

Machine Learning

Laurea Magistrale Ing. Informatica

Fabio Vandin

October 10th, 2019

Labs and Homeworks

Not compulsory but HIGHLY RECOMMENDED

1. **Intro to Python: Thursday October 17th (Rooms Te-Ue) [compulsory], during lectures time (2:30-4:30pm)**
2. **Linear models for regression and classification**
 - ?: first homework will be released
 - ?: lab assistance for those interested (Room Te-Ue)
 - ?: first homework due.
3. **Regularization and SVMs**
 - ?: second homework will be released
 - ?: lab assistance for those interested (Room Te-Ue)
 - ?: second homework due.
4. **Neural Networks and unsupervised learning**
 - ?: third homework will be released
 - ?: lab assistance for those interested (Room Te-Ue)
 - ?: third homework due.

Example: Movie Rating Prediction

Prediction: What will be the user rating on a movie?

Challenge: Must target current user preferences

Noisy training data: ratings of all users on movies

Challenge: Data is very sparse = each user rates very few movies!



Example: Simplified

Movie has 2 features:

- fraction of the movie with action
- fraction of the movie with romance

Rating is only LIKED or NOT LIKED

Prediction: will the user like a movie?

The essence of learning:

- A pattern exists
- We do not know it, i.e. we cannot pin it down mathematically or with very simple rules
- We have data to try to “learn” it



A Formal Model (Statistical Learning)

We have a *learner* (us, or the machine) has access to:

- ① **Domain set \mathcal{X}** : set of all possible objects to make predictions about
 - domain point $x \in \mathcal{X} = \text{instance}$, usually represented by a vector of *features*
 - \mathcal{X} is the *instance space*
- ② **Label set \mathcal{Y}** : set of possible labels.
 - often two labels, e.g $\{-1, +1\}$ or $\{0, 1\}$
- ③ **Training data $S = ((x_1, y_1), \dots, (x_m, y_m))$** : finite sequence of labeled domain points, i.e. pairs in $\mathcal{X} \times \mathcal{Y}$
 - this is the learner's **input**
 - S : *training example* or *training set*

A Formal Model

- ④ **Learner's output h :** prediction rule $h : \mathcal{X} \rightarrow \mathcal{Y}$
 - also called *predictor*, *hypothesis*, or *classifier*
 - $A(S)$: prediction rule produced by learning algorithm A when training set S is given to it
 - sometimes \hat{f} used instead of h
- ⑤ **Data-generation model:** instances are generated by some probability distribution and labeled according to a function
 - \mathcal{D} : probability distribution over \mathcal{X} (**NOT KNOWN TO THE LEARNER!**)
 - labeling function $f: \mathcal{X} \rightarrow \mathcal{Y}$ (**NOT KNOWN TO THE LEARNER!**)
 - label y_i of instance x_i : $y_i = f(x_i)$, for all $i = 1, \dots, m$
 - each point in training set S : first sample x_i according to \mathcal{D} , then label it as $y_i = f(x_i)$
- ⑥ **Measures of success:** *error of a classifier* = probability it does not predict the correct label on a random data point generate by distribution \mathcal{D}

LOSS

Given domain subset $A \subset \mathcal{X}$, $\mathcal{D}(A)$ = probability of observing a point $x \in A$.

In many cases, we refer to A as *event* and express it using a function $\pi : \mathcal{X} \rightarrow \{0, 1\}$, that is:

$$A = \{x \in \mathcal{X} : \pi(x) = 1\}$$

In this case we have $\mathbb{P}_{x \sim \mathcal{D}}[\pi(x)] = \mathcal{D}(A)$

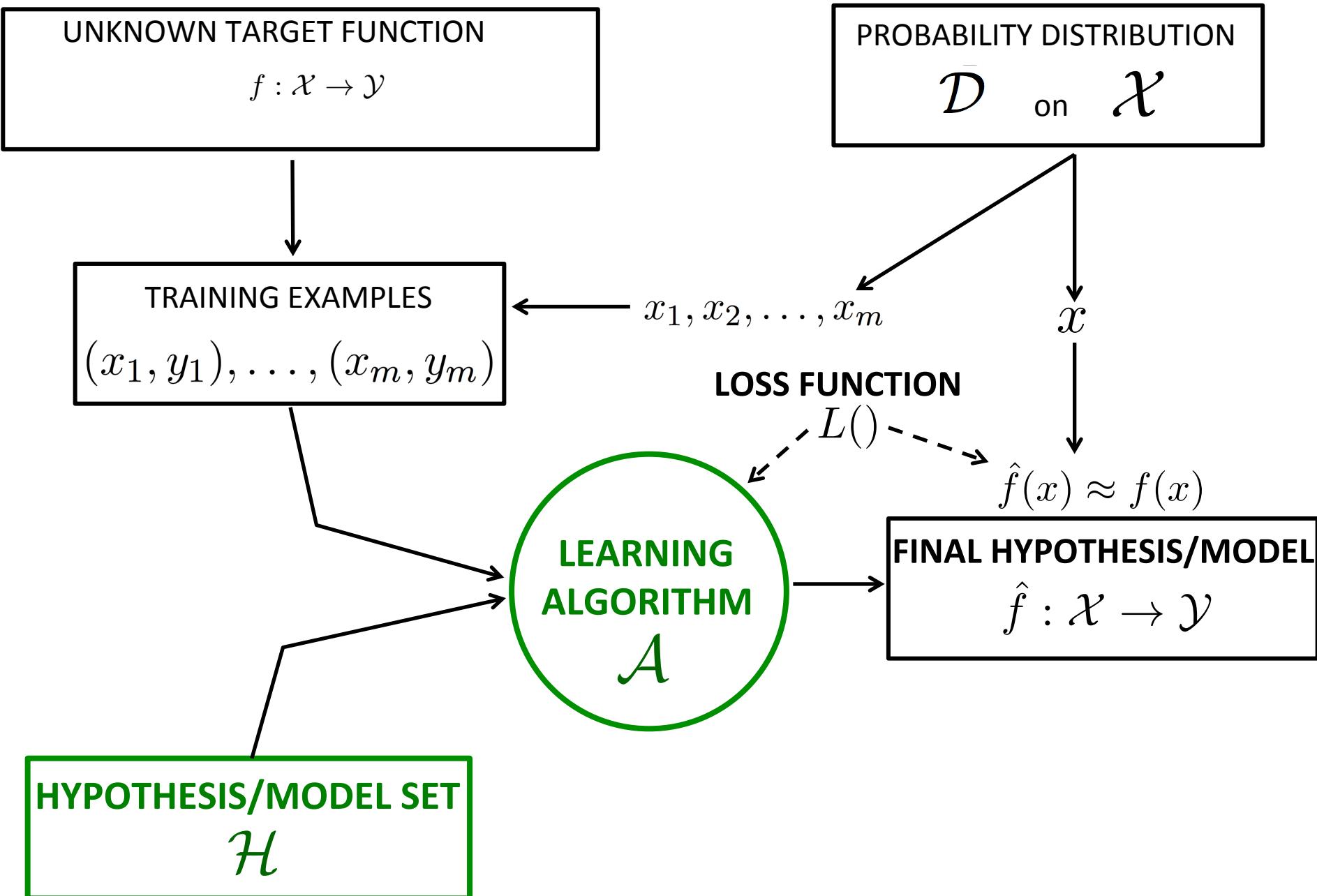
Error of prediction rule $h : \mathcal{X} \rightarrow \mathcal{Y}$ is

$$L_{\mathcal{D}, f}(h) \stackrel{\text{def}}{=} \mathbb{P}_{x \sim \mathcal{D}}[h(x) \neq f(x)] \stackrel{\text{def}}{=} \mathcal{D}(\{x : h(x) \neq f(x)\})$$

Notes:

- $L_{\mathcal{D}, f}(h)$ has many different names: **generalization error**, **true error**, **risk**, **loss**, ...
- often f is obvious, so omitted: $L_{\mathcal{D}}(h)$

Learning Process (Simplified)



Types of Learning

y_i are known: **training set** $(x_1, y_1), \dots, (x_m, y_m)$

→ **supervised learning**

y_i are **not** known: **training set** x_1, x_2, \dots, x_m

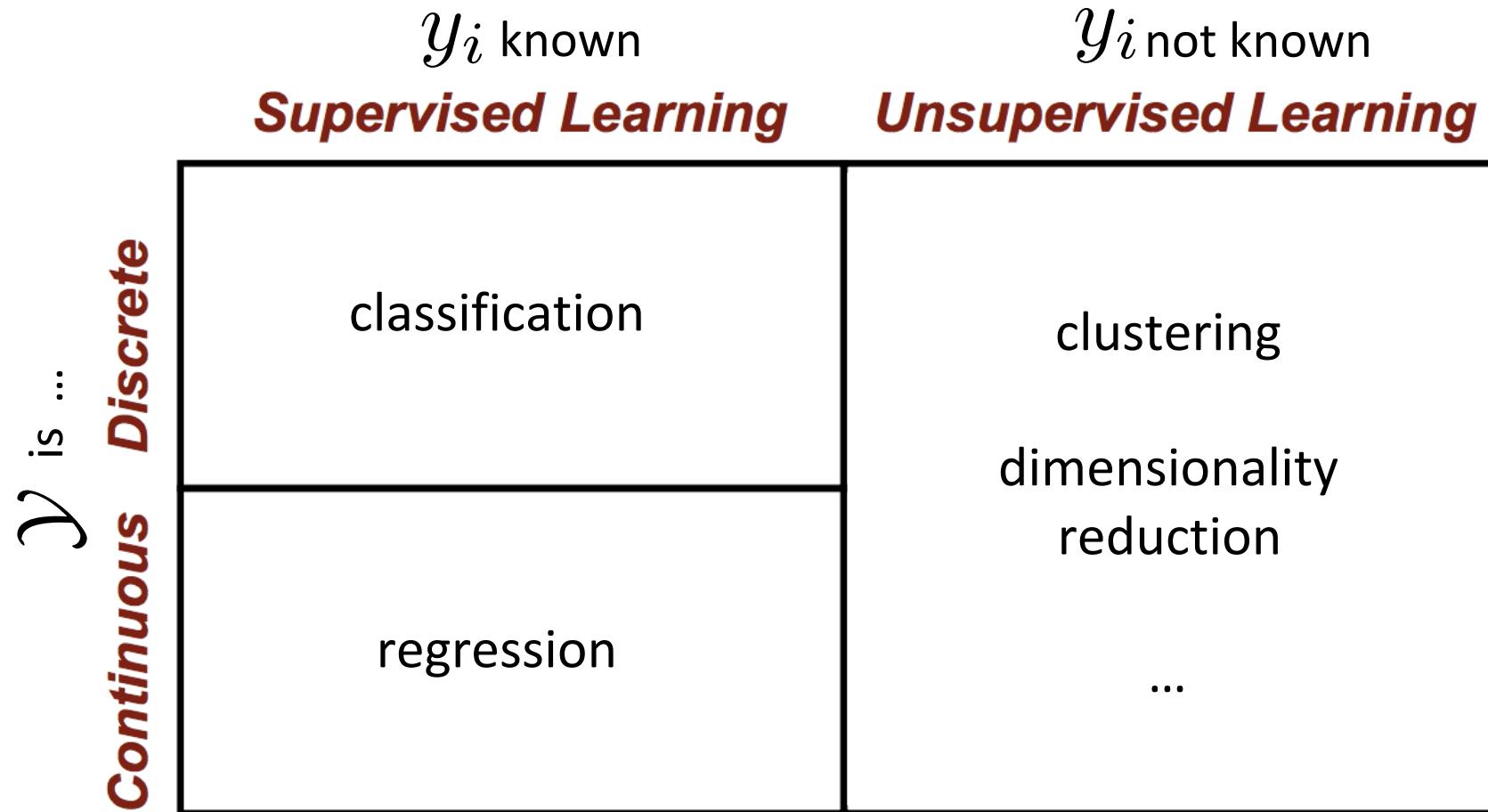
→ **unsupervised learning**

There can be different types of output:

- \mathcal{Y} is **discrete**
- \mathcal{Y} is **continuous**

Notes: we will see a more general learning model soon,
main ideas are the same!

Types of Learning



(Rough) Course Plan

PART I: Supervised Learning

Introduction: Data, Classes of models, Losses, Probabilistic models and assumptions on the data, Models, Losses

Regression and Classification

When is a model good? Model complexity, bias variance tradeoff/generalization (VC dimension generalization error)

Models for Regression: Linear Regression (scalar and multivariate), Regularization, Subset Selection

Classes of non linear models: Sigmoids, Neural Networks

Kernel Methods: SVM

(Rough) Course Plan

Models for Classification: Logistic Regression, NN, SVM

Validation and Model Selection

Model complexity determination

PART II: Unsupervised learning

Cluster analysis: K-means Clustering, Mixtures of Gaussians and the EM estimation

Dimensionality reduction: Principal Component Analysis (PCA)

Part III: Advanced Topics

Deep Learning

Statistically Significant Patterns

Graph Embeddings

...

Objectives

Provide the **fundamentals** and **basic principles** of the **learning problem**

Introduce the **most common algorithms** for **regression** and **classification**

Analytical and **practical ability** in using these tools for the solution of basic problems

Some **hands-on** experience

Course Prerequisites!

Calculus

Programming

Linear Algebra

Probability

Calculus

- derivatives
- minimization of functions
- partial derivatives of functions of multiple variables
- integrals

Programming

- You should already know Java
... learning Python will be easy!

Linear Algebra

- matrix factorization
- matrix inversion
- linear independence
- rank, column space, null space
- orthogonality, projections
- eigenvalues, eigenvectors
- symmetric positive definite matrices
- matrix differentiation

Probability

- discrete random variables (r.v.), moments, expectation
- continuous r.v.'s, probability density function (PDF), cumulative distribution function (CDF)
- joint, marginal, conditional distribution
- some famous distributions:
 - discrete: binomial
 - continuous: Gaussian
- Independence and conditional independence
- Bayes Theorem
- Law of large numbers

Usefull link: *Seeing theory (visualization for probability, statistics, etc.)*

<https://students.brown.edu/seeing-theory/basic-probability/index.html>

Machine Learning

Probability Review

Fabio Vandin

October 10th, 2019

Probability Refresher

Definition

A probability space has three components:

- ① A sample space Z , which is the set of all possible outcomes of the random process modeled by the probability space;
- ② A family \mathcal{F} of sets representing the allowable events, where each event A is a subset of Z : $A \subseteq Z$ (Must be a σ -field...)
- ③ A probability distribution $\mathcal{D} : \mathcal{F} \rightarrow [0, 1]$ that satisfies the following conditions:
 - ① $\mathcal{D}[Z] = 1$;
 - ② Let E_1, E_2, E_3, \dots be any finite or countably infinite sequence of pairwise mutually disjoint events ($E_i \cap E_j = \emptyset$ for all $i \neq j$):

$$\mathcal{D} \left[\bigcup_{i \geq 1} E_i \right] = \sum_{i \geq 1} \mathcal{D}[E_i].$$

Distributions and Probability

We use $z \sim \mathcal{D}$ to say that $z \in Z$ is *sampled* according to \mathcal{D}

Given a function $f : Z \rightarrow \{\text{true}, \text{false}\}$, define the probability of $f(z)$

$$\mathbb{P}_{z \sim \mathcal{D}}[f(z)] = \mathcal{D}(\{z \in Z : f(z) = \text{true}\})$$

In many cases, we express an event $A \subseteq Z$ using a function $\pi : Z \rightarrow \{\text{true}, \text{false}\}$, that is:

$$A = \{z \in Z : \pi(z) = \text{true}\}$$

where $\pi(z) = \text{true}$ if $z \in A$ and $\pi(z) = 0$ otherwise.

In this case we have $\mathbb{P}_{z \sim \mathcal{D}}[\pi(z)] = \mathcal{D}(A)$

Notes:

- for an event A we may use $\mathbb{P}[A]$ instead of $\mathbb{P}[\pi(z)]$ for the corresponding π
- sometimes we use $\pi : Z \rightarrow \{0, 1\}$ instead of $\pi : Z \rightarrow \{\text{true}, \text{false}\}$

Independent Events

Definition

Two events E and F are independent ($E \perp F$) if and only if

$$\mathbb{P}[E \cap F] = \mathbb{P}[E] \cdot \mathbb{P}[F]$$

More generally, events E_1, E_2, \dots, E_k are mutually independent if and only if for any subset $I \subseteq [1, k]$,

$$\mathbb{P}\left[\bigcap_{i \in I} E_i\right] = \prod_{i \in I} \mathbb{P}[E_i].$$

Random Variable (R.V.)

Definition

A (scalar) random variable $X(z)$ on a sample space Z is a real-valued function on Z ; that is, $X : z \in Z \rightarrow \mathbb{R}$.

Discrete random variable: codomain is finite or countable (countably infinite).

Example: $\mathbb{Z}, \mathbb{N}, \{0; 1\}, \dots$

Continuous random variable: codomain is continuous.

Example: $\mathbb{R}, [a, b], \dots$

Description of R.V.

Discrete:

- $p_X(x) = \mathbb{P}[X = x]$ [Probability Mass Function - PMF]
- $F_X(x) = \mathbb{P}[X \leq x] = \sum_{k \leq x} p_X(k)$ [Cumulative Distribution Function - CDF]

Continuous:

- $F_X(x) = \mathbb{P}[X \leq x]$ [CDF or Distribution]
- $p_X(x) = \frac{dF_X(x)}{dx}$ [Probability Distribution Function - PDF]
- $F_X(x) = \int_{-\infty}^x p_X(a)da$

Vector Valued R.V.

Example

$$\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

X_1, X_2 discrete:

$$p_{\mathbf{X}}(\mathbf{x}) \doteq p_{X_1, X_2}(x_1, x_2) = \mathbb{P}_{\mathbf{X}}[X_1 = x_1, X_2 = x_2]$$

X_1, X_2 continuous:

$$F_{\mathbf{X}}(\mathbf{x}) \doteq F_{X_1, X_2}(x_1, x_2) = \mathbb{P}_{\mathbf{X}}[X_1 \leq x_1, X_2 \leq x_2]$$

[“ $X_1 = x_1, X_2 = x_2$ ” and “ $X_1 \leq x_1, X_2 \leq x_2$ ” are joint events]

Note: If \mathbf{X} is obvious, we may write $\mathbb{P}[X_1 = x_1, X_2 = x_2]$ instead of $\mathbb{P}_{\mathbf{X}}[X_1 = x_1, X_2 = x_2]$

Independence

Definition

Two discrete random variables X and Y are **independent** ($X \perp Y$) if and only if

$$\mathbb{P}((X = x) \cap (Y = y)) = \mathbb{P}(X = x) \cdot \mathbb{P}(Y = y)$$

for all values x and y . Similarly, discrete random variables X_1, X_2, \dots, X_k are mutually independent if and only if for **any** subset $I \subseteq [1, k]$ and any values $x_i, i \in I$,

$$\mathbb{P}_x(x) = \prod_{i \in I} \mathbb{P}(X_i = x_i) = \prod_{i \in I} p_{X_i}(x_i).$$

Independence

Definition

The continuous random variables X_1 and X_2 are *independent* if for all x_1, x_2 ,

$$F_{X_1, X_2}(x_1, x_2) = F_{X_1}(x_1)F_{X_2}(x_2)$$

In general, for X_1, \dots, X_n :

$$F_{\mathbf{X}}(\mathbf{x}) = \prod_{i=1}^n F_{X_i}(x_i)$$

Expected Value and Moments

Definition

The expectation of a random variable X is:

- X is discrete: $\mathbb{E}[X] = \sum_x x p_X(x)$
- X is continuous: $\mathbb{E}[X] = \int_{-\infty}^{+\infty} x p_X(x) dx$

Theorem

Let $g(X)$ be a function of a random variable X . Then:

- if X is discrete: $\mathbb{E}[g(X)] = \sum_x g(x)p_X(x)$
- if X is continuous: $\mathbb{E}[g(X)] = \int_{-\infty}^{+\infty} g(x)p_X(x)dx$

For a random variable X we define:

- **Mean:** $m_X \doteq \mathbb{E}[X]$
- **Variance:** $\sigma_X^2 \doteq \mathbb{E}[(X - m_X)^2] = \mathbb{E}[X^2] - m_X^2 = \text{Var}[X]$
- k -th moment: $\mathbb{E}[X^k]$

For a vector valued r.v. $\mathbf{X} \in \mathbb{R}^n$

Expectation:

$$\mathbb{E}[\mathbf{X}] = \begin{bmatrix} m_{X_1} \\ \vdots \\ m_{X_n} \end{bmatrix}$$

Instead of the variance, we have the *covariance matrix*:

$$\Sigma = \mathbb{E}[(\mathbf{X} - m_{\mathbf{X}})(\mathbf{X} - m_{\mathbf{X}})^T] = \begin{bmatrix} \sigma_{X_1}^2 & \sigma_{X_1, X_2} & \cdots & \sigma_{X_1, X_n} \\ \sigma_{X_2, X_1} & \sigma_{X_2}^2 & \vdots & \sigma_{X_2, X_n} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{X_n, X_1} & \sigma_{X_n, X_2} & \cdots & \sigma_{X_n}^2 \end{bmatrix}$$

where

$$\sigma_{X_i, X_j} = \text{Cov}(X_i, X_j) \doteq \mathbb{E}[(X_i - m_{X_i})(X_j - m_{X_j})]$$

σ_{X_i, X_j} is the **covariance** of X_i and X_j

Theorem

If X_1, X_2 are independent then $\sigma_{X_1, X_2} = 0$.

The other direction is not true!

Counterexample

X_1 such that: $\mathbb{P}[X_1 = -1] = \mathbb{P}[X_1 = 1] = 1/2$

$$X_2 = \begin{cases} 0 & \text{(with probability 1)} & \text{if } X_1 = -1 \\ -1 & \text{with probability 1/2} & \text{if } X_1 = 1 \\ 1 & \text{with probability 1/2} & \text{if } X_1 = 1 \end{cases}$$

X_1 and X_2 are clearly not independent (if I know X_2 , I know X_1)!

But

$$\sigma_{X_1, X_2} = \mathbb{E}[(X_1 - m_{X_1})(X_2 - m_{X_2})] = \mathbb{E}[X_1 X_2] = 0$$

Theorem (Properties of Mean, Variance, etc.)

- $\mathbb{E}[X_1 + X_2] = \mathbb{E}[X_1] + \mathbb{E}[X_2]$
- $\text{Var}[aX + b] = a^2\text{Var}[X]$
- $\text{Var}[X_1 + X_2] = \text{Var}[X_1] + \text{Var}[X_2] + 2\sigma_{X_1, X_2}$

Corollary

If $\sigma_{X_1, X_2} = 0$ then $\text{Var}[X_1 + X_2] = \text{Var}[X_1] + \text{Var}[X_2]$

Conditional Probability

Definition (Conditional probability)

A, B are events: $\mathbb{P}[A|B] \doteq \frac{\mathbb{P}[A \cap B]}{\mathbb{P}[B]}$. Well defined only if $\mathbb{P}[B] > 0$.

Example (Relative frequency, convergence, and conditional probability)

Consider an event A . X_1, \dots, X_n that are independent and identically distributed (i.i.d.) random variables that are indicator functions:

$$X_i(z) = \begin{cases} 1 & \text{if } z \in A \\ 0 & \text{if } z \notin A \end{cases}$$

$$S_n = \sum_{i=1}^n X_i$$

Relative frequency: $f_n(A) \doteq \frac{S_n}{n}$

Example

Coin flips, event A = “the result of the coin flip is head”

Each X_i is a **Bernoulli r.v.** of parameter p : $X_i \sim B(p)$

$$p = \mathbb{P}[X_i = 1] = \mathbb{P}[z \in A]$$

Then $S_n = \sum_{i=1}^n X_i$ is a **Binomial r.v.** of parameters n, p :

$$S_n \sim Bin(n, p)$$

$$\mathbb{P}[S_n = k] = \binom{n}{k} p^k (1-p)^{n-k}$$

Then

$$\mathbb{E}[S_n] = np$$

$$\mathbf{Var}[S_n] = np(1-p)$$

Exercise

Derive $\mathbb{E}[S_n]$ and $\text{Var}[S_n]$.

The going back to the relative frequency $f_n(A) \doteq \frac{S_n}{n}$:

$$\mathbb{E}[f_n(A)] = p$$

and

$$\text{Var}[f_n(A)] = \frac{p(1-p)}{n}$$

Theorem (Chebyshev's inequality)

Let X be a r.v. with $\mathbb{E}[X] = \mu$ and $\text{Var}[X] = \sigma^2$. Then:

$$\mathbb{P}[|X - \mu| > \varepsilon] \leq \frac{\sigma^2}{\varepsilon^2}.$$

Machine Learning

Probability Review

Fabio Vandin

October 14th, 2019

Example (Relative frequency, convergence, and conditional probability)

Consider an event A . X_1, \dots, X_n that are independent and identically distributed (i.i.d.) random variables that are indicator functions:

$$X_i(z) = \begin{cases} 1 & \text{if } z \in A \\ 0 & \text{if } z \notin A \end{cases}$$

$$S_n = \sum_{i=1}^n X_i$$

Relative frequency: $f_n(A) \doteq \frac{S_n}{n}$

Example

Coin flips, event A = “the result of the coin flip is head”

Each X_i is a **Bernoulli r.v.** of parameter p : $X_i \sim B(p)$

$$p = \mathbb{P}[X_i = 1] = \mathbb{P}[z \in A]$$

Then $S_n = \sum_{i=1}^n X_i$ is a **Binomial r.v.** of parameters n, p :

$$S_n \sim Bin(n, p)$$

$$\mathbb{P}[S_n = k] = \binom{n}{k} p^k (1-p)^{n-k}$$

Then

$$\mathbb{E}[S_n] = np$$

$$\mathbf{Var}[S_n] = np(1-p)$$

Exercise

Derive $\mathbb{E}[S_n]$ and $\text{Var}[S_n]$.

The going back to the relative frequency $f_n(A) \doteq \frac{S_n}{n}$:

$$\mathbb{E}[f_n(A)] = p$$

and

$$\text{Var}[f_n(A)] = \frac{p(1-p)}{n}$$

Theorem (Chebyshev's inequality)

Let X be a r.v. with $\mathbb{E}[X] = \mu$ and $\text{Var}[X] = \sigma^2$. Then:

$$\mathbb{P}[|X - \mu| > \varepsilon] \leq \frac{\sigma^2}{\varepsilon^2}.$$

Therefore

$$\mathbb{P}[|f_n(A) - p| > \varepsilon] \leq \frac{p(1-p)}{n\varepsilon^2}$$

and

$$\lim_{n \rightarrow +\infty} f_n(A) = p$$

Note: there are tighter bounds than Chebyshev's, like Chernoff's and Hoeffding's - we will see them later.

Intermission

Theorem (Law of Large Numbers)

Let $X_i, i = 1, \dots, n$ be i.i.d. with $\mathbb{E}[X_i] = \mu$ and $\text{Var}[X] = \sigma^2 < +\infty$.

Then

$$\lim_{n \rightarrow +\infty} \mathbb{P} \left[\left| \frac{1}{n} \sum X_i - \mu \right| > \varepsilon \right] = 0.$$

Note: See Jupyter notebook for an example.

Example (continue)

Remark 1:

$$\lim_{n \rightarrow +\infty} f_n(A) = \mathbb{P}[A]$$

Remark 2:

$$\frac{\mathbb{P}[A \cap B]}{\mathbb{P}[B]} = \lim_{n \rightarrow +\infty} \frac{f_n(A \cap B)}{f_n(B)}$$

$$\frac{f_n(A \cap B)}{f_n(B)} = \frac{S_n(A \cap B)}{S_n(B)}$$

it's the fraction of times $A \cap B$ happens among those in which B happens.

Computing Conditional Probabilities

Definition (Conditional probability)

A, B are events: $\mathbb{P}[A|B] \doteq \frac{\mathbb{P}[A \cap B]}{\mathbb{P}[B]}$. Well defined only if $\mathbb{P}[B] > 0$.

Theorem (Bayes Rule)

$$\mathbb{P}[A|B] = \frac{\mathbb{P}[B|A]\mathbb{P}[A]}{\mathbb{P}[B]}$$

Theorem (Law of Total Probability)

Let C_1, C_2, \dots, C_n be a partition of Ω :

- $\cup_{i=1}^n C_i = \Omega$
- $C_i \cap C_j = \emptyset$

For all $A \subset \Omega$:

$$\mathbb{P}[A] = \sum_{i=1}^n \mathbb{P}[A|C_i]\mathbb{P}[C_i]$$

Example: $\mathbb{P}[B] = \mathbb{P}[B|A]\mathbb{P}[A] + \mathbb{P}[B|A^c]\mathbb{P}[A^c]$

Example

M = “have a rare disease”, with $\mathbb{P}[M] = 10^{-9}$

T = “test for the disease is positive” with:

- $\mathbb{P}[T|M] = 0.99$ (1% false negatives)
- $\mathbb{P}[T|M^c] = 0.001$ (0.1% false positives)

If you test positive, what is the probability that you have the disease?

$$\begin{aligned}\mathbb{P}[M|T] &= \frac{\mathbb{P}[T|M]}{\mathbb{P}[T]}\mathbb{P}[M] = \frac{0.99 * 10^{-9}}{0.99 * 10^{-9} + 0.001(1 - 10^{-9})} \\ &\approx \frac{1}{1 + 10^6} \approx 10^{-6}\end{aligned}$$

Example: Gaussian (Normal) R. V.

Let \mathbf{X} be a Gaussian (Normal) random vector.

$$\mathbf{X} \sim \mathcal{N}(\mu, \Sigma), \mathbf{X} \in \mathbb{R}^n$$

$$p_{\mathbf{X}}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} \sqrt{\det \Sigma}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)}$$

Scalar case: $\Sigma = \sigma^2$

$$p_{\mathbf{X}}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

Why Normal Distribution?

Empirical observation: Many random phenomena follow (at least approximately) a Normal distribution.

- Height, weight, income,....
- The velocity of molecule in gas (Brownian Motion);
- Measurement error;
-

The Central Limit Theorem:

“The distribution of the average of large number of independent random variable converges to the Normal distribution”.

Machine Learning

Learning Model

Fabio Vandin

October 14th, 2019

A Formal Model (Statistical Learning)

We have a *learner* (us, or the machine) has access to:

- ① **Domain set \mathcal{X}** : set of all possible objects to make predictions about
 - domain point $x \in \mathcal{X} = \text{instance}$, usually represented by a vector of *features*
 - \mathcal{X} is the *instance space*
- ② **Label set \mathcal{Y}** : set of possible labels.
 - often two labels, e.g $\{-1, +1\}$ or $\{0, 1\}$
- ③ **Training data $S = ((x_1, y_1), \dots, (x_m, y_m))$** : finite sequence of labeled domain points, i.e. pairs in $\mathcal{X} \times \mathcal{Y}$
 - this is the learner's **input**
 - S : *training example* or *training set*

④ **Learner's output h :** prediction rule $h : \mathcal{X} \rightarrow \mathcal{Y}$

- also called *predictor*, *hypothesis*, or *classifier*
- $A(S)$: prediction rule produced by learning algorithm A when training set S is given to it
- sometimes \hat{f} used instead of h

⑤ **Data-generation model:** instances are generated by some probability distribution and labeled according to a function

- \mathcal{D} : probability distribution over \mathcal{X} (**NOT KNOWN TO THE LEARNER!**)
- labeling function $f: \mathcal{X} \rightarrow \mathcal{Y}$ (**NOT KNOWN TO THE LEARNER!**)
- label y_i of instance x_i : $y_i = f(x_i)$, for all $i = 1, \dots, m$
- each point in training set S : first sample x_i according to \mathcal{D} , then label it as $y_i = f(x_i)$

⑥ **Measures of success:** *error of a classifier* = probability it does not predict the correct label on a random data point generate by distribution \mathcal{D}

Loss

Given domain subset $A \subset \mathcal{X}$, $\mathcal{D}(A)$ = probability of observing a point $x \in A$.

In many cases, we refer to A as *event* and express it using a function $\pi : \mathcal{X} \rightarrow \{0, 1\}$, that is:

$$A = \{x \in \mathcal{X} : \pi(x) = 1\}$$

In this case we have $\mathbb{P}_{x \sim \mathcal{D}}[\pi(x)] = \mathcal{D}(A)$

Error of prediction rule $h : \mathcal{X} \rightarrow \mathcal{Y}$ is

$$L_{\mathcal{D}, f}(h) \stackrel{\text{def}}{=} \mathbb{P}_{x \sim \mathcal{D}}[h(x) \neq f(x)] \stackrel{\text{def}}{=} \mathcal{D}(\{x : h(x) \neq f(x)\})$$

Notes:

- $L_{\mathcal{D}, f}(h)$ has many different names: **generalization error**, *true error*, *risk*, **loss**, ...
- often f is obvious, so omitted: $L_{\mathcal{D}}(h)$

Empirical Risk Minimization

Learner outputs $h_S : \mathcal{X} \rightarrow \mathcal{Y}$.

Goal: find h_S which minimizes the generalization error $L_{\mathcal{D},f}(h)$

$L_{\mathcal{D},f}(h)$ is unknown!

What about considering the error on the training data?

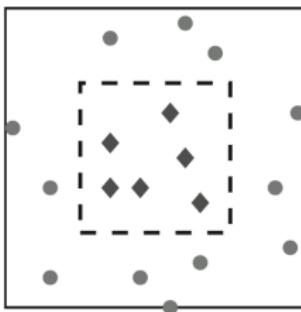
Training error: $L_S(h) \stackrel{\text{def}}{=} \frac{|\{i:h(x_i) \neq y_i, 1 \leq i \leq m\}|}{m}$

Note: also called *empirical error* or *empirical risk*

Empirical Risk Minimization (ERM): produce in output h minimizing $L_S(h)$

What can go wrong?

Consider our simplified movie ratings prediction problem. Assume data is given by:



Assume \mathcal{D} is:

- instance x is taken uniformly at random in the square
- label is 1 if x inside the inner square, 0 otherwise
- area inner square = 1, area larger square = 2

Consider classifier given by

$$h_S(x) = \begin{cases} y_i & \text{if } \exists i \in \{1, \dots, m\} : x_i = x \\ 0 & \text{otherwise} \end{cases}$$

Is it a good predictor?

$$L_S(h_S) = 0 \text{ but } L_{\mathcal{D},f}(h_S) = 1/2$$

Good results on training data but poor generalization error
⇒ **overfitting**

When does ERM lead to good performances in terms of generalization error?

Hypothesis Class and ERM

Apply ERM over a restricted set of hypotheses $\mathcal{H} = \textit{hypothesis class}$

- each $h \in \mathcal{H}$ is a function $h : \mathcal{X} \rightarrow \mathcal{Y}$

$\text{ERM}_{\mathcal{H}}$ learner:

$$\text{ERM}_{\mathcal{H}} \in \arg \min_{h \in \mathcal{H}} L_S(h)$$

Which hypothesis classes \mathcal{H} do not lead to overfitting?

Finite Hypothesis Classes

Assume \mathcal{H} is a finite class: $|\mathcal{H}| < \infty$

Let h_S be the output of $\text{ERM}_{\mathcal{H}}(S)$, i.e. $h_S \in \arg \min_{h \in \mathcal{H}} L_S(h)$

Assumptions

- **Realizability:** there exists $h^* \in \mathcal{H}$ such that $L_{D,f}(h^*) = 0$
- **i.i.d.:** examples in the training set are independently and identically distributed (i.i.d) according to \mathcal{D} , that is $S \sim \mathcal{D}^m$

Observation: realizability assumption implies that $L_S(h^*) = 0$

Can we learn h^* ?

(Simplified) PAC learning

Probably Approximately Correct (PAC) learning

Since the training data comes from \mathcal{D} :

- we can only be **approximately** correct
- we can only be **probably** correct

Parameters:

- *accuracy parameter* ε : we are satisfied with a *good* h_S :
 $L_{\mathcal{D},f}(h_S) \leq \varepsilon$
- *confidence parameter* δ : want h_S to be a *good* hypothesis
with probability $\geq 1 - \delta$

Theorem

Let \mathcal{H} be a finite hypothesis class. Let $\delta \in (0, 1)$, $\varepsilon \in (0, 1)$, and $m \in \mathbb{N}$ such that

$$m \geq \frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}.$$

Then for any f and any \mathcal{D} for which the realizability assumption holds, with probability $\geq 1 - \delta$ we have that for every ERM hypothesis h_S it holds that

$$L_{\mathcal{D}, f}(h_S) \leq \varepsilon.$$

PAC Learning

Definition (PAC learnability)

A hypothesis class \mathcal{H} is PAC learnable if there exist a function $m_{\mathcal{H}}: (0, 1)^2 \rightarrow \mathbb{N}$ and a learning algorithm such that for every $\delta, \varepsilon \in (0, 1)$, for every distribution \mathcal{D} over \mathcal{X} , and for every labeling function $f: \mathcal{X} \rightarrow \{0, 1\}$, if the realizability assumption holds with respect to $\mathcal{H}, \mathcal{D}, f$, then when running the learning algorithm on $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$ i.i.d. examples generate by \mathcal{D} and labeled by f , the algorithm returns a hypothesis h such that, with probability $\geq 1 - \delta$ (over the choice of examples): $\mathcal{L}_{\mathcal{D}, f}(h) \leq \varepsilon$.

$m_{\mathcal{H}}: (0, 1)^2 \rightarrow \mathbb{N}$: sample complexity of learning \mathcal{H} .

- $m_{\mathcal{H}}$ is the minimal integer that satisfies the requirements.

Machine Learning

Learning Model

Fabio Vandin

October 21th, 2019

Theorem

Let \mathcal{H} be a finite hypothesis class. Let $\delta \in (0, 1)$, $\varepsilon \in (0, 1)$, and $m \in \mathbb{N}$ such that

$$m \geq \frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}.$$

Then for any f and any \mathcal{D} for which the realizability assumption holds, with probability $\geq 1 - \delta$ we have that for every ERM hypothesis h_S it holds that

$$L_{\mathcal{D}, f}(h_S) \leq \varepsilon.$$

Proof: on the board and in the book.

