UNIVERSITÀ DEGLI STUDI DI PADOVA

NOTES OF

MACHINE LEARNING

(Version 09/02/2022)

Edited by
Stefano Ivancich

CONTENTS

Homeworks: https://github.com/ivaste/MachineLearningLAB

If you find errors, please report them here:
www.stefanoivancich.com
ivancich.stefano.1@gmail.com
The document will be updated as soon as possible

1. Probability review

Probability space has three components:

- **Sample space** Z: set of all possible outcomes of the random process modeled by the probability space. (Eg. dice rolling: $Z = \{1,2,\dots,6\}$)
- **Family** F of sets of **allowable events**, where each event A is a subset of Z: $A \subseteq Z$ (Eg. dice rolling: $F = \{\{1\},\{2\},\dots,\{6\}\}$)
- **Probability distribution** $D: F \to [0,1]$ that satisfies the following conditions:
    - $D[Z] = 1$
    - Let $E_1, E_2, \dots$ be any finite or countably infinite sequence of pairwise mutually disjoint events ($E_i \cap E_j = \emptyset$ for all $i \neq j$)
    $$D\left[\bigcup_{i\geq 1} E_i\right] = \sum_{i\geq 1} D[E_i]$$
    (Eg. dice rolling: $D(\{i\}) = \frac{1}{6}$)

Distributions and Probability:

- We use $z \sim D$ to say that event $z \in Z$ is sampled according to D
- Given the function $f: Z \to \{\text{true, false}\}$, we define the Probability of $f(z)$:
$$\mathbb{P}_{z\sim D}[f(z)] = D(\{z \in Z: f(z) = \text{true}\})$$
- An event $A \subseteq Z$ can be expressed using a function $\pi_A: Z \to \{\text{true, false}\}$. That is
$$A = \{z \in Z: \pi_A(z) = \text{true}\}$$
Where $\pi_A(z) = \text{true}$ if $z \in A$ otherwise $\pi_A(z) = \text{false}$.
In this case we have $\mathbb{P}[A] = \mathbb{P}_{z\sim D}[\pi_A(z)] = D(A)$

Independent Events: E and F are independent ($E \perp F$) iif $\mathbb{P}[E \cap F] = \mathbb{P}[E] * \mathbb{P}[F]$
$E_1, E_2, \dots, E_k$ are mutually independent iff for any subset $I \subseteq [1,k]$, $\mathbb{P}[\bigcap_{i\in I} E_i] = \prod_{i\in I} \mathbb{P}[E_i]$

Random Variable X_Z on a sample space Z is a function $X: z \in Z \to \mathbb{R}$

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

Description of R.V.:

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

Vector Valued R.V.: $\boldsymbol{X} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$

- X_1, X_2 discrete: $p_{\boldsymbol{X}}(x) = p_{X_1,X_2}(x_1,x_2) = \mathbb{P}_{\boldsymbol{X}}[X_1 = x_1, X_2 = x_2]$
- $X_1 = x_1, X_2 = x_2$ are joint events

Independence: discrete random variables X and Y are independent ($X \perp Y$) iif $\mathbb{P}[(X = x) \cap (Y = y)] = \mathbb{P}[X = x] * \mathbb{P}[Y = y]$ for all values x and y.
Discrete random variables $X_1, \dots, X_k$ are mutually independent iff for any subset $I \subseteq [1,k]$ and any values $x_i, i \in I$: $\mathbb{P}_{\boldsymbol{X}}(x) = \prod_{i\in I} \mathbb{P}[X_i = x] = \prod_{i\in I} p_{X_i}(x_i)$

Expectation of a discrete rv:

- **Mean**: $m_X = \mathbb{E}[X] = \sum_x x p_X(x)$
  - $\mathbb{E}[X_1 + X_2] = \mathbb{E}[X_1] + \mathbb{E}[X_2]$
  - $\mathbb{E}[g(X)] = \sum_x g(x) p_X(x)$ with $g(X)$ a function
- **Variance**: $Var[X] = \sigma_X^2 = \mathbb{E}[(X - m_X)^2] = \mathbb{E}[X^2] - m_X^2$
  - $Var[aX + b] = a^2 Var[X]$
  - $Var[X_1 + X_2] = Var[X_1] + Var[X_2] + 2\sigma_{X_1,X_2}$
- k-th moment: $\mathbb{E}[X^k]$
- **Covariance** $\sigma_{X_i,X_j} = Cov(X_i, X_j) = \mathbb{E}\left[\left(X_i - m_{X_i}\right)\left(X_j - m_{X_j}\right)\right]$
  - If X_1, X_2 are independent then $\sigma_{X_1,X_2} = 0$
  - If $\sigma_{X_1,X_2} = 0$ then $Var[X_1 + X_2] = Var[X_1] + Var[X_2]$

For a vector:

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

Conditional Probability: $\mathbb{P}[A|B] = \frac{\mathbb{P}[A \cap B]}{\mathbb{P}[B]}$. Well defined only if $\mathbb{P}[B] > 0$

- Bayes rule: $\mathbb{P}[A|B] = \frac{\mathbb{P}[B|A]\mathbb{P}[A]}{\mathbb{P}[B]}$
- Law of Total Probability $\mathbb{P}[A] = \sum_{i=1}^{n} \mathbb{P}[A|C_i]\mathbb{P}[C_i]$ where $C_1, \dots, C_n$ partition of Ω

Bernoulli rv of parameter $p \in [0,1]$:

- X=1 success, X=0, unsuccess, p= probability of success
- $p = \mathbb{P}[X = 1]$ and $\mathbb{P}[X = 0] = 1 - p$
- $\mathbb{E}[Be(p)] = p$
- $Var = p(1 - p)$

Binomial …..

Chebyshev's inequality: $\mathbb{P}[|X - \mu| > \varepsilon] \leq \frac{\sigma^2}{\varepsilon^2}$

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

Law of Large Numbers: $\lim_{n\to+\infty} \mathbb{P}\left[\left|\frac{1}{n}\sum X_i - \mu\right| > \varepsilon\right] = 0$

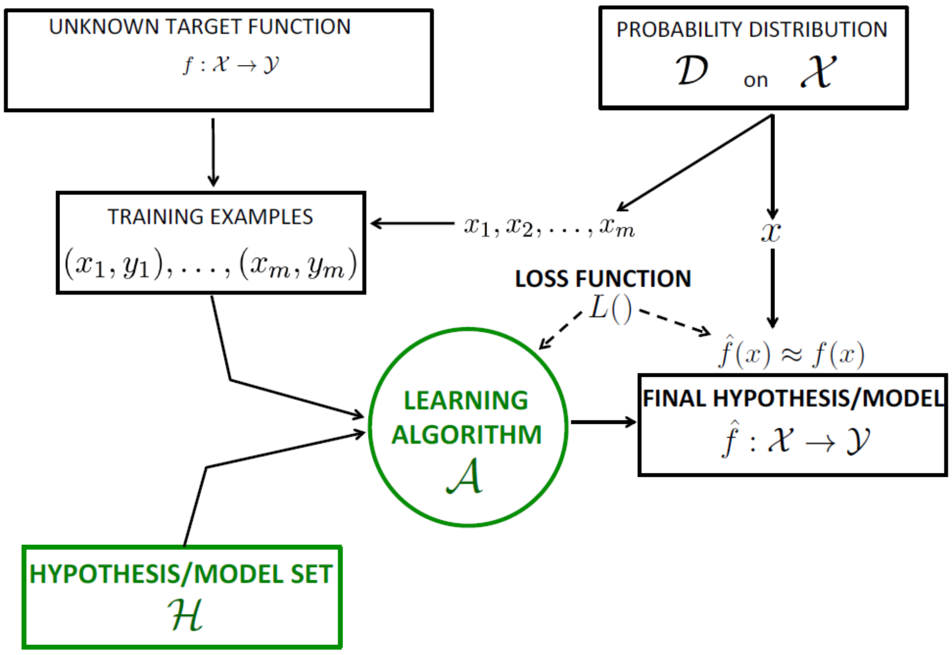
2.A Formal Learning Model

2.1. Formal Model

Formal Model (Statistical Learning):

- **Domain set** X, each domain point x is a vector of features (called instance).
- **Label set** Y. Often $\{-1,+1\}$ or $\{0,1\}$
- **Training data** $S = \big((x_1, y_1), \dots, (x_m, y_m)\big)$
    - Finite sequence of m labeled domain points
    - It's the learner input
- **Learner's Output** $h: X \to Y$ prediction rule (hypothesis, classifier, predictor). Produced by an algorithm $A(S)$
- **Data-generation model:** instances are generated by some probability distribution and labeled according to a function.
    - D probability distribution over X (not known by the learner)
    - labeling function $f: X \to Y$ (not known by the learner)
    - Each point in S is picked by sampling x_i according to D then label it as $y_i = f(x_i)$
- **Measures of success:** error of a classifier = probability it does not predict the correct label on a random data point generated by distribution D.
    - $D(A)$ = prob of observing a point $x \in A \subset X$
    - Sometimes A is an event expressed by the function $\pi: X \to \{0,1\}$ that is
      $$A = \{x \in X: \pi(x) = 1\}$$
      In this case $\mathbb{P}_{x\sim D}[\pi(x)] = D(A)$
    - **Loss** (error of the prediction rule h): $L_{D,f}(h) \overset{\text{def}}{=} \mathbb{P}_{x\sim D}[h(x) \neq f(x)] \overset{\text{def}}{=} D(\{x: h(x) \neq f(x)\})$ is the probability of randomly choosing an example x for which $h(x) \neq f(x)$

Learning process:



2.2. ERM (Empirical Risk Minimization)

Empirical Risk Minimization
A learning algorithm receives as input a training set S, sampled from an unknown distribution D, labeled by some target function f and output a predictor $h_S : X \to Y$
Its goal is to find h_S that minimize the generalization error $L_{D,f}(h) = D(\{x : h(x) \neq f(x)\})$
But $L_{D,f}(h)$ is unknown. So, we minimize the training error.
Training error (empirical risk): $L_S(h) = \frac{|\{i : h(x_i) \neq y_i, 1 \leq i \leq m\}|}{m}$ Fraction of times that the predictor makes an error on the training set.

