Large Scale Machine Learning

Introduction to large-scale ML & optimization

March 3, 2025

Katia Antonenko

ekaterina.antonenko@minesparis.psl.eu

Mines Paris – PSL

Centre for Computational Biology CBIO

Acknowledgements

Slides inspired by:

- Adeline Fermanian
- Chloé-Agathe Azencott

Contents

TECH

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

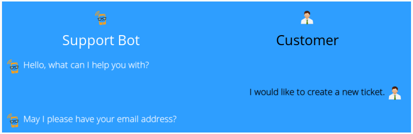
GAURAV SANGWANI | 0 | JAN 13, 2017, 12.51 PM

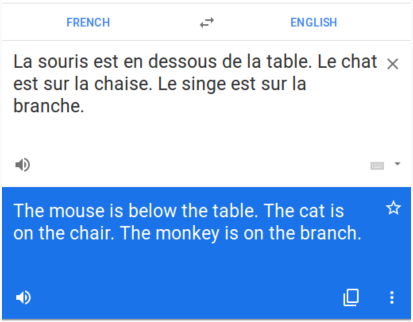
Facebook Linkedin Twitter Google+ Reddit

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

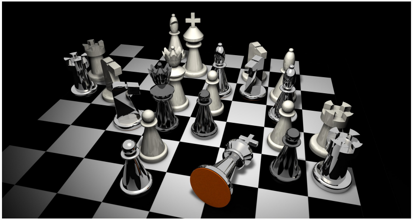
Communication

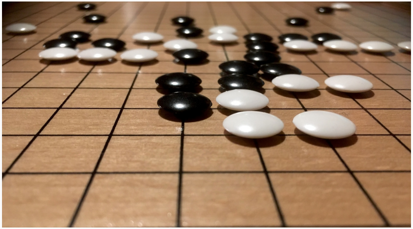




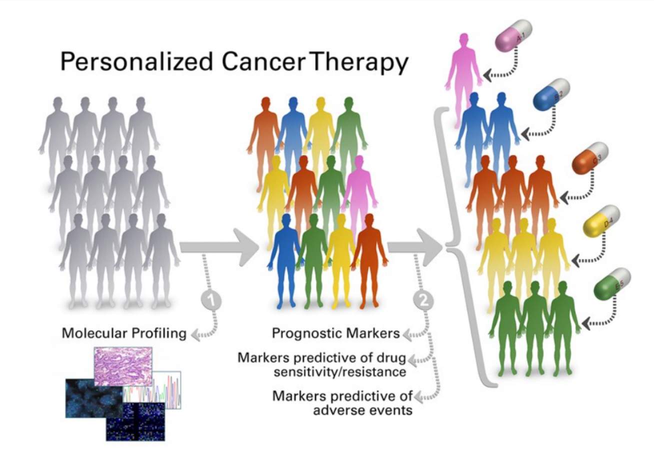
Mobility

Reasoning



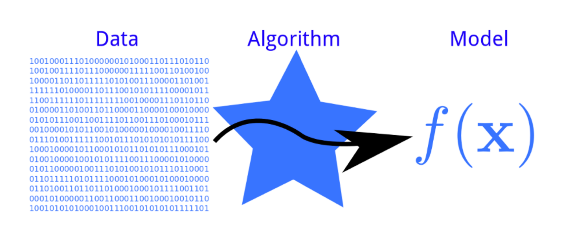


Health



https://pct.mdanderson.org

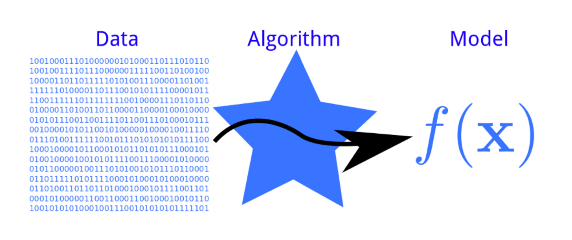
Learning from data



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

- Given: examples (training data)
  Goal: predict on new samples, or discover patterns in data

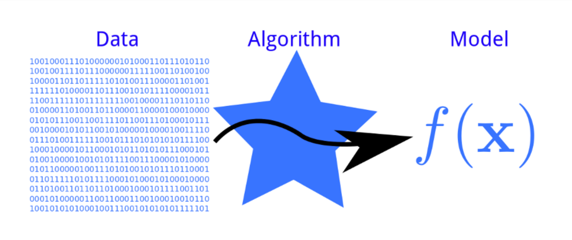
Learning from data



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

- Given: examples (training data)
  Goal: predict on new samples, or discover patterns in data
- Statistics + optimization + computer science

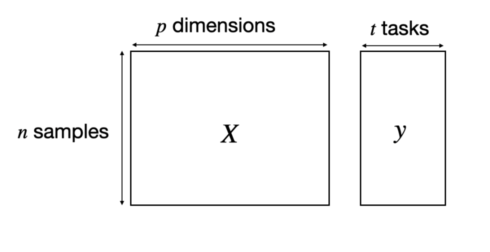
Learning from data



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

- Given: examples (training data)
  Goal: predict on new samples, or discover patterns in data
- Statistics + optimization + computer science
- Training gets better with more training examples and more powerful computers

Large-Scale Machine Learning



Dataset examples:

- 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... $n = \mathcal{O}(10^9)$, $p = \mathcal{O}(10^9)$, $t = \mathcal{O}(10^9)$

Objectives for this lecture

1. Review several standard ML methods

Objectives for this lecture

1. Review several standard ML methods

2. Discuss complexity of these methods

Objectives for this lecture

1. Review several standard ML methods
2. Discuss complexity of these methods
3. Introduce some techniques to scale these methods to big/large datasets

Contents

Learning scenarios

ML develops generic methods for solving different types of problems:

- Unsupervised learning
  Goal: learning from unlabeled data, exploring structure of the data

Learning scenarios

ML develops generic methods for solving different types of problems:

- Unsupervised learning
  Goal: learning from unlabeled data, exploring structure of the data
- Supervised learning
  Goal: learning from data samples and their labels, predicting labels for new samples

Learning scenarios

ML develops generic methods for solving different types of problems:

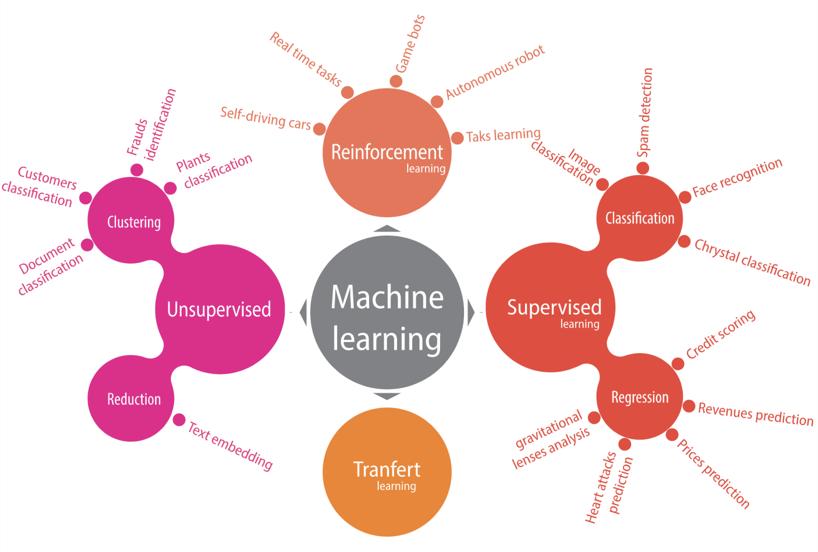
- Unsupervised learning
  Goal: learning from unlabeled data, exploring structure of the data
- Supervised learning
  Goal: learning from data samples and their labels, predicting labels for new samples
- Reinforcement learning
  Goal: learning by exploring the environment (e.g. games or autonomous vehicle)

Learning scenarios

ML develops generic methods for solving different types of problems:

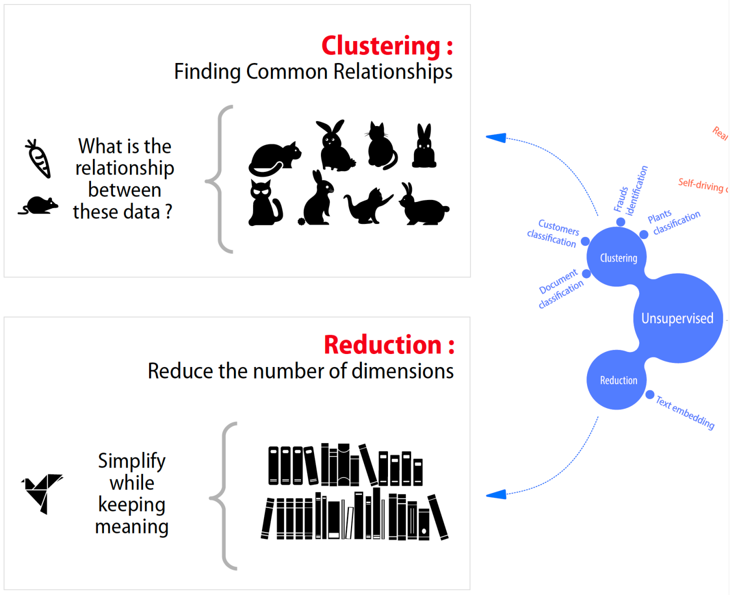
- Unsupervised learning
  Goal: learning from unlabeled data, exploring structure of the data
- Supervised learning
  Goal: learning from data samples and their labels, predicting labels for new samples
- Reinforcement learning
  Goal: learning by exploring the environment (e.g. games or autonomous vehicle)
- Transfer learning
  Goal: applying trained model to data of another type

Learning scenarios



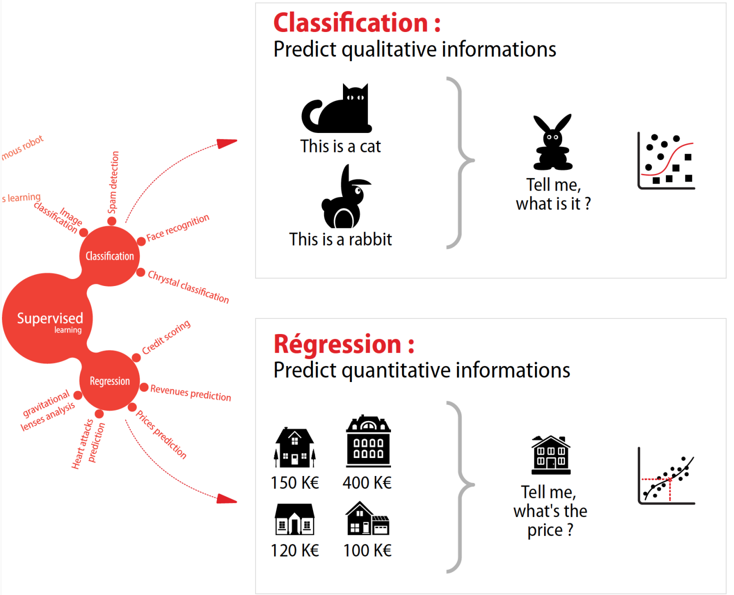
source: fidle-cnrs

Unsupervised learning



source: fidle-cnrs

Supervised learning



source: fidle-cnrs

Main ML paradigms

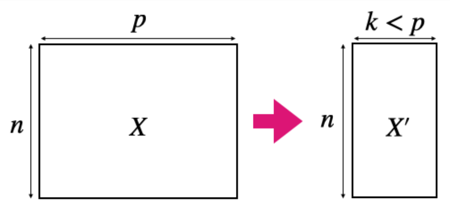
- Unsupervised learning
  - Dimension reduction
  - Clustering
  - Density estimation
  - Feature learning
- Supervised learning
  - Regression
  - Classification

Main ML paradigms

- Unsupervised learning
  - Dimension reduction: PCA
  - Clustering: k-means
  - Density estimation
  - Feature learning
- Supervised learning
  - Regression: linear (OLS), linear ridge regression
  - Classification: logistic regression, SVM