PAC Learning

Definition (PAC learnability)

A hypothesis class \mathcal{H} is PAC learnable if there exist a function $m_{\mathcal{H}}: (0, 1)^2 \rightarrow \mathbb{N}$ and a learning algorithm such that for every $\delta, \varepsilon \in (0, 1)$, for every distribution \mathcal{D} over \mathcal{X} , and for every labeling function $f: \mathcal{X} \rightarrow \{0, 1\}$, if the realizability assumption holds with respect to $\mathcal{H}, \mathcal{D}, f$, then when running the learning algorithm on $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$ i.i.d. examples generate by \mathcal{D} and labeled by f , the algorithm returns a hypothesis h such that, with probability $\geq 1 - \delta$ (over the choice of examples): $\mathcal{L}_{\mathcal{D}, f}(h) \leq \varepsilon$.

$m_{\mathcal{H}}: (0, 1)^2 \rightarrow \mathbb{N}$: sample complexity of learning \mathcal{H} .

- $m_{\mathcal{H}}$ is the minimal integer that satisfies the requirements.

Corollary

Every finite hypothesis class is PAC learnable with sample complexity $m_{\mathcal{H}}(\varepsilon, \delta) \leq \lceil \frac{\log(|\mathcal{H}|/\delta)}{\varepsilon} \rceil$.

A More General Learning Model: Remove Realizability Assumption (Agnostic PAC Learning)

Realizability Assumption: there exists $h^* \in \mathcal{H}$ such that
 $L_{\mathcal{D}, f}(h^*) = 0$

Informally: the label is fully determined by the instance x

⇒ Too strong in many applications!

Relaxation: \mathcal{D} is a probability distribution over $\mathcal{X} \times \mathcal{Y}$

⇒ \mathcal{D} is the *joint distribution* over domain points and labels.

For example, two components of \mathcal{D} :

- \mathcal{D}_x : (marginal) distribution over domain points
- $\mathcal{D}((x, y)|x)$: conditional distribution over labels for each domain point

Given x , label y is obtained according to a conditional probability $\mathbb{P}[y|x]$.

The Empirical and True Error

With \mathcal{D} that is a probability distribution over $\mathcal{X} \times \mathcal{Y}$ the *true error* (or risk) is:

$$L_{\mathcal{D}}(h) \stackrel{\text{def}}{=} \mathbb{P}_{(x,y) \sim \mathcal{D}}[h(x) \neq y]$$

As before \mathcal{D} is not known to the learner; the learner only knows the training data S

Empirical risk: as before, that is

$$L_S(h) \stackrel{\text{def}}{=} \frac{|\{i, 0 \leq i \leq m : h(x_i) \neq y_i\}|}{m}$$

Note: $L_S(h)$ = probability that for a pair (x_i, y_i) taken uniformly at random from S the event “ $h(x_i) \neq y_i$ ” holds.

An Optimal Predictor

Learner's goal: find $h : \mathcal{X} \rightarrow \mathcal{Y}$ minimizing $L_{\mathcal{D}}(h)$

Is there a *best predictor*?

Given a probability distribution \mathcal{D} over $\mathcal{X} \times \{0, 1\}$, the best predictor is the **Bayes Optimal Predictor**

$$f_{\mathcal{D}}(x) = \begin{cases} 1 & \text{if } \mathbb{P}[y = 1|x] \geq 1/2 \\ 0 & \text{otherwise} \end{cases}$$

Proposition

For any classifier $g : \mathcal{X} \rightarrow \{0, 1\}$, it holds $L_{\mathcal{D}}(f_{\mathcal{D}}) \leq L_{\mathcal{D}}(g)$.

PROOF: Exercise

Can we use such predictor?

Agnostic PAC Learnability

Consider only predictors from a hypothesis class \mathcal{H} .

We are going to be ok with not finding the best predictor, but not being too far off.

Definition

A hypothesis class \mathcal{H} is agnostic PAC learnable if there exist a function $m_{\mathcal{H}}: (0, 1)^2 \rightarrow \mathbb{N}$ and a learning algorithm such that for every $\delta, \varepsilon \in (0, 1)$, for every distribution \mathcal{D} over $\mathcal{X} \times \mathcal{Y}$, when running the learning algorithm on $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$ i.i.d. examples generated by \mathcal{D} the algorithm returns a hypothesis h such that, with probability $\geq 1 - \delta$ (over the choice of the m training examples):

$$\mathcal{L}_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} \mathcal{L}_{\mathcal{D}}(h') + \varepsilon.$$

Note: this is a generalization of the previous learning model.

A More General Learning Model: Beyond Binary Classification

Binary classification: $\mathcal{Y} = \{0, 1\}$

Other learning problems:

- multiclass classification: classification with > 2 labels
- regression: $\mathcal{Y} = \mathbb{R}$

Multiclass classification: same as before!

Regression

Domain set: \mathcal{X} is usually \mathbb{R}^p for some p .

Target set: \mathcal{Y} is \mathbb{R}

Training data: (as before) $S = ((x_1, y_1), \dots, (x_m, y_m))$

Learner's output: (as before) $h : \mathcal{X} \rightarrow \mathcal{Y}$

Loss: the previous one does not make much sense...

(Generalized) Loss Functions

Definition

Given any hypotheses set \mathcal{H} and some domain Z , a *loss function* is any function $\ell : \mathcal{H} \times Z \rightarrow \mathbb{R}_+$

Risk function = expected loss of a hypothesis $h \in \mathcal{H}$ with respect to \mathcal{D} over Z :

$$L_{\mathcal{D}}(h) \stackrel{\text{def}}{=} \mathbb{E}_{z \sim \mathcal{D}}[\ell(h, z)]$$

Empirical risk = expected loss over a given sample
 $S = (z_1, \dots, z_m) \in Z^m$:

$$L_S(h) \stackrel{\text{def}}{=} \frac{1}{m} \sum_{i=1}^m \ell(h, z_i)$$

Some Common Loss Functions

0-1 loss: $Z = \mathcal{X} \times \mathcal{Y}$

$$\ell_{0-1}(h, (x, y)) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y \end{cases}$$

Commonly used in binary or multiclass classification.

Squared loss: $Z = \mathcal{X} \times \mathcal{Y}$

$$\ell_{sq}(h, (x, y)) \stackrel{\text{def}}{=} (h(x) - y)^2$$

Commonly used in **regression**.

Note: in general, the loss function may depend on the application!
But computational considerations play a role...

Machine Learning

Learning Model

Fabio Vandin

October 24th, 2019

(Generalized) Loss Functions

Definition

Given any hypotheses set \mathcal{H} and some domain Z , a *loss function* is any function $\ell : \mathcal{H} \times Z \rightarrow \mathbb{R}_+$

Risk function = expected loss of a hypothesis $h \in \mathcal{H}$ with respect to \mathcal{D} over Z :

$$L_{\mathcal{D}}(h) \stackrel{\text{def}}{=} \mathbb{E}_{z \sim \mathcal{D}}[\ell(h, z)]$$

Empirical risk = expected loss over a given sample
 $S = (z_1, \dots, z_m) \in Z^m$:

$$L_S(h) \stackrel{\text{def}}{=} \frac{1}{m} \sum_{i=1}^m \ell(h, z_i)$$

Some Common Loss Functions

0-1 loss: $Z = \mathcal{X} \times \mathcal{Y}$

$$\ell_{0-1}(h, (x, y)) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y \end{cases}$$

Commonly used in binary or multiclass classification.

Squared loss: $Z = \mathcal{X} \times \mathcal{Y}$

$$\ell_{sq}(h, (x, y)) \stackrel{\text{def}}{=} (h(x) - y)^2$$

Commonly used in **regression**.

Note: in general, the loss function may depend on the application!
But computational considerations play a role...

Agnostic PAC Learnability for General Loss Functions

Definition

A hypothesis class \mathcal{H} is agnostic PAC learnable with respect to a set Z and a loss function $\ell : \mathcal{H} \times Z \rightarrow \mathbb{R}_+$ if there exist a function $m_{\mathcal{H}} : (0, 1)^2 \rightarrow \mathbb{N}$ and a learning algorithm such that for every $\delta, \varepsilon \in (0, 1)$, for every distribution \mathcal{D} over Z , when running the learning algorithm on $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$ i.i.d. examples generated by \mathcal{D} the algorithm returns a hypothesis h such that, with probability $\geq 1 - \delta$ (over the choice of the m training examples):

$$\mathcal{L}_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} \mathcal{L}_{\mathcal{D}}(h') + \varepsilon$$

where $\mathcal{L}_{\mathcal{D}}(h) = \mathbb{E}_{z \sim \mathcal{D}}[\ell(h, z)]$

Leslie Valiant, Turing award 2010

For transformative contributions to the theory of computation, including the theory of probably approximately correct (PAC) learning, the complexity of enumeration and of algebraic computation, and the theory of parallel and distributed computing.



Bibliography

Up to now:
[UML] Chapter 2 and Chapter 3

Machine Learning

Uniform Convergence

Fabio Vandin

October 24th, 2019

When is an Hypothesis Class PAC Learnable?

Previously seen result: for binary classification with

- realizability assumption
- 0-1 loss

any finite hypothesis class is PAC learnable by ERM.

What about the more general PAC learning model we have seen last? Recall the (agnostic) PAC learnability for general loss:

Definition

A hypothesis class \mathcal{H} is agnostic PAC learnable with respect to a set Z and a loss function $\ell : \mathcal{H} \times Z \rightarrow \mathbb{R}_+$ if there exist a function $m_{\mathcal{H}} : (0, 1)^2 \rightarrow \mathbb{N}$ and a learning algorithm such that for every $\delta, \varepsilon \in (0, 1)$, for every distribution \mathcal{D} over Z , when running the learning algorithm on $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$ i.i.d. examples generated by \mathcal{D} the algorithm returns a hypothesis h such that, with probability $\geq 1 - \delta$ (over the choice of the m training examples):

$$L_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \varepsilon$$

where $L_{\mathcal{D}}(h) = \mathbb{E}_{z \sim \mathcal{D}}[\ell(h, z)]$

Uniform Convergence and Learnability

Uniform convergence: the empirical risks (training error) of *all* members of \mathcal{H} are good approximations of their true risk (generalization error).

Definition

A training set S is called ε -representative (w.r.t. domain Z , hypothesis class \mathcal{H} , loss function ℓ , and distribution D) if

$$\forall h \in \mathcal{H}, |L_S(h) - L_D(h)| \leq \varepsilon$$

Proposition

Assume that training set S is $\frac{\varepsilon}{2}$ -representative (w.r.t. domain Z , hypothesis class \mathcal{H} , loss function ℓ , and distribution \mathcal{D}). Then, any output of $\text{ERM}_{\mathcal{H}}(S)$ (i.e., any $h_S \in \arg \min_{h \in \mathcal{H}} L_S(h)$) satisfies

$$L_{\mathcal{D}}(h_S) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \varepsilon$$

Proof.

For every $h \in \mathcal{H}$:

$$\begin{aligned} L_{\mathcal{D}}(h_S) &\leq L_S(h_S) + \frac{\varepsilon}{2} \\ &\leq L_S(h) + \frac{\varepsilon}{2} \\ &\leq L_{\mathcal{D}}(h) + \frac{\varepsilon}{2} + \frac{\varepsilon}{2} \\ &= L_{\mathcal{D}}(h) + \varepsilon \end{aligned}$$



Uniform convergence depends on training set: when do we have uniform convergence?

Definition

A hypothesis class \mathcal{H} has the uniform convergence property (w.r.t. a domain Z and a loss function ℓ) if there exists a function $m_{\mathcal{H}}^{UC} : (0, 1)^2 \rightarrow \mathbb{N}$ such that for every $\varepsilon, \delta \in (0, 1)$ and for every probability distribution \mathcal{D} over Z , if S is a sample of $m \geq m_{\mathcal{H}}^{UC}(\varepsilon, \delta)$ i.i.d. examples drawn from \mathcal{D} , then with probability $\geq 1 - \delta$, S is ε -representative.

Proposition

If a class \mathcal{H} has the uniform convergence property with a function $m_{\mathcal{H}}^{UC}$ then the class is agnostically PAC learnable with the sample complexity $m_{\mathcal{H}}(\varepsilon, \delta) \leq m_{\mathcal{H}}^{UC}(\varepsilon/2, \delta)$. Furthermore, in that case the ERM $_{\mathcal{H}}$ paradigm is a successful agnostic PAC learner for \mathcal{H} .

What classes of hypotheses have uniform convergence?

Finite Classes are Agnostic PAC Learnable

We prove that finite sets of hypotheses are agnostic PAC learnable under some restriction for the loss.

Proposition

Let \mathcal{H} be a finite hypothesis class, let Z be a domain, and let $\ell : \mathcal{H} \times Z \rightarrow [0, 1]$ be a loss function. Then:

- \mathcal{H} enjoys the uniform convergence property with sample complexity

$$m_{\mathcal{H}}^{UC}(\varepsilon, \delta) \leq \left\lceil \frac{\log(2|\mathcal{H}|/\delta)}{2\varepsilon^2} \right\rceil$$

- \mathcal{H} is agnostically PAC learnable using the ERM algorithm with sample complexity

$$m_{\mathcal{H}}(\varepsilon, \delta) \leq m_{\mathcal{H}}^{UC}(\varepsilon/2, \delta) \leq \left\lceil \frac{2 \log(2|\mathcal{H}|/\delta)}{\varepsilon^2} \right\rceil$$

Idea of the proof:

- ① prove that uniform convergence holds for a finite hypothesis class
- ② use previous result on uniform convergence and PAC learnability

Proof: on the board and on the book.

Useful tool: Hoeffding's Inequality

Hoeffding's Inequality

Let $\theta_1, \dots, \theta_m$ be a sequence of i.i.d. random variables and assume that for all i , $\mathbb{E}[\theta_i] = \mu$ and $\mathbb{P}[a \leq \theta_i \leq b] = 1$. Then, for any $\varepsilon > 0$

$$\mathbb{P}\left[\left|\frac{1}{m} \sum_{i=1}^m \theta_i - \mu\right| > \varepsilon\right] \leq 2e^{-\frac{2m\varepsilon^2}{(b-a)^2}}$$

Bibliography

[UML] Chapter 4

Machine Learning

Linear Models

Fabio Vandin

October 24th, 2019

Linear Predictors and Affine Functions

Consider $\mathcal{X} = \mathbb{R}^d$

“Linear” (affine) functions:

$$L_d = \{h_{\mathbf{w}, b} : \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

where

$$h_{\mathbf{w}, b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \left(\sum_{i=1}^d w_i x_i \right) + b$$

Note:

- each member of L_d is a function $\mathbf{x} \rightarrow \langle \mathbf{w}, \mathbf{x} \rangle + b$
- b : bias

Linear Models

Hypothesis class \mathcal{H} : $\phi \circ L_d$, where $\phi : \mathbb{R} \rightarrow \mathcal{Y}$

- $h \in \mathcal{H}$ is $h : \mathbb{R}^d \rightarrow \mathcal{Y}$

ϕ depends on the learning problem

Example

- binary classification, $\mathcal{Y} = \{-1, 1\} \Rightarrow \phi(z) = \text{sign}(z)$
- regression, $\mathcal{Y} = \mathbb{R} \Rightarrow \phi(z) = z$

Equivalent Notation

Given $\mathbf{x} \in \mathcal{X}$, $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$, define:

- $\mathbf{w}' = (b, w_1, w_2, \dots, w_d) \in \mathbb{R}^{d+1}$
- $\mathbf{x}' = (1, x_1, x_2, \dots, x_d) \in \mathbb{R}^{d+1}$

Then:

$$h_{\mathbf{w}, b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \langle \mathbf{w}', \mathbf{x}' \rangle \quad (1)$$

⇒ we will consider bias term as part of \mathbf{w} and assume
 $\mathbf{x} = (1, x_1, x_2, \dots, x_d)$ when needed, with $h_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$

Machine Learning

Linear Models

Fabio Vandin

October 28th, 2019

Linear Predictors and Affine Functions

Consider $\mathcal{X} = \mathbb{R}^d$

“Linear” (affine) functions:

$$L_d = \{h_{\mathbf{w}, b} : \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

where

$$h_{\mathbf{w}, b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \left(\sum_{i=1}^d w_i x_i \right) + b$$

Note:

- each member of L_d is a function $\mathbf{x} \rightarrow \langle \mathbf{w}, \mathbf{x} \rangle + b$
- b : bias

Linear Models

Hypothesis class \mathcal{H} : $\phi \circ L_d$, where $\phi : \mathbb{R} \rightarrow \mathcal{Y}$

- $h \in \mathcal{H}$ is $h : \mathbb{R}^d \rightarrow \mathcal{Y}$

ϕ depends on the learning problem

Example

- binary classification, $\mathcal{Y} = \{-1, 1\} \Rightarrow \phi(z) = \text{sign}(z)$
- regression, $\mathcal{Y} = \mathbb{R} \Rightarrow \phi(z) = z$

Equivalent Notation

Given $\mathbf{x} \in \mathcal{X}$, $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$, define:

- $\mathbf{w}' = (b, w_1, w_2, \dots, w_d) \in \mathbb{R}^{d+1}$
- $\mathbf{x}' = (1, x_1, x_2, \dots, x_d) \in \mathbb{R}^{d+1}$

Then:

$$h_{\mathbf{w}, b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \langle \mathbf{w}', \mathbf{x}' \rangle \quad (1)$$

⇒ we will consider bias term as part of \mathbf{w} and assume
 $\mathbf{x} = (1, x_1, x_2, \dots, x_d)$ when needed, with $h_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$

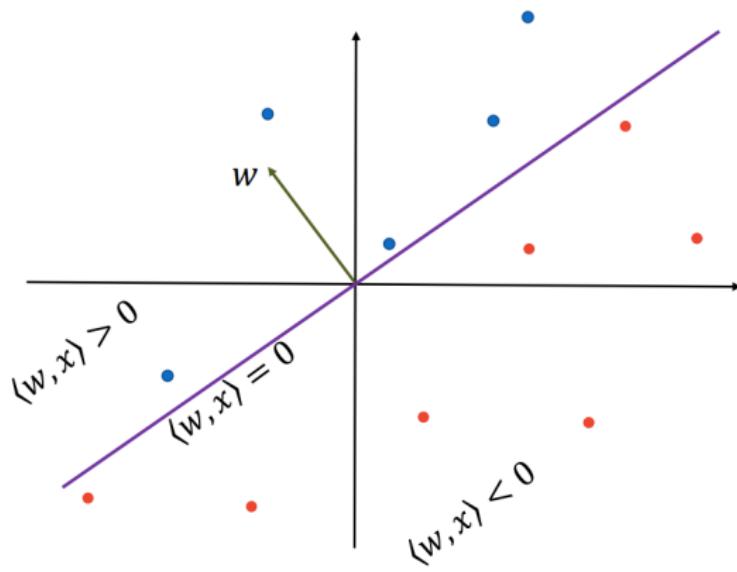
Linear Classification

$\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1, 1\}$, 0-1 loss

Hypothesis class = halfspaces

$$HS_d = \text{sign} \circ L_d = \{\mathbf{x} \rightarrow \text{sign}(h_{\mathbf{w}, b}(\mathbf{x})) : h_{\mathbf{w}, b} \in L_d\} \quad (2)$$

Example: $\mathcal{X} = \mathbb{R}^2$



Finding a Good Hypothesis

Linear classification with hypothesis set \mathcal{H} = halfspaces.

How do we find a good hypothesis?

Good = ERM rule

⇒ Perceptron Algorithm (Rosenblatt, 1958)

Note:

if $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 0$ for all $i = 1, \dots, m \Rightarrow$ all points are classified correctly by model $\mathbf{w} \Rightarrow$ *realizability assumption* for training set

Linearly separable data: there exists \mathbf{w} such that: $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 0$

Perceptron

Input: training set $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$

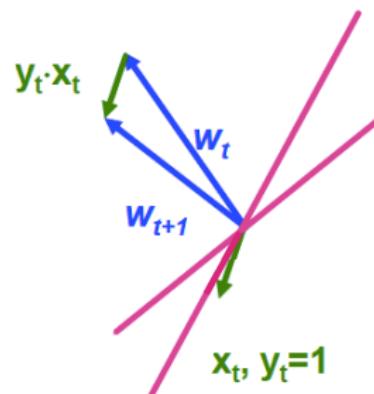
initialize $\mathbf{w}^{(1)} = (0, \dots, 0)$;

for $t = 1, 2, \dots$ **do**

if $\exists i$ s.t. $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0$ **then** $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$;
else return $\mathbf{w}^{(t)}$;

Interpretation of update:

Note that:



$$\begin{aligned} y_i \langle \mathbf{w}^{(t+1)}, \mathbf{x}_i \rangle &= y_i \langle \mathbf{w}^{(t)} + y_i \mathbf{x}_i, \mathbf{x}_i \rangle \\ &= y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle + ||\mathbf{x}_i||^2 \end{aligned}$$

\Rightarrow update guides \mathbf{w} to be “more correct” on (\mathbf{x}_i, y_i) .

Termination? Depends on the realizability assumption!

Perceptron with Linearly Separable Data

Proposition

Assume that $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$ is linearly separable, let:

- $B = \min\{||\mathbf{w}|| : y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1 \quad \forall i, i = 1, \dots, m, \}$, and
- $R = \max_i \|\mathbf{x}_i\|$.

Then the Perceptron algorithm stops after at most $(RB)^2$ iterations (and when it stops it holds that

$$\forall i, i \in \{1, \dots, m\} : y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle > 0.$$

Perceptron: Notes

- simple to implement
- for separable data
 - convergence is guaranteed
 - converge depends on B , which can be exponential in d ...
⇒ other approaches (e.g., ILP - Integer Linear Programming) may be better to find ERM solution in such cases
 - potentially multiple solutions, which one is picked depends on starting values
- non separable data?
 - run for some time and keep best solution found up to that point (*pocket algorithm*)

Machine Learning

Bias-Complexity Trade-off

Fabio Vandin

October 28th, 2019

Our Goal in Learning

Given:

- training set: $S = ((x_1, y_1), \dots, (x_m, y_m))$
- loss function: $\ell(h, (x, y))$

Want: a function \hat{h} such that $L_{\mathcal{D}}(\hat{h})$ is small

We can pick: the learning algorithm A , that given S will produce $\hat{h} = A(S)$

Note: A comprises:

- the hypothesis set \mathcal{H}
- the procedure to pick $\hat{h} = A(S)$ from \mathcal{H}

Question: is there a *universal learner*, i.e., an algorithm A that predicts the best \hat{h} for any distribution \mathcal{D} ?

The No Free Lunch Theorem

The following answers the previous question for some specific settings.

Theorem (No-Free Lunch)

Let A be any learning algorithm for the task of binary classification with respect to the 0-1 loss over a domain \mathcal{X} . Let m be any number smaller than $|\mathcal{X}|/2$, representing a training set size. Then, there exists a distribution \mathcal{D} over $\mathcal{X} \times \{0, 1\}$ such that:

- there exists a function $f: \mathcal{X} \rightarrow \{0, 1\}$ with $L_{\mathcal{D}}(f) = 0$
- with probability of at least $1/7$ over the choice of $S \sim \mathcal{D}^m$ we have that $L_{\mathcal{D}}(A(S)) \geq 1/8$.

Note: there are similar results for other learning tasks.

No Free Lunch and Prior Knowledge

Corollary

Let \mathcal{X} be an infinite domain set and let \mathcal{H} be the set of all functions from \mathcal{X} to $\{0, 1\}$. Then, \mathcal{H} is not PAC learnable.

What's the implication?

We need to use our prior knowledge about \mathcal{D} to pick a good hypothesis set.

How do we choose \mathcal{H} ?

- we would like \mathcal{H} to be *large*, so that it may contain a function h with small $L_{\mathcal{D}}(h)$
- no free lunch $\Rightarrow \mathcal{H}$ cannot be too large!

Error Decomposition

Let h_S be an $\text{ERM}_{\mathcal{H}}$ hypothesis.

Then

$$L_{\mathcal{D}}(h_S) = \epsilon_{\text{app}} + \epsilon_{\text{est}}$$

where

- $\epsilon_{\text{app}} = \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$ (approximation error)
- $\epsilon_{\text{est}} = L_{\mathcal{D}}(h_S) - \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$ (estimation error)

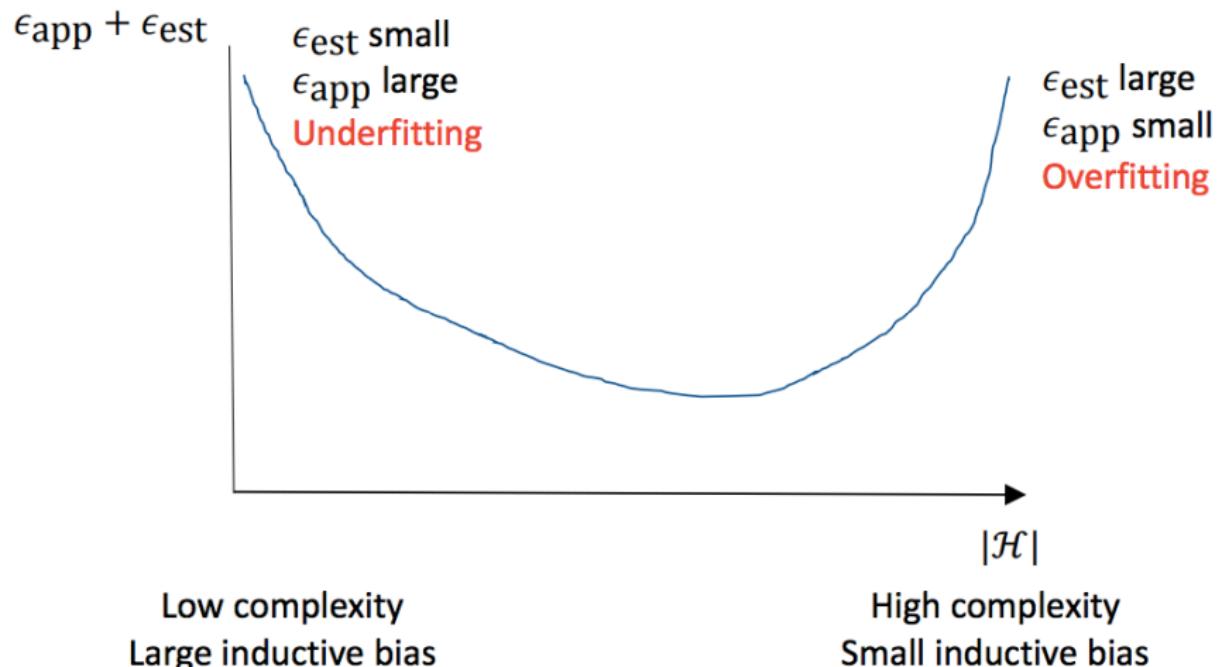
Approximation error: $\epsilon_{\text{app}} = \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$

- derives from our choice of \mathcal{H}
- once we have chosen $\mathcal{H} \Rightarrow \epsilon_{\text{app}}$ is unavoidable!
- to decrease it, chose a “larger” \mathcal{H}

Estimation error: $\epsilon_{\text{est}} = L_{\mathcal{D}}(h_s) - \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$

- derives from our inability to choose (with ERM) the best hypothesis
- could be avoided if had chosen the best hypothesis!
- to decrease, we need a low number of hypotheses in \mathcal{H} so that training error is good estimate of generalization error for all of them \Rightarrow need a “small” \mathcal{H}

Complexity of \mathcal{H} and Error Decomposition



Estimating $L_{\mathcal{D}}(h_S)$

How can we estimate the generalization error $L_{\mathcal{D}}(h)$ for a function h , for example $h_S \in \text{ERM}_{\mathcal{H}}$?

We can use a **test set**: new set of samples not used for picking h_S (=the training set).

Notes:

- the test must not be looked at until we have picked our **final** hypothesis!
- in practice: we have 1 set of samples and we split it in *training set* and *test set*.

Bibliography

[UML] Chapter 5

Machine Learning

Linear Models

Fabio Vandin

October 31st, 2019

Linear Regression

$$\mathcal{X} = \mathbb{R}^d, \mathcal{Y} = \mathbb{R}$$

Hypothesis class:

$$\mathcal{H}_{reg} = L_d = \{\mathbf{x} \rightarrow \langle \mathbf{w}, \mathbf{x} \rangle + b : \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

Note: $h \in H_{reg} : \mathbb{R}^d \rightarrow \mathbb{R}$

Commonly used loss function: *squared-loss*

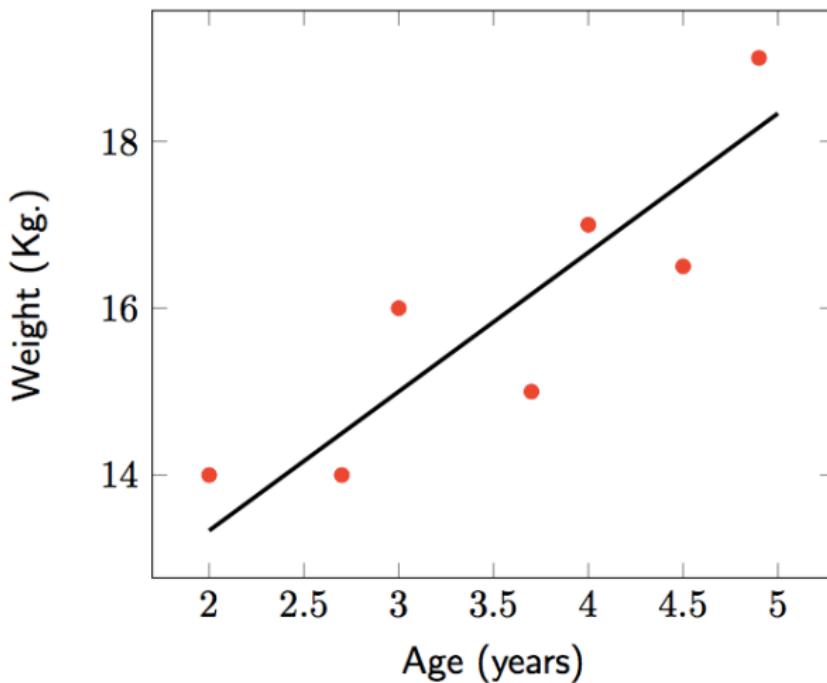
$$\ell(h, (\mathbf{x}, y)) \stackrel{def}{=} (h(\mathbf{x}) - y)^2$$

⇒ empirical risk function (training error): *Mean Squared Error*

$$L_S(h) = \frac{1}{m} \sum_{i=1}^m (h(\mathbf{x}_i) - y_i)^2$$

Linear Regression - Example

$d = 1$



Least Squares

How to find a ERM hypothesis? *Least Squares* algorithm

Best hypothesis:

$$\arg \min_{\mathbf{w}} L_S(h_{\mathbf{w}}) = \arg \min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2$$

Equivalent formulation: \mathbf{w} minimizing *Residual Sum of Squares* (RSS), i.e.

$$\arg \min_{\mathbf{w}} \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2$$

RSS: Matrix Form

Let

$$\mathbf{X} = \begin{bmatrix} \cdots & \mathbf{x}_1 & \cdots \\ \cdots & \mathbf{x}_2 & \cdots \\ \cdots & \vdots & \cdots \\ \cdots & \mathbf{x}_m & \cdots \end{bmatrix}$$

\mathbf{X} : design matrix

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

\Rightarrow we have that RSS is

$$\sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

Want to find \mathbf{w} that minimizes RSS ($=$ objective function):

$$\arg \min_{\mathbf{w}} RSS(\mathbf{w}) = \arg \min_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

How?

Compute gradient $\frac{\partial RSS(\mathbf{w})}{\partial \mathbf{w}}$ of objective function w.r.t \mathbf{w} and compare it to 0.

$$\frac{\partial RSS(\mathbf{w})}{\partial \mathbf{w}} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Then we need to find \mathbf{w} such that

$$-2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0$$

$$-2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0$$

is equivalent to

$$\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

If $\mathbf{X}^T \mathbf{X}$ is invertible \Rightarrow solution to ERM problem is:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Complexity Considerations

We need to compute

$$(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Algorithm:

- ① compute $\mathbf{X}^T \mathbf{X}$: product of $(d + 1) \times m$ matrix and $m \times (d + 1)$ matrix
- ② compute $(\mathbf{X}^T \mathbf{X})^{-1}$ inversion of $(d + 1) \times (d + 1)$ matrix
- ③ compute $(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$: product of $(d + 1) \times (d + 1)$ matrix and $(d + 1) \times m$ matrix
- ④ compute $(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$: product of $(d + 1) \times m$ matrix and $m \times 1$ matrix

Most expensive operation? Inversion!

⇒ done for $(d + 1) \times (d + 1)$ matrix

$$\mathbf{X}^T \mathbf{X} \text{ not invertible?}$$

How do we get \mathbf{w} such that

$$\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

if $\mathbf{X}^T \mathbf{X}$ is not invertible?

Let

$$\mathbf{A} = \mathbf{X}^T \mathbf{X}$$

Let \mathbf{A}^+ be the *generalized inverse* of \mathbf{A} , i.e.:

$$\mathbf{A} \mathbf{A}^+ \mathbf{A} = \mathbf{A}$$

Proposition

If $\mathbf{A} = \mathbf{X}^T \mathbf{X}$ is not invertible, then $\hat{\mathbf{w}} = \mathbf{A}^+ \mathbf{X}^T \mathbf{y}$ is a solution to $\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}$.

Computing the Generalized Inverse of \mathbf{A}

Note $\mathbf{A} = \mathbf{X}^T \mathbf{X}$ is symmetric \Rightarrow eigenvalue decomposition of \mathbf{A} :

$$\mathbf{A} = \mathbf{V} \mathbf{D} \mathbf{V}^T$$

with

- \mathbf{D} : diagonal matrix (entries = eigenvalues of \mathbf{A})
- \mathbf{V} : orthonormal matrix ($\mathbf{V}^T \mathbf{V} = \mathbf{I}_{d \times d}$)

Define \mathbf{D}^+ diagonal matrix such that:

$$\mathbf{D}_{i,i}^+ = \begin{cases} 0 & \text{if } \mathbf{D}_{i,i} = 0 \\ \frac{1}{\mathbf{D}_{i,i}} & \text{otherwise} \end{cases}$$

Let $\mathbf{A}^+ = \mathbf{V}\mathbf{D}^+\mathbf{V}^T$

Then

$$\begin{aligned}\mathbf{A}\mathbf{A}^+\mathbf{A} &= \mathbf{V}\mathbf{D}\mathbf{V}^T\mathbf{V}\mathbf{D}^+\mathbf{V}^T\mathbf{V}\mathbf{D}\mathbf{V}^T \\ &= \mathbf{V}\mathbf{D}\mathbf{D}^+\mathbf{V}^T \\ &= \mathbf{V}\mathbf{D}\mathbf{V}^T \\ &= \mathbf{A}\end{aligned}$$

$\Rightarrow \mathbf{A}^+$ is a generalized inverse of \mathbf{A} .

In practice: the Moore-Penrose generalized inverse \mathbf{A}^\dagger of \mathbf{A} is used, since it can be efficiently computed from the Singular Value Decomposition of \mathbf{A} .

Exercise 1

Consider a linear regression problem, where $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$, with mean squared loss. The hypothesis set is the set of *constant* functions, that is $\mathcal{H} = \{h_a : a \in \mathbb{R}\}$, where $h_a(\mathbf{x}) = a$. Let $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$ denote the training set.

- Derive the hypothesis $h \in \mathcal{H}$ that minimizes the training error.
- Use the result above to explain why, for a given hypothesis \hat{h} from the set of all linear models, the coefficient of determination

$$R^2 = 1 - \frac{\sum_{i=1}^m (\hat{h}(\mathbf{x}_i) - y_i)^2}{\sum_{i=1}^m (y_i - \bar{y})^2}$$

where \bar{y} is the average of the $y_i, i = 1, \dots, m$ is a measure of how well \hat{h} performs (on the training set).

Exercise 2

Your friend has developed a new machine learning algorithm for binary classification (i.e., $y \in \{-1, 1\}$) with 0-1 loss and tells you that it achieves a generalization error of only 0.05. However, when you look at the learning problem he is working on, you find out that $\Pr_{\mathcal{D}}[y = 1] = 0.95\dots$

- Assume that $\Pr_{\mathcal{D}}[y = \ell] = p_\ell$. Derive the generalization error of the (dumb) hypothesis/model that *always* predicts ℓ .
- Use the result above to decide if your friend's algorithm has learned something or not.

Machine Learning

Statistics: Basic Results & Terminology

Fabio Vandin

November 4th, 2019

Example: Gaussian (Normal) R. V.

Let X_i be a Gaussian (Normal) random variable

$$X_i \sim \mathcal{N}(\mu, \sigma_{X_i}^2), X_i \in \mathbb{R}$$

$$p_{X_i}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

Assume that we have *data* generated from X_i : how do we *learn* something about X_i ?

- most likely value for μ ?
- interval containing μ with high probability?
- decide (...) if μ is a given value?

Definition (Confidence set)

We say that $I(\mathbf{x})$ is a *confidence interval* (for a parameter θ) of level $1 - \alpha$, if

$$\mathbb{P}[I(\mathbf{x}) \ni \theta] > 1 - \alpha$$

Desiderata:

we would like the set $I(\mathbf{x})$ to be

- small (e.g. in terms of area of $I(\mathbf{x})$)
- contain θ with high probability, i.e. $\mathbb{P}[I(\mathbf{x}) \ni \theta] > 1 - \alpha$
($\alpha > 0$ and small)

so that we are able to locate θ with:

- *high precision*
- *high confidence.*

Point estimates

Given some data for which we assume some model, and a parameter that we want to estimate (e.g., μ for a Gaussian r.v.), how do we get the *best* estimate of the parameter?

What does “*best*” mean?

Maximum Likelihood Estimation (MLE) [UML, 24.1]

MLE is a statistical approach for finding the parameters that maximize the joint probability of a given dataset *assuming a specific parametric probability function*.

Note: MLE essentially assumes a *generative model* for the data

General approach:

- ① given training set $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$, assume each (\mathbf{x}_i, y_i) is i.i.d. from some probability distribution of parameters θ ; (sometimes $S = (\mathbf{x}_1, \dots, \mathbf{x}_m)$)
- ② consider $\mathbb{P}[S|\theta]$ (*likelihood* of data given parameters)
- ③ *log likelihood*: $L(S; \theta) = \log(\mathbb{P}[S|\theta])$
- ④ *maximum likelihood estimator*: $\hat{\theta} = \arg \max_{\theta} L(S; \theta)$

Point estimate of θ

Going back to the example, we can find the maximum likelihood estimator for θ .

