\boldsymbol{W}\|$
  - projection: $\|\boldsymbol{W}\| \leq r$

- Projection seems to work better as it only affects large weights.
- Avoid **dead** units.

Regularization by Early Stopping



Early Stopping

- SGD optimizes Train error but true goal is Test error.
- **Early Stopping:** stop SGD when test error rises.
- In practice, store the parameters each time the test error improves and use the best set of parameters in the end.

- Most widely used of regularization in DL.
- Number of iterations becomes a method parameter.
- For least-squares regression and gradient descent, early stopping $\simeq$ ridge penalization on the weights

Source: Goodfellow, Bengio and Courville

Regularization by Dropout

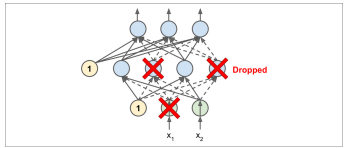
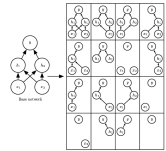


Figure 11-9. Dropout regularization



Dropout

- Idea: At each step, disable temporarily some connections by modifying temporarily the weights.
- Importance sampling idea:

$$\tilde{\boldsymbol{W}}_{j,k}^{(\cdot)} = \begin{cases} 0 & \text{with probability } p \\ \frac{1}{(1-p)} \boldsymbol{W}_{j,k}^{(\cdot)} & \text{with probability } 1-p \end{cases}$$

- The average value of the weights are preserved.
- Interpretation in term of ensemble methods (bagging)
- No dropout after training!

Sources: Géron / Goodfellow, Bengio and Courville

Regularization by Batch Normalization

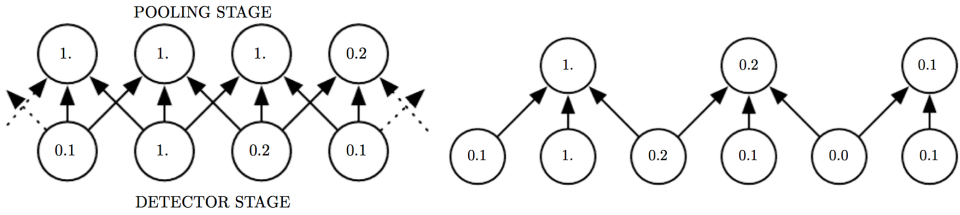
- Idea: Maintain the scales of the neuron outputs during training.

Batch Normalization Layer

- During training, at each step and for each neuron
  - Compute the mean value on the current batch
$$\mu_k^{(l)} = \frac{1}{m}\sum_{i=1}^{m} h_k^{(l)}(\underline{X}_i, \boldsymbol{w})$$
  - Compute the variance on the current batch
$$\sigma_k^{2,(l)} = \frac{1}{m}\sum_{i=1}^{m}(h_k^{(l)}(\underline{X}_i, \boldsymbol{w}) - \mu_k^{(l)})^2$$
  - Renormalize the entries $\tilde{h}_k^{(l)}(\underline{X}_i, \boldsymbol{w}) = \dfrac{h_k^{(l)}(\underline{X}_i, \boldsymbol{w}) - \mu_k^{(l)}}{\sigma_k^{(l)}}$
  - Train (update) the scale and shift parameters:
$$y_k^{(l+1)} = \gamma_k^{(l+1)}\tilde{h}_k^{(l)}(\underline{X}_i, \boldsymbol{w}) + \beta_k^{(l+1)}$$

- Differentiability in γ and β: SGD to train them.
- Final freeze of the mean and the variance yield the final NN.
- Helps a lot!

Regularization by Pooling



- Idea: replace some outputs by a statistic.

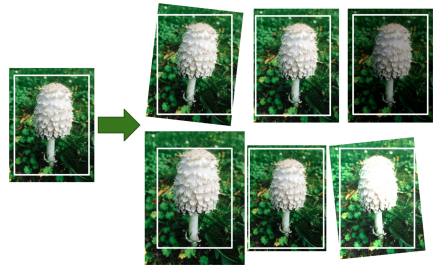
Pooling

- Replace values by a local summary statistic.
- Most classical summary choice: maximum.
- (Sub)Gradient can still be computed.
- Often combine with a subsampling to reduce the network size.

- Most useful when output position matters as in images.
- Very useful to deal with pictures of various sizes.

Source: Goodfellow, Bengio and Courville

Regularization by Data Augmentation



- Idea: Augment artificially the number of samples

Data Augmentation

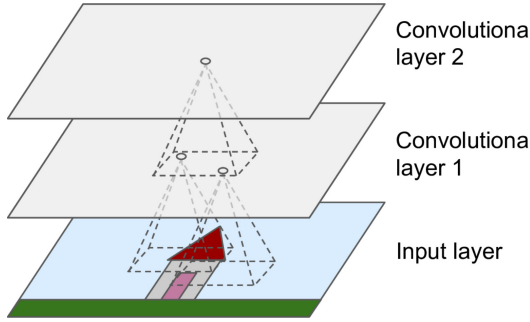
- Augment the dataset by generating new samples from original ones.
- Often mild variations of existing one: shift, rotation, zoom, noise...

- Often very efficient if the variations used correspond to specific problem invariance.
- Not so easy to generate completely new samples (GAN?)

Source: Géron

Outline

Convolutional Neural Networks

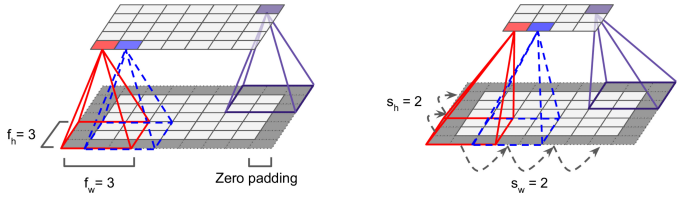


Convolutional Neural Networks (CNN)

- Reduce number of network weights:
  - Use only **local** computation (locality of information),
  - Use the **same weights** everywhere (translation invariance)
- Keep the hierarchical structure.

Source: Géron

Convolutional Layer



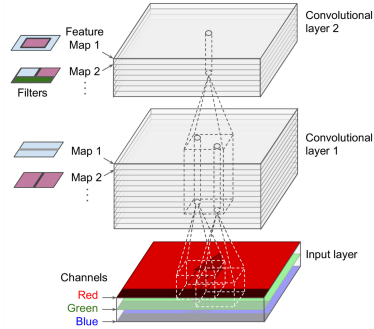
Convolution

- Filter structure:
  - Output is computed using only neighbors (locality of information)
  - Same computation with respect to the neighbors position everywhere (translation invariance)
- Translation invariance $\Leftrightarrow$ convolution

- Subsampling of resulting output possible (stride) but issues with high frequency part of filters.
- Better subsampling stability after pooling.

Source: Géron

CNN Layers



Maps and Layers

- Each layer contains a **stack** of outputs (3D tensor).
- Filter : convolution in 2D but free along the stack direction.
- Further reduction of the weight number using 2D spatial convolution followed by arbitrary 1D filter along stack.

Source: Géron

CNN

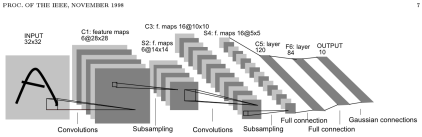
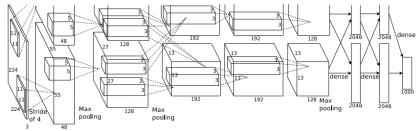


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet (LeCun, 89)



AlexNet (Krishevsky et al., 12)

CNN

- Stack of convolutional layers followed by a fully connected one.
- Key is to be able to learn in less than a few days!

Sources: Y. LeCun / A. Krizhevsky

Outline

Text as Sequences

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented connected handwriting recognition or speech recognition.

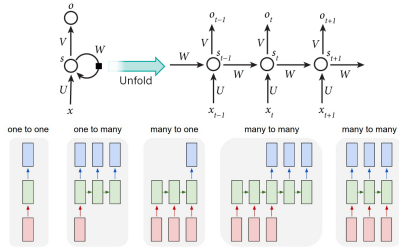
Sequences

- Word = sequence of letters.
- Text = sequence of letters/words.

- Capitalize on this structure.

Source: Wikipedia

Recurrent Neural Networks



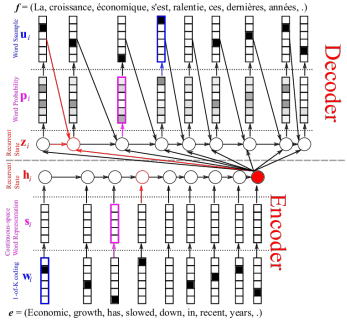
Recurrent Neural Network Unit

- Input seen as a sequence.
- **Simple** computational units with shared weights.
- Information transfer through a context!

- Several architectures!

Sources: Y. LeCun / Unknown

Automatic Translation

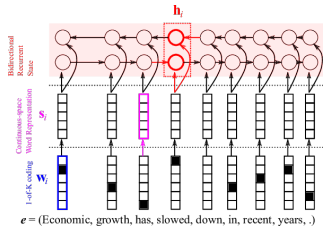


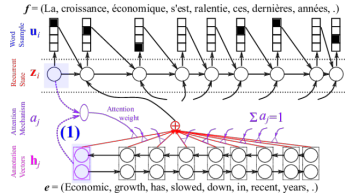
Encoder/Decoder structure

- Word vectors, RNN, stacked structure.

Source: Nvidia

Automatic Translation



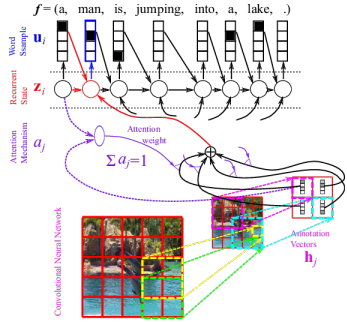


Encoder/Decoder structure

- Much more complex structure: asymmetric, attention order...

Source: Nvidia

Automatic Captioning

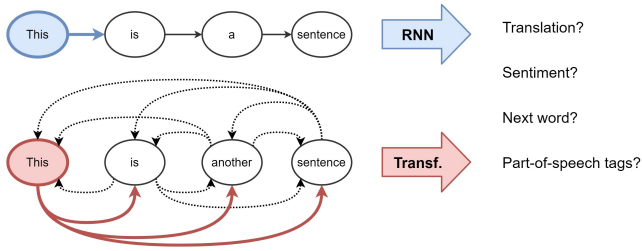


Encoder/Decoder structure

- Much more complex structure: asymmetric, attention order...

Source: Nvidia

Text as Graph

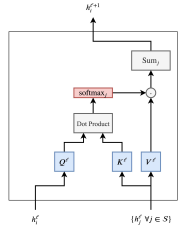


Text as Graph

- More than just sequential dependency.
- Each word is related to (all the) other words.
- Graph structure with words and directed relations between words.

Source: Chaitanya Joshi

Attention



Attention between words

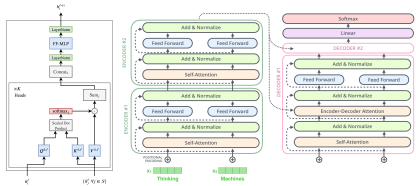
- Words encoded by h_i at layer l.
- Compute individual value for each word: $v_i = V^l h_i$
- Compute combined value for each word: $h_i' = \sum_j w_{i,j} v_j$
- **(Self) Attention:** weight $w_{i,j}$ defined by

$$w_{i,j} = \text{SoftMax}\left(\left\langle Q^l h_i, K^l h_j \right\rangle\right)$$

- $Q^l h_i$ is called a query and $K^l h_j$ a key.

Source: Chaitanya Joshi

Transformer



Transformer

- Block combining several attention heads and a classical MLP.

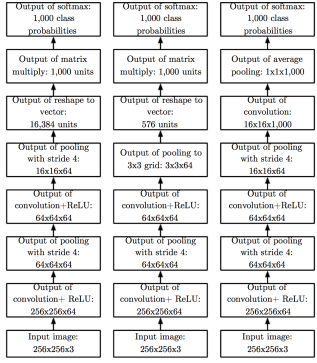
Encoder/Decoder Architecture

- Combine several transformers and more MLP in a task-adapted architecture.
- End-to-end training is not easy (initialization, optimization...).
- Initial embedding at token level rather than word level to cope with new words!

Sources: Chaitanya Joshi / Jay Alamar

Outline

Basic CNN Architecture

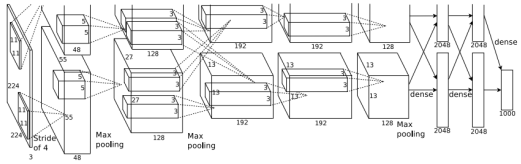


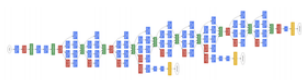
Basic choices

- Nb of layers, size of layers,
- Activation function, pooling...
- Optimization algorithm!

Source: Goodfellow, Bengio and Courville

CNN Architecture

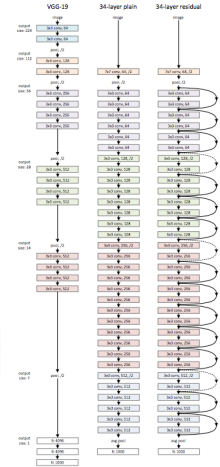




[1]Inception 5 (GoogLeNet)

Inception 7a

[1]Going Deeper with Convolutions, [C. Szegedy et al, CVPR 2015]

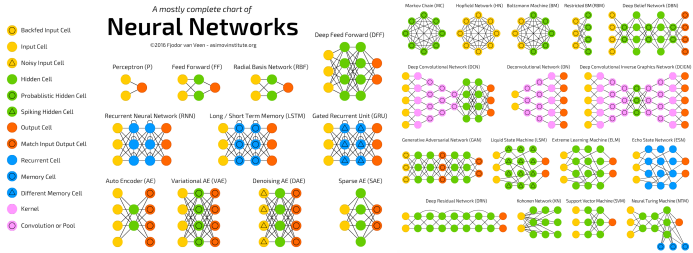


CNN Architecture

- Lot of freedom!

Sources: A. Krizhevsky / Google / Microsoft

DNN Architecture



Other Architectures

- Several structures:
  - Different tasks...
  - Different inputs.
- **Representation Learning is everywhere**

- Key: Differentiability by composition of basic differentiable units.

Source: Asimov Institute

Outline

References

T. Hastie, R. Tibshirani, and J. Friedman.
The Elements of Statistical Learning.
Springer Series in Statistics, 2009

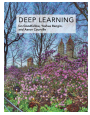
M. Mohri, A. Rostamizadeh, and A. Talwalkar.
Foundations of Machine Learning (2nd ed.)
MIT Press, 2018

A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd ed.)
O'Reilly, 2022

Ch. Giraud.
Introduction to High-Dimensional Statistics.
CRC Press, 2014

S. Bubeck.
Convex Optimization: Algorithms and Complexity.
Now Publisher, 2015

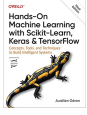
References



I. Goodfellow, Y. Bengio, and A. Courville.
Deep Learning.
MIT Press, 2016

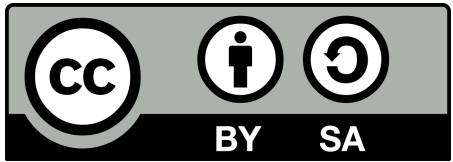
F. Chollet.
Deep Learning with Python (2nd ed.)
Manning, 2021

F. Chollet, T. Kalinowski, and J. J. Allaire.
Deep Learning with R (2nd ed.)
Manning, 2022



A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd ed.)
O'Reilly, 2022

Licence and Contributors



Contributors

- Main contributor: E. Le Pennec
- Contributors: S. Boucheron, A. Dieuleveut, A.K. Fermin, S. Gadat, S. Gaiffas, A. Guilloux, Ch. Keribin, E. Matzner, M. Sangnier, E. Scornet.

Outline

Outline

Guess Who?

A game of questions

- Game invented in 1979 in the UK.
- **Goal:** discover the character chosen by your opponent before he discovers yours.
- **Optimal strategy:** choose at each step the question that splits the remaining characters in two groups with the least possible difference in size.
- **Information Theory!**

- Adaptive construction of a tree of questions!
- Optimal tree of questions can be constructed without knowing the answers... but during a game only a path of the tree is used...

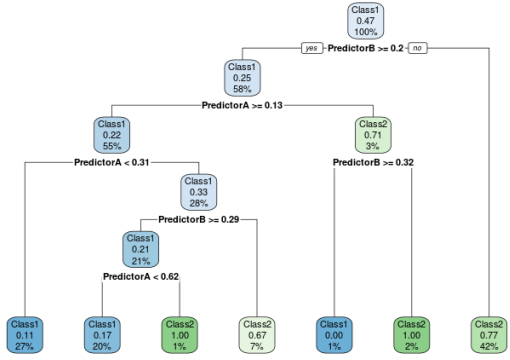
Classification And Regression Trees

Tree principle (CART by Breiman (85) / ID3 by Quinlan (86))

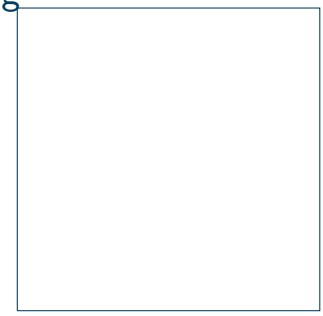
- Construction of a recursive partition through a tree structured set of questions (splits around a given value of a variable)
- For a given partition, probabilistic approach **and** optimization approach yield the same predictor!
- A simple majority vote/averaging in each leaf

- Quality of the prediction depends on the tree (the partition).
- **Intuitively:**
  - small leaves lead to low bias, but large variance
  - large leaves lead to large bias, but low variance...
- **Issue:** Minim. of the (penalized) empirical risk is NP hard!
- Practical tree construction are all based on two steps:
  - a top-down step in which branches are created (branching)
  - a bottom-up in which branches are removed (pruning)

CART


Class1
0.47
100%
yes
PredictorB >= 0.2
no
Class1
0.25
58%
PredictorA >= 0.13
Class1
0.22
55%
Class2
0.71
3%
PredictorA < 0.31
PredictorB >= 0.32
Class1
0.33
28%
PredictorB >= 0.29
Class1
0.21
21%
PredictorA < 0.62
Class1
0.11
27%
Class1
0.17
20%
Class2
1.00
1%
Class2
0.67
7%
Class1
0.00
1%
Class2
1.00
2%
Class2
0.77
42%

Branching

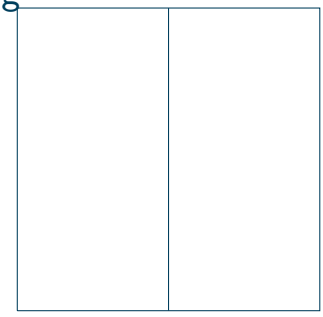


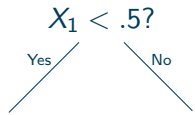
Greedy top-bottom approach

- Start from a single region containing all the data
- Recursively split those regions along a certain variable and a certain value

- **No regret strategy** on the choice of the splits!
- **Heuristic:** choose a split so that the two new regions are as *homogeneous* possible...

Branching



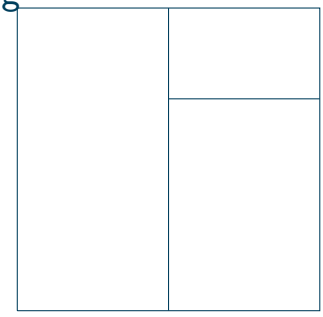


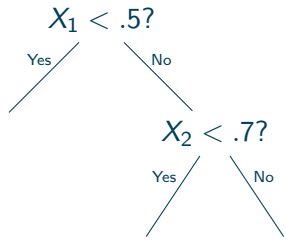
Greedy top-bottom approach

- Start from a single region containing all the data
- Recursively split those regions along a certain variable and a certain value

- **No regret strategy** on the choice of the splits!
- **Heuristic:** choose a split so that the two new regions are as *homogeneous* possible...

Branching



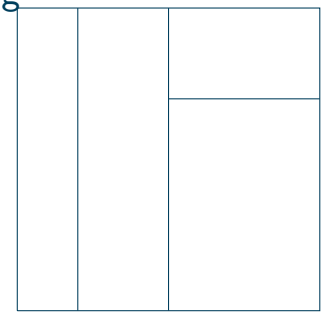


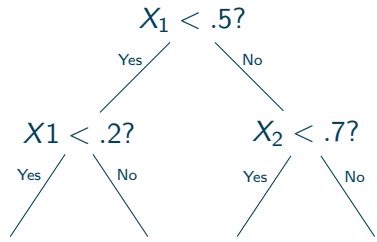
Greedy top-bottom approach

- Start from a single region containing all the data
- Recursively split those regions along a certain variable and a certain value

- **No regret strategy** on the choice of the splits!
- **Heuristic:** choose a split so that the two new regions are as *homogeneous* possible...

Branching





Greedy top-bottom approach

- Start from a single region containing all the data
- Recursively split those regions along a certain variable and a certain value

- **No regret strategy** on the choice of the splits!
- **Heuristic:** choose a split so that the two new regions are as *homogeneous* possible...

Branching

Various definition of in*homogeneous*

- **CART:** empirical loss based criterion (least squares/prediction error)
$$C(R, \overline{R}) = \sum_{\underline{x}_i \in R} \bar{\ell}(y_i, y(R)) + \sum_{\underline{x}_i \in \overline{R}} \bar{\ell}(y_i, y(\overline{R}))$$
- **CART:** Gini index (Classification)
$$C(R, \overline{R}) = \sum_{\underline{x}_i \in R} p(R)(1 - p(R)) + \sum_{\underline{x}_i \in \overline{R}} p(\overline{R})(1 - p(\overline{R}))$$
- **C4.5:** entropy based criterion (Information Theory)
$$C(R, \overline{R}) = \sum_{\underline{x}_i \in R} H(R) + \sum_{\underline{x}_i \in \overline{R}} H(\overline{R})$$

- CART with Gini is probably the most used technique...
- Other criterion based on χ^2 homogeneity or based on different local predictors (generalized linear models...)

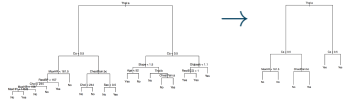
Branching

Choice of the split in a given region

- Compute the criterion for **all features and all possible splitting points** (necessarily among the data values in the region)
- Choose the split **minimizing** the criterion

- **Variations:** split at all categories of a categorical variable using a clever category ordering (ID3), split at a restricted set of points (quantiles or fixed grid)
- **Stopping rules:**
  - when a leaf/region contains less than a prescribed number of observations
  - when the region is sufficiently homogeneous...
- May lead to a quite complex tree: over-fitting possible!
- Additional pruning often use.

