Large Scale Machine Learning

Introduction

Adeline Fermanian
adeline.fermanian@mines-paristech.fr
March 2023

Mines ParisTech - PSL

Acknowledgement

Slides inspired by

- Chloé-Agathe Azencott
- Jean-Philippe Vert
- Claire Boyer

Sommaire

TECH

2017 is the year of Machine Learning. Here's why

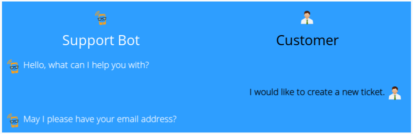
GAURAV SANGWANI | 0 | JAN 13, 2017, 12.51 PM

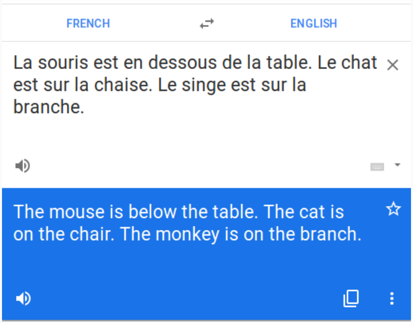
Facebook | Linkedin | Twitter | Google+ | Reddit

Machine learning is maybe the most sweltering thing in Silicon Valley at this moment. Particularly deep learning. The reason why it is so hot is on the grounds that it can assume control of numerous repetitive, thoughtless tasks. It'll improve doctors, and make lawyers better lawyers. What's more, it makes cars drive themselves.

Perception

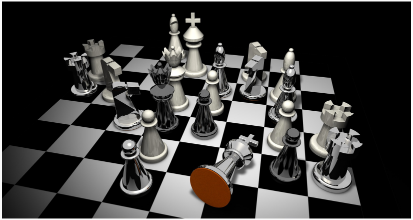
Communication

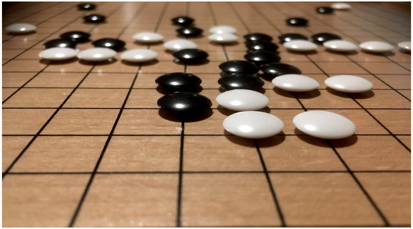




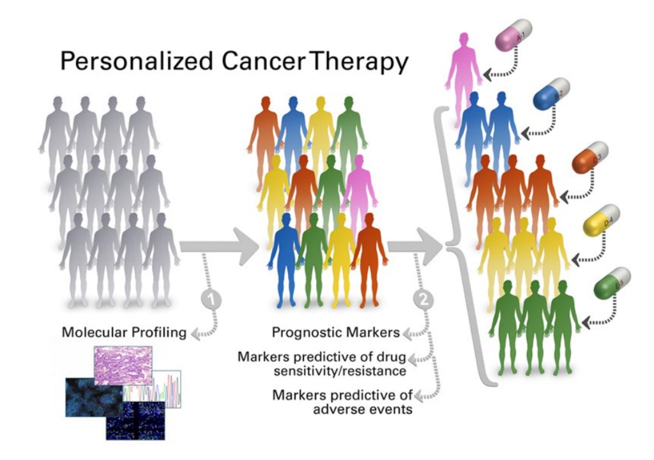
Mobility

Reasoning



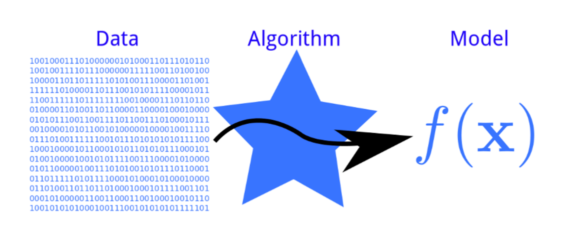


Health



https://pct.mdanderson.org

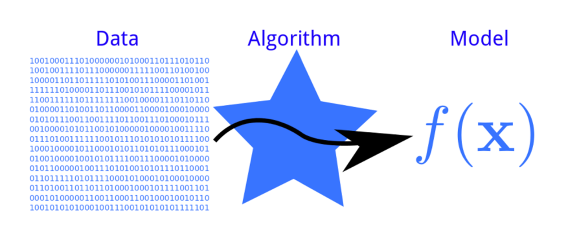
A common process: learning from data



https://www.linkedin.com/pulse/supervised-machine-learning-pega-decisioning-solution-nizam-muhammad

- Given examples (training data), make a machine learn how to predict on new samples, or discover patterns in data

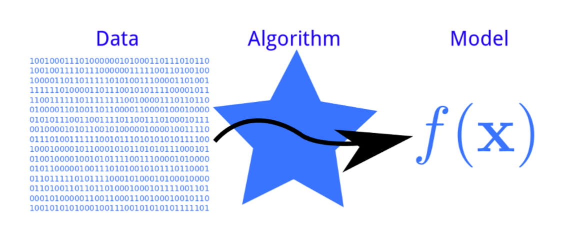
A common process: learning from data



https://www.linkedin.com/pulse/supervised-machine-learning-pega-decisioning-solution-nizam-muhammad

- Given examples (training data), make a machine learn how to predict on new samples, or discover patterns in data
- Statistics + optimization + computer science

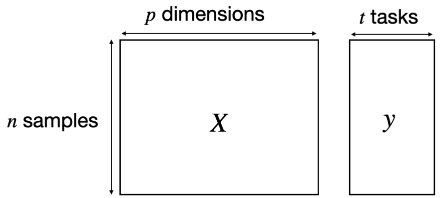
A common process: learning from data



https://www.linkedin.com/pulse/supervised-machine-learning-pega-decisioning-solution-nizam-muhammad

- Given examples (training data), make a machine learn how to predict on new samples, or discover patterns in data
- Statistics + optimization + computer science
- Gets better with more training examples and bigger computers

Large-scale ML?



- Iris dataset: $n = 150, p = 4, t = 1$
- Cancer drug sensitivity: $n = 10^3, p = 10^6, t = 100$
- Imagenet: $n = 14.10^6, p = 60.10^3, t = 22.10^3$
- Shopping, e-marketing $n = \mathcal{O}(10^6), p = \mathcal{O}(10^9), t = \mathcal{O}(10^8)$
- Astronomy, GAFAMs, web... $n = \mathcal{O}(10^9), p = \mathcal{O}(10^9), t = \mathcal{O}(10^9)$

Today's goals

1. Review a few standard ML techniques
2. Introduce a few ideas and techniques to scale them to modern, big datasets

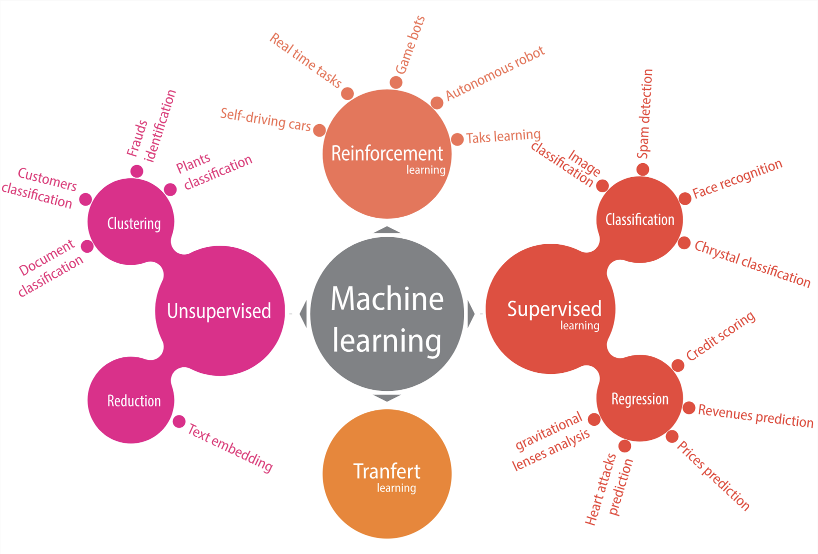
Sommaire

Learning scenarios

ML develops generic methods for solving different types of problems:

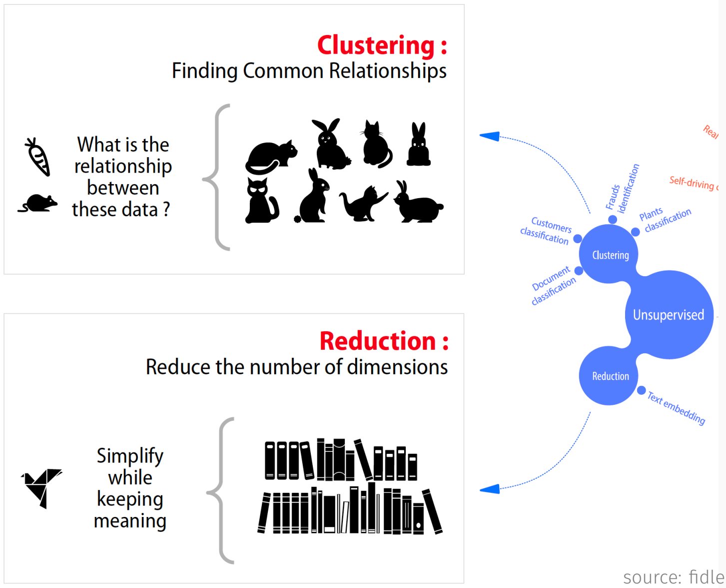
- Supervised learning
  Goal: learn from examples
- Unsupervised learning
  Goal: learn from data alone, extract structure in the data
- Reinforcement learning
  Goal: learn by exploring the environment (e.g. games or autonomous vehicle)

Learning scenarios



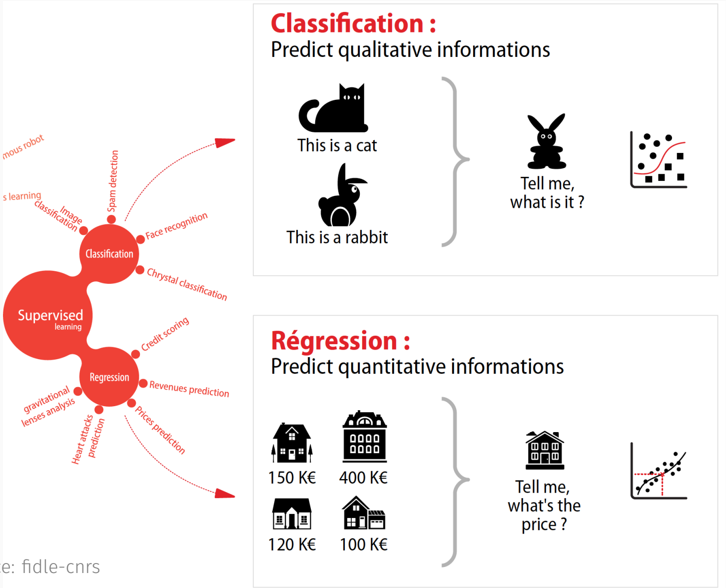
source: fidle-cnrs

Unsupervised learning



source: fidle-cnrs

Supervised learning



source: fidle-cnrs

Main ML paradigms

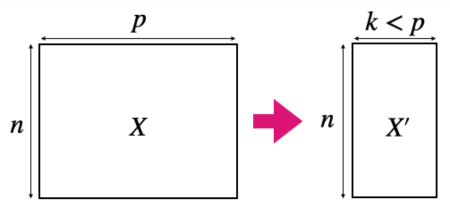
- Unsupervised learning
  - Dimension reduction
  - Clustering
  - Density estimation
  - Feature learning
- Supervised learning
  - Regression
  - Classification
  - Structured output classification
- Semi-supervised learning
- Reinforcement learning