Consider the log likelihood function (for *continuous* random variable):

$$L(\mathbf{x}, \theta) = \log p_{\mathbf{X}}(\mathbf{x}; \theta) = -\frac{m}{2} \log(2\pi\sigma^2) - \frac{1}{2} \sum_{i=1}^m \frac{(x_i - \theta)^2}{\sigma^2}$$

How to find the maximum likelihood estimator $\hat{\theta}_m$ of θ ?

Compute $\frac{d}{d\theta} L(\mathbf{X}, \theta)$ and compare it to 0.

Then the maximum likelihood estimator of θ is

$$\hat{\theta}_m := \arg \max_{\theta} L(\mathbf{x}, \theta) = \frac{1}{m} \sum_{i=1}^m x_i$$

which is the sample mean.

Confidence set for θ

The samples X_1, \dots, X_m are i.i.d. r.v. $\sim \mathcal{N}(\theta, \sigma^2)$

\Rightarrow the sample mean $\hat{\theta}$ is also normal:

$$\hat{\theta} = \frac{1}{m} \sum_{i=1}^m X_i \sim \mathcal{N}\left(\theta, \frac{\sigma^2}{m}\right)$$

it is easy to show (HOMEWORK!) that, for any given confidence $1 - \alpha$, the *smallest* confidence set for θ is symmetric and centered around the sample mean:

$$I(\mathbf{x}) = \left[\hat{\theta}_m - z_{\alpha/2} \frac{\sigma}{\sqrt{m}}, \hat{\theta}_m + z_{\alpha/2} \frac{\sigma}{\sqrt{m}} \right]$$

where $z_{\alpha/2}$ is the $1 - \alpha/2$ -percentile of a zero mean unit variance Normal distribution $\mathcal{N}(0, 1)$, i.e. such that for random variable $Y \sim \mathcal{N}(0, 1)$:

$$\mathbb{P}[Y \leq z_{\alpha/2}] = 1 - \frac{\alpha}{2} \quad \Leftrightarrow \quad \mathbb{P}[Y \geq z_{\alpha/2}] = \frac{\alpha}{2}$$

Examples: $z_{0.05/2} = 1.96$, $z_{0.01/2} \approx 2.58$

Confidence sets: graphical representation

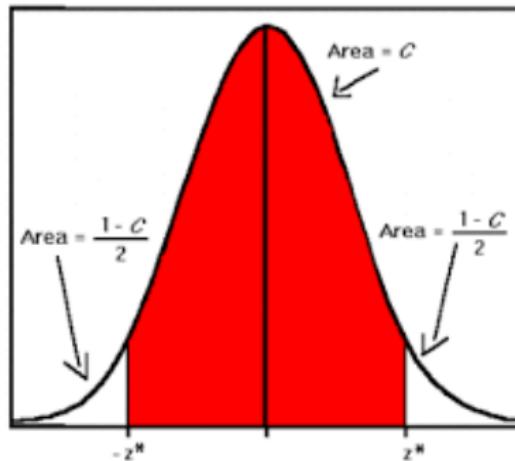


Figure: The smallest confidence set (the symmetric region around the origin) of level (or Confidence) $C = 1 - \alpha$

Machine Learning

Linear Models

Fabio Vandin

November 4th, 2019

Significance Testing for Linear Regression Coefficients

Assumption: data is generated from linear model + noise

- noise: $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
- σ^2 is known

Model:

$$Y = \mathbf{E}[Y|X_1, \dots, X_d] + \varepsilon = b + \sum_{i=1}^d w_i X_i + \varepsilon$$

Least squares: obtain estimate \hat{w}_i of w_i for all $i = 1, \dots, d$

Note: $\hat{b} = \frac{1}{m} \sum_{j=1}^m y_j - \sum_{i=1}^d \hat{w}_i \left(\frac{1}{m} \sum_{j=1}^m x_{ji} \right)$ (x_{ji} is the value of the i -th feature in the j -th sample)

The estimates \hat{w}_i that we get from the linear regression are normally distributed:

$$\hat{w}_i = \mathcal{N}(w_i, v_i \sigma^2)$$

where v_i is the i -th diagonal element of $(\mathbf{X}^T \mathbf{X})^{-1}$

⇒ we can use confidence intervals: see other set of slides!

Consider $z_i = \frac{\hat{w}_i}{\sigma}$: confidence intervals computed as before!
(HOMEWORK).

What about σ^2 ?

σ^2 is unknown.... \Rightarrow need to estimate it from the data!

Estimate $\hat{\sigma}^2$ of σ^2 is given by

$$\hat{\sigma}^2 = \frac{1}{m-d-1} \sum_{i=1}^m (y_i - \hat{y}_i)^2$$

where

$$\hat{y}_i = \hat{b} + \sum_{i=1}^m \hat{w}_i x_i$$

The corresponding normalized variable is

$$z_i = \frac{\hat{w}_i}{\hat{\sigma} \sqrt{v_i}}$$

where v_i is the i -th diagonal element of $(\mathbf{X}^T \mathbf{X})^{-1}$

Proposition

z_i follows a Student's t distribution with $m - d - 1$ degrees of freedom.

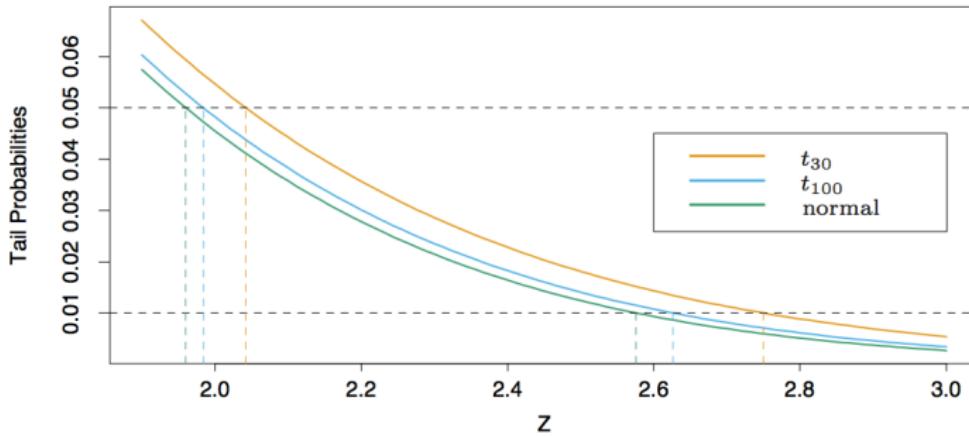


FIGURE 3.3. The tail probabilities $\Pr(|Z| > z)$ for three distributions, t_{30} , t_{100} and standard normal. Shown are the appropriate quantiles for testing significance at the $p = 0.05$ and 0.01 levels. The difference between t and the standard normal becomes negligible for N bigger than about 100.

σ^2 Unknown: Confidence Intervals

The $1 - \alpha$ confidence interval for w_i is then

$$\left(\hat{w}_i - t_{\alpha/2}^{(m-d-1)} \sqrt{v_i} \hat{\sigma}, \hat{w}_i + t_{\alpha/2}^{(m-d-1)} \sqrt{v_i} \hat{\sigma} \right)$$

where

$$t_{\alpha/2}^{(m-d-1)}$$

is the $1 - \alpha/2$ -percentile of the Student's t distribution with $m - d - 1$ degrees of freedom.

Logistic Regression

Learn a function h from \mathbb{R}^d to $[0, 1]$.

What can this be used for?

Classification!

Example: binary classification ($\mathcal{Y} = \{-1, 1\}$) - $h(\mathbf{x})$ = probability that label of \mathbf{x} is 1.

For simplicity of presentation, we consider binary classification with $\mathcal{Y} = \{-1, 1\}$, but similar considerations apply for multiclass classification.

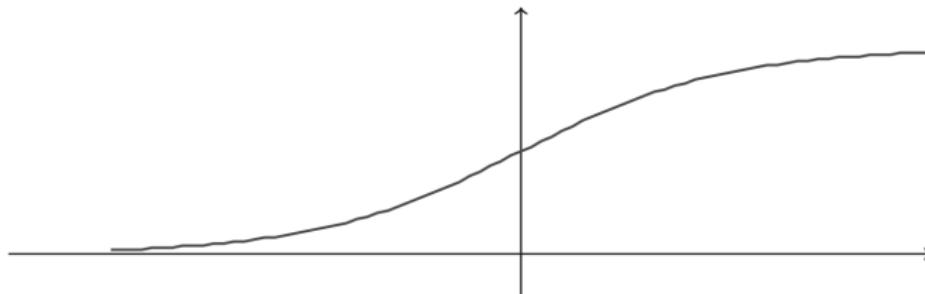
Logistic Regression: Model

Hypothesis class \mathcal{H} : $\phi_{\text{sig}} \circ L_d$, where $\phi_{\text{sig}} : \mathbb{R} \rightarrow [0, 1]$ is *sigmoid function*

Sigmoid function = “S-shaped” function

For logistic regression, the sigmoid ϕ_{sig} used is the *logistic regression*:

$$\phi_{\text{sig}}(z) = \frac{1}{1 + e^{-z}}$$



Therefore

$$H_{\text{sig}} = \phi_{\text{sig}} \circ L_d = \{\mathbf{x} \rightarrow \phi_{\text{sig}}(\langle \mathbf{w}, \mathbf{x} \rangle) : \mathbf{w} \in \mathbb{R}^d\}$$

and $h_{\mathbf{w}}(\mathbf{x}) \in H_{\text{sig}}$ is:

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\langle \mathbf{w}, \mathbf{x} \rangle}}$$

Main difference with binary classification with halfspaces: when
 $\langle \mathbf{w}, \mathbf{x} \rangle \approx 0$

- halfspace prediction is deterministically 1 or -1
- $\phi_{\text{sig}}(\langle \mathbf{w}, \mathbf{x} \rangle) \approx 1/2 \Rightarrow$ uncertainty in predicted label

Loss Function

Need to define how bad it is to predict $h_{\mathbf{w}}(\mathbf{x}) \in [0, 1]$ given that true label is $y = \pm 1$

Desiderata

- $h_{\mathbf{w}}(\mathbf{x})$ “large” if $y = 1$
- $1 - h_{\mathbf{w}}(\mathbf{x})$ “large” if $y = -1$

Note that

$$\begin{aligned}1 - h_{\mathbf{w}}(\mathbf{x}) &= 1 - \frac{1}{1 + e^{-\langle \mathbf{w}, \mathbf{x} \rangle}} \\&= \frac{e^{-\langle \mathbf{w}, \mathbf{x} \rangle}}{1 + e^{-\langle \mathbf{w}, \mathbf{x} \rangle}} \\&= \frac{1}{1 + e^{\langle \mathbf{w}, \mathbf{x} \rangle}}\end{aligned}$$

Then *reasonable* loss function: increases monotonically with

$$\frac{1}{1 + e^{y\langle \mathbf{w}, \mathbf{x} \rangle}}$$

⇒ *reasonable* loss function: increases monotonically with

$$1 + e^{-y\langle \mathbf{w}, \mathbf{x} \rangle}$$

Loss function for logistic regression:

$$\ell(h_{\mathbf{w}}, (\mathbf{x}, y)) = \log \left(1 + e^{-y\langle \mathbf{w}, \mathbf{x} \rangle} \right)$$

Therefore, given training set $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$ the ERM problem for logistic regression is:

$$\arg \min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^m \log \left(1 + e^{-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle} \right)$$

Notes: logistic loss function is a *convex function* \Rightarrow ERM problem can be solved efficiently

Definition may look a bit arbitrary: actually, ERM formulation is the same as the one arising from *Maximum Likelihood Estimation*

Maximum Likelihood Estimation (MLE) [UML, 24.1]

MLE is a statistical approach for finding the parameters that maximize the joint probability of a given dataset *assuming a specific parametric probability function*.

Note: MLE essentially assumes a *generative model* for the data

General approach:

- ① given training set $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$, assume each (\mathbf{x}_i, y_i) is i.i.d. from some probability distribution of parameters θ
- ② consider $\mathbb{P}[S|\theta]$ (likelihood of data given parameters)
- ③ *log likelihood*: $L(S; \theta) = \log(\mathbb{P}[S|\theta])$
- ④ *maximum likelihood estimator*: $\hat{\theta} = \arg \max_{\theta} L(S; \theta)$

Logistic Regression and MLE

Assuming $\mathbf{x}_1, \dots, \mathbf{x}_m$ are fixed, the probability that \mathbf{x}_i has label $y_i = 1$ is

$$h_{\mathbf{w}}(\mathbf{x}_i) = \frac{1}{1 + e^{-\langle \mathbf{w}, \mathbf{x}_i \rangle}}$$

while the probability that \mathbf{x}_i has label $y_i = -1$ is

$$(1 - h_{\mathbf{w}}(\mathbf{x}_i)) = \frac{1}{1 + e^{\langle \mathbf{w}, \mathbf{x}_i \rangle}}$$

Then the likelihood for training set S is:

$$\prod_{i=1}^m \left(\frac{1}{1 + e^{-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle}} \right)$$

Therefore the log likelihood is:

$$-\sum_{i=1}^m \log \left(1 + e^{-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle} \right)$$

And note that the maximum likelihood estimator for \mathbf{w} is:

$$\arg \max_{\mathbf{w} \in \mathbb{R}^d} -\sum_{i=1}^m \log \left(1 + e^{-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle} \right) = \arg \min_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^m \log \left(1 + e^{-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle} \right)$$

⇒ MLE solution is equivalent to ERM solution!

Bibliography

[UML] Chapter 9:

- no 9.1.1

Machine Learning

Statistics: Basic Results & Terminology

Fabio Vandin

November 4th, 2019

Example: Gaussian (Normal) R. V.

Let X_i be a Gaussian (Normal) random variable

$$X_i \sim \mathcal{N}(\mu, \sigma_{X_i}^2), X_i \in \mathbb{R}$$

$$p_{X_i}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

Assume that we have *data* generated from X_i : how do we *learn* something about X_i ?

- most likely value for μ ?
- interval containing μ with high probability?
- decide (...) if μ is a given value?

Definition (Confidence set)

We say that $I(\mathbf{x})$ is a *confidence interval* (for a parameter θ) of level $1 - \alpha$, if

$$\mathbb{P}[I(\mathbf{x}) \ni \theta] > 1 - \alpha$$

Desiderata:

we would like the set $I(\mathbf{x})$ to be

- small (e.g. in terms of area of $I(\mathbf{x})$)
- contain θ with high probability, i.e. $\mathbb{P}[I(\mathbf{x}) \ni \theta] > 1 - \alpha$
($\alpha > 0$ and small)

so that we are able to locate θ with:

- *high precision*
- *high confidence.*

Point estimates

Given some data for which we assume some model, and a parameter that we want to estimate (e.g., μ for a Gaussian r.v.), how do we get the *best* estimate of the parameter?

What does “*best*” mean?

Maximum Likelihood Estimation (MLE) [UML, 24.1]

MLE is a statistical approach for finding the parameters that maximize the joint probability of a given dataset *assuming a specific parametric probability function*.

Note: MLE essentially assumes a *generative model* for the data

General approach:

- ① given training set $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$, assume each (\mathbf{x}_i, y_i) is i.i.d. from some probability distribution of parameters θ ; (sometimes $S = (\mathbf{x}_1, \dots, \mathbf{x}_m)$)
- ② consider $\mathbb{P}[S|\theta]$ (*likelihood* of data given parameters)
- ③ *log likelihood*: $L(S; \theta) = \log(\mathbb{P}[S|\theta])$
- ④ *maximum likelihood estimator*: $\hat{\theta} = \arg \max_{\theta} L(S; \theta)$

Point estimate of θ

Going back to the example, we can find the maximum likelihood estimator for θ .

Consider the log likelihood function (for *continuous* random variable):

$$L(\mathbf{x}, \theta) = \log p_{\mathbf{X}}(\mathbf{x}; \theta) = -\frac{m}{2} \log(2\pi\sigma^2) - \frac{1}{2} \sum_{i=1}^m \frac{(x_i - \theta)^2}{\sigma^2}$$

How to find the maximum likelihood estimator $\hat{\theta}_m$ of θ ?

Compute $\frac{d}{d\theta} L(\mathbf{X}, \theta)$ and compare it to 0.

Then the maximum likelihood estimator of θ is

$$\hat{\theta}_m := \arg \max_{\theta} L(\mathbf{x}, \theta) = \frac{1}{m} \sum_{i=1}^m x_i$$

which is the sample mean.

Confidence set for θ

The samples X_1, \dots, X_m are i.i.d. r.v. $\sim \mathcal{N}(\theta, \sigma^2)$

\Rightarrow the sample mean $\hat{\theta}$ is also normal:

$$\hat{\theta} = \frac{1}{m} \sum_{i=1}^m X_i \sim \mathcal{N}\left(\theta, \frac{\sigma^2}{m}\right)$$

it is easy to show (HOMEWORK!) that, for any given confidence $1 - \alpha$, the *smallest* confidence set for θ is symmetric and centered around the sample mean:

$$I(\mathbf{x}) = \left[\hat{\theta}_m - z_{\alpha/2} \frac{\sigma}{\sqrt{m}}, \hat{\theta}_m + z_{\alpha/2} \frac{\sigma}{\sqrt{m}} \right]$$

where $z_{\alpha/2}$ is the $1 - \alpha/2$ -percentile of a zero mean unit variance Normal distribution $\mathcal{N}(0, 1)$, i.e. such that for random variable $Y \sim \mathcal{N}(0, 1)$:

$$\mathbb{P}[Y \leq z_{\alpha/2}] = 1 - \frac{\alpha}{2} \quad \Leftrightarrow \quad \mathbb{P}[Y \geq z_{\alpha/2}] = \frac{\alpha}{2}$$

Examples: $z_{0.05/2} = 1.96$, $z_{0.01/2} \approx 2.58$

Confidence sets: graphical representation

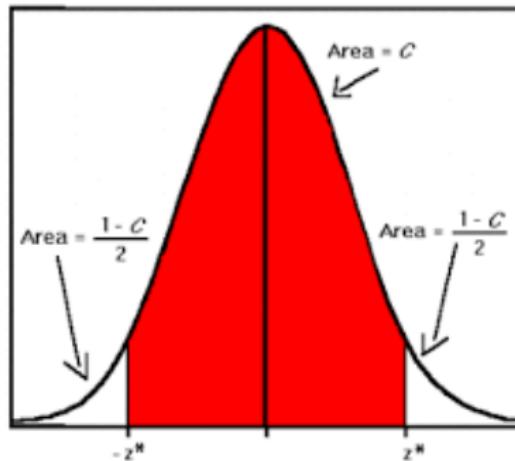


Figure: The smallest confidence set (the symmetric region around the origin) of level (or Confidence) $C = 1 - \alpha$

Machine Learning

VC-Dimension

Fabio Vandin

November 7th, 2019

PAC Learning

Question: which hypothesis classes \mathcal{H} are PAC learnable?

Up to now: if $|\mathcal{H}| < +\infty \Rightarrow \mathcal{H}$ is PAC learnable.

What about \mathcal{H} : $|\mathcal{H}| = +\infty$? Not PAC learnable?

We focus on:

- *binary classification:* $\mathcal{Y} = \{0, 1\}$
- 0-1 loss

but similar results apply to other learning tasks and losses.

Restrictions

Definition (Restriction of \mathcal{H} to \mathcal{C})

Let \mathcal{H} be a class of functions from \mathcal{X} to $\{0, 1\}$ and let $\mathcal{C} = \{c_1, \dots, c_m\} \subset \mathcal{X}$. The restriction $\mathcal{H}_{\mathcal{C}}$ of \mathcal{H} to \mathcal{C} is:

$$\mathcal{H}_{\mathcal{C}} = \{[h(c_1), \dots, h(c_m)] : h \in \mathcal{H}\}$$

where we represent each function from \mathcal{C} to $\{0, 1\}$ as a vector in $\{0, 1\}^{|\mathcal{C}|}$.

Note: $\mathcal{H}_{\mathcal{C}}$ is the set of functions from \mathcal{C} to $\{0, 1\}$ that can be derived from \mathcal{H} .

VC-dimension and Shattering

Definition (Shattering)

Given $C \subset \mathcal{X}$, \mathcal{H} shatters C if \mathcal{H}_C contains all $2^{|C|}$ functions from C to $\{0, 1\}$.

Definition (VC-dimension)

The *VC-dimension* $VCdim(\mathcal{H})$ of a hypothesis class \mathcal{H} , is the maximal size of a set $C \subset \mathcal{X}$ that can be shattered by \mathcal{H} .

Notes:

- VC = Vapnik-Chervonenkis, that introduced it in 1971
- if \mathcal{H} can shatter sets of arbitrarily large size then we say that $VCdim(\mathcal{H}) = +\infty$;
- if $|\mathcal{H}| < +\infty \Rightarrow VCdim(\mathcal{H}) \leq \log_2 |\mathcal{H}|$

Intuition: the VC-dimension measures the *complexity* of \mathcal{H} (\approx how large a dataset that is perfectly classified using the functions in \mathcal{H} can be)

Example

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
h_1	0	0	1	0	0	0	1	0	0
h_2	0	1	0	0	0	1	0	0	0
h_3	1	0	0	0	1	1	0	0	0
h_4	0	0	0	1	1	0	0	0	1
h_5	0	0	1	0	0	0	0	1	0
h_6	0	1	0	0	0	0	1	0	0
h_7	1	0	0	0	0	1	0	0	0
h_8	0	0	0	0	0	0	0	0	0

VC dimension?

$$VCdim(\mathcal{H}) = 2$$

Note

To show that $VCdim(\mathcal{H}) = d$ we need to show that:

- ① $VCdim(\mathcal{H}) \geq d$
- ② $VCdim(\mathcal{H}) \leq d$

that translates to

- ① there exists a set C of size d which is shattered by \mathcal{H}
- ② every set of size $d + 1$ is not shattered by \mathcal{H}

Question: why don't we need to consider sets of size $> d + 1$?

Example: Threshold Functions

$$\mathcal{H} = \{h_a : a \in \mathbb{R}\}$$

where $h_a : \mathbb{R} \rightarrow \{0, 1\}$ is

$$h_a(x) = \mathbb{1}[x < a] = \begin{cases} 1 & \text{if } x < a \\ 0 & \text{if } x \geq a \end{cases}$$

VC-dimension?

$$VCdim(\mathcal{H}) = 1$$

Example: Intervals

$$\mathcal{H} = \{h_{a,b} : a, b \in \mathbb{R}, a < b\}$$

where $h_{a,b} : \mathbb{R} \rightarrow \{0, 1\}$ is

$$h_{a,b}(x) = \mathbb{1}[x \in (a, b)] = \begin{cases} 1 & \text{if } a < x < b \\ 0 & \text{otherwise} \end{cases}$$

VC-dimension?

$$VCdim(\mathcal{H}) = 2$$

Example: Axis Aligned Rectangles

$$\mathcal{H} = \{h_{(a_1, a_2, b_1, b_2)} : a_1, a_2, b_1, b_2 \in \mathbb{R}, a_1 \leq a_2, b_1 \leq b_2\}$$

$$h_{(a_1, a_2, b_1, b_2)}(x_1, x_2) = \begin{cases} 1 & \text{if } a_1 \leq x_1 < a_2, b_1 \leq x_2 \leq b_2 \\ 0 & \text{otherwise} \end{cases}$$

VC-dimension?

$$VCdim(\mathcal{H}) = 4$$

Example: Convex Sets

Model set \mathcal{H} such that for $h \in \mathcal{H}$, $h : \mathbb{R}^2 \rightarrow \{0, 1\}$ with

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in S \\ 0 & \text{otherwise} \end{cases}$$

where S is a convex subset of \mathbb{R}^2

VC-dimension?

$$VCdim(\mathcal{H}) = +\infty$$

The Fundamental Theorems of Statistical Learning

Theorem

Let \mathcal{H} be a hypothesis class of functions from a domain \mathcal{X} to $\{0, 1\}$ and consider the 0-1 loss function. Assume that $VCdim(\mathcal{H}) = d < +\infty$. Then there are absolute constants C_1, C_2 such that

- \mathcal{H} has the uniform convergence property with sample complexity

$$C_1 \frac{d + \log(1/\delta)}{\varepsilon^2} \leq m_{\mathcal{H}}^{UC}(\varepsilon, \delta) \leq C_2 \frac{d + \log(1/\delta)}{\varepsilon^2}$$

- \mathcal{H} is agnostic PAC learnable with sample complexity

$$C_1 \frac{d + \log(1/\delta)}{\varepsilon^2} \leq m_{\mathcal{H}}(\varepsilon, \delta) \leq C_2 \frac{d + \log(1/\delta)}{\varepsilon^2}$$

Exercise

Consider the classification problem with $\mathcal{X} = \mathbb{R}^2, \mathbb{Y} = \{0, 1\}$.

Consider the hypothesis class $\mathcal{H} = \{h_{(\mathbf{c}, a)}, \mathbf{c} \in \mathbb{R}^2, a \in \mathbb{R}\}$ with

$$h_{(\mathbf{c}, a)}(\mathbf{x}) = \begin{cases} 1 & \text{if } \|\mathbf{x} - \mathbf{c}\| \leq a \\ 0 & \text{otherwise} \end{cases}$$

Find the VC-dimension of \mathcal{H} .

Equivalently:

Theorem

Let \mathcal{H} be an hypothesis class with VC-dimension

$VCdim(\mathcal{H}) < +\infty$. Then, with probability $\geq 1 - \delta$ (over $S \sim \mathcal{D}^m$) we have:

$$\forall h \in \mathcal{H}, L_{\mathcal{D}}(h) \leq L_S(h) + C \sqrt{\frac{VCdim(\mathcal{H}) + \log(1/\delta)}{2m}}$$

where C is a universal constant.

Note: finding $h \in \mathcal{H}$ that minimizes the upper bound (above) to $L_{\mathcal{D}}(h) \Rightarrow$ ERM rule

Theorem

Let \mathcal{H} be a class with $\text{VCdim}(\mathcal{H}) = +\infty$. Then \mathcal{H} is not PAC learnable.

Notes:

- the VC-dimension characterizes PAC learnable hypothesis classes

VC-dimension of halfspaces

Theorem

Let \mathcal{H} be the class of (homogeneous) halfspaces in \mathbb{R}^d . Then
 $VCdim(\mathcal{H}) = d$.

Proof: on the board and in the book.

Note: in this case (and the previous ones) the VC-dimension is equivalent to the number of parameters that define the model... but it is not always the case!

Example

Function of one parameter: $f_\theta(x) = \sin^2 [2^{8x} \arcsin \sqrt{\theta}]$

VC-dimension of $f_\theta(x)$ is infinite!

In fact, $f_\theta(x)$ can approximate any function $\mathbb{R} \rightarrow \mathbb{R}$ by changing the value of θ !

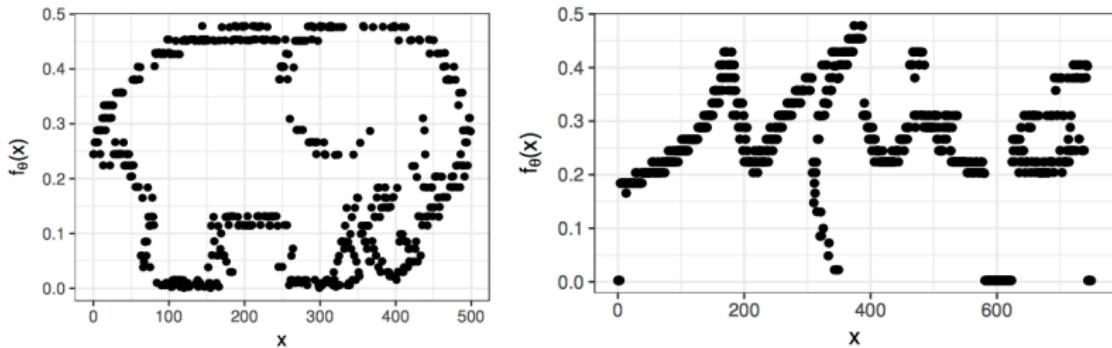


FIG. 1: A scatter plot of f_θ for $\theta = 0.2446847266734745458227540656\cdots$ plotted at integer x values, showing that a single parameter can fit an elephant (left). The same model run with parameter $\theta = 0.0024265418055000401935387620\cdots$ showing a fit of a scatter plot to Joan Miró's signature (right). Both use $r = 8$ and require hundreds to thousands of digits of precision in θ .

[“One parameter is always enough”, Piantadosi, 2018]

Bibliography

[UML] Chapter 6

[UML] Section 9.1.3

Machine Learning

Model Selection and Validation

Fabio Vandin

November 11, 2019

Model Selection

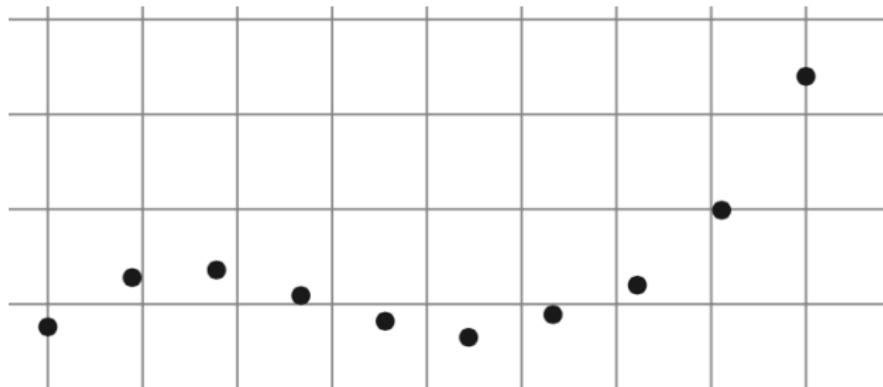
When we have to solve a machine learning task:

- there are different algorithms/classes
- algorithms have parameters

Question: how do we choose a algorithm or value of the parameters?

Example

Regression task, $\mathcal{X} = \mathbb{R}$, $\mathcal{Y} = \mathbb{R}$



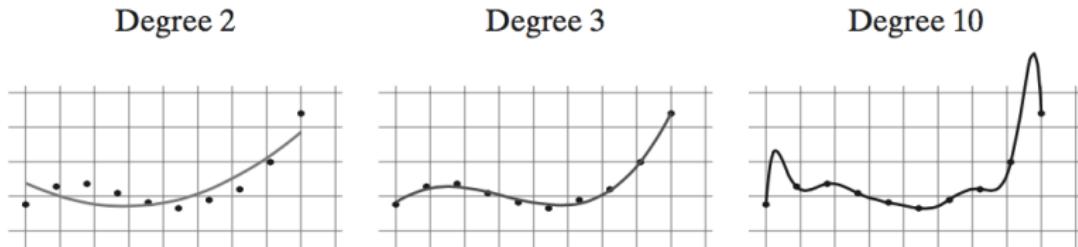
Decision: $\mathcal{H} = \text{polynomials.}$

Note: can be done using the linear regression machinery we have seen... how?

How do we pick the degree d of the polynomial?

What about considering the empirical risk of best hypothesis of various degrees (e.g., $d=2, 3, 10$)?

Best hypotheses for degree $d \in \{2, 3, 10\}$



Empirical risk is not enough!

We will see 2 approaches:

- validation
- structural risk minimization

Validation

Idea: once you pick an hypothesis, use new data to estimate its true error

Assume we have picked a predictor h (e.g., by ERM rule on a \mathcal{H}_d).

Let $V = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_{m_v}, y_{m_v})$ be a set of m_v *fresh* samples from \mathcal{D} and let $L_V(h) = \frac{1}{m_v} \sum_{i=1}^{m_v} \ell(h, (\mathbf{x}_i, y_i))$

Assume the loss function is in $[0, 1]$. Then by Hoeffding inequality we have the following.

Proposition

For every $\delta \in (0, 1)$, with probability $\geq 1 - \delta$ (over the choice of V) we have

$$|L_V(h) - L_{\mathcal{D}}(h)| \leq \sqrt{\frac{\log(2/\delta)}{2m_v}}$$

Comparison with VC-dimension bound

Assume:

- h has been picked from \mathcal{H}_d
- VC-dimension of \mathcal{H}_d is $VCdim(\mathcal{H}_d)$

Then (by fundamental theorem of learning):

$$L_{\mathcal{D}}(h) \leq L_S(h) + \sqrt{C \frac{VCdim(\mathcal{H}_d) + \log(1/\delta)}{2m}}$$

where C is a constant.

From previous proposition:

$$L_{\mathcal{D}}(h) \leq L_V(h) + \sqrt{\frac{\log(2/\delta)}{2m_v}}$$

⇒ if we pick $m_v \in \Theta(m)$, the second bound is more accurate!

Note: possible only because we use *fresh* (new) samples...

In practice:

- we have only 1 dataset
- we split it into 2 parts:
 - training set
 - *hold out* or *validation* set

A similar approach can be used for model selection...

Validation for Model Selection

Assume we have $\mathcal{H} = \cup_{i=1}^r \mathcal{H}_i$

Given a training set S , let h_i be the hypothesis obtained by ERM rule from \mathcal{H}_i using S

⇒ how do we pick a final hypothesis from $\{h_1, h_2, \dots, h_r\}$?

Validation set: $V = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_{m_v}, y_{m_v})$ be a set of *fresh* m_v samples from \mathcal{D}

⇒ choose final hypothesis (or class or value of the parameter) from $\{h_1, h_2, \dots, h_r\}$ by ERM over validation set

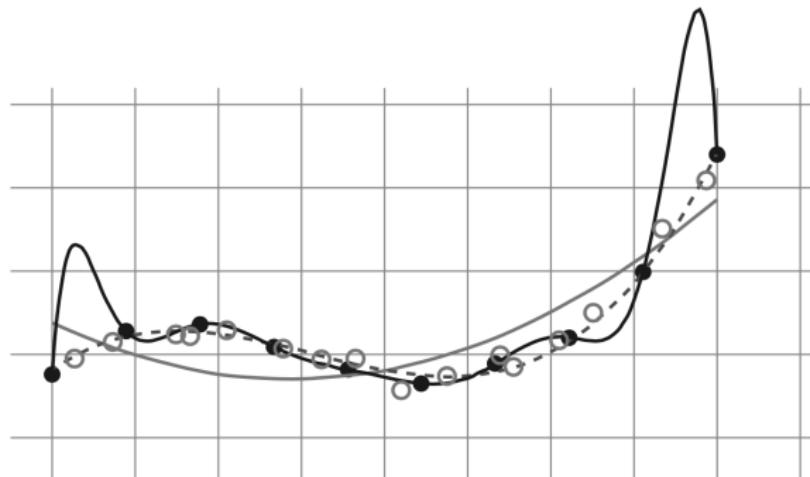
Assume loss function is in $[0, 1]$. Then we have the following.

Proposition

With probability $\geq 1 - \delta$ over the choice of V we have

$$\forall h \in \{h_1, \dots, h_r\} : |L_{\mathcal{D}}(h) - L_V(h)| \leq \sqrt{\frac{\log(2r/\delta)}{2m_v}}$$

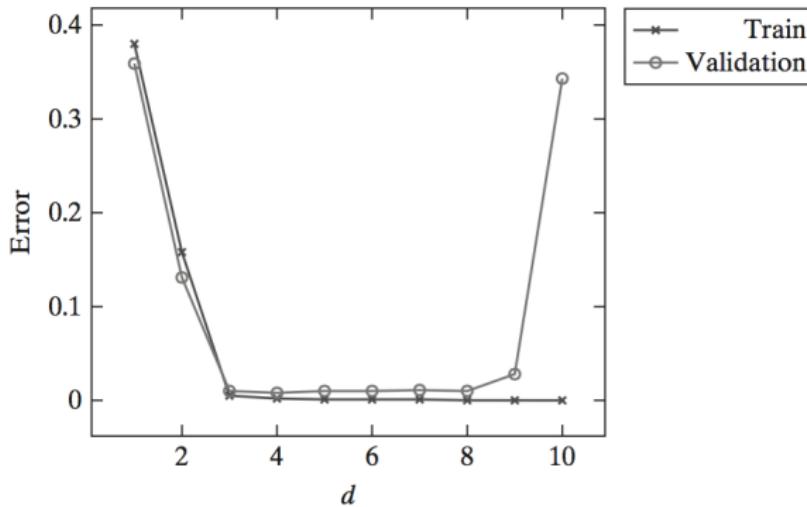
Example



Model-Selection Curve

Shows the training error and validation error as a function of the complexity of the model considered

Example



Training error decreases but validation error increases \Rightarrow overfitting

What if we have one or more parameters with values in \mathbb{R} ?

- ① Start with a rough *grid* of values
- ② Plot the corresponding model-selection curve
- ③ Based on the curve, zoom in to the correct *regime*
- ④ Restart from 1) with a finer grid

Note: the empirical risk on the validation set *is not* an estimate of the true risk, in particular if r is large (i.e., we choose among many models)!

Question: how can we estimate the true risk after model selection?

Train-Validation-Test Split

Assume we have $\mathcal{H} = \cup_{i=1}^r \mathcal{H}_i$

Idea: instead of splitting data in 2 parts, divide into 3 parts

- ① *training set*: used to learn the best model h_i from each \mathcal{H}_i
- ② *validation set*: used to pick one hypothesis h from $\{h_1, h_2, \dots, h_r\}$
- ③ *test set*: used to estimate the true risk $L_{\mathcal{D}}(h)$

⇒ the estimate from the test set has the guarantees provided by the proposition on estimate of $L_{\mathcal{D}}(h)$ for 1 class

Note:

- the test set *is not involved* in the choice of h
- if after using the test set to estimate $L_{\mathcal{D}}(h)$ we decide to choose another hypothesis (*because we have seen the estimate of $L_{\mathcal{D}}(h)$ from the test set...*)
⇒ we cannot use the test set again to estimate $L_{\mathcal{D}}(h)$!