Pruning



- **Model selection** within the (rooted) subtrees of previous tree!
- Number of subtrees can be quite large, but the tree structure allows to find the best model efficiently.

Key idea

- The predictor in a leaf depends only on the values in this leaf.
- **Efficient bottom-up (dynamic programming) algorithm** if the criterion used satisfies an additive property

$$C(\mathcal{T}) = \sum_{\mathcal{L} \in \mathcal{T}} c(\mathcal{L})$$

- Example: AIC / CV.

Examples of criterion satisfying this assumptions

- AIC type criterion:

$$\sum_{i=1}^{n} \bar{\ell}(y_i, f_{\mathcal{L}(\underline{x}_i)}(\underline{x}_i)) + \lambda|\mathcal{T}| = \sum_{\mathcal{L}\in\mathcal{T}} \left(\sum_{\underline{x}_i \in \mathcal{L}} \bar{\ell}(y_i, f_{\mathcal{L}}(\underline{x}_i)) + \lambda \right)$$

- Simple cross-Validation (with $(\underline{x}'_i, y'_i)$ a different dataset):

$$\sum_{i=1}^{n'} \bar{\ell}(y'_i, f_{\mathcal{L}}(\underline{x}'_i)) = \sum_{\mathcal{L}\in\mathcal{T}} \left(\sum_{\underline{x}'_i \in \mathcal{L}} \bar{\ell}(y'_i, f_{\mathcal{L}}(\underline{x}'_i)) \right)$$

- Limit over-fitting for a single tree.
- **Rk:** almost never used when combining several trees...

Pruning and Dynamic Algorithm

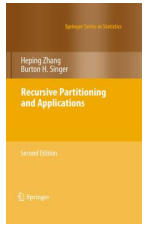
- **Key observation:** at a given node, the best subtree is either the current node or the union of the best subtrees of its child.

Dynamic programming algorithm

- Compute the individual cost $c(\mathcal{L})$ of each node (including the leaves)
- Scan all the nodes in reverse order of depth:
  - If the node $\mathcal{L}$ has no child, set its best subtree $\mathcal{T}(\mathcal{L})$ to $\{\mathcal{L}\}$ and its current best cost $c'(\mathcal{L})$ to $c(\mathcal{L})$
  - If the children $\mathcal{L}_1$ and $\mathcal{L}_2$ are such that $c'(\mathcal{L}_1) + c'(\mathcal{L}_2) \geq c(\mathcal{L})$, then prune the child by setting $\mathcal{T}(\mathcal{L}) = \{\mathcal{L}\}$ and $c'(\mathcal{L}) = c(\mathcal{L})$
  - Otherwise, set $\mathcal{T}(\mathcal{L}) = \mathcal{T}(\mathcal{L}_1) \cup \mathcal{T}(\mathcal{L}_2)$ and $c'(\mathcal{L}) = c'(\mathcal{L}_1) + c'(\mathcal{L}_2)$
- The best subtree is the best subtree $\mathcal{T}(\mathcal{R})$ of the root $\mathcal{R}$.

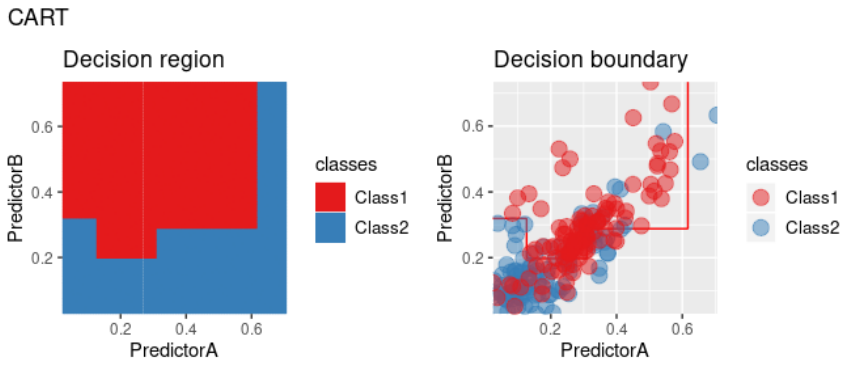
- Optimization cost proportional to the **number of nodes** and not the number of subtrees!

Extensions



- **Local estimation** of the proportions or of the conditional mean.
- **Recursive Partitioning methods:**
  - Recursive construction of a partition
  - Use of simple local model on each part of the partition
- **Examples:**
  - CART, ID3, C4.5, C5
  - MARS (local linear regression models)
  - Piecewise polynomial model with a dyadic partition...
- **Book:** *Recursive Partitioning and Applications* by Zhang and Singer

CART


CART
Decision region
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2
Decision boundary
PredictorB
0.6
0.4
0.2
0.2
0.4
0.6
PredictorA
classes
Class1
Class2

CART: Pros and Cons

Pros

- Leads to an easily interpretable model
- Fast computation of the prediction
- Easily deals with categorical features (and missing values)

Cons

- Greedy optimization
- Hard decision boundaries
- Lack of stability

Ensemble methods

- Lack of robustness for single trees.
- How to combine trees?

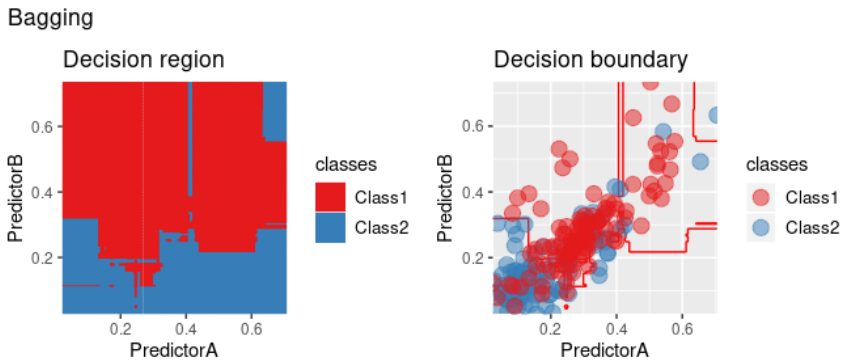
Parallel construction

- Construct several trees from bootstrapped samples and average the responses **(Bagging)**
- Add more randomness in the tree construction **(Random Forests)**

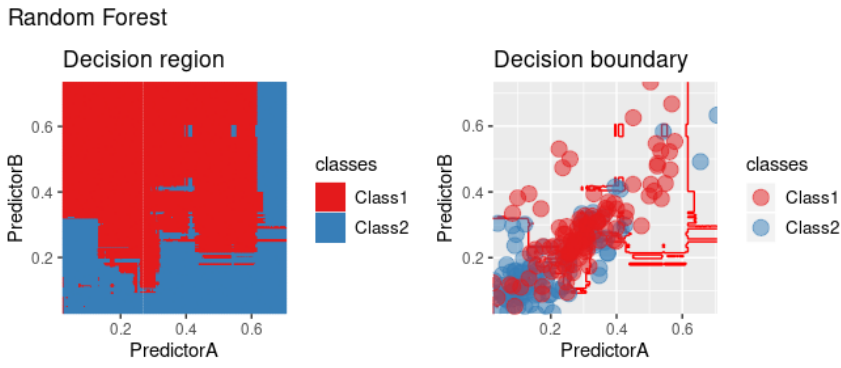
Sequential construction

- Construct a sequence of trees by reweighting sequentially the samples according to their difficulties **(AdaBoost)**
- Reinterpretation as a stagewise additive model **(Boosting)**

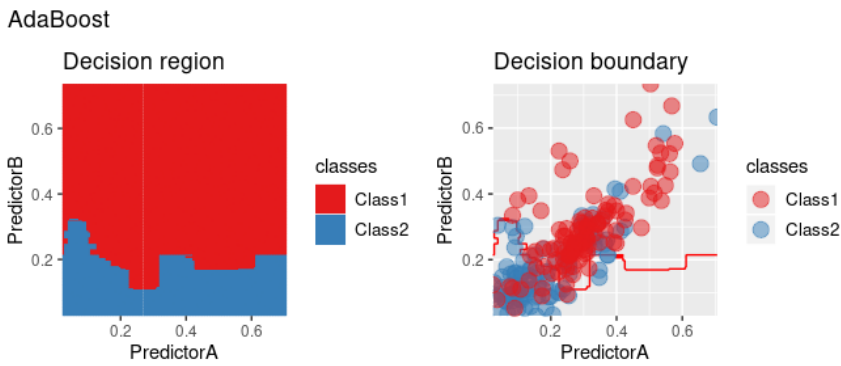
Ensemble methods



Ensemble methods



Ensemble methods



Outline

Outline

Independent Average

Stability through averaging

- Very simple idea to obtain a more stable estimator.
- **Vote/average** of B predictors $f_1, \ldots, f_B$ obtained with **independent datasets** of size n!

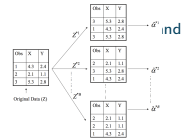
$$f_{\text{agr}} = \text{sign}\left(\frac{1}{B}\sum_{b=1}^{B} f_b\right) \quad \text{or} \quad f_{\text{agr}} = \frac{1}{B}\sum_{i=1}^{B} f_b$$

- **Regression:** $\mathbb{E}[f_{\text{agr}}(x)] = \mathbb{E}[f_b(x)]$ and $\mathbb{V}\text{ar}\left[f_{\text{agr}}(x)\right] = \frac{\mathbb{V}\text{ar}[f_b(x)]}{B}$
- **Prediction:** slightly more complex analysis
- Averaging leads to **variance reduction**, i.e. stability!

- **Issue:** cost of obtaining B independent datasets of size n!

Bagging and Bootstrap

- Strategy proposed by Breiman in 1994.



Stability through bootstrapping

- Instead of using B independent datasets of size n, draw B datasets from a single one using a **uniform with replacement** scheme (Bootstrap).
- **Rk:** On average, a fraction of $(1 - 1/e) \simeq .63$ examples are unique among each drawn dataset...
- The f_b are still identically distributed but **not independent** anymore.
- Price for the non independence: $\mathbb{E}[f_{\text{agr}}(x)] = \mathbb{E}[f_b(x)]$ and

$$\mathbb{V}\text{ar}\left[f_{\text{agr}}(x)\right] = \frac{\mathbb{V}\text{ar}\left[f_b(x)\right]}{B} + \left(1 - \frac{1}{B}\right)\rho(x)$$

  with $\rho(x) = \mathbb{C}\text{ov}\left[f_b(x), f_{b'}(x)\right] \leq \mathbb{V}\text{ar}\left[f_b(x)\right]$ with $b \neq b'$.
- **Bagging:** Bootstrap Aggregation

- Better aggregation scheme exists...

Outline

Randomized Predictors

- Correlation leads to less variance reduction:
$$\mathbb{V}\mathrm{ar}\left[f_{\mathrm{agr}}(x)\right] = \frac{\mathbb{V}\mathrm{ar}\left[f_b(x)\right]}{B} + \left(1 - \frac{1}{B}\right)\rho(x)$$
with $\rho(x) = \mathbb{C}\mathrm{ov}\left[f_b(x), f_{b'}(x)\right]$ with $b \neq b'$.
- **Idea:** Reduce the correlation by adding more randomness in the predictor.

Randomized Predictors

- Construct predictors that depend on a **randomness source** R that may be chosen independently for all bootstrap samples.
- This **reduces** the correlation between the estimates and thus the **variance**...
- But may **modify heavily the estimates** themselves!

- **Performance gain** not obvious from theory...

Random Forest

- Example of randomized predictors based on trees proposed by Breiman in 2001...

Random Forest

- Draw B resampled datasets from a single one using a uniform with replacement scheme (**Bootstrap**)
- For each resampled dataset, construct a tree using a different **randomly drawn subset of variables** at each split.

- Most important parameter is the **subset size**:
  - if it is too large then we are back to bagging
  - if it is too small the mean of the predictors is probably not a good predictor...
- **Recommendation:**
  - Classification: use a proportion of $1/\sqrt{p}$
  - Regression: use a proportion of $1/3$
- **Sloppier stopping rules** and pruning than in CART...

Extra Trees

- Extremely randomized trees!

Extra Trees

- Variation of random forests.
- Instead of trying all possible cuts, try only K cuts at random for each variable.
- No bootstrap in the original article.

- Cuts are defined by a threshold drawn uniformly in the feature range.
- Much faster than the original forest and similar performance.
- Theoretical performance analysis very challenging!

Error Estimate and Variable Ranking

Out Of the Box Estimate

- For each sample x_i, a prediction can be made using only the resampled datasets not containing x_i...
- The corresponding empirical prediction error is **not prone to overfitting** but does not correspond to the final estimate...
- Good proxy nevertheless.

Forests and Variable Ranking

- **Importance:** Number of use or criterion gain at each split can be used to rank the variables.
- **Permutation tests:** Difference between OOB estimate using the true value of the jth feature and a value drawn a random from the list of possible values.

- Up to OOB error, the permutation technique is not specific to trees.

Outline

Outline

AdaBoost

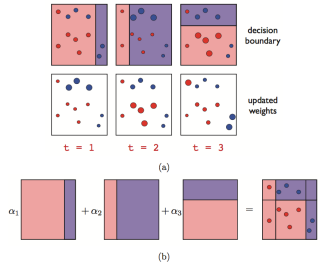
- **Idea:** learn a sequence of predictors trained on weighted dataset with weights depending on the loss so far.

Iterative scheme proposed by Schapire and Freud

- Set $w_{1,i} = 1/n$; $t = 0$ and $f = 0$
- For $t = 1$ to $t = T$
  - $h_t = \text{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} w_{t,i} \ell^{0/1}(y_i, h(x_i))$
  - Set $\epsilon_t = \sum_{i=1}^{n} w_{t,i} \ell^{0/1}(y_i, h_t(x_i))$ and $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
  - let $w_{t+1,i} = \frac{w_{t,i} e^{-\alpha_t y_i h_t(x_i)}}{Z_{t+1}}$ where Z_{t+1} is a renormalization constant such that $\sum_{i=1}^{n} w_{t+1,i} = 1$
  - $f = f + \alpha_t h_t$
- Use $f = \sum_{i=1}^{T} \alpha_t h_t$

- **Intuition:** $w_{t,i}$ measures the difficulty of learning the sample i at step t...

AdaBoost



AdaBoost Intuition

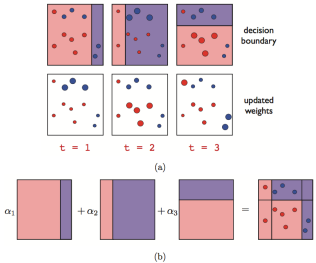
- h_t obtained by minimizing a weighted loss

$$h_t = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} w_{t,i} \ell^{0/1}(y_i, h(\underline{x}_i))$$

- Update the current estimate with

$$f_t = f_{t-1} + \alpha_t h_t$$

AdaBoost

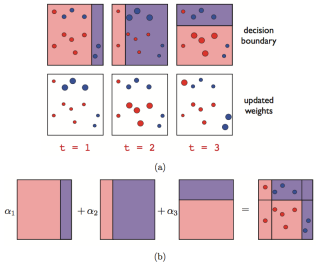


AdaBoost Intuition

- Weight $w_{t,i}$ should be large if $\underline{x}_i$ is not well-fitted at step $t-1$ and small otherwise.
- Use a weight proportional to $e^{-y_i f_{t-1}(\underline{x}_i)}$ so that it can be recursively updated by

$$w_{t+1,i} = w_{t,i} \times \frac{e^{-\alpha_t y_i h_t(\underline{x}_i)}}{Z_t}$$

AdaBoost



AdaBoost Intuition

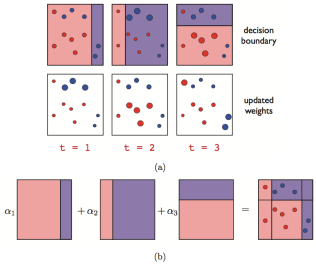
- Set α_t such that

$$\sum_{y_i h_t(\underline{x}i)=1} w_{t+1,i} = \sum_{y_i h_t(\underline{x}i)=-1} w_{t+1,i}$$

or equivalently

$$\left(\sum_{y_i h_t(\underline{x}i)=1} w_{t,i} \right) e^{-\alpha_t} = \left(\sum_{y_i h_t(\underline{x}i)=-1} w_{t,i} \right) e^{\alpha_t}$$

AdaBoost



AdaBoost Intuition

- Using

$$\epsilon_t = \sum_{y_i h_t(\underline{x} i) = -1} w_{t,i}$$

leads to

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t} \quad \text{and} \quad Z_t = 2\sqrt{\epsilon_t (1 - \epsilon_t)}$$

Exponential Stagewise Additive Modeling

- Set $t = 0$ and $f = 0$.
- For $t = 1$ to T,
  - $(h_t, \alpha_t) = \text{argmin}_{h,\alpha} \sum_{i=1}^{n} e^{-y_i(f(\underline{x}_i)+\alpha h(\underline{x}_i))}$
  - $f = f + \alpha_t h_t$
- Use $f = \sum_{t=1}^{T} \alpha_t h_t$

- **Greedy optimization** of a classifier as a linear combination of T classifier for the **exponential loss**.
- Those two algorithms are **exactly the same**!

Revisited AdaBoost

AdaBoost

- Set $t = 0$ and $f = 0$.
- For $t = 1$ to T,
  - $(h_t, \alpha_t) = \text{argmin}_{h,\alpha} \sum_{i=1}^{n} e^{-y_i(f(\underline{x}_i)+\alpha h(\underline{x}_i))}$
  - $f = f + \alpha_t h_t$
- Use $f = \sum_{t=1}^{T} \alpha_t h_t$

- **Greedy iterative scheme** with only two parameters: the class $\mathcal{H}$ of *weak* classifiers and the number of step T.
- In the literature, one can read that Adaboost does not overfit! This is not true and T should be chosen with care...

Outline

Weak Learners

Weak Learner

- Simple predictor belonging to a set $\mathcal{H}$.
- Easy to learn.
- Need to be only slightly better than a constant predictor.

Weak Learner Examples

- **Decision Tree** with few splits.
- **Stump** decision tree with one split.
- **(Generalized) Linear Regression** with few variables.

Boosting

- Sequential Linear Combination of Weak Learner
- Attempt to minimize a loss.

- Example of ensemble method.
- Link with Generalized Additive Modeling.

Generic Boosting

- **Greedy optim.** yielding a linear combination of *weak* learners.

Generic Boosting

- Algorithm:
  - Set $t = 0$ and $f = 0$.
  - For $t = 1$ to T,
    - $(h_t, \alpha_t) = \text{argmin}_{h,\alpha} \sum_{i=1}^{n} \bar{\ell}(y_i, f(x_i) + \alpha h(x_i))$
    - $f = f + \alpha_t h_t$
  - Use $f = \sum_{t=1}^{T} \alpha_t h_t$
- AKA as **Forward Stagewise Additive Modeling**
  - AdaBoost with $\bar{\ell}(y, h) = e^{-yh}$
  - LogitBoost with $\bar{\ell}(y, h) = \log_2(1 + e^{-yh})$
  - L_2Boost with $\bar{\ell}(y, h) = (y - h)^2$ (Matching pursuit)
  - L_1Boost with $\bar{\ell}(y, h) = |y - h|$
  - HuberBoost with $\bar{\ell}(y, h) = |y - h|^2 \mathbf{1}_{|y-h|<\epsilon} + (2\epsilon|y - h| - \epsilon^2)\mathbf{1}_{|y-h|\geq\epsilon}$

- Simple principle but **no easy numerical scheme** except for AdaBoost and L_2Boost...

Gradient Boosting

- **Issue:** At each boosting step, one need to solve

$$(h_t, \alpha_t) = \underset{h,\alpha}{\operatorname{argmin}} \sum_{i=1}^{n} \bar{\ell}(y_i, f(x_i) + \alpha h(x_i)) = L(y, f + \alpha h)$$

- **Idea:** Replace the function by a **first order approximation**

$$L(y, f + \alpha h) \sim L(y, f) + \alpha \langle \nabla L(y, f), h \rangle$$

Gradient Boosting

- Replace the minimization step by a **gradient descent** step:
  - Choose h_t as the best possible descent direction in $\mathcal{H}$ according to the approximation
  - Choose α_t that minimizes $L(y, f + \alpha h_t)$ (line search)

- **Rk:** Exact gradient direction often not possible!
- Need to find efficiently this best possible direction...

Best Direction

- Gradient direction:

$$\nabla L(y,f) \quad \text{with} \quad \nabla_i L(y,f) = \frac{\partial}{df(x_i)} \left(\sum_{i'=1}^{n} \bar{\ell}(y_{i'}, f(x_{i'})) \right)$$
$$= \frac{\partial}{df(x_i)} \bar{\ell}(y_i, f(x_i))$$

Best Direction within $\mathcal{H}$

- Direct formulation:

$$h_t \in \operatorname*{argmin}_{h \in \mathcal{H}} \frac{\sum_{i=1}^{n} \nabla_i L(y,f) h(x_i)}{\sqrt{\sum_{i=1}^{n} |h(x_i)|^2}} \left(= \frac{\langle \nabla L(y,f), h \rangle}{\|h\|} \right)$$

- Equivalent (least-squares) formulation: $h_t = -\beta_t h'_t$ with

$$(\beta_t, h'_t) \in \operatorname*{argmin}_{(\beta,h) \in \mathbb{R} \times \mathcal{H}} \sum_{i=1}^{n} |\nabla_i L(y,f) - \beta h(x_i)|^2 \left(= \|\nabla L - \beta h\|^2 \right)$$

- Choice of the formulation will depend on $\mathcal{H}$...

Gradient Boosting of Classifiers

- **Assumptions:**
  - h is a binary classifier, $h(x) = \pm 1$ and thus $\|h\|^2 = n$.
  - $\bar{\ell}(y, f(x)) = l(yf(x))$ so that $\nabla_i L(y, f) = y_i l'(y_i f(x_i))$.
- Best direction h_t in $\mathcal{H}$ using the first formulation

$$h_t = \operatorname*{argmin}_{h \in \mathcal{H}} \sum_i \nabla_i L(y, f) h(x_i)$$

AdaBoost Type Minimization

- Best direction rewriting

$$\begin{aligned} h_t &= \operatorname*{argmin}_{h \in \mathcal{H}} \sum_i l'(y_i f(x_i)) y_i h(x_i) \\ &= \operatorname*{argmin}_{h \in \mathcal{H}} \sum_i (-l')(y_i f(x_i))(2\ell^{0/1}(y_i, h(x_i)) - 1) \end{aligned}$$

- **AdaBoost type weighted loss minimization** as soon as $(-l')(y_i f(x_i)) \geq 0$:

$$h_t = \operatorname{argmin} \sum_i (-l')(y_i f(x_i)) \ell^{0/1}(y_i, h(x_i))$$

Gradient Boosting of Classifiers

Gradient Boosting

- **(Gradient) AdaBoost:** $\bar{\ell}(y, f) = \exp(-yf)$
  - $l(x) = \exp(-x)$ and thus $(-l')(y_i f(x_i)) = e^{-y_i f(x_i)} \geq 0$
  - h_t is the same as in AdaBoost
  - α_t also... (explicit computation)
- **LogitBoost:** $\bar{\ell}(y, f) = \log_2(1 + e^{-yf})$
  - $l(x) = \log_2(1 + e^{-x})$ and thus $(-l')(y_i f(x_i)) = \frac{e^{-y_i f(x_i)}}{\log(2)(1+e^{-y_i f(x_i)})} \geq 0$
  - Less weight on misclassified samples than in AdaBoost...
  - No explicit formula for α_t (line search)
  - Different path than with the (non-computable) classical boosting!
- **SoftBoost:** $\bar{\ell}(y, f) = \max(1 - yf, 0)$
  - $l(x) = \max(1 - x, 0)$ and $(-l')(y_i f(x_i)) = \mathbf{1}_{y_i f(x_i) \leq 1} \geq 0$
  - Do not use the samples that are sufficiently well classified!