Main ML paradigms

- Unsupervised learning
  - Dimension reduction: PCA
  - Clustering: k-means
  - Density estimation
  - Feature learning
- Supervised learning
  - Regression: OLS, ridge regression
  - Classification: logistic regression, SVM
  - Structured output classification
- Semi-supervised learning
- Reinforcement learning

Sommaire

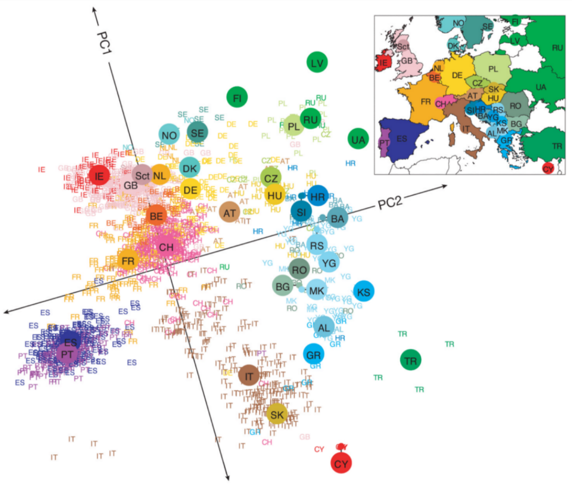
Motivation



- Reduce the dimension without losing the variability in the data;
- Visualization $(k = 2, 3)$
- Discover structure

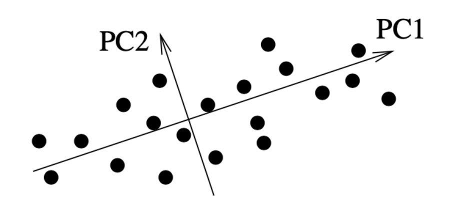
Motivation: Population genetics

- Genetic data of 1387 Europeans



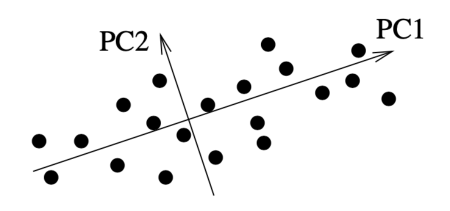
source: Novembre et al, 2008

PCA definition



- The kth principal component:

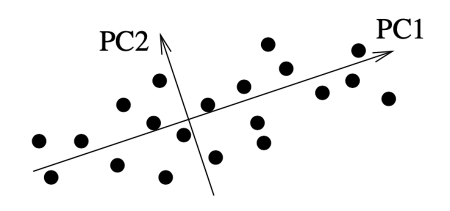
PCA definition



- The kth principal component:
  - Is orthogonal to all previous components:

$$\langle w_k, w_1 \rangle = \langle w_k, w_2 \rangle = \cdots = \langle w_k, w_{k-1} \rangle = 0$$

PCA definition



- The kth principal component:
  - Is orthogonal to all previous components:

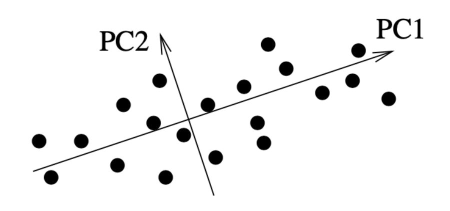
$$\langle w_k, w_1 \rangle = \langle w_k, w_2 \rangle = \cdots = \langle w_k, w_{k-1} \rangle = 0$$

  - Captures the largest amount of variance:

$$\max_{\|w\|=1} w^\top X^\top X w = \max_{\|w\|=1} \|Xw\|^2$$

($X^\top X$: empirical covariance of X (centered))

PCA definition



- The kth principal component:
  - Is orthogonal to all previous components:

$$\langle w_k, w_1 \rangle = \langle w_k, w_2 \rangle = \cdots = \langle w_k, w_{k-1} \rangle = 0$$

  - Captures the largest amount of variance:

$$\max_{\|w\|=1} w^\top X^\top X w = \max_{\|w\|=1} \|Xw\|^2$$

    ($X^\top X$: empirical covariance of X (centered))
  - Solution: w is the kth eigenvector of $X^\top X$.

PCA complexity

- Memory: store X and covariance matrix $X^\top X$:

PCA complexity

- Memory: store X and covariance matrix $X^\top X$: $\mathcal{O}(\max(np, p^2))$

PCA complexity

- Memory: store X and covariance matrix $X^\top X$: $\mathcal{O}(\max(np, p^2))$
- Runtime:
  - Compute $X^\top X$: $\mathcal{O}(np^2)$
  - Compute k eigenvectors of $X^\top X$ with power methods: $\mathcal{O}(kp^2)$

PCA complexity

- Memory: store X and covariance matrix $X^\top X$: $\mathcal{O}(\max(np, p^2))$
- Runtime:
  - Compute $X^\top X$: $\mathcal{O}(np^2)$
  - Compute k eigenvectors of $X^\top X$ with power methods: $\mathcal{O}(kp^2)$

  Computing the covariance matrix is more expensive than computing its eigenvectors ($n > k$)!

PCA complexity

- Memory: store X and covariance matrix $X^\top X$: $\mathcal{O}(\max(np, p^2))$
- Runtime:
  - Compute $X^\top X$: $\mathcal{O}(np^2)$
  - Compute k eigenvectors of $X^\top X$ with power methods: $\mathcal{O}(kp^2)$

  Computing the covariance matrix is more expensive than computing its eigenvectors ($n > k$)!

Example

$n = 10^9, p = 10^8$

- Store $X^\top X$: 10^{16} B = 9000 TB
- Compute $X^\top X$: 10^{25} FLOPS

A US Supercomputer Just Broke The Exascale Barrier, Ranking Fastest in The World

TECH 07 June 2022 By PETER DOCKRILL

Frontier. (Oak Ridge National Laboratory/YouTube)

The US has succeeded in developing the world's first 'true' exascale supercomputer, honoring a pledge made by President Obama almost seven years ago, and ushering the world into a new era of computational capability.

Until now, the most speedy supercomputers in the world were still working in the petascale, achieving a quadrillion calculations per second. The exascale brings this to a whole new level, reaching a quintillion operations per second.

The Frontier supercomputer, built at the Department of Energy's Oak Ridge National Laboratory in Tennessee, has now become the world's first known supercomputer to demonstrate a processor speed of 1.1 exaFLOPS (1.1 quintillion floating point operations per second, or FLOPS).

PCA complexity

- Memory: store X and covariance matrix $X^\top X$: $\mathcal{O}(\max(np, p^2))$
- Runtime:
  - Compute $X^\top X$: $\mathcal{O}(np^2)$
  - Compute k eigenvectors of $X^\top X$ with power methods: $\mathcal{O}(kp^2)$

  Computing the covariance matrix is more expensive than computing its eigenvectors ($n > k$)!

Example

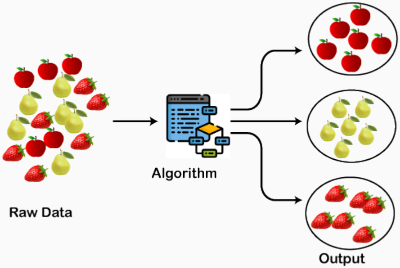
$n = 10^9, p = 10^8$

- Store $X^\top X$: 10^{16} B = 9000 TB
- Compute $X^\top X$: 10^{25} FLOPS *(Floating Point Operations per Second)*

  World's fastest computer (2022): 1.1 exaFLOPS = 10^{18} FLOPS
  $\rightarrow$ **115 days!**

Sommaire

Motivation



- Unsupervised learning
- Discover groups
- Reduce dimension

k-means definition

- Dataset $\{\boldsymbol{x}^{\mathbf{1}}, \ldots, \boldsymbol{x}^{\boldsymbol{n}}\} \subset \mathbb{R}^p$.

k-means definition

- Dataset $\{\boldsymbol{x^1}, \dots, \boldsymbol{x^n}\} \subset \mathbb{R}^p$.
- Find a cluster assignment $c_i \in \{1, \dots, k\}$ for all $i = 1, \dots, n$
- that minimizes the intra-cluster variance:

k-means definition

- Dataset $\{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n\} \subset \mathbb{R}^p$.
- Find a cluster assignment $c_i \in \{1, \ldots, k\}$ for all $i = 1, \ldots, n$
- that minimizes the intra-cluster variance:

$$\min_{c_i} \sum_{i=1}^{n} \|\boldsymbol{x}^i - \boldsymbol{\mu}_{c_i}\|^2,$$

k-means definition

- Dataset $\{\boldsymbol{x^1}, \dots, \boldsymbol{x^n}\} \subset \mathbb{R}^p$.
- Find a cluster assignment $c_i \in \{1, \dots, k\}$ for all $i = 1, \dots, n$
- that minimizes the intra-cluster variance:

$$\min_{c_i} \sum_{i=1}^{n} \|\boldsymbol{x^i} - \boldsymbol{\mu}_{c_i}\|^2,$$

where the $\boldsymbol{\mu}_j$, $j = 1, \dots, k$, are the centroids

$$\boldsymbol{\mu}_j = \frac{1}{|\{i : c_i = j\}|} \sum_{i:c_i=j} \boldsymbol{x^i}$$

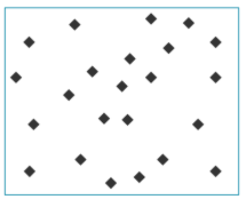
k-means definition

- Dataset $\{\boldsymbol{x}^{\mathbf{1}}, \dots, \boldsymbol{x}^{\boldsymbol{n}}\} \subset \mathbb{R}^p$.
- Find a cluster assignment $c_i \in \{1, \dots, k\}$ for all $i = 1, \dots, n$
- that minimizes the intra-cluster variance:

$$\min_{c_i} \sum_{i=1}^{n} \|\boldsymbol{x}^{\boldsymbol{i}} - \boldsymbol{\mu}_{c_i}\|^2,$$

where the $\boldsymbol{\mu}_j$, $j = 1, \dots, k$, are the centroids

$$\boldsymbol{\mu}_j = \frac{1}{|\{i : c_i = j\}|} \sum_{i : c_i = j} \boldsymbol{x}^{\boldsymbol{i}}$$



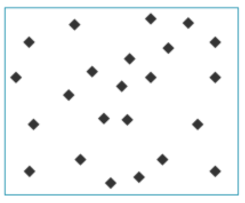
k-means definition

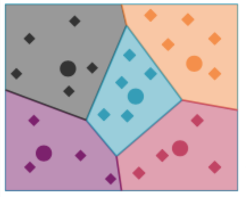
- Dataset $\{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n\} \subset \mathbb{R}^p$.
- Find a cluster assignment $c_i \in \{1, \ldots, k\}$ for all $i = 1, \ldots, n$
- that minimizes the intra-cluster variance:

$$\min_{c_i} \sum_{i=1}^{n} \|\boldsymbol{x}^i - \boldsymbol{\mu}_{c_i}\|^2,$$

where the $\boldsymbol{\mu}_j$, $j = 1, \ldots, k$, are the centroids

$$\boldsymbol{\mu}_j = \frac{1}{|\{i : c_i = j\}|} \sum_{i: c_i = j} \boldsymbol{x}^i$$