Machine Learning

Model Selection and Validation

Fabio Vandin

November 14, 2019

Train-Validation-Test Split

Assume we have $\mathcal{H} = \cup_{i=1}^r \mathcal{H}_i$

Idea: instead of splitting data in 2 parts, divide into 3 parts

- ① *training set*: used to learn the best model h_i from each \mathcal{H}_i
- ② *validation set*: used to pick one hypothesis h from $\{h_1, h_2, \dots, h_r\}$
- ③ *test set*: used to estimate the true risk $L_{\mathcal{D}}(h)$

⇒ the estimate from the test set has the guarantees provided by the proposition on estimate of $L_{\mathcal{D}}(h)$ for 1 class

Note:

- the test set *is not involved* in the choice of h
- if after using the test set to estimate $L_{\mathcal{D}}(h)$ we decide to choose another hypothesis (*because we have seen the estimate of $L_{\mathcal{D}}(h)$ from the test set...*)
⇒ we cannot use the test set again to estimate $L_{\mathcal{D}}(h)$!

k-Fold Cross Validation

When data is not plentiful, we cannot afford to use a *fresh* validation set \Rightarrow cross validation

\Rightarrow *k*-fold cross validation:

- ① partition (training) set into *k* *folds* of size m/k
- ② for each fold:
 - train on union of other folds
 - estimate error (for learned hypothesis) from the fold
- ③ estimate of the true error = average of the estimated errors above

Leave-one-out cross validation: $k = m$

Often cross validation is used for model selection

- at the end, the final hypothesis is obtained from training on the entire training set

k -Fold Cross Validation for Model Selection

input:

training set $S = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$

set of parameter values Θ

learning algorithm A

integer k

partition S into S_1, S_2, \dots, S_k

foreach $\theta \in \Theta$

for $i = 1 \dots k$

$$h_{i,\theta} = A(S \setminus S_i; \theta)$$

$$\text{error}(\theta) = \frac{1}{k} \sum_{i=1}^k L_{S_i}(h_{i,\theta})$$

output

$$\theta^* = \operatorname{argmin}_{\theta} [\text{error}(\theta)]$$

$$h_{\theta^*} = A(S; \theta^*)$$

What if learning fails?

You use training data S and validation to pick a model h_S ...
everything looks good!
But then, on test set results are bad...

What can we do?

Need to understand where the error comes from!

Two cases:

- $L_S(h_s)$ is large
- $L_S(h_s)$ is small

$L_S(h_s)$ is large

Let $h^* \in \arg \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$.

Note that:

$$L_S(h_S) = (L_S(h_S) - L_S(h^*)) + (L_S(h^*) - L_{\mathcal{D}}(h^*)) + L_{\mathcal{D}}(h^*)$$

and

- $L_S(h_S) - L_S(h^*) < 0$
- $L_S(h^*) \approx L_{\mathcal{D}}(h^*)$

Therefore:

$L_S(h_S)$ large $\Rightarrow L_{\mathcal{D}}(h^*)$ is large \Rightarrow approximation error is large

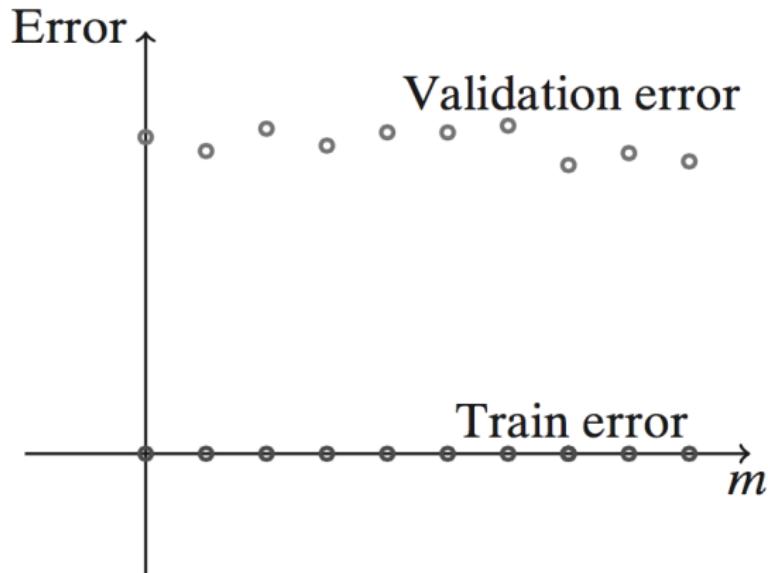
$L_S(h_S)$ is small

Need to understand if $L_{\mathcal{D}}(h^*)$ is large or not!

How?

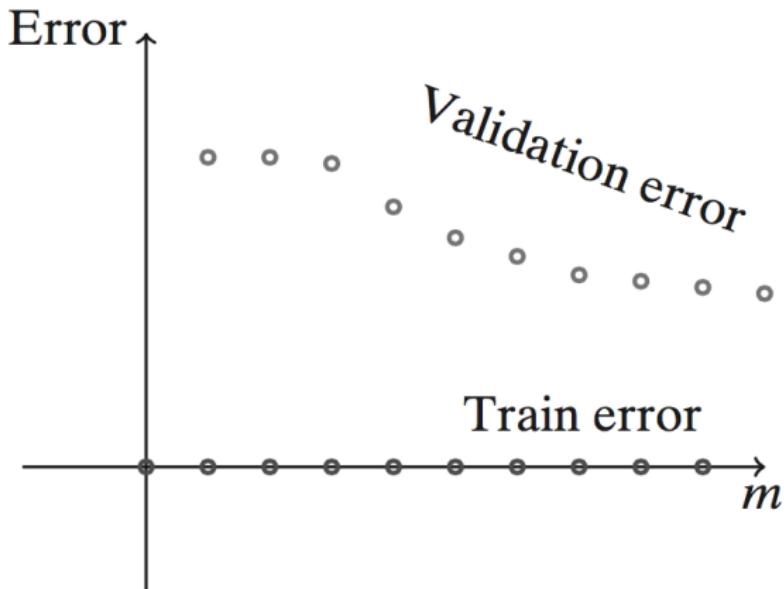
Learning curves: plot of training error and validation error when we run our algorithms on *prefixes of the data of increasing size m*

Case 1



⇒ There is no evidence that the approximation error of \mathcal{H} is good (i.e., that is small)

Case 2



⇒ \mathcal{H} may have a good approximation error but maybe we do not have enough data

Summarizing

Some potential steps to follow if learning fails:

- if you have parameters to tune, plot model-selection curve to make sure they are tuned appropriately
- if training error is excessively large consider:
 - enlarge \mathcal{H}
 - change \mathcal{H}
 - change feature representation of the data
- if training error is small, use learning curves to understand whether problem is approximation error (or estimation error)
- if approximation error seems small:
 - get more data
 - reduce complexity of \mathcal{H}
- if approximation error seems large:
 - change \mathcal{H}
 - change feature representation of the data

Model Selection using Structural Risk Minimization (SRM)

SRM can be used to tune tradeoff between bias and complexity.
Let's focus on binary classification.

Let $\mathcal{H} = \bigcup_{n \in N} \mathcal{H}_n$, where N is a finite set and each \mathcal{H}_n has VC-dimension $VCdim(\mathcal{H}_n)$

SRM

SRM follows a *bound minimization* approach

Bound: with probability $\geq 1 - \delta$, for every $n \in N$ and $h \in \mathcal{H}_n$

$$L_{\mathcal{D}}(h) \leq L_S(h) + \sqrt{C \frac{(\log(1/\delta) + \log |N| + VCdim(\mathcal{H}_n))}{2m}}$$

where C is a constant.

SRM rule: pick n and $h \in \mathcal{H}_n$ minimizing

$$L_S(h) + \sqrt{C \frac{(\log(1/\delta) + \log |N| + VCdim(\mathcal{H}_n))}{2m}}$$

Note:

- upper bound may be pessimistic
- this is a specific form of SRM, more general schemes are possible

Similar approaches

Minimum Description Length (MDL)

For $h \in \mathcal{H}$, let $|h|$ be the *length* of the representation of h (e.g., number of bits to represent h)

MDL rule: pick h minimizing

$$L_S(h) + \sqrt{C \frac{\log(1/\delta) + |h|}{2m}}$$

Can be derived from the (general) SRM paradigm.

Similar approaches (continue)

Akaike Information Criterion (AIC)

Let $d(h)$ be the number of parameters for hypothesis h :

AIC rule: pick h minimizing

$$L_S(h) + 2d(h)$$

AIC is founded in information theory.

Bibliography

[UML] Chapter 11

Machine Learning

Regularization and Feature Selection

Fabio Vandin

November 14, 2019

Learning Model

- A : learning algorithm for a machine learning task
- S : m i.i.d. pairs $z_i = (x_i, y_i)$, $i = 1, \dots, m$, with $z_i \in Z = \mathcal{X} \times \mathcal{Y}$, generated from distribution \mathcal{D}
⇒ training set available to A to produce $A(S)$;
- \mathcal{H} : the hypothesis (or model) set for A
- loss function: $\ell(h, (x, y))$, $\ell : \mathcal{H} \times Z \rightarrow \mathbb{R}^+$
- $L_S(h)$: empirical risk or training error of hypothesis $h \in \mathcal{H}$

$$L_S(h) = \frac{1}{m} \sum_{i=1}^m \ell(h, z_i)$$

- $L_{\mathcal{D}}(h)$: true risk or generalization error of hypothesis $h \in \mathcal{H}$:

$$L_{\mathcal{D}}(h) = \mathbb{E}_{z \in \mathcal{D}}[\ell(h, z)]$$

Learning Paradigms

We would like A to produce $A(S)$ such that $L_D(A(S))$ is *small*, or at least close to the smallest generalization error $L_D(h^*)$ achievable by the “best” hypothesis h^* in \mathcal{H} :

$$h^* = \arg \min_{h \in \mathcal{H}} L_D(h)$$

We have seen two *learning paradigms*:

- Empirical Risk Minimization
- Structural Risk Minimization \Rightarrow Minimum Description Length

- Empirical Risk Minimization (ERM): pick

$$A(S) \in \arg \min_{h \in \mathcal{H}} L_S(h)$$

\Rightarrow guarantees learning for PAC learnable \mathcal{H}

- Minimum Description Length: pick

$$A(S) \in \arg \min_{h \in \mathcal{H}} \left(L_S(h) + \sqrt{C \frac{|h| + \ln(1/\delta)}{2m}} \right)$$

where $|h|$ is the length of the description of h .

\Rightarrow considers trade-off between training error and complexity

We now see another learning paradigm.

Regularized Loss Minimization

Assume h is defined by a vector $\mathbf{w} = (w_1, \dots, w_d)^T \in \mathbb{R}^d$ (e.g., linear models)

Regularization function $R : \mathbb{R}^d \rightarrow \mathbb{R}$

Regularized Loss Minimization (RLM): pick h obtained as

$$\arg \min_{\mathbf{w}} (L_S(\mathbf{w}) + R(\mathbf{w}))$$

Intuition: $R(\mathbf{w})$ is a “measure of complexity” of hypothesis h defined by \mathbf{w}

⇒ regularization balances between low empirical risk and “less complex” hypotheses

We will see some of the most common regularization functions

Tikhonov regularization

Regularization function: $R(\mathbf{w}) = \lambda \|\mathbf{w}\|^2$

- $\lambda \in \mathbb{R}, \lambda > 0$
- ℓ_2 norm: $\|\mathbf{w}\|^2 = \sum_{i=1}^d w_i^2$

Therefore the *learning rule* is: pick

$$A(S) = \arg \min_{\mathbf{w}} (L_S(\mathbf{w}) + \lambda \|\mathbf{w}\|^2)$$

Intuition:

- $\|\mathbf{w}\|^2$ measures the “complexity” of hypothesis defined by \mathbf{w}
- λ regulates the tradeoff between the empirical risk ($L_S(\mathbf{w})$) or overfitting and the complexity ($\|\mathbf{w}\|^2$) of the model we pick

Ridge Regression

Linear regression with squared loss + Tikhonov regularization
⇒ *ridge regression*

Linear regression with squared loss:

- **given:** training set $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$, with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$
- **want:** \mathbf{w} which minimizes empirical risk:

$$\mathbf{w} = \arg \min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2$$

equivalently, find \mathbf{w} which minimizes the *residual sum of squares* $RSS(\mathbf{w})$

$$\mathbf{w} = \arg \min_{\mathbf{w}} RSS(\mathbf{w}) = \arg \min_{\mathbf{w}} \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2$$

Linear regression: pick

$$\mathbf{w} = \arg \min_{\mathbf{w}} RSS(\mathbf{w}) = \arg \min_{\mathbf{w}} \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2$$

Ridge regression: pick

$$\mathbf{w} = \arg \min_{\mathbf{w}} \left(\lambda \|\mathbf{w}\|^2 + \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2 \right)$$

Machine Learning

Regularization and Feature Selection

Fabio Vandin

November 18, 2019

Regularized Loss Minimization

Assume h is defined by a vector $\mathbf{w} = (w_1, \dots, w_d)^T \in \mathbb{R}^d$ (e.g., linear models)

Regularization function $R : \mathbb{R}^d \rightarrow \mathbb{R}$

Regularized Loss Minimization (RLM): pick h obtained as

$$\arg \min_{\mathbf{w}} (L_S(\mathbf{w}) + R(\mathbf{w}))$$

Intuition: $R(\mathbf{w})$ is a “measure of complexity” of hypothesis h defined by \mathbf{w}

⇒ regularization balances between low empirical risk and “less complex” hypotheses

We will see some of the most common regularization functions

Tikhonov regularization

Regularization function: $R(\mathbf{w}) = \lambda \|\mathbf{w}\|^2$

- $\lambda \in \mathbb{R}, \lambda > 0$
- ℓ_2 norm: $\|\mathbf{w}\|^2 = \sum_{i=1}^d w_i^2$

Therefore the *learning rule* is: pick

$$A(S) = \arg \min_{\mathbf{w}} (L_S(\mathbf{w}) + \lambda \|\mathbf{w}\|^2)$$

Intuition:

- $\|\mathbf{w}\|^2$ measures the “complexity” of hypothesis defined by \mathbf{w}
- λ regulates the tradeoff between the empirical risk ($L_S(\mathbf{w})$) or overfitting and the complexity ($\|\mathbf{w}\|^2$) of the model we pick

Ridge Regression

Linear regression with squared loss + Tikhonov regularization
⇒ *ridge regression*

Linear regression with squared loss:

- **given:** training set $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$, with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$
- **want:** \mathbf{w} which minimizes empirical risk:

$$\mathbf{w} = \arg \min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2$$

equivalently, find \mathbf{w} which minimizes the *residual sum of squares* $RSS(\mathbf{w})$

$$\mathbf{w} = \arg \min_{\mathbf{w}} RSS(\mathbf{w}) = \arg \min_{\mathbf{w}} \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2$$

Linear regression: pick

$$\mathbf{w} = \arg \min_{\mathbf{w}} RSS(\mathbf{w}) = \arg \min_{\mathbf{w}} \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2$$

Ridge regression: pick

$$\mathbf{w} = \arg \min_{\mathbf{w}} \left(\lambda \|\mathbf{w}\|^2 + \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2 \right)$$

RSS: Matrix Form

Let

$$\mathbf{X} = \begin{bmatrix} \cdots & \mathbf{x}_1 & \cdots \\ \cdots & \mathbf{x}_2 & \cdots \\ \cdots & \vdots & \cdots \\ \cdots & \mathbf{x}_m & \cdots \end{bmatrix}$$

\mathbf{X} : design matrix

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

\Rightarrow we have that RSS is

$$\sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

Ridge Regression: Matrix Form

Linear regression: pick

$$\arg \min_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

Ridge regression: pick

$$\arg \min_{\mathbf{w}} \left(\lambda \|\mathbf{w}\|^2 + (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \right)$$

Want to find \mathbf{w} which minimizes

$$f(\mathbf{w}) = \lambda \|\mathbf{w}\|^2 + (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}).$$

How?

Compute gradient $\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}$ of objective function w.r.t \mathbf{w} and compare it to 0.

$$\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} = 2\lambda\mathbf{w} - 2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Then we need to find \mathbf{w} such that

$$2\lambda\mathbf{w} - 2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0$$

$$2\lambda \mathbf{w} - 2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0$$

is equivalent to

$$(\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X}) \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

Note:

- $\mathbf{X}^T \mathbf{X}$ is positive semidefinite
- $\lambda \mathbf{I}$ is positive definite

$\Rightarrow \lambda \mathbf{I} + \mathbf{X}^T \mathbf{X}$ is positive definite

$\Rightarrow \lambda \mathbf{I} + \mathbf{X}^T \mathbf{X}$ is invertible

Ridge regression solution:

$$\mathbf{w} = (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Tikhonov Regularization: Some Theoretical Guarantees

What we could show: Tikhonov regularization makes the learner *stable* w.r.t. small perturbations of the training set, which in turn leads to small bounds on generalization error

Informally: an algorithm A is *stable* if a small change of the training data (i.e., its input) S will lead to a small change of its output hypothesis

Questions:

- what is a “small change of the training data”?
- what is a “small change of its output hypothesis”?

There is deep theory to answer such questions, but we will only see the main intuition.

The Fitting-Stability Tradeoff

Note that

$$\mathbb{E}_S[L_{\mathcal{D}}(A(S))] = \mathbb{E}_S[L_S(A(S))] + \mathbb{E}_S[L_{\mathcal{D}}(A(S)) - L_S(A(S))]$$

Notes:

- $\mathbb{E}_S[L_S(A(S))]$: how well A fits the training set
- $\mathbb{E}_S[L_{\mathcal{D}}(A(S)) - L_S(A(S))] = \text{overfitting} \Rightarrow$ bounded by stability of A
- in Tikhonov regularization, λ controls tradeoff between the two terms

Using the fitting-stability tradeoff decomposition and the result for convex, Lipschitz losses we can prove that knowing the properties (e.g., ρ in ρ -Lipschitz continuity, etc.) one can pick λ to guarantee that

$$\mathbb{E}_S[L_{\mathcal{D}}(A(S))] \leq \min_{\mathbf{w} \in \mathcal{H}} L_{\mathcal{D}}(\mathbf{w}) + \sqrt{\frac{c}{m}}$$

where $c > 0$ depends on the parameters of the loss function.

(A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is ρ -Lipschitz continuous if for every $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d$ we have that $\|f(\mathbf{w}_1) - f(\mathbf{w}_2)\| \leq \rho \|\mathbf{w}_1 - \mathbf{w}_2\|$.)

Question: how do we pick λ in practice?

Answer: validation!

ℓ_1 Regularization

Regularization function: $R(\mathbf{w}) = \lambda \|\mathbf{w}\|_1$

- $\lambda \in \mathbb{R}, \lambda > 0$
- ℓ_1 norm: $\|\mathbf{w}\|_1 = \sum_{i=1}^d |w_i|$

Therefore the *learning rule* is: pick

$$A(S) = \arg \min_{\mathbf{w}} (L_S(\mathbf{w}) + \lambda \|\mathbf{w}\|_1)$$

Intuition:

- $\|\mathbf{w}\|_1$ measures the “complexity” of hypothesis defined by \mathbf{w}
- λ regulates the tradeoff between the empirical risk ($L_S(\mathbf{w})$) or overfitting and the complexity ($\|\mathbf{w}\|_1$) of the model we pick

LASSO

Linear regression with squared loss + ℓ_1 regularization \Rightarrow LASSO
(least absolute shrinkage and selection operator)

LASSO: pick

$$\mathbf{w} = \arg \min_{\mathbf{w}} \lambda \|\mathbf{w}\|_1 + \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2$$

How?

Notes:

- no closed form solution!
- ℓ_1 norm is a convex function and squared loss is convex
 \Rightarrow problem can be solved efficiently! (true for every convex loss function)

LASSO and Sparse Solutions: Example

(Equivalent) one dimensional regression problem with squared loss:

$$\arg \min_{w \in \mathbb{R}} \left(\frac{1}{2m} \sum_{i=1}^m (x_i w - y_i)^2 + \lambda |w| \right)$$

Is equivalent to:

$$\arg \min_{w \in \mathbb{R}} \left(\frac{1}{2} \left(\frac{1}{m} \sum_{i=1}^m x_i^2 \right) w^2 - \left(\frac{1}{m} \sum_{i=1}^m x_i y_i \right) w + \lambda |w| \right)$$

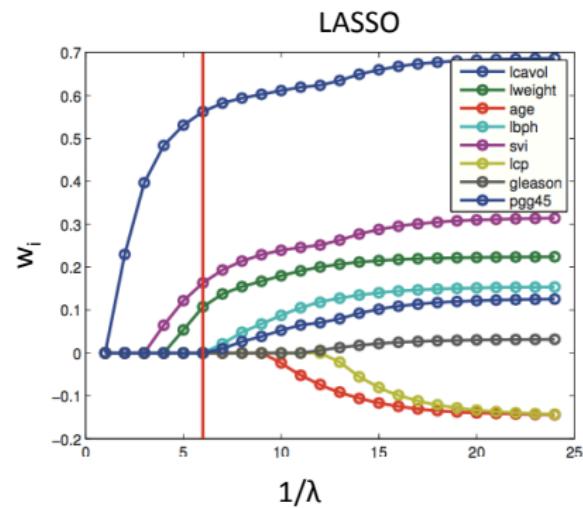
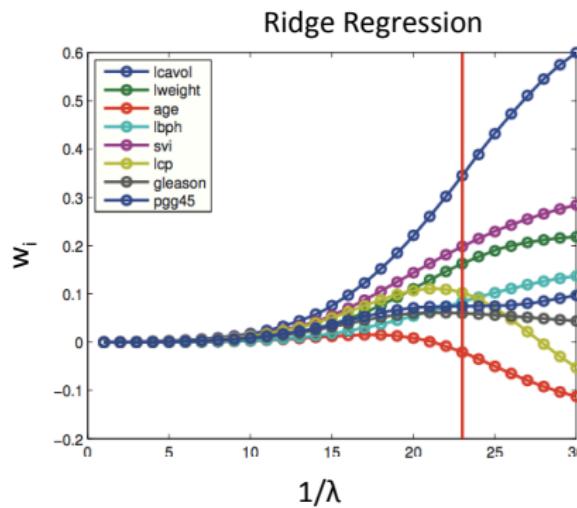
Assume for simplicity that $\frac{1}{m} \sum_{i=1}^m x_i^2 = 1$, and let
 $\sum_{i=1}^m x_i y_i = \langle \mathbf{x}, \mathbf{y} \rangle$.

Then the optimal solution is

$$w = \text{sign}(\langle \mathbf{x}, \mathbf{y} \rangle)[\langle \mathbf{x}, \mathbf{y} \rangle / m - \lambda]_+$$

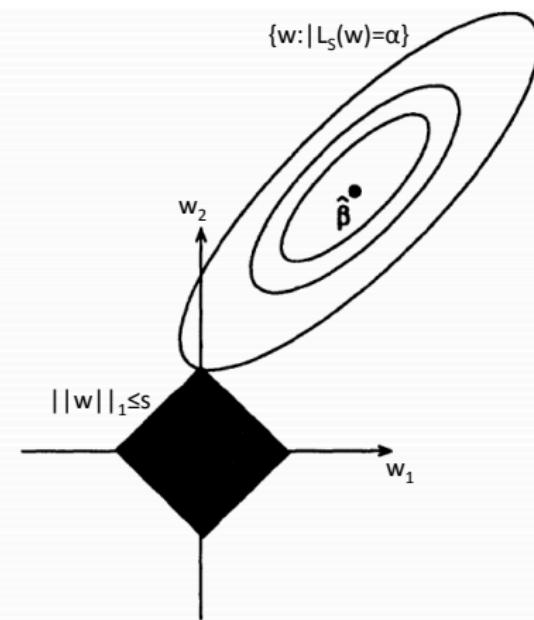
where $[a]_+ =^{(def)} \max\{a, 0\}$.

LASSO and Sparse Solution

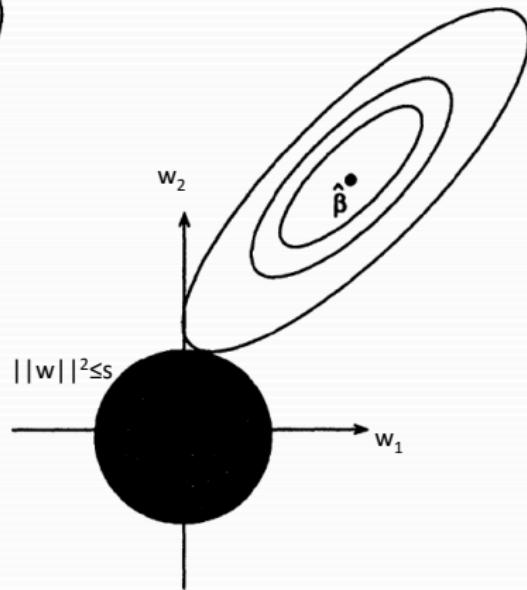


Ridge Regression vs LASSO

LASSO



RIDGE REGRESSION



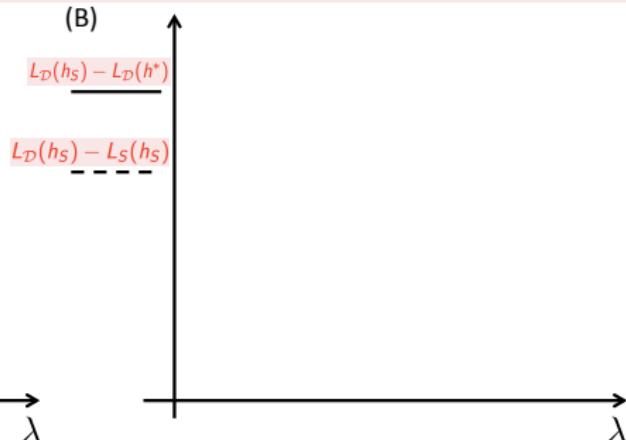
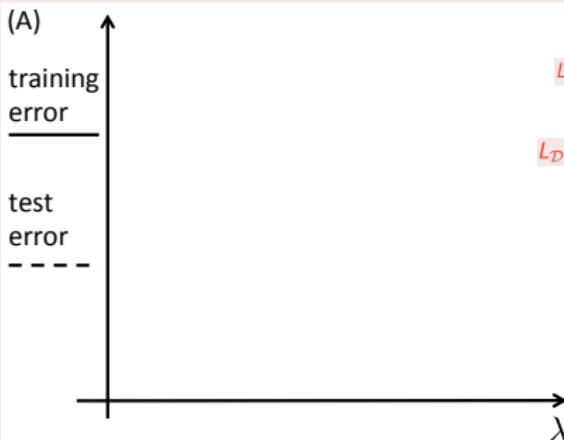
ℓ_1 regularization performs a sort of **feature selection**

Exercise 5

Consider the ridge regression problem

$\arg \min_{\mathbf{w}} \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^m (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2$. Let: h_S be the hypothesis obtained by ridge regression on with training set S ; h^* be the hypothesis of minimum generalization error among all linear models.

- (A) Draw, in the plot below, a *typical* behaviour of (i) *the training error* and (ii) *the test/generalization error* of h_S as a function of λ .
- (B) Draw, in the plot below, a *typical* behaviour of (i) $L_D(h_S) - L_D(h^*)$ and (ii) $L_D(h_S) - L_S(h_S)$ as a function of λ .



Machine Learning

Regularization and Feature Selection

Fabio Vandin

November 21, 2019

Feature Selection

In general, in machine learning one has to decide what to use as features (= input) for learning.

Even if somebody gives us a representation as a feature vector, maybe there is a “better” representation?

What is “better”?

Example

- features x_1, x_2 , output y
- $x_1 \sim Uniform(-1, 1)$
- $y = x_1^2$
- $x_2 \sim y + Uniform(-0.01, 0.01)$

If we want to predict y , which feature is better: x_1 or x_2 ?

No-free lunch...

Feature Selection: Scenario

We have a large pool of features

Goal: select a small number of features that will be used by our (final) predictor

Assume $\mathcal{X} = \mathbb{R}^d$.

Goal: learn (final) predictor using $k \ll d$ predictors

Motivation?

- prevent overfitting: less predictors \Rightarrow hypotheses of lower complexity!
- predictions can be done faster
- useful in many applications!

Feature Selection: Computational Problem

Assume that we use the Empirical Risk Minimization (ERM) procedure.

The problem of selecting k features that minimize the empirical risk can be written as:

$$\min_{\mathbf{w}} L_S(\mathbf{w}) \text{ subject to } \|\mathbf{w}\|_0 \leq k$$

where $\|\mathbf{w}\|_0 = |\{i : w_i \neq 0\}|$

How can we solve it?

Subset Selection

How do we find the solution to the problem below?

$$\min_{\mathbf{w}} L_S(\mathbf{w}) \text{ subject to } \|\mathbf{w}\|_0 \leq k$$

Note: the solution will always include k features

Let:

- $\mathcal{I} = \{1, \dots, d\}$;
- given $p = \{i_1, \dots, i_k\} \subseteq \mathcal{I}$: \mathcal{H}_p = hypotheses/models where only features $w_{i_1}, w_{i_2}, \dots, w_{i_k}$ are used

$$P^{(k)} \leftarrow \{J \subseteq \mathcal{I} : |J| = k\};$$

foreach $p \in P^{(k)}$ **do**

$$h_p \leftarrow \arg \min_{h \in \mathcal{H}_p} L_S(h);$$

$$\text{return } h^{(k)} \leftarrow \arg \min_{p \in P^{(k)}} L_S(h_p);$$

Complexity? Learn $\Theta\left(\binom{d}{k}\right) \in \Theta(d^k)$ models \Rightarrow exponential algorithm!

Can we do better?

Proposition

The optimization problem of feature selection NP-hard.

What can we do?

Heuristic solution \Rightarrow greedy algorithms

Greedy Algorithms for Feature Selection

Forward Selection: start from the empty solution, add one feature at the time, until solution has cardinality k

```
sol ← ∅;  
while |sol| < k do  
    foreach  $i \in \mathcal{I} \setminus sol$  do  
         $p \leftarrow sol \cup \{i\};$   
         $h_p \leftarrow \arg \min_{h \in \mathcal{H}_p} L_S(h);$   
    sol ← sol ∪ arg  $\min_{i \in \mathcal{I} \setminus sol} L_S(h_{sol \cup \{i\}});$   
return sol;
```

Complexity? Learns $\Theta(kd)$ models

Backward Selection: start from the solution which includes all features, remove one features at the time, until solution has cardinality k

Pseudocode: analogous to forward selection [Exercize!]

Complexity? Learns $\Theta(kd)$ models

Notes

We have used only training set to select the best hypothesis...

⇒ we may overfit!

Solution? Use validation! (or cross-validation)

Split data into training data and validation data, learn models on training, evaluate (= pick among different hypothesis models) on validation data. Algorithms are similar.

Note: now the best model (in terms of validation error) may include less than k features!

Subset Selection with Validation Data

S = training data (from data split)

V = validation data (from data split)

Using training and validation:

```
for  $\ell \leftarrow 0$  to  $k$  do
     $P^{(\ell)} \leftarrow \{J \subseteq \mathcal{I} : |J| = \ell\};$ 
    foreach  $p \in P^{(\ell)}$  do
         $h_p \leftarrow \arg \min_{h \in \mathcal{H}_p} L_S(h);$ 
     $h^{(\ell)} \leftarrow \arg \min_{p \in P^{(\ell)}} L_V(h_p);$ 
return  $\arg \min_{h \in \{h^{(0)}, h^{(1)}, \dots, h^{(k)}\}} L_V(h)$ 
```

Forward Selection with Validation Data

Using training and validation:

```
sol ← ∅;  
while |sol| < k do  
    foreach i ∈ I \ sol do  
        p ← sol ∪ {i};  
         $h_p \leftarrow \arg \min_{h \in \mathcal{H}_p} L_S(h);$   
    sol ← sol ∪ arg  $\min_{i \in I \setminus sol} L_V(h_{sol \cup \{i\}});$   
return sol;
```

Backward Selection with validation: similar [Exercize]

Similar approach for all algorithm with cross-validation [Exercize]

Bibliography [UML]

Regularization and Ridge Regression: Chapter 12

- no Section 13.3;
- Section 13.4 only up to Corollary 13.8 (excluded)

Feature Selection and LASSO: Chapter 25

- only Section 25.1.2 (introduction and “Backward Elimination”) and 25.1.3

Machine Learning

Support Vector Machines

Fabio Vandin

November 21, 2019

Classification and Margin

Consider a classification problem with two classes:

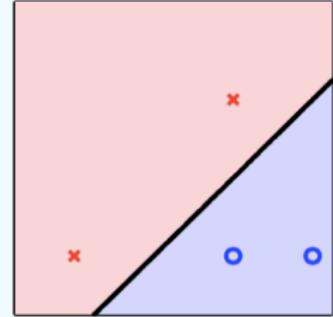
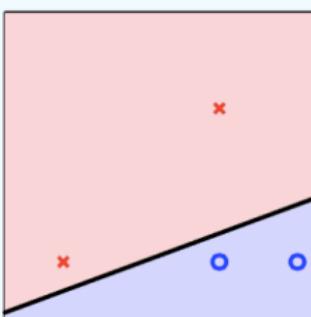
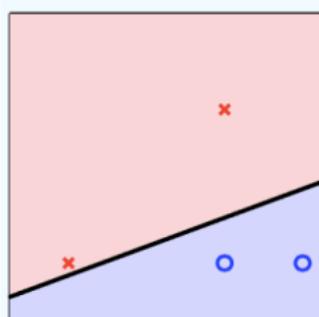
- instance set $\mathcal{X} = \mathbb{R}^d$
- label set $\mathcal{Y} = \{-1, 1\}$.

Training data: $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$

Hypothesis set \mathcal{H} = halfspaces

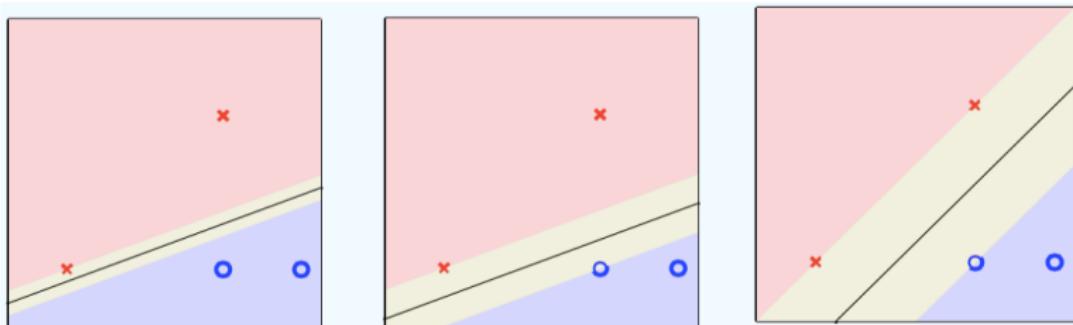
Assumption: data is linearly separable \Rightarrow there exist a halfspace that perfectly classify the training set

In general: multiple separating hyperplanes: \Rightarrow which one is the best choice?



Classification and Margin

The last one seems the best choice, since it can tolerate more “noise”.



Informally, for a given separating halfspace we define its *margin* as its minimum distance to an example in the training set S .

Intuition: best separating hyperplane is the one with largest margin.

How do we find it?

Linearly Separable Training Set

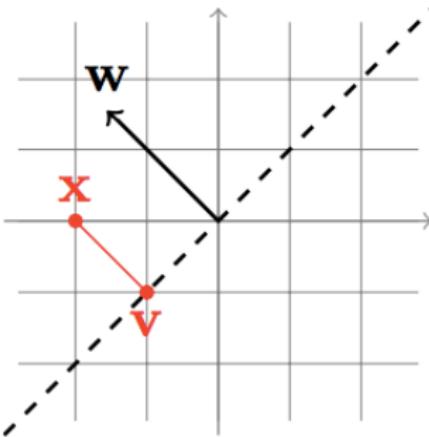
Training set $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$ is *linearly separable* if there exists a halfspace (\mathbf{w}, b) such that $y_i = \text{sign}(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)$ for all $i = 1, \dots, m$.

Equivalent to:

$$\forall i = 1, \dots, m : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0$$

Informally: *margin* of a separating hyperplane is its minimum distance to an example in the training set S

Separating Hyperplane and Margin



Given hyperplane defined by $L = \{v : \langle w, v \rangle + b = 0\}$, and given x , the distance of x to L is

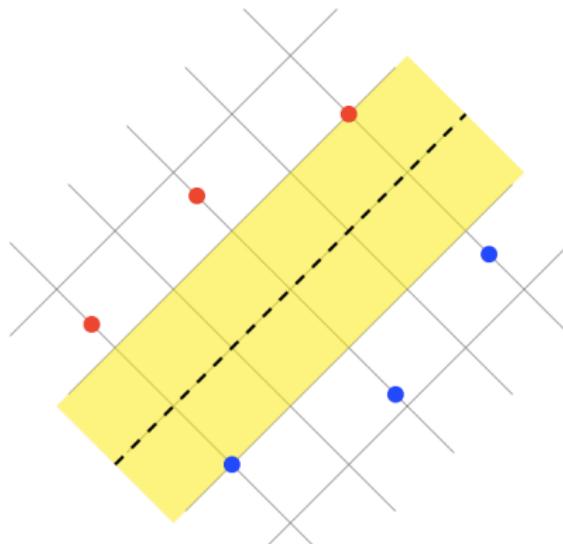
$$d(x, L) = \min\{||x - v|| : v \in L\}$$

Claim: if $\|w\| = 1$ then $d(x, L) = |\langle w, x \rangle + b|$ (Proof: Claim 15.1 [UML])

Margin and Support Vectors

The *margin* of a separating hyperplane is the distance of the closest example in training set to it. If $\|\mathbf{w}\| = 1$ the margin is:

$$\min_{i \in \{1, \dots, m\}} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b|$$



The closest examples are called *support vectors*

Support Vector Machine (SVM)

Hard-SVM: seek for the separating hyperplane with largest margin
(only for linearly separable data)

Computational problem:

$$\arg \max_{(\mathbf{w}, b): \|\mathbf{w}\|=1} \min_{i \in \{1, \dots, m\}} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b|$$

subject to $\forall i : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0$

Equivalent formulation (due to separability assumption):

$$\arg \max_{(\mathbf{w}, b): \|\mathbf{w}\|=1} \min_{i \in \{1, \dots, m\}} y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)$$

Hard-SVM: Quadratic Programming Formulation

- **input:** $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$
- **solve:**

$$(\mathbf{w}_0, b_0) = \arg \min_{(\mathbf{w}, b)} \|\mathbf{w}\|^2$$

subject to $\forall i : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$

- **output:** $\hat{\mathbf{w}} = \frac{\mathbf{w}_0}{\|\mathbf{w}_0\|}$, $\hat{b} = \frac{b_0}{\|\mathbf{w}_0\|}$

How do we get a solution? Quadratic optimization problem:
objective is convex quadratic function, constraints are linear
inequalities \Rightarrow Quadratic Programming solvers!

Proposition

The output of algorithm above is a solution to the *Equivalent Formulation* in the previous slide.