Gradient Boosting and Least Squares

- Least squares formulation is often preferred when $|h| \neq 1$.

Least Squares Gradient Boosting

- Find $h_t = -\beta_t h'_t$ with

$$(\beta_t, h'_t) \in \underset{(\beta,h)\in\mathbb{R}\times\mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} |\nabla_i L(y, f) - \beta h(x_i)|^2$$

- Classical least squares if $\mathcal{H}$ is a finite dimensional vector space!
- Not a usual least squares in general but a classical regression problem!

- Numerical scheme depends on the loss...

Gradient Boosting and Least Squares

Examples

- **Gradient L_2Boost:**
  - $\ell(y, f) = |y - f|^2$ and $\nabla_i L(y_i, f(x_i)) = -2(y_i - f(x_i))$:

$$(\beta_t, h'_t) \in \underset{(\beta,h)\in\mathbb{R}\times\mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} |2y_i - 2(f(x_i) - \beta/2h(x_i))|^2$$

  - $\alpha_t = -\beta_t/2$
  - Equivalent to classical L_2-Boosting
- **Gradient L_1Boost:**
  - $\ell(y, f) = |y - f|$ and $\nabla_i L(y_i, f(x_i)) = -\operatorname{sign}(y_i - f(x_i))$:

$$(\beta_t, h'_t) \in \underset{(\beta,h)\in\mathbb{R}\times\mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} |-\operatorname{sign}(y_i - f(x_i)) - \beta h(x_i)|^2$$

  - Robust to outliers...
- Classical choice for $\mathcal{H}$: Linear Model in which each h depends on a small subset of variables.

Gradient Boosting and Least Squares

- Least squares formulation can also be used in classification!
- Assumption:
  - $\ell(y, f(x)) = l(yf(x))$ so that $\nabla_i L(y_i, f(x_i)) = y_i l'(y_i f(x_i))$

Least Squares Gradient Boosting for Classifiers

- Least Squares formulation:

$$(\beta_t, h'_t) \in \underset{(\beta,h)\in\mathbb{R}\times\mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} |y_i l'(y_i f(x_i)) - \beta h(x_i)|^2$$

- Equivalent formulation:

$$(\beta_t, h'_t) \in \underset{(\beta,h)\in\mathbb{R}\times\mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} |(-l')(y_i f(x_i)) - (-\beta) y_i h(x_i))|^2$$

- **Intuition:** Modify misclassified examples without modifying too much the well-classified ones...

Boosting Variations

Stochastic Boosting

- **Idea:** change the learning set at each step.
- Two possible reasons:
  - Optimization over all examples too costly
  - Add variability to use an averaged solution
- Two different samplings:
  - Use sub-sampling, if you need to reduce the complexity
  - Use re-sampling, if you add variability...
- Stochastic Gradient name mainly used for the first case...

Second Order Boosting

- Replace the first order approximation by a second order one and avoid the line search...

XGBoost

- Very efficient boosting algorithm proposed by Chen and Guestrin in 2014.

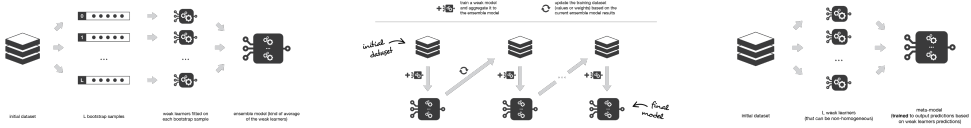
eXtreme Gradient Boosting

- Gradient boosting for a (penalized) smooth loss using a second order approximation and the least squares approximation.
- Reduced stepsize with a shrinkage of the *optimal* parameter.
- Feature subsampling.
- Weak learners:
  - Trees: limited depth, penalized size and parameters, fast approximate best split.
  - Linear model: elastic-net penalization.

- Excellent baseline!
- Lightgbm and CatBoost are also excellent similar choices!

Outline

Ensemble Methods



Ensemble Methods

- **Averaging:** combine several models by averaging (bagging, random forests,...)
- **Boosting:** construct a sequence of (weak) classifiers (XGBoost, LightGBM, CatBoost)
- **Stacking:** use the outputs of several models as features (tpot...)

- Loss of interpretability but gain in performance
- Beware of overfitting with stacking: the second learning step should be done with fresh data.
- No end to end optimization as in deep learning!

Source: J. Rocca

Outline

References

T. Hastie, R. Tibshirani, and J. Friedman.
The Elements of Statistical Learning.
Springer Series in Statistics, 2009

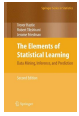
M. Mohri, A. Rostamizadeh, and A. Talwalkar.
Foundations of Machine Learning (2nd ed.)
MIT Press, 2018

A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd ed.)
O'Reilly, 2022

Ch. Giraud.
Introduction to High-Dimensional Statistics.
CRC Press, 2014

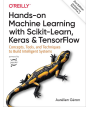
S. Bubeck.
Convex Optimization: Algorithms and Complexity.
Now Publisher, 2015

References



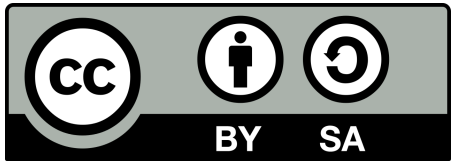
T. Hastie, R. Tibshirani, and J. Friedman.
The Elements of Statistical Learning.
Springer Series in Statistics, 2009

H. Zhang and B. Singer.
Recursive Partitioning and Applications.
Springer, 2010



A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (2nd ed.)
O'Reilly, 2019

Licence and Contributors



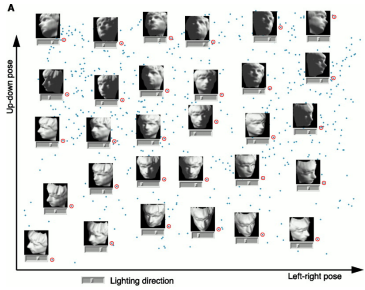
Contributors

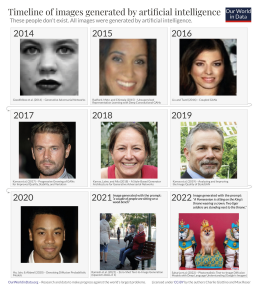
- Main contributor: E. Le Pennec
- Contributors: S. Boucheron, A. Dieuleveut, A.K. Fermin, S. Gadat, S. Gaiffas, A. Guilloux, Ch. Keribin, E. Matzner, M. Sangnier, E. Scornet.

Outline

Outline

Motivation





What is possible with data without labels?

- To group them?
- To visualize them in a 2 dimensional space?
- To generate more data?

Sources:: Unknown/Tenenbaum et al./World Economic Forum

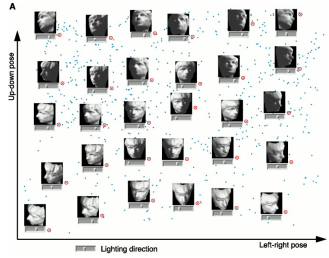
Marketing and Groups

To group them?

- **Data:** Base of customer data containing their properties and past buying records
- **Goal:** Use the customers *similarities* to find groups.
- **Clustering:** propose an explicit *grouping* of the customers
- **Visualization:** propose a representation of the customers so that the groups are *visible*. (Bonus)

Source: Unknown

Image and Visualization

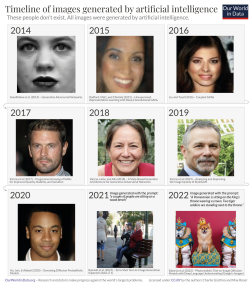


To visualize them?

- **Data:** Images of a single object
- **Goal:** Visualize the *similarities* between images.
- **Visualization:** propose a representation of the images so that similar images are *close*.
- **Clustering:** use this representation to cluster the images. (Bonus)

Source: Tenenbaum et al.

Images and Generation

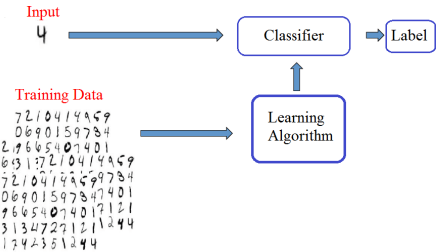


To generate more data?

- **Data:** Images.
- **Goal:** Generate images similar to the ones in the dataset.
- **Generative Modeling:** propose (and train) a generator.

Source: World Economic Forum

Machine Learning



A definition by Tom Mitchell

A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E.

Source: Unknown

Supervised Learning

Experience, Task and Performance measure

- **Training data** : $\mathcal{D} = \{(\underline{X}_1, Y_1), \ldots, (\underline{X}_n, Y_n)\}$ (i.i.d. $\sim \mathbb{P}$)
- **Predictor**: $f : \mathcal{X} \to \mathcal{Y}$ measurable
- **Cost/Loss function**: $\ell(f(\underline{X}), Y)$ measure how well $f(\underline{X})$ *predicts* Y
- **Risk**:

$$\mathcal{R}(f) = \mathbb{E}[\ell(Y, f(\underline{X}))] = \mathbb{E}_X\Big[\mathbb{E}_{Y|\underline{X}}[\ell(Y, f(\underline{X}))]\Big]$$

- Often $\ell(f(\underline{X}), Y) = \|f(\underline{X}) - Y\|^2$ or $\ell(f(\underline{X}), Y) = \mathbf{1}_{Y \neq f(\underline{X})}$

Goal

- Learn a rule to construct a **predictor** $\widehat{f} \in \mathcal{F}$ from the training data $\mathcal{D}_n$ s.t. **the risk** $\mathcal{R}(\widehat{f})$ is **small on average** or with high probability with respect to $\mathcal{D}_n$.

Unsupervised Learning

Experience, Task and Performance measure

- **Training data** : $\mathcal{D} = \{\underline{X}_1, \ldots, \underline{X}_n\}$ (i.i.d. $\sim \mathbb{P}$)
- **Task**: ???
- **Performance measure**: ???

- No obvious task definition!

Tasks for this lecture

- **Dimension reduction:** construct a map of the data in a **low dimensional** space without **distorting** it too much.
- **Clustering (or unsupervised classification):** construct a **grouping** of the data in **homogeneous** classes.

Dimension Reduction

- **Training data** : $\mathcal{D} = \{\underline{X}_1, \dots, \underline{X}_n\} \in \mathcal{X}^n \quad$ (i.i.d. $\sim \mathbb{P}$)
- Space $\mathcal{X}$ of possibly high dimension.

Dimension Reduction Map

- Construct a map Φ from the space $\mathcal{X}$ into a space $\mathcal{X}'$ of **smaller dimension**:

$$\begin{aligned} \Phi : \quad & \mathcal{X} \to \mathcal{X}' \\ & \underline{X} \mapsto \Phi(\underline{X}) \end{aligned}$$

- Map can be defined only on the dataset.

Motivations

- Visualization of the data
- Dimension reduction (or embedding) before further processing

Dimension Reduction

- Need to control the **distortion** between $\mathcal{D}$ and $\Phi(\mathcal{D}) = \{\Phi(\underline{X}_1), \ldots, \Phi(\underline{X}_n)\}$

Distortion(s)

- Reconstruction error:
  - Construct $\widetilde{\Phi}$ from $\mathcal{X}'$ to $\mathcal{X}$
  - Control the error between $\underline{X}$ and its reconstruction $\widetilde{\Phi}(\Phi(\underline{X}))$
- Relationship preservation:
  - Compute a *relation* $\underline{X}_i$ and $\underline{X}_j$ and a *relation* between $\Phi(\underline{X}_i)$ and $\Phi(\underline{X}_j)$
  - Control the difference between those two *relations*.

- Lead to different constructions. . . .

Clustering

- **Training data** : $\mathcal{D} = \{\underline{X}_1, \ldots, \underline{X}_n\} \in \mathcal{X}^n \quad (\text{i.i.d. } \sim \mathbb{P})$
- Latent groups?

Clustering

- Construct a map f from $\mathcal{D}$ to $\{1, \ldots, K\}$ where K is a number of classes to be fixed:

$$f : \quad \underline{X}_i \mapsto k_i$$

- Similar to classification except:
  - no ground truth (no given labels)
  - label only elements of the dataset!

Motivations

- Interpretation of the groups
- Use of the groups in further processing

Clustering

- Need to define the **quality** of the cluster.
- No obvious measure!

Clustering quality

- Inner homogeneity: samples in the same group should be similar.
- Outer inhomogeneity: samples in two different groups should be different.

- Several possible definitions of similar and different.
- Often based on the distance between the samples.
- Example based on the Euclidean distance:
  - Inner homogeneity = intra-class variance,
  - Outer inhomogeneity = inter-class variance.
- **Beware:** choice of the number of clusters K often complex!

Generative Modeling

- **Training data** : $\mathcal{D} = \{(\underline{X}_1, \underline{Y}_1), \ldots, (\underline{X}_n, \underline{Y}_n)\} \in (\mathcal{X} \times \mathcal{Y})^n \quad (\text{i.i.d.} \sim \mathbb{P})$
- Same kind of data than for supervised learning if $\mathcal{Y} \neq \emptyset$.

Generative Modeling

- Construct a map G from the product of $\mathcal{Y}$ and a randomness source Ω to $\mathcal{X}$

$$G : \mathcal{Y} \times \Omega \to \mathcal{X}$$
$$(Y, \omega) \mapsto X$$

- Unconditional model if $\mathcal{Y} = \emptyset \ldots$

Motivation

- Generate plausible novel conditional samples based on a given dataset.

Sample Quality

- Related to the proximity between the law of $G(Y, \omega)$ and the law of $X|Y$.
- Most classical choice is the Kullback-Leibler divergence.

Generative Modeling

Ingredients

- Generator $F_\theta(Y, \omega)$ and cond. density prob. $p_\theta(X|Y)$ (Explicit vs implicit link)
- Simple / Complex / Approximate estimation...

Some Possible Choices

	Probabilistic model	Generator	Estimation
Base	Simple (parametric)	Explicit	Simple (ML)
Flow	Image of simple model	Explicit	Simple (ML)
Factorization	Factorization of simple model	Explicit	Simple (ML)
VAE	Simple model with latent var.	Explicit	Approximate (ML)
EBM	Arbitrary	Implicit (MCMC)	Complex (ML/score/discrim.)
Diffusion	Continuous noise	Implicit (MCMC)	Complex (score)
	Discrete Noise with latent var.	Explicit	Approximate (ML)
GAN	Implicit	Explicit	Complex (Discrimination)

- SOTA: Diffusion based approach!

ML: Maximum Likelihood/VAE: Variational AutoEncoder/EBM: Energy Based Model/MCMC: Monte Carlo Markov Chain/GAN: Generative Aversarial Network

Bonus Task: Representation Learning

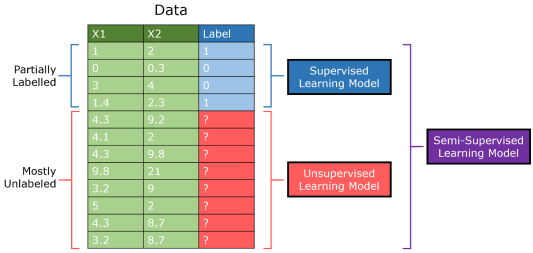
- **General observation:** most data do not have a label !
- **Example:** The number of images on which someone has described the content of the image is a *tiny fraction* of the images online.
- Labeling is very expensive and time consuming
- A lot of information can be extracted from the structure of the data, before seeing any label.

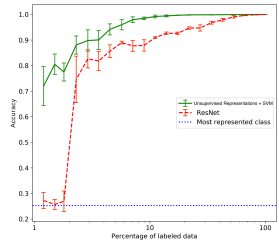
How can we leverage the large quantity of un-labeled data?

- Learn relevant features (= **representations**) in an unsupervised fashion
- Use those features to solve a supervised task with a fraction of labeled data.

- **Semi-supervised framework**
- ↬ Very useful in practice, for images, time series, text.

Semi-supervised Framework





Semu-Supervised Framework

- With representation learned in an unsupervised fashion + a simple linear model, one can achieve the same performance with 10% of data labeled than with a fully annotated dataset.

- Complementary regularization based approaches also exist.

Source: Andre Ye, J.Y. Franceschi et al.

Unsupervised Learning is a Versatile Approach!

Except for generative modeling, the learner is always right

- A subjective measure of performance
- Subjective choices for the algorithmic constraints (e.g., the type of transformation of the data we allow for low-dimensional representation, type of groups in clustering)
- $\Rightarrow$ Very difficult or impossible to tell which is the *best* method.

- Yet:
  - Extremely important in practice:
    - 90-99% of the data is un-labeled!
    - the tasks themselves are fundamental
  - Huge success in various fields (NLP, images...)

Unsupervised Learning is a Versatile Approach!

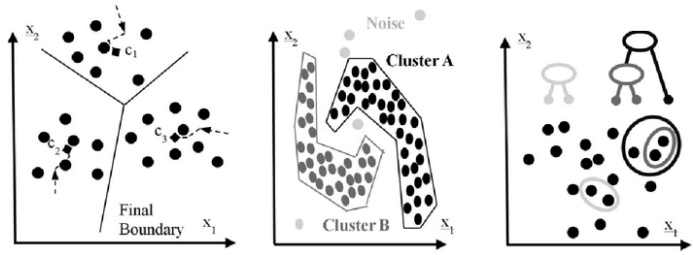
Lecture goals for the three main tasks

- Discussing possible choices of measures of performance and algorithmic constraints
- Understand the correspondences between those choices and a variety of classical algorithms
- For the simplest algorithms (PCA, k-means), get a precise mathematical understanding of the learning process.

Outline

Outline

What's a group?



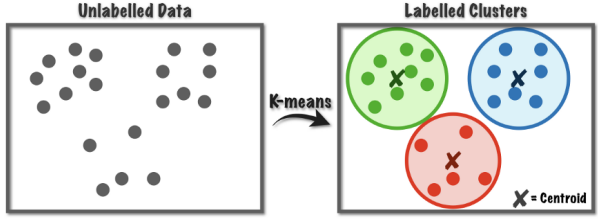
- No simple or unanimous definition!
- Require a notion of similarity/difference...

Three main approaches

- A group is a set of samples similar to a prototype.
- A group is a set of samples that can be linked by contiguity.
- A group can be obtained by fusing some smaller groups...

Source: J. Pitel

Prototype Approach



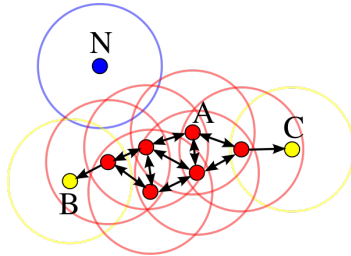
Prototype Approach

- A group is a set of samples similar to a prototype.
- Most classical instance: k-means algorithm.
- Principle: alternate prototype choice for the current groups and group update based on those prototypes.

- Number of groups fixed at the beginning
- No need to compare the samples between them!

Source: A. Jeffares

Contiguity Approach



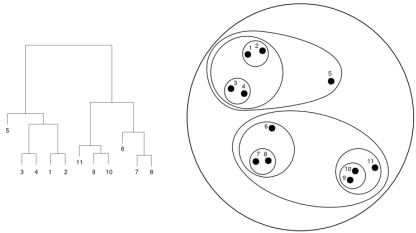
Contiguity Approach

- A group is the set of samples that can be linked by contiguity.
- Most classical instance: DBScan
- Principle: group samples by contiguity if possible (proximity and density)

- Some samples may remain isolated.
- Number of groups controlled by the scale parameter.

Source: Wikipedia

Agglomerative Approach

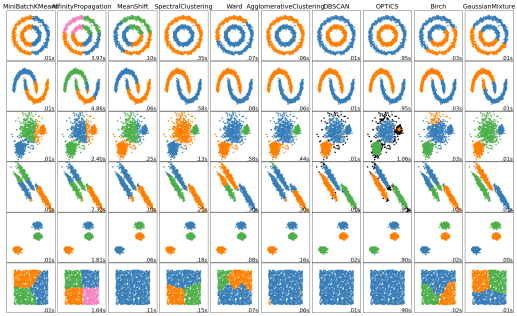


Agglomerative Approach

- A group can be obtained by fusing some smaller groups...
- Hierachical clustering principle: sequential merging of groups according to a *best merge* criterion

- Numerous variations on the merging criterion...
- Number of groups chosen afterward.

Choice of the method and of the number of groups



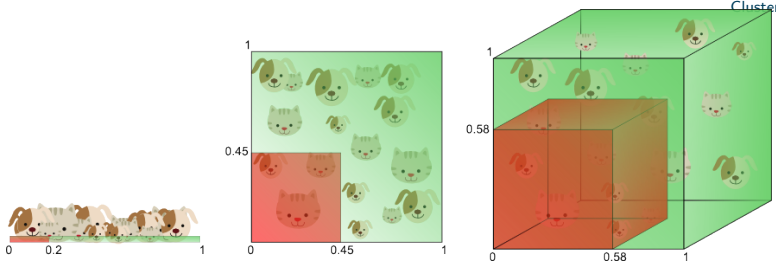
No method or number of groups is better than the others. . .

- Criterion not necessarily explicit!
- No cross validation possible
- Choice of the number of groups: a priori, heuristic, *based on the final usage. . .*

Source: Scikit-Learn

Outline

Dimensionality Curse



- **DISCLAIMER: Even if they are used everywhere beware of the usual distances in high dimension!**

Dimensionality Curse

- Previous approaches based on distances.
- Surprising behavior in high dimension: everything is ((often) as) far away.
- Beware of categories...