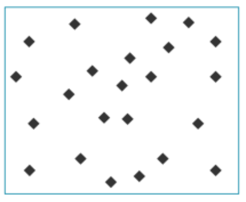
k-means definition

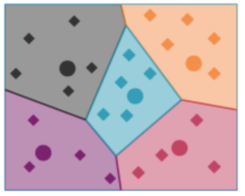
- Dataset $\{\boldsymbol{x}^1, \dots, \boldsymbol{x}^n\} \subset \mathbb{R}^p$.
- Find a cluster assignment $c_i \in \{1, \dots, k\}$ for all $i = 1, \dots, n$
- that minimizes the intra-cluster variance:

$$\min_{c_i} \sum_{i=1}^{n} \|\boldsymbol{x}^i - \boldsymbol{\mu}_{c_i}\|^2,$$

where the $\boldsymbol{\mu}_j$, $j = 1, \dots, k$, are the centroids

$$\boldsymbol{\mu}_j = \frac{1}{|\{i : c_i = j\}|} \sum_{i : c_i = j} \boldsymbol{x}^i$$





→ *Voronoi diagram*

k-means definition

- NP-hard problem!

k-means definition

- **NP-hard problem!** Approximate solution by iterating
    1. Assignment step: fix the centroids $\boldsymbol{\mu}_j$, optimize assignments c_i

$$\forall i = 1, \ldots, n, \quad c_i \leftarrow \text{argmin}_{c \in \{1, \ldots, k\}} \|\boldsymbol{x}^{\boldsymbol{i}} - \boldsymbol{\mu}_c\|$$

k-means definition

- **NP-hard problem!** Approximate solution by iterating
  1. Assignment step: fix the centroids $\boldsymbol{\mu}_j$, optimize assignments c_i

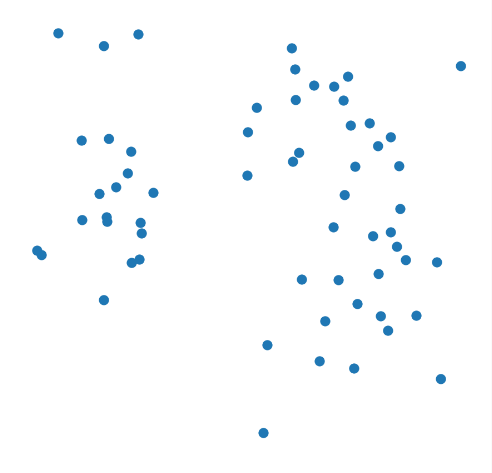
$$\forall i = 1, \ldots, n, \quad c_i \leftarrow \text{argmin}_{c \in \{1,\ldots,k\}} \|\boldsymbol{x}^{\boldsymbol{i}} - \boldsymbol{\mu}_c\|$$

  2. Update step: update the centroids

$$\forall i = 1, \ldots, k, \quad \boldsymbol{\mu}_i \leftarrow \frac{1}{|\{i : c_i = j\}|} \sum_{i : c_i = j} \boldsymbol{x}^{\boldsymbol{i}}$$

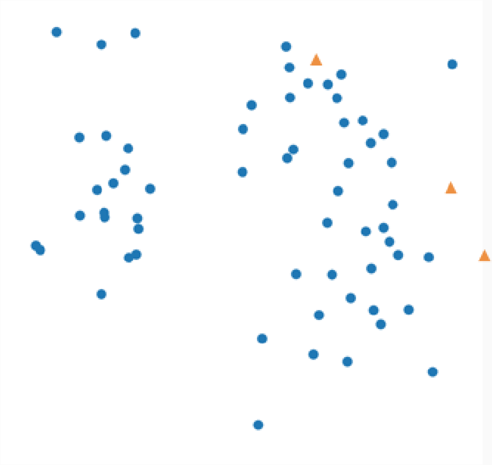
k-means example

$k = 3$



k-means example

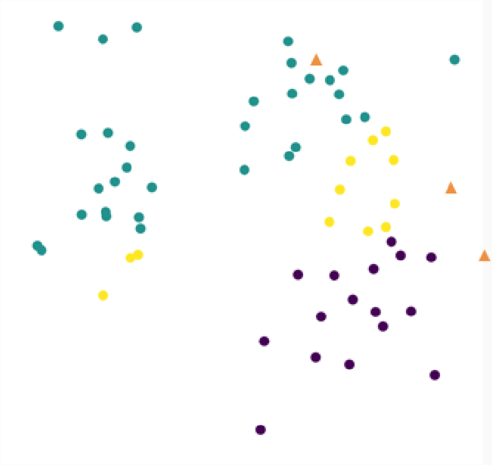
- Pick 3 centroids at random. $k = 3$



k-means example

- Assign each observation to the nearest centroid

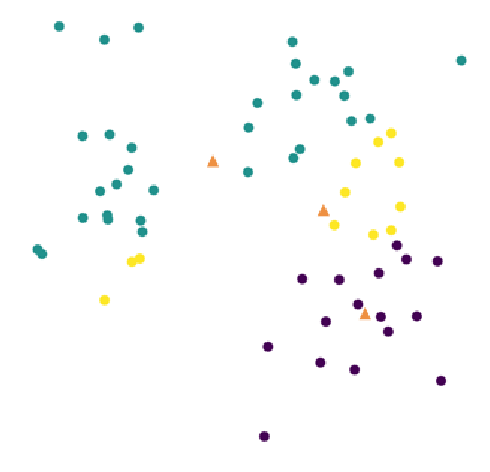
$k = 3$



k-means example

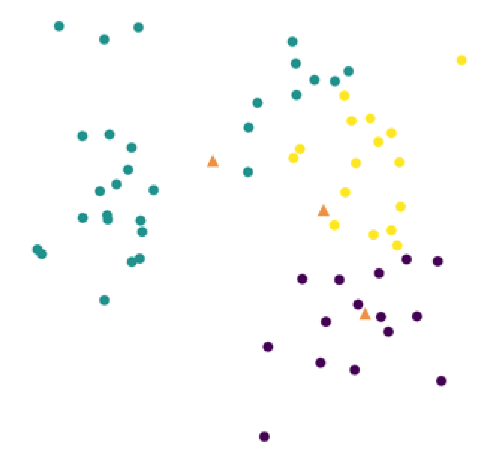
- Recompute centroids

$k = 3$



k-means example

- Re-assign each observation to the nearest centroid $k = 3$



k-means example

- Recompute centroids, and iterate process until convergence
  $k = 3$

k-means complexity

- Runtime:

k-means complexity

- Runtime:
  - Assignment step:

$$\forall i = 1, \ldots, n, \quad c_i \leftarrow \text{argmin}_{c \in \{1,\ldots,k\}} \|\boldsymbol{x}^{\boldsymbol{i}} - \boldsymbol{\mu}_c\|$$

k-means complexity

- Runtime:
  - **Assignment step**:

$$\forall i = 1, \dots, n, \quad c_i \leftarrow \text{argmin}_{c \in \{1, \dots, k\}} \|\boldsymbol{x}^{\boldsymbol{i}} - \boldsymbol{\mu}_c\|$$

Compute $n \times k$ distances in $\mathbb{R}^p$: $\mathcal{O}(knp)$

k-means complexity

- Runtime:
  - **Assignment step**:

$$\forall i = 1, \dots, n, \quad c_i \leftarrow \text{argmin}_{c \in \{1, \dots, k\}} \|\boldsymbol{x}^{\boldsymbol{i}} - \boldsymbol{\mu}_c\|$$

    Compute $n \times k$ distances in $\mathbb{R}^p$: $\mathcal{O}(knp)$
  - **Update step**:

$$\forall j = 1, \dots, k, \quad \boldsymbol{\mu}_j \leftarrow \frac{1}{|\{i : c_i = j\}|} \sum_{i : c_i = j} \boldsymbol{x}^{\boldsymbol{i}}$$

k-means complexity

- Runtime:
  - **Assignment step**:

$$\forall i = 1, \ldots, n, \quad c_i \leftarrow \text{argmin}_{c \in \{1,\ldots,k\}} \|\boldsymbol{x}^{\boldsymbol{i}} - \boldsymbol{\mu}_c\|$$

    Compute $n \times k$ distances in $\mathbb{R}^p$: $\mathcal{O}(knp)$
  - **Update step**:

$$\forall j = 1, \ldots, k, \quad \boldsymbol{\mu}_j \leftarrow \frac{1}{|\{i : c_i = j\}|} \sum_{i:c_i=j} \boldsymbol{x}^{\boldsymbol{i}}$$

    Sum n values in $\mathbb{R}^p$ for each centroid: $\mathcal{O}(knp)$

k-means complexity

- Runtime:
  - **Assignment step**:

$$\forall i = 1, \dots, n, \quad c_i \leftarrow \text{argmin}_{c \in \{1,\dots,k\}} \|\boldsymbol{x}^i - \boldsymbol{\mu}_c\|$$

    Compute $n \times k$ distances in $\mathbb{R}^p$: $\mathcal{O}(knp)$
  - **Update step**:

$$\forall j = 1, \dots, k, \quad \boldsymbol{\mu}_j \leftarrow \frac{1}{|\{i : c_i = j\}|} \sum_{i : c_i = j} \boldsymbol{x}^i$$

    Sum n values in $\mathbb{R}^p$ for each centroid: $\mathcal{O}(knp)$
  - Do T **iterations**: $\mathcal{O}(kTnp)$

k-means complexity

- Runtime:
  - **Assignment step**:

$$\forall i = 1, \ldots, n, \quad c_i \leftarrow \text{argmin}_{c \in \{1,\ldots,k\}} \|\boldsymbol{x}^{\boldsymbol{i}} - \boldsymbol{\mu}_c\|$$

    Compute $n \times k$ distances in $\mathbb{R}^p$: $\mathcal{O}(knp)$
  - **Update step**:

$$\forall j = 1, \ldots, k, \quad \boldsymbol{\mu}_j \leftarrow \frac{1}{|\{i : c_i = j\}|} \sum_{i:c_i=j} \boldsymbol{x}^{\boldsymbol{i}}$$

    Sum n values in $\mathbb{R}^p$ for each centroid: $\mathcal{O}(knp)$
  - Do T **iterations**: $\mathcal{O}(kTnp)$
- Memory:

k-means complexity

- Runtime:
  - **Assignment step**:

$$\forall i = 1, \dots, n, \quad c_i \leftarrow \text{argmin}_{c \in \{1,\dots,k\}} \|\boldsymbol{x}^{\boldsymbol{i}} - \boldsymbol{\mu}_c\|$$

    Compute $n \times k$ distances in $\mathbb{R}^p$: $\mathcal{O}(knp)$
  - **Update step**:

$$\forall j = 1, \dots, k, \quad \boldsymbol{\mu}_j \leftarrow \frac{1}{|\{i : c_i = j\}|} \sum_{i : c_i = j} \boldsymbol{x}^{\boldsymbol{i}}$$

    Sum n values in $\mathbb{R}^p$ for each centroid: $\mathcal{O}(knp)$
  - Do T **iterations**: $\mathcal{O}(kTnp)$
- Memory:
  - Store n cluster assignments and k centroids: $\mathcal{O}(n + kp)$

k-means complexity

- Runtime:
  - **Assignment step**:

$$\forall i = 1, \dots, n, \quad c_i \leftarrow \text{argmin}_{c \in \{1, \dots, k\}} \|\boldsymbol{x}^i - \boldsymbol{\mu}_c\|$$