Empirical Risk Minimization with Inductive Bias
Problem: Overfitting occurs when our hypothesis fits the training data "too well".
Solution: apply the ERM learning rule over a restricted set of hypotheses. (**Inductive Bias**)
Hypothesis class H: a set of predictors $h : X \to Y$ chosen in advance.
ERM_H learner uses the ERM rule to choose a predictor $h \in H$

$$\mathrm{ERM}_H(S) \in \operatorname*{argmin}_{h \in H} L_S(h)$$

So, if H is **a finite class** then ERM_H will not overfit.
Assumptions:

- **Realizability**: $\exists h^* \in H$ s.t. $L_{D,f}(h^*) = 0$. This implies that $L_S(h^*) = 0$
  Informally: the label is fully determined by the instance $\boldsymbol{x}$
- **i.i.d.:** The examples in the training set are independently and identically distributed (i.i.d.) according to the distribution D. That is $S \sim D^m$

2.3. (Simplied) PAC learning

(Simplied) PAC learning
Probably Approximately Correct (PAC) learning
Since the training data comes from D:

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

Parameters:

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

We want ϵ and δ to be small.

Theorem:
LET

- H be a finite hypothesis class,
- $\delta, \epsilon \in (0,1)$ and
- $m = |S| \geq \frac{\ln(|H|/\delta)}{\epsilon} \in \mathbb{N}$

THEN:

- $\forall f$, and $\forall D$, for which the realizability assumption holds,
- with probability $\geq 1 - \delta$ over the choice of an i.i.d. sample S of size m,
- we have that for every **ERM** hypothesis, h_S, it holds that $L_{D,f}(h_S) \leq \epsilon$

Explanations:

- For a sufficiently large m, the ERM_H rule over a finite hypothesis class will be probably (with confidence $1-\delta$) approximately (up to an error of ϵ) correct.
- With finite hypothesis class, I can always (with prob $\geq 1-\delta$) find a good hypothesis ($L_{D,f}(h_S) \leq \epsilon$) if I have enough data ($m = |S| \geq \frac{\ln(|H|/\delta)}{\epsilon}$)

Proof: (pdf 5, lec 5)

- Let $S_x = \{x_1, \dots, x_m\}$ be the instances in the training set S
- We want an upper bound to $D^m\big(\{S_x : L_{D,f}(h_S) > \epsilon\}\big)$
- Let:
  - $H_B = \{h \in H : L_{D,f}(h) > \epsilon\}$ (bad Hypothesis)
  - $M = \{S_x : \exists h \in H_B, L_S(h) = 0\}$ (misleading samples)
- Since we have the realizability assumption: $L_S(h_S) = 0$, this implies that $L_{D,f}(h_S) > \epsilon$ only if some $h \in H_B$ has $L_S(h) = 0$.
- That is, our training data must be in the set M: this only happens if $\{S_x : L_{D,f}(h_S) > \epsilon\} \leq M$
- Different way to write M is: $M = \bigcup_{h \in H_B} \{S_x : L_S(h) = 0\}$
- Therefore $D^m\big(\{S_x : L_{D,f}(h_S) > \epsilon\}\big) \leq D^m(M) = D^m\big(\bigcup_{h \in H_B} \{S_x : L_S(h) = 0\}\big)$
  - $\leq \sum_{h \in H_B} D^m\big(\{S_x : L_{D,f}(h_S) > \epsilon\}\big)$ (Union bound) **(1)**
- Now let's fix the hypothesis $h \in H_B : L_S(h) = 0 \text{ IIF } \forall i = 1, \dots, m : h(x_i) = f(x_i)$
- Therefore, we have that $D^m(\{S_x : L_S(h) = 0\}) = D(\{S_x : \forall i = 1, \dots, m : h(x_i) = f(x_i)\})$
  - $= \prod_{i=1}^{m} D(\{x_i : h(x_i) = f(x_i)\})$ Since those samples are iid. **(2)**
- Consider $i, 1 \leq i \leq m : D(\{x_i : h(x_i) = f(x_i)\}) = 1 - D(\{x_i : h(x_i) \neq f(x_i)\})$
  - Where $D(\{x_i : h(x_i) \neq f(x_i)\}) = L_{D,f}(h) = \mathbb{P}_{x \sim D}[h(x) \neq f(x)]$ is the generalization error.
  - So, $1 - D(\{x_i : h(x_i) \neq f(x_i)\}) = 1 - L_{D,f}(h) \leq 1 - \epsilon \leq e^{-\epsilon}$
- Combining this with (2): $D^m(\{S_x : L_S(h) = 0\}) \leq \prod_{i=1}^{m} e^{-\epsilon} = e^{-\epsilon m}$
- Combining with (1): $D^m\big(\{S_x : L_{D,f}(h_S) > \epsilon\}\big) \leq \sum_{h \in H_B} e^{-\epsilon m} = |H_B| e^{-\epsilon m} \leq |H| e^{-\epsilon m}$
- Now, given the choice of m: $D^m\big(\{S_x : L_{D,f}(h_S) > \epsilon\}\big) \leq |H| e^{-\epsilon \ln\left(\frac{|H|}{\delta}\right) * \frac{1}{\epsilon}} = |H| * \frac{\delta}{|H|} = \delta$

2.4. PAC Learning

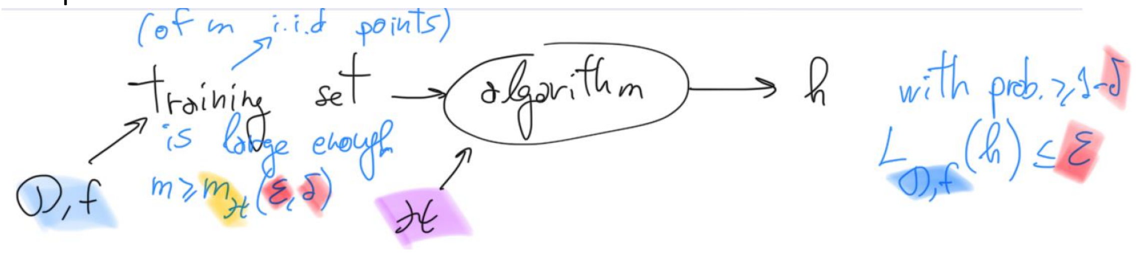
Probably Approximately Correct (PAC) learning.

A hypothesis class H is **PAC learnable** if there exist a function $m_H: (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following property:
$\forall \epsilon, \delta \in (0,1), \forall D$ over $X, \forall f: X \to \{0,1\}$

- IF the realizability assumption holds with respect to H, D, f,
- THEN when running the learning algorithm on $m \geq m_H(\epsilon, \delta)$ i.i.d. examples generated by D and labeled by f,
  the algorithm returns a hypothesis h such that, with probability $\geq 1 - \delta$ (over the choice of examples): $L_{D,f}(h) \leq \epsilon$

Approximation Parameters:

- accuracy parameter ϵ determines how far the output classifier can be from the optimal one.
- confidence parameter δ indicates how likely the classifier is to meet that accuracy requirement.



Sample Complexity: how many examples are required to guarantee a probably approximately correct solution.

Every finite hypothesis class is PAC learnable with sample complexity $m_H(\epsilon, \delta) \leq \left\lceil \frac{\log(|H|/\delta)}{\epsilon} \right\rceil$

2.5. Agnostic PAC Learning

Agnostic PAC Learning: we relax the realizability assumption by replacing the "target labeling function" with a more flexible notion, a data-labels generating distribution.

Relaxation: D over $X \times Y$ is a joint distribution over domain points and labels. Composed of two parts:

- D_x (marginal) distribution over unlabeled domain points
- $D\big((x,y)\big|x\big)$ conditional distribution over labels for each domain point

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

True error: $L_D(h) \overset{\text{def}}{=} \mathbb{P}_{(x,y)\sim D}[h(x) \neq y] = D(\{(x,y): h(x) \neq y\})$

Training error (empirical risk): $L_S(h) \overset{\text{def}}{=} \frac{|\{i: h(x_i) \neq y_i, 1 \leq i \leq m\}|}{m}$ (not changed after the relaxation)

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

Essentially $\mathbb{E}[L_s(h)] = L_D(h)$

Goal: find $h: X \to Y$ that minimizes $L_D(h)$

Bayes Optimal Predictor: the best label predicting function is $f_D(x) = \begin{cases} 1, & \mathbb{P}[y = 1|x] \geq 1/2 \\ 0, & \text{otherwise} \end{cases}$

No other classifier has a lower error. But since we do not know D, we cannot use this optimal predictor f_D.