Source: Vision Dummy

Dimensionality Curse

- **DISCLAIMER: Even if they are used everywhere beware of the usual distances in high dimension!**

High Dimensional Geometry Curse

- Folks theorem: In high dimension, everyone is alone.
- Theorem: If $\underline{X}_1, \ldots, \underline{X}_n$ in the hypercube of dimension d such that their coordinates are i.i.d then

$$d^{-1/p}\left(\max \|\underline{X}_i - \underline{X}_j\|_p - \min \|\underline{X}_i - \underline{X}_j\|_p\right) = 0 + O_P\left(\sqrt{\frac{\log n}{d}}\right)$$

$$\frac{\min \|\underline{X}_i - \underline{X}_j\|_p}{\max \|\underline{X}_i - \underline{X}_j\|_p} = 1 + O_P\left(\sqrt{\frac{\log n}{d}}\right).$$

- When d is large, all the points are almost equidistant...
- Nearest neighbors are meaningless!

Outline

Visualization and Dimension Reduction

Visualization and Dimension Reduction

- How to view a dataset in high dimension !
- High dimension: dimension larger than 2!
- **Projection** onto a 2D space.

Source: F. Belardi

Visualization and Dimension Reduction

Visualization and Dimension Reduction

- How to view a dataset in high dimension !
- High dimension: dimension larger than 2!
- **Projection** onto a 2D space.

Source: F. Belardi

Visualization and Dimension Reduction

Visualization and Dimension Reduction

- How to view a dataset in high dimension !
- High dimension: dimension larger than 2!
- **Projection** onto a 2D space.

Source: F. Belardi

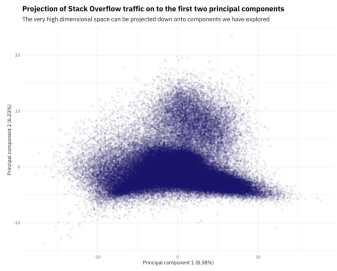
Visualization and Dimension Reduction

Visualization and Dimension Reduction

- How to view a dataset in high dimension !
- High dimension: dimension larger than 2!
- **Projection** onto a 2D space.

Source: F. Belardi

Principal Component Analysis



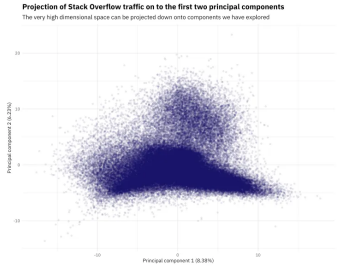
- Simple formula: $\tilde{X} = P(X - m)$

How to chose P?

- Maximising the dispersion of the points?
- Allowing to well reconstruct X from $\tilde{X}$?
- Preserving the relationship between the X through those between the $\tilde{X}$?

Source: J. Silge

Principal Component Analysis



- Simple formula: $\tilde{X} = P(X - m)$

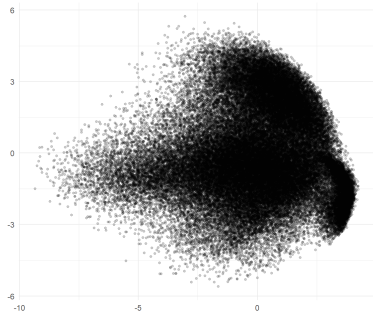
How to chose P?

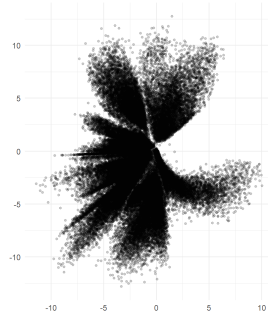
- Maximising the dispersion of the points?
- Allowing to well reconstruct X from $\tilde{X}$?
- Preserving the relationship between the X through those between the $\tilde{X}$?

- The 3 approaches yield the same solution!

Source: J. Silge

Reconstruction Approaches



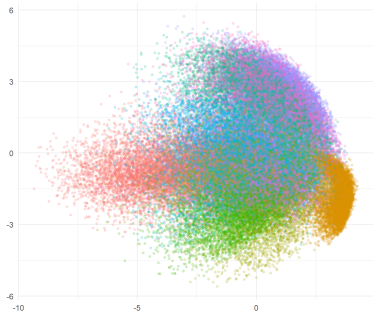


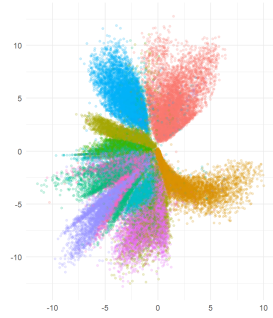
Reconstruction Approaches

- Learn a formula to encode and one formula to decode.
- Auto-encoder structure

- Yields a formula for new points.

Reconstruction Approaches



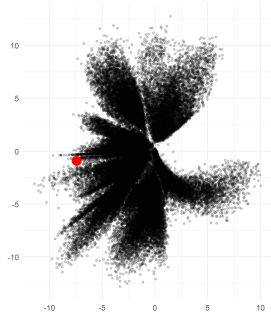


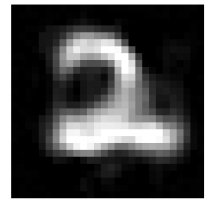
Reconstruction Approaches

- Learn a formula to encode and one formula to decode.
- Auto-encoder structure

- Yields a formula for new points.

Reconstruction Approaches



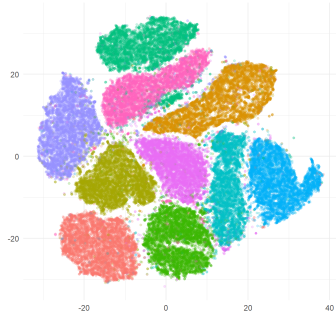


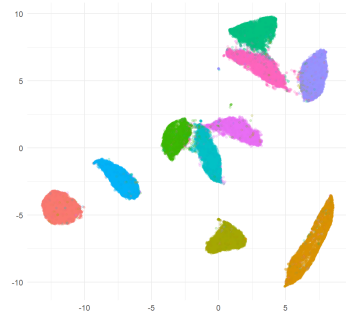
Reconstruction Approaches

- Learn a formula to encode and one formula to decode.
- Auto-encoder structure

- Yields a formula for new points.

Relationship Preservation Approaches



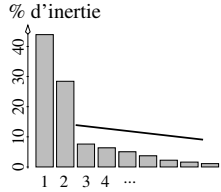


Relationship Preservation Approaches

- Based on the definition of the relationship notion (in both worlds).
- Huge flexibility

- Not always yields a formula for new points.

Choices of Methods and Dimension

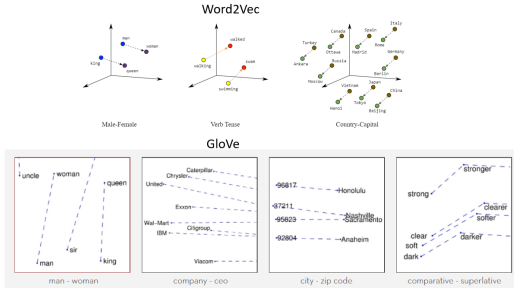


No Better Choice?

- Different criterion for different methods: impossible to use cross-validation.
- The larger the dimension the easier is to be faithful!
- In visualization, dimension 2 is the only choice.
- Heuristic criterion for the dimension choice: elbow criterion (no more gain), stability...

- Dimension Reduction is rarely used standalone but rather as a step in a predictive/prescriptive method.
- The dimension becomes an hyper-parameter of this method.

Representation Learning



Representation Learning

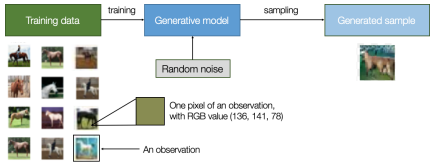
- How to transform arbitrary objects into numerical vectors?
- Objects: Categorical variables, Words, Images/Sounds...

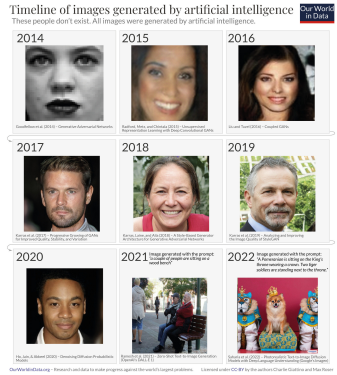
- The two previous dimension reduction approaches can be used (given possibly a first simple high dimensional representation)

Source: R. Khandelwal

Outline

Generative Modeling





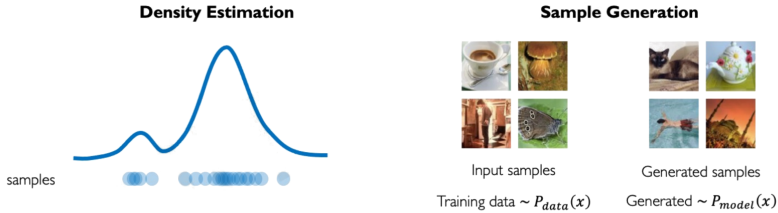
Generative Modeling

- Generate new samples similar to the ones in an original dataset.
- Generation may be conditioned by a input.

- Key for image generation. . . and chatbot!

Sources:: D. Foster/World Economic Forum

Density Estimation and Simulation



How can we learn $P_{model}(x)$ similar to $P_{data}(x)$?

- **Heuristic:** If we can estimate the (conditional) law $\mathbb{P}$ of the data and can simulate it, we can obtain new samples similar to the input ones.

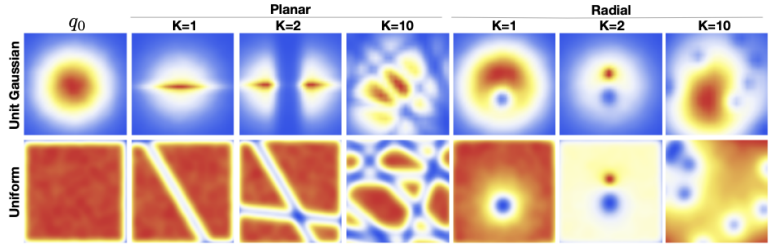
Estimation and Simulation

- How to estimate the density?
- How to simulate the estimate density?

- Other possibilities?

Source: Pipe Galera

Simple Estimation and Simple Simulation



Parametric Model, Image and Factorization

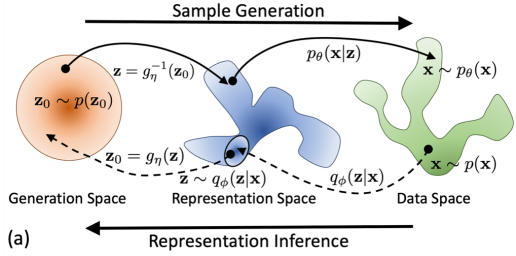
- Use
  - a simple parametric model,. . .
  - or the image of a parametric model (flow),. . .
  - or a factorization of a parametric model (recurrent model)

  as they are *simple* to estimate and to simulate.

- Estimation by Maximum Likelihood principle.
- Recurrent models are used in Large Language Models!

Source: Rezende et al.

Complex Estimation and Simple Simulation



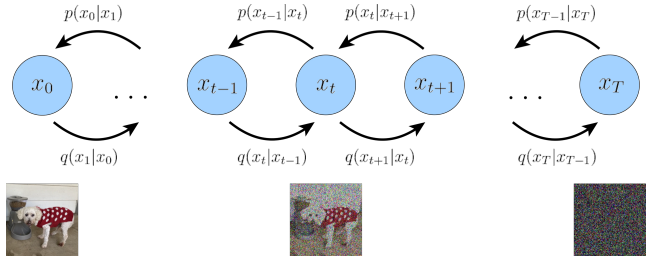
Latent Variable

- Introduce a latent variable Z from which X is easy to sample.
- Estimation based on approximate Maximum Likelihood (VAE/ELBO)

- The latent variable Z often lives in a smaller dimensional space.
- It can be generated by a simple method (or a more complex one. . .).

Source: S. Elhabian

Complex Estimation and Complex Simulation



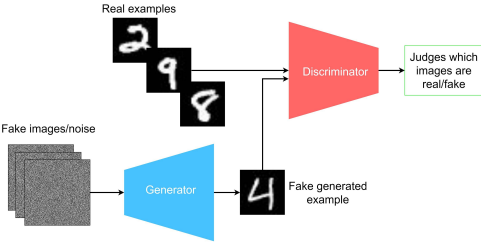
Monte Carlo Markov Chain

- Rely on much more complex probability model...
- which can only be simulated numerically.
- Often combined with noise injection to stabilizes the numerical scheme (Diffusion).

- Much more expensive to simulate than with Latent Variable approaches.

Source: Calvin Huo

Complex (non)Estimation and Simple Simulation



Generative Adversarial Network

- Bypass the density estimation problem, by transforming the problem into a competition between the generator and a discriminator.
- The better the generator, the harder it is for the generator to distinguish true samples from synthetic ones.
- No explicit density!

- Fast simulator but unstable training...

Source: IBM

Outline

Dimension Reduction

- **Training data** : $\mathcal{D} = \{\underline{X}_1, \ldots, \underline{X}_n\} \in \mathcal{X}^n \quad (\text{i.i.d. } \sim \mathbb{P})$
- Space $\mathcal{X}$ of possibly high dimension.

Dimension Reduction Map

- Construct a map Φ from the space $\mathcal{X}$ into a space $\mathcal{X}'$ of **smaller dimension**:

$$\begin{aligned} \Phi : \quad & \mathcal{X} \to \mathcal{X}' \\ & \underline{X} \mapsto \Phi(\underline{X}) \end{aligned}$$

Criterion

- Reconstruction error
- Relationship preservation

Outline

How to Simplify?

A Projection Based Approach

- Observations: $\underline{X}_1, \ldots, \underline{X}_n \in \mathbf{R}^d$
- Simplified version: $\Phi(\underline{X}_1), \ldots, \Phi(\underline{X}_n) \in \mathbf{R}^d$ with Φ an affine projection preserving the mean $\Phi(\underline{X}) = P(\underline{X} - m) + m$ with $P^\top = P = P^2$ and $m = \frac{1}{n}\sum_i \underline{X}_i$.

How to choose P?

- **Inertia criterion**:
$$\max_P \sum_{i,j} \|\Phi(\underline{X}_i) - \Phi(\underline{X}_j)\|^2 ?$$
- **Reconstruction criterion**:
$$\min_P \sum_i \|\underline{X}_i - \Phi(\underline{X}_i)\|^2 ?$$
- **Relationship criterion**:
$$\min_P \sum_{i,j} |(\underline{X}_i - m)^\top (\underline{X}_j - m) - (\Phi(\underline{X}_i) - m)^\top (\Phi(\underline{X}_j) - m)|^2 ?$$

- **Rk:** Best solution is $P = I$! Need to reduce the rank of the projection to $d' < d \ldots$

Inertia criterion

- **Heuristic:** a good representation is such that the projected points are far apart.

Two views on inertia

- Inertia:

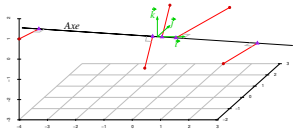
$$I = \frac{1}{2n^2} \sum_{i,j} \|\underline{X}_i - \underline{X}_j\|^2 = \frac{1}{n} \sum_{i=1}^{n} \|\underline{X}_i - m\|^2$$

- 2 times the mean squared distance to the mean = Mean squared distance between individual

Inertia criterion (Principal Component Analysis)

- Criterion: $\max_P \sum_{i,j} \frac{1}{2n^2} \|P\underline{X}_i - P\underline{X}_j\|^2 = \max_P \frac{1}{n} \sum_i \|P\underline{X}_i - m\|^2$
- **Solution:** Choose P as a projection matrix on the space spanned by the d' first eigenvectors of $\Sigma = \frac{1}{n} \sum_i (\underline{X}_i - m)(\underline{X}_i - m)^\top$

First Component of the PCA



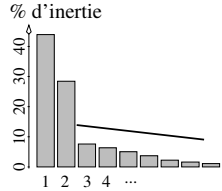
- $\underline{\widetilde{X}} = m + a^\top(\underline{X} - m)a$ with $\|a\| = 1$
- Inertia: $\frac{1}{n}\sum_{i=1}^{n} a^\top(\underline{X}_i - m)(\underline{X}_i - m)^\top a$

Principal Component Analysis: optimization of the projection

- Maximization of $\widetilde{I} = \frac{1}{n}\sum_{i=1}^{n} a^\top(\underline{X}_i - m)(\underline{X}_i - m)^\top a = a^\top \Sigma a$ with

  $\Sigma = \frac{1}{n}\sum_{i=1}^{n}(\underline{X}_i - m)(\underline{X}_i - m)^\top$ the empirical covariance matrix.
- Explicit optimal choice given by the eigenvector of the largest eigenvalue of Σ.

Source: E. Matzner-Løber



Principal Component Analysis : sequential optimization of the projection

- Explicit optimal solution obtain by the projection on the eigenvectors of the largest eigenvalues of Σ.
- Projected inertia given by the sum of those eigenvalues.

- Often fast decay of the eigenvalues: some dimensions are much more important than others.
- Not exactly the curse of dimensionality setting...
- Yet a lot of *small* dimension can drive the distance!

Reconstruction Criterion

- **Heuristic:** a good representation is such that the projected points are close to the original ones.

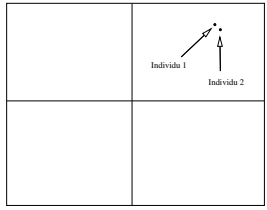
Reconstruction Criterion

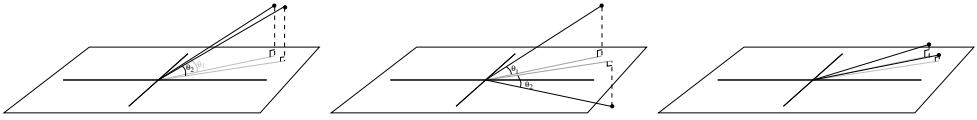
- Criterion: $\min_P \sum_i \frac{1}{n} \|\underline{X}_i - (P(\underline{X}_i - m) + m)\|^2 = \min_P \frac{1}{n} \sum_i \|(I - P)(\underline{X}_i - m)\|^2$
- **Solution:** Choose P as a projection matrix on the space spanned by the d' first eigenvectors of $\Sigma = \frac{1}{n} \sum_i (\underline{X}_i - m)(\underline{X}_i - m)^\top$

- Same solution with a different heuristic!
- Proof (Pythagora):

$$\sum_i \|\underline{X}_i - m\|^2 = \sum_i \left(\|P(\underline{X}_i - m)\|^2 + \|(I - P)(\underline{X}_i - m)\|^2 \right)$$

PCA, Reconstruction and Distances





Close projection doesn't mean close individuals!

- Same projections but different situations.
- Quality of the reconstruction measured by the angle with the projection space!

Source: E. Matzner-Løber

Relationship Criterion

- **Heuristic:** a good representation is such that the projected points scalar products are similar to the original ones.

Relationship Criterion (Multi Dimensional Scaling)

- Criterion: $\min_P \sum_{i,j} |(\underline{X}_i - m)^\top (\underline{X}_j - m) - (\Phi(\underline{X}_i) - m)^\top (\Phi(\underline{X}_j) - m)|^2$
- **Solution:** Choose P as a projection matrix on the space spanned by the d' first eigenvectors of $\Sigma = \frac{1}{n} \sum_i (\underline{X}_i - m)(\underline{X}_i - m)^\top$

- Same solution with a different heuristic!
- Much more involved justification!

Link with SVD

- PCA model: $\underline{X} - m \simeq P(\underline{X} - m)$
- **Prop:** $P = VV^\top$ with V an orthormal family in dimension d of size d'.
- PCA model with V: $\underline{X} - m \simeq VV^\top(\underline{X} - m)$ where $\tilde{\underline{X}} = V^\top(\underline{X} - m) \in \mathbb{R}^{d'}$
- Row vector rewriting: $\underline{X}^\top - m^\top \simeq \tilde{\underline{X}}^\top V^\top$

Matrix Rewriting and Low Rank Factorization

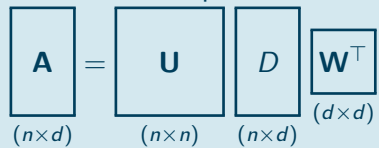
- Matrix rewriting

$$\begin{bmatrix} \underline{X}_1^\top - m^\top \\ \vdots \\ \underline{X}_n^\top - m^\top \end{bmatrix}_{(n\times d)} \simeq \begin{bmatrix} \tilde{\underline{X}}_1^\top \\ \vdots \\ \tilde{\underline{X}}_n^\top \end{bmatrix}_{(n\times d')} \boxed{\mathbf{V}^\top}_{(d'\times d)}$$

- Low rank matrix factorization! (Truncated SVD solution...)

SVD Decomposition

- Any matrix $n \times d$ matrix A can be decomposed as



with U and W two orthonormal matrices and D a *diagonal* matrix with decreasing values.

Low Rank Approximation

- The best low rank approximation or rank r is obtained by restriction of the matrices to the first r dimensions:

$$\underset{(n\times d)}{\boxed{\mathbf{A}}} \simeq \underset{(n\times r)}{\boxed{\mathbf{U_r}}} \underset{(r\times r)}{\boxed{D_{r,r}}} \underset{(r\times d)}{\boxed{\mathbf{W_r}^\top}}$$

  for both the operator norm and the Frobenius norm!

- PCA: Low rank approximation with Frobenius norm, $d' = r$ and

$$\begin{pmatrix} \underline{X}_1^\top - m^\top \\ \vdots \\ \underline{X}_n^\top - m^\top \end{pmatrix} \leftrightarrow A, \quad \begin{pmatrix} \underline{\tilde{X}}_1^\top \\ \vdots \\ \underline{\tilde{X}}_n^\top \end{pmatrix} \leftrightarrow \mathbf{U_r} D_{r,r}, \quad \mathbf{V}^\top \leftrightarrow \mathbf{W_r}^\top$$

SVD Decompositions

- Recentered data:
$$\mathbf{R} = \begin{pmatrix} \underline{X}_1^\top - m^\top \\ \vdots \\ \underline{X}_n^\top - m^\top \end{pmatrix} = UDW^\top$$
- Covariance matrix:
$$\Sigma = \mathbf{R}^\top \mathbf{R} = WD^\top DW$$
with $D^\top D$ diagonal.
- Gram matrix (matrix of scalar products):
$$G = \mathbf{R}\mathbf{R}^\top = UDD^\top U$$
with $DD^\top$ diagonal.

- Those are the same U, W and D, hence the link between all the approaches.