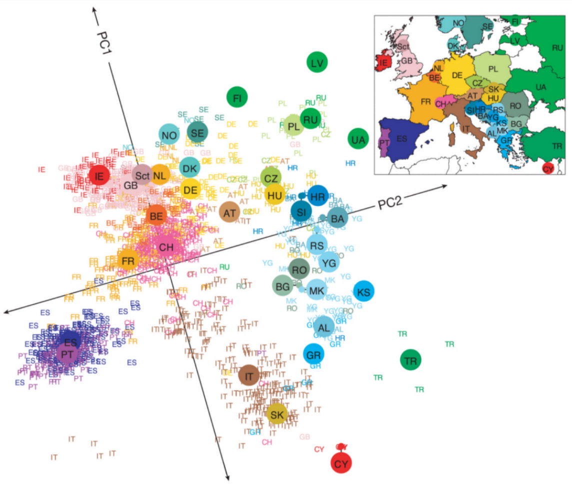
Contents

PCA: motivation



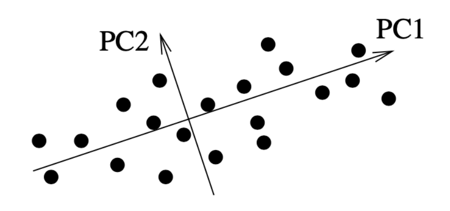
- Reduce the dimension without losing the variability in the data
- Visualization ($k = 2, 3$)
- Discover structure

Genetic data of 1387 Europeans: PCA ($k = 2$)



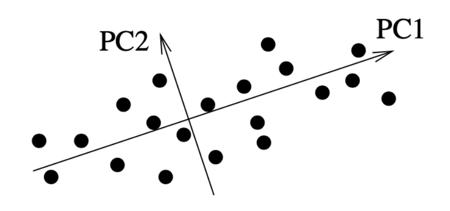
source: Novembre et al, 2008

PCA: definition



- The k-th principal component w_k:

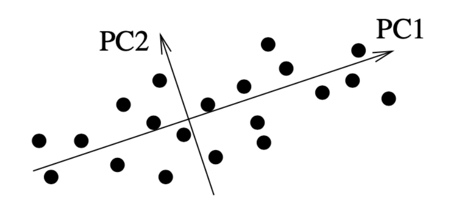
PCA: definition



- The k-th principal component w_k:
  - 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 k-th principal component w_k:
  - is orthogonal to all previous components

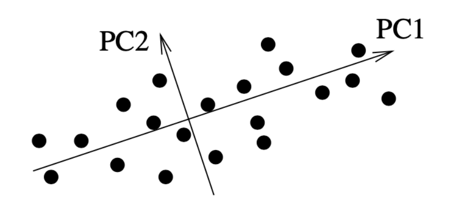
$$\langle w_k, w_1 \rangle = \langle w_k, w_2 \rangle = \cdots = \langle w_k, w_{k-1} \rangle = 0$$

  - 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 k-th principal component w_k:
  - 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 k-th eigenvector of $X^\top X$

PCA: complexity

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

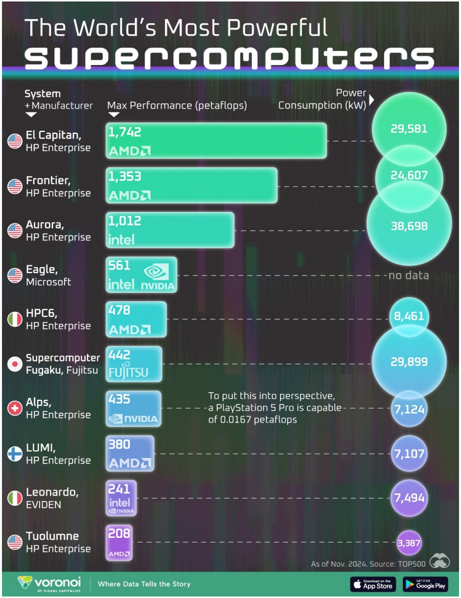
  Computing the covariance matrix is more expensive than computing its eigenvectors ($n > k$)!

Example

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

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

The most powerful computers



PCA: complexity

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

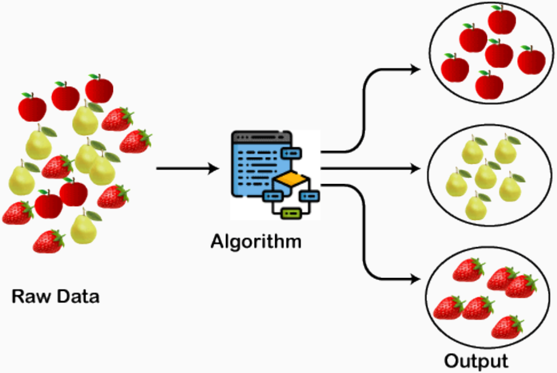
$n = 10^9$, $p = 10^8$

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

  World's fastest computer (Nov 2024):
  1,742 petaFLOPS (*Floating Point Operations per Second*) $\sim$ 1.7×10^{18} FLOPS
  $\rightarrow$ **68 days!**

Contents

Clustering: motivation



- Unsupervised learning
- Group samples
- Reduce dimensionality

k-means algorithm

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

k-means algorithm

- 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

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

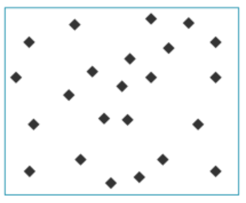
k-means algorithm

- 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

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



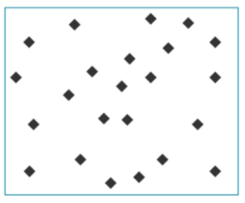
k-means algorithm

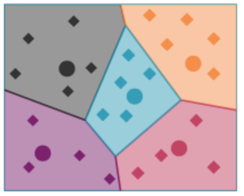
- Dataset $\{\boldsymbol{x^1}, \ldots, \boldsymbol{x^n}\} \subset \mathbb{R}^p$
- Find a cluster assignment $c_i \in \{1, \ldots, k\}$ for all $i = 1, \ldots, n$ that minimizes the intra-cluster variance:

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

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

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





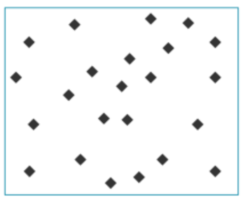
k-means algorithm

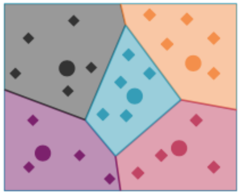
- Dataset $\{\boldsymbol{x^1}, \dots, \boldsymbol{x^n}\} \subset \mathbb{R}^p$
- Find a cluster assignment $c_i \in \{1, \dots, k\}$ for all $i = 1, \dots, n$ that minimizes the intra-cluster variance:

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

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*

Lloyd's algorithm (naïve k-means)

- Initialization:
  Randomly select k centroids $\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_k$
- Iterations:
  1. Assignment step: assign the points to their nearest centroids