H is **Agnostic PAC Learnable** if there exist a function $m_H: (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following property: $\forall \epsilon, \delta \in (0,1)$ and $\forall D$ over $X \times Y$,

- when running the learning algorithm on $m \geq m_H(\epsilon, \delta)$ i.i.d. examples generated by D,
- the algorithm returns a hypothesis h such that, with probability $\geq 1 - \delta$ (over the choice of m training samples): $L_D(h) \leq \min_{h' \in H} L_D(h') + \epsilon$

Agnostic PAC learning generalizes the definition of PAC learning.

General Loss Functions: any function $l: H \times Z \to \mathbb{R}_+$ where H is any hypotheses set and Z is some domain. Tells us how much we lose by predicting $h(x)$ instead of the correct label y.
Risk function = expected loss of a hypothesis $h \in H$ with respect to D over Z: $L_D(h) \overset{\text{def}}{=} \mathbb{E}_{z\sim D}[l(h,z)]$
Empirical risk: the expected loss over a given sample $S = (z_1, \dots, z_m) \in Z^m$: $L_S(h) \overset{\text{def}}{=} \frac{1}{m}\sum_{i=1}^{m} l(h, z_i)$ where $z_i = (x_i, y_i)$

Common loss functions:

- **0-1 loss**: $Z = X \times Y$. Used in binary or multiclass classification
$$l_{0-1}\big(h,(x,y)\big) \overset{\text{def}}{=} \begin{cases} 0, & h(x) = y \\ 1, & h(x) \neq y \end{cases}$$
- **Squared loss:** $Z = X \times Y$. Used in regression.
$$l_{sq}\big(h,(x,y)\big) \overset{\text{def}}{=} (h(x) - y)^2$$

Agnostic PAC Learnability for General Loss Functions:
H is agnostic PAC learnable wrt a set Z and a loss function $l: H \times Z \to \mathbb{R}_+$
IF there exist a function $m_H: (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following property: $\forall \epsilon, \delta \in (0,1)$ and $\forall D$ over Z,

- when running the learning algorithm on $m \geq m_H(\epsilon, \delta)$ i.i.d. examples generated by D,
- the algorithm returns a hypothesis h such that, with probability $\geq 1 - \delta$:
$$L_D(h) \leq \min_{h' \in H} L_D(h') + \epsilon \text{ where } L_D(h') = \mathbb{E}_{z\sim D}[l(h,z)]$$

3. Uniform Convergence

Uniform convergence is a tool used to show that any finite class is learnable in the agnostic PAC model with general loss functions, as long as the range loss function is bounded.

Uniform convergence: the empirical risk (training error) of all members of H are good approximations of their true risk (generalization error)

A training set S is called **ϵ-representative** if $\forall h \in H, |L_S(h) - L_D(h)| \leq \epsilon$

If S is **$\frac{\epsilon}{2}$-representative**, then any output of the $\mathrm{ERM}_H(S)$ (eg. $h_s \in \underset{h\in H}{\operatorname{argmin}} L_S(h)$) satisfies:

$$L_D(h_s) \leq \min_{h\in H} L_D(h) + \epsilon$$

It means that ERM return a good hypothesis.

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

$$L_{\mathcal{D}}(h_s) \leq L_S(h_s) + \frac{\varepsilon}{2}$$ S is $\frac{\varepsilon}{2}$-representative

$$\leq L_S(h) + \frac{\varepsilon}{2}$$ since h_s is picked by ERM ($\Rightarrow L_S(h_s) \leq L_S(h)$)

$$\leq L_{\mathcal{D}}(h) + \frac{\varepsilon}{2} + \frac{\varepsilon}{2}$$ S is $\frac{\varepsilon}{2}$-representative

$$\leq L_{\mathcal{D}}(h) + \varepsilon$$

This true also for $h = \arg\min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h')$

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

H has the **uniform convergence property** IF exists a function $m_H^{UC}: (0,1)^2 \to \mathbb{N}$ such that $\forall \epsilon, \delta \in (0,1)$ and $\forall D$ over Z

- IF S is a sample of $m \geq m_H^{UC}(\epsilon, \delta)$ i.i.d. examples generated by D,
- THEN with probability $\geq 1 - \delta$, S is ϵ-representative.

If H has the uniform convergence property with a function m_H^{UC} then this class is agnostically PAC learnable with the sample complexity $m_H(\epsilon, \delta) \leq m_H^{UC}(\epsilon/2, \delta)$.
This means that ERM_H is a successful agnostic PAC learner for H.

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 $\epsilon > 0$:

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

P(Average of the observations – expected value of each observation>…)

Every finite hypothesis class is agnostic PAC learnable under some restrictions on the loss.
LET:

- H be a finite hypothesis class,
- Z be a domain,
- $l: H \times Z \to [0,1]$

THEN:

- H enjoys the uniform convergence property with sample complexity
$$m_H^{UC}(\epsilon, \delta) \leq \left\lceil \frac{\log(2|H|/\delta)}{2\epsilon^2} \right\rceil$$
- H is agnostically PAC learnable using the ERM algorithm with sample complexity
$$m_H(\epsilon, \delta) \leq m_H^{UC}(\epsilon/2, \delta) \leq \left\lceil \frac{2\log(2|H|/\delta)}{\epsilon^2} \right\rceil$$

Idea of the proof:

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

Proof: (lecture 6)

Fix $\varepsilon, \delta \in (0,1)$. We need sample size m such that, for any $\mathcal{D}$, with probability $\geq 1-\delta$ (on the choice of $S=(z_1, z_2, \dots, z_m)$, $z_i=(x_i,y_i)\ \forall 1 \leq i \leq m$, sampled i.i.d from $\mathcal{D}$) we have:

$$\text{for all } h \in \mathcal{H}: |L_S(h) - L_{\mathcal{D}}(h)| \leq \varepsilon.$$

That: $\mathcal{D}^m(\{S : \forall h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| \leq \varepsilon\}) \geq 1-\delta.$

Equivalently, we need to show:

$$\underbrace{\mathcal{D}^m(\{S : \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon\})}_{(\star)} < \delta.$$

We have

$$\{S : \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon\} = \bigcup_{h \in \mathcal{H}} \{S : |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon\}$$

Then

$$(\star) \overset{\text{(by union bound)}}{\leq} \sum_{h \in \mathcal{H}} \mathcal{D}^m(\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon\}) \quad (\star\star)$$

Now we want to bound each term in $(\star\star)$

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

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

Important note: each z_i is sampled i.i.d. from $\mathcal{D}$

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

Therefore $\mathbb{E}[L_S(h)] \underset{\text{by def of } L_S(h)}{=} \mathbb{E}\left[\frac{1}{m} \sum_{i=1}^m \ell(h,z_i)\right]$

by linearity of expectation $\rightarrow$ $= \frac{1}{m} \sum_{i=1}^m \underbrace{\mathbb{E}[\ell(h,z_i)]}_{L_{\mathcal{D}}(h)}$

$$= \frac{1}{m} \cdot \sum_{i=1}^m L_{\mathcal{D}}(h) = \frac{1}{m} \cdot m L_{\mathcal{D}}(h)$$

$$= L_{\mathcal{D}}(h)$$

Let $\underset{\text{because } z_i \sim \mathcal{D}}{\theta_i}$ be the r.v. given by $\ell(h, \underset{\text{i-th point } (x_i,y_i) \text{ in training set } S}{z_i})$

Since h is fixed, z_i are sampled i.i.d. from $\mathcal{D}$
$\Rightarrow \theta_1, \theta_2, \dots, \theta_m$ are i.i.d. r.v.

Note that: $\boxed{L_S(h) = \frac{1}{m}\sum_{i=1}^{m} \theta_i}$; let's define $\boxed{\mu = L_{\mathcal{D}}(h)}$.