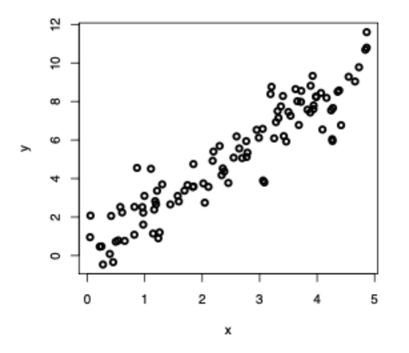
    Compute $n \times k$ distances in $\mathbb{R}^p$: $\mathcal{O}(knp)$
  - **Update step**:

$$\forall j = 1, \dots, k, \quad \boldsymbol{\mu}_j \leftarrow \frac{1}{|\{i : c_i = j\}|} \sum_{i : c_i = j} \boldsymbol{x}^i$$

    Sum n values in $\mathbb{R}^p$ for each centroid: $\mathcal{O}(knp)$
  - Do T **iterations**: $\mathcal{O}(kTnp)$
- Memory:
  - Store n cluster assignments and k centroids: $\mathcal{O}(n + kp)$
  - Store X: $\mathcal{O}(np)$

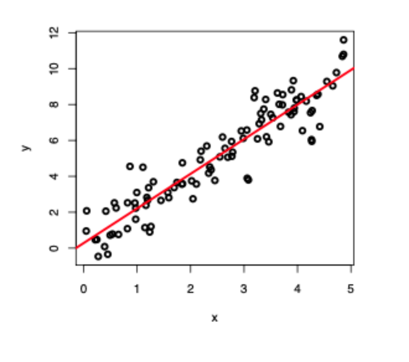
Sommaire

Motivation



- Predict a continuous output $y \in \mathbb{R}$ from an input $\boldsymbol{x} \in \mathbb{R}^p$

Motivation



- Predict a continuous output $y \in \mathbb{R}$ from an input $\boldsymbol{x} \in \mathbb{R}^p$

Linear regression

- Dataset:
  $\mathcal{S} = \{(\boldsymbol{x^1}, y^1), \ldots, (\boldsymbol{x^n}, y^n)\} \subset \mathbb{R}^p \times \mathbb{R} \Leftrightarrow X \in \mathbb{R}^{n \times p}, \boldsymbol{y} \in \mathbb{R}^n$

Linear regression

- Dataset:
  $\mathcal{S} = \{(\boldsymbol{x}^{\mathbf{1}}, y^1), \ldots, (\boldsymbol{x}^{\boldsymbol{n}}, y^n)\} \subset \mathbb{R}^p \times \mathbb{R} \Leftrightarrow X \in \mathbb{R}^{n \times p}, \boldsymbol{y} \in \mathbb{R}^n$
- Fit a linear function:

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x} = \sum_{j=1}^{p} \beta_j x_j$$

Linear regression

- Dataset:
  $\mathcal{S} = \{(\boldsymbol{x^1}, y^1), \ldots, (\boldsymbol{x^n}, y^n)\} \subset \mathbb{R}^p \times \mathbb{R} \Leftrightarrow X \in \mathbb{R}^{n \times p}, \boldsymbol{y} \in \mathbb{R}^n$
- Fit a linear function:

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x} = \sum_{j=1}^{p} \beta_j x_j$$

- Goodness of fit measured by residual sum of squares:

$$\begin{aligned}\widehat{\boldsymbol{\beta}}^{\text{OLS}} = \underset{\boldsymbol{\beta}}{\text{argmin}}\ \text{RSS}(\boldsymbol{\beta}) &= \underset{\boldsymbol{\beta}}{\text{argmin}} \sum_{i=1}^{n} (y^i - f_{\boldsymbol{\beta}}(\boldsymbol{x^i}))^2 \\ &= \underset{\boldsymbol{\beta}}{\text{argmin}} \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2\end{aligned}$$

Linear regression

- Dataset:
  $\mathcal{S} = \{(\boldsymbol{x}^{\boldsymbol{1}}, y^1), \dots, (\boldsymbol{x}^{\boldsymbol{n}}, y^n)\} \subset \mathbb{R}^p \times \mathbb{R} \Leftrightarrow X \in \mathbb{R}^{n \times p}, \boldsymbol{y} \in \mathbb{R}^n$
- Fit a linear function:

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x} = \sum_{j=1}^{p} \beta_j x_j$$

- Goodness of fit measured by residual sum of squares:

$$\begin{aligned}\widehat{\boldsymbol{\beta}}^{\mathsf{OLS}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \ \mathsf{RSS}(\boldsymbol{\beta}) &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} (y^i - f_{\boldsymbol{\beta}}(\boldsymbol{x}^{\boldsymbol{i}}))^2 \\ &= \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2\end{aligned}$$

- Solution:

$$\widehat{\boldsymbol{\beta}}^{\mathsf{OLS}} = (X^\top X)^{-1} X^\top \boldsymbol{y}$$

  (uniquely defined when $X^\top X$ invertible)

Ridge regression

- Hoerl and Kennard, 1970
- Ridge regression minimizes the regularized RSS:

$$\widehat{\boldsymbol{\beta}}^{\text{ridge}} = \underset{\boldsymbol{\beta}}{\text{argmin}} \ \text{RSS}(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} \beta_j^2$$

Ridge regression

- Hoerl and Kennard, 1970
- Ridge regression minimizes the regularized RSS:

$$\widehat{\boldsymbol{\beta}}^{\text{ridge}} = \underset{\boldsymbol{\beta}}{\text{argmin}} \ \text{RSS}(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} \beta_j^2$$

- Solution:

$$\widehat{\boldsymbol{\beta}}^{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top \boldsymbol{y}$$

$\rightarrow$ *unique and always exists !*

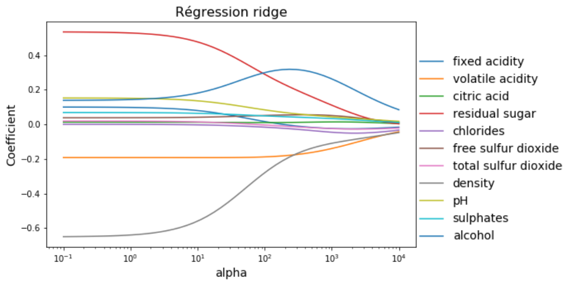
- Correlated features get similar weights

Limit cases

$$\widehat{\boldsymbol{\beta}}_\lambda^{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top \boldsymbol{y}$$

Corollary

- As $\lambda \to 0$, $\widehat{\boldsymbol{\beta}}_\lambda^{\text{ridge}} \to \widehat{\boldsymbol{\beta}}^{\text{OLS}}$ (low bias, high variance).
- As $\lambda \to +\infty$, $\widehat{\boldsymbol{\beta}}_\lambda^{\text{ridge}} \to 0$ (high bias, low variance).



Ridge regression complexity

$$\widehat{\boldsymbol{\beta}}_\lambda^{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top \boldsymbol{y}$$

- Compute $X^\top X + \lambda I$: $\mathcal{O}(np^2)$

When $n \gg p$, computing $X^\top X + \lambda I$ is more expensive than inverting it!

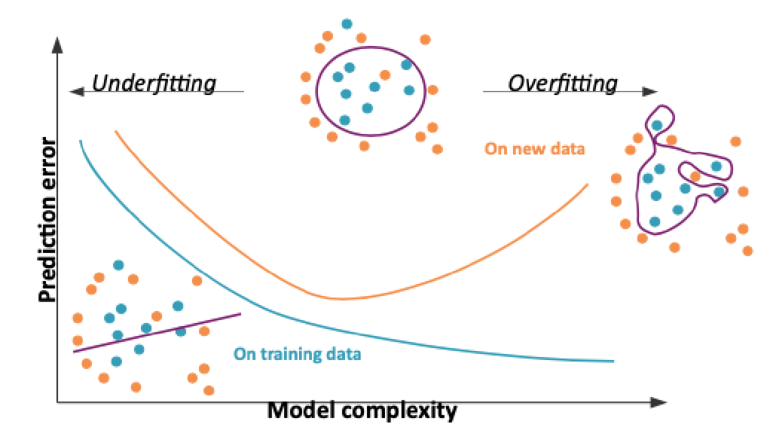
Ridge regression complexity

$$\widehat{\boldsymbol{\beta}}_{\lambda}^{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top \boldsymbol{y}$$

- Compute $X^\top X + \lambda I$: $\mathcal{O}(np^2)$
- Invert $X^\top X + \lambda I$: $\mathcal{O}(p^3)$

When $n \gg p$, computing $X^\top X + \lambda I$ is more expensive than inverting it!

Choice of λ

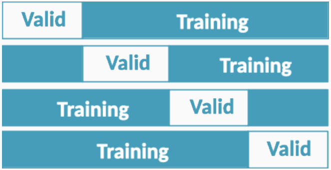


Choice of λ

- Data splitting strategies: cross-validation

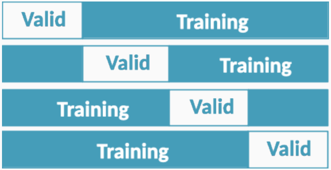
Choice of λ

- Data splitting strategies: cross-validation
  - Split the training set (of size n) into K equally-sized chunks



Choice of λ

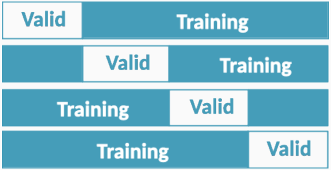
- Data splitting strategies: cross-validation
  - Split the training set (of size n) into K equally-sized chunks



  - K folds: one for testing, the $K - 1$ others for training

Choice of λ

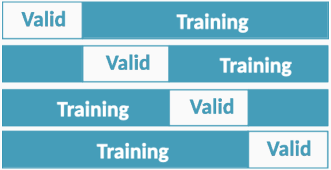
- Data splitting strategies: cross-validation
  - Split the training set (of size n) into K equally-sized chunks



  - K folds: one for testing, the $K-1$ others for training
  - Cross-validation score: average performance over the K folds

Choice of λ

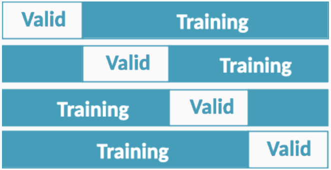
- Data splitting strategies: cross-validation
  - Split the training set (of size n) into K equally-sized chunks



  - K folds: one for testing, the $K-1$ others for training
  - Cross-validation score: average performance over the K folds
- For selection of λ: take a grid of values $(\lambda_1, \dots, \lambda_M)$ and choose the λ with the best cross-validation score.

Choice of λ

- Data splitting strategies: cross-validation
  - Split the training set (of size n) into K equally-sized chunks



  - K folds: one for testing, the $K-1$ others for training
  - Cross-validation score: average performance over the K folds
- For selection of λ: take a grid of values $(\lambda_1, \dots, \lambda_M)$ and choose the λ with the best cross-validation score.
- Multiplies complexity by KM!