Equivalent Formulation and Support Vectors

Equivalent formulation (homogeneous halfspaces): assume first component of $\mathbf{x} \in \mathcal{X}$ is 1, then

$$\mathbf{w}_0 = \min_{\mathbf{w}} \|\mathbf{w}\|^2 \text{ subject to } \forall i : y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1$$

“Support Vectors” = vectors at minimum distance from \mathbf{w}_0

The support vectors are the only ones that matter for defining \mathbf{w}_0 !

Proposition

Let \mathbf{w}_0 be as above. Let $I = \{i : |\langle \mathbf{w}_0, \mathbf{x}_i \rangle| = 1\}$. Then there exist coefficients $\alpha_1, \dots, \alpha_m$ such that

$$\mathbf{w}_0 = \sum_{i \in I} \alpha_i \mathbf{x}_i$$

“Support vectors” = $\{\mathbf{x}_i : i \in I\}$

Note: Solving Hard-SVM is equivalent to find α_i for $i = 1, \dots, m$, and $\alpha_i \neq 0$ only for support vectors

Soft-SVM

Hard-SVM works if data is linearly separable.

What if data is not linearly separable? \Rightarrow soft-SVM

Idea: modify constraints of Hard-SVM to allow for some violation, but take into account violations into objective function

Soft-SVM Constraints

Hard-SVM constraints:

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$$

Soft-SVM constraints:

- slack variables: $\xi_1, \dots, \xi_m \geq 0 \Rightarrow$ vector ξ
- for each $i = 1, \dots, m$: $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i$
- ξ_i : how much constraint $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$ is violated

Soft-SVM minimizes combinations of

- norm of \mathbf{w}
- average of ξ_i

Tradeoff among two terms is controlled by a parameter
 $\lambda \in \mathbb{R}, \lambda > 0$

Soft-SVM: Optimization Problem

- **input:** $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$, parameter $\lambda > 0$
- **solve:**

$$\min_{\mathbf{w}, b, \xi} \left(\lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^m \xi_i \right)$$

subject to $\forall i : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$

- **output:** \mathbf{w}, b

Equivalent formulation: consider the *hinge loss*

$$\ell^{\text{hinge}}((\mathbf{w}, b), (\mathbf{x}, y)) = \max\{0, 1 - y(\langle \mathbf{w}, \mathbf{x} \rangle + b)\}$$

Given (\mathbf{w}, b) and a training S , the empirical risk $L_S^{\text{hinge}}((\mathbf{w}, b))$ is

$$L_S^{\text{hinge}}((\mathbf{w}, b)) = \frac{1}{m} \sum_{i=1}^m \ell^{\text{hinge}}((\mathbf{w}, b), (\mathbf{x}_i, y_i))$$

Soft-SVM as RLM

Soft-SVM: solve

$$\min_{\mathbf{w}, b, \xi} \left(\lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^m \xi_i \right)$$

subject to $\forall i : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$

Equivalent formulation with hinge loss:

$$\min_{\mathbf{w}, b} \left(\lambda \|\mathbf{w}\|^2 + L_S^{\text{hinge}}(\mathbf{w}, b) \right)$$

that is

$$\min_{\mathbf{w}, b} \left(\lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^m \ell^{\text{hinge}}((\mathbf{w}, b), (\mathbf{x}_i, y_i)) \right)$$

Note:

- $\lambda \|\mathbf{w}\|^2$: ℓ_2 regularization
- $L_S^{\text{hinge}}(\mathbf{w}, b)$: empirical risk for hinge loss

Machine Learning

Support Vector Machines

Fabio Vandin

November 25, 2019

Soft-SVM: Solution

We need to solve:

$$\min_{\mathbf{w}, b} \left(\lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^m \ell^{\text{hinge}}((\mathbf{w}, b), (\mathbf{x}_i, y_i)) \right)$$

where

$$\ell^{\text{hinge}}((\mathbf{w}, b), (\mathbf{x}, y)) = \max\{0, 1 - y(\langle \mathbf{w}, \mathbf{x} \rangle + b)\}$$

How?

- standard solvers for optimization problems
- **Stochastic Gradient Descent**

Gradient Descent (GD)

General approach for *minimizing* a differentiable convex function
 $f(\mathbf{w})$

Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a differentiable function

Definition

The *gradient* $\nabla f(\mathbf{w})$ of f at $\mathbf{w} = (w_1, \dots, w_d)$ is

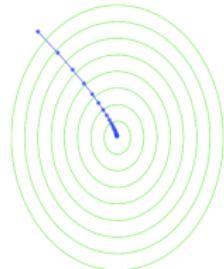
$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \dots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right)$$

Intuition: the gradient points in the direction of the greatest rate of increase of f around \mathbf{w}

Let $\eta \in \mathbb{R}, \eta > 0$ be a parameter.

GD algorithm:

```
w(0) ← 0;  
for t ← 0 to T - 1 do  
    w(t+1) ← w(t) - η ∇f(w(t));  
return w̄ = 1/T ∑t=1T w(t);
```



Notes:

- output vector could also be $w^{(T)}$ or $\arg \min_{w^{(t)} \in \{1, \dots, T\}} f(w^{(t)})$
- returning \bar{w} is useful for nondifferentiable functions (using *subgradients* instead of gradients...) and for stochastic gradient descent...
- η : *learning rate*; sometimes a time dependent $\eta^{(t)}$ is used (e.g., “move” more at the beginning than at the end)

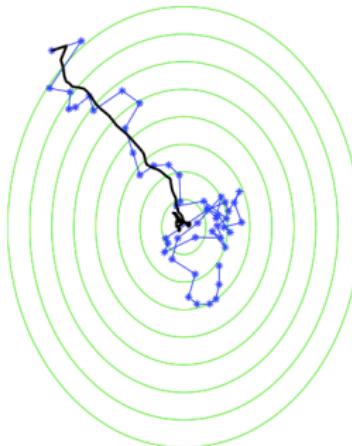
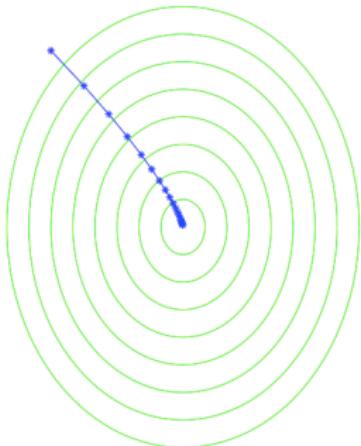
Guarantees: Let f be convex and ρ -Lipschitz continuous; let $w^* \in \arg \min_{w: \|w\| \leq B} f(w)$. Then for every $\epsilon > 0$, to find \bar{w} such that $f(\bar{w}) - f(w^*) \leq \epsilon$ it is sufficient to run the GD algorithm for $T \geq B^2 \rho^2 / \epsilon^2$ iterations.

Stochastic Gradient Descent (SGD)

Idea: instead of using exactly the gradient, we take a (random) vector with *expected value* equal to the gradient direction.

SGD algorithm:

```
w(0) ← 0;  
for t ← 0 to T - 1 do  
    choose vt at random from distribution such that E[vt|w(t)] ∈ ∇f(w(t));  
    w(t+1) ← w(t) - ηvt;  
return w̄ =  $\frac{1}{T} \sum_{t=1}^T w^{(t)}$ ;
```



SGD iterations
average of w^(t)

Guarantees: Let f be convex and ρ -Lipschitz continuous; let $\mathbf{w}^* \in \arg \min_{\mathbf{w}: \|\mathbf{w}\| \leq B} f(\mathbf{w})$. Then for every $\epsilon > 0$, to find $\bar{\mathbf{w}}$ such that $\mathbf{E}[f(\bar{\mathbf{w}})] - f(\mathbf{w}^*) \leq \epsilon$ it is sufficient to run the SGD algorithm for $T \geq B^2 \rho^2 / \epsilon^2$ iterations.

Why should we use SGD instead of GD?

Question: when do we use GD in the first place?

Answer: for example to find \mathbf{w} that minimizes $L_S(\mathbf{w})$

That is: we use GD for $f(\mathbf{w}) = L_S(\mathbf{w})$

$\Rightarrow \nabla f(\mathbf{w})$ depends on all pairs $(\mathbf{x}_i, y_i) \in S, i = 1, \dots, m$: may require long time to compute it!

What about SGD?

We need to pick \mathbf{v}_t such that $E[\mathbf{v}_t | \mathbf{w}^{(t)}] \in \nabla f(\mathbf{w}^{(t)})$: **how?**

Pick a random $(\mathbf{x}_i, y_i) \in S \Rightarrow$ pick $\mathbf{v}_t \in \nabla \ell(\mathbf{w}^{(t)}, (\mathbf{x}_i, y_i))$:

- satisfies the requirement!
- requires much less computation than GD

Analogously we can use SGD for regularized losses, etc.

SGD for Solving Soft-SVM

We want to solve

$$\min_{\mathbf{w}} \left(\frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^m \max\{0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle\} \right)$$

Note: it's standard to add a $\frac{1}{2}$ in the regularization term to simplify some computations.

SGD algorithm:

```
 $\theta^{(1)} \leftarrow \mathbf{0};$ 
for  $t \leftarrow 1$  to  $T$  do
     $\eta^{(t)} \leftarrow \frac{1}{\lambda t}; \mathbf{w}^{(t)} \leftarrow \eta^{(t)} \theta^{(t)};$ 
    choose  $i$  uniformly at random from  $\{1, \dots, m\}$ ;
    if  $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle < 1$  then  $\theta^{(t+1)} \leftarrow \theta^{(t)} + y_i \mathbf{x}_i;$ 
    else  $\theta^{(t+1)} \leftarrow \theta^{(t)};$ 
return  $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^T \mathbf{w}^{(t)};$ 
```

Exercise

Consider a binary classification problem with $\mathcal{Y} = \{-1, 1\}$. You decide to solve it with linear models and loss:

$$\ell(\mathbf{w}, (\mathbf{x}, y)) = \max \{0, -y \langle \mathbf{w}, \mathbf{x} \rangle\}.$$

Derive the corresponding SGD algorithm and compare it to the perceptron algorithm.

Machine Learning

Support Vector Machines

Fabio Vandin

November 28, 2019

Exercise

Consider a binary classification problem with $\mathcal{Y} = \{-1, 1\}$. You decide to solve it with linear models and loss:

$$\ell(\mathbf{w}, (\mathbf{x}, y)) = \max \{0, -y \langle \mathbf{w}, \mathbf{x} \rangle\}.$$

Derive the corresponding SGD algorithm and compare it to the perceptron algorithm.

Duality

We now present (Hard-)SVM in a different way which is very useful for *kernels*

We want to solve

$$\mathbf{w}_0 = \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \text{ subject to } \forall i : y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1$$

One can prove (details in the book!) that \mathbf{w} that minimizes the function above is equivalent to find α that solves the *dual problem*:

$$\max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \min_{\mathbf{w}} \left(\frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle) \right)$$

$$\max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \min_{\mathbf{w}} \left(\frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle) \right)$$

Note: once α is fixed

- the optimization with respect to \mathbf{w} is unconstrained
- the objective is differentiable

⇒ at the optimum the gradient is equal to $\mathbf{0}$:

$$\mathbf{w} - \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i = \mathbf{0} \Rightarrow \mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i$$

Replacing in the dual problem we get:

$$\max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \left(\frac{1}{2} \left\| \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \right\|^2 + \sum_{i=1}^m \alpha_i \left(1 - y_i \left\langle \sum_{j=1}^m \alpha_j y_j \mathbf{x}_j, \mathbf{x}_i \right\rangle \right) \right)$$

$$\max_{\alpha \in \mathbb{R}^m: \alpha \geq 0} \left(\frac{1}{2} \left\| \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \right\|^2 + \sum_{i=1}^m \alpha_i \left(1 - y_i \left\langle \sum_{j=1}^m \alpha_j y_j \mathbf{x}_j, \mathbf{x}_i \right\rangle \right) \right)$$

rearranging we get the problem:

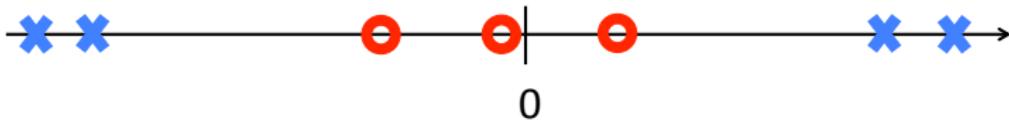
$$\max_{\alpha \in \mathbb{R}^m: \alpha \geq 0} \left(\sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_j, \mathbf{x}_i \rangle \right)$$

Note:

- solution is the vector α which defines the support vectors = $\{\mathbf{x}_i : \alpha_i \neq 0\}$
- dual problem requires only to compute inner products $\langle \mathbf{x}_j, \mathbf{x}_i \rangle$, does not need to consider \mathbf{x}_i by itself

SVM is a powerful algorithm, but still limited to linear models...
and linear models cannot always be used (directly)!

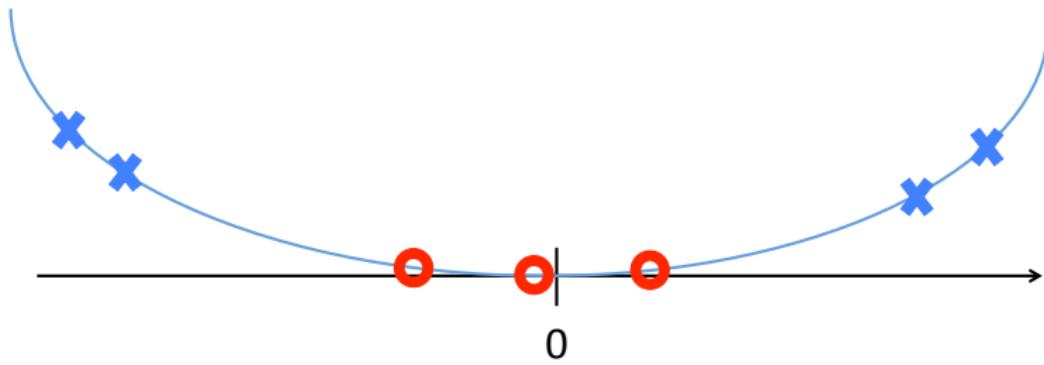
Example



We can:

- apply a nonlinear transformation $\psi()$ to each point in training set S first: $S' = ((\psi(\mathbf{x}_1), y_1), \dots, (\psi(\mathbf{x}_m), y_m))$;
- learn a linear predictor h in the transformed space using S' ;
- make prediction for a new instance \mathbf{x} as $\hat{h}(\psi(\mathbf{x}))$

Example (continued)



Kernel Trick for SVM

What if we want to apply a nonlinear transformation before using SVM?

Let $\psi()$ be the nonlinear transformation

Considering the dual formulation \Rightarrow we only need to be able to compute $\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$ for some \mathbf{x}, \mathbf{x}' .

Definition

A *kernel function* is a function of the type:

$$K_\psi(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$$

where $\psi(\mathbf{x})$ is a transformation of \mathbf{x} .

Intuition: we can think of K_ψ as specifying *similarity* between instances and of ψ as mapping the domain set \mathcal{X} into a space where these similarities are realized as inner products.

Kernel Trick for SVM(2)

$$K_{\psi}(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$$

It seems that to compute $K_{\psi}(\mathbf{x}, \mathbf{x}')$ requires to be able to compute $\psi(\mathbf{x})$...

Not always... sometimes we can compute $K_{\psi}(\mathbf{x}, \mathbf{x}')$ without computing $\psi(\mathbf{x})$!

Kernel: Example

Consider $\mathbf{x} \in \mathbb{R}^d$

$$\psi(\mathbf{x}) = (1, x_1, x_2, \dots, x_d, x_1x_1, x_1x_2, x_1x_3, \dots, x_dx_d)^T$$

The dimension of $\psi(\mathbf{x})$ is $1 + d + d^2$.

$$\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle = 1 + \sum_{i=1}^d x_i x'_i + \sum_{i=1}^d \sum_{j=1}^d x_i x_j x'_i x'_j$$

Note that

$$\sum_{i=1}^d \sum_{j=1}^d x_i x_j x'_i x'_j = \left(\sum_{i=1}^d x_i x'_i \right) \left(\sum_{j=1}^d x_j x'_j \right) = (\langle \mathbf{x}, \mathbf{x}' \rangle)^2$$

therefore

$$K_\psi(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle = 1 + \langle \mathbf{x}, \mathbf{x}' \rangle + (\langle \mathbf{x}, \mathbf{x}' \rangle)^2$$

We have:

$$\psi(\mathbf{x}) = (1, x_1, x_2, \dots, x_d, x_1x_1, x_1x_2, x_1x_3, \dots, x_dx_d)^T$$

$$K_\psi(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle = 1 + \langle \mathbf{x}, \mathbf{x}' \rangle + (\langle \mathbf{x}, \mathbf{x}' \rangle)^2$$

Observation

Computing $\psi(\mathbf{x})$ requires $\Theta(d^2)$ time; computing $K_\psi(\mathbf{x}, \mathbf{x}')$ from the last formula requires $\Theta(d)$ time

When $K_\psi(\mathbf{x}, \mathbf{x}')$ is efficiently computable, we don't need to explicitly compute $\psi(\mathbf{x})$
⇒ *kernel trick*

Some Kernels

The following are the most commonly used kernels

- linear kernel: $\psi(\mathbf{x}) = \mathbf{x}$
- sigmoid: $K(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \langle \mathbf{x}, \mathbf{x}' \rangle + \zeta)$ (for $\gamma, \zeta > 0$)
- degree- Q polynomial kernel
- Gaussian-radial basis function (RBF) kernel

Degree- Q polynomial kernel

Definition

For given constants $\gamma > 0, \zeta > 0$ and for $Q \in \mathbb{N}$, the *degree- Q polynomial kernel* is

$$K(\mathbf{x}, \mathbf{x}') = (\zeta + \gamma \langle \mathbf{x}, \mathbf{x}' \rangle)^Q$$

Example

For $Q = 2$:

$$\begin{aligned}\psi(\mathbf{x}) = & [\zeta, \sqrt{2\zeta\gamma}x_1, \sqrt{2\zeta\gamma}x_2, \dots, \sqrt{2\zeta\gamma}x_d, \\ & \gamma x_1 x_1, \gamma x_1 x_2, \dots, \gamma x_d x_d]^T\end{aligned}$$

Gaussian-RBF Kernel

Definition

For a given constant $\gamma > 0$ the *Gaussian-RBF kernel* is

$$K(\mathbf{x}, \mathbf{x}') = e^{-\gamma \|\mathbf{x} - \mathbf{x}'\|^2}$$

What is $\psi(\mathbf{x})$? Assume $\gamma = 1$ and $\mathbf{x} = x \in \mathbb{R}$ for simplicity, then

$$\begin{aligned} K(x, x') &= e^{-\|x - x'\|^2} \\ &= e^{-x^2} e^{2xx'} e^{-(x')^2} \\ &= e^{-x^2} \left(\sum_{k=0}^{+\infty} \frac{2^k (x)^k (x')^k}{k!} \right) e^{-(x')^2} \end{aligned}$$

$$\Rightarrow \psi(x) = e^{-x^2} \left(1, \sqrt{\frac{2}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \sqrt{\frac{2^3}{3!}}x^3, \dots \right)^T$$

$\Rightarrow \psi(x)$ has infinite number of dimensions!

Machine Learning

Support Vector Machines

Fabio Vandin

December 2, 2019

Kernel Trick for SVM

What if we want to apply a nonlinear transformation before using SVM?

Let $\psi()$ be the nonlinear transformation

Considering the dual formulation \Rightarrow we only need to be able to compute $\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$ for some \mathbf{x}, \mathbf{x}' .

Definition

A *kernel function* is a function of the type:

$$K_\psi(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$$

where $\psi(\mathbf{x})$ is a transformation of \mathbf{x} .

Intuition: we can think of K_ψ as specifying *similarity* between instances and of ψ as mapping the domain set \mathcal{X} into a space where these similarities are realized as inner products.

Kernel Trick for SVM(2)

$$K_{\psi}(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$$

It seems that to compute $K_{\psi}(\mathbf{x}, \mathbf{x}')$ requires to be able to compute $\psi(\mathbf{x})$...

Not always... sometimes we can compute $K_{\psi}(\mathbf{x}, \mathbf{x}')$ without computing $\psi(\mathbf{x})$!

Some Kernels

The following are the most commonly used kernels

- linear kernel: $\psi(\mathbf{x}) = \mathbf{x}$
- sigmoid: $K(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \langle \mathbf{x}, \mathbf{x}' \rangle + \zeta)$ (for $\gamma, \zeta > 0$)
- degree- Q polynomial kernel
- Gaussian-radial basis function (RBF) kernel

Degree- Q polynomial kernel

Definition

For given constants $\gamma > 0, \zeta > 0$ and for $Q \in \mathbb{N}$, the *degree- Q polynomial kernel* is

$$K(\mathbf{x}, \mathbf{x}') = (\zeta + \gamma \langle \mathbf{x}, \mathbf{x}' \rangle)^Q$$

Example

For $Q = 2$:

$$\begin{aligned}\psi(\mathbf{x}) = & [\zeta, \sqrt{2\zeta\gamma}x_1, \sqrt{2\zeta\gamma}x_2, \dots, \sqrt{2\zeta\gamma}x_d, \\ & \gamma x_1 x_1, \gamma x_1 x_2, \dots, \gamma x_d x_d]^T\end{aligned}$$

Gaussian-RBF Kernel

Definition

For a given constant $\gamma > 0$ the *Gaussian-RBF kernel* is

$$K(\mathbf{x}, \mathbf{x}') = e^{-\gamma \|\mathbf{x} - \mathbf{x}'\|^2}$$

What is $\psi(\mathbf{x})$? Assume $\gamma = 1$ and $\mathbf{x} = x \in \mathbb{R}$ for simplicity, then

$$\begin{aligned} K(x, x') &= e^{-\|x - x'\|^2} \\ &= e^{-x^2} e^{2xx'} e^{-(x')^2} \\ &= e^{-x^2} \left(\sum_{k=0}^{+\infty} \frac{2^k (x)^k (x')^k}{k!} \right) e^{-(x')^2} \end{aligned}$$

$$\Rightarrow \psi(x) = e^{-x^2} \left(1, \sqrt{\frac{2}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \sqrt{\frac{2^3}{3!}}x^3, \dots \right)^T$$

$\Rightarrow \psi(x)$ has infinite number of dimensions!

Choice of Kernel

Notes

- polynomial kernel: usually used with $Q \leq 10$
- Gaussian-RBF kernel: usually $\gamma \in [0, 1]$
- many other choices are possible!

Mercer's condition

$K(\mathbf{x}, \mathbf{x}')$ is a valid kernel function if and only if the kernel matrix

$$K = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) \dots & K(\mathbf{x}_1, \mathbf{x}_m) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) \dots & K(\mathbf{x}_2, \mathbf{x}_m) \\ \vdots & \vdots & \vdots \\ K(\mathbf{x}_m, \mathbf{x}_1) & K(\mathbf{x}_m, \mathbf{x}_2) \dots & K(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

is always symmetric positive semi-definite for any given
 $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$.

Support Vector Machines for Regression

SVMs can be also used for regression. The function to be minimized will be

$$\frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m V_\varepsilon(y_i - \langle \mathbf{x}_i, \mathbf{w} \rangle - b)$$

where

$$V_\varepsilon(r) = \begin{cases} 0 & \text{if } |r| < \varepsilon \\ |r| - \varepsilon & \text{otherwise} \end{cases}$$

One can prove that the solution has the form:

$$\mathbf{w} = \sum_{i=1}^m (\alpha_i^* - \alpha_i) \mathbf{x}_i$$

and that the final model produced in output is

$$h(\mathbf{x}) = \sum_{i=1}^m (\alpha_i^* - \alpha_i) \langle \mathbf{x}_i, \mathbf{x} \rangle + b$$

where $\alpha_i^*, \alpha_i \geq 0$ and are the solution to a suitable QP.

Definition

Support vector: \mathbf{x}_i such that $\alpha_i^* - \alpha_i \neq 0$

One can define kernels, similarly to SVM for classification.

Exercise 4

Assuming we have the following dataset ($x_i \in \mathbb{R}^2$) and by solving the SVM for classification we get the corresponding optimal dual variables:

i	x_i^T	y_i	α_i^*
1	[0.2 -1.4]	-1	0
2	[-2.1 1.7]	1	0
3	[0.9 1]	1	0.5
4	[-1 -3.1]	-1	0
5	[-0.2 -1]	-1	0.25
6	[-0.2 1.3]	1	0
7	[2.0 -1]	-1	0.25
8	[0.5 2.1]	1	0

Answer to the following:

- (A) Which are the support vectors?
- (B) Draw a schematic picture reporting the data points (approximately) and the optimal separating hyperplane, and mark the support vectors. Would it be possible, by moving only two data points, to obtain the SAME separating hyperplane with only 2 support vectors? If so, draw the modified configuration (approximately).

Bibliography [UML]

SVM: Chapter 15

- no sections 15.1.2, 15.2.1, 15.2.2, 15.2.3,

Kernels: Chapter 16

- no section 16.3

Machine Learning

Neural Networks

Fabio Vandin

December 2, 2019

Neural Networks

Informal definition: simplified models of the brain

- large number of basic computing units: *neurons*
- connected in a complex network

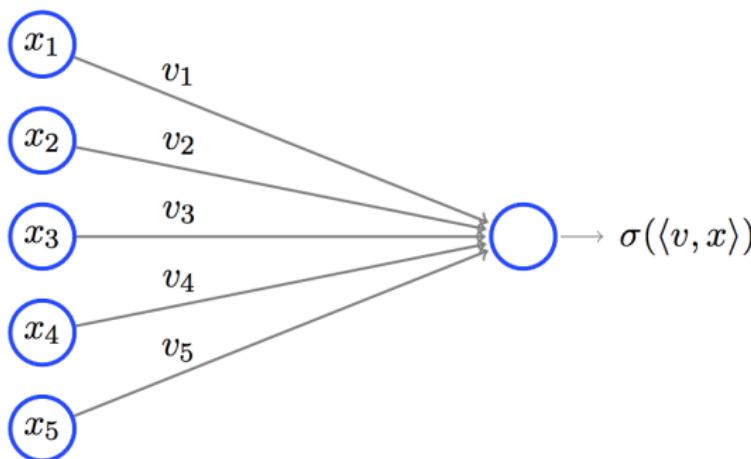


Neuron

Neuron: function $\mathbf{x} \rightarrow \sigma(\langle \mathbf{v}, \mathbf{x} \rangle)$, with $\mathbf{x} \in \mathbb{R}^d$

$\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is the *activation function*

Example: \mathbb{R}^5



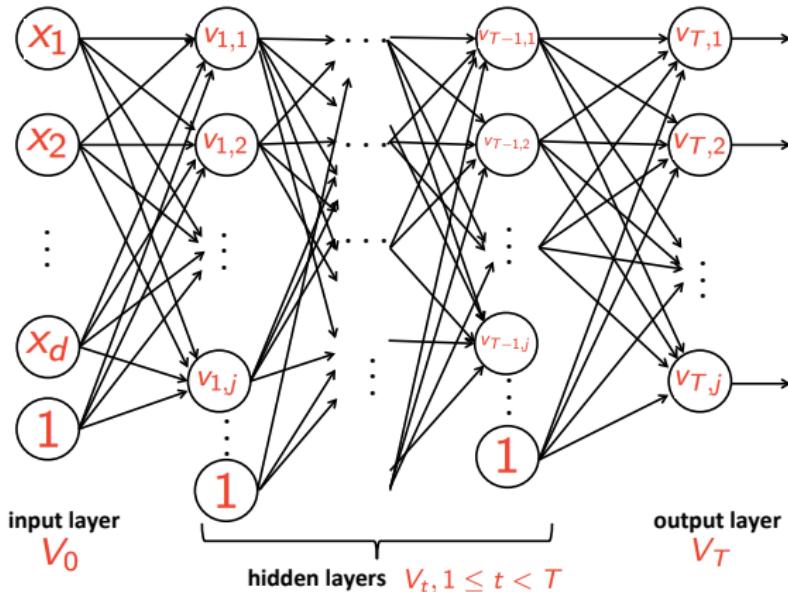
We will consider σ to be one among:

- sign function: $\sigma(a) = \text{sign}(a)$
- threshold function: $\sigma(a) = \mathbb{1}[a > 0]$
- sigmoid function: $\sigma(a) = \frac{1}{1+e^{-a}}$

Neural Network (NN)

Obtained by connecting many neurons together.

We focus on **feedforward neural networks**, defined by a directed acyclic graph $G = (V, E)$ organized in layers

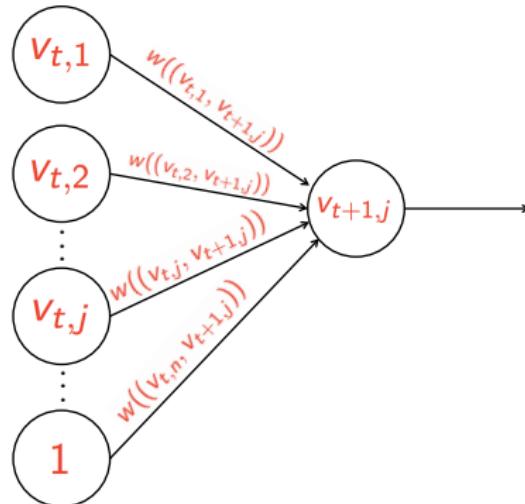


Each edge e has a *weight* $w(e)$ specified by $w : E \rightarrow \mathbb{R}$

Point of View of One Node

Consider node $v_{t+1,j}$, $0 \leq t < T$. Let

- $a_{t+1,j}(\mathbf{x})$: its *input* when \mathbf{x} is fed to the NN
- $o_{t+1,j}(\mathbf{x})$: its *output* when \mathbf{x} is fed to the NN



Then: $a_{t+1,j}(\mathbf{x}) = \sum_{r:(v_{t,r}, v_{t+1,j}) \in E} w((v_{t,r}, v_{t+1,j})) o_{t,r}(\mathbf{x})$

$$o_{t+1,j}(\mathbf{x}) = \sigma(a_{t+1,j}(\mathbf{x}))$$

Neural Network: Formalism

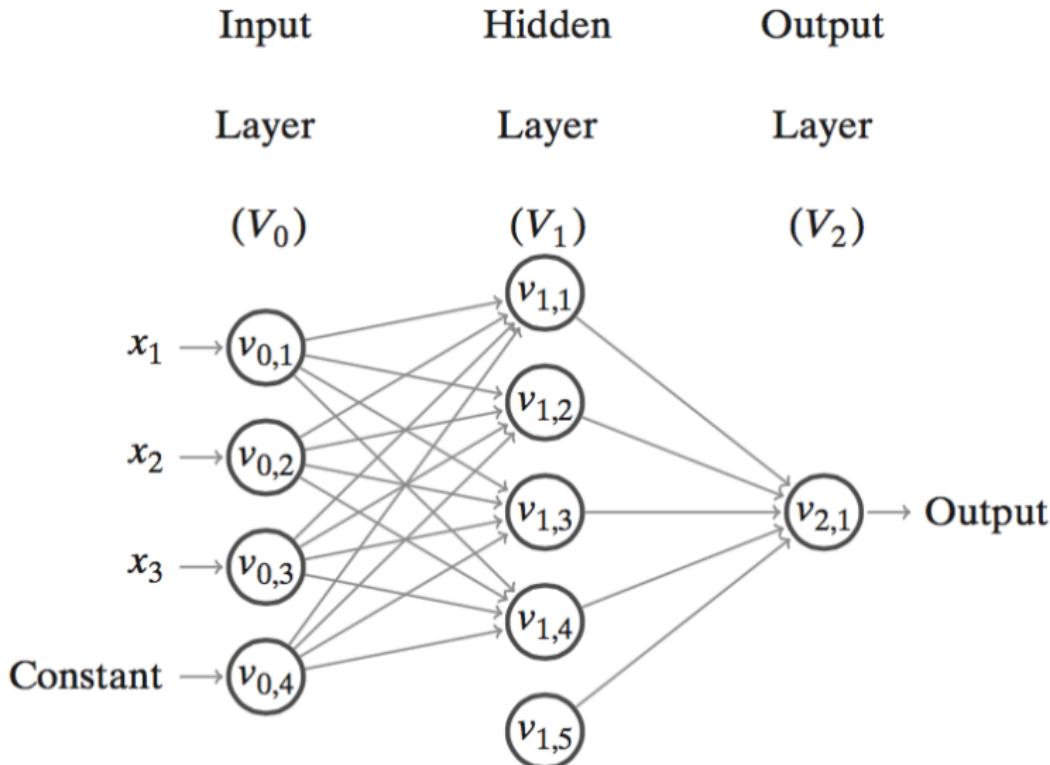
Neural network: described by directed acyclic graph $G = (V, E)$ and weight function $w : E \rightarrow \mathbb{R}$

- $V = \cup_{t=0}^T V_t$, $V_i \cap V_j = \emptyset \quad \forall i \neq j$
- $e \in E$ can only go from V_t to V_{t+1} for some t
- $V_0 = \text{input layer}$
- $V_T = \text{output layer}$
- $V_t, 0 < t < T = \text{hidden layers}$
- $T = \text{depth}$
- $|V| = \text{size of the network}$
- $\max_t |V_t| = \text{width of the network}$

Notes:

- for binary classification and regression (1 variable): output layer has 1 node
- different layers could have different activation functions (e.g., output layer)

Example



depth = 2, size = 10, width = 5

Exercize

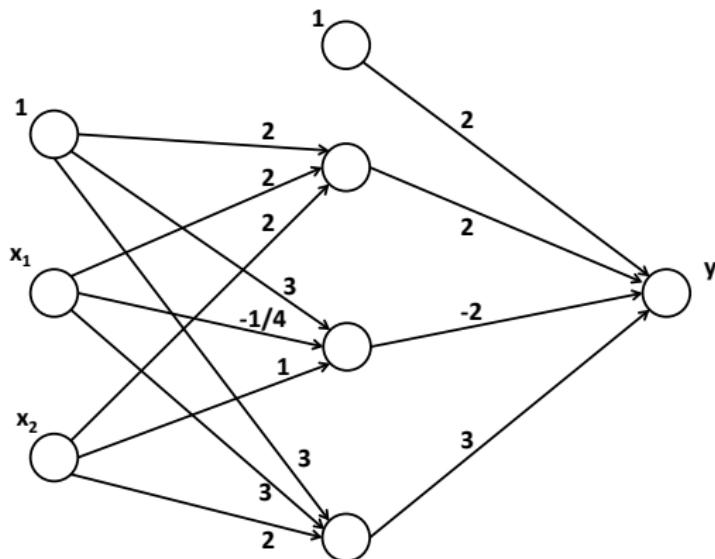
Assume that for each node the activation function $\sigma(z) : \mathbb{R} \rightarrow \mathbb{R}$ is defined as

$$\sigma(z) = \begin{cases} 1 & z \geq 1 \\ z & -1 \leq z < 1 \\ -1 & z < -1 \end{cases}$$

and consider the neural network in the next slide, compute the value of the output y when the input $x \in \mathbb{R}^2$ is

$$x = [1 \quad -3]^\top$$

Exercise (continue)



Hypothesis Set of a NN

Architecture of a NN: (V, E, σ)

Once we specify the architecture and w , we obtain a function:

$$h_{V,E,\sigma,w} : \mathbb{R}^{|V_0|-1} \rightarrow \mathbb{R}^{|V_T|}$$

The *hypothesis class* of a neural network is defined by *fixing* its architecture:

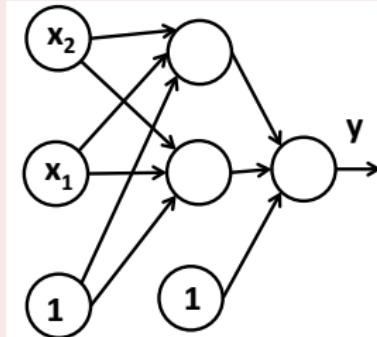
$$\mathcal{H}_{V,E,\sigma} = \{h_{V,E,\sigma,w} : w \text{ is a mapping from } E \text{ to } \mathbb{R}\}$$

Question: what type of functions can be implemented using a neural network?

Exercise (expressiveness of NNs)

Let $\mathbf{x} = [x_1, x_2] \in \{-1, 1\}^2$, and let the training data be represented by the following table:

x_1	x_2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



Consider the NN in the figure above, where the activation function for each hidden node and the output node is the *sign* function. Assume that the network's weights are constrained to be in $\{-1, 1\}$.

- ① Find network's weights so that the training error is 0.
- ② Use example above to motivate the fact that NNs are *richer* models than linear models.

Expressiveness of NN

Proposition

For every d , there exists a graph (V, E) of depth 2 such that $\mathcal{H}_{V,E,\text{sign}}$ contains all functions from $\{-1, 1\}^d$ to $\{-1, 1\}$

NN can implement every boolean function!

Unfortunately the graph (V, E) is very big...

Proposition

For every d , let $s(d)$ be the minimal integer such that there exists a graph (V, E) with $|V| = s(d)$ such that $\mathcal{H}_{V,E,\text{sign}}$ contains all functions from $\{-1, 1\}^d$ to $\{-1, 1\}$. Then $s(d)$ is an exponential function of d .

Note: similar result for $\sigma = \text{sigmoid}$

Machine Learning

Neural Networks

Fabio Vandin

December 5, 2019

Hypothesis Set of a NN

Architecture of a NN: (V, E, σ)

Once we specify the architecture and w , we obtain a function:

$$h_{V,E,\sigma,w} : \mathbb{R}^{|V_0|-1} \rightarrow \mathbb{R}^{|V_T|}$$

The *hypothesis class* of a neural network is defined by *fixing* its architecture:

$$\mathcal{H}_{V,E,\sigma} = \{h_{V,E,\sigma,w} : w \text{ is a mapping from } E \text{ to } \mathbb{R}\}$$

Question: what type of functions can be implemented using a neural network?

Expressiveness of NN

Proposition

For every d , there exists a graph (V, E) of depth 2 such that $\mathcal{H}_{V,E,\text{sign}}$ contains all functions from $\{-1, 1\}^d$ to $\{-1, 1\}$

NN can implement every boolean function!

Unfortunately the graph (V, E) is very big...