Given assumption:
$\ell : \mathcal{H} \times Z \to [0,1] \Rightarrow \theta_i \in [0,1) \quad \forall\ i = 1, \dots, m$

We can apply Hoeffding's inequality with $a_i = 0, b_i = 1$ $\forall i = 1, \dots, m$:

$$\mathcal{D}^m(\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon\}) = \Pr\left[\left|\frac{1}{m}\left(\sum_{i=1}^{m}\theta_i\right) - \mu\right| > \varepsilon\right]$$

$$\underset{\text{by Hoeffding's inequality}}{\leq} 2 \cdot e^{-2m\varepsilon^2}$$

Combining the above with (A A):

$$\mathcal{D}^m(\{S : \exists\, h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon\}) \leq \sum_{h \in \mathcal{H}} 2e^{-2m\varepsilon^2} \overset{\text{since } |\mathcal{H}| < +\infty}{=} 2|\mathcal{H}|e^{-2m\varepsilon^2}$$

By choosing $\underbrace{m \geq \log\left(\frac{2|\mathcal{H}|}{\delta}\right) / (2\varepsilon^2)}$ then

for example:
$m = \left\lceil \log\left(\frac{2|\mathcal{H}|}{\delta}\right) / (2\varepsilon^2) \right\rceil$

$$\mathcal{D}^m(\{S : \exists\, h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon\}) \leq 2|\mathcal{H}|e^{-\cancel{2\varepsilon^2}\log\left(\frac{2|\mathcal{H}|}{\delta}\right)\cdot\frac{1}{\cancel{2\varepsilon^2}}}$$

$$= 2|\mathcal{H}| \cdot \frac{\delta}{2|\mathcal{H}|} = \delta$$

$\square$

4. Bias-Complexity Tradeoff

The No-Free-Lunch Theorem:

LET:

- A be any learning algorithm for the task of binary classification with respect to the 0 − 1 loss over a domain X.
- training set size $m < \frac{|X|}{2}$

THEN ∃ a distribution D over $X \times \{0,1\}$ such that:

- $\exists f: X \to \{0,1\}$ with $L_D(f) = 0$
- With probability of at least 1/7 over the choice of $S \sim D^m$ we have that $L_D\big(A(S)\big) \geq 1/8$

This theorem states that for every learner, there exists a task on which it fails, even though that task can be successfully learned by another learner.

H is **NOT PAC learnable** when X is an infinite domain and H is the set of all function $X \to \{0,1\}$

So, when approaching a particular learning problem, we should have some prior knowledge on D.
Types of such prior knowledge:

- D comes from some specific parametric family of distributions.
- Exists h in some predefined hypothesis class H, such that $L_D(h) = 0$.
- A softer type of prior knowledge on D is assuming that $\min_{h \in H} L_D(h)$ is small.

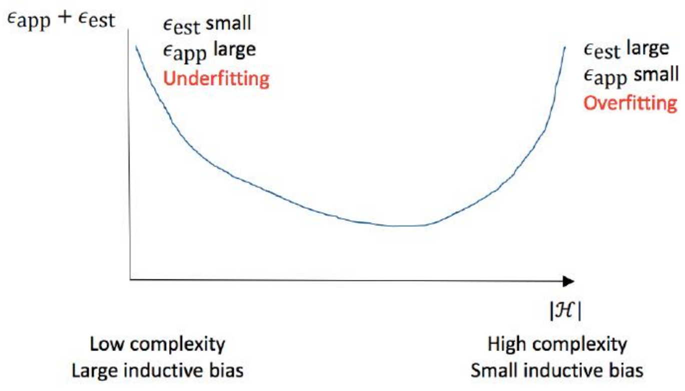
Trade-off: We would like H large so it will contain a lot of h with small $L_D(h)$ but for the no free lunch theorem H cannot be too large.

We decompose the error of an ERM algorithm over a class H into two components:

$$L_D(h_S) = \epsilon_{app} + \epsilon_{est}$$

- **Approximation error** (**Bias**) $\epsilon_{app} = \min_{h \in H} L_D(h)$. reflects the quality of our prior knowledge, measured by the minimal risk of a hypothesis in our hypothesis class.
  To decrease it: choose H larger
- **Estimation error** $\epsilon_{est} = L_D(h_S) - \min_{h \in H} L_D(h)$. Error due to overfitting.
  To decrease it: choose H smaller

These two terms imply a tradeoff between choosing a more complex H or a less complex H.



The generalization error $L_D(h)$ for a function h is estimated using a test set.

5.VC-Dimension

Goal: figure out which classes H are PAC learnable.

Restriction of H to C: $H_C = \{(h(c_1), \dots, h(c_m)): h \in H\}$ where

- $H =$ class of functions $h: X \to \{0,1\}$
- $C = \{c_1, \dots, c_m\} \subset X$

Each function $C \to \{0,1\}$ in H_C is represented as a vector in $\{0,1\}^{|C|}$
H_C is the set of functions from C to $\{0,1\}$ that can be derived from H.
Note: $0 \leq |H_C| \leq 2^m$

H **shatters** the set C: if H_C contains all $2^{|C|}$ functions $C \to \{0,1\}$

VC-dimension of H: $\mathrm{VCdim}(H)$ is the maximal size of a set $C \subset X$ that can be shattered by H.
VC-dimension measures the complexity of H, how large a dataset that is perfectly classified using the functions in H can be.

H is not PAC learnable if $\mathrm{VCdim}(H) = +\infty$
A finite VCdimension guarantees learnability. If $|H| < +\infty$ then $\mathrm{VCdim}(H) \leq \log_2|H|$

To **show that $\mathbf{VCdim}(H) = d$**: show that

- there exists a set C of size d which is shattered by H ($\mathrm{VCdim}(H) \geq d$, take d points and for each possible labeling of the points (0,1) 2^d prove that you can place those point in the plot and label that with that combination of 0/1 (shatters))
- every set of size $d+1$ is not shattered by H ($\mathrm{VCdim}(H) \leq d$, take $d+1$ points and prove that there is at least one labelling not possible, it means that there is no way to place the point in the graph to obtain that labeling)

Threshold Functions:

- $H = \{h_a: a \in \mathbb{R}\}$ where $h_a: \mathbb{R} \to \{0,1\}$, $h_a(x) = \begin{cases} 1, & x < a \\ 0, & x \geq a \end{cases}$
- $\mathrm{VCdim}(H) = 1$

Intervals:

- $H = \{h_{a,b}: a, b \in \mathbb{R}, a < b\}$ where $h_{a,b}: \mathbb{R} \to \{0,1\}$, $h_{a,b}(x) = \begin{cases} 1, & a < x < b \\ 0, & otherwise \end{cases}$
- $\mathrm{VCdim}(H) = 2$

Axis Aligned Rectangles:

- $H = \{h_{(a_1,a_2,b_1,b_2)}: a_1, a_2, b_1, b_2 \in \mathbb{R}, a_1 \leq a_2, b_1 \leq b_2\}$
- $h_{(a_1,a_2,b_1,b_2)}(x_1, x_2) = \begin{cases} 1, & a_1 \leq x_1 \leq a_2, b_1 \leq x_2 \leq b_2 \\ 0, & otherwise \end{cases}$
- $\mathrm{VCdim}(H) = 4$

Convex Sets:

- $h: \mathbb{R}^2 \to \{0,1\}$, $h(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in S \\ 0, & otherwise \end{cases}$ where S is a convex subset of $\mathbb{R}^2$
- $\mathrm{VCdim}(H) = +\infty$

The Fundamental Theorem of Statistical learning: a class of infinite VC-dimension is not learnable.
Let

- H be a hypothesis class of functions from a domain X to $\{0, 1\}$
- loss function be the 0 – 1 loss.
- Assume that $VCdim(H) = d < +\infty$.

Then, there are absolute constants C_1, C_2 such that:

- H has the uniform convergence property with sample complexity
$$C_1 \frac{d + \log(1/\delta)}{\epsilon^2} \leq m_H^{UC}(\epsilon, \delta) \leq C_2 \frac{d + \log(1/\delta)}{\epsilon^2}$$
- H is agnostic PAC learnable with sample complexity
$$C_1 \frac{d + \log(1/\delta)}{\epsilon^2} \leq m_H(\epsilon, \delta) \leq C_2 \frac{d + \log(1/\delta)}{\epsilon^2}$$

Equivalently:

- LET: $H, \mathrm{VCdim}(H) < +\infty$.
- THEN: with probability $\geq 1 - \delta$ (over $S \sim D^m$) we have:
$$\forall h \in H, L_D(h) \leq L_S(h) + C \sqrt{\frac{\mathrm{VCdim}(H) + \log(1/\delta)}{2m}}$$
Where C is universal constant
We can use ERM rule to find the $h \in H$ that minimizes the upper bound.

If H is the class of **halfspaces** in $\mathbb{R}^d$ then $\mathbf{VCdim}(\boldsymbol{H}) = \boldsymbol{d}$

6. Linear Models

Halfspaces, linear regression predictors, and logistic regression predictors are hypothesis classes. We use the ERM approach to learn linear predictors.