Generalization: ℓ_2-regularized learning

- Generalization of the ridge regression to any loss:

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\boldsymbol{\beta}}(\boldsymbol{x^i}), y^i) + \lambda \|\boldsymbol{\beta}\|^2$$

Generalization: ℓ_2-regularized learning

- Generalization of the ridge regression to any loss:

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\boldsymbol{\beta}}(\boldsymbol{x^i}), y^i) + \lambda \|\boldsymbol{\beta}\|^2$$

- Empirical risk: $R(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\boldsymbol{\beta}}(\boldsymbol{x^i}), y^i)$

Generalization: ℓ_2-regularized learning

- Generalization of the ridge regression to any loss:

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\boldsymbol{\beta}}(\boldsymbol{x}^{\boldsymbol{i}}), y^i) + \lambda \|\boldsymbol{\beta}\|^2$$

- Empirical risk: $R(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\boldsymbol{\beta}}(\boldsymbol{x}^{\boldsymbol{i}}), y^i)$
- If the loss is convex, then the problem is strictly convex and has a unique global solution, which can be found numerically.

Generalization: ℓ_2-regularized learning

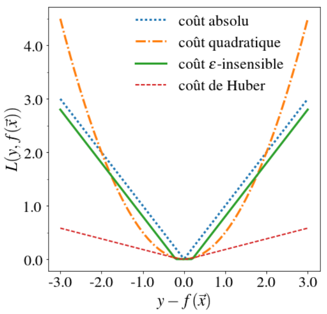
- Generalization of the ridge regression to any loss:

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\boldsymbol{\beta}}(\boldsymbol{x^i}), y^i) + \lambda \|\boldsymbol{\beta}\|^2$$

- Empirical risk: $R(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\boldsymbol{\beta}}(\boldsymbol{x^i}), y^i)$
- If the loss is convex, then the problem is strictly convex and has a unique global solution, which can be found numerically.

Losses for regression

- Square loss : $\ell(u, y) = (u - y)^2$
  $\rightarrow$ *Ridge regression*
- Absolute loss: $\ell(u, y) = |u - y|$
- ϵ-insensitive loss :
  $\ell(u, y) = (|u - y| - \epsilon)_+$
- Huber loss : mix quadratic/linear

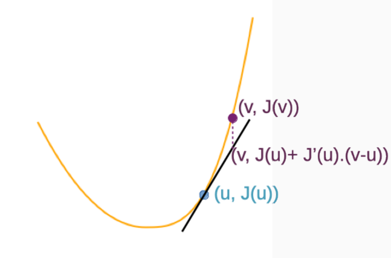


Gradient descent

If the loss is convex, then the problem is strictly convex and has a unique global solution, which can be found numerically.

- Assume the function to minimize is differentiable, then

$$J(v) \geq J(u) + \nabla J(u)^\top (v - u)$$

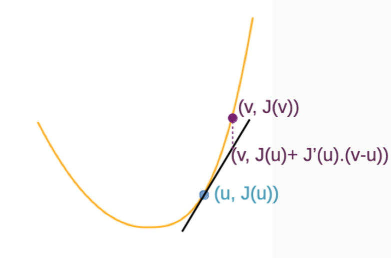


If the loss is convex, then the problem is strictly convex and has a unique global solution, which can be found numerically.

- Assume the function to minimize is differentiable, then

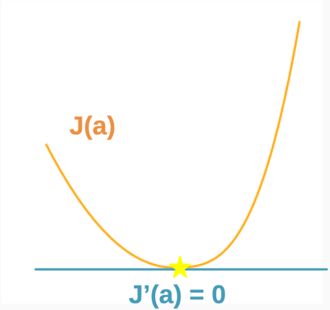
$$J(v) \geq J(u) + \nabla J(u)^\top (v - u)$$

- $\nabla J(u) = 0 \Leftrightarrow u$ minimizes J



Gradient descent

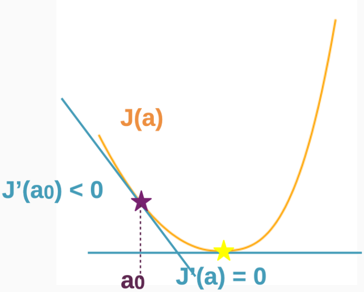
Idea: Minimize a differentiable, strictly convex function J by finding where its gradient is 0.



Gradient descent

Idea: Minimize a differentiable, strictly convex function J by finding where its gradient is 0.

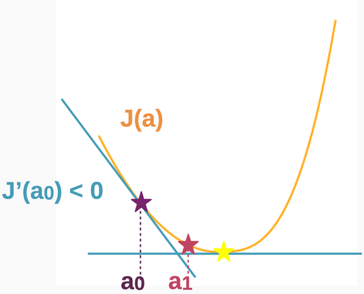
- Algorithm:
  - Pick a_0 randomly



Gradient descent

Idea: Minimize a differentiable, strictly convex function J by finding where its gradient is 0.

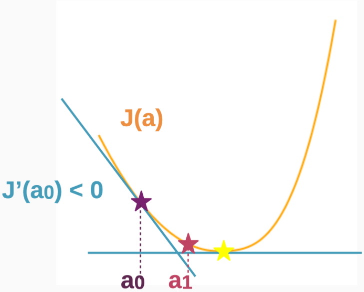
- Algorithm:
  - Pick a_0 randomly
  - Update $a_1 = a_0 - \alpha \nabla J(a_0)$



Gradient descent

Idea: Minimize a differentiable, strictly convex function J by finding where its gradient is 0.

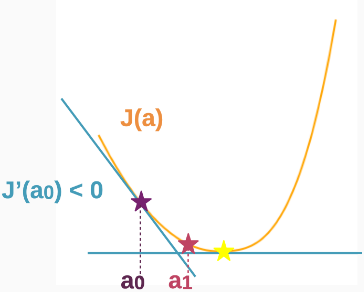
- Algorithm:
  - Pick a_0 randomly
  - Update $a_1 = a_0 - \alpha \nabla J(a_0)$
  - Repeat



Gradient descent

Idea: Minimize a differentiable, strictly convex function J by finding where its gradient is 0.

- Algorithm:
  - Pick a_0 randomly
  - Update $a_1 = a_0 - \alpha \nabla J(a_0)$
  - Repeat
  - Stop when $|\nabla J(a_0)| < \varepsilon$

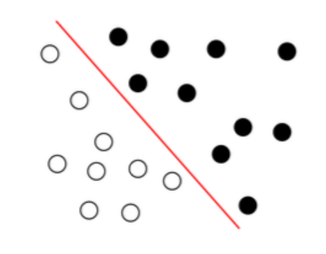


Sommaire

Motivation

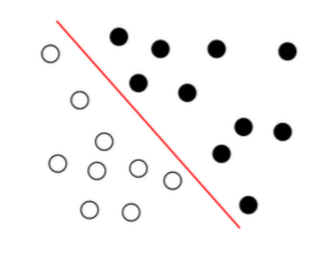
- Predict the category of data
- 2 or more (sometimes many) categories

Linear models for classification



- Training set $\mathcal{S} = \{(\boldsymbol{x}^{\mathbf{1}}, y^1), \ldots, (\boldsymbol{x}^{\boldsymbol{n}}, y^n)\} \subset \mathbb{R}^p \times \{-1, 1\}$

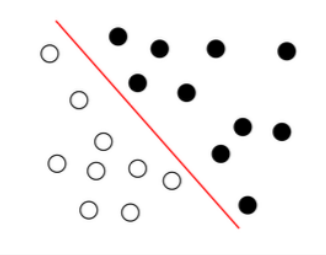
Linear models for classification



- Training set $\mathcal{S} = \{(\boldsymbol{x}^{\mathbf{1}}, y^1), \dots, (\boldsymbol{x}^{\boldsymbol{n}}, y^n)\} \subset \mathbb{R}^p \times \{-1, 1\}$
- Fit a linear function

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x}$$

Linear models for classification



- Training set $\mathcal{S} = \{(\boldsymbol{x}^{\mathbf{1}}, y^1), \ldots, (\boldsymbol{x}^{\boldsymbol{n}}, y^n)\} \subset \mathbb{R}^p \times \{-1, 1\}$
- Fit a linear function

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x}$$

- Prediction on a new point $\boldsymbol{x} \in \mathbb{R}^p$:

$$\begin{cases} +1 & \text{if } f_{\boldsymbol{\beta}}(\boldsymbol{x}) > 0\,, \\ -1 & \text{otherwise.} \end{cases}$$

The $0/1$ loss

- The $0/1$ loss measures if a prediction is correct or not:

$$\ell_{0/1}(f(\boldsymbol{x}), y)) = \mathbb{1}(yf(\boldsymbol{x}) < 0) = \begin{cases} 0 & \text{if } y = \text{sign}(f(\boldsymbol{x})) \\ 1 & \text{otherwise.} \end{cases}$$

The $0/1$ loss

- The $0/1$ loss measures if a prediction is correct or not:

$$\ell_{0/1}(f(\boldsymbol{x}), y)) = \mathbb{1}(yf(\boldsymbol{x}) < 0) = \begin{cases} 0 & \text{if } y = \text{sign}(f(\boldsymbol{x})) \\ 1 & \text{otherwise.} \end{cases}$$

- It is them tempting to learn $f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x}$ by solving:

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \underbrace{\frac{1}{n} \sum_{i=1}^{n} \ell_{0/1}(f_{\boldsymbol{\beta}}(\boldsymbol{x}^{\boldsymbol{i}}), y^i)}_{\text{misclassification rate}} + \underbrace{\lambda \|\boldsymbol{\beta}\|^2}_{\text{regularization}}$$

The $0/1$ loss

- The $0/1$ loss measures if a prediction is correct or not:

$$\ell_{0/1}(f(\boldsymbol{x}), y)) = \mathbb{1}(yf(\boldsymbol{x}) < 0) = \begin{cases} 0 & \text{if } y = \text{sign}(f(\boldsymbol{x})) \\ 1 & \text{otherwise.} \end{cases}$$

- It is them tempting to learn $f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x}$ by solving:

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \underbrace{\frac{1}{n} \sum_{i=1}^{n} \ell_{0/1}(f_{\boldsymbol{\beta}}(\boldsymbol{x}^{\boldsymbol{i}}), y^i)}_{\text{misclassification rate}} + \underbrace{\lambda \|\boldsymbol{\beta}\|^2}_{\text{regularization}}$$

- However:
  - The problem is non-smooth, and typically NP-hard to solve

The $0/1$ loss

- The $0/1$ loss measures if a prediction is correct or not:

$$\ell_{0/1}(f(\boldsymbol{x}), y)) = \mathbb{1}(yf(\boldsymbol{x}) < 0) = \begin{cases} 0 & \text{if } y = \text{sign}(f(\boldsymbol{x})) \\ 1 & \text{otherwise.} \end{cases}$$

- It is them tempting to learn $f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x}$ by solving:

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \underbrace{\frac{1}{n} \sum_{i=1}^{n} \ell_{0/1}(f_{\boldsymbol{\beta}}(\boldsymbol{x}^{\boldsymbol{i}}), y^i)}_{\text{misclassification rate}} + \underbrace{\lambda \|\boldsymbol{\beta}\|^2}_{\text{regularization}}$$

- However:
  - The problem is non-smooth, and typically NP-hard to solve
  - The regularization has no effect since the $0/1$ loss is invariant by scaling of $\boldsymbol{\beta}$

The logistic loss

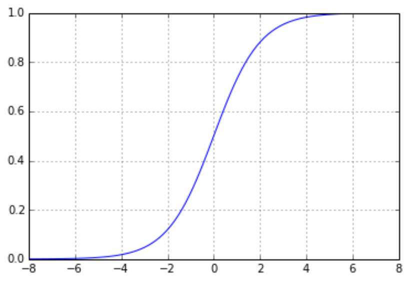
- An alternative is to define a probabilistic model of y parametrized by $f(\boldsymbol{x})$, e.g.:

$$\forall y \in \{-1, 1\}, \quad \mathbb{P}(y \,|\, f(\boldsymbol{x})) = \frac{1}{1 + e^{-yf(\boldsymbol{x})}} = \sigma(yf(\boldsymbol{x}))$$