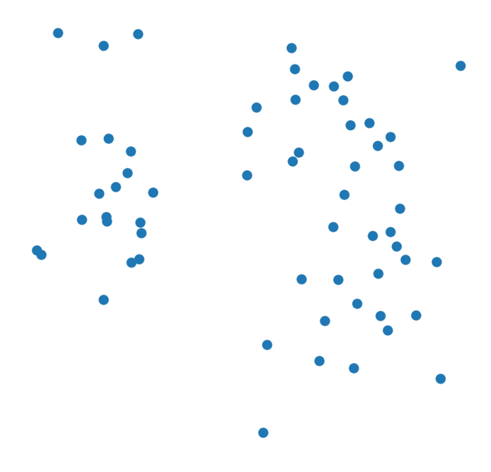
$$\forall i = 1, \ldots, n, \quad c_i \leftarrow \mathrm{argmin}_{c \in \{1,\ldots,k\}} \|\boldsymbol{x}^{\boldsymbol{i}} - \boldsymbol{\mu}_c\|$$

Lloyd's algorithm (naïve k-means)

- Initialization:
  Randomly select k centroids $\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_k$
- Iterations:
  1. Assignment step: assign the points to their nearest centroids
  $$\forall i = 1, \ldots, n, \quad c_i \leftarrow \mathsf{argmin}_{c \in \{1,\ldots,k\}} \|\boldsymbol{x}^{\boldsymbol{i}} - \boldsymbol{\mu}_c\|$$
  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

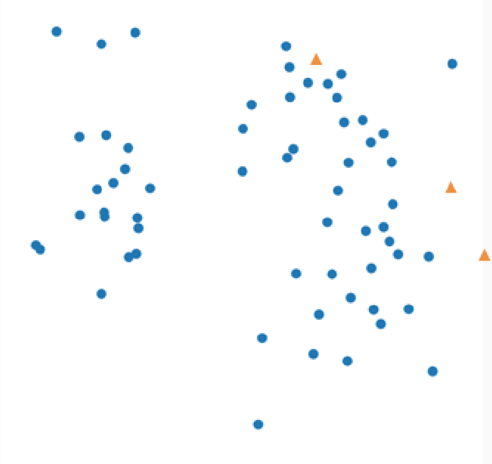
$k = 3$



k-means example

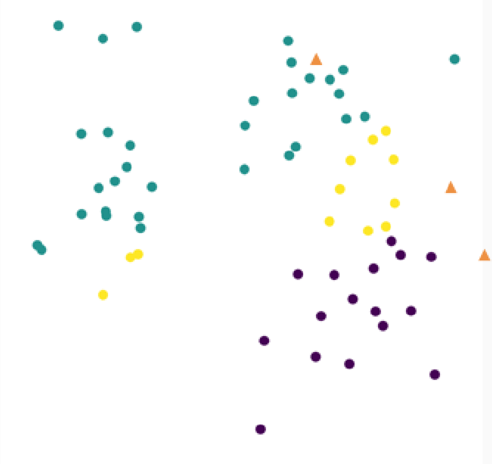
- Select 3 centroids at random

$k = 3$



k-means example

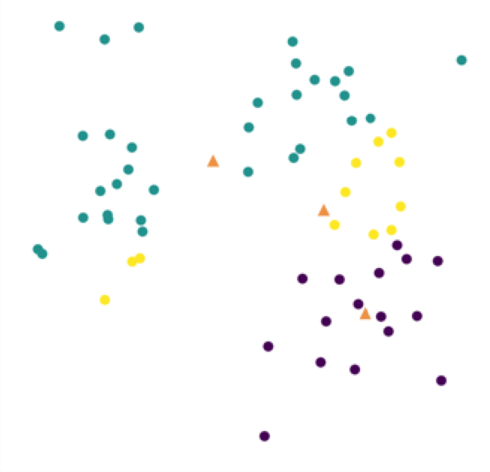
- Assign each observation to the nearest centroid $k = 3$



k-means example

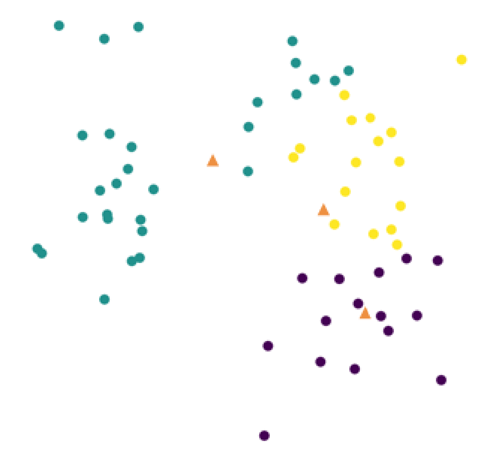
- Recompute centroids

$k = 3$



k-means example

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



k-means example

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

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**:

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

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

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

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: $\mathcal{O}(knp)$

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**: $\mathcal{O}(kTnp)$

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:

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**:

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

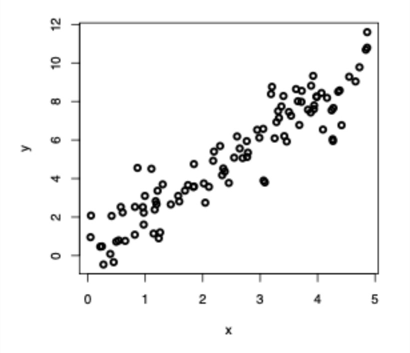
    Compute $n \times k$ distances in $\mathbb{R}^p$: $\mathcal{O}(knp)$
  - **Update step**:

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

    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: $\mathcal{O}(n + kp)$
  - Store X: $\mathcal{O}(np)$

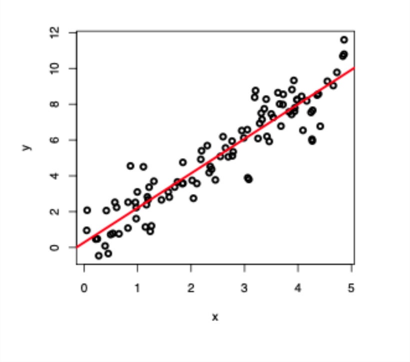
Contents

Linear regression: motivation



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

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

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

- Quality of fit is measured as a Residual Sum of Squares (RSS):