Outline

Reconstruction Error Approach

Goal

- Construct a map Φ from the space $\mathcal{X}$ into a space $\mathcal{X}'$ of **smaller dimension**:

$$\Phi : \quad \mathcal{X} \to \mathcal{X}'$$
$$\underline{X} \mapsto \Phi(\underline{X})$$

- Construct $\widetilde{\Phi}$ from $\mathcal{X}'$ to $\mathcal{X}$
- Control the error between $\underline{X}$ and its reconstruction $\widetilde{\Phi}(\Phi(\underline{X}))$

- Canonical example for $\underline{X} \in \mathbb{R}^d$: find Φ and $\widetilde{\Phi}$ in a parametric family that minimize

$$\frac{1}{n}\sum_{i=1}^{n} \|\underline{X}_i - \widetilde{\Phi}(\Phi(\underline{X}_i))\|^2$$

Principal Component Analysis

- $\mathcal{X} \in \mathbb{R}^d$ and $\mathcal{X}' = \mathbb{R}^{d'}$
- Affine model $\underline{X} \sim m + \sum_{l=1}^{d'} \underline{X}'^{(l)} V^{(l)}$ with $(V^{(l)})$ an orthonormal family.
- Equivalent to:
$$\Phi(\underline{X}) = V^\top(\underline{X} - m) \quad \text{and} \quad \widetilde{\Phi}(\underline{X}') = m + V\underline{X}'$$
- Reconstruction error criterion:
$$\frac{1}{n}\sum_{i=1}^{n} \|\underline{X}_i - (m + VV^\top(\underline{X}_i - m)\|^2$$
- **Explicit solution:** m is the empirical mean and V is any orthonormal basis of the space spanned by the d' first eigenvectors (the one with largest eigenvalues) of the empirical covariance matrix $\frac{1}{n}\sum_{i=1}^{n}(\underline{X}_i - m)(\underline{X}_i - m)^\top$.

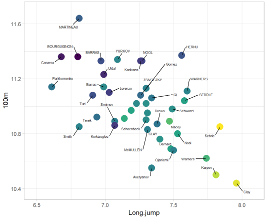
Principal Component Analysis

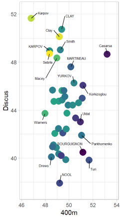
PCA Algorithm

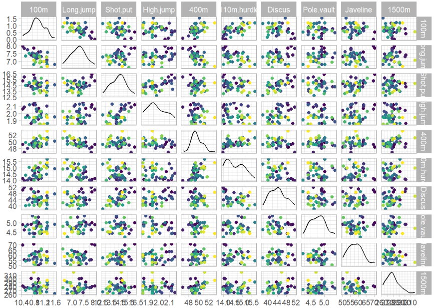
- Compute the empirical mean $m = \frac{1}{n}\sum_{i=1}^{n} \underline{X}_i$
- Compute the empirical covariance matrix $\frac{1}{n}\sum_{i=1}^{n}(\underline{X}_i - m)(\underline{X}_i - m)^\top$.
- Compute the d' first eigenvectors of this matrix: $V^{(1)}, \ldots, V^{(d')}$
- Set $\Phi(\underline{X}) = V^\top(\underline{X} - m)$

- Complexity: $O(n(d + d^2) + d'd^2)$
- Interpretation:
  - $\Phi(\underline{X}) = V^\top(\underline{X} - m)$: coordinates in the restricted space.
  - $V^{(i)}$: influence of each original coordinates in the ith new one.
- **Scaling:** This method is not invariant to a scaling of the variables! It is custom to normalize the variables (at least within groups) before applying PCA.

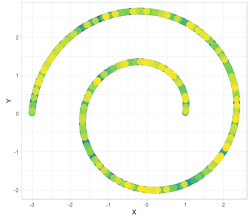
Decathlon

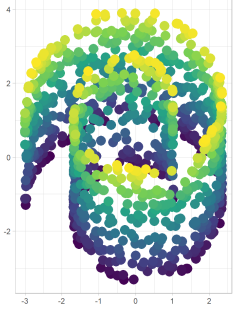

100m
Long.jump


Discus
400m

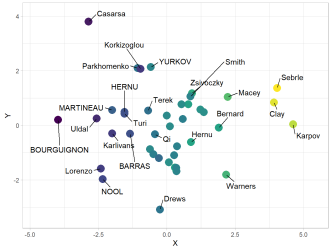

100m
Long.jump
Shot.put
High.jump
400m
Discus
Pole.vault
Javeline
1500m

Swiss Roll

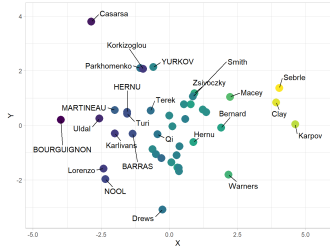




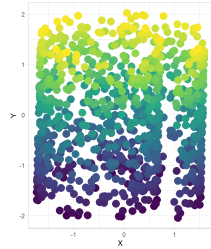
Principal Component Analysis



Decathlon



Decathlon
Renormalized



Swiss Roll

Multiple Factor Analysis

- PCA assumes $\mathcal{X} = \mathbb{R}^d$!
- How to deal with categorical values?
- MFA = PCA with clever coding strategy for categorical values.

Categorical value code for a single variable

- Classical redundant dummy coding:
$$\underline{X} \in \{1, \dots, V\} \mapsto P(\underline{X}) = (\mathbf{1}_{\underline{X}=1}, \dots, \mathbf{1}_{\underline{X}=V})^\top$$
- Compute the mean (i.e. the empirical proportions): $\overline{P} = \frac{1}{n} \sum_{i=1}^{n} P(\underline{X}_i)$
- *Renormalize* $P(\underline{X})$ by $1/\sqrt{(V-1)\overline{P}}$:
$$P(\underline{X}) = (\mathbf{1}_{\underline{X}=1}, \dots \mathbf{1}_{\underline{X}=V}) \mapsto \left(\frac{\mathbf{1}_{\underline{X}=1}}{\sqrt{(V-1)\overline{P}_1}}, \dots, \frac{\mathbf{1}_{\underline{X}=V}}{\sqrt{(V-1)\overline{P}_V}} = P^r(\underline{X}) \right)$$

- χ^2 type distance!

Multiple Factor Analysis

- PCA becomes the minimization of

$$\frac{1}{n}\sum_{i=1}^{n}\|P^r(\underline{X}_i) - (m + VV^\top(P^r(\underline{X}_i) - m))\|^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}\sum_{v=1}^{V}\frac{\left|\mathbf{1}_{\underline{X}_i=v} - (m' + \sum_{l=1}^{d'} V^{(l)\top}(P(\underline{X}_i) - m')V^{(l,v)})\right|^2}{(V-1)\overline{P}_v}$$

- Interpretation:
  - $m' = \overline{P}$
  - $\Phi(\underline{X}) = V^\top(P^r(\underline{X}) - m)$: coordinates in the restricted space.
  - $V^{(l)}$ can be interpreted s as a probability profile.
- Complexity: $O(n(V + V^2) + d'V^2)$
- Link with Correspondence Analysis (CA)

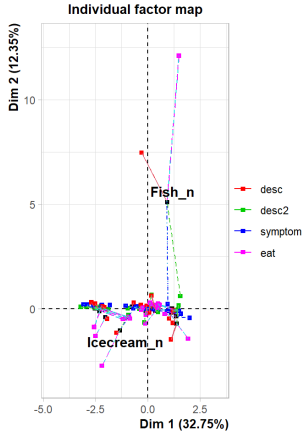
Multiple Factor Analysis

MFA Algorithm

- Redundant dummy coding of each categorical variable.
- Renormalization of each block of dummy variable.
- Classical PCA algorithm on the resulting variables

- Interpretation as a reconstruction error with a rescaled/χ^2 metric.
- Interpretation:
  - $\Phi(\underline{X}) = V^\top(P^r(\underline{X}) - m)$: coordinates in the restricted space.
  - $V^{(l)}$: influence of each modality/variable in the ith new coordinates.
- **Scaling:** This method is not invariant to a scaling of the continuous variables! It is custom to normalize the variables (at least within groups) before applying PCA.

Multiple Factor Analysis



PCA Model

- PCA: Linear model assumption

$$\underline{X} \simeq m + \sum_{l=1}^{d'} \underline{X}^{\prime,(l)} V^{(l)} = m + V\underline{X}'$$

- with
  - $V^{(l)}$ orthonormal
  - $\underline{X}^{\prime,(l)}$ without constraints.

- Two directions of extension:
  - Other constraints on V (or the coordinates in the restricted space): ICA, NMF, Dictionary approach
  - PCA on a non-linear image of $\underline{X}$: kernel-PCA
- Much more complex algorithm!

Non Linear PCA

ICA (Independent Component Analysis)

- Linear model assumption

$$\underline{X} \simeq m + \sum_{l=1}^{d'} \underline{X}'^{,(l)} V^{(l)} = m + V\underline{X}'$$

- with
  - $V^{(l)}$ without constraints.
  - $\underline{X}'^{,(l)}$ *independent*

NMF (Non Negative Matrix Factorization)

- (Linear) Model assumption

$$\underline{X} \simeq \sum_{l=1}^{d'} \underline{X}'^{,(l)} V^{(l)} = V\underline{X}'$$

- with
  - $V^{(l)}$ non-negative
  - $\underline{X}'^{,(l)}$ non-negative.

Non Linear PCA

Dictionary

- (Linear) Model assumption

$$\underline{X} \simeq m + \sum_{l=1}^{d'} \underline{X}^{\prime,(l)} V^{(l)} = m + V\underline{X}'$$

- with
  - $V^{(l)}$ without constraints
  - $\underline{X}'$ sparse (with a lot of 0)

kernel PCA

- Linear model assumption

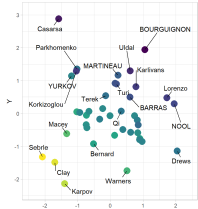
$$\Psi(\underline{X} - m) \simeq \sum_{l=1}^{d'} \underline{X}^{\prime,(l)} V^{(l)} = V\underline{X}'$$

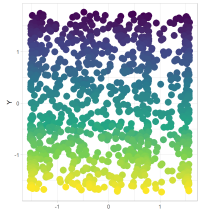
- with
  - $V^{(l)}$ orthonormal
  - $\underline{X}'_l$ without constraints.

Non Linear PCA

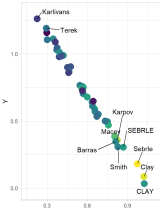
Decathlon

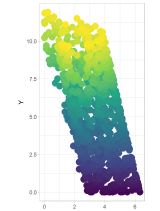
Swiss Roll



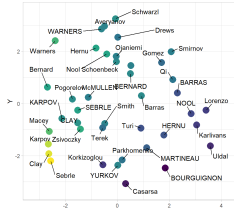


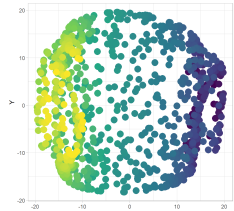
ICA





NMF





Kernel PCA

Deep Auto Encoder

- Construct a map Φ with a **NN** from the space $\mathcal{X}$ into a space $\mathcal{X}'$ of smaller dimension:

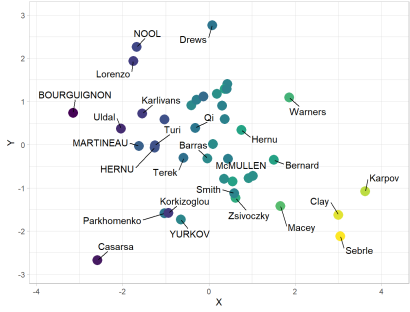
$$\begin{aligned} \Phi : \quad & \mathcal{X} \to \mathcal{X}' \\ & \underline{X} \mapsto \Phi(\underline{X}) \end{aligned}$$

- Construct $\widetilde{\Phi}$ with a **NN** from $\mathcal{X}'$ to $\mathcal{X}$
- Control the error between $\underline{X}$ and its reconstruction $\widetilde{\Phi}(\Phi(\underline{X}))$:

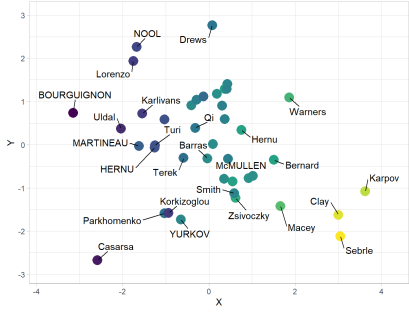
$$\frac{1}{n}\sum_{i=1}^{n} \|\underline{X}_i - \widetilde{\Phi}(\Phi(\underline{X}_i))\|^2$$

- Optimization by gradient descent.
- NN can be replaced by another parametric function...

Deep Auto Encoder



Shallow Auto Encoder
(PCA)



Deep Auto Encoder

Outline

Pairwise Relation

- Different point of view!
- Focus on pairwise relation $\mathcal{R}(\underline{X}_i, \underline{X}_j)$.

Distance Preservation

- Construct a map Φ from the space $\mathcal{X}$ into a space $\mathcal{X}'$ of **smaller dimension**:

$$\begin{aligned} \Phi : \quad & \mathcal{X} \to \mathcal{X}' \\ & \underline{X} \mapsto \Phi(\underline{X}) = \underline{X}' \end{aligned}$$

- such that

$$\mathcal{R}(\underline{X}_i, \underline{X}_j) \sim \mathcal{R}'(\underline{X}'_i, \underline{X}'_j)$$

- Most classical version (MDS):
  - Scalar product relation: $\mathcal{R}(\underline{X}_i, \underline{X}_j) = (\underline{X}_i - m)^\top (\underline{X}_j - m)$
  - Linear mapping $\underline{X}' = \Phi(\underline{X}) = V^\top (\underline{X} - m)$.
  - Euclidean scalar product matching:

$$\frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \left| (\underline{X}_i - m)^\top (\underline{X}_j - m) - (\underline{X}'_i)^\top \underline{X}'_j \right|^2$$

- Φ often defined only on $\mathcal{D}\ldots$

MultiDimensional Scaling

MDS Heuristic

- Match the *scalar* products:
$$\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left|(\underline{X}_i - m)^\top(\underline{X}_j - m) - \underline{X}_i{}'^\top \underline{X}'_j\right|^2$$
- Linear method: $\underline{X}' = U^\top(\underline{X} - m)$ with U orthonormal

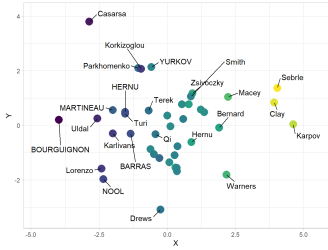
- **Beware:** $\underline{X}$ can be unknown, only the scalar products are required!
- Resulting criterion: minimization in $U^\top(\underline{X}_i - m)$ of
$$\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left|(\underline{X}_i - m)^\top(\underline{X}_j - m) - (\underline{X}_i - m)^\top UU^\top(\underline{X}_j - m)\right|^2$$
without using explicitly $\underline{X}$ in the algorithm...
- Explicit solution obtained through the eigendecomposition of the know Gram matrix $(\underline{X}_i - m)^\top(\underline{X}_j - m)$ by keeping only the d' largest eigenvalues.

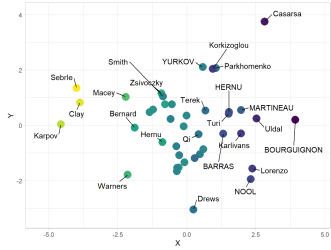
MultiDimensional Scaling

- In this case, MDS yields the same result as the PCA (but with different inputs, distance between observation vs correlations)!
- **Explanation:** Same SVD problem up to a transposition:
  - MDS
  $$\overline{X}_{(n)}{}^{\top}\overline{X}_{(n)} \sim \overline{X}_{(n)}{}^{\top} U U^{\top} \overline{X}_{(n)}$$
  - PCA
  $$\overline{X}_{(n)}\overline{X}_{(n)}{}^{\top} \sim U^{\top}\overline{X}_{(n)}\overline{X}_{(n)}{}^{\top} U$$
- Complexity: PCA $O((n+d')d^2)$ vs MDS $O((d+d')n^2)\ldots$

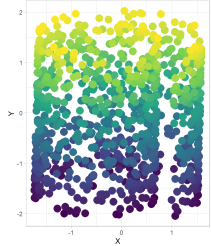
MultiDimensional Scaling

Decathlon

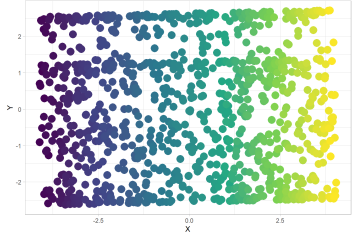




Swiss Roll



PCA



MDS

Generalized MDS

- Preserving the scalar products amounts to preserve the Euclidean distance.
- Easier **generalization** if we work in terms of distance!

Generalized MDS

- Generalized MDS:
  - Distance relation: $\mathcal{R}(\underline{X}_i, \underline{X}_j) = d(\underline{X}_i, \underline{X}_j)$
  - Linear mapping $\underline{X}' = \Phi(\underline{X}) = V^\top(\underline{X} - m)$.
  - Euclidean matching:

$$\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left|d(\underline{X}_i, \underline{X}_j) - d'(\underline{X}'_i, \underline{X}'_j)\right|^2$$

- Strong connection (but no equivalence) with MDS when $d(x, y) = \|x - y\|^2$!
- **Minimization:** Simple gradient descent can be used (can be stuck in local minima).

ISOMAP

- MDS: equivalent to PCA (but more expensive) if $d(x, y) = \|x - y\|^2$!
- ISOMAP: use a *localized* distance instead to limit the influence of very far point.

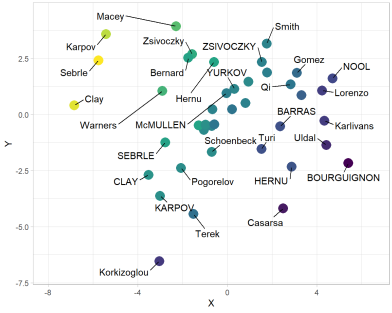
ISOMAP

- For each point $\underline{X}_i$, define a neighborhood $\mathcal{N}_i$ (either by a distance or a number of points) and let

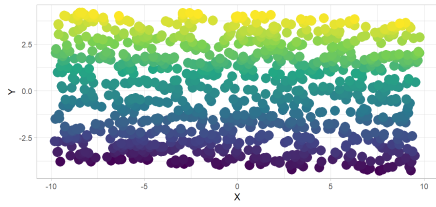
$$d_0(\underline{X}_i, \underline{X}_j) = \begin{cases} +\infty & \text{if } \underline{X}_j \notin \mathcal{N}_i \\ \|\underline{X}_i - \underline{X}_j\|^2 & \text{otherwise} \end{cases}$$

- Compute the shortest path distance for each pair.
- Use the MDS algorithm with this distance

ISOMAP



Decathlon



Swiss Roll

Random Projection

Random Projection Heuristic

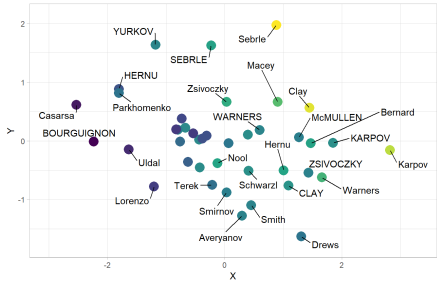
- Draw at random d' unit vector (direction) U_i.
- Use $\underline{X}' = U^\top(\underline{X} - m)$ with $m = \frac{1}{n}\sum_{i=1}^n \underline{X}_i$

- **Property:** If $\underline{X}$ lives in a space of dimension d'', then, as soon as, $d' \sim d'' \log(d'')$,

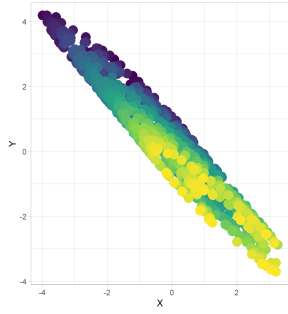
$$\|\underline{X}_i - \underline{X}_j\|^2 \sim \frac{d}{d'}\|\underline{X}'_i - \underline{X}'_j\|^2$$

- Do not really use the data!

Random Projection



Decathlon



Swiss Roll

t-Stochastic Neighbor Embedding

SNE heuristic

- From $\underline{X}_i \in \mathcal{X}$, construct a set of conditional probability:

$$P_{j|i} = \frac{e^{-\|\underline{X}_i - \underline{X}_j\|^2/2\sigma_i^2}}{\sum_{k \neq i} e^{-\|\underline{X}_i - \underline{X}_k\|^2/2\sigma_i^2}} \qquad P_{i|i} = 0$$

- Find $\underline{X}'_i$ in $\mathbb{R}^{d'}$ such that the set of conditional probability:

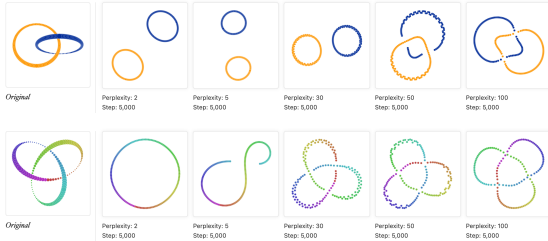
$$Q_{j|i} = \frac{e^{-\|\underline{X}'_i - \underline{X}'_j\|^2/2\sigma_i^2}}{\sum_{k \neq i} e^{-\|\underline{X}'_i - \underline{X}'_k\|^2/2\sigma_i^2}} \qquad Q_{i|i} = 0$$

  is close from P.

- **t-SNE:** use a Student-t term $(1 + \|\underline{X}'_i - \underline{X}'_j\|^2)^{-1}$ for $\underline{X}'_i$
- Minimize the Kullback-Leibler divergence $(\sum_{i,j} P_{j|i} \log \frac{P_{j|i}}{Q_{j|i}})$ by a simple gradient descent (can be stuck in local minima).
- Parameters σ_i such that $H(P_i) = -\sum_{j=1}^{n} P_{j|i} \log P_{j|i} = \text{cst}$.

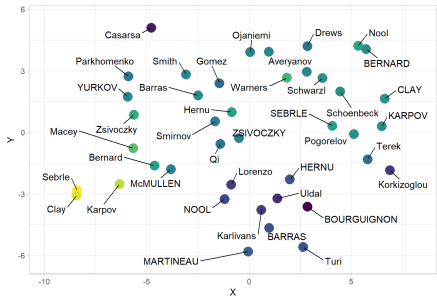
t-Stochastic Neighbor Embedding

- Very successful/ powerful technique in practice
- Convergence may be long, unstable, or strongly depending on parameters.
- See this distill post for many impressive examples

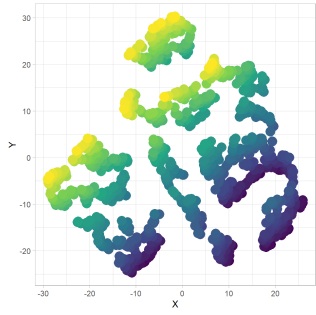


Representation depending on t-SNE parameters

t-SNE



Decathlon



Swiss Roll

UMAP

- Topological Data Analysis inspired.