Class of affine functions $L_d = \{h_{\boldsymbol{w},b} : \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$ where $h_{\boldsymbol{w},b}(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = \left(\sum_{i=1}^{d} w_i x_i\right) + b$.
Written also as:

- $L_d = \{\boldsymbol{x} \to \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b : \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$
- $h_{\boldsymbol{w},b}(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = \langle \boldsymbol{w}', \boldsymbol{x}' \rangle$ where $\boldsymbol{w}' = (b, w_1, \dots, w_d), \boldsymbol{x}' = (1, x_1, \dots, x_d)$

Linear models hypothesis class: $H: \phi \circ L_d$ where $\phi: \mathbb{R} \to Y$

- $h \in H$ is $h: \mathbb{R}^d \to Y$
- ϕ depends on the learning problem
  - Binary classification $Y = \{-1,1\}$, $\phi(z) = \text{sign}(z)$
  - Regression $Y = \mathbb{R}$, $\phi(z) = z$

6.1. Linear Regression

Used to learn a linear function $h: \mathbb{R}^d \to \mathbb{R}$
Hypothesis class: $H_{reg} = L_d = \{\boldsymbol{x} \to \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b : \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$
Squared-loss function: $l(h, (\boldsymbol{x}, y)) \overset{\text{def}}{=} (h(\boldsymbol{x}) - y)^2$
Empirical risk=training error= **Mean Squared Error** $L_S(h) = \frac{1}{m}\sum_{i=1}^{m}(h(\boldsymbol{x_i}) - y_i)^2$

Least Squares: algorithm that solves the ERM problem for the hypothesis class of linear regression predictors with respect to the squared loss.
Best hypothesis: $\underset{\boldsymbol{w}}{\text{argmin}}\, L_S(h_{\boldsymbol{w}}) = \underset{\boldsymbol{w}}{\text{argmin}} \frac{1}{m}\sum_{i=1}^{m}(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - y_i)^2$ (We want to find $\boldsymbol{w}$)
Residual Sum of Squares $\text{RSS}(\boldsymbol{w}) = \sum_{i=1}^{m}(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - y_i)^2 = (\boldsymbol{y} - \boldsymbol{Xw})^T(\boldsymbol{y} - \boldsymbol{Xw})$

RSS: Matrix Form

Let

$$\mathbf{X} = \begin{bmatrix} \cdots & \mathbf{x}_1 & \cdots \\ \cdots & \mathbf{x}_2 & \cdots \\ \cdots & \vdots & \cdots \\ \cdots & \mathbf{x}_m & \cdots \end{bmatrix} \qquad \mathbf{X}\text{: } \textit{design matrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

We want to find the $\boldsymbol{w}$ that minimize RSS: so we compute $\frac{\partial RSS(\boldsymbol{w})}{\partial w} = 0$

- $-2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{Xw}) = 0$ is equivalent to $\boldsymbol{X^T X w} = \boldsymbol{X^T y}$

Let $A = \boldsymbol{X^T X} = \sum_{i=1}^{m} x_i x_i^T$, $\boldsymbol{b} = \boldsymbol{X}^T\boldsymbol{y} = \sum_{i=1}^{m} y_i x_i$

- if A is invertible: $\boldsymbol{w} = A^{-1}\boldsymbol{b} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$
- if A is not invertible: $\boldsymbol{w} = A^{\dagger}\boldsymbol{b}$ where $A^{\dagger}$ is the Moore-Penrose generalized inverse.

Polynomial models:

- Assume $X = \mathbb{R}$
- Polynomial of degree r: $w_0 * 1 + w_1 * x + w_2 * x^2 + \cdots + w_r * x^r$

Assume: $\mathcal{X} = \mathbb{R}$

polynomial of degree r: $w_0 \cdot 1 + w_1 \cdot x + w_2 \cdot x^2 + \ldots + w_r \cdot x^r$

Given $x \in \mathbb{R}$, compute the vector: (feature expansion)

$$\vec{x}\,' = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \\ \vdots \\ x^{r-1} \\ x^r \end{bmatrix} \Rightarrow \vec{w} = [w_0, \ldots, w_r]^T \Rightarrow \langle \vec{x}\,', \vec{w} \rangle =$$

$$= w_0 + w_1 x + w_2 x^2 + \ldots + w_{r-1} x^{r-1} + w_r x^r$$

$\Rightarrow$ the hypothesis class of linear models for $\vec{x}\,'$ corresponds to the hyp. class of polynomials of degree r for x.

- $\boldsymbol{x}'$ is the instance (Feature expansion): we pick $\boldsymbol{x}$ and we apply a transformation to get $\boldsymbol{x}'$ so we can use linear regression formulas.

If $\boldsymbol{x} \in \mathbb{R}^d$

feature expansion:

$$\vec{x}\,' = \begin{bmatrix} 1 \\ x_0 \\ x_0^2 \\ \vdots \\ x_0^r \\ x_1 \\ x_1^2 \\ \vdots \\ x_1^r \\ \vdots \\ x_{d-1} \\ \vdots \\ x_{d-1}^r \end{bmatrix}$$

Different feature expansion:

$\vec{x} \in \mathbb{R}^3$, $\vec{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$, $r = 2$

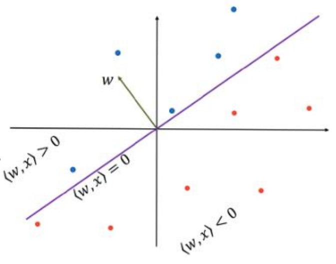
$$\vec{x}\,' = \begin{bmatrix} 1 \\ x_0 \\ x_1 \\ x_2 \\ x_0^2 \\ x_1^2 \\ x_2^2 \\ x_0 x_1 \\ x_0 x_2 \\ x_1 x_2 \end{bmatrix} \Rightarrow \text{build linear models for } \vec{x}\,'$$

Coefficient of determination $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$

- $SS_{res} = \frac{1}{m}\sum_{i=1}^{m}(h(\boldsymbol{x_i}) - y_i)^2$ Sum of squares residual
- $SS_{tot} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \bar{y})^2$, where $\bar{y}$ is the average of the y_i. Total sum of squares.

In regression, the R^2 is a statistical measure of how well the regression predictions approximate the real data points. An R^2 of 1 indicates that the regression predictions perfectly fit the data.
Is a measure how well h performs against the best naïve predictor.

6.2. Linear Classification



Used in binary classification problems $h: \mathbb{R}^d \to \{-1,1\}$

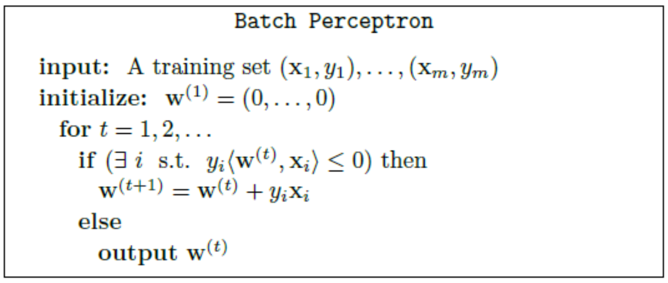
Class of halfspaces: $H_d = \text{sign} \circ L_d = \{\boldsymbol{x} \to \text{sign}(\langle \boldsymbol{w}, \boldsymbol{x}\rangle + b) : \boldsymbol{w} \in \mathbb{R}^d\}$

The instances that are "above" the hyperplane are labeled positively. Instances that are "below" the hyperplane are labeled negatively.

Loss function: 0-1

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

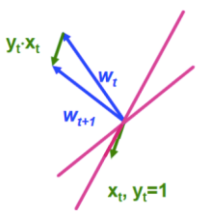
Perceptron algorithm: algorithm that find a good hypothesis implementing the ERM rule



```
Batch Perceptron

input:  A training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
initialize:  $\mathbf{w}^{(1)} = (0, \ldots, 0)$
  for $t = 1, 2, \ldots$
    if $(\exists\ i$ s.t. $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \le 0)$ then
      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$
    else
      output $\mathbf{w}^{(t)}$
```

Interpretation of update:



Note that:

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

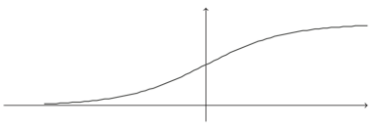
$\Rightarrow$ update guides $\mathbf{w}$ to be "more correct" on $(\mathbf{x}_i, y_i)$.

If $(x_i, y_1), \ldots, (x_m, y_m)$ is linearly separable, $B = \min\{\|\boldsymbol{w}\| : y_i \langle \boldsymbol{w}, x_i \rangle \geq 1\ \forall i, i = 1, \ldots, m\}$, $R = \max_{\text{i}} \|x_i\|$. The algorithm stops after at most $(RB)^2$ iterations

For separable data: convergence is guaranteed, potentially multiple solutions.
For non-separable data: run for some time and keep best solution found up to that point

6.3. Logistic Regression

Used to learn a function $h: \mathbb{R}^d \to [0,1]$, for classification tasks, is interpreted as the probability that the label of x is 1.



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

Hypothesis class: $H_{sig} = \phi_{sig} \circ L_d = \left\{\boldsymbol{x} \to \phi_{sig}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle) : \boldsymbol{w} \in \mathbb{R}^d\right\}$

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{1 + \exp(-\langle \boldsymbol{w}, \boldsymbol{x} \rangle)} \in [0,1]$$

$$\frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) = \frac{e^{-x}}{(1+e^{-x})^2}$$

Difference with halfspaces: when $\langle \boldsymbol{w}, \boldsymbol{x} \rangle \approx 0$ halfspaces are 1 or -1, logistic is ½ (uncertain)

Loss function: $l\big(h_{\boldsymbol{w}}, (\boldsymbol{x}, y)\big) = \log(1 + \exp(-y\langle \boldsymbol{w}, \boldsymbol{x} \rangle))$

If $y = +1$ then $h_{\boldsymbol{w}}(\boldsymbol{x})$ is large. If $y = -1$ then $h_{\boldsymbol{w}}(\boldsymbol{x})$ is small

ERM problem associated with logistic regression is $\underset{\boldsymbol{w} \in \mathbb{R}^d}{\text{argmin}}\left(\sum_{i=1}^m \log(1 + \exp(-y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle))\right)$

ERM formulation is the same as the one arising from Maximum Likelihood Estimation.
MLE is a statistical approach for finding the parameters that maximize the joint probability of a given dataset assuming a specific parametric probability function.