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

Linear regression

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

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

- Quality of fit is measured as a Residual Sum of Squares (RSS):

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

- Solution:

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

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

Ridge regression

- *Hoerl and Kennard, Ridge regression: Biased estimation for nonorthogonal problems, 1970*

Ridge regression

- *Hoerl and Kennard, Ridge regression: Biased estimation for nonorthogonal problems, 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, Ridge regression: Biased estimation for nonorthogonal problems, 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

Ridge regression

- *Hoerl and Kennard, Ridge regression: Biased estimation for nonorthogonal problems, 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:

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

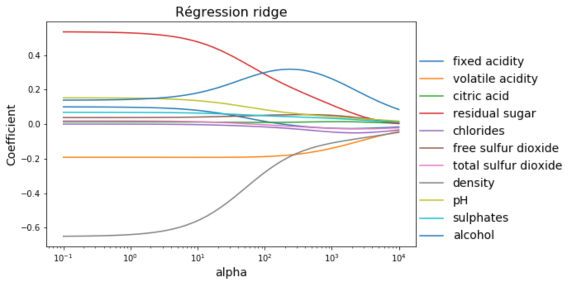
  $\rightarrow$ *unique and always exists !*
- Correlated features get similar weights
- Regularization reduces overfitting by penalizing larger weights, encouraging the model to prioritize simpler hypotheses

Ridge regression: limit cases

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

Corollary

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



Ridge regression: complexity

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

Ridge regression: complexity

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

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

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 $X^\top X + \lambda I$ ($p \times p$ matrix): $\mathcal{O}(p^3)$

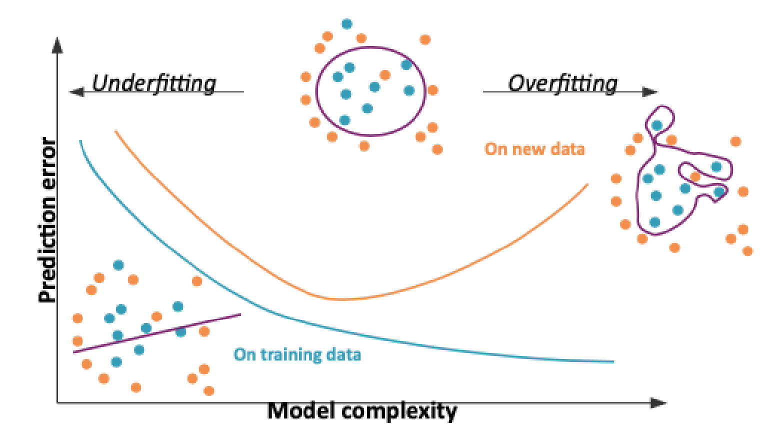
Ridge regression: complexity

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

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

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

Ridge regression: choice of λ

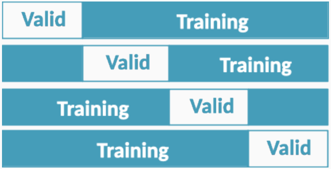


Ridge regression: choice of λ

- Data splitting strategies: cross-validation

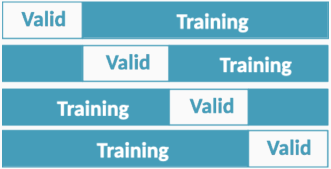
Ridge regression: choice of λ

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



Ridge regression: choice of λ

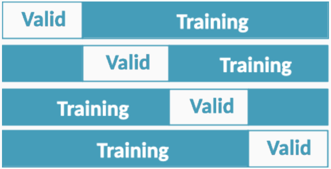
- Data splitting strategies: cross-validation
  - Split the training set (of size n) into K equally-sized chunks



  - K folds: train on $K-1$ folds, test on the left out one

Ridge regression: choice of λ

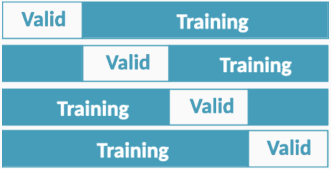
- Data splitting strategies: cross-validation
  - Split the training set (of size n) into K equally-sized chunks



  - K folds: train on $K-1$ folds, test on the left out one
  - Cross-validation score: average performance over the K folds

Ridge regression: choice of λ

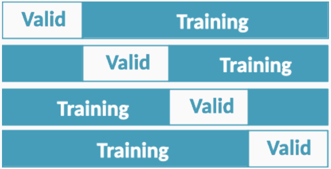
- Data splitting strategies: cross-validation
  - Split the training set (of size n) into K equally-sized chunks



  - K folds: train on $K-1$ folds, test on the left out one
  - Cross-validation score: average performance over the K folds
- For selection of λ: take a grid of values $(\lambda_1, \ldots, \lambda_M)$ and choose the λ with the best cross-validation score

Ridge regression: choice of λ

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



  - K folds: train on $K-1$ folds, test on the left out one
  - Cross-validation score: average performance over the K folds
- For selection of λ: take a grid of values $(\lambda_1, \ldots, \lambda_M)$ and choose the λ with the best cross-validation score
- Multiplies complexity by KM!

Linear regression generalization: ℓ_2-regularized learning

- Generalization of the ridge regression to any loss:

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

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

Linear regression 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

Linear regression generalization: ℓ_2-regularized learning

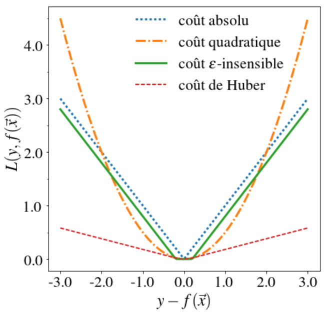
- Generalization of the ridge regression to any loss:

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

- 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

Loss examples

- 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

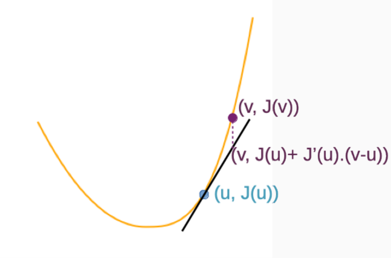


Numerical solution: gradient descent

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

- If we assume that the loss is differentiable, then

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



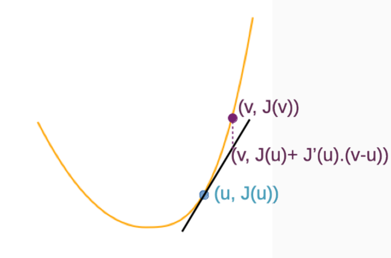
Numerical solution: gradient descent

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

- If we assume that the loss is differentiable, then

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

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



Numerical solution: gradient descent

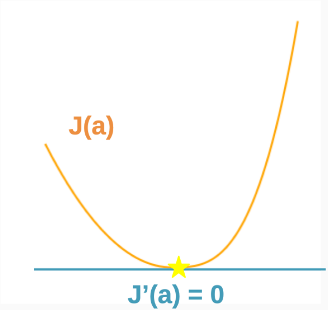
Idea: to minimize a differentiable, strictly convex function J, we find where its gradient is equal to 0