Uniform Manifold Approximation and Projection

- Define a notion of asymmetric scaled local proximity between neighbors:
  - Compute the k-neighborhood of $\underline{X}_i$, its diameter σ_i and the distance ρ_i between $\underline{X}_i$ and its nearest neighbor.
  - Define

$$w_i(\underline{X}_i, \underline{X}_j) = \begin{cases} e^{-(d(\underline{X}_i, \underline{X}_j) - \rho_i)/\sigma_i} & \text{for } \underline{X}_j \text{ in the } k\text{-neighborhood} \\ 0 & \text{otherwise} \end{cases}$$

- Symmetrize into a *fuzzy* nearest neighbor criterion

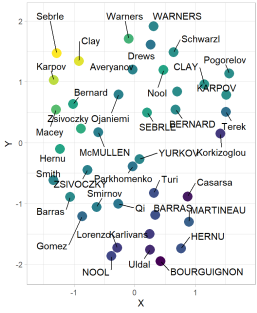
$$w(\underline{X}_i, \underline{X}_j) = w_i(\underline{X}_i, \underline{X}_j) + w_j(\underline{X}_j, \underline{X}_i) - w_i(\underline{X}_i, \underline{X}_j) w_j(\underline{X}_j, \underline{X}_i)$$

- Determine the points $\underline{X}'_i$ in a low dimensional space such that

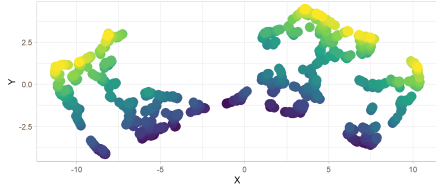
$$\sum_{i \neq j} w(\underline{X}_i, \underline{X}_j) \log \left(\frac{w(\underline{X}_i, \underline{X}_j)}{w'(\underline{X}'_i, \underline{X}'_j)} \right) + (1 - w(\underline{X}_i, \underline{X}_j)) \log \left(\frac{(1 - w(\underline{X}_i, \underline{X}_j))}{(1 - w'(\underline{X}'_i, \underline{X}'_j))} \right)$$

- Can be performed by local gradient descent.

UMAP



Decathlon



Swiss Roll

Graph based

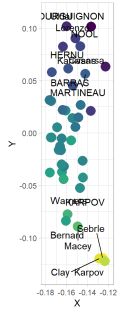
Graph heuristic

- Construct a graph with weighted edges $w_{i,j}$ measuring the *proximity* of $\underline{X}_i$ and $\underline{X}_j$ ($w_{i,j}$ large if close and 0 if there is no information).
- Find the points $\underline{X}'_i \in \mathbb{R}^{d'}$ minimizing

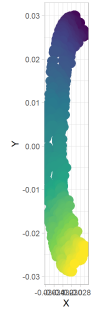
$$\frac{1}{n}\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}\|\underline{X}'_i - \underline{X}'_j\|^2$$

- Need of a constraint on the size of $\underline{X}'_i \ldots$
- Explicit solution through linear algebra: d' eigenvectors with smallest eigenvalues of the Laplacian of the graph $D - W$, where D is a diagonal matrix with $D_{i,i} = \sum_j w_{i,j}$.
- Variation on the definition of the Laplacian...

Graph



Decathlon



Swiss Roll

Outline

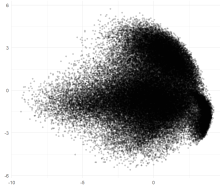
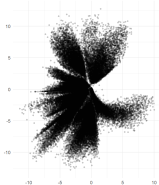
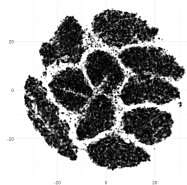
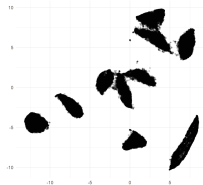
How to Compare Different Dimensionality Reduction Methods ?

- **Difficult!** Once again, the metric is very subjective.

However, a few possible attempts

- Did we preserve a lot of inertia with only a few directions?
- Do those directions *make sense* from an expert point of view?
- Do the low dimension representation *preserve* some important information?
- Are we better on **subsequent task**?

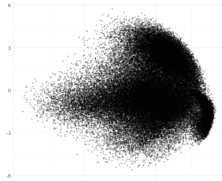
A Challenging Example: MNIST

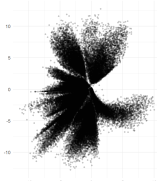
MNIST Dataset

- Images of 28×28 pixels.
- No label used!
- 4 different embeddings.

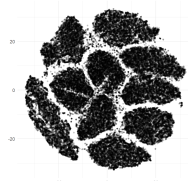
A Challenging Example: MNIST



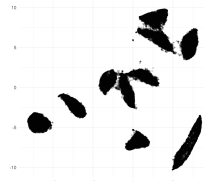
PCA



autoencoder



t-SNE

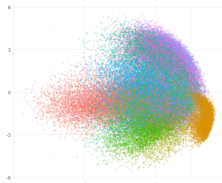


UMAP

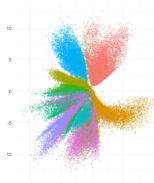
MNIST Dataset

- Images of 28×28 pixels.
- No label used!
- 4 different embeddings.

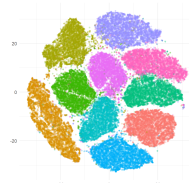
A Challenging Example: MNIST



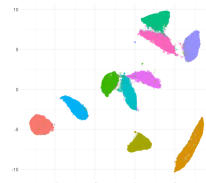
PCA



autoencoder



t-SNE



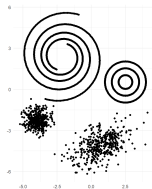
UMAP

MNIST Dataset

- Images of 28×28 pixels.
- No label used!
- 4 different embeddings.

- Quality evaluated by visualizing the true labels **not used to obtain the embeddings.**
- Only a few labels could have been used.

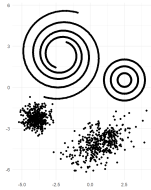
A Simpler Example: A 2D Set



Cluster Dataset

- Set of points in 2D.
- No label used!
- 3 different embeddings.

A Simpler Example: A 2D Set



PCA

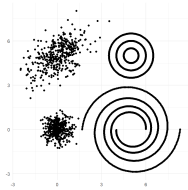
t-SNE

UMAP

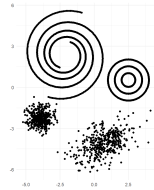
Cluster Dataset

- Set of points in 2D.
- No label used!
- 3 different embeddings.

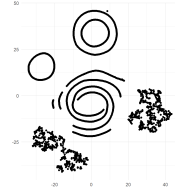
A Simpler Example: A 2D Set



Original



PCA



t-SNE

UMAP

Cluster Dataset

- Set of points in 2D.
- No label used!
- 3 different embeddings.

- Quality evaluated by stability...

Outline

Clustering

- **Training data** : $\mathcal{D} = \{\underline{X}_1, \ldots, \underline{X}_n\} \in \mathcal{X}^n \quad (\text{i.i.d. } \sim \mathbb{P})$
- Latent groups?

Clustering

- Construct a map f from $\mathcal{D}$ to $\{1, \ldots, K\}$ where K is a number of classes to be fixed:

$$f : \quad \underline{X}_i \mapsto k_i$$

Motivations

- Interpretation of the groups
- Use of the groups in further processing

- Several strategies possible!
- Can use dimension reduction as a preprocessing.

Outline

Partition Based

Partition Heuristic

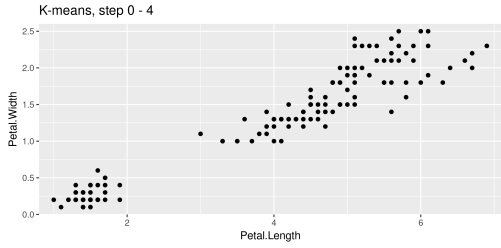
- Clustering is defined by a partition in K classes...
- that minimizes a homogeneity criterion.

K- Means

- Cluster k defined by a *center* μ_k.
- Each sample is associated to the closest center.
- Centers defined as the minimizer of $\sum_{i=1}^{n} \min_k \|\underline{X}_i - \mu_k\|^2$

- Iterative scheme (Loyd):
  - Start by a (pseudo) random choice for the centers μ_k
  - Assign each samples to its nearby center
  - Replace the center of a cluster by the mean of its assigned samples.
  - Repeat the last two steps until convergence.

Partition Based



Partition based

- Other schemes:
  - McQueen: modify the mean each time a sample is assigned to a new cluster.
  - Hartigan: modify the mean by removing the considered sample, assign it to the nearby center and recompute the new mean after assignment.

A good initialization is crucial!

- Initialize by samples.
- k-Mean++: try to take them as separated as possible.
- No guarantee to converge to a global optimum: repeat and keep the best result!

- Complexity : $O(n \times K \times T)$ where T is the number of steps in the algorithm.

Partition based

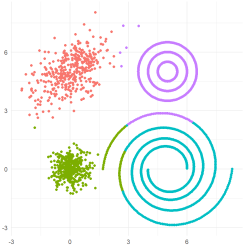
- k-Medoid: use a sample as a center
  - PAM: for a given cluster, use the sample that minimizes the intra distance (sum of the squared distance to the other points)
  - Approximate medoid: for a given cluster, assign the point that is the closest to the mean.

Complexity

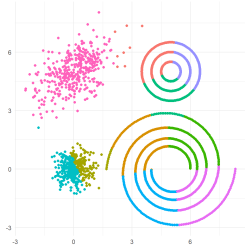
- PAM: $O(n^2 \times T)$ in the worst case!
- Approximate medoid: $O(n \times K \times T)$ where T is the number of steps in the algorithm.

- **Remark:** Any distance can be used... but the complexity of computing the centers can be very different.

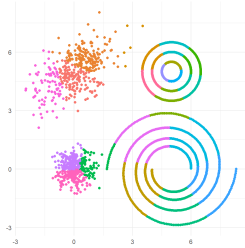
K-Means



$k = 4$



$k = 10$



$k = 10$

Model Based

Model Heuristic

- Use a generative model of the data:

$$\mathbb{P}(\underline{X}) = \sum_{k=1}^{K} \pi_k \mathbb{P}_{\theta_k}(\underline{X}|k)$$

  where π_k are proportions and $\mathbb{P}_\theta(\underline{X}|k)$ are parametric probability models.
- Estimate those parameters (often by a ML principle).
- Assign each observation to the class maximizing the a posteriori probability (obtained by Bayes formula)

$$\frac{\widehat{\pi_k}\mathbb{P}_{\widehat{\theta_k}}(\underline{X}|k)}{\sum_{k'=1}^{K}\widehat{\pi_{k'}}\mathbb{P}_{\widehat{\theta_{k'}}}(\underline{X}|k')}$$

- Link with Generative model in supervised classification!

Model Based

- Large choice of parametric models.

Gaussian Mixture Model

- Use
$$\mathbb{P}_{\theta_k}\left(\vec{\underline{X}}|k\right) \sim \mathcal{N}(\mu_k, \Sigma_k)$$
with $\mathcal{N}(\mu, \Sigma)$ the Gaussian law of mean μ and covariance matrix Σ.

- Efficient optimization algorithm available (EM)
- Often some constraint on the covariance matrices: identical, with a similar structure...
- Strong connection with K-means when the covariance matrices are assumed to be the same multiple of the identity.

Model Based

Probabilistic latent semantic analysis (PLSA)

- Documents described by their word counts w
- Model:

$$\mathbb{P}(w) = \sum_{k=1}^{K} \pi_k \mathbb{P}_{\theta_k}(w|k)$$

with k the (hidden) topic, π_k a topic probability and $\mathbb{P}_{\theta_k}(w|k)$ a multinomial law for a given topic.

- Clustering according to

$$\mathbb{P}(k|w) = \frac{\widehat{\pi_k}\mathbb{P}_{\widehat{\theta_k}}(w|k)}{\sum_{k'} \widehat{\pi_{k'}}\mathbb{P}_{\widehat{\theta_{k'}}}(w|k')}$$

- Same idea than GMM!
- Bayesian variant called LDA.

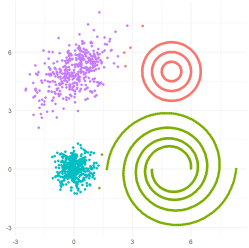
Model Based

Parametric Density Estimation Principle

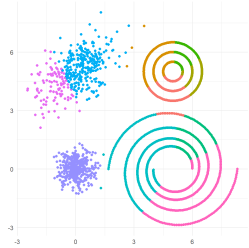
- Assign a probability of membership.
- Lots of theoretical studies...
- Model selection principle can be used to select K the number of classes (or rather to avoid using a nonsensical K...):
  - AIC / BIC / MDL penalization
  - Cross Validation is also possible!

- Complexity: $O(n \times K \times T)$

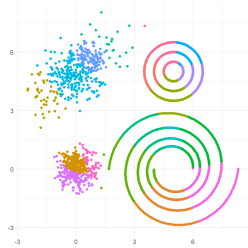
Gaussian Mixture Models



$k = 4$



$k = 10$



$k = 10$

Outline

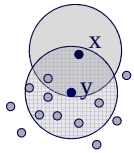
(Non Parametric) Density Based

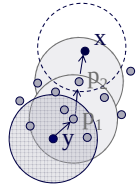
Density Heuristic

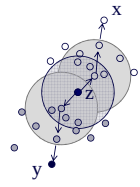
- Cluster are connected dense zone separated by low density zone.
- Not all points belong to a cluster.

- Basic bricks:
  - Estimate the density.
  - Find points with high densities.
  - Gather those points according to the density.
- Density estimation:
  - Classical kernel density estimators...
- Gathering:
  - Link points of high density and use the resulted component.
  - Move them toward top of density *hill* by following the gradient and gather all the points arriving at the same *summit*.

(Non Parametric) Density Based





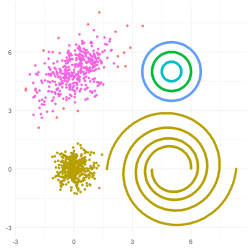


Examples

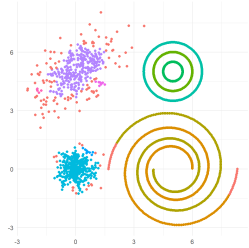
- DBSCAN: link point of high densities using a very simple kernel.
- PdfCLuster: find connected zone of high density.
- Mean-shift: move points toward top of density *hill* following an evolving kernel density estimate.

- Complexity: $O(n^2 \times T)$ in the worst case.
- Can be reduced to $O(n \log(n) T)$ if samples can be encoded in a tree structure (n-body problem type approximation).

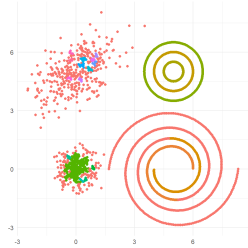
DBSCAN



$\epsilon = .45$



$\epsilon = .2$



$\epsilon = .1$

Outline

Agglomerative Clustering

Agglomerative Clustering Heuristic

- Start with very small clusters (a sample by cluster?)
- Sequential merging of the most similar clusters...
- according to some *greedy* criterion Δ.

- Generates a hierarchy of clustering instead of a single one.
- Need to select the number of cluster afterwards.
- Several choices for the merging criterion...
- Examples:
  - Minimum Linkage: merge the closest cluster in term of the usual distance
  - Ward's criterion: merge the two clusters yielding the less inner inertia loss (k-means criterion)

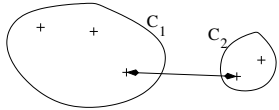
Agglomerative Clustering

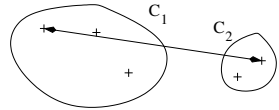
Algorithm

- Start with $(\mathcal{C}_i^{(0)}) = (\{X_i\})$ the collection of all singletons.
- At step s, we have $n - s$ clusters $(\mathcal{C}_i^{(s)})$:
  - Find the two most similar clusters according to a criterion Δ:
    $$(i, i') = \underset{(j,j')}{\operatorname{argmin}}\, \Delta(\mathcal{C}_j^{(s)}, \mathcal{C}_{j'}^{(s)})$$
  - Merge $\mathcal{C}_i^{(s)}$ and $\mathcal{C}_{i'}^{(s)}$ into $\mathcal{C}_i^{(s+1)}$
  - Keep the $n - s - 2$ other clusters $\mathcal{C}_{i''}^{(s+1)} = \mathcal{C}_{i''}^{(s)}$
- Repeat until there is only one cluster.

- Complexity: $O(n^3)$ in general.
- Can be reduced to $O(n^2)$
  - if only a bounded number of merging is possible for a given cluster,
  - for the most classical distances by maintaining a nearest neighbors list.

Agglomerative Clustering





Merging criterion based on the distance between points

- Minimum linkage:

$$\Delta(\mathcal{C}_i, \mathcal{C}_j) = \min_{\underline{X}_i \in \mathcal{C}_i} \min_{\underline{X}_\in \mathcal{C}_j} d(\underline{X}_i, \underline{X}_j)$$

- Maximum linkage:

$$\Delta(\mathcal{C}_i, \mathcal{C}_j) = \max_{\underline{X}_i \in \mathcal{C}_i} \max_{\underline{X}_\in \mathcal{C}_j} d(\underline{X}_i, \underline{X}_j)$$

- Average linkage:

$$\Delta(\mathcal{C}_i, \mathcal{C}_j) = \frac{1}{|\mathcal{C}_i||\mathcal{C}_j|} \sum_{\underline{X}_i \in \mathcal{C}_i} \sum_{\underline{X}_\in \mathcal{C}_j} d(\underline{X}_i, \underline{X}_j)$$

- Clustering based on the proximity...

Source: E. Matzner-Löber

Agglomerative Clustering

Merging criterion based on the inertia (distance to the mean)

- Ward's criterion:

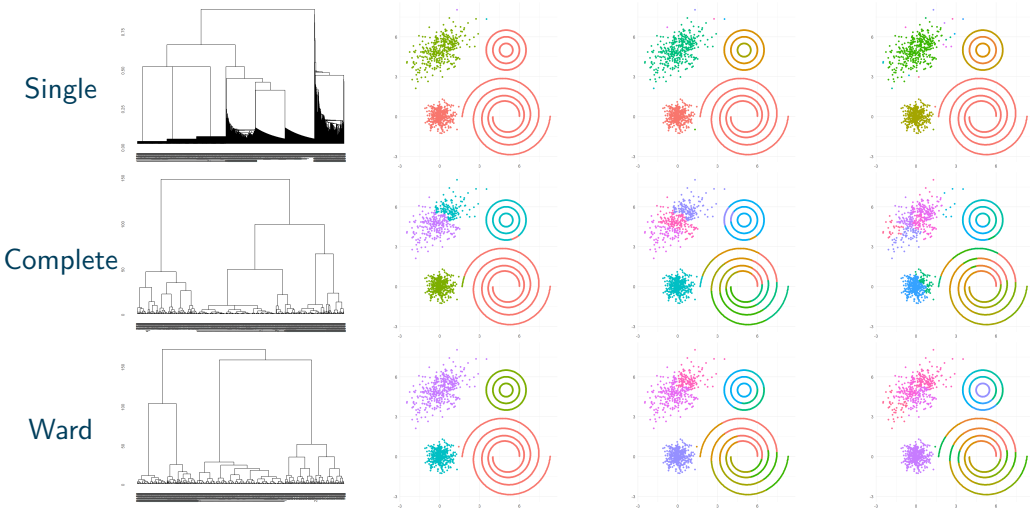
$$\Delta(\mathcal{C}_i, \mathcal{C}_j) = \sum_{\underline{X}_i \in \mathcal{C}_i} \left(d^2(\underline{X}_i, \mu_{\mathcal{C}_i \cup \mathcal{C}_j}) - d^2(\underline{X}_i, \mu_{\mathcal{C}_i}) \right) + \sum_{\underline{X}_j \in \mathcal{C}_j} \left(d^2(\underline{X}_j, \mu_{\mathcal{C}_i \cup \mathcal{C}_j}) - d^2(\underline{X}_j, \mu_{\mathcal{C}_j}) \right)$$

- If d is the Euclidean distance:

$$\Delta(\mathcal{C}_i, \mathcal{C}_j) = \frac{2|\mathcal{C}_i||\mathcal{C}_j|}{|\mathcal{C}_i| + |\mathcal{C}_j|} d^2(\mu_{\mathcal{C}_i}, \mu_{\mathcal{C}_j})$$

- Same criterion than in the k-means algorithm but greedy optimization.

Agglomerative Clustering



Outline

Grid based

Grid heuristic

- Split the space in pieces
- Group those of high density according to their proximity

- Similar to density based estimate (with partition based initial clustering)
- Space splitting can be fixed or adaptive to the data.
- Examples:
  - STING (Statistical Information Grid): Hierarchical tree construction plus DBSCAN type algorithm
  - AMR (Adaptive Mesh Refinement): Adaptive tree refinement plus k-means type assignment from high density leaves.
  - CLIQUE: Tensorial grid and 1D detection.
- Linked to Divisive clustering (DIANA)

Others

Graph based

- Spectral clustering: dimension reduction + k-means.
- Message passing: iterative local algorithm.
- Graph cut: min/max flow.

- Kohonen Map,
- ...

Outline

Scalability

Large dataset issue

- When n is large, a $O(n^\alpha \log n)$ with $\alpha > 1$ is not acceptable!
- How to deal with such a situation?

- **Beware:** Computing all the pairwise distance requires $O(n^2)$ operations!

Ideas

- Sampling
- Online processing
- Simplification
- Parallelization

Sampling

Sampling heuristic

- Use only a subsample to construct the clustering.
- Assign the other points to the constructed clusters afterwards.

- Requires a clustering method that can assign new points (partition, model...)
- Often repetition and choice of the best clustering
- Example:
  - CLARA: K-medoid with sampling and repetition
- Two-steps algorithm:
  - Generate a large number n' of clusters using a fast algorithm (with $n' \ll n$)
  - Cluster the clusters with a more accurate algorithm.

Online heuristic

- Modify the current clusters according to the value of a single observation.

- Requires compactly described clusters.
- Examples:
  - Add to an existing cluster (and modify it) if it is close enough and create a new cluster otherwise (k-means without reassignment)
  - Stochastic descent gradient (GMM)
- May leads to far from optimal clustering.

Simplification

Simplification heuristic

- Simplify the algorithm to be more efficient at the cost of some precision.

- Algorithm dependent!
- Examples:
  - Replace groups of observation (preliminary cluster) by the (approximate) statistics.
  - Approximate the distances by cheaper ones.
  - Use n-body type techniques.

Parallelization

Parallelization heuristic

- Split the computation on several computers.