- Given training set $S = \big((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\big)$, assume each $(\boldsymbol{x}_i, y_i)$ is iid from some probability distribution of parameters θ (sometimes $S = (x_1, \ldots, x_m)$)
- Consider $\mathbb{P}[S|\theta]$ (likelihood of data given the parameters)
- Log likelihood: $L(S; \theta) = \log(\mathbb{P}[S|\theta])$
- Maximum likelihood estimator: $\theta = \arg\max_{\theta} L(S; \theta)$

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:

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

$\Rightarrow$ MLE solution is equivalent to ERM solution!

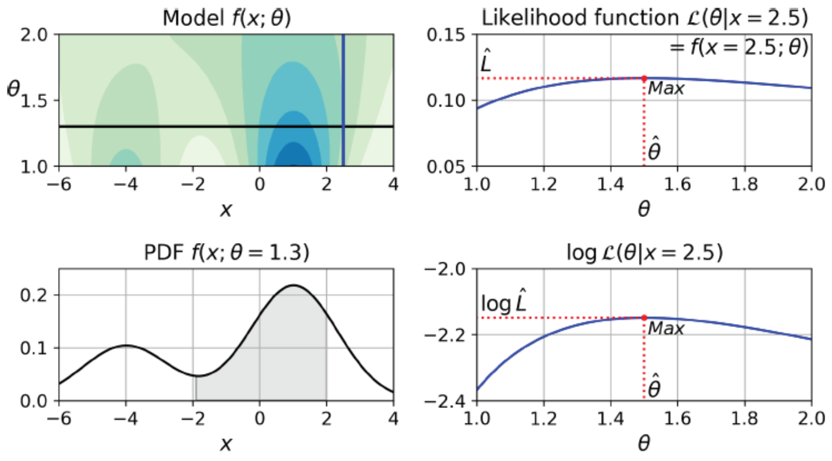
$$D[f(g(x))] = f\prime[g(x)] \cdot g\prime(x)$$

$$D[f(x) \cdot g(x)] = f\prime(x) \cdot g(x) + f(x) \cdot g\prime(x)$$

$$D\left[\frac{f(x)}{g(x)}\right] = \frac{f\prime(x) \cdot g(x) - f(x) \cdot g\prime(x)}{[g(x)]^2} \text{ , con } g(x) \neq 0$$

7.Statistics

Probability: Given a statistical model with some parameters $\boldsymbol{\theta}$**,** describe how plausible a future outcome $\boldsymbol{x}$ is.
Likelihood: describe how plausible a particular set of parameter values $\boldsymbol{\theta}$ are, after the outcome $\boldsymbol{x}$ is known. Is not a probability distribution.
PDF is a function of x (with θ fixed) while the likelihood function is a function of θ (with x fixed).



Maximum likelihood estimate (MLE): given a dataset $\boldsymbol{X}$ try to estimate the most likely values for the model parameters. To do this, you must find the values that maximize the likelihood function, given $\boldsymbol{X}$.

$I(\boldsymbol{x})$ is a **Confidence interval (set)** (for a parameter θ) of level $1 - \alpha$ if $\mathbb{P}[\theta \in I(\boldsymbol{x})] > 1 - \alpha$
To locate θ with high precision and high confidence, we would like that the set $I(\boldsymbol{x})$ is small and contains θ with high probability (α small)

8. Model Selection and Validation

Model selection: the task of choosing the best algorithm and its parameters for a particular problem.
Two approaches:

- **Validation**: partition the training set into two sets. One is used for training each of the candidate models, and the second is used for deciding which of them yields the best results.
- **Structural Risk Minimization (SRM)** paradigm: useful when a learning algorithm depends on a parameter that controls the bias-complexity tradeoff.

Validation: Once you pick a hypothesis, use new data to estimate its true error.
Let h be some predictor and assume that the loss function is in $[0,1]$. Then, for every $\delta \in (0,1)$ with probability of at least $1-\delta$ over the choice of a validation set V of size m_v:

$$|L_V(h) - L_D(h)| \le \sqrt{\frac{\log(2/\delta)}{2m_v}}$$

Compared to VCdim, if m_v is in the order of m, validation is more accurate.

Validation for Model Selection:

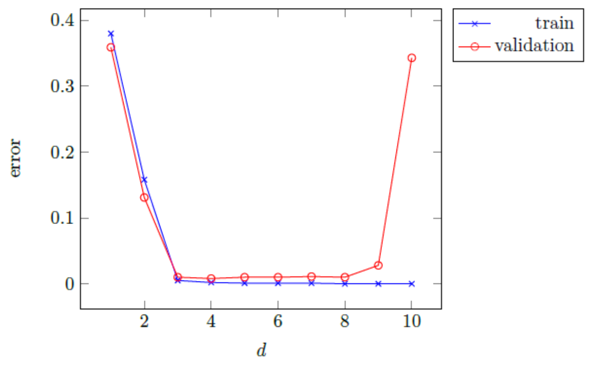
- Train different algorithms, returning to us a set of predictors $H = \{h_1, \dots, h_r\}$
- Choose the predictor that minimizes the error over the validation set. In other words, we apply ERM_H over the validation set.

Assume that the loss function is in $[0,1]$. Then, for every $\delta \in (0,1)$ with probability $\ge 1-\delta$ over the choice of a validation set V of size m_v:

$$\forall h \in H, \qquad |L_V(h) - L_D(h)| \le \sqrt{\frac{\log(2|H|/\delta)}{2m_v}}$$

The error on the validation set approximates the true error as long as H is not too large, because if we have too many hypotheses we're in danger of overfitting.

The Model-Selection Curve: shows the training error and validation error as a function of the complexity of the model considered. If Training error decreases but validation error increases it means overfitting.



If we have one or more parameters

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

Train-Validation-Test Split: (estimate the true risk after model selection)

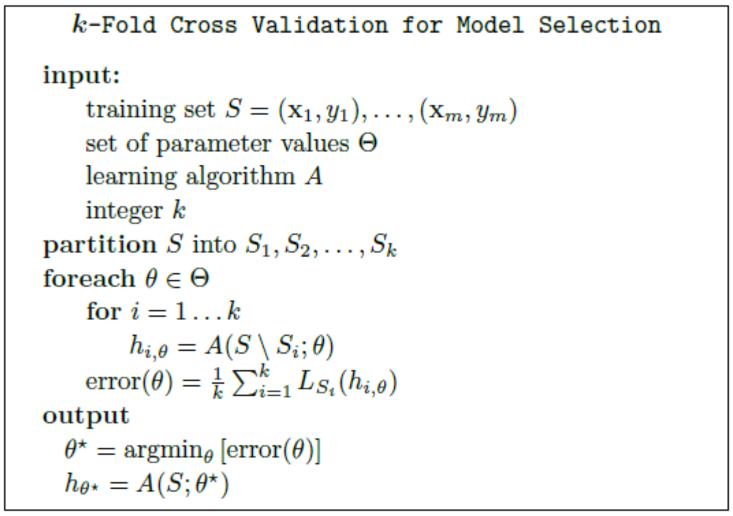
- Training set: used to learn the best model h_i from each H_i
- Validation set: used to pick one hypothesis h from $\{h_1, \dots, h_r\}$
- Test set: used to estimate the true risk $L_D(h)$. Not involved in the choice of h.