Numerical solution: gradient descent

Idea: to minimize a differentiable, strictly convex function J, we find where its gradient is equal to 0

Algorithm:

- Pick a_0 randomly

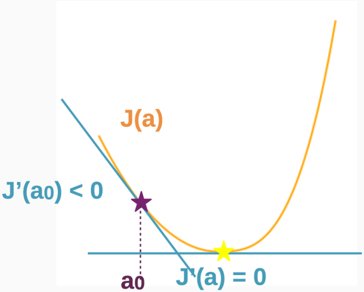


Numerical solution: gradient descent

Idea: to minimize a differentiable, strictly convex function J, we find where its gradient is equal to 0

Algorithm:

- Pick a_0 randomly
- Keep updating
  $a_t = a_{t-1} - \alpha \nabla J(a_{t-1})$

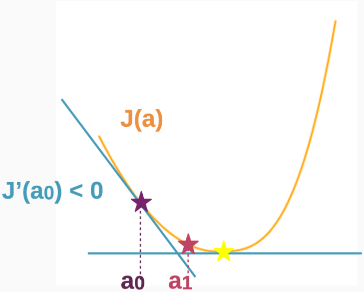


Numerical solution: gradient descent

Idea: to minimize a differentiable, strictly convex function J, we find where its gradient is equal to 0

Algorithm:

- Pick a_0 randomly
- Keep updating $a_t = a_{t-1} - \alpha \nabla J(a_{t-1})$
- Stop when $|\nabla J(a_t))| < \varepsilon$

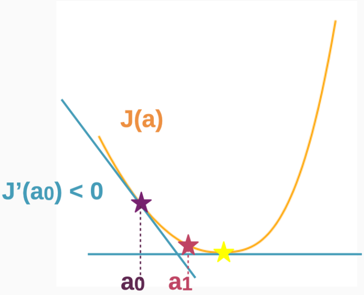


Numerical solution: gradient descent

Idea: to minimize a differentiable, strictly convex function J, we find where its gradient is equal to 0

Algorithm:

- Pick a_0 randomly
- Keep updating
  $a_t = a_{t-1} - \alpha \nabla J(a_{t-1})$
- Stop when $|\nabla J(a_t))| < \varepsilon$



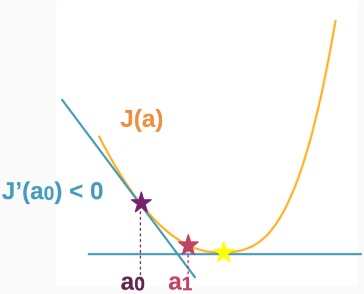
α – learning rate

Numerical solution: gradient descent

Idea: to minimize a differentiable, strictly convex function J, we find where its gradient is equal to 0

Algorithm:

- Pick a_0 randomly
- Keep updating
  $a_t = a_{t-1} - \alpha \nabla J(a_{t-1})$
- Stop when $|\nabla J(a_t))| < \varepsilon$



α – learning rate

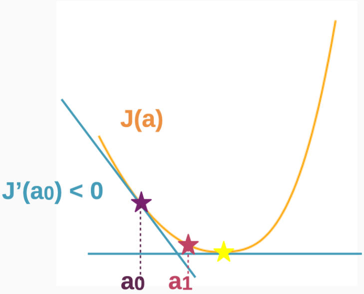
- if α is too small $\rightarrow$ the algorithm is very slow

Numerical solution: gradient descent

Idea: to minimize a differentiable, strictly convex function J, we find where its gradient is equal to 0

Algorithm:

- Pick a_0 randomly
- Keep updating
  $a_t = a_{t-1} - \alpha \nabla J(a_{t-1})$
- Stop when $|\nabla J(a_t))| < \varepsilon$



α – learning rate

- if α is too small $\rightarrow$ the algorithm is very slow
- if α is too big $\rightarrow$ a might oscillate around the minimum

Numerical solution: gradient descent

When to use gradient descent:

- Computing the analytical solution is too time-intensive (e.g., Ridge regression)
- Analytical solution for $\nabla J(a) = 0$ does not exist

Numerical solution: gradient descent

When to use gradient descent:

- Computing the analytical solution is too time-intensive (e.g., Ridge regression)
- Analytical solution for $\nabla J(a) = 0$ does not exist

NB!

If J is not convex, we are not guaranteed to find a <u>global</u> minimum

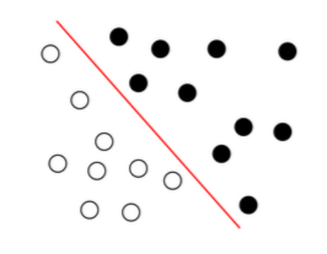
(we may need multiple restarts)

Contents

Classification: motivation

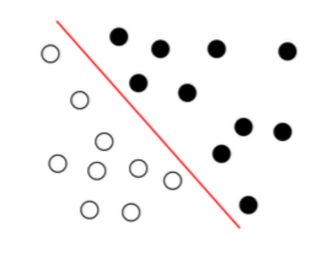
- Predict data labels (categories)
- There can be 2 or more (sometimes many) labels

Classification: linear models



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

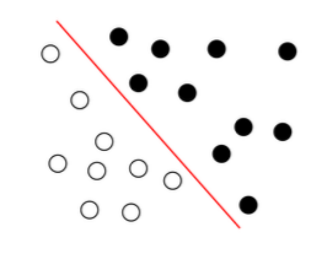
Classification: linear models



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

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

Classification: linear models



- Training set $\mathcal{S} = \{(\boldsymbol{x^1}, y^1), \dots, (\boldsymbol{x^n}, y^n)\} \subset \mathbb{R}^p \times \{-1, 1\}$
- Fit a linear function
$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \boldsymbol{\beta}^\top \boldsymbol{x}$$
- Prediction for a new sample $\boldsymbol{x} \in \mathbb{R}^p$:
$$\begin{cases} +1 & \text{if } f_{\boldsymbol{\beta}}(\boldsymbol{x}) > 0\,, \\ -1 & \text{otherwise.} \end{cases}$$