- Algorithm dependent!
- Examples:
  - Distance computation in k-means, parameter gradient in model based clustering
  - Grid density estimation, Space splitting strategies
- Classical batch sampling not easy to perform as partitions are not easily merged...

Outline

Generative Modeling

- **Training data** : $\mathcal{D} = \{(\underline{X}_1, \underline{Y}_1), \ldots, (\underline{X}_n, \underline{Y}_n)\} \in (\mathcal{X} \times \mathcal{Y})^n \quad (\text{i.i.d.} \sim \mathbb{P})$
- Same kind of data than for supervised learning if $\mathcal{Y} \neq \emptyset$.

Generative Modeling

- Construct a map G from the product of $\mathcal{Y}$ and a randomness source Ω to $\mathcal{X}$

$$G : \mathcal{Y} \times \Omega \to \mathcal{X}$$
$$(Y, \omega) \mapsto X$$

- Unconditional model if $\mathcal{Y} = \emptyset \ldots$

Motivation

- Generate plausible novel conditional samples based on a given dataset.

Sample Quality

- Related to the proximity between the law of $G(Y, \omega)$ and the law of $X|Y$.
- Most classical choice is the Kullback-Leibler divergence.

Generative Modeling

Ingredients

- Generator $F_\theta(Y, \omega)$ and cond. density prob. $p_\theta(X|Y)$ (Explicit vs implicit link)
- Simple / Complex / Approximate estimation...

Some Possible Choices

	Probabilistic model	Generator	Estimation
Base	Simple (parametric)	Explicit	Simple (ML)
Flow	Image of simple model	Explicit	Simple (ML)
Factorization	Factorization of simple model	Explicit	Simple (ML)
VAE	Simple model with latent var.	Explicit	Approximate (ML)
EBM	Arbitrary	Implicit (MCMC)	Complex (ML/score/discrim.)
Diffusion	Continuous noise	Implicit (MCMC)	Complex (score)
	Discrete Noise with latent var.	Explicit	Approximate (ML)
GAN	Implicit	Explicit	Complex (Discrimination)

- SOTA: Diffusion based approach!

ML: Maximum Likelihood/VAE: Variational AutoEncoder/EBM: Energy Based Model/MCMC: Monte Carlo Markov Chain/GAN: Generative Aversarial Network

Outline

Density Estimation

Flow

Factorization

Outline

Latent

Outline

EBM

Outline

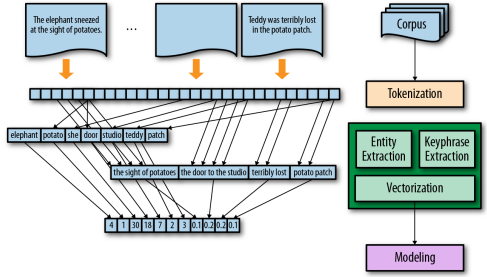
Diffusion

Outline

GAN

Outline

Text and Representation



Text and Representation

- Need to transform a text into a numerical vector to reuse the previous algorithms!
- Art still in progress.

- Important steps:
  - Token extraction
  - Token vectorization
  - Learning algorithm

Stemming

adjustable → adjust
formality → formaliti
formaliti → formal
airliner → airlin ⚠

Lemmatization

was → (to) be
better → good
meeting → meeting

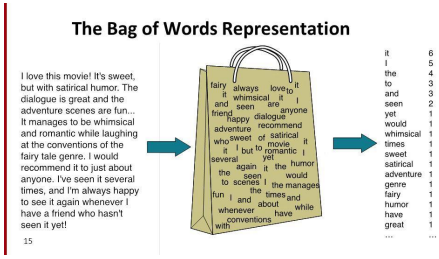
Token Extraction

- From a text to a sequence of *tokens* (words, characters, subwords...).

- Need of cleaning or pre-processing: spelling checker, stemming, lemmatization...
- Often with a further reduction of the number of possible tokens.
- Beware to not oversimplify!

Source: Quora

Bag of Words



Bag of Words

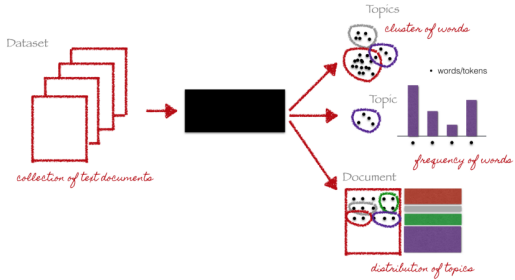
- Most simple approach to transform a text into a vector.
- Simple count of the words belonging to a predefined vocabulary.

- Counts preferably replaced by frequences (or tf-idf. . .)
- Often combined with dimension reduction:
  - restriction to an interesting vocabulary
  - use of principal component analysis (latent semantic analysis)

td-idf: text frequency - inverse document frequency

Source: Programmer Sought

Article Clustering

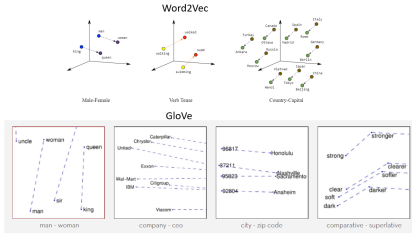


Article Clustering

- Clustering algorithms directly on the bag-of-words representation
- Most used algorithm is a variation around the k-means algorithm.

Source: C. Doig

Word Representation

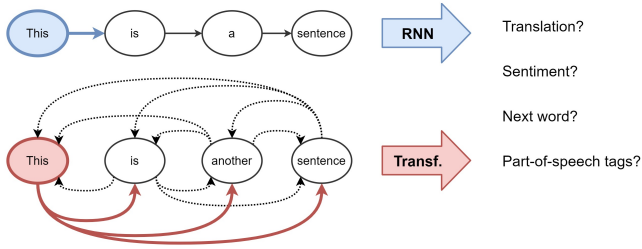


Word Representation

- More accuracy by working at the token (word) scale.
- Two approaches:
  - Associate to a word the frequency of the other words in its neighborhood and performing dimension reduction on this first representation.
  - Learn for each word a vector allowing to predict by a simple formula (scalar product) whether one word appears in the neighborhood of the other one.
- Similar results but the second approach is more flexible.

Source: R. Khandelwal

Deep Learning



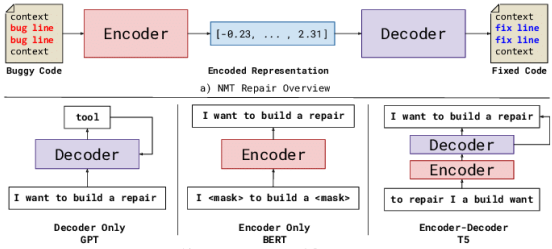
Deep Learning

- Propose a formula allowing to do computations on the word starting by associating vectors to each word.
- Learning the best possible vectors for a given task: auto-prediction (self-supervised) or prediction (supervised).

- Tremendous progress in the last years thanks to deep neural net architectures (RNN, Transformer...).

Source: Chaitanya Joshi

Large Language Models



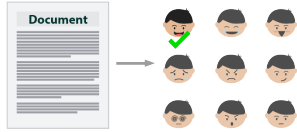
Large Language Models

- Huge neural networks relying on transformers and (pre)trained on huge corpus with self-supervised tasks.
- Three architectures:
  - Decoder: prediction of next word (online).
  - Encoder: prediction of inner word(s) (offline).
  - Encoder/Decoder: prediction of a sentence from anotherv(offline).

- Can be used as a basis for further specialized training or directly.

Source: Xia et al.

Sentiment Analysis



How to associate a sentiment to a text?

- Four possible approaches:
  - Simple approach (without learning) that averages the *sentiments* of the words used in a text using a fixed table.
  - Simple approach (supervised and linear) where this table is learned from examples.
  - Direct approach (supervised) where one predicts directly the sentiment from examples.
  - Zero-Shot approach (without learning?) where one uses directely a Large Language Model trained on a huge corpus (unrelated to the application).
- Direct approach more efficient provided one has sufficient data and one starts from a pretrained model.

Source: pngwing.com

Outline

References

T. Hastie, R. Tibshirani, and J. Friedman.
The Elements of Statistical Learning.
Springer Series in Statistics, 2009

M. Mohri, A. Rostamizadeh, and A. Talwalkar.
Foundations of Machine Learning (2nd ed.)
MIT Press, 2018

A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd ed.)
O'Reilly, 2022

Ch. Giraud.
Introduction to High-Dimensional Statistics.
CRC Press, 2014

S. Bubeck.
Convex Optimization: Algorithms and Complexity.
Now Publisher, 2015

References

F. Husson, S. Le, and J. Pagès.
Exploratory Multivariate Analysis by Example Using R (2nd ed.)
Chapman and Hall/CRC, 2017

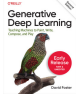
B. Ghojogh, M. Crowley, F. Karray, and A. Ghodsi.
Elements of Dimensionality Reduction and Manifold Learning.
Springer, 2023

Ch. Aggarwal and Ch. Reddy.
Data Clustering: Algorithms and Applications.
Chapman and Hall/CRC, 2013

Ch. Hennig, M. Meila, F. Murtagh, and R. Rocci.
Handbook of Cluster Analysis.
Chapman and Hall/CRC, 2015

Ch. Bouveyron, G. Celeux, B. Murphy, and A. Raftery.
Model-Based Clustering and Classification for Data Science.
Cambridge University Press, 2019

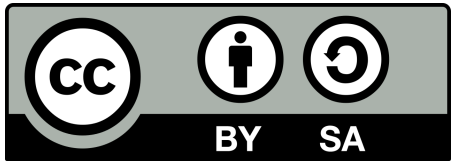
J. Tomczak.
Deep Generative Modeling.
Springer, 2021



D. Foster.
Generative Deep Learning (2nd ed.)
O'Reilly, 2023

A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd ed.)
O'Reilly, 2022

Licence and Contributors



Contributors

- Main contributor: E. Le Pennec
- Contributors: S. Boucheron, A. Dieuleveut, A.K. Fermin, S. Gadat, S. Gaiffas, A. Guilloux, Ch. Keribin, E. Matzner, M. Sangnier, E. Scornet.

Outline

Outline

Supervised Learning

Supervised Learning Framework

- Input measurement $\underline{X} \in \mathcal{X}$
- Output measurement $Y \in \mathcal{Y}$.
- $(\underline{X}, Y) \sim \mathbb{P}$ with $\mathbb{P}$ unknown.
- **Training data** : $\mathcal{D}_n = \{(\underline{X}_1, Y_1), \ldots, (\underline{X}_n, Y_n)\}$ (i.i.d. $\sim \mathbb{P}$)

- Often
  - $\underline{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (classification)
  - or $\underline{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A **predictor** is a function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y} \text{ meas.}\}$

Goal

- Construct a **good** predictor $\widehat{f}$ from the training data.

- Need to specify the meaning of good.
- Classification and regression are almost the **same** problem!

Loss and Probabilistic Framework

Loss function for a generic predictor

- **Loss function**: $\ell(Y, f(\underline{X}))$ measures the goodness of the prediction of Y by $f(\underline{X})$
- Examples:
  - 0/1 loss: $\ell(Y, f(\underline{X})) = \mathbf{1}_{Y \neq f(\underline{X})}$
  - Quadratic loss: $\ell(Y, f(\underline{X})) = |Y - f(\underline{X})|^2$

Risk function

- Risk measured as the average loss for a new couple:

$$\mathcal{R}(f) = \mathbb{E}_{(X,Y)\sim\mathbb{P}}[\ell(Y, f(\underline{X}))]$$

- Examples:
  - 0/1 loss: $\mathbb{E}[\ell(Y, f(\underline{X}))] = \mathbb{P}(Y \neq f(\underline{X}))$
  - Quadratic loss: $\mathbb{E}[\ell(Y, f(\underline{X}))] = \mathbb{E}\left[|Y - f(\underline{X})|^2\right]$

- **Beware:** As $\widehat{f}$ depends on $\mathcal{D}_n$, $\mathcal{R}(\widehat{f})$ is a random variable!

Best Solution

- The best solution $f^\star$ (which is independent of $\mathcal{D}_n$) is

$$f^\star = \arg\min_{f\in\mathcal{F}} \mathcal{R}(f) = \arg\min_{f\in\mathcal{F}} \mathbb{E}[\ell(Y, f(\underline{X}))] = \arg\min_{f\in\mathcal{F}} \mathbb{E}_{\underline{X}}\Big[\mathbb{E}_{Y|\underline{X}}[\ell(Y, f(\underline{X}))]\Big]$$

Bayes Predictor (explicit solution)

- In binary classification with $0-1$ loss:

$$f^\star(\underline{X}) = \begin{cases} +1 & \text{if } \mathbb{P}(Y=+1|\underline{X}) \geq \mathbb{P}(Y=-1|\underline{X}) \\ & \Leftrightarrow \mathbb{P}(Y=+1|\underline{X}) \geq 1/2 \\ -1 & \text{otherwise} \end{cases}$$

- In regression with the quadratic loss

$$f^\star(\underline{X}) = \mathbb{E}[Y|\underline{X}]$$

Issue: Solution requires to **know** $\mathbb{E}[Y|\underline{X}]$ for all values of $\underline{X}$!

Goal

Machine Learning

- Learn a rule to construct a **predictor** $\widehat{f} \in \mathcal{F}$ from the training data $\mathcal{D}_n$ s.t. **the risk** $\mathcal{R}(\widehat{f})$ is **small on average** or with high probability with respect to $\mathcal{D}_n$.

- In practice, the rule should be an algorithm!

Canonical example: Empirical Risk Minimizer

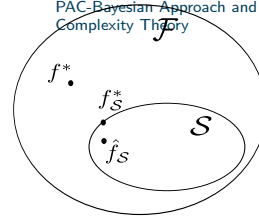
- One restricts f to a subset of functions $\mathcal{S} = \{f_\theta, \theta \in \Theta\}$
- One replaces the minimization of the average loss by the minimization of the empirical loss

$$\widehat{f} = f_{\widehat{\theta}} = \underset{f_\theta, \theta \in \Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f_\theta(\underline{X}_i))$$

- Examples:
  - Linear regression
  - Linear classification with

$$\mathcal{S} = \{\underline{x} \mapsto \operatorname{sign}\{\underline{x}^\top \beta + \beta^{(0)}\} \, / \beta \in \mathbb{R}^d, \beta^{(0)} \in \mathbb{R}\}$$

Bias-Variance Dilemma



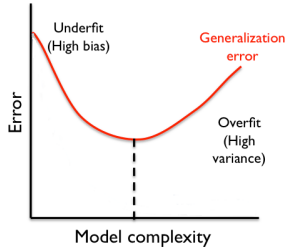
- General setting:
  - $\mathcal{F} = \{\text{measurable functions } \mathcal{X} \to \mathcal{Y}\}$
  - Best solution: $f^\star = \text{argmin}_{f \in \mathcal{F}} \mathcal{R}(f)$
  - Class $\mathcal{S} \subset \mathcal{F}$ of functions
  - Ideal target in $\mathcal{S}$: $f^\star_{\mathcal{S}} = \text{argmin}_{f \in \mathcal{S}} \mathcal{R}(f)$
  - Estimate in $\mathcal{S}$: $\widehat{f}_{\mathcal{S}}$ obtained with some procedure

Approximation error and estimation error (Bias/Variance)

$$\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star) = \underbrace{\mathcal{R}(f^\star_{\mathcal{S}}) - \mathcal{R}(f^\star)}_{\text{Approximation error}} + \underbrace{\mathcal{R}(\widehat{f}_{\mathcal{S}}) - \mathcal{R}(f^\star_{\mathcal{S}})}_{\text{Estimation error}}$$

- Approx. error can be large if the model $\mathcal{S}$ is not suitable.
- Estimation error can be large if the model is complex.

Under-fitting / Over-fitting Issue



- Different behavior for different model complexity
- **Low complexity model** are easily learned but the approximation error (**bias**) may be large (**Under-fit**).
- **High complexity model** may contain a good ideal target but the estimation error (**variance**) can be large (**Over-fit**)

Bias-variance trade-off $\iff$ avoid **overfitting** and **underfitting**

- **Rk:** Better to think in term of method (including feature engineering and specific algorithm) rather than only of model.

Source: Unknown

Outline

Outline

Empirical Risk Minimization

Empirical Risk Minimizer (ERM)

- For any loss ℓ and function class $\mathcal{S}$,

$$\widehat{f} = \operatorname*{argmin}_{f \in \mathcal{S}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f(\underline{X}_i)) = \operatorname*{argmin}_{f \in \mathcal{S}} \mathcal{R}_n(f)$$

- Key property:

$$\mathcal{R}_n(\widehat{f}) \leq \mathcal{R}_n(f), \forall f \in \mathcal{S}$$

- **Minimization not always tractable in practice!**
- Focus on the $\ell^{0/1}$ case:
  - only algorithm is to try all the functions,
  - not feasible is there are many functions
  - but interesting hindsight!

Outline

ERM and PAC Analysis

- Theoretical control of the random (error estimation) term:
$$\mathcal{R}(\widehat{f}) - \mathcal{R}(f^\star_{\mathcal{S}})$$

Probably Almost Correct Analysis

- **Theoretical guarantee** that
$$\mathbb{P}\Big(\mathcal{R}(\widehat{f}) - \mathcal{R}(f^\star_{\mathcal{S}}) \leq \epsilon_{\mathcal{S}}(\delta)\Big) \geq 1 - \delta$$
for a suitable $\epsilon_{\mathcal{S}}(\delta) \geq 0$.
- Implies:
  - $\mathbb{P}\Big(\mathcal{R}(\widehat{f}) - \mathcal{R}(f^\star) \leq \mathcal{R}(f^\star_{\mathcal{S}}) - \mathcal{R}(f^\star) + \epsilon_{\mathcal{S}}(\delta)\Big) \geq 1 - \delta$
  - $\mathbb{E}\Big[\mathcal{R}(\widehat{f}) - \mathcal{R}(f^\star_{\mathcal{S}})\Big] \leq \displaystyle\int_0^{+\infty} \delta_{\mathcal{S}}(\epsilon) d\epsilon$

- The result should hold without any assumption on the law $\boldsymbol{P}$!

A General Decomposition

- By construction:

$$\begin{aligned}\mathcal{R}(\widehat{f}) - \mathcal{R}(f_{\mathcal{S}}^{\star}) &= \mathcal{R}(\widehat{f}) - \mathcal{R}_n(\widehat{f}) + \mathcal{R}_n(\widehat{f}) - \mathcal{R}_n(f_{\mathcal{S}}^{\star}) + \mathcal{R}_n(f_{\mathcal{S}}^{\star}) - \mathcal{R}(f_{\mathcal{S}}^{\star}) \\ &\leq \mathcal{R}(\widehat{f}) - \mathcal{R}_n(\widehat{f}) + \mathcal{R}_n(f_{\mathcal{S}}^{\star}) - \mathcal{R}(f_{\mathcal{S}}^{\star}) \\ &\leq \Big(\mathcal{R}(\widehat{f}) - \mathcal{R}(f_{\mathcal{S}}^{\star})\Big) - \Big(\mathcal{R}_n(\widehat{f}) - \mathcal{R}_n(f_{\mathcal{S}}^{\star})\Big)\end{aligned}$$

Four possible upperbounds

- $\mathcal{R}(\widehat{f}) - \mathcal{R}(f_{\mathcal{S}}^{\star}) \leq \sup_{f \in \mathcal{S}} \left((\mathcal{R}(f) - \mathcal{R}(f_{\mathcal{S}}^{\star})) - (\mathcal{R}_n(f) - \mathcal{R}_n(f_{\mathcal{S}}^{\star})) \right)$
- $\mathcal{R}(\widehat{f}) - \mathcal{R}(f_{\mathcal{S}}^{\star}) \leq \sup_{f \in \mathcal{S}} (\mathcal{R}(f) - \mathcal{R}_n(f)) + (\mathcal{R}_n(f_{\mathcal{S}}^{\star}) - \mathcal{R}(f_{\mathcal{S}}^{\star}))$
- $\mathcal{R}(\widehat{f}) - \mathcal{R}(f_{\mathcal{S}}^{\star}) \leq \sup_{f \in \mathcal{S}} (\mathcal{R}(f) - \mathcal{R}_n(f)) + \sup_{f \in \mathcal{S}} (\mathcal{R}_n(f) - \mathcal{R}(f))$
- $\mathcal{R}(\widehat{f}) - \mathcal{R}(f_{\mathcal{S}}^{\star}) \leq 2 \sup_{f \in \mathcal{S}} |\mathcal{R}(f) - \mathcal{R}_n(f)|$

- Supremum of centered random variables!
- **Key:** Concentration of each variable...

Risk Bounds

- By construction, for any $f' \in \mathcal{S}$,
$$\mathcal{R}(f') = \mathcal{R}_n(f') + (\mathcal{R}(f') - \mathcal{R}_n(f'))$$

A uniform upper bound for the risk

- Simultaneously $\forall f' \in \mathcal{S}$,
$$\mathcal{R}(f') \leq \mathcal{R}_n(f') + \sup_{f \in \mathcal{S}} (\mathcal{R}(f) - \mathcal{R}_n(f))$$

- Supremum of centered random variables!
- **Key:** Concentration of each variable...
- Can be interpreted as a justification of the ERM!

Outline

Concentration of the Empirical Loss

- Empirical loss:

$$\mathcal{R}_n(f) = \frac{1}{n}\sum_{i=1}^{n} \ell^{0/1}(Y_i, f(\underline{X}_i))$$

Properties

- $\ell^{0/1}(Y_i, f(\underline{X}_i))$ are i.i.d. random variables in $[0,1]$.