The logistic loss

- An alternative is to define a probabilistic model of y parametrized by $f(\boldsymbol{x})$, e.g.:

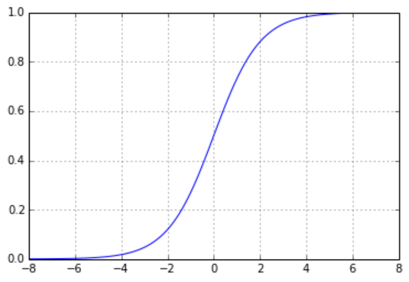
$$\forall y \in \{-1, 1\}, \quad \mathbb{P}(y \,|\, f(\boldsymbol{x})) = \frac{1}{1 + e^{-yf(\boldsymbol{x})}} = \sigma(yf(\boldsymbol{x}))$$



The logistic loss

- An alternative is to define a probabilistic model of y parametrized by $f(\boldsymbol{x})$, e.g.:

$$\forall y \in \{-1, 1\}, \quad \mathbb{P}(y \,|\, f(\boldsymbol{x})) = \frac{1}{1 + e^{-yf(\boldsymbol{x})}} = \sigma(yf(\boldsymbol{x}))$$



- The logistic loss is the negative conditional likelihood:

$$\ell_{\text{logistic}}(f(\boldsymbol{x}), y) = -\ln p(y \,|\, f(\boldsymbol{x})) = \ln(1 + e^{-yf(\boldsymbol{x})})$$

Ridge logistic regression

- Cessie and Houwelingen (1992)

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} J(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + e^{-y^i \boldsymbol{\beta}^\top \boldsymbol{x}^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

Ridge logistic regression

- Cessie and Houwelingen (1992)

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} J(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} \ln(1 + e^{-y^i \boldsymbol{\beta}^\top \boldsymbol{x}^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

- Can be interpreted as a regularized conditional maximum likelihood estimator

Ridge logistic regression

- Cessie and Houwelingen (1992)

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} J(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} \ln(1 + e^{-y^i \boldsymbol{\beta}^\top \boldsymbol{x}^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

- Can be interpreted as a regularized conditional maximum likelihood estimator
- No explicit solution, but smooth convex optimization problem that can be solved numerically

Newton-Raphson iteratins

- Goal: minimize J convex, differentiable

Newton-Raphson iteratins

- Goal: minimize J convex, differentiable
- Gradient descent:

$$u^{\text{new}} \leftarrow u^{\text{old}} - \alpha \nabla J(u^{\text{old}})$$

Newton-Raphson iteratins

- Goal: minimize J convex, differentiable
- Gradient descent:

$$u^{\text{new}} \leftarrow u^{\text{old}} - \alpha \nabla J(u^{\text{old}})$$

- Assume J is twice differentiable
  - Second-order Taylor's expansion:

$$J(v) \approx J(u) \nabla J(u)^\top (v-u) + \frac{1}{2}(v-u)^\top \nabla^2 J(u)^\top (v-u)$$

Newton-Raphson iteratins

- Goal: minimize J convex, differentiable
- Gradient descent:

$$u^{\text{new}} \leftarrow u^{\text{old}} - \alpha \nabla J(u^{\text{old}})$$

- Assume J is twice differentiable
  - Second-order Taylor's expansion:

$$J(v) \approx J(u) \nabla J(u)^\top (v-u) + \frac{1}{2}(v-u)^\top \nabla^2 J(u)^\top (v-u) = g(v)$$

Newton-Raphson iteratins

- Goal: minimize J convex, differentiable
- Gradient descent:

$$u^{\text{new}} \leftarrow u^{\text{old}} - \alpha \nabla J(u^{\text{old}})$$

- Assume J is twice differentiable
  - Second-order Taylor's expansion:

$$J(v) \approx J(u)\nabla J(u)^\top (v-u) + \frac{1}{2}(v-u)^\top \nabla^2 J(u)^\top (v-u) = g(v)$$

  - Minimum in v:

$$\nabla g(v) = \nabla J(u) + \nabla^2 J(u)^\top (v-u)$$
$$\nabla g(v) = 0 \Leftrightarrow v = u - (\nabla^2 J(u))^{-1} \nabla J(u).$$

Newton-Raphson iteratins

- Goal: minimize J convex, differentiable
- Gradient descent:

$$u^{\text{new}} \leftarrow u^{\text{old}} - \alpha \nabla J(u^{\text{old}})$$

- Assume J is twice differentiable
  - Second-order Taylor's expansion:

$$J(v) \approx J(u) \nabla J(u)^\top (v-u) + \frac{1}{2}(v-u)^\top \nabla^2 J(u)^\top (v-u) = g(v)$$

  - Minimum in v:

$$\nabla g(v) = \nabla J(u) + \nabla^2 J(u)^\top (v-u)$$
$$\nabla g(v) = 0 \Leftrightarrow v = u - (\nabla^2 J(u))^{-1} \nabla J(u).$$

  - Take $\alpha = (\nabla^2 J(u^{\text{old}}))^{-1}$ in the gradient step

Solving ridge logistic regression

$$\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} \ln(1 + e^{-y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}}) + \lambda \|\boldsymbol{\beta}\|_2^2$$

Solving ridge logistic regression

$$\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} \ln(1 + e^{-y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}}) + \lambda \|\boldsymbol{\beta}\|_2^2$$

- Solve with Newton-Raphson iterations

$$\begin{aligned}
\nabla_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) &= -\frac{1}{n} \sum_{i=1}^{n} \frac{y^i \boldsymbol{x^i}}{1 + e^{y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}}} + 2\lambda \boldsymbol{\beta} \\
&= -\frac{1}{n} \sum_{i=1}^{n} y^i [1 - \mathbb{P}_{\boldsymbol{\beta}}(y^i \mid \boldsymbol{x^i})] \boldsymbol{x^i} + 2\lambda \boldsymbol{\beta} \\
\nabla^2_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) &= \frac{1}{n} \sum_{i=1}^{n} \frac{\boldsymbol{x^i} \boldsymbol{x}^{i\top} e^{y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}}}{(1 + e^{y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}})^2} + 2\lambda I \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbb{P}_{\boldsymbol{\beta}}(1 \mid \boldsymbol{x^i})(1 - \mathbb{P}_{\boldsymbol{\beta}}(1 \mid \boldsymbol{x^i})) \boldsymbol{x^i} \boldsymbol{x}^{i\top} + 2\lambda I
\end{aligned}$$

Solving ridge logistic regression (cont.)

$$\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + e^{-y^i \boldsymbol{\beta}^\top \boldsymbol{x}^i}) + \lambda \|\boldsymbol{\beta}\|_2^2$$

- Solve with Newton-Raphson iterations

$$\boldsymbol{\beta}^{\text{new}} \leftarrow \boldsymbol{\beta}^{\text{old}} - [\nabla^2_{\boldsymbol{\beta}} J(\boldsymbol{\beta}^{\text{old}})]^{-1} \nabla_{\boldsymbol{\beta}} J(\boldsymbol{\beta}^{\text{old}}) \,.$$

Solving ridge logistic regression (cont.)

$$\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + e^{-y^i \boldsymbol{\beta}^\top \boldsymbol{x}^i}) + \lambda \|\boldsymbol{\beta}\|_2^2$$

- Solve with Newton-Raphson iterations

$$\boldsymbol{\beta}^{\text{new}} \leftarrow \boldsymbol{\beta}^{\text{old}} - [\nabla^2_{\boldsymbol{\beta}} J(\boldsymbol{\beta}^{\text{old}})]^{-1} \nabla_{\boldsymbol{\beta}} J(\boldsymbol{\beta}^{\text{old}})\,.$$

- Each step is equivalent to solving a weighted ridge regression problem → iteratively reweighted least squares (IRLS).

Solving ridge logistic regression (cont.)

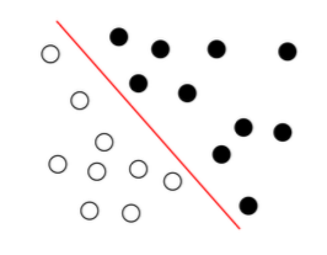
$$\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} \ln(1 + e^{-y^i \boldsymbol{\beta}^\top \boldsymbol{x}^i}) + \lambda \|\boldsymbol{\beta}\|_2^2$$

- Solve with Newton-Raphson iterations

$$\boldsymbol{\beta}^{\text{new}} \leftarrow \boldsymbol{\beta}^{\text{old}} - [\nabla^2_{\boldsymbol{\beta}} J(\boldsymbol{\beta}^{\text{old}})]^{-1} \nabla_{\boldsymbol{\beta}} J(\boldsymbol{\beta}^{\text{old}}) \,.$$

- Each step is equivalent to solving a weighted ridge regression problem $\rightarrow$ iteratively reweighted least squares (IRLS).
- Complexity $\mathcal{O}(T(np^2 + p^3))$

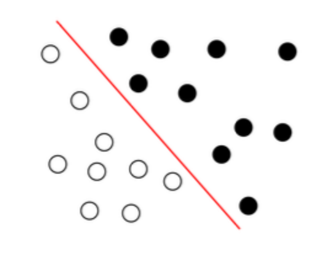
Large-margin classifiers



- For any $f : \mathbb{R}^p \to \mathbb{R}$, the margin of f on an $(\boldsymbol{x}, y)$ pair is

$$yf(\boldsymbol{x})$$

Large-margin classifiers



- For any $f : \mathbb{R}^p \to \mathbb{R}$, the margin of f on an $(\boldsymbol{x}, y)$ pair is

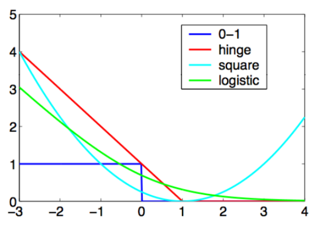
$$yf(\boldsymbol{x})$$

- Large-margin classifiers: maximize $yf(\boldsymbol{x})$

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \phi(y^i f_{\boldsymbol{\beta}}(\boldsymbol{x}^{\boldsymbol{i}})) + \lambda \boldsymbol{\beta}^\top \boldsymbol{\beta}$$

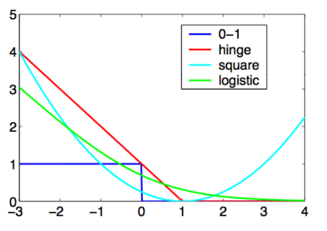
for a convex, non-increasing function $\phi : \mathbb{R} \to \mathbb{R}+$

Loss function examples



Loss	Method	$\phi(u)$
0-1	none	$1(u \leq 0)$
Hinge	Support vector machine (SVM)	$\max(1-u, 0)$
Logistic	Logistic regression	$\log(1+e^{-u})$
Square	Ridge regression	$(1-u)^2$
Exponential	Boosting	e^{-u}

Which ϕ?