k-Fold Cross Validation: used when we have not much data

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

Lease-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 $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
    set of parameter values $\Theta$
    learning algorithm $A$
    integer $k$
partition $S$ into $S_1, S_2, \ldots, S_k$
foreach $\theta \in \Theta$
    for $i = 1 \ldots k$
        $h_{i,\theta} = A(S \setminus S_i; \theta)$
    $\text{error}(\theta) = \frac{1}{k} \sum_{i=1}^{k} L_{S_i}(h_{i,\theta})$
output
  $\theta^\star = \text{argmin}_\theta \, [\text{error}(\theta)]$
  $h_{\theta^\star} = A(S; \theta^\star)$
```

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 H
    - change 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
    - educe complexity of H
- if approximation error seems large:
    - change H
    - change feature representation of the data

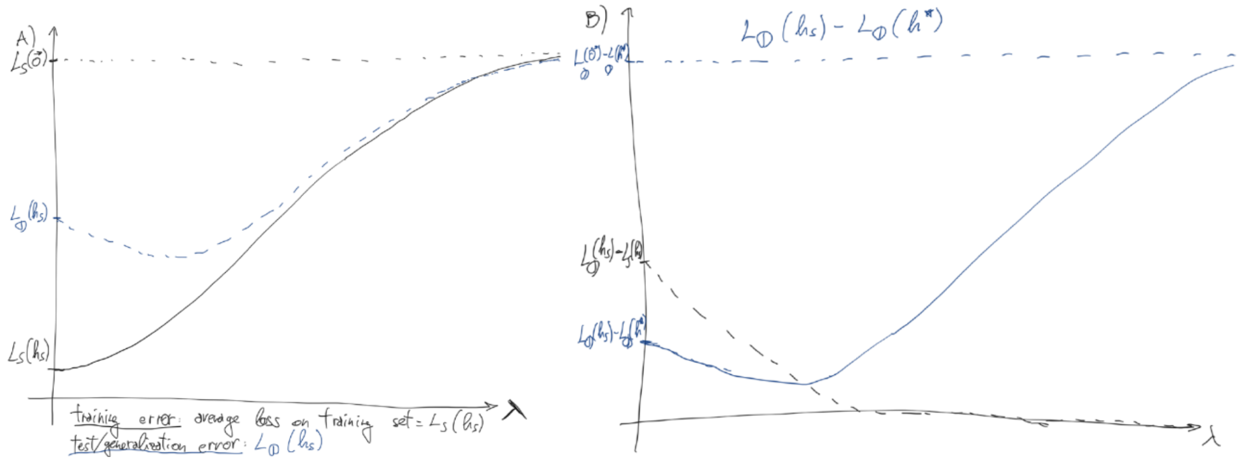
9.Regularization and Feature Selection

9.1. Regularization

Stable algorithm: if a slight change of its input does not change its output much.
Regularization: is as a stabilizer of the learning algorithm.
Regularization function: $R: \mathbb{R}^d \rightarrow \mathbb{R}$, measures the complexity of hypotheses.
Regularized Loss Minimization (RLM) is a learning paradigm that minimize the sum of the empirical risk and a regularization function.

$$A(S) = \underset{\boldsymbol{w}}{\operatorname{argmin}}\big(L_S(\boldsymbol{w}) + R(\boldsymbol{w})\big)$$

The algorithm balances between low empirical risk and "less complex" hypotheses.



For low values of λ we are ignoring the $R(\boldsymbol{w})$ term, so we have L_S low and L_D high, that corresponds to overfitting. Increasing λ we are taking the complexity more into account, and for some value of λ we can have a good balance between training error and complexity of the model.

L1 Regularization: $R(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|_1$

- $\lambda \in \mathbb{R}, \lambda > 0$
- $\|\boldsymbol{w}\|_1 = \sum_{i=1}^{d} |w_i|$
- $A(S) = \underset{\boldsymbol{w}}{\operatorname{argmin}}(L_S(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1)$

LASSO: Linear regression with squared loss + L1 regularization

$$\boldsymbol{w} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\left(\lambda \|\boldsymbol{w}\|_1 + \frac{1}{m}\sum_{i=1}^{m}(\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle - y_i)^2\right)$$

L2 (Tikhonov) regularization function: $R(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|^2$

- $\lambda \in \mathbb{R}, \lambda > 0$. Regulates the tradeoff between the empirical risk $L_S(\boldsymbol{w})$ and the complexity $\|\boldsymbol{w}\|^2$ of the model
- $\|\boldsymbol{w}\|^2 = \sum_{i=1}^{d} w_i^2$ L2 norm: measures the complexity of hypothesis defined by $\boldsymbol{w}$

Ridge Regression: linear regression with the squared loss + Tikhonov.

$$\boldsymbol{w} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\left(\lambda \|\boldsymbol{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m}(\langle \boldsymbol{w}, x_i\rangle - y_i)^2\right) = (\lambda \boldsymbol{I} + \boldsymbol{X^T X})^{-1}\boldsymbol{X^T y}$$

Derivation proof:

- $\boldsymbol{w} = \underset{\boldsymbol{w}}{\operatorname{argmin}}(\lambda \|\boldsymbol{w}\|^2 + RSS) = \underset{\boldsymbol{w}}{\operatorname{argmin}}\big(\lambda \|\boldsymbol{w}\|^2 + (\boldsymbol{y} - \boldsymbol{Xw})^T(\boldsymbol{y} - \boldsymbol{Xw})\big)$

- We want to minimize $f(\boldsymbol{w}) = \lambda\|\boldsymbol{w}\|^2 + (\boldsymbol{y} - \boldsymbol{Xw})^T(\boldsymbol{y} - \boldsymbol{Xw})$
- So, we compute gradient $\frac{\partial f(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0$ ---> $2\lambda\boldsymbol{w} - 2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{Xw}) = 0$
- Math passages to find w:

$$\lambda\vec{w} - X^T(\vec{y} - X\vec{w}) = 0$$
$$\lambda\vec{w} + X^T X\vec{w} = X^T\vec{y}$$
$$(\lambda I + X^T X)\vec{w} = X^T\vec{y}$$
$$\vec{w} = (\lambda I + X^T X)^{-1} X^T \vec{y}$$

Fitting-Stability Tradeoff

Expected risk of a learning algorithm: $\mathbb{E}_S[L_D(A(S))] = \mathbb{E}_S[L_S(A(S))] + \mathbb{E}_S[L_D(A(S)) - L_S(A(S))]$

- $\mathbb{E}_S[L_S(A(S))]$ reflects how well A(S) fits the training set
- $\mathbb{E}_S[L_D(A(S)) - L_S(A(S))]$ reflects the difference between the true and empirical risks of A(S). = Overfitting, bounded by stability of A(S)

We need that the sum of both terms will be small. The tradeoff between the two terms is controlled by λ

λ is chosen with Validation.

9.2. Feature selection

Feature function: any measurement of the real-world object can be regarded as a feature.

Feature vector: $\boldsymbol{x} \in X^d$

Feature selection: task of selecting a small number of features from a large pool, that will be used by the predictor. This prevent overfitting and prediction can be done faster.

Feature manipulations and normalization: simple transformations that we apply on our original features. These transformations may decrease the sample complexity of our learning algorithm, its bias, or its computational complexity.

Feature learning: methods that automate the process of feature construction.

Problem to solve: Select k features that minimize the empirical risk

$$\min_{\boldsymbol{w}} L_S(\boldsymbol{w}) \text{ subject to } \|\boldsymbol{w}\|_0 \le k \text{ where } \|\boldsymbol{w}\|_0 = |\{i: w_i \neq 0\}|$$

Subset selection: $O(d^k)$, NP-Hard

Let:

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

```
$P^{(k)} \leftarrow \{J \subseteq \mathcal{I} : |J| = k\}$;
foreach $p \in P^{(k)}$ do
    $h_p \leftarrow \arg\min_{h \in \mathcal{H}_p} L_S(h)$;
return $h^{(k)} \leftarrow \arg\min_{p \in P^{(k)}} L_S(h_p)$;
```

Greedy algorithms: Forward Selection and Backward Selection

Forward Selection: $O(kd)$
Start from the empty solution, add one feature at the time, until solution has cardinality k

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

Backward Selection: $O\big((d-k)d\big)$
start from the solution which includes all features, remove one features at the time, until solution has cardinality k

These 3 algorithms are trained using just the training set, this can lead to overfit. So, we can use a validation set in each iteration. Or also cross-validation.

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 $\arg\min_{h \in \{h^{(0)}, h^{(1)}, \ldots, h^{(k)}\}} L_V(h)$
```

Using training and validation:

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

10. Support Vector Machines

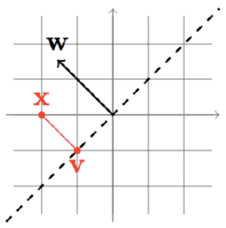
Learn linear predictors in high dimensional feature spaces.

10.1. Margin and Hard-SVM

Let:

- Instance set: $X = \mathbb{R}^d$
- Label set $Y = \{-1, +1\}$
- Training set: $S = \big((\boldsymbol{x_1}, y_1), \dots, (\boldsymbol{x_m}, y_m)\big)$
- Hypothesis set H: halfspaces $H_{\boldsymbol{w},b} = \{\boldsymbol{x} \to \operatorname{sign}(\langle \boldsymbol{w}, \boldsymbol{x}\rangle + b) : \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$

Training set linearly separable: if exists a halfspace $(\boldsymbol{w}, b)$ $s.t.$ $y_i = \operatorname{sign}(\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b)\ \forall i$
Equivalent to $\forall i = 1, \dots, m$: $y_i(\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b) > 0$

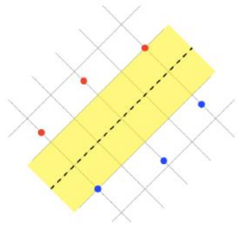


Distance between point $\boldsymbol{x}$ and hyperplane $L = \{\boldsymbol{v} : \langle \boldsymbol{w}, \boldsymbol{v}\rangle + b = 0\}$ is

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

If $\|\boldsymbol{w}\| = 1$ then $d(\boldsymbol{x}, L) = |\langle \boldsymbol{w}, \boldsymbol{x}\rangle + b|$

Margin of a separating hyperplane: is its minimum distance to an example in the training set S
If $\|\boldsymbol{w}\| = 1$ then margin $= \min\limits_{i \in \{1,\dots,m\}} |\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b|$
The closest examples are called **support vectors**



Hard-SVM: learning rule in which we return an ERM hyperplane that separates the training set with the largest possible margin. (only for linearly separable data)

$$\underset{(\boldsymbol{w},b):\|\boldsymbol{w}\|=1}{\operatorname{argmax}} \left(\min_{i \in \{1,\dots,m\}} |\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b| \right) \qquad \text{(maximum margin)}$$

$$\text{subject to } \forall i = 1, \dots, m: y_i(\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b) > 0 \qquad \text{(Training set linearly separable)}$$