Concentration

$$\mathbb{P}(\mathcal{R}(f) - \mathcal{R}_n(f) \leq \epsilon) \geq 1 - e^{-2n\epsilon^2}$$

$$\mathbb{P}(\mathcal{R}_n(f) - \mathcal{R}(f) \leq \epsilon) \geq 1 - e^{-2n\epsilon^2}$$

$$\mathbb{P}(|\mathcal{R}_n(f) - \mathcal{R}(f)| \leq \epsilon) \geq 1 - 2e^{-2n\epsilon^2}$$

- Concentration of sum of bounded independent variables!
- Hoeffding theorem.
- Equiv. to $\mathbb{P}\left(\mathcal{R}(f) - \mathcal{R}_n(f) \leq \sqrt{\log(1/\delta)/(2n)}\right) \geq 1 - \delta$

Hoeffding

Theorem

- Let Z_i be a sequence of ind. centered r.v. supported in $[a_i, b_i]$ then

$$\mathbb{P}\left(\sum_{i=1}^{n} Z_i \geq \epsilon\right) \leq e^{-\frac{2\epsilon^2}{\sum_{i=1}^{n}(b_i-a_i)^2}}$$

- Proof ingredients:
  - Chernov bounds:

$$\mathbb{P}\left(\sum_{i=1}^{n} Z_i \geq \epsilon\right) \leq \frac{\mathbb{E}\left[e^{\lambda \sum_{i=1}^{n} Z_i}\right]}{e^{\lambda\epsilon}} \leq \frac{\prod_{i=1}^{n}\mathbb{E}\left[e^{\lambda Z_i}\right]}{e^{\lambda\epsilon}}$$

  - Exponential moment bounds: $\mathbb{E}\left[e^{\lambda Z_i}\right] \leq e^{\frac{\lambda^2(b_i-a_i)^2}{8}}$
  - Optimization in λ
- **Prop:**

$$\mathbb{E}\left[e^{\lambda \sum_{i=1}^{n} Z_i}\right] \leq e^{\frac{\lambda^2 \sum_{i=1}^{n}(b_i-a_i)^2}{8}}.$$

Hoeffding Inequality

Theorem

- Let Z_i be a sequence of independent centered random variables supported in $[a_i, b_i]$ then

$$\mathbb{P}\left(\sum_{i=1}^{n} Z_i \geq \epsilon\right) \leq e^{-\frac{2\epsilon^2}{\sum_{i=1}^{n}(b_i-a_i)^2}}$$

- $Z_i = \frac{1}{n}\left(\mathbb{E}\left[\ell^{0/1}(Y, f(\underline{X}))\right] - \ell^{0/1}(Y_i, f(\underline{X}_i))\right)$
- $\mathbb{E}[Z_i] = 0$ and $Z_i \in [\frac{1}{n}\left(\mathbb{E}\left[\ell^{0/1}(Y, f(\underline{X}))\right] - 1\right), \frac{1}{n}\mathbb{E}\left[\ell^{0/1}(Y, f(\underline{X}))\right]]$
- Concentration:

$$\mathbb{P}(\mathcal{R}(f) - \mathcal{R}_n(f) \geq \epsilon) \leq e^{-2n\epsilon^2}$$

- By symmetry,

$$\mathbb{P}(\mathcal{R}_n(f) - \mathcal{R}(f) \geq \epsilon) \leq e^{-2n\epsilon^2}$$

- Combining the two yields

$$\mathbb{P}(|\mathcal{R}_n(f) - \mathcal{R}(f)| \geq \epsilon) \leq 2e^{-2n\epsilon^2}$$

Finite Class Case

Concentration

- If $\mathcal{S}$ is finite of cardinality $|\mathcal{S}|$,

$$\mathbb{P}\left(\sup_f \left(\mathcal{R}(f) - \mathcal{R}_n(f)\right) \leq \sqrt{\frac{\log|\mathcal{S}| + \log(1/\delta)}{2n}}\right) \geq 1 - \delta$$

$$\mathbb{P}\left(\sup_f \left|\mathcal{R}_n(f) - \mathcal{R}(f)\right| \leq \sqrt{\frac{\log|\mathcal{S}| + \log(1/\delta)}{2n}}\right) \geq 1 - 2\delta$$

- Control of the supremum by a quantity depending on the cardinality and the probability parameter δ.
- Simple combination of Hoeffding and a union bound.

Finite Class Case

PAC Bounds

- If $\mathcal{S}$ is finite of cardinality $|\mathcal{S}|$, with proba greater than $1-2\delta$

$$\mathcal{R}(\widehat{f}) - \mathcal{R}(f^\star_{\mathcal{S}}) \leq \sqrt{\frac{\log|\mathcal{S}| + \log(1/\delta)}{2n}} + \sqrt{\frac{\log(1/\delta)}{2n}}$$
$$\leq 2\sqrt{\frac{\log|\mathcal{S}| + \log(1/\delta)}{2n}}$$

- If $\mathcal{S}$ is finite of cardinality $|\mathcal{S}|$, with proba greater than $1-\delta$, simultaneously $\forall f' \in \mathcal{S}$,

$$\mathcal{R}(f') \leq \mathcal{R}_n(f') + \sqrt{\frac{\log|\mathcal{S}| + \log(1/\delta)}{2n}}$$
$$\leq \mathcal{R}_n(f') + \sqrt{\frac{\log|\mathcal{S}|}{2n}} + \sqrt{\frac{\log(1/\delta)}{2n}}$$

PAC Bounds

- If $\mathcal{S}$ is finite of cardinality $|\mathcal{S}|$, with proba greater than $1-2\delta$

$$\mathcal{R}(\widehat{f}) - \mathcal{R}(f^\star_{\mathcal{S}}) \leq \sqrt{\frac{\log|\mathcal{S}|}{2n}} + \sqrt{\frac{2\log(1/\delta)}{n}}$$

- If $\mathcal{S}$ is finite of cardinality $|\mathcal{S}|$, with proba greater than $1-\delta$, simultaneously $\forall f' \in \mathcal{S}$,

$$\mathcal{R}(f') \leq \mathcal{R}_n(f') + \sqrt{\frac{\log|\mathcal{S}|}{2n}} + \sqrt{\frac{\log(1/\delta)}{2n}}$$

- Risk increases with the cardinality of $\mathcal{S}$.
- Similar issue in cross-validation!
- No direct extension for an infinite $\mathcal{S}\ldots$

Outline

Concentration of the Supremum of Empirical Losses

- Supremum of Empirical losses:

$$\begin{aligned}\Delta_n(\mathcal{S})(\underline{X}_1, \ldots, \underline{X}_n) &= \sup_{f \in \mathcal{S}} \mathcal{R}(f) - \mathcal{R}_n(f) \\ &= \sup_{f \in \mathcal{S}} \left(\mathbb{E}\left[\ell^{0/1}(Y, f(\underline{X}))\right] - \frac{1}{n}\sum_{i=1}^{n} \ell^{0/1}(Y_i, f(\underline{X}_i)) \right)\end{aligned}$$

Properties

- Bounded difference:

$$|\Delta_n(\mathcal{S})(\underline{X}_1, \ldots, \underline{X}_i, \ldots \underline{X}_n) - \Delta_n(\mathcal{S})(\underline{X}_1, \ldots \underline{X}'_i, \ldots, \underline{X}_n)| \leq 1/n$$

Concentration

$$\mathbb{P}(\Delta_n(\mathcal{S}) - \mathbb{E}[\Delta_n(\mathcal{S})] \leq \epsilon) \geq 1 - e^{-2n\epsilon^2}$$

- Concentration of bounded difference function.
- Generalization of Hoeffding theorem: McDiarmid Theorem.

McDiarmid Inequality

Bounded difference function

- $g : \mathcal{X}^n \to \mathbb{R}$ is a bounded difference function if it exist c_i such that

$$\forall (\underline{X}_i)_{i=1}^n, (\underline{X}'_i)_{i=1}^n \in \mathbb{R},$$
$$\left| g(\underline{X}_1, \ldots, \underline{X}_i, \ldots, \underline{X}_n) - g(\underline{X}_1, \ldots, \underline{X}'_i, \ldots, \underline{X}_n) \right| \leq c_i$$

Theorem

- If g is a bounded difference function and $\underline{X}_i$ are independent random variables then

$$\mathbb{P}(g(\underline{X}_1, \ldots, \underline{X}_n) - \mathbb{E}[g(\underline{X}_1, \ldots, \underline{X}_n)] \geq \epsilon) \leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2}}$$

$$\mathbb{P}(\mathbb{E}[g(\underline{X}_1, \ldots, \underline{X}_n)] - g(\underline{X}_1, \ldots, \underline{X}_n) \geq \epsilon) \leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2}}$$

- Proof ingredients:
  - Chernov bounds
  - Martingale decomposition...

McDiarmid Inequality

Theorem

- If g is a bounded difference function and $\underline{X}_i$ are independent random variables then

$$\mathbb{P}(g(\underline{X}_1, \ldots, \underline{X}_n) - \mathbb{E}[g(\underline{X}_1, \ldots, \underline{X}_n)] \geq \epsilon) \leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2}}$$

- Using $g = \Delta_n(\mathcal{S})$ for which $c_i = 1/n$ yields immediately

$$\mathbb{P}(\Delta_n(\mathcal{S}) - \mathbb{E}[\Delta_n(\mathcal{S})] \geq \epsilon) \leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2}} = e^{-2n\epsilon^2}$$

- We derive then

$$\mathbb{P}(\Delta_n(\mathcal{S}) \geq \mathbb{E}[\Delta_n(\mathcal{S})] + \epsilon) \leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2}} = e^{-2n\epsilon^2}$$

- It remains to upperbound

$$\mathbb{E}[\Delta_n] = \mathbb{E}\left[\sup_{f \in \mathcal{S}} \mathcal{R}(f) - \mathcal{R}_n(f)\right]$$

Rademacher Complexity

Theorem

- Let σ_i be a sequence of i.i.d. random symmetric Bernoulli variables (Rademacher variables):

$$\mathbb{E}\left[\sup_{f\in\mathcal{S}}\left(\mathcal{R}(f)-\mathcal{R}_n(f)\right)\right]\leq 2\mathbb{E}\left[\sup_{f\in\mathcal{S}}\frac{1}{n}\sum_{i=1}^{n}\sigma_i\ell^{0/1}(Y_i,f(\underline{X}_i))\right]$$

Rademacher complexity

- Let $B\subset\mathbf{R}^n$, the Rademacher complexity of B is defined as

$$R_n(B)=\mathbb{E}\left[\sup_{b\in B}\frac{1}{n}\sum_{i=1}^{n}\sigma_i b_i\right]$$

- Theorem gives an upper bound of the expectation in terms of the **average Rademacher complexity of the random set** $B_n(\mathcal{S})=\{(\ell^{0/1}(Y_i,f(\underline{X}_i)))_{i=1}^n, f\in\mathcal{S}\}$.
- **Back to finite setting:** This set is at most of cardinality 2^n.

Finite Set Rademacher Complexity Bound

Theorem

- If B is finite and such that $\forall b \in B, \frac{1}{n}\|b\|_2^2 \leq M^2$, then

$$R_n(B) = \mathbb{E}\left[\sup_{b \in B} \frac{1}{n} \sum_{i=1}^{n} \sigma_i b_i\right] \leq \sqrt{\frac{2M^2 \log |B|}{n}}$$

- If $B = B_n(\mathcal{S}) = \{(\ell^{0/1}(Y_i, f(\underline{X}_i)))_{i=1}^n, f \in \mathcal{S}\}$, we have $M = 1$ and thus

$$R_n(B) \leq \sqrt{\frac{2 \log |B_n(\mathcal{S})|}{n}}$$

- We obtain immediately

$$\mathbb{E}\left[\sup_{f \in \mathcal{S}} (\mathcal{R}(f) - \mathcal{R}_n(f))\right] \leq \mathbb{E}\left[\sqrt{\frac{8 \log |B_n(\mathcal{S})|}{n}}\right].$$

Finite Set Rademacher Complexity Bound

Theorem

- With probability greater than $1 - 2\delta$,

$$\mathcal{R}(\widehat{f}) - \mathcal{R}(f_{\mathcal{S}}^{\star}) \leq \mathbb{E}\left[\sqrt{\frac{8 \log |B_n(\mathcal{S})|}{n}}\right] + \sqrt{\frac{2 \log(1/\delta)}{n}}$$

- With probability greater than $1 - \delta$, simultaneously $\forall f' \in \mathcal{S}$

$$\mathcal{R}(f') \leq \mathcal{R}_n(f') + \mathbb{E}\left[\sqrt{\frac{8 \log |B_n(\mathcal{S})|}{n}}\right] + \sqrt{\frac{\log(1/\delta)}{2n}}$$

- This is a direct consequence of the previous bound.

Finite Set Rademacher Complexity Bound

Corollary

- If $\mathcal{S}$ is finite then with probability greater than $1-2\delta$

$$\mathcal{R}(\widehat{f}) - \mathcal{R}(f^\star_{\mathcal{S}}) \leq \sqrt{\frac{8\log|\mathcal{S}|}{n}} + \sqrt{\frac{2\log(1/\delta)}{n}}$$

- If $\mathcal{S}$ is finite then with probability greater than $1-\delta$, simultaneously $\forall f' \in \mathcal{S}$

$$\mathcal{R}(f') \leq \mathcal{R}_n(f') + \sqrt{\frac{8\log|\mathcal{S}|}{n}} + \sqrt{\frac{\log(1/\delta)}{2n}}$$

- It suffices to notice that

$$|B_n(\mathcal{S})| = |\{(\ell^{0/1}(Y_i, f(\underline{X}_i)))_{i=1}^n, f \in \mathcal{S}\}| \leq |\mathcal{S}|$$

Finite Set Rademacher Complexity Bound

- Same result with Hoeffding but with **better** constants!

$$\mathcal{R}(\widehat{f}) - \mathcal{R}(f_{\mathcal{S}}^{\star}) \leq \sqrt{\frac{\log|\mathcal{S}|}{2n}} + \sqrt{\frac{2\log(1/\delta)}{n}}$$

$$\mathcal{R}(f') \leq \mathcal{R}_n(f') + \sqrt{\frac{\log|\mathcal{S}|}{2n}} + \sqrt{\frac{\log(1/\delta)}{2n}}$$

- Difference due to the *crude* upperbound of

$$\mathbb{E}\left[\sup_{f\in\mathcal{S}}\left(\mathcal{R}(f) - \mathcal{R}_n(f)\right)\right]$$

- **Why bother?:** We do not have to assume that $\mathcal{S}$ is finite!

$$|B_n(\mathcal{S})| \leq 2^n$$

Outline

Back to the Bound

Theorem

$$\mathbb{E}\left[\sup_{f\in\mathcal{S}}\left(\mathcal{R}(f)-\mathcal{R}_n(f)\right)\right]\leq\mathbb{E}\left[\sqrt{\frac{8\log|B_n(\mathcal{S})|}{n}}\right]$$

- Key quantity: $\mathbb{E}\left[\sqrt{\frac{8\log|B_n(\mathcal{S})|}{n}}\right]$
- Hard to control due to its structure!

A first data dependent upperbound

$$\mathbb{E}\left[\sqrt{\frac{8\log|B_n(\mathcal{S})|}{n}}\right]\leq\sqrt{\frac{8\log\mathbb{E}[|B_n(\mathcal{S})|]}{n}}\quad\text{(Jensen)}$$

- Depends on the unknown $\boldsymbol{P}$!

Shattering Coefficient

Shattering Coefficient (or Growth Function)

- The shattering coefficient of the class $\mathcal{S}$, $s(\mathcal{S}, n)$, is defined as

$$s(\mathcal{S}, n) = \sup_{\left((\underline{X}_1, Y_1), \ldots, (\underline{X}_n, Y_n)\right) \in (\mathcal{X} \times \{-1, 1\})^n} |\{(\ell^{0/1}(Y_i, f(\underline{X}_i)))_{i=1}^n, f \in \mathcal{S}\}|$$

- By construction, $|B_n(\mathcal{S})| \leq s(\mathcal{S}, n) \leq \min(2^n, |\mathcal{S}|)$.

A data independent upperbound

$$\mathbb{E}\left[\sqrt{\frac{8 \log |B_n(\mathcal{S})|}{n}}\right] \leq \sqrt{\frac{8 \log s(\mathcal{S}, n)}{n}}$$

Shattering Coefficient

Theorem

- With probability greater than $1 - 2\delta$,
$$\mathcal{R}(\widehat{f}) - \mathcal{R}(f^\star_{\mathcal{S}}) \leq \sqrt{\frac{8 \log s(\mathcal{S}, n)}{n}} + \sqrt{\frac{2 \log(1/\delta)}{n}}$$
- With probability greater than $1 - \delta$, simultaneously $\forall f' \in \mathcal{S}$,
$$\mathcal{R}(f') \leq \mathcal{R}_n(f') + \sqrt{\frac{8 \log s(\mathcal{S}, n)}{n}} + \sqrt{\frac{\log(1/\delta)}{2n}}$$

- Depends only on the class $\mathcal{S}$!

Vapnik-Chervonenkis Dimension

VC Dimension

- The VC dimension d_{VC} of $\mathcal{S}$ is defined as the largest integer d such that
$$s(\mathcal{S}, d) = 2^d$$

- The VC dimension can be infinite!

VC Dimension and Dimension

- **Prop:** If span($\mathcal{S}$) corresponds to the sign of functions in a linear space of dimension d then $d_{VC} \leq d$.

- VC dimension similar to the usual dimension.

Sauer's Lemma

- If the VC dimension d_{VC} of $\mathcal{S}$ is finite

$$s(\mathcal{S}, n) \leq \begin{cases} 2^n & \text{if } n \leq d_{VC} \\ \left(\frac{en}{d_{VC}}\right)^{d_{VC}} & \text{if } n > d_{VC} \end{cases}$$

- **Cor.:** $\log s(\mathcal{S}, n) \leq d_{VC} \log\left(\frac{en}{d_{VC}}\right)$ if $n > d_{VC}$.

PAC Bounds

- If $\mathcal{S}$ is of VC dimension d_{VC} then if $n > d_{VC}$
- With probability greater than $1 - 2\delta$,

$$\mathcal{R}(\widehat{f}) - \mathcal{R}(f_{\mathcal{S}}^{\star}) \leq \sqrt{\frac{8 d_{VC} \log\left(\frac{en}{d_{VC}}\right)}{n}} + \sqrt{\frac{2\log(1/\delta)}{n}}$$

- With probability greater than $1 - \delta$, simultaneously $\forall f' \in \mathcal{S}$,

$$\mathcal{R}(f') \leq \mathcal{R}_n(f') + \sqrt{\frac{8 d_{VC} \log\left(\frac{en}{d_{VC}}\right)}{n}} + \sqrt{\frac{\log(1/\delta)}{2n}}$$

- **Rk:** If $d_{VC} = +\infty$ no uniform PAC bounds exists!

Outline

PAC Bounds

- Let $\pi_f > 0$ such that $\sum_{f \in \mathcal{S}} \pi_f = 1$
- With proba greater than $1 - 2\delta$,

$$\mathcal{R}(\widehat{f}) - \mathcal{R}(f^\star_{\mathcal{S}}) \leq \sqrt{\frac{\log(1/\pi_f)}{2n}} + \sqrt{\frac{2\log(1/\delta)}{n}}$$

- With proba greater than $1 - \delta$, simultaneously $\forall f' \in \mathcal{S}$,

$$\mathcal{R}(f') \leq \mathcal{R}_n(f') + \sqrt{\frac{\log(1/\pi_f)}{2n}} + \sqrt{\frac{\log(1/\delta)}{2n}}$$

- Very similar proof than the uniform one!
- Much more interesting idea when combined with several models...

Models, Non Uniform Risk Bounds and SRM

- Assume we have a countable collection of set $(\mathcal{S}_m)_{m\in\mathcal{M}}$ and let π_m be such that $\sum_{m\in\mathcal{M}} \pi_m = 1$.

Non Uniform Risk Bound

- With probability $1-\delta$, simultaneously for all $m \in \mathcal{M}$ and all $f \in \mathcal{S}_m$,

$$\mathcal{R}(f) \leq \mathcal{R}_n(f) + \mathbb{E}\left[\sqrt{\frac{8\log|B_n(\mathcal{S}_m)|}{n}}\right] + \sqrt{\frac{\log(1/\pi_m)}{2n}} + \sqrt{\frac{\log(1/\delta)}{2n}}$$

Structural Risk Minimization

- Choose $\hat{f}$ as the minimizer over $m \in \mathcal{M}$ and $f \in \mathcal{S}_m$ of

$$\mathcal{R}_n(f) + \mathbb{E}\left[\sqrt{\frac{8\log|B_n(\mathcal{S}_m)|}{n}}\right] + \sqrt{\frac{\log(1/\pi_m)}{2n}}$$

- Mimics the minimization of the integrated risk!

SRM and PAC Bound

PAC Bound

- If $\hat{f}$ is the SRM minimizer then with probability $1-2\delta$,

$$\mathcal{R}(\widehat{f}) \leq \inf_{m\in\mathcal{M}} \inf_{f\in\mathcal{S}_m} \left(\mathcal{R}(f) + \mathbb{E}\left[\sqrt{\frac{8\log|B_n(\mathcal{S}_m)|}{n}}\right] + \sqrt{\frac{\log(1/\pi_m)}{2n}} \right) + \sqrt{\frac{2\log(1/\delta)}{n}}$$

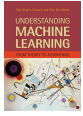
- The SRM minimizer balances the risk $\mathcal{R}(f)$ and the upper bound on the estimation error $\mathbb{E}\left[\sqrt{\frac{8\log|B_n(\mathcal{S}_m)|}{n}}\right] + \sqrt{\frac{\log(1/\pi_m)}{2n}}$.
- $\mathbb{E}\left[\sqrt{\frac{8\log|B_n(\mathcal{S}_m)|}{n}}\right]$ can be replaced by an upper bound (for instance a VC based one)...

Outline

References

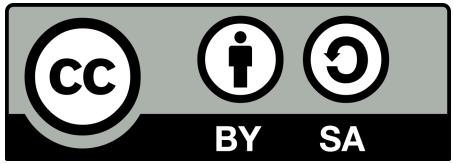
Ch. Giraud.
Introduction to High-Dimensional Statistics.
CRC Press, 2014

M. Mohri, A. Rostamizadeh, and A. Talwalkar.
Foundations of Machine Learning (2nd ed.)
MIT Press, 2018



S. Shalev-Shwartz and S. Ben-David.
Understanding Machine Learning.
Cambridge University Press, 2014

Licence and Contributors



Contributors

- Main contributor: E. Le Pennec
- Contributors: S. Boucheron, A. Dieuleveut, A.K. Fermin, S. Gadat, S. Gaiffas, A. Guilloux, Ch. Keribin, E. Matzner, M. Sangnier, E. Scornet.

Outline

References

T. Hastie, R. Tibshirani, and J. Friedman.
The Elements of Statistical Learning.
Springer Series in Statistics, 2009

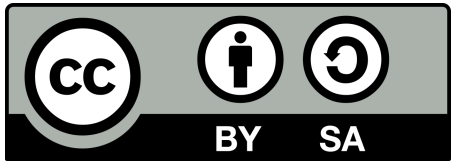
M. Mohri, A. Rostamizadeh, and A. Talwalkar.
Foundations of Machine Learning (2nd ed.)
MIT Press, 2018

A. Géron.
Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd ed.)
O'Reilly, 2022

Ch. Giraud.
Introduction to High-Dimensional Statistics.
CRC Press, 2014

S. Bubeck.
Convex Optimization: Algorithms and Complexity.
Now Publisher, 2015

Licence and Contributors



Contributors

- Main contributor: E. Le Pennec
- Contributors: S. Boucheron, A. Dieuleveut, A.K. Fermin, S. Gadat, S. Gaiffas, A. Guilloux, Ch. Keribin, E. Matzner, M. Sangnier, E. Scornet.