Proposition

For every d , let $s(d)$ be the minimal integer such that there exists a graph (V, E) with $|V| = s(d)$ such that $\mathcal{H}_{V,E,\text{sign}}$ contains all functions from $\{-1, 1\}^d$ to $\{-1, 1\}$. Then $s(d)$ is an exponential function of d .

Note: similar result for $\sigma = \text{sigmoid}$

Proposition

For every fixed $\varepsilon > 0$ and every Lipschitz function $f : [-1, 1]^d \rightarrow [-1, 1]$ it is possible to construct a neural network such that for every input $x \in [-1, 1]^d$ the output of the neural network is in $[f(x) - \varepsilon, f(x) + \varepsilon]$.

Note: first result proved by Cybenko (1989) for sigmoid activation function, requires only 1 hidden layer!

NNs are **universal approximators!**

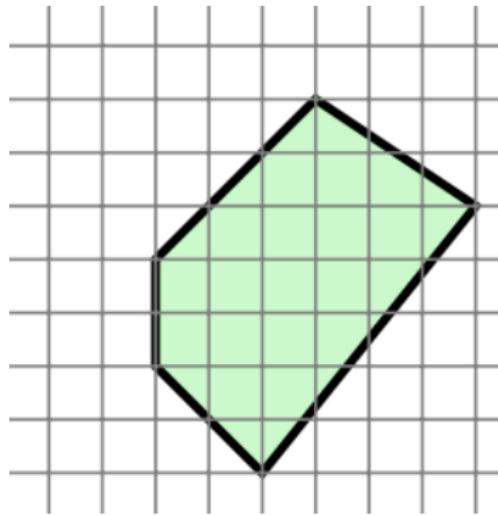
But again...

Proposition

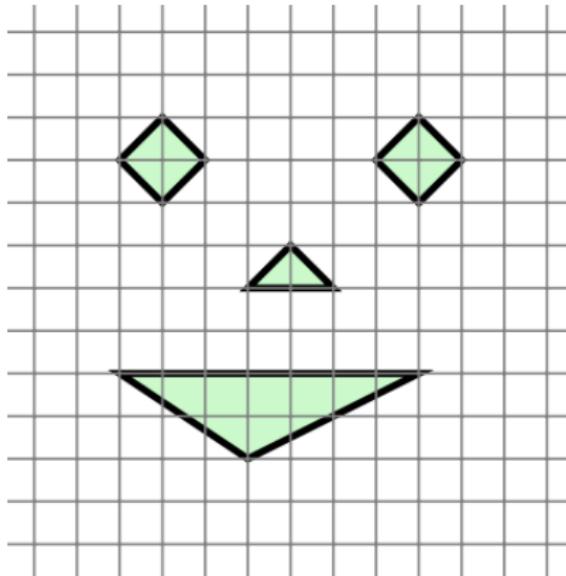
Fix some $\varepsilon \in (0, 1)$. For every d , let $s(d)$ be the minimal integer such that there exists a graph (V, E) with $|V| = s(d)$ such that $\mathcal{H}_{V, E, \sigma}$, with $\sigma = \text{sigmoid}$, can approximate, with precision ε , every 1-Lipschitz function $f : [-1, 1]^d \rightarrow [-1, 1]$. Then $s(d)$ is exponential in d .

Geometric Intuition

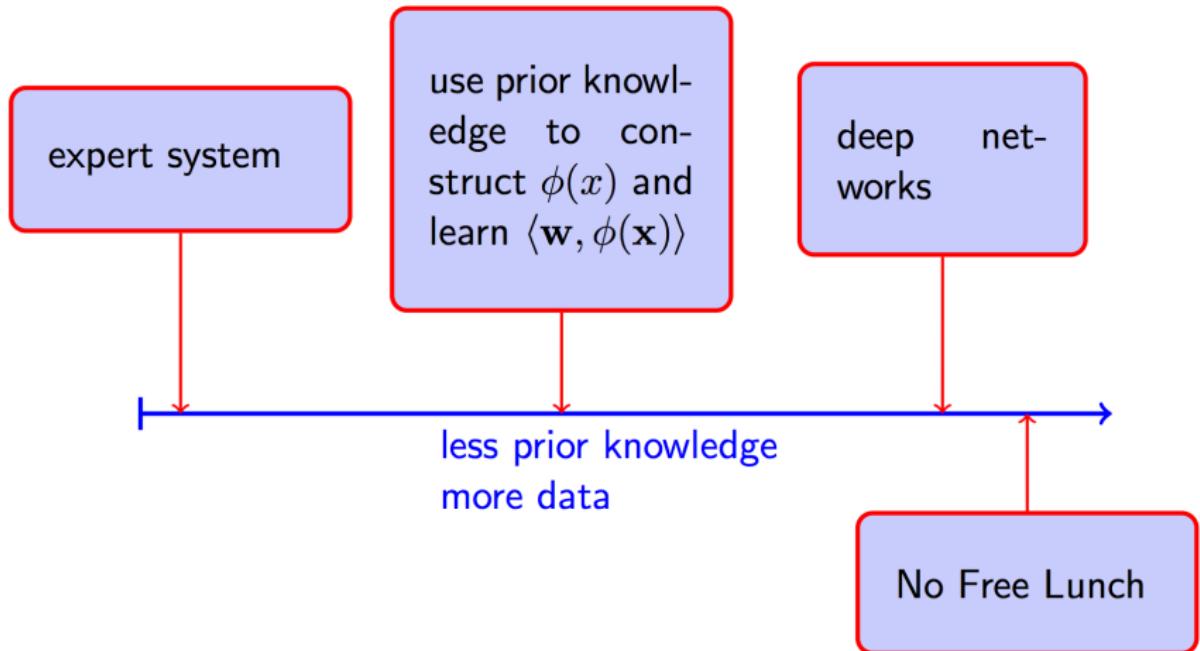
2 layer neural networks (with $\sigma = \text{sign}$) can express intersection of halfspaces



3 layer neural networks (with $\sigma = \text{sign}$) can express unions of intersection of halfspaces



An Extremely Powerful Hypothesis Class...



Sample Complexity of NNs

How much data is needed to learn with NNs?

Proposition

The VC dimension of $\mathcal{H}_{V,E,\text{sign}} = O(|E| \log |E|)$

Different σ ?

Proposition

Let σ be the sigmoid function. The VC dimension of $\mathcal{H}_{V,E,\sigma}$ is:

- $\Omega(|E|^2)$
- $O(|V|^2|E|^2)$

⇒ large NNs require a lot of data!

Question: assume we have a lot of data, can we find the best hypothesis?

Runtime of Learning NNs

Informally: applying the ERM rule with respect to $\mathcal{H}_{V,E,\text{sign}}$ is *computationally difficult*, even for small NN...

Proposition

Let $k \geq 3$. For every d , let (V, E) be a layered graph with d input nodes, $k+1$ nodes at the (only) hidden layer, where one of them is the constant neuron, and a single output node. Then, it is NP-hard to implement the ERM rule with respect to $\mathcal{H}_{V,E,\text{sign}}$.

Well maybe the above is only for very specific cases...

- instead of ERM rule, find h close to ERM? **Computationally infeasible!** (probably)
- other activation functions (e.g., sigmoid)? **Computationally infeasible!** (probably)
- smart embedding in larger network? **Computationally infeasible!** (probably)

So? *Heuristic* for training NNs \Rightarrow SGD algorithm and its improved versions are used: gives good results in practice!

Machine Learning

Neural Networks

Fabio Vandin

December 9, 2019

Matrix Notation

Consider layer t , $0 < t < T$:

- let $d^{(t)} + 1$ the number of nodes:
 - constant node 1
 - values of nodes for (hidden) variables: $v_{t,1}, \dots, v_{t,d^{(t)}}$
- arc from $v_{t-1,i}$ to $v_{t,j}$ has weight $w_{ij}^{(t)}$

Let

$$\mathbf{v}^{(t)} = \left(1, v_{t,1}, \dots, v_{t,d^{(t)}} \right)^T$$

$$\mathbf{w}_j^{(t)} = \left(w_{0j}^{(t)}, w_{1j}^{(t)}, \dots, w_{d^{(t-1)}j}^{(t)} \right)^T$$

Then

$$v_{t,j} = \sigma \left(\langle \mathbf{w}_j^{(t)}, \mathbf{v}^{(t-1)} \rangle \right)$$

Note:

$$\mathbf{v}^{(t)} = \begin{bmatrix} 1 \\ v_{t,1} \\ \vdots \\ v_{t,d^{(t)}} \end{bmatrix} = \begin{bmatrix} 1 \\ \sigma(\langle \mathbf{w}_1^{(t)}, \mathbf{v}^{(t-1)} \rangle) \\ \vdots \\ \sigma(\langle \mathbf{w}_{d^{(t)}}^{(t)}, \mathbf{v}^{(t-1)} \rangle) \end{bmatrix}$$

Let

$$a_{t,j} := \langle \mathbf{w}_j^{(t)}, \mathbf{v}^{(t-1)} \rangle$$

and

$$\mathbf{a}^{(t)} = \begin{bmatrix} a_{t,1} \\ \vdots \\ a_{t,d^{(t)}} \end{bmatrix} \quad \sigma(\mathbf{a}^{(t)}) = \begin{bmatrix} \sigma(a_{t,1}) \\ \vdots \\ \sigma(a_{t,d^{(t)}}) \end{bmatrix}$$

Then

$$\mathbf{v}^{(t)} = \begin{bmatrix} 1 \\ \sigma(\mathbf{a}^{(t)}) \end{bmatrix}$$

Let

$$\mathbf{w}^{(t)} = \begin{bmatrix} w_{01}^{(t)} & w_{02}^{(t)} & \dots & w_{0d^{(t)}}^{(t)} \\ w_{11}^{(t)} & w_{12}^{(t)} & \dots & w_{1d^{(t)}}^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d^{(t-1)1}}^{(t)} & w_{d^{(t-1)2}}^{(t)} & \dots & w_{d^{(t-1)d(t)}}^{(t)} \end{bmatrix}$$

($\mathbf{w}^{(t)}$ describes the weights of edges from layer $t - 1$ to layer t)

Then

$$\mathbf{a}^{(t)} = (\mathbf{w}^{(t)})^T \mathbf{v}^{(t-1)}$$

Using Matrix Notation Warm-Up: Forward Propagation Algorithm

Input: $\mathbf{x} = (x_1, \dots, x_d)^T$; NN with 1 output node

Output: prediction y of NN;

$\mathbf{v}^{(0)} \leftarrow (1, x_1, \dots, x_d)^T$;

for $t \leftarrow 1$ **to** T **do**

$\mathbf{a}^{(t)} \leftarrow (\mathbf{w}^{(t)})^T \mathbf{v}^{(t-1)}$;

$\mathbf{v}^{(t)} \leftarrow \left(1, \sigma(\mathbf{a}^{(t)})^T\right)^T$;

$y \leftarrow \sigma(\mathbf{a}^{(T)})$;

return y ;

Learning NN parameters

How do we compute the weights $w_{ij}^{(t)}$?

ERM: given training data $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$ pick $w_{ij}^{(t)}, \forall i, j, t$
(defining a specific model h) minimizing the training error:

$$L_S(h) = \frac{1}{m} \sum_{i=1}^m \ell(h, (\mathbf{x}_i, y_i))$$

How?

Not easy!

Learning NN parameters (2)

We use SGD seeing $L_S(h)$ as a function of $\mathbf{w}^{(t)}$, $\forall 1 \leq t \leq T$:

SGD Update rule:

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla L_S(\mathbf{w}^{(t)})$$

where $\nabla L_S(\mathbf{w}^{(t)})$ is the gradient of L_S (and η is the learning parameter). To compute it we need $\forall t, 1 \leq t \leq T$:

$$\frac{\partial L_S}{\partial \mathbf{w}^{(t)}} = \frac{\partial}{\partial \mathbf{w}^{(t)}} \left(\frac{1}{m} \sum_{i=1}^m \ell(h, (\mathbf{x}_i, y_i)) \right) = \frac{1}{m} \sum_{i=1}^m \frac{\partial \ell(h, (\mathbf{x}_i, y_i))}{\partial \mathbf{w}^{(t)}}$$

\Rightarrow need $\frac{\partial \ell}{\partial \mathbf{w}^{(t)}}$

Learning NN parameters (3)

Definition: Sensitivity vector for layer t

$$\delta^{(t)} = \frac{\partial L}{\partial \mathbf{a}^{(t)}} = \begin{bmatrix} \frac{\partial L}{\partial a_{t,1}} \\ \vdots \\ \frac{\partial L}{\partial a_{t,d^{(t)}}} \end{bmatrix} = \begin{bmatrix} \delta_1^{(t)} \\ \vdots \\ \delta_{d^{(t)}}^{(t)} \end{bmatrix}$$

$\delta^{(t)}$ quantifies how the training error changes with $\mathbf{a}^{(t)}$ (the inputs to the t layer - before the nonlinear transformation)

Learning NN parameters (4)

Consider a weight $w_{ij}^{(t)}$: a change in $w_{ij}^{(t)}$ changes only $a_{t,j}$ therefore by chain rule we have

$$\begin{aligned}\frac{\partial L}{\partial w_{ij}^{(t)}} &= \frac{\partial L}{\partial a_{t,j}} \cdot \frac{\partial a_{t,j}}{\partial w_{ij}^{(t)}} \\ &= \delta_j^{(t)} \cdot \frac{\partial}{\partial w_{ij}^{(t)}} \left(\sum_{k=0}^{d^{(t-1)}} w_{kj}^{(t)} v_{t-1,k} \right) \\ &= \delta_j^{(t)} \cdot v_{t-1,i}\end{aligned}$$

Therefore to compute the gradient we only need $\delta^{(t)} = \frac{\partial L}{\partial \mathbf{a}^{(t)}}$ $\forall t$. How can we compute it?

Learning NN parameters (5)

Since L depends from $a_{t,j}$ only through $v_{t,j}$, then from chain rule:

$$\begin{aligned}\delta_j^{(t)} &= \frac{\partial L}{\partial a_{t,j}} \\ &= \frac{\partial L}{\partial v_{t,j}} \cdot \frac{\partial v_{t,j}}{\partial a_{t,j}} \\ &= \frac{\partial L}{\partial v_{t,j}} \cdot \sigma'(a_{t,j})\end{aligned}$$

(the last equality derives from the definition of $v_{t,j}$)

Learning NN parameters (6)

Consider $\frac{\partial L}{\partial v_{t,j}}$: we need to understand how loss L changes due to changes in $v_{t,j}$

- change in $v^{(t)}$ affects only $a^{(t+1)}$ (and then L)
- changes in $v_{t,j}$ can affect every $a_{t+1,k}$

⇒ sum chain rule contributions

Then

$$\begin{aligned}\frac{\partial L}{\partial v_{t,j}} &= \sum_{k=1}^{d^{(t+1)}} \frac{\partial a_{t+1,k}}{\partial v_{t,j}} \cdot \frac{\partial L}{\partial a_{t+1,k}} \\ &= \sum_{k=1}^{d^{(t+1)}} w_{jk}^{(t+1)} \cdot \delta_k^{(t+1)}\end{aligned}$$

Learning NN parameters (7)

Putting everything together:

$$\delta_j^{(t)} = \sigma'(a_{t,j}) \cdot \sum_{k=1}^{d^{(t+1)}} w_{jk}^{(t+1)} \delta_k^{(t+1)}$$

Notes:

- $\sigma'(a_{t,j})$ depends on the function σ chosen
- To compute $\delta_j^{(t)}$ need $\delta_k^{(t+1)}$, $1 \leq k \leq d^{(t+1)}$
 \Rightarrow backpropagation algorithm
- To start: need $\delta^{(L)} = \frac{\partial L}{\partial a^{(L)}}$ (sensitivity of final layer): depends on the loss L used

Algorithm to compute sensitivities $\delta^{(t)}$, $\forall t$, for a given data point (\mathbf{x}_i, y_i) .

Input: data point (\mathbf{x}_i, y_i) , NN (with weights $w_{ij}^{(t)}$, for $1 \leq t \leq T$)

Output: $\delta^{(t)}$ for $t = 1, \dots, T$

compute $\mathbf{a}^{(t)}$ and $\mathbf{v}^{(t)}$ for $t = 1, \dots, T$;

$$\delta^{(T)} \leftarrow \frac{\partial L}{\partial a^{(T)}};$$

for $t = T - 1$ **downto** 1 **do**

$$\quad \delta_j^{(t)} \leftarrow \sigma'(a_{t,j}) \cdot \sum_{k=1}^{d^{(\ell+1)}} w_{jk}^{(t+1)} \delta_k^{(t+1)} \text{ for all } j = 1, \dots, d^{(t)};$$

return $\delta^{(1)}, \dots, \delta^{(T)}$;

Backpropagation Algorithm

This is the final backpropagation algorithm, based on SGD, to train a NN

Input: training data $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$, NN (no weights $w_{ij}^{(t)}$)

Output: NN with weights $w_{ij}^{(t)}$

initialize $w_{ij}^{(t)}$ for all i, j, t ;

```
for  $s \leftarrow 0, 1, 2, \dots$  do /* until convergence */  
    pick  $(\mathbf{x}_k, y_k)$  at random from training data;  
    /* forward propagation */  
    compute  $v_{t,j}$  for all  $j, t$  from  $(\mathbf{x}_k, y_k)$ ;  
    /* backward propagation */  
    compute  $\delta_j^{(t)}$  for all  $j, t$  from  $(\mathbf{x}_k, y_k)$ ;  
     $w_{ij}^{(t)} \leftarrow w_{ij}^{(t)} - \eta v_{t-1,i} \delta_j^{(t)}$  for all  $i, j, t$ ; /* update  
    weights */  
    if converged then return  $w_{ij}^{(t)}$  for all  $i, j, t$ ;
```

Notes on Backpropagation Algorithm

- preprocessing: all inputs are normalized and centered
- initialization of $w_{ij}^{(t)}$?
Random values around 0 - regime where model is \approx linear
 - $w_{ij}^{(t)} \sim U(-0.7, 0.7)$ (uniform distribution)
 - $w_{ij}^{(t)} \sim N(0, \sigma^2)$ with small σ^2
 - if all weights set to 0 \Rightarrow all neurons get the same weights
- when to stop?
Usually combination of:
 - “small” (training) error;
 - “small” marginal improvement in error;
 - upper bound on number of iterations
- $L_S(h)$ usually has multiple local minima
 \Rightarrow run stochastic gradient descent for different (random) initial weights

Regularized NN

Instead of training a NN by minimizing $L_S(h)$, find h that minimizes:

$$L_S(h) + \frac{\lambda}{2} \sum_{i,j,t} (w_{ij}^{(t)})^2$$

where $\lambda = \text{regularization parameter}$

How do we find h ? SGD or improved algorithms.

Note: for layer t , gradient is $\nabla(L_S(h)) + \lambda w^{(t)}$

This is called *squared weight decay regularizer*

Other regularizations are possible.

Machine Learning

NN and Deep Learning

Fabio Vandin

December 12, 2019

Neural Networks

Informal definition: simplified models of the brain

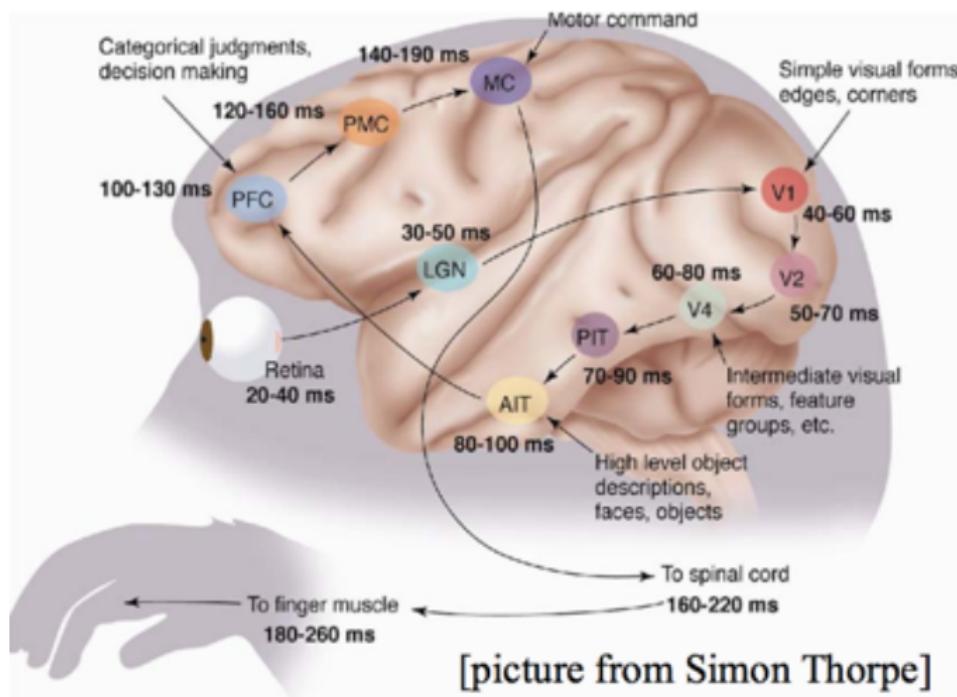
- large number of basic computing units: *neurons*
- connected in a complex network



Deep Learning

Deep Learning = learning hierarchical representations

Biological inspiration e.g. *our visual cortex is hierarchical*



Hierarchical Representations and ML

Traditional Machine Learning: Fixed/Handcrafted Feature Extractor



Advanced Machine Learning: Unsupervised mid-level features



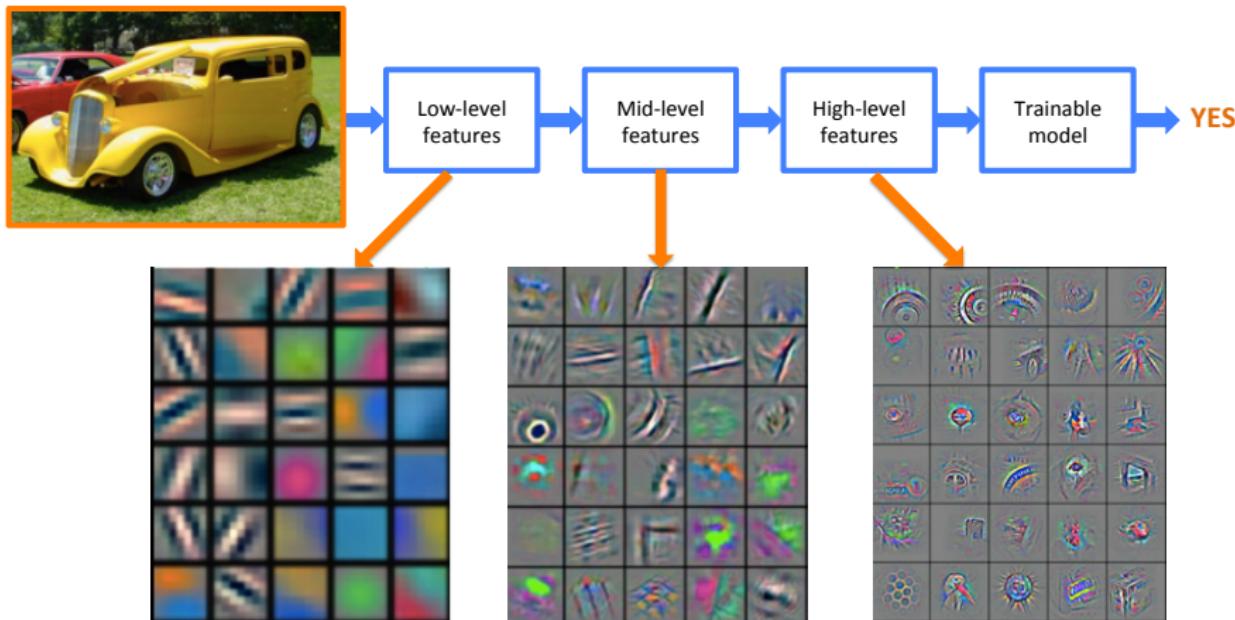
Deep Learning: Representations are hierarchical and trained



Deep Learning \Rightarrow End-to-end Learning

Hierarchical Representations and ML (continue)

Example: object recognition (object = car)



Neural Networks: Historical Remarks

- 1940s-70s:
 - Inspired by learning/modeling the brain (Pitts, Hebb, and others)
 - Perceptron (Rosenblatt), Multilayer perceptron (Minsky and Papert)
 - Backpropagation (Werbos, 1975)
- 1980s-early 1990s:
 - Practical Backpropagation (Rumelhart, Hinton et al., 1986) and SGD (Bottou)
 - Initial empirical success
- 1990s-2000s:
 - Lost favor to implicit linear methods: SVM, Boosting
- 2006-:
 - Regain popularity because of unsupervised pre-training (Hinton, Bengio, LeCun, Ng, and others)
 - Computational advances and several new tricks allow training *huge* networks. Empirical success leads to renewed interest
 - 2012: Krizhevsky, Sustkever, Hinton: significant improvement of state-of-the-art on imangenet dataset (object recognition of 1000 classes), without unsupervised pre-training

A brief history of computers

	1970	1980	1990	2000	2010	2020
Data (samples)	10^2 (e.g. iris)	10^3	10^4 PCR deep nets	$100x$ 10^{7-8} web	$100x$ 10^{10} advertising	$1,000x$ 10^{12} social nets deep nets
RAM	1kB	100kB	10MB	100MB	1GB	100GB
CPU	100kF (8080)	1MF (80186)	10MF (80486)	1GF (Intel Core)	100GF NVIDIA	$>1PF$ (P3 Volta) $10,000x$



KDD2018

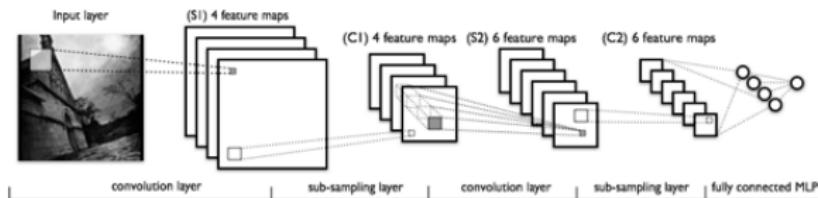


Issues with “Traditional NNs”

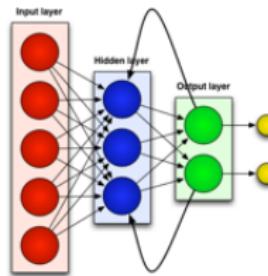
- Huge number of edges \Rightarrow huge number of weights (due to *full connectivity*)
- Domain structure is not taken into account
 - in some domains (e.g., images, text) there is a lot of *structure* \Rightarrow parts of the input have specific relations!
 - what about having *similar/related* neurons?

Modern Neural Networks

- Convolutional Neural Networks (CNN): used mostly for computer vision, imaging, etc.



- Recurrent Neural Networks (RNN): used mostly for time series analysis, speech recognition, machine translation, etc.



• ...

This lecture: focus on CNNs

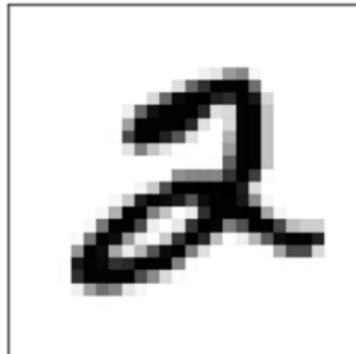
Convolutional Neural Networks (CNNs)

Convolutional NNs: employ the operation of *convolution*

Definition. Convolutional networks are neural networks that use convolution in place of general matrix multiplication in at least one of their layers

CNNs are a specialized kind of neural network for processing data that has a known grid-like topology

Example



Convolution in CNNs

Assume we have a two-dimensional input I and a two-dimensional function K , the (discrete) convolution S of I and K is:

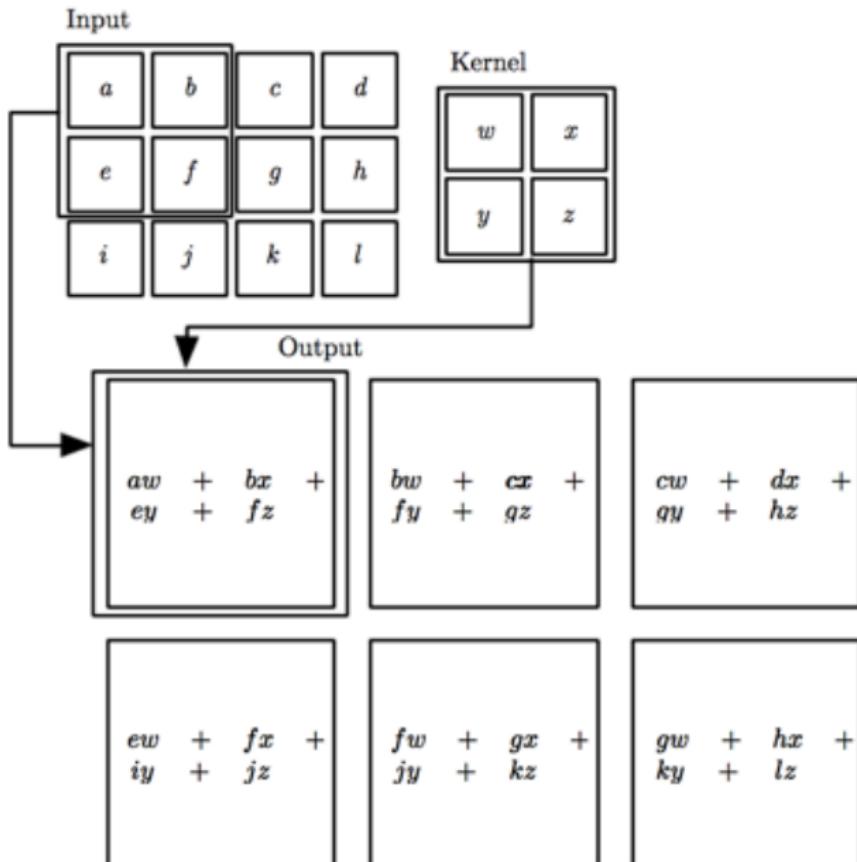
$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

Discrete convolution can be viewed as multiplication by a matrix, but the matrix has several entries constrained to be equal to other entries

Note: K is usually called a *kernel*

In CNNs $K(m, n)$ is zero for all but small values of m, n

Convolution in CNNs (continue)



Convolution: Example

Note: the kernel is learned from training data

Convolution: Example

INPUT



OUTPUT



KERNEL

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

Filter: *sharpen*

Convolution: Example (2)

INPUT



OUTPUT



KERNEL

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Filter: blur

Convolution: Example (3)

INPUT



OUTPUT



KERNEL

0	0	0
-1	1	0
0	0	0

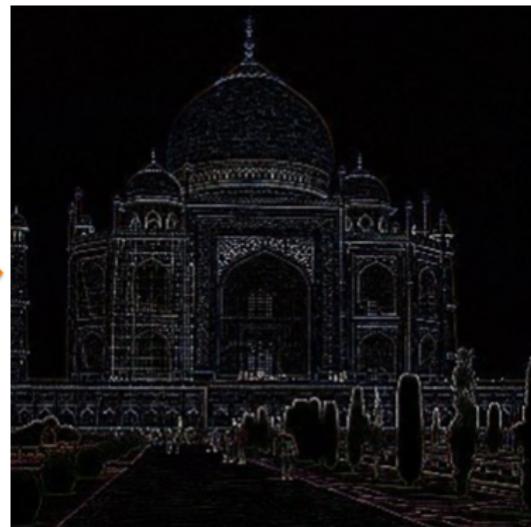
Filter: *edge enhance*

Convolution: Example (4)

INPUT



OUTPUT



KERNEL

0	1	0
1	-4	1
0	1	0

Filter: *edge detect*

Convolution: Properties

The use of convolution leads to useful properties:

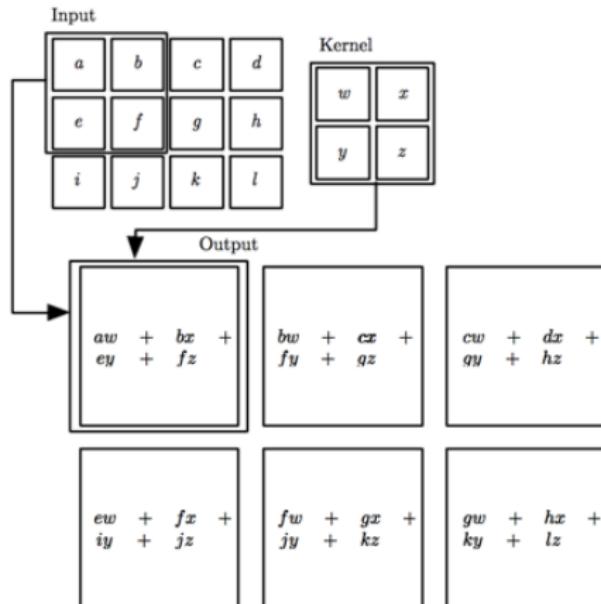
- sparse interactions
- parameter sharing
- equivariant representations

Sparse interactions and parameter sharing: reduce the number of parameters \Rightarrow can be thought as a form of regularization!