- Computation
  - ϕ convex means we need to solve a convex optimization problem.
  - A ”good” ϕ may be one which allows for fast optimization
- Theory
  - Most ϕ lead to consistent estimators
  - Some may be more efficient

Linear SVM

- Boser et al. (1992)

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \quad \sum_{i=1}^{n} \max(0, 1 - y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

Linear SVM

- Boser et al. (1992)

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \quad \sum_{i=1}^{n} \max(0, 1 - y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

- Non-smooth convex optimization problem (quadratic program)

Linear SVM

- Boser et al. (1992)

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \quad \sum_{i=1}^{n} \max(0, 1 - y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

- Non-smooth convex optimization problem (quadratic program)
- Equivalent to the dual problem

$$\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k (\boldsymbol{x^j}^\top \boldsymbol{x^k})$$

$$\text{s.t.} \quad 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \text{ for } i = 1, \ldots, n \text{ and } \sum_{i=1}^{n} \alpha_i y^i = 0.$$

Linear SVM

- Boser et al. (1992)

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \quad \sum_{i=1}^{n} \max(0, 1 - y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}) + \lambda\|\boldsymbol{\beta}\|^2$$

- Non-smooth convex optimization problem (quadratic program)
- Equivalent to the dual problem

$$\max_{\alpha\in\mathbb{R}^n} 2\sum_{i=1}^{n}\alpha_i - \sum_{j,k=1}^{n}\alpha_j\alpha_k y^j y^k (\boldsymbol{x^j}^\top \boldsymbol{x^k})$$

$$\text{s.t.} \quad 0 \leq y^i\alpha_i \leq \frac{1}{2\lambda} \text{ for } i = 1, \ldots, n \text{ and } \sum_{i=1}^{n}\alpha_i y^i = 0.$$

- Solution: $\boldsymbol{\beta}^* = \sum_{j=1}^{n}\alpha_j y^j \boldsymbol{x^j} \qquad f_{\boldsymbol{\beta}^*}(\boldsymbol{x}) = \boldsymbol{\beta}^{*\top}\boldsymbol{x} = \sum_{j=1}^{n}\alpha_j y^j \boldsymbol{x^j}^\top \boldsymbol{x}$

Linear SVM

- Boser et al. (1992)

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \quad \sum_{i=1}^{n} \max(0, 1 - y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

- Non-smooth convex optimization problem (quadratic program)
- Equivalent to the dual problem

$$\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k (\boldsymbol{x^j}^\top \boldsymbol{x^k})$$

$$\text{s.t.} \quad 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \text{ for } i = 1, \dots, n \text{ and } \sum_{i=1}^{n} \alpha_i y^i = 0.$$

- Solution: $\boldsymbol{\beta}^* = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j} \qquad f_{\boldsymbol{\beta}^*}(\boldsymbol{x}) = \boldsymbol{\beta}^{*\top} \boldsymbol{x} = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j}^\top \boldsymbol{x}$

Complexity (training)

Linear SVM

- Boser et al. (1992)

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \quad \sum_{i=1}^{n} \max(0, 1 - y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

- Non-smooth convex optimization problem (quadratic program)
- Equivalent to the dual problem

$$\max_{\alpha\in\mathbb{R}^n} 2\sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k (\boldsymbol{x^j}^\top \boldsymbol{x^k})$$

$$\text{s.t.} \quad 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \text{ for } i = 1, \dots, n \text{ and } \sum_{i=1}^{n} \alpha_i y^i = 0.$$

- Solution: $\boldsymbol{\beta}^* = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j} \quad f_{\boldsymbol{\beta}^*}(\boldsymbol{x}) = \boldsymbol{\beta}^{*\top} \boldsymbol{x} = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j}^\top \boldsymbol{x}$

Complexity (training)

- Memory: $\mathcal{O}(n^2)$ to store $XX^\top$

Linear SVM

- Boser et al. (1992)

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \quad \sum_{i=1}^{n} \max(0, 1 - y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

- Non-smooth convex optimization problem (quadratic program)
- Equivalent to the dual problem

$$\max_{\alpha\in\mathbb{R}^n} 2\sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k (\boldsymbol{x^j}^\top \boldsymbol{x^k})$$

$$\text{s.t.} \quad 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \text{ for } i = 1, \ldots, n \text{ and } \sum_{i=1}^{n} \alpha_i y^i = 0.$$

- Solution: $\boldsymbol{\beta}^* = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j} \qquad f_{\boldsymbol{\beta}^*}(\boldsymbol{x}) = \boldsymbol{\beta}^{*\top} \boldsymbol{x} = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j}^\top \boldsymbol{x}$

Complexity (training)

- Memory: $\mathcal{O}(n^2)$ to store $XX^\top$
- Runtime: $\mathcal{O}(n^3)$ to find α^*

Linear SVM

- Boser et al. (1992)

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \quad \sum_{i=1}^{n} \max(0, 1 - y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

- Non-smooth convex optimization problem (quadratic program)
- Equivalent to the dual problem

$$\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k (\boldsymbol{x^j}^\top \boldsymbol{x^k})$$

$$\text{s.t.} \quad 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \text{ for } i = 1, \dots, n \text{ and } \sum_{i=1}^{n} \alpha_i y^i = 0.$$

- Solution: $\boldsymbol{\beta}^* = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j} \quad f_{\boldsymbol{\beta}^*}(\boldsymbol{x}) = \boldsymbol{\beta}^{*\top} \boldsymbol{x} = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j}^\top \boldsymbol{x}$

Complexity (training)

- Memory: $\mathcal{O}(n^2)$ to store $XX^\top$
- Runtime: $\mathcal{O}(n^3)$ to find α^*

Complexity (prediction)

Linear SVM

- Boser et al. (1992)

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \quad \sum_{i=1}^{n} \max(0, 1 - y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

- Non-smooth convex optimization problem (quadratic program)
- Equivalent to the dual problem

$$\max_{\alpha\in\mathbb{R}^n} 2\sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k (\boldsymbol{x^j}^\top \boldsymbol{x^k})$$

$$\text{s.t.} \quad 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \text{ for } i = 1, \ldots, n \text{ and } \sum_{i=1}^{n} \alpha_i y^i = 0.$$

- Solution: $\boldsymbol{\beta}^* = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j} \qquad f_{\boldsymbol{\beta}^*}(\boldsymbol{x}) = \boldsymbol{\beta}^{*\top} \boldsymbol{x} = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j}^\top \boldsymbol{x}$

Complexity (training)

- Memory: $\mathcal{O}(n^2)$ to store $XX^\top$
- Runtime: $\mathcal{O}(n^3)$ to find α^*

Complexity (prediction)

- Primal: $\mathcal{O}(p)$ for $(\boldsymbol{\beta}^*)^\top \boldsymbol{x}$

Linear SVM

- Boser et al. (1992)

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \quad \sum_{i=1}^{n} \max(0, 1 - y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}) + \lambda \|\boldsymbol{\beta}\|^2$$

- Non-smooth convex optimization problem (quadratic program)
- Equivalent to the dual problem

$$\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k (\boldsymbol{x^j}^\top \boldsymbol{x^k})$$

$$\text{s.t.} \quad 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \text{ for } i = 1, \ldots, n \text{ and } \sum_{i=1}^{n} \alpha_i y^i = 0.$$

- Solution: $\boldsymbol{\beta}^* = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j} \quad f_{\boldsymbol{\beta}^*}(\boldsymbol{x}) = \boldsymbol{\beta}^{*\top} \boldsymbol{x} = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j}^\top \boldsymbol{x}$

Complexity (training)

- Memory: $\mathcal{O}(n^2)$ to store $XX^\top$
- Runtime: $\mathcal{O}(n^3)$ to find α^*

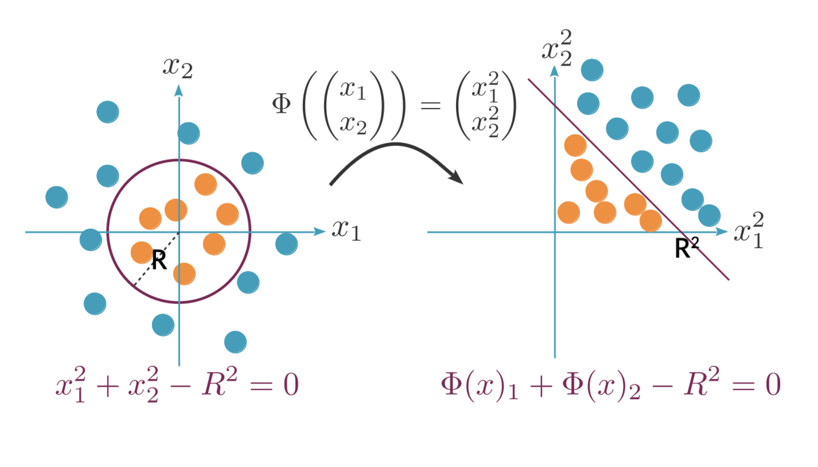
Complexity (prediction)

- Primal: $\mathcal{O}(p)$ for $(\boldsymbol{\beta}^*)^\top \boldsymbol{x}$
- Dual: $\mathcal{O}(np)$ for $(\alpha^*)^\top X \boldsymbol{x}$

Sommaire

Motivation

Non-linear mapping to a feature space



SVM in the feature space

$$\phi : \mathbb{R}^p \to \mathcal{H}$$

- Training:

$$\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k (\boldsymbol{x^j}^\top \boldsymbol{x^k})$$

$$\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k \langle \phi(\boldsymbol{x^j}), \phi(\boldsymbol{x^k}) \rangle_{\mathcal{H}}$$

SVM in the feature space

$$\phi : \mathbb{R}^p \to \mathcal{H}$$

- Training:

$$\max_{\alpha \in \mathbb{R}^n} 2\sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k (\boldsymbol{x^j}^\top \boldsymbol{x^k})$$

$$\max_{\alpha \in \mathbb{R}^n} 2\sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k \langle \phi(\boldsymbol{x^j}), \phi(\boldsymbol{x^k}) \rangle_{\mathcal{H}}$$

- **Predict** with the decision function

$$f_{\boldsymbol{\beta}^*}(x) = \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j}^\top x$$

$$f_{\boldsymbol{\beta}^*}(x) = \sum_{j=1}^{n} \alpha_j y^j \langle \phi(\boldsymbol{x^j}), \phi(x) \rangle_{\mathcal{H}}$$

Kernel SVM

$$k : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$$
$$(x, x') \mapsto k(x, x') = \langle \phi(x), \phi(x') \rangle$$

- Training:

$$\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k \langle \phi(\boldsymbol{x^j}), \phi(\boldsymbol{x^k}) \rangle_{\mathcal{H}}$$
$$\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k k(\boldsymbol{x^j}, \boldsymbol{x^k})$$

Kernel SVM

$$k : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$$
$$(x, x') \mapsto k(x, x') = \langle \phi(x), \phi(x') \rangle$$

- Training:

$$\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k \langle \phi(\boldsymbol{x^j}), \phi(\boldsymbol{x^k}) \rangle_{\mathcal{H}}$$

$$\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{j,k=1}^{n} \alpha_j \alpha_k y^j y^k k(\boldsymbol{x^j}, \boldsymbol{x^k})$$

- **Predict** with the decision function

$$f_{\boldsymbol{\beta}^*}(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j y^j \langle \phi(\boldsymbol{x^j}), \phi(\boldsymbol{x}) \rangle_{\mathcal{H}}$$

$$f_{\boldsymbol{\beta}^*}(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j y^j k(\boldsymbol{x^j}, \boldsymbol{x})$$

Kernel trick

- k may be quite efficient to compute, even if $\mathcal{H}$ is a very high-dimensional or even infinite-dimensional space.