Classification: the $0/1$ loss

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

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

Classification: the $0/1$ loss

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

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

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

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

Classification: the $0/1$ loss

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

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

- Model is then 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

Classification: the $0/1$ loss

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

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

- Model is then 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 to scaling of $\boldsymbol{\beta}$

Classification: logistic loss

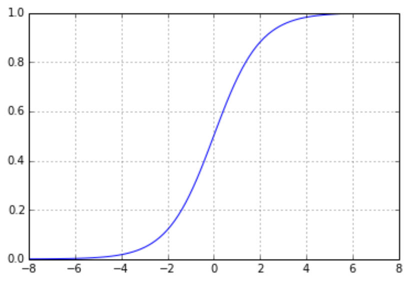
- Alternative approach:
  to define a probabilistic model of y parametrized by $f(\boldsymbol{x})$, e.g.:

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

Classification: logistic loss

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

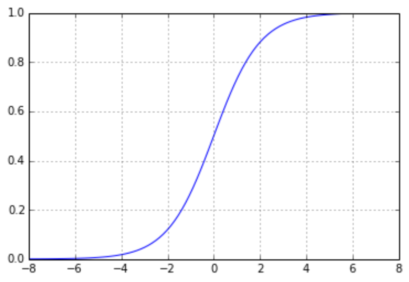
$$\forall y \in \{-1, 1\}, \quad \mathbb{P}(y \,|\, f(\boldsymbol{x})) = \frac{1}{1 + e^{-y f(\boldsymbol{x})}} = \sigma(y f(\boldsymbol{x}))$$



Classification: logistic loss

- Alternative approach:
  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:

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

Classification: Ridge logistic regression

- *Cessie and Van Houwelingen, Ridge estimators in logistic regression, 1992*

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

Classification: Ridge logistic regression

- *Cessie and Van Houwelingen, Ridge estimators in logistic regression, 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

Classification: Ridge logistic regression

- *Cessie and Van Houwelingen, Ridge estimators in logistic regression, 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 analytical solution, but smooth convex optimization problem that can be solved numerically

Newton-Raphson iterations

- Numerical method to minimize convex and differentiable J

Newton-Raphson iterations

- Numerical method to minimize convex and differentiable J
- Gradient descent:

$$u_t = u_{t-1} - \alpha \nabla J(u_{t-1})$$

where α is not constant

Newton-Raphson iterations

- Numerical method to minimize convex and differentiable J
- Gradient descent:

$$u_t = u_{t-1} - \alpha \nabla J(u_{t-1})$$

  where α is not constant
- 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 iterations

- Numerical method to minimize convex and differentiable J
- Gradient descent:

$$u_t = u_{t-1} - \alpha \nabla J(u_{t-1})$$

  where α is not constant
- 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 iterations

- Numerical method to minimize convex and differentiable J
- Gradient descent:

$$u_t = u_{t-1} - \alpha \nabla J(u_{t-1})$$

  where α is not constant
- 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 iterations

- Numerical method to minimize convex and differentiable J
- Gradient descent:

$$u_t = u_{t-1} - \alpha \nabla J(u_{t-1})$$

  where α is not constant

- 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 $\alpha = (\nabla^2 J(u_{t-1}))^{-1}$ in the gradient step

Solving ridge logistic regression

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

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

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

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

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

Solving ridge logistic regression

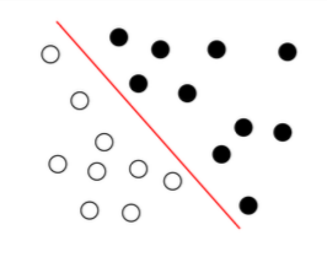
$$\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + e^{-y^i \boldsymbol{\beta}^\top \boldsymbol{x^i}}) + \lambda \|\boldsymbol{\beta}\|_2^2$$

- Solve with Newton-Raphson iterations

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

- Time complexity $\mathcal{O}(T(np^2 + p^3))$

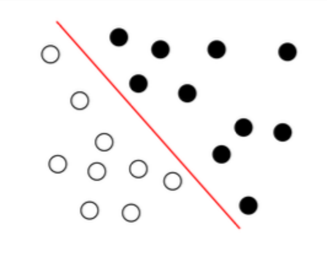
Large-margin classifiers



- 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

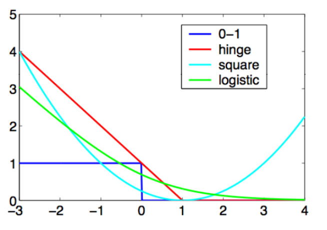
$$y f(\boldsymbol{x})$$

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

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

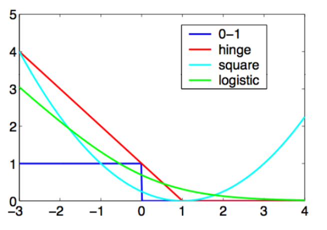
for a convex, non-increasing function $\phi : \mathbb{R} \to \mathbb{R}+$

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}

How to choose ϕ?



- Computation
  - convex $\phi \implies$ need to solve a convex optimization problem
  - Good choice of ϕ may allow fast optimization
- Theory
  - Most ϕ lead to consistent estimators
  - Some may be more efficient than others

Linear SVM

- *Boser, Guyon, and Vapnik, A training algorithm for optimal margin classifiers, 1992*

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

Linear SVM

- *Boser, Guyon, and Vapnik, A training algorithm for optimal margin classifiers, 1992*

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

- Primal problem is <u>convex, but non-smooth</u>

Linear SVM

- *Boser, Guyon, and Vapnik, A training algorithm for optimal margin classifiers, 1992*

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

- Primal problem is convex, but non-smooth
- Equivalent to Dual problem

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

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

Linear SVM

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

Linear SVM

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

Complexity (training)

Linear SVM

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

Complexity (training)

- Primal: $\mathcal{O}(np^2 + p^3)$

Linear SVM

- 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)

- Primal: $\mathcal{O}(np^2 + p^3)$
- Dual: $\mathcal{O}(n^3 + pn^2)$

Linear SVM

- 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)