Equivariant representations: useful for images (same image/object moved in space)

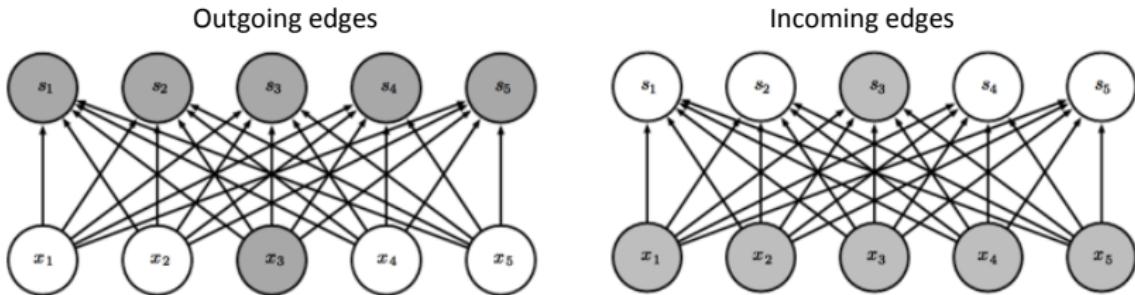
Convolution: Sparse Interactions

Sparse interactions: many edges do not exist in the network \Rightarrow have 0 weight

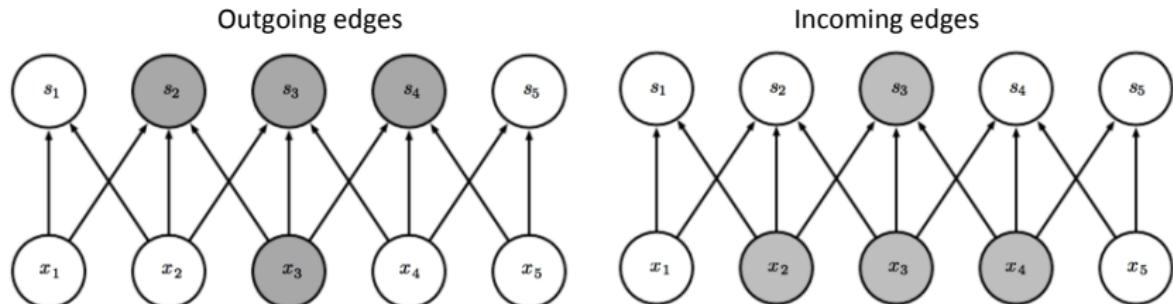


Convolution: Sparse Interactions (continue)

Standard NN



Convolutional NN

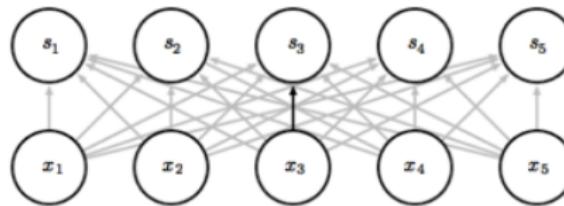


Parameters Sharing

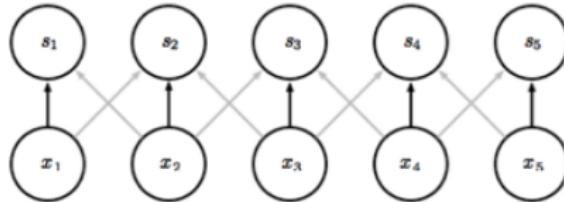
Parameters Sharing = using the same parameter for more than one function in a model

Therefore: rather than learning a separate set of parameters for every input location, we learn only one set

traditional NN



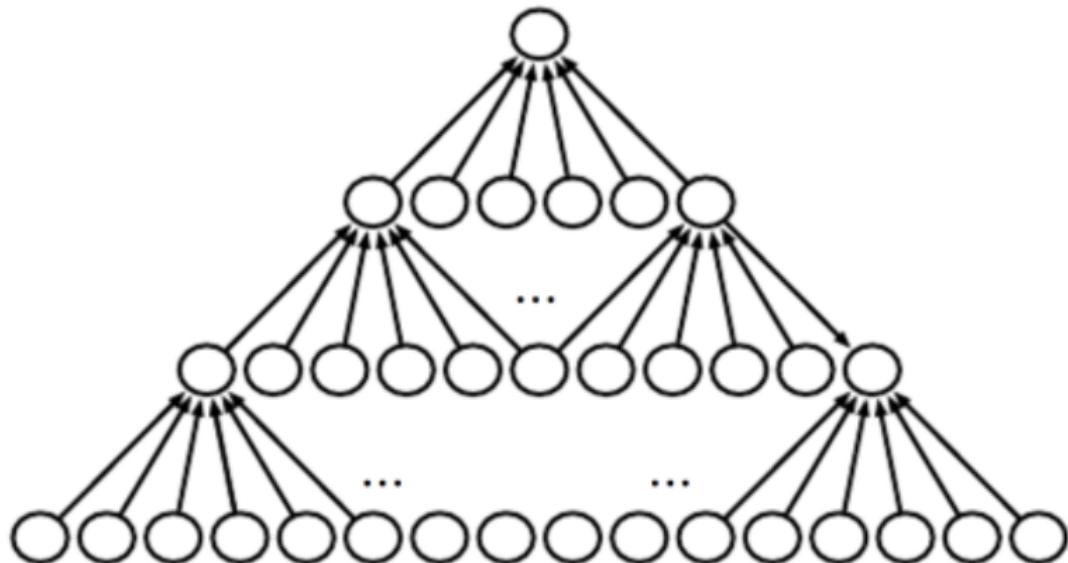
convolutional NN



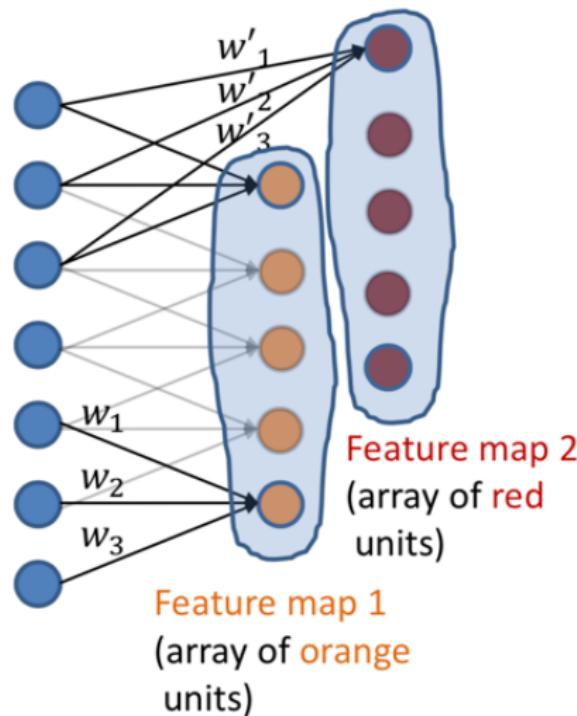
CNNs: Zero Padding

Sometimes want to apply the kernel the same number of times to each input or not to reduce dimension of next layer \Rightarrow need to add extra (virtual) inputs \Rightarrow **zero-padding**

No padding:



Multiple Feature Maps



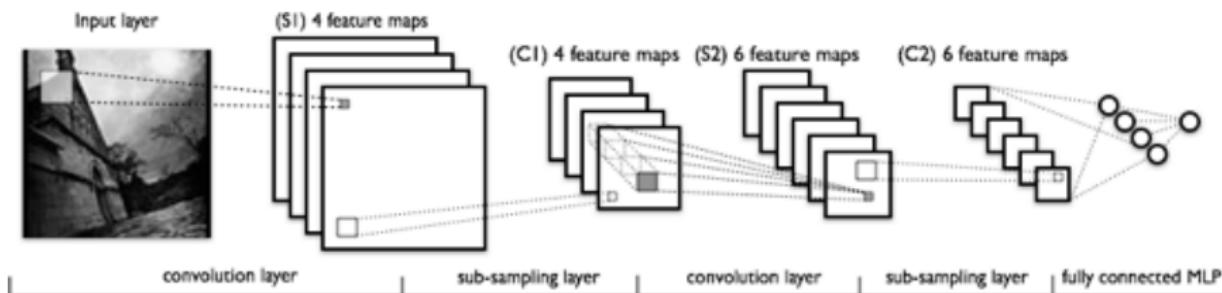
- All **orange units** compute the same function but with a different input windows
- **Orange** and **red** units compute different functions

CNNs: Convolutional Layers

In a CNN:

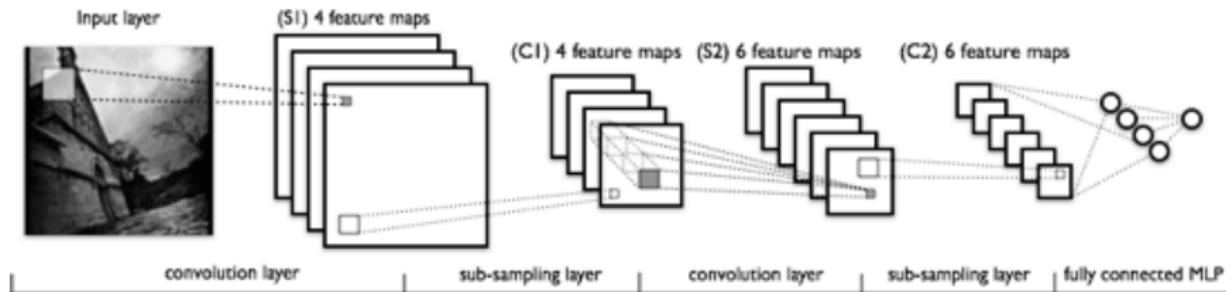
- there are **convolutional layers** (implement convolution) and other types of layers
- the convolutional layer models consists of several filters/kernels

Example



- 4 kernels are learned by the CNN in the first convolutional layer
- 6 kernels are learned by the CNN in the second convolutional layer

CNNs: After Convolution

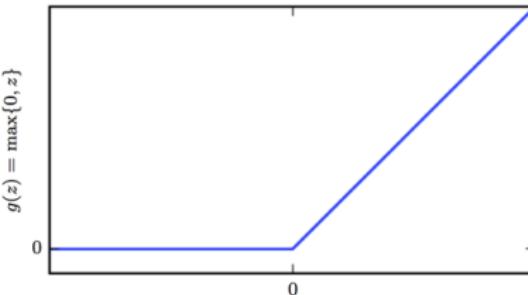


After a convolutional layer:

- nonlinear function: **ReLU (Rectified Linear Unit)**
- **pooling** layer, often combined with **subsampling**

ReLU (Rectified Linear Unit)

$$\sigma(z) = \max\{0, z\}$$



- helps convergence (almost linear)
- does not hurt expressiveness
- prevents the *vanishing gradient function* (gradient for sigmoid o threshold functions is ≈ 0 when weights have large absolute value \Rightarrow weights do not change)
- most common activation function for deep NNs
- extremely common for CNNs
- leads to sparse activation
- more efficiently computable than other activation functions

The layer with ReLU activation is sometimes called **detector layer**

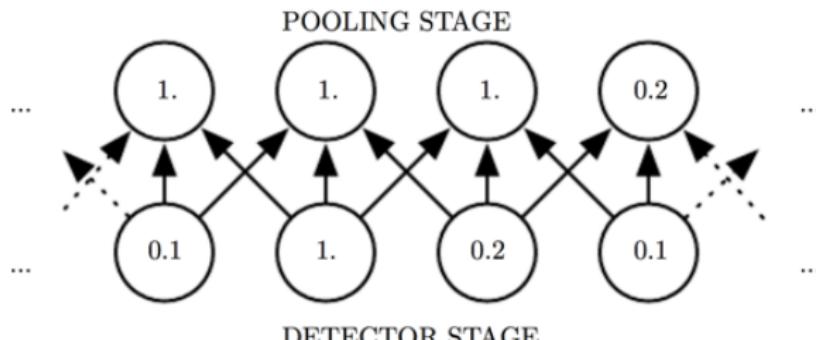
Pooling

Pooling function: replaces the output of the network at a certain location with a summary statistic of the nearby outputs.

(Popular) Examples:

- *maximum* of a rectangular neighborhood (**max pooling**)
- *average* of a rectangular neighborhood (**average pooling**)
- weighted average based on the distance from the network location
- ...

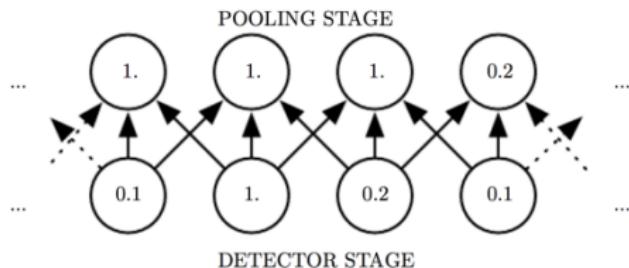
Example: Max Pooling



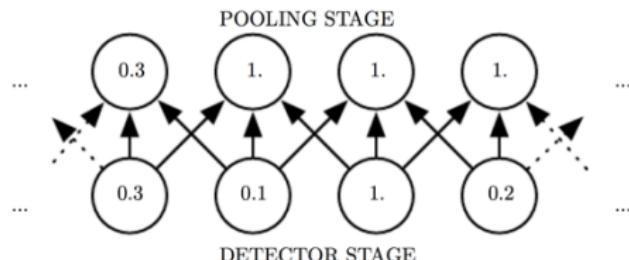
Max Pooling and Invariance

Max pooling introduces **invariance** to local translation: many neurons have the same output value even if input values are “shifted” a bit

Before shift

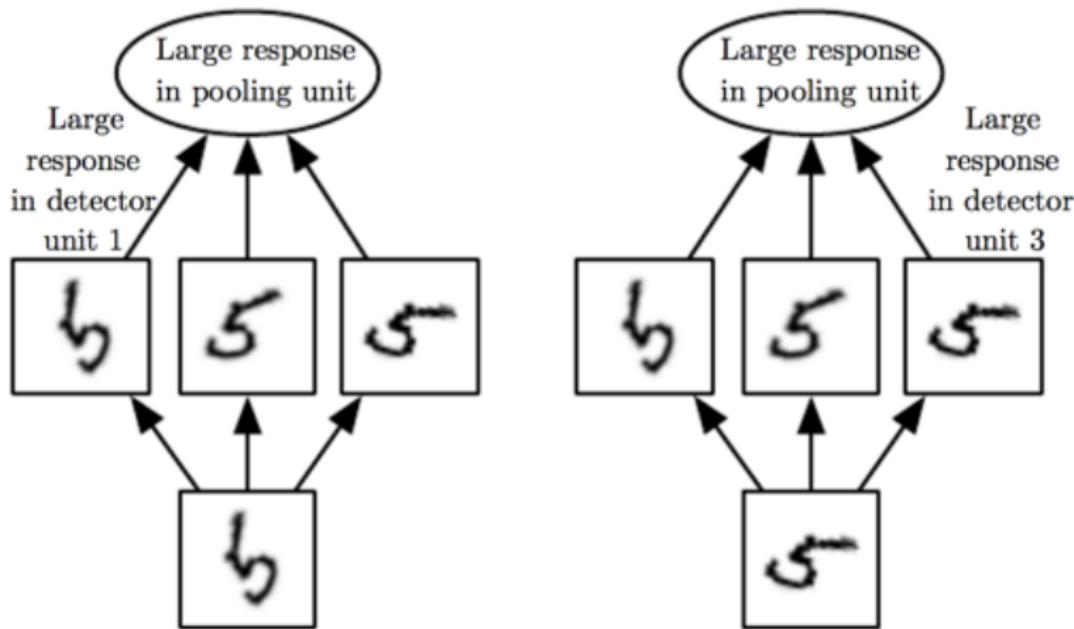


After shift (to the right by 1)



Max Pooling and Invariance (continue)

Invariance to local translation: useful if we care more about whether some feature *is present* than exactly *where* it is or *how it appears*

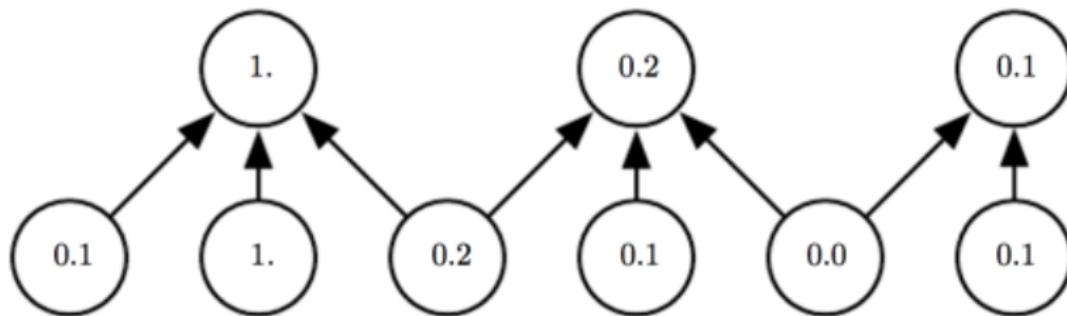


Pooling and Subsampling

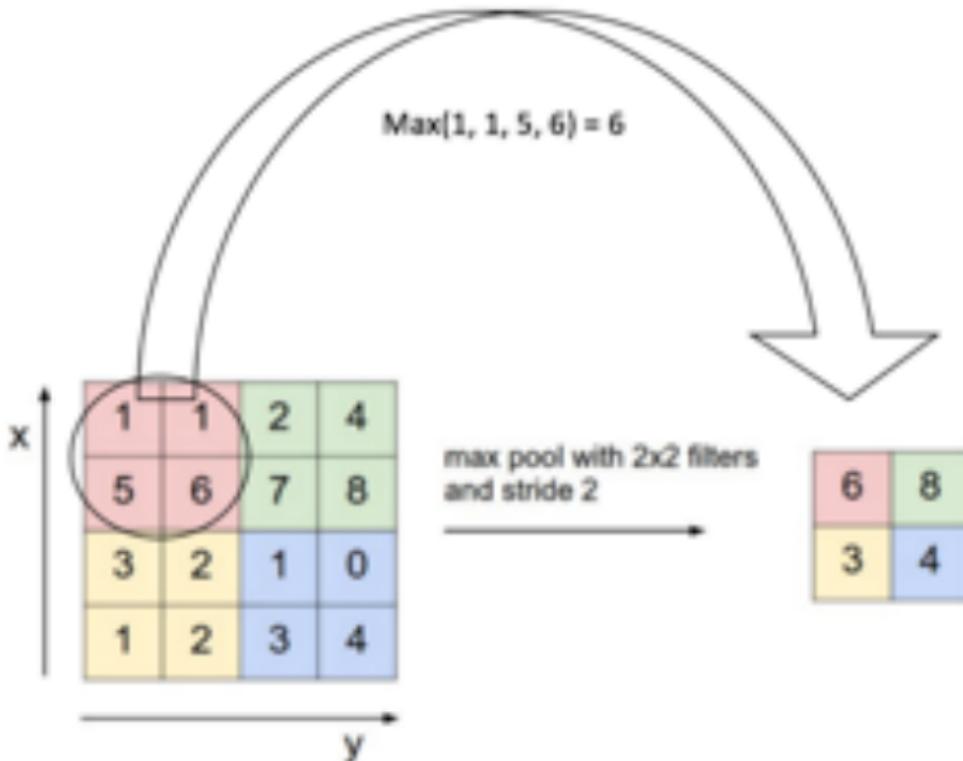
Often pooling is combined with subsampling

Skip the application of pooling for a given number of steps = **stride**

Example: max pooling, pool width = 3, stride = 2



Max Pooling and Subsampling: Example



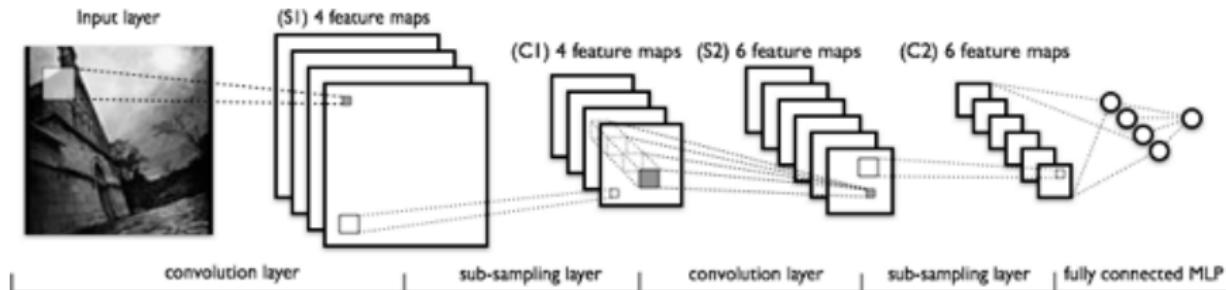
Rectified Feature Map

Pooling and Subsampling: Notes

Pooling and subsampling have some nice properties:

- almost scale invariant representation
- makes the input representations (*feature dimension*) smaller and more manageable
- reduces the number of parameters \Rightarrow controls overfitting
- reduces computation

CNN: Last Layer(s)



At the end: fully connected NN (MLP - Multi Layer Perceptron), which learns from the features extracted at the previous hidden layers

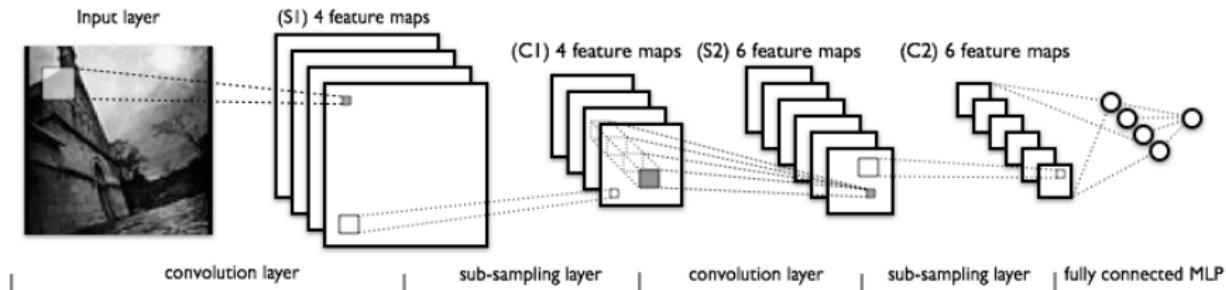
Machine Learning

NN and Deep Learning

Fabio Vandin

December 16, 2019

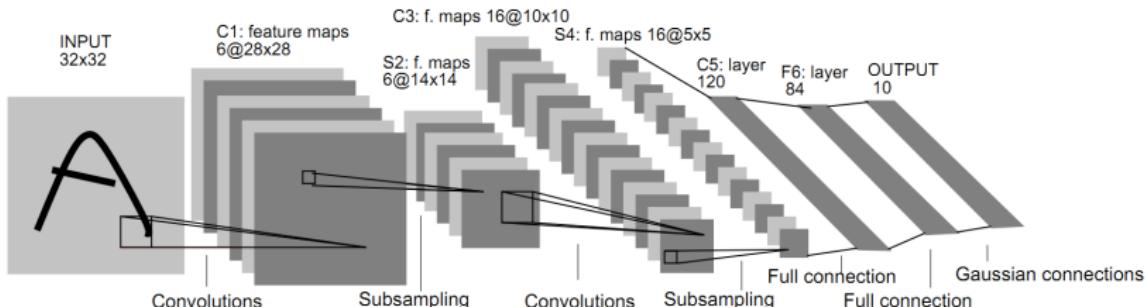
CNN: Last Layer(s)



At the end: fully connected NN (MLP - Multi Layer Perceptron), which learns from the features extracted at the previous hidden layers

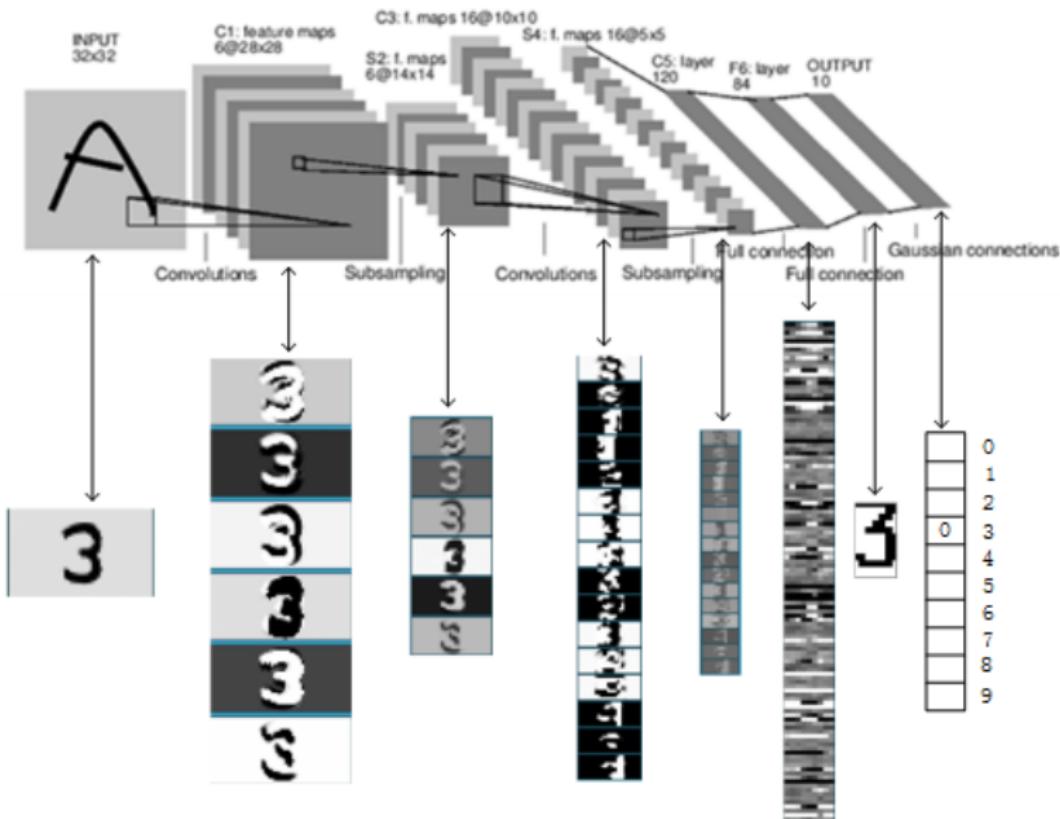
CNN: LeNet-5

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE 86(11): 2278-2324, 1998.

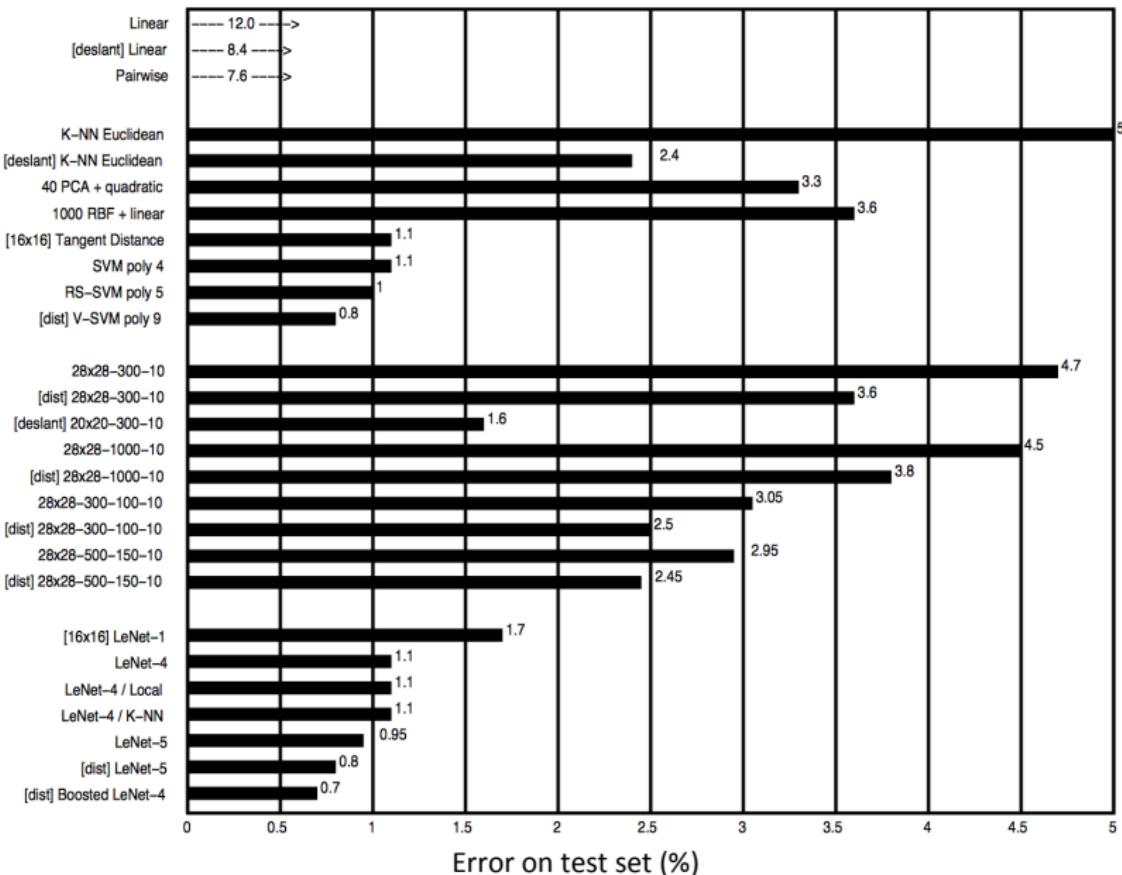


- task: digit recognition (output: $\text{Pr}[\text{INPUT} = i]$ for $i = 0, 1, \dots, 9$)
- convolutional filters: 5×5
- subsampling and pooling filters: 2×2 , stride = 2
- used (some form of) average pooling
- trained on MNIST digit dataset with 60K training examples

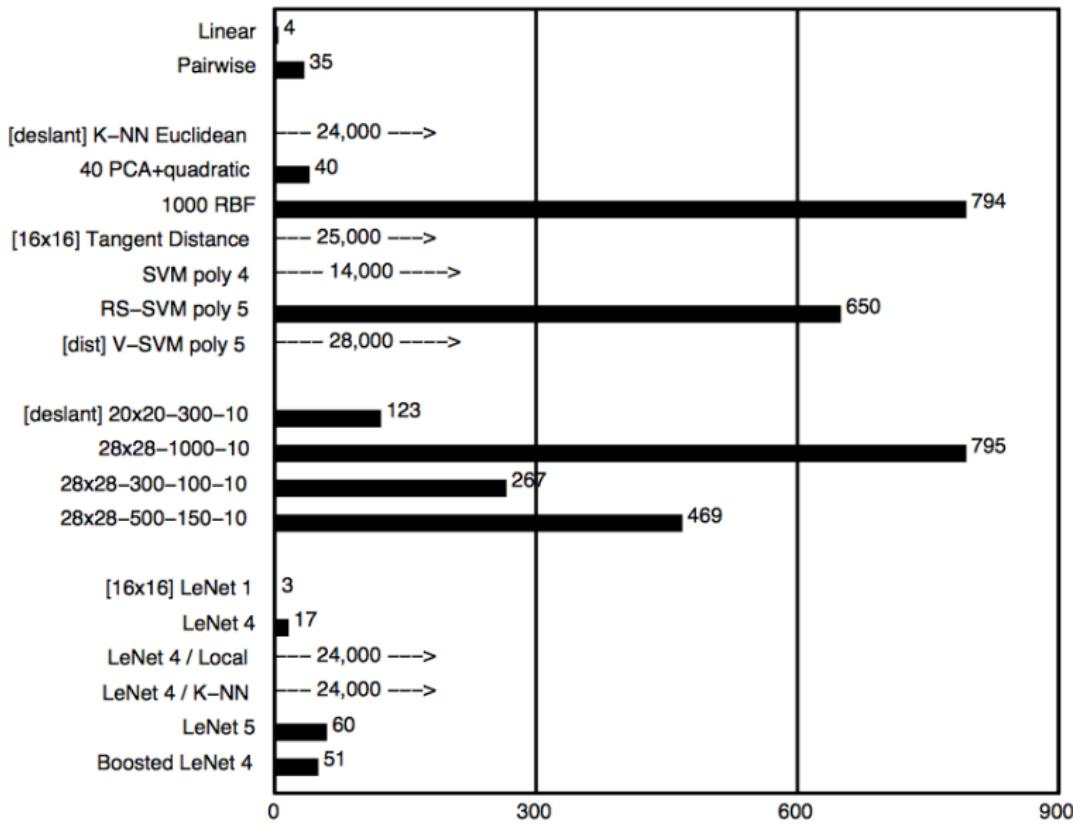
LeNet-5 Features



LeNet-5 Results



LeNet-5 Results (continue)



Memory Requirements (≈number of variables to be stored per pixel)

How well do CNN perform?

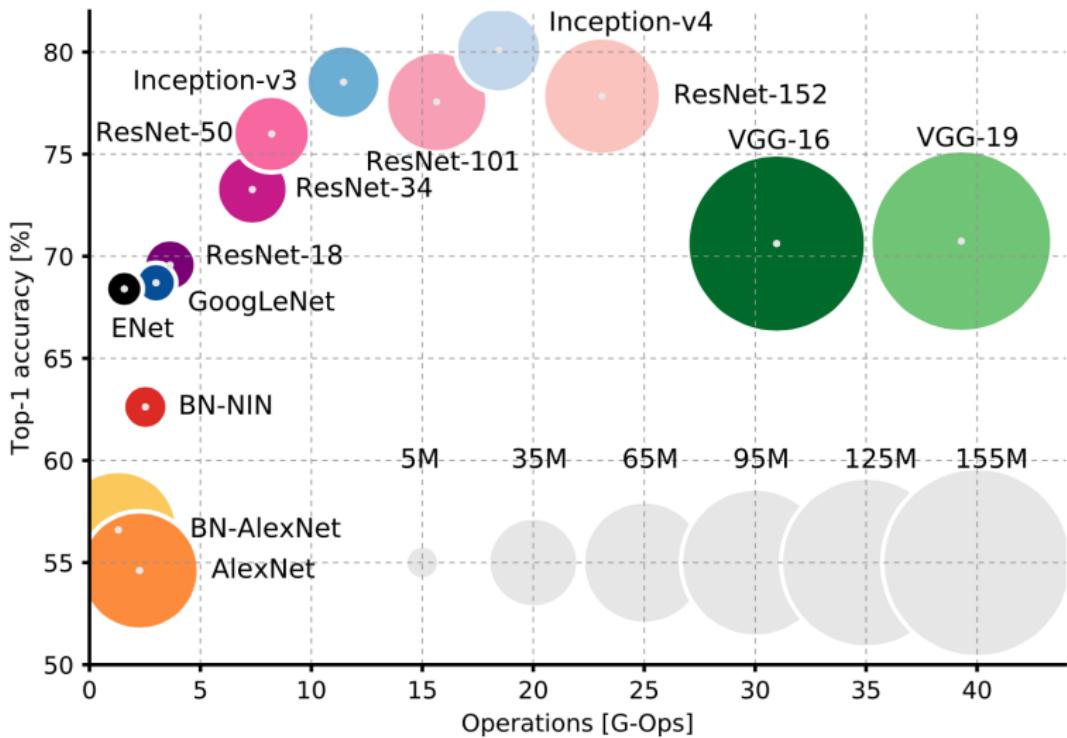
ImageNet competition: “The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale”
(<http://www.image-net.org/challenges/LSVRC/>)

Uses CNN; does not uses CNN

2012 Teams	%error	2013 Teams	%error	2014 Teams	%error
Supervision (Toronto)	15.3	Clarifai (NYU spinoff)	11.7	GoogLeNet	6.6
ISI (Tokyo)	26.1	NUS (singapore)	12.9	VGG (Oxford)	7.3
VGG (Oxford)	26.9	Zeiler-Fergus (NYU)	13.5	MSRA	8.0
XRCE/INRIA	27.0	A. Howard	13.5	A. Howard	8.1
UvA (Amsterdam)	29.6	OverFeat (NYU)	14.1	DeeperVision	9.5
INRIA/LEAR	33.4	UvA (Amsterdam)	14.2	NUS-BST	9.7
		Adobe	15.2	TTIC-ECP	10.2
		VGG (Oxford)	15.2	XYZ	11.2
		VGG (Oxford)	23.0	UvA	12.1

2017: WMW wins with error < 2.3% - CNN with > 150 layers!

Many more CNNs have been proposed...



CNNs: Other Details

- how to learn parameters for a CNN? SGD with backpropagation!
Note: ReLU is not differentiable \Rightarrow use right/left/average derivative...
- **Input normalization:** divide each element of input to make sure it is in $[0, 1]$
- **Initialization is important:** one trick that works well in practice is to initialize the bias to be zero and initialize the each weight to be random in $[-1/d, 1/d]$, where d is the dimension of the input
- **Mini-batches:** at each iteration of SGD we calculate the average loss on k random examples for $k > 1$. Advantages:
 - reduces the variance of the update direction (w.r.t. the full gradient) \Rightarrow converges faster
 - don't loose a lot in running time (parallel implementation)
- many variants of SGD... e.g., ADAM
- techniques to help avoiding overfitting
- a specific loss function
- ...

ADAM [Kingma and Ba 2014]

ADAM = Adaptive Moment Estimation Iterative method. Let $\mathbf{g}^{(t)} \in \nabla f(\mathbf{x})$. Updates at iteration t :

- ① $\mathbf{m}^{(t)} \leftarrow \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \mathbf{g}^{(t)}$
- ② $\mathbf{v}^{(t)} \leftarrow \beta_2 \mathbf{m}^{(t-1)} + (1 - \beta_2) (\mathbf{g}^{(t)})^2$
- ③ $\hat{\mathbf{m}}^{(t)} \leftarrow \mathbf{m}^{(t)} / (1 - \beta_1^t)$
- ④ $\hat{\mathbf{v}}^{(t)} \leftarrow \mathbf{v}^{(t)} / (1 - \beta_2^t)$
- ⑤ $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \hat{\mathbf{m}}^{(t)} / (\hat{\mathbf{v}}^{(t)} + \delta)$

$\mathbf{m}^{(t)}$: first moment (*momentum*, helps for non-smooth loss functions)

$\mathbf{v}^{(t)}$: second moment (better estimates and convergence)

$\hat{\mathbf{m}}^{(t)}$: unbiased estimate of first moment (because estimates start at 0)

$\hat{\mathbf{v}}^{(t)}$: unbiased estimate of second moment (because estimates start at 0)

ADAM (continue)

β_1, β_2, η : parameters

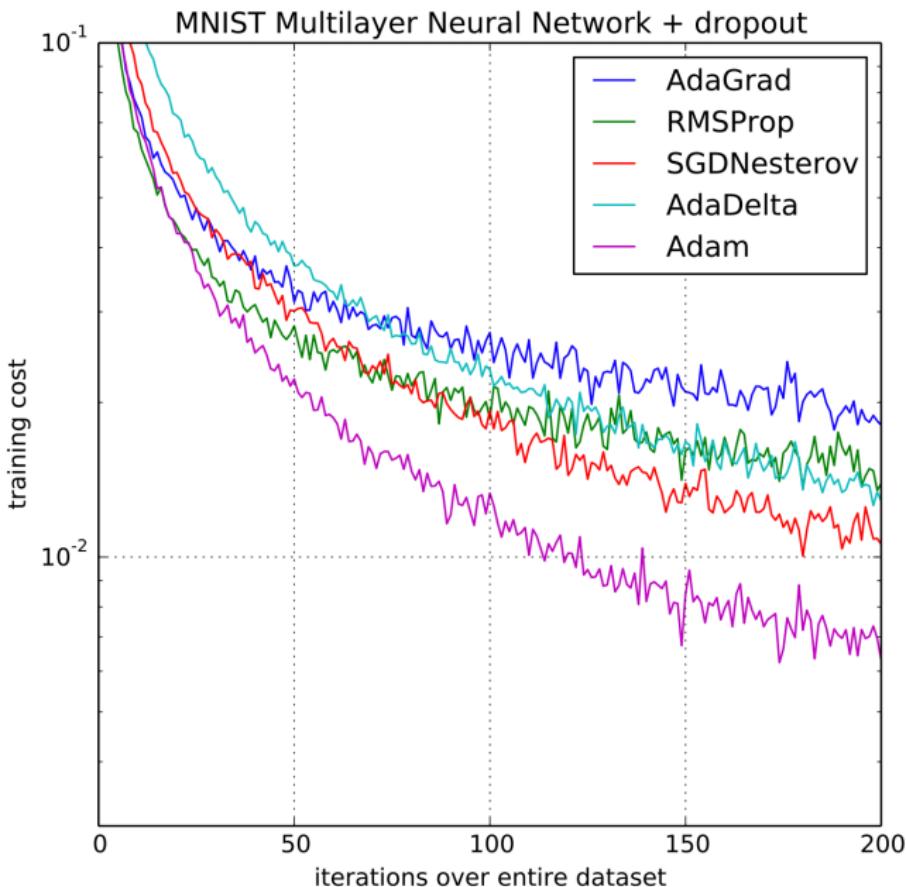
How to set them?

Validation, etc...

Good starting point:

- $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- $\eta = 0.001$ or $\eta = 5 \times 10^{-4}$

ADAM: Results



Techniques to Help Avoiding Overfitting

- regularization! (same as for other models!)
- dropout
- early stopping
- data augmentation

Dropout

It provides a computationally inexpensive but powerful method of regularizing a broad family of models

Is a **bagging method**

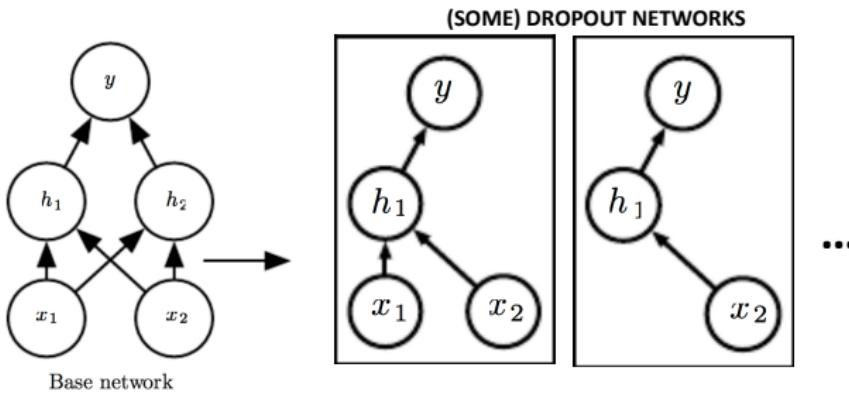
- **bagging** = training multiple models and evaluating multiple models on each test example
- impractical when each model is a large neural network: training and evaluating such networks is costly in terms of runtime and memory

Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of many neural networks

Dropout (continue)

In particular: dropout trains the ensemble consisting of all subnetworks that can be formed by removing nonoutput units from an underlying base network

- *remove unit* = set its output weights to 0's



Dropout (continue)

To train with dropout:

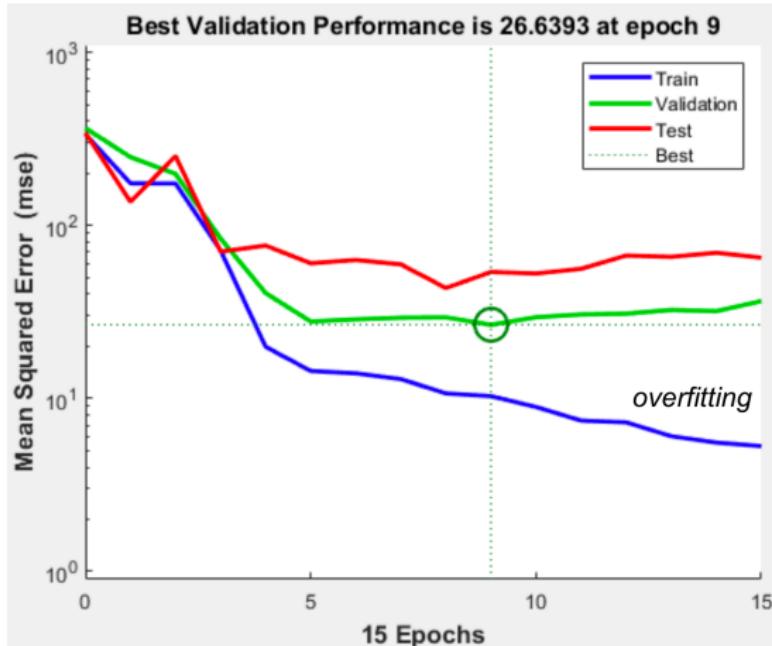
- use a minibatch-based SGD (or other learning algorithm that makes small steps)
- each time an input example is loaded into a minibatch:
 - randomly sample a different binary mask to apply to all the input and hidden units in the network;
 - mask for each unit is sampled independently from all the others;
 - probability of sampling a mask value of one (causing a unit to be included) is a hyperparameter fixed before training begins
 - run forward propagation, back-propagation, and learning update

Typically:

- input unit is included with probability 0.8
- hidden unit is included with probability 0.5

Why does it work? Still not well understood!