Kernel trick

- k may be quite efficient to compute, even if $\mathcal{H}$ is a very high-dimensional or even infinite-dimensional space.
- For any positive semi-definite function k, there exists a feature space $\mathcal{H}$ and a feature map ϕ such that

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \rangle_{\mathcal{H}}$$

Kernel trick

- k may be quite efficient to compute, even if $\mathcal{H}$ is a very high-dimensional or even infinite-dimensional space.
- For any positive semi-definite function k, there exists a feature space $\mathcal{H}$ and a feature map ϕ such that

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \rangle_{\mathcal{H}}$$

- Hence you can define mappings implicitely.

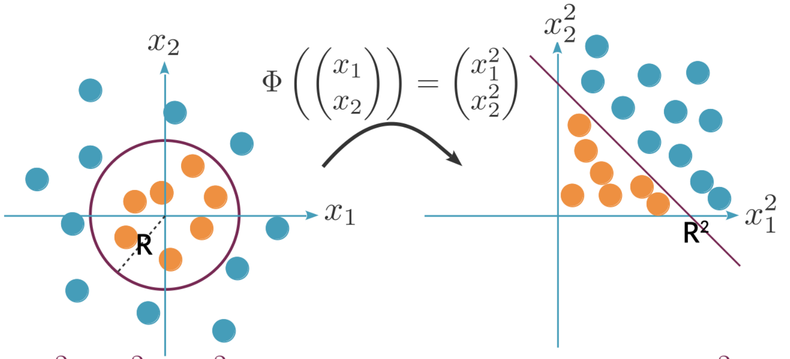
Kernel trick

- k may be quite efficient to compute, even if $\mathcal{H}$ is a very high-dimensional or even infinite-dimensional space.
- For any positive semi-definite function k, there exists a feature space $\mathcal{H}$ and a feature map ϕ such that

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x'}) \rangle_{\mathcal{H}}$$

- Hence you can define mappings implicitely.
- Kernel trick: algorithms that only involve the samples through their dot products can be rewritten using kernels in such a way that they can be applied in the initial space without ever computing the mapping ϕ.

Non-linear mapping to a feature space



$x_1^2 + x_2^2 - R^2 = 0$ $\Phi(x)_1 + \Phi(x)_2 - R^2 = 0$

$$K(\boldsymbol{x}, \boldsymbol{x}') = \left\langle \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix}, \begin{pmatrix} {x'_1}^2 \\ {x'_2}^2 \end{pmatrix} \right\rangle = x_1^2 {x'_1}^2 + x_2^2 {x'_2}^2$$

[1]

Linear	$k(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{x}^\top \boldsymbol{x'}$
Polynomial	$k(\boldsymbol{x}, \boldsymbol{x'}) = (\boldsymbol{x}^\top \boldsymbol{x'} + c)^d$
Gaussian	$k(\boldsymbol{x}, \boldsymbol{x'}) = \exp(-\frac{\lVert \boldsymbol{x} - \boldsymbol{x'} \rVert^2}{2\sigma^2})$
Min/max	$k(\boldsymbol{x}, \boldsymbol{x'}) = \sum_{j=1}^{p} \frac{\min(\lvert x_j \rvert, \lvert x'_j \rvert)}{\max(\lvert x_j \rvert, \lvert x'_j \rvert)}$

Kernel ridge regression (KRR)

- Ridge regression in input space $\mathbb{R}^p$:

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{x}^\top \widehat{\boldsymbol{\beta}}^{\text{ridge}} = \boldsymbol{x}^\top \underbrace{(X^\top X + \lambda I)^{-1}}_{p \times p} X^\top \boldsymbol{y},$$

Kernel ridge regression (KRR)

- Ridge regression in input space $\mathbb{R}^p$:

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{x}^\top \widehat{\boldsymbol{\beta}}^{\text{ridge}} = \boldsymbol{x}^\top \underbrace{(X^\top X + \lambda I)^{-1}}_{p \times p} X^\top \boldsymbol{y},$$

- In a feature space of dimension $\mathbb{R}^d$:

$$\begin{aligned} f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \Phi(\boldsymbol{x})^\top \widehat{\boldsymbol{\beta}} &= \Phi(\boldsymbol{x})^\top \underbrace{(\Phi(X)^\top \Phi(X) + \lambda I)^{-1}}_{d \times d} \Phi(X)^\top \boldsymbol{y} \\ &= \Phi(\boldsymbol{x})^\top \Phi(X)^\top \underbrace{(\Phi(X)\Phi(X)^\top + \lambda I)^{-1}}_{n \times n} \boldsymbol{y} \end{aligned}$$

Kernel ridge regression (KRR)

- Ridge regression in input space $\mathbb{R}^p$:

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{x}^\top \widehat{\boldsymbol{\beta}}^{\text{ridge}} = \boldsymbol{x}^\top \underbrace{(X^\top X + \lambda I)^{-1}}_{p \times p} X^\top \boldsymbol{y},$$

- In a feature space of dimension $\mathbb{R}^d$:

$$\begin{aligned} f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \Phi(\boldsymbol{x})^\top \widehat{\boldsymbol{\beta}} &= \Phi(\boldsymbol{x})^\top \underbrace{(\Phi(X)^\top \Phi(X) + \lambda I)^{-1}}_{d \times d} \Phi(X)^\top \boldsymbol{y} \\ &= \Phi(\boldsymbol{x})^\top \Phi(X)^\top \underbrace{(\Phi(X)\Phi(X)^\top + \lambda I)^{-1}}_{n \times n} \boldsymbol{y} \end{aligned}$$

- Ridge regression in sample space $\mathbb{R}^n$:

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \kappa \underbrace{(K + \lambda I)^{-1}}_{n \times n} \boldsymbol{y}, \quad \kappa_i = k(\boldsymbol{x}, \boldsymbol{x}^{\boldsymbol{i}}), \quad K_{ij} = k(\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x}^j)$$

Complexity of KRR

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \kappa(K + \lambda I)^{-1}\boldsymbol{y}, \quad \kappa_i = k(\boldsymbol{x}, \boldsymbol{x}^{\boldsymbol{i}}), \quad K_{ij} = k(\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x}^j)$$

- Computing K: $\mathcal{O}(pn^2)$

Complexity of KRR

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \kappa(K + \lambda I)^{-1}\boldsymbol{y}, \quad \kappa_i = k(\boldsymbol{x}, \boldsymbol{x^i}), \quad K_{ij} = k(\boldsymbol{x^i}, \boldsymbol{x}^j)$$

- Computing K: $\mathcal{O}(pn^2)$
- Storing K: $\mathcal{O}(n^2)$

Complexity of KRR

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \kappa(K + \lambda I)^{-1}\boldsymbol{y}, \quad \kappa_i = k(\boldsymbol{x}, \boldsymbol{x}^{\boldsymbol{i}}), \quad K_{ij} = k(\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x}^{j})$$

- Computing K: $\mathcal{O}(pn^2)$
- Storing K: $\mathcal{O}(n^2)$
- Inverting $K + \lambda I$: $\mathcal{O}(n^3)$

Complexity of KRR

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \kappa(K + \lambda I)^{-1}\boldsymbol{y}, \quad \kappa_i = k(\boldsymbol{x}, \boldsymbol{x^i}), \quad K_{ij} = k(\boldsymbol{x^i}, \boldsymbol{x^j})$$

- Computing K: $\mathcal{O}(pn^2)$
- Storing K: $\mathcal{O}(n^2)$
- Inverting $K + \lambda I$: $\mathcal{O}(n^3)$
- Computing a prediction for one sample:

Complexity of KRR

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \kappa(K + \lambda I)^{-1}\boldsymbol{y}, \quad \kappa_i = k(\boldsymbol{x}, \boldsymbol{x^i}), \quad K_{ij} = k(\boldsymbol{x^i}, \boldsymbol{x^j})$$

- Computing K: $\mathcal{O}(pn^2)$
- Storing K: $\mathcal{O}(n^2)$
- Inverting $K + \lambda I$: $\mathcal{O}(n^3)$
- Computing a prediction for one sample:
  - Computing κ: $\mathcal{O}(nd)$

Complexity of KRR

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \kappa(K + \lambda I)^{-1}\boldsymbol{y}, \quad \kappa_i = k(\boldsymbol{x}, \boldsymbol{x}^{\boldsymbol{i}}), \quad K_{ij} = k(\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{x}^j)$$

- Computing K: $\mathcal{O}(pn^2)$
- Storing K: $\mathcal{O}(n^2)$
- Inverting $K + \lambda I$: $\mathcal{O}(n^3)$
- Computing a prediction for one sample:
  - Computing κ: $\mathcal{O}(nd)$
  - Computing the products: $\mathcal{O}(n)$

Sommaire

Summary

Method	Memory	Training time	Test time
PCA	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
k-means	$\mathcal{O}(np)$	$\mathcal{O}(npk)$	$\mathcal{O}(kp)$
Ridge regression	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
Logistic regression	$\mathcal{O}(np)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
SVM, kernel methods	$\mathcal{O}(np)$	$\mathcal{O}(n^3)$	$\mathcal{O}(np)$

Summary

Method	Memory	Training time	Test time
PCA	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
k-means	$\mathcal{O}(np)$	$\mathcal{O}(npk)$	$\mathcal{O}(kp)$
Ridge regression	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
Logistic regression	$\mathcal{O}(np)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
SVM, kernel methods	$\mathcal{O}(np)$	$\mathcal{O}(n^3)$	$\mathcal{O}(np)$

Things to worry about:

- Training time (can usually take place offline)

Summary

Method	Memory	Training time	Test time
PCA	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
k-means	$\mathcal{O}(np)$	$\mathcal{O}(npk)$	$\mathcal{O}(kp)$
Ridge regression	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
Logistic regression	$\mathcal{O}(np)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
SVM, kernel methods	$\mathcal{O}(np)$	$\mathcal{O}(n^3)$	$\mathcal{O}(np)$

Things to worry about:

- Training time (can usually take place offline)
- Memory requirements

Summary

Method	Memory	Training time	Test time
PCA	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
k-means	$\mathcal{O}(np)$	$\mathcal{O}(npk)$	$\mathcal{O}(kp)$
Ridge regression	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
Logistic regression	$\mathcal{O}(np)$	$\mathcal{O}(np^2)$	$\mathcal{O}(p)$
SVM, kernel methods	$\mathcal{O}(np)$	$\mathcal{O}(n^3)$	$\mathcal{O}(np)$

Things to worry about:

- Training time (can usually take place offline)
- Memory requirements
- Test time: prediction should be fast!

Techniques for large-scale ML

- Understand modern architecture, and how to distribute data / computation

Techniques for large-scale ML

- Understand modern architecture, and how to distribute data / computation
- Trade optimization accuracy for speed

Techniques for large-scale ML

- Understand modern architecture, and how to distribute data / computation
- Trade optimization accuracy for speed
- Use the deep learning tricks

References i

Boser, Bernhard E, Isabelle M Guyon, and Vladimir N Vapnik (1992). "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152.

Cessie, S Le and JC Van Houwelingen (1992). "Ridge estimators in logistic regression". In: *Journal of the Royal Statistical Society Series C: Applied Statistics* 41.1, pp. 191–201.

Hoerl, Arthur E and Robert W Kennard (1970). "Ridge regression: Biased estimation for nonorthogonal problems". In: *Technometrics* 12.1, pp. 55–67.