- Primal: $\mathcal{O}(np^2 + p^3)$
- Dual: $\mathcal{O}(n^3 + pn^2)$

Complexity (predicting)

Linear SVM

- 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)

- Primal: $\mathcal{O}(np^2 + p^3)$
- Dual: $\mathcal{O}(n^3 + pn^2)$

Complexity (predicting)

- Primal: $\mathcal{O}(p)$

Linear SVM

- 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)

- Primal: $\mathcal{O}(np^2 + p^3)$
- Dual: $\mathcal{O}(n^3 + pn^2)$

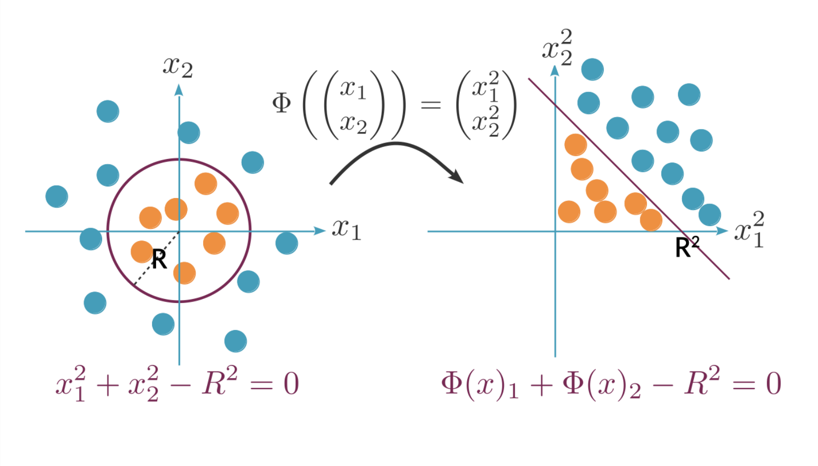
Complexity (predicting)

- Primal: $\mathcal{O}(p)$
- Dual: $\mathcal{O}(np)$

Contents

Kernel methods: motivation

Kernel methods: non-linear mapping



Kernel SVM

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

- **We have to use the dual form:**

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

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

- **Kernel k:**

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

Kernels

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

Kernels

- 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

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

Kernels

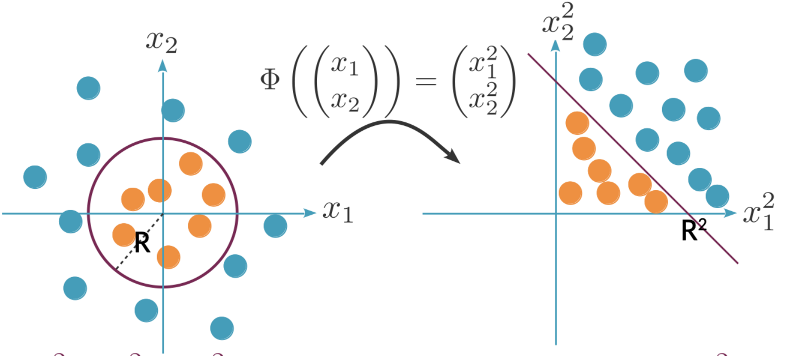
- k may be quite efficient to compute, even if $\mathcal{H}$ is a very high-dimensional or even infinite-dimensional space
- For any positive semi-definite function k, there exists a feature space $\mathcal{H}$ and a feature map ϕ such that

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

- **Kernels:**

$$\begin{aligned}
\text{Linear} \quad & k(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{x}^\top \boldsymbol{x'} \\
\text{Polynomial} \quad & k(\boldsymbol{x}, \boldsymbol{x'}) = (\boldsymbol{x}^\top \boldsymbol{x'} + c)^d \\
\text{Gaussian} \quad & k(\boldsymbol{x}, \boldsymbol{x'}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x'}\|^2}{2\sigma^2}\right) \\
\text{Min/max} \quad & k(\boldsymbol{x}, \boldsymbol{x'}) = \sum_{j=1}^{p} \frac{\min(|x_j|, |x'_j|)}{\max(|x_j|, |x'_j|)}
\end{aligned}$$

Non-linear mapping to a feature space



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

$$K(\boldsymbol{x}, \boldsymbol{x}') = \left\langle \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix}, \begin{pmatrix} {x_1'}^2 \\ {x_2'}^2 \end{pmatrix} \right\rangle = x_1^2 {x_1'}^2 + x_2^2 {x_2'}^2$$ [1]

Contents

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	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2 + p^3)$	$\mathcal{O}(p)$
Logistic regression	$\mathcal{O}(np)$	$\mathcal{O}(np^2 + p^3)$	$\mathcal{O}(p)$
SVM, kernel methods	$\mathcal{O}(np)$	$\mathcal{O}(n^3 + pn^2)$	$\mathcal{O}(np)$

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	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2 + p^3)$	$\mathcal{O}(p)$
Logistic regression	$\mathcal{O}(np)$	$\mathcal{O}(np^2 + p^3)$	$\mathcal{O}(p)$
SVM, kernel methods	$\mathcal{O}(np)$	$\mathcal{O}(n^3 + pn^2)$	$\mathcal{O}(np)$

Things to worry about:

- 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	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2 + p^3)$	$\mathcal{O}(p)$
Logistic regression	$\mathcal{O}(np)$	$\mathcal{O}(np^2 + p^3)$	$\mathcal{O}(p)$
SVM, kernel methods	$\mathcal{O}(np)$	$\mathcal{O}(n^3 + pn^2)$	$\mathcal{O}(np)$

Things to worry about:

- Memory requirements
- Training time: usually can 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	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2 + p^3)$	$\mathcal{O}(p)$
Logistic regression	$\mathcal{O}(np)$	$\mathcal{O}(np^2 + p^3)$	$\mathcal{O}(p)$
SVM, kernel methods	$\mathcal{O}(np)$	$\mathcal{O}(n^3 + pn^2)$	$\mathcal{O}(np)$

Things to worry about:

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

Techniques for large-scale ML

- Take use of modern architecture:
  how to distribute data and computation

Techniques for large-scale ML

- Take use of modern architecture:
  how to distribute data and computation

- Trade optimization accuracy for speed:
  numerical solutions

Techniques for large-scale ML

- Take use of modern architecture:
  how to distribute data and computation

- Trade optimization accuracy for speed:
  numerical solutions

- Use the deep learning tricks