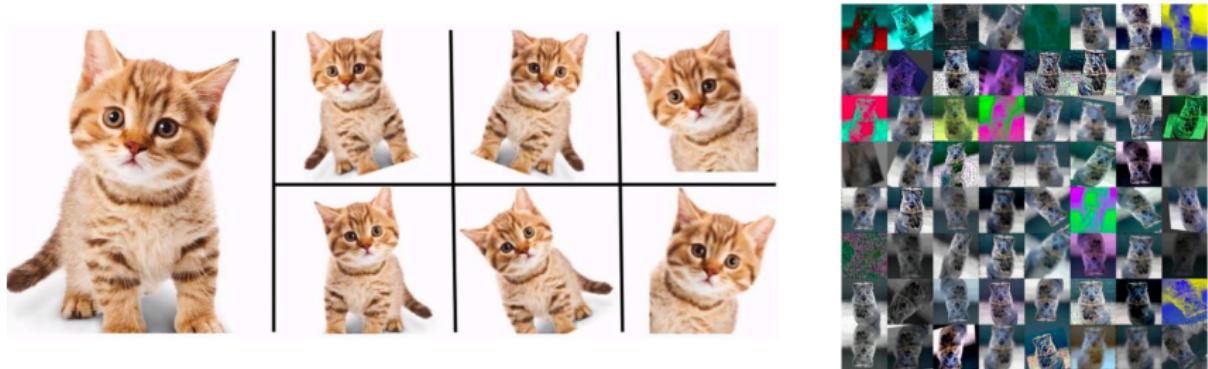
Early Stopping



- Use validation error to decide when to stop training
- Stop when monitored loss has not improved after n subsequent epochs
- Parameter n is called *patience*

Data Augmentation

- Add a little bit of variance to the data and "virtually" increase number of training samples
- Artificially add noise
- Apply random transformations o crop part of the data
 - Resize/rescaledata
 - Rotate
 - Custom transformations depending on data type (e.g. for images: flip horizontally, contrast and saturation, ...)



Loss Function

Classification: *cross entropy* is commonly used (instead of 0-1 loss)

Hypotheses set: functions with (prediction) value in 0 and 1
⇒ use sigmoid activation function for output

Cross entropy loss:

$$\ell(h, (\mathbf{x}, y)) = -y \log h(\mathbf{x}) - (1 - y) \log (1 - h(\mathbf{x}))$$

Cross entropy is a convex function ⇒ SGD works better

The best hypothesis h w.r.t. to the cross entropy loss is:

$h(\mathbf{x}) = \Pr[y = 1 | \mathbf{x}]$ (related to Bayes optimal predictor!)

For multiclass classification: k classes $(0, 1, \dots, k)$

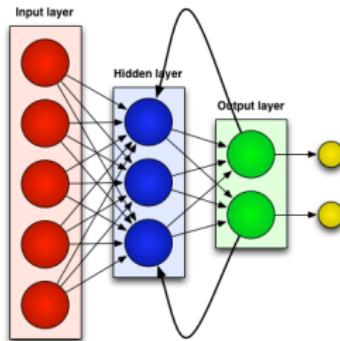
- output is vector \mathbf{y} with $y_i = 1$ if correct class is i , 0 otherwise
- $h((\mathbf{x})) \in (0, 1)^d$ with $h_i(\mathbf{x})$ = probability label of \mathbf{x} is i ,
 $1 \leq i \leq k$
- $\ell(h, (\mathbf{x}, \mathbf{y})) = -\sum_{i=1}^k y_i \log (h_i(\mathbf{x}))$

Recurrent Neural Networks (RNNs)

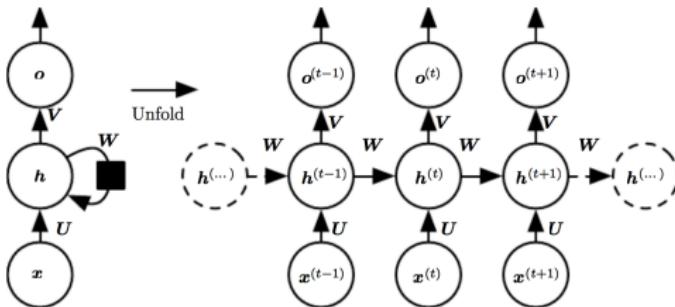
Input: sequential data $x^{(t)}$, for $t = 1, 2, \dots$

Main idea: hidden state at time t are function of

- input at time t : $x^{(t)}$
- hidden state $h^{(t-1)}$ at time $t-1$ and/or output $o^{(t-1)}$ at time $t-1$



Example: hidden state depends on hidden state at previous time step



$$h^{(t)} = f(h^{(t-1)}, \mathbf{x}^t)$$

Extremely powerful model! The recurrent neural network above *is universal*: can compute any function computable by a Turing machine

Applications:

- speech recognition
- machine translation
- generating image descriptions
- ...

Main Reasons for Deep Learning Success

- ① availability of large datasets
- ② advances in hardware/systems (e.g., GPUs)
- ③ advances in computational techniques/algorithms (e.g., dropout)

What's Missing?

- running time for training: still very high!
- design choices: number of layers, types of layers, number of nodes, etc.... \Rightarrow how?
- theoretical foundations still not completed...
 - “Machine Learning has become the new alchemy”
Ali Rahimi (talk at NIPS 2017 - Test-of-time award presentation)
- interpretability of models not always possible
 - how do you explain hiring somebody? not giving them a loan?
deciding to cure somebody?
 - *fairness* in ML...

Credits

- Yann LeCun CVPR 2015 keynote
- *Deep Learning*, I. Goodfellow, Y. Bengio, A. Courville,
(<http://www.deeplearningbook.org>)

Machine Learning

Clustering

Fabio Vandin

December 19, 2019

Unsupervised Learning

In unsupervised learning, the training dataset is $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m)$
⇒ no target values!

We are interested in finding some interesting *structure* in the data,
or, equivalently, to organize it in some meaningful way.

We are going to see few unsupervised learning techniques

- *clustering*
- *dimensionality reduction*

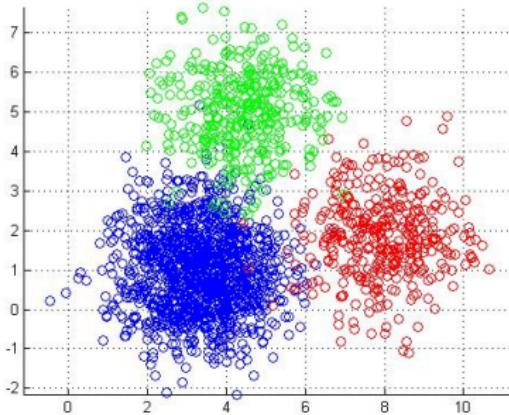
We are going to see some of the most commonly used techniques:

- clustering: linkage-based clustering, K-means, mixture of Gaussians (if time permits)
- dimensionality reduction: PCA

There are also other general techniques: association analysis,...

Clustering

Informal definition: the task of identifying meaningful groups among data points.



Definition

Clustering is the task of grouping a set of objects such that similar objects end up in the same group and dissimilar objects are separated into different groups.

Example



- Data: features (e.g. product bought, demographic info, etc.) for a large number of customers
- Goal: **customers segmentation** = identify subgroups of homogeneous customers
- useful for: advertising, product development, ...

Example (2)

Data:

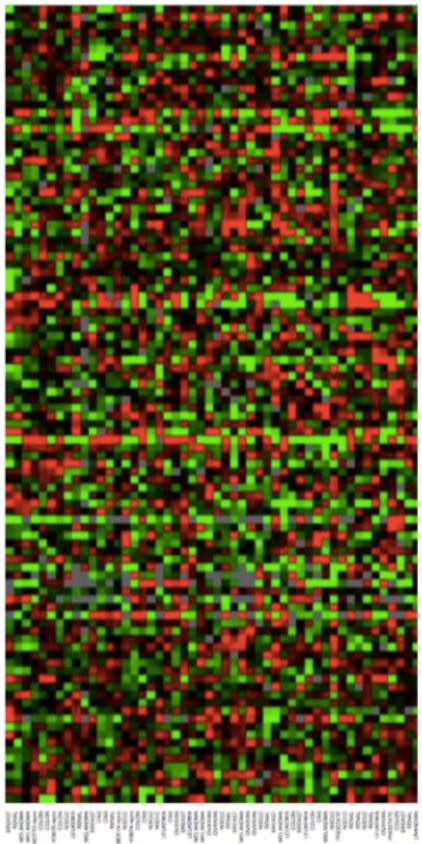
- rows = genes ($\approx 20 \times 10^3$)
- columns = samples, cancer patients ($\approx 10^3 - 10^4$)
- values = expression of a gene in a patient ($\in \mathbb{R}$)

Goal: find similar cancer samples

- cluster columns (samples) to find similar subgroups of patients (e.g., *disease subtypes*)

Goal: find genes with similar gene expression profiles

- cluster rows (genes) to deduce function of unknown genes from experimentally known genes with similar profiles



Clustering Definition

Definition

Clustering is the task of grouping a set of objects such that similar objects end up in the same group and dissimilar objects are separated into different groups.

Note: the definition above is not rigorous and may be ambiguous

⇒ different definitions have been proposed that may lead to different types of clustering. We will see only few of them.

Note: there are some difficulties that are somehow inherent in clustering...

Clustering: Difficulties

Similarity is *not transitive*

⇒ “similar objects in same group” and “dissimilar objects into different groups” may contradict each other...

Example

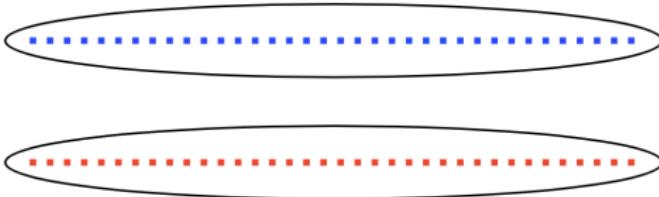
Assume we have data points in \mathbb{R}^2 as in figure



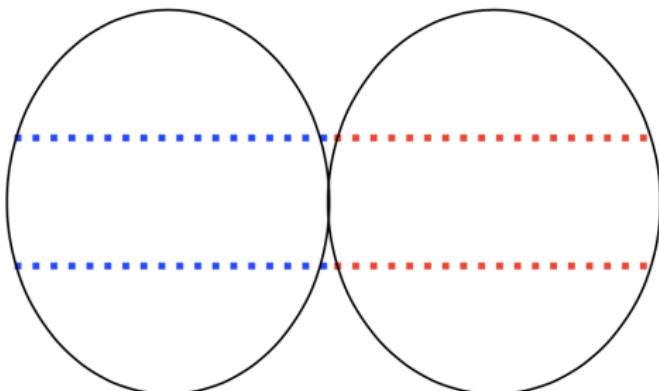
Assume we want to cluster the data into $k = 2$ clusters. How should we cluster the data?

Clustering: Difficulties (continue)

If we focus on “similar objects in same group”:



If we focus on "dissimilar objects into different groups":

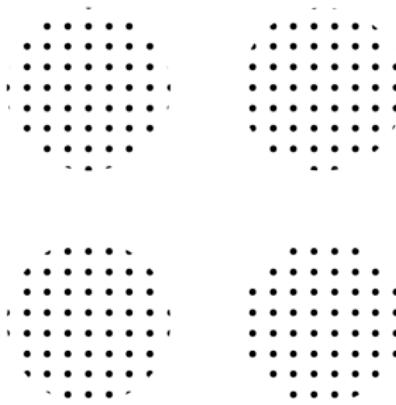


Clustering: Difficulties (continue)

In general we do not have a *ground truth* to evaluate our clustering
(*unsupervised learning*)

Example

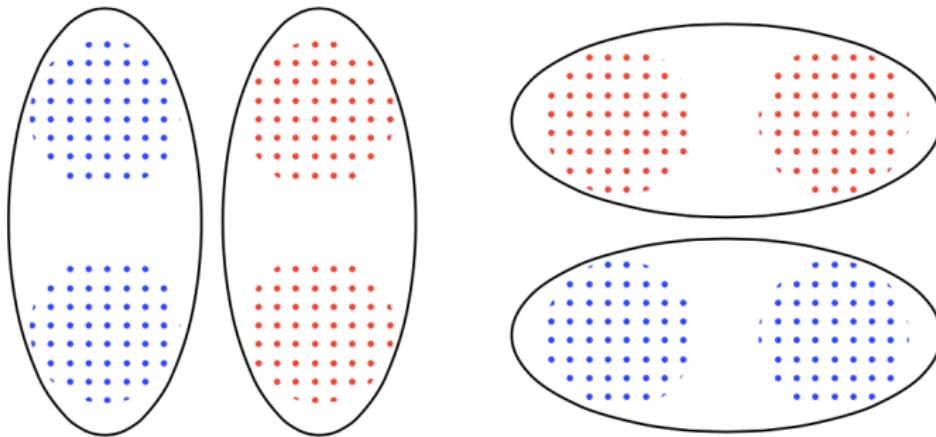
Assume we have data points in \mathbb{R}^2 as in figure



Assume we want to cluster the data into $k = 2$ clusters. What is a correct clustering?

Clustering: Difficulties (continue)

The following clusterings are different but both justifiable



In practice: a given set of objects can be clustered in various different *meaningful* ways

A Model for Clustering

Let's formulate the clustering problem more formally:

- **Input:** set of elements \mathcal{X} and *distance* function $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+$, that is a function that
 - is symmetric: $d(\mathbf{x}, \mathbf{x}') = d(\mathbf{x}', \mathbf{x})$ for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$
 - $d(\mathbf{x}, \mathbf{x}) = 0$ for all $\mathbf{x} \in \mathcal{X}$
 - (often) d satisfies the triangle inequality:
$$d(\mathbf{x}, \mathbf{x}') \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{x}')$$
- **Output:** a partition of \mathcal{X} into *clusters*, that is $C = (C_1, C_2, \dots, C_k)$ with
 - $\bigcup_{i=1}^k C_i = \mathcal{X}$
 - for all $i \neq j$: $C_i \cap C_j = \emptyset$
- **Notes:**
 - sometimes the input also includes the number k of clusters to produce in output
 - sometimes, the output is a **dendrogram** (from Greek *dendron* = tree, *gramma* = drawing), a tree diagram showing the arrangement of the clusters

A Model for Clustering (continue)

Sometimes instead of a distance function we have a similarity function $s : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+$, that is a function that:

- is symmetric: $s(\mathbf{x}, \mathbf{x}') = s(\mathbf{x}', \mathbf{x})$ for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$
- $s(\mathbf{x}, \mathbf{x}) = 1$ for all $\mathbf{x} \in \mathcal{X}$

Choice of distances/similarity: depends on the type of data

Classes of Algorithms for Clustering

- ① Cost minimization algorithms
- ② Linkage-based algorithms

Cost Minimization Clustering

Common approach in clustering:

- define a cost function over possible partitions of the objects
- find the partition (=clustering) of minimal cost

Assumptions:

- data points $\mathbf{x} \in \mathcal{X}$ come from a larger space \mathcal{X}' , that is
 $\mathcal{X} \subseteq \mathcal{X}'$
- distance function $d(\mathbf{x}, \mathbf{x}')$ for $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$

For simplicity: assume $\mathcal{X}' = \mathbb{R}^d$ and $d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$

K-Means Clustering

Input: data points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$; $k \in \mathbb{N}^+$

Goal: find

- partition $C = (C_1, C_2, \dots, C_k)$ of $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$;
- centroids* $\mu_1, \mu_2, \dots, \mu_k$ with $\mu_i \in \mathcal{X}'$ centroid for C_i ,
 $1 \leq i \leq k$

that minimizes the **k -means objective** (cost)

$$\sum_{i=1}^k \sum_{\mathbf{x} \in C_i} d(\mathbf{x}, \mu_i)^2$$

Other Objectives

k -medoids objective:

$$\min_{\mu_1, \dots, \mu_k \in \mathcal{X}} \sum_{i=1}^k \sum_{x \in C_i} d(x, \mu_i)^2$$

k -median objective:

$$\min_{\mu_1, \dots, \mu_k \in \mathcal{X}} \sum_{i=1}^k \sum_{x \in C_i} d(x, \mu_i)$$

Back to k -mean clustering

Proposition

Given a cluster C_i , the center μ_i that minimizes $\sum_{x \in C_i} d(x, \mu_i)^2$ is

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

Naive (brute-force) algorithm to solve K-Means Clustering?

Try all possible partitions of the m points into k clusters, evaluate each partition, and find the best one.

Is it efficient?

Depends on the number of partitions of m points into k clusters:

- exact count: number of ways in which we can partition a set of m objects into k subsets \Rightarrow Stirling number of the second kind:

$$S(m, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^m$$

- simple bounds:

- $S(m, k) \in O\left(\frac{k^m}{k!}\right)$
- $S(m, k) \in \Omega\left(k^{m-k+1}\right)$

Fact

Finding the optimal solution for k -means clustering is computationally difficult (NP-hard). This is true for most optimization problems of cost minimization clusterings (including k -medoids and k -median)

Lloyd's Algorithm

A good practical heuristic to solve k -means

Input: data points $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$; $k \in \mathbb{N}^+$

Output: clustering $C = (C_1, C_2, \dots, C_k)$ of \mathcal{X} ; centers

$\mu_1, \mu_2, \dots, \mu_k$ with μ_i center for C_i , $1 \leq i \leq k$;

randomly choose $\mu_1^{(0)}, \dots, \mu_k^{(0)}$;

for $t \leftarrow 0, 1, 2, \dots$ **do** /* until convergence */

for $i = 1, \dots, k$: $C_i \leftarrow \{\mathbf{x} \in \mathcal{X} : i = \arg \min_j d(\mathbf{x}, \mu_j^{(t)})\}$;

for $i = 1, \dots, k$: $\mu_i^{(t+1)} \leftarrow \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$;

if convergence reached **then**

return $C = (C_1, \dots, C_k)$ and $\mu_1^{(t+1)}, \mu_2^{(t+1)}, \dots, \mu_k^{(t+1)}$

Notes

Convergence: commonly used criteria

- $\sum_{i=1}^k d(\mu_i^{(t+1)}, \mu_i^{(t)}) \leq \varepsilon$
- $\max_{1 \leq i \leq k} d(\mu_i^{(t+1)}, \mu_i^{(t)}) \leq \varepsilon$

Complexity:

- Assignment of points $x \in \mathcal{X}$ to clusters C_i : time $O(kmd)$
- Computation of centers μ_i : time $O(md)$

If convergence after t iterations $\Rightarrow O(tkmd)$

There are many other clustering algorithms:

- spectral clustering
- information bottleneck
- ...

Exercize

Draw (approximately) the solution (clusters and centers) found by Lloyd algorithm for the 2 clusters ($k = 2$) problem, when the data ($x_i \in \mathbb{R}$) are the crosses in the figure below and the algorithm is initialised with center values indicated with the circle (○, cluster 1) and triangle (△, cluster 2) shown in the figure.



Linkage-Based Clustering

General class of algorithms that follow the general scheme below.

Algorithm

- ① start from the trivial clustering: each data point is a (single-point) cluster
- ② **until “termination condition”:** repeatedly merge the “closest” clusters of the previous clustering

We need to specify two “parameters”:

- how to define distance between clusters
- termination condition

Linkage-Based Clustering (continue)

Different distances $D(A, B)$ between two clusters A and B can be used, resulting into different linkage methods:

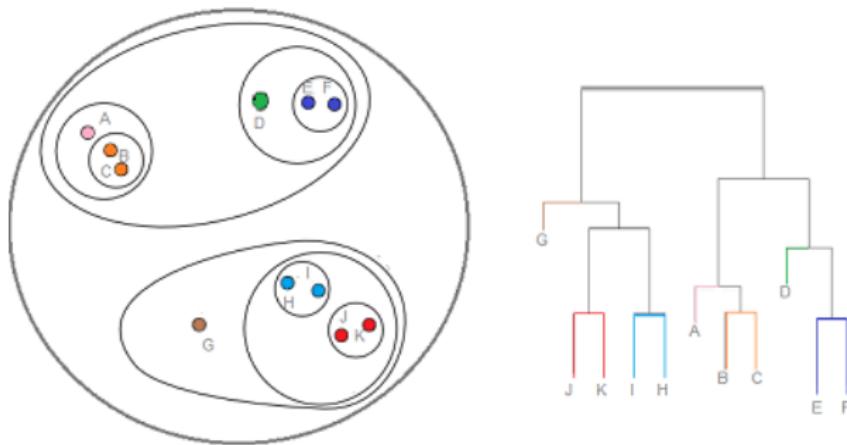
- **single linkage:** $D(A, B) = \min\{d(\mathbf{x}, \mathbf{x}') : \mathbf{x} \in A, \mathbf{x}' \in B\}$
- **average linkage:** $D(A, B) = \frac{1}{|A||B|} \sum_{\mathbf{x} \in A, \mathbf{x}' \in B} d(\mathbf{x}, \mathbf{x}')$
- **max linkage:** $D(A, B) = \max\{d(\mathbf{x}, \mathbf{x}') : \mathbf{x} \in A, \mathbf{x}' \in B\}$

Common termination condition:

- data points are partitioned into k clusters
- minimum distance between pairs of clusters is $> r$, where r is a parameter provided in input
- all points are in a cluster \Rightarrow output is a dendrogram

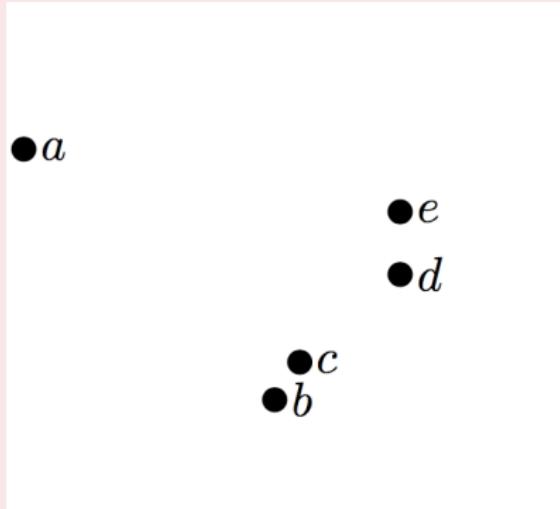
Dendrogram: Example

Dendrogram: tree, with input points $\mathbf{x} \in \mathcal{X}$ as leaves, that shows the arrangement/relation between clusters.



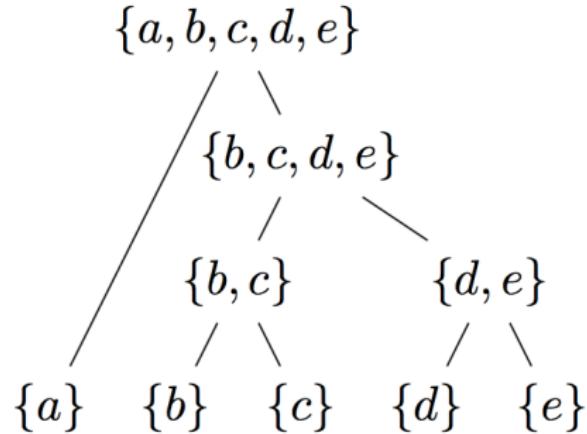
Exercize

Let the dataset \mathcal{X} be as in figure below. Show the output of running the single linkage clustering algorithm when the termination condition is given by having all points in a cluster.



Solution

The output is a dendrogram:



Choice of number k of clusters

Choosing the number k of clusters (e.g., for k -means) is not easy.

Common approach:

- ① run clustering algorithm for various values of k , obtaining a clustering $C^{(k)} = \{C_1^{(k)}, C_2^{(k)}, \dots, C_k^{(k)}\}$ for each value of k considered;
- ② use a score S to evaluate each clustering $C^{(k)}$, getting scores $S(C^{(k)})$ for each value of k
- ③ pick the value of k (and clustering) of maximum score:
$$C = \arg \max_{C^{(k)}} \{S(C^{(k)})\}$$

A very common score based on distances alone: *silhouette*

Silhouette

Given a clustering $C = (C_1, C_2, \dots, C_k)$ of \mathcal{X} , and a point $x \in \mathcal{X}$, let $C(x)$ be the cluster to which x is assigned to. Assume $|C_i| \geq 2 \forall 1 \leq i \leq k$. Define:

$$A(x) = \frac{\sum_{x' \neq x, x' \in C(x)} d(x, x')}{|C(x)| - 1}$$

Given a cluster $C_i \neq C(x)$, let

$$d(x, C_i) = \frac{\sum_{x' \in C_i} d(x, x')}{|C_i|}$$

and $B(x) = \min_{C_i \neq C(x)} d(x, C_i)$.

Then the *silhouette* $s(\mathbf{x})$ of \mathbf{x} is

$$s(\mathbf{x}) = \frac{B(\mathbf{x}) - A(\mathbf{x})}{\max\{A(\mathbf{x}), B(\mathbf{x})\}}$$

Intuition: $s(\mathbf{x})$ measures if \mathbf{x} is closer to points in its “nearest cluster” than to the cluster it is assigned to.

Question: what is the range for $s(\mathbf{x})$?

The silhouette of clustering $C = (C_1, C_2, \dots, C_k)$ is

$$S(C) = \frac{\sum_{\mathbf{x} \in \mathcal{X}} s(\mathbf{x})}{|\mathcal{X}|}$$

The higher $S(C)$, the better the clustering quality.

Machine Learning

Principal Component Analysis

Fabio Vandin

January 9, 2020

Change of Basis

Let $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m \in \mathbb{R}^d$ be our data points. The **data matrix** is

$$D = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_m^T \end{bmatrix}$$

\mathbb{R}^d is the space spanned by the standard basis $\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_d$ with \mathbf{e}_i = vector of all 0's and a 1 in the i -th component.

Let $\mathbf{u}_1, \dots, \mathbf{u}_d$ be d orthonormal vectors, that is:

- $\mathbf{u}_i^T \mathbf{u}_j = 0, \forall i \neq j$
- $\|\mathbf{u}_i\| = 1 = \mathbf{u}_i^T \mathbf{u}_i, \forall i$

Let

$$U = [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_d]$$

Then for any $\mathbf{x} \in \mathbb{R}^d$ we have $\mathbf{x} = U\mathbf{a} = \sum_{i=1}^d a_i \mathbf{u}_i$, where \mathbf{a} is the vector of coordinates of \mathbf{x} in basis $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_d$:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_d \end{bmatrix}$$

Note: U is orthogonal (columns are orthonormal) $\Rightarrow U^{-1} = U^T$

Therefore: $U^T \mathbf{x} = U^T U \mathbf{a} = \mathbf{a}$

Dimensionality Reduction

Goal

Find a *good* r -dimensional representation of D , with $r \ll d$

Given basis $\mathbf{u}_1, \dots, \mathbf{u}_d$ for \mathbb{R}^d , consider the first r basis vectors $\mathbf{u}_1, \dots, \mathbf{u}_r$ (that is a basis for a subspace of dimension r of \mathbb{R}^d) and project \mathbf{x} on those:

$$\mathbf{x}' = \sum_{i=1}^r a_i \mathbf{u}_i = [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_r] \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_r \end{bmatrix} = U_r \mathbf{a}_r$$

and

$$\mathbf{a}_r = U_r^T \mathbf{x}$$

\Rightarrow

$$\mathbf{x}' = U_r U_r^T \mathbf{x} = P_r \mathbf{x}$$

where $P_r = U_r U_r^T$ is the *projection matrix*. (Note that $\mathbf{x}' \in \mathbb{R}^d$)

Projection Matrix: Properties

The following properties hold for the projection matrix

$$P_r = U_r U_r^T$$

- P_r is symmetric
- $P_r^2 = P_r P_r = P_r$

Definition

Error vector: $\varepsilon = \sum_{i=r+1}^d a_i \mathbf{u}_i = \mathbf{x} - \mathbf{x}'$

Proposition

\mathbf{x}' and ε are orthogonal.

Proof.

$$(\mathbf{x}')^T \varepsilon = \sum_{i=1}^r \sum_{j=r+1}^d a_i a_j \mathbf{u}_i^T \mathbf{u}_j = 0$$

□

Dimensionality Reduction: Applications

- number of samples is limited: “reduce” number of features
- capture most important aspects of the data for subsequent analysis (e.g., clustering with many noisy dimensions)
- visualization
- ...

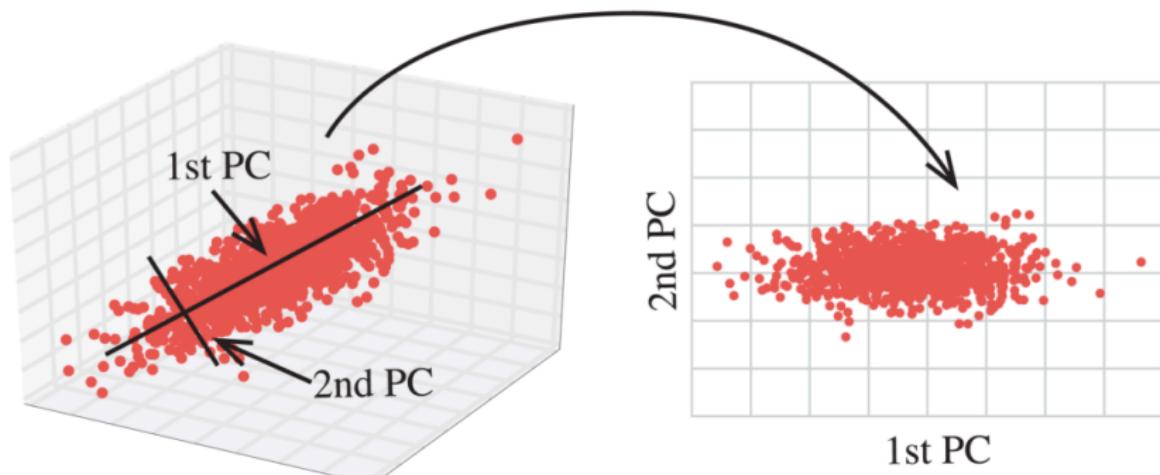
Principal Component Analysis (PCA)

PCA seeks a r -dimensional basis that best captures the *variance* in the data

First principal component (p. c.) = direction with largest projected variance

Second p. c. = orthogonal direction with largest projected variance

⋮



Computing the principal components

How do we get the first r principal components?

Assumptions:

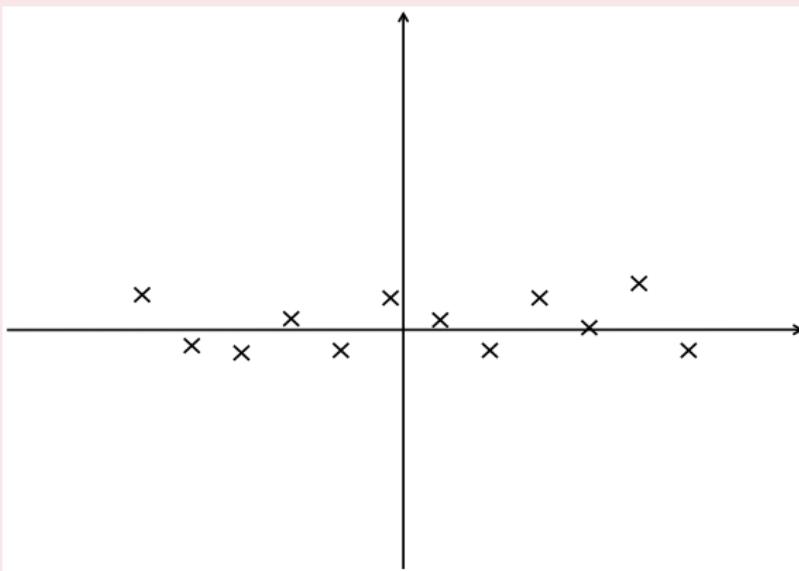
- data is given by matrix D , where the i -th row is the i -th input vector $\mathbf{x}_i \in \mathbb{R}^d$
- has been centered \Rightarrow mean μ of the data is $\mu = \sum_{i=0}^m \mathbf{x}_i = \mathbf{0}$ ($\mathbf{0}$ vector of dimension d)

To compute the first r principal components:

- ① compute the (sample) covariance matrix of (centered) data D : $\Sigma = D^T D$
- ② compute eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \lambda_d \geq 0$ of Σ (all eigenvalues of Σ are ≥ 0 since Σ is positive semidefinite);
- ③ let $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_d$ be the eigenvectors associated to $\lambda_1 \geq \lambda_2 \geq \dots \lambda_d$;
- ④ the first r principal components are $\mathbf{u}_1, \dots, \mathbf{u}_r$

Exercize

With respect to the dataset shown in the figure below: draw approximately the first and second right principal components. In a separate plot, show (approximately) the projection of the dataset on the first principal component.



PCA: Projected Variance

We now study some properties of the PCA.

Remember that

$$U_r = [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_r]$$

and

$$P_r = U_r U_r^T$$

Let A denote the *projected* dataset, containing points $\mathbf{a}_i = U_r^T \mathbf{x}_i$ for $i = 1, \dots, m$.

Let $\text{var}(A)$ denote the variance of the projected dataset A . PCA maximizes $\text{var}(A)$, i.e. the variance of the project dataset A (we will not see the proof). Can we compute $\text{var}(A)$?

Question: what is the projection of $\mu = \mathbf{0} \in \mathbb{R}^d$? **Answer:** $\mathbf{0} \in \mathbb{R}^r$

$$\begin{aligned}
\text{var}(A) &= \frac{1}{m} \sum_{i=1}^m \|\mathbf{a}_i - \mathbf{0}\|^2 \\
&= \frac{1}{m} \sum_{i=1}^m (U_r^T \mathbf{x}_i)^T (U_r^T \mathbf{x}_i) \\
&= \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i^T U_r U_r^T \mathbf{x}_i \\
&= \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i^T P_r \mathbf{x}_i \\
&= \frac{1}{m} \sum_{i=1}^m \left(\mathbf{x}_i^T \left(\sum_{j=1}^r \mathbf{u}_j \mathbf{u}_j^T \right) \mathbf{x}_i \right) \\
&= \sum_{j=1}^r \left(\frac{1}{m} \sum_{i=1}^m (\mathbf{u}_j^T \mathbf{x}_i) (\mathbf{x}_i^T \mathbf{u}_j) \right)
\end{aligned}$$

$$\begin{aligned}
 &= \sum_{j=1}^r \mathbf{u}_j^T \Sigma \mathbf{u}_j \\
 &= \sum_{j=1}^r \lambda_j \mathbf{u}_j^T \mathbf{u}_j \\
 &= \sum_{j=1}^r \lambda_j
 \end{aligned}$$

Note that in the process we have also shown that

$$\text{var}(A) = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i^T P_r \mathbf{x}_i \quad (1)$$

PCA: Mean Squared Error

Let's compute the Mean Squared Error (MSE) for PCA:

$$\begin{aligned} MSE &= \frac{1}{m} \sum_{i=1}^m \|\mathbf{x}_i - \mathbf{x}'_i\|^2 \\ &= \frac{1}{m} \sum_{i=1}^m (\mathbf{x}_i - \mathbf{x}'_i)^T (\mathbf{x}_i - \mathbf{x}'_i) \\ &= \frac{1}{m} \sum_{i=1}^m \left(\|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T \mathbf{x}'_i + (\mathbf{x}'_i)^T \mathbf{x}'_i \right) \\ &= \text{var}(D) + \frac{1}{m} \sum_{i=1}^m \left(-2\mathbf{x}_i^T P_r \mathbf{x}_i + (P_r \mathbf{x}_i)^T P_r \mathbf{x}_i \right) \\ &= \text{var}(D) + \frac{1}{m} \sum_{i=1}^m \left(-2\mathbf{x}_i^T P_r \mathbf{x}_i + \mathbf{x}_i^T P_r^2 \mathbf{x}_i \right) \end{aligned}$$

$$\begin{aligned}&= \text{var}(D) + \frac{1}{m} \sum_{i=1}^m \left(-2\mathbf{x}_i^T P_r \mathbf{x}_i + \mathbf{x}_i^T P_r \mathbf{x}_i \right) \\&= \text{var}(D) - \frac{1}{m} \sum_{i=1}^m \left(\mathbf{x}_i^T P_r \mathbf{x}_i \right) \\&= \text{var}(D) - \text{var}(A) \\&= \text{var}(D) - \sum_{j=1}^r \lambda_j\end{aligned}$$

⇒ PCA finds a basis of r vectors that minimizes MSE

PCA: Choice of r

How do we choose r ?

Note that $\text{var}(D) = \sum_{i=1}^d \lambda_i$ ($\text{var}(D)$ is invariant to base change and by running PCA with $r = d$).

Let $f(r)$ = fraction of variance of the data captured by PCA with r p. c., then:

$$f(r) = \frac{\sum_{i=1}^r \lambda_i}{\text{var}(D)} = \frac{\sum_{i=1}^r \lambda_i}{\sum_{i=1}^d \lambda_i}$$

Common Rule

Pick smallest r such that $f(r) \geq \alpha$ (usually $\alpha \geq 0.9$)