Equivalent formulation (due to separability assumption): $\underset{(\boldsymbol{w},b):\|\boldsymbol{w}\|=1}{\operatorname{argmax}} \left(\min\limits_{i \in \{1,\dots,m\}} y_i(\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b) \right)$

Equivalent formulation as quadratic optimization problem (easily solvable by solvers):

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

solve:

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

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

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

(Equivalent formulation) **Homogenous halfspaces:** pass through the origin and are defined by $\operatorname{sign}(\langle \boldsymbol{w}, \boldsymbol{x}\rangle)$, where the bias term b is set to be zero.
Hard-SVM for homogenous halfspaces: $\boldsymbol{w_0} = \min\limits_{\boldsymbol{w}} \|\boldsymbol{w}\|^2$ subject to $\forall i = 1, \dots, m$: $y_i(\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle) \geq 1$

Equivalent Dual formulation: $\max\limits_{\boldsymbol{\alpha} \in \mathbb{R}^m : \boldsymbol{\alpha} \geq \boldsymbol{0}} \left(\sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x_j}, \boldsymbol{x_i}\rangle \right)$
Solution is the vector $\boldsymbol{\alpha}$ which defines the support vectors $= \{\boldsymbol{x_i} : \alpha_i \neq 0\}$
Dual problem only involves inner products between instances and does not require direct access to $\boldsymbol{x_i}$. This property is important when implementing SVM with kernels.

10.2. Soft-SVM

Used if the training set is not linearly separable. It allows to have some points inside the margin or wrongly classified.

Hard-SVM constraints: $y_i(\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b) \geq 1$

Soft-SVM constraints:

- Slack variables $\xi_1, \ldots, \xi_m \geq 0$
- $\forall i \in \{1, \ldots, m\}$: $y_i(\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b) \geq 1 - \xi_i$
- each ξ_i measure how much the constraint $y_i(\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b) \geq 1$ is violated.
    - $\xi_i = 0$ means $\boldsymbol{x_i}$ correctly classified
    - $0 < \xi_i \leq 1$: x_i correctly classified but is inside the margin
    - $\xi_i > 1$ means $\boldsymbol{x_i}$ is wrongly classified.

Soft-SVM jointly minimizes the norm of $\boldsymbol{w}$ (corresponding to the margin) and the average of ξ_i (corresponding to the violations of the constraints).

Parameter $\lambda > 0$: tradeoff between norm of $\boldsymbol{w}$ and average of ξ_i. Bigger λ means bigger margin.

Soft-SVM Optimization Problem:

input: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, parameter $\lambda > 0$

solve:

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

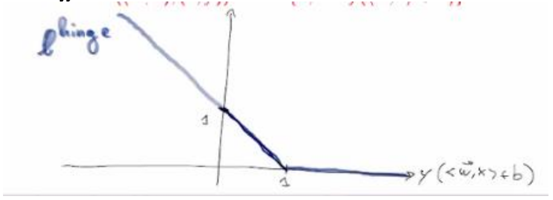
subject to $\forall i : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$

output: $\mathbf{w}, b$

Hinge Loss: $L_S^{hinge}\big((\boldsymbol{w}, b)\big) = \frac{1}{m}\sum_{i=1}^{m} l^{hinge}\big((\boldsymbol{w}, b), (\boldsymbol{x}_i, y_i)\big)$

where $l^{hinge}\big((\boldsymbol{w}, b), (\boldsymbol{x}_i, y_i)\big) = \max\{0, 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b)\}$

Equivalent Hinge formulation: $\min\limits_{\boldsymbol{w}} \big(\lambda \|\boldsymbol{w}\|^2 + L_S^{hinge}(\boldsymbol{w}, b)\big)$



Hinge formulation can be solved with optimization solvers OR with **Stochastic Gradient Descent**.

Gradient Descent: General approach for minimizing a differentiable convex function $f(\boldsymbol{w})$. Measures the local gradient of the error function with regards to the parameter vector $\boldsymbol{\theta}$, and it goes in the direction of descending gradient.

Gradient: vector of partial derivatives $\nabla f(\mathbf{w}) = \left(\frac{\partial f(\boldsymbol{w})}{\partial w_1}, \ldots, \frac{\partial f(\boldsymbol{w})}{\partial w_d}\right)$ (where $f = l\big(\boldsymbol{w}, (\boldsymbol{x_i}, y_i)\big)$)

```
$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$;
for $t \leftarrow 0$ to $T - 1$ do
    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)})$;
return $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$;
```

Notes:

- output vector could also be $\mathbf{w}^{(T)}$ or $\arg\min_{\mathbf{w}^{(t)} \in \{1,\ldots,T\}} f(\mathbf{w}^{(t)})$
- returning $\bar{\mathbf{w}}$ 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)

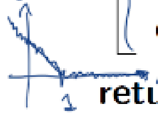
Stochastic Gradient Descent (SGD): at every step instead of using exactly the gradient, we take a (random) vector (random from the training set) with expected value equal to the gradient direction.

```
SGD algorithm:
$\mathbf{w}^{(0)} \leftarrow \mathbf{0}$; // or $\vec{w}^{(0)} \leftarrow$ random vector in $\mathbb{R}^d$
for $t \leftarrow 0$ to $T-1$ do
    choose $\mathbf{v}_t$ at random from distribution such that $\mathbf{E}[\mathbf{v}_t|\mathbf{w}^{(t)}] \in \nabla f(\mathbf{w}^{(t)})$;
    /* $\mathbf{v}_t$ has expected value equal to the gradient of $f(\mathbf{w}^{(t)})$ */
    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \mathbf{v}_t$;
return $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{w}^{(t)}$; $\rightarrow$ see notes from GD for $\vec{w}$
```

SGD for Solving Soft-SVM:

```
SGD algorithm:
$\boldsymbol{\theta}^{(1)} \leftarrow \mathbf{0}$; // or $\vec{\theta}^{(1)} \leftarrow$ random ~~pector~~ vector in $\mathbb{R}^d$
for $t \leftarrow 1$ to $T$ do                  $\rightarrow$ time-dependent learning rate
    $\eta^{(t)} \leftarrow \frac{1}{\lambda t}$; $\mathbf{w}^{(t)} \leftarrow \eta^{(t)}\boldsymbol{\theta}^{(t)}$;
    choose $i$ uniformly at random from $\{1,\ldots,m\}$; $\rightarrow$ (take a point from $S=((\vec{x}_1,y_1),\ldots,(\vec{x}_m,y_m))$ uniformly at random)
    if $y_i\langle \mathbf{w}^{(t)}, \mathbf{x}_i\rangle < 1$ then $\boxed{\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \underbrace{y_i\mathbf{x}_i}_{\text{contribution of the gradient}};}$
    else $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)}$;
return $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{w}^{(t)}$;
```

taking into account the fact that ℓ^{hinge}:



10.3. Kernels

The expressive power of halfspaces is rather restricted.
To make the class of halfspaces more expressive we can first map the original instance space into another space (possibly of a higher dimension) and then learn a halfspace in that space.
Basic paradigm:

- Given some domain set X choose a non-linear mapping $\psi: X \to F$, usually $F = \mathbb{R}^n$
- Given a sequence of labeled examples $S = (\boldsymbol{x_1}, y_1), \dots, (\boldsymbol{x_m}, y_m)$ create the image sequence $\hat{S} = \big((\psi(\boldsymbol{x_1}), y_1), \dots, (\psi(\boldsymbol{x_m}), y_m)\big)$
- Train a linear predictor h over $\hat{S}$
- Predict the label of a test point, $\boldsymbol{x}$, to be $h\big(\psi(\boldsymbol{x})\big)$

The dual formulation requires to compute $\langle \psi(\boldsymbol{x}), \psi(\boldsymbol{x}') \rangle$
Kernel function: $K_\psi(\boldsymbol{x}, \boldsymbol{x}') = \langle \psi(\boldsymbol{x}), \psi(\boldsymbol{x}') \rangle$

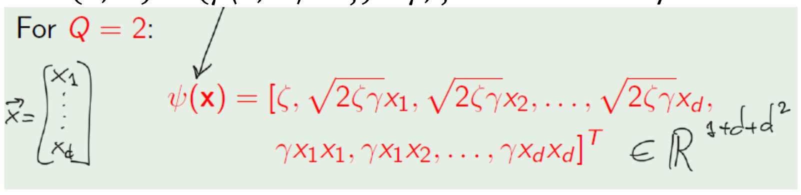
- K specify similarity between instances.
- ψ maps the domain set X into a space where these similarities are realized as inner products.

To compute K_ψ we must be able to compute $\psi(\boldsymbol{x})$ that is very expensive.

Kernel trick: in some cases we can compute directly K_ψ without computing $\psi(\boldsymbol{x})$

Common kernels:

- Linear: $\psi(\boldsymbol{x}) = \boldsymbol{x}$
- Q-Polynomial: $K(\boldsymbol{x}, \boldsymbol{x}') = (\gamma\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + \zeta)^Q \;\; \gamma, \zeta > 0$ usually used with $Q \leq 10 \in \mathbb{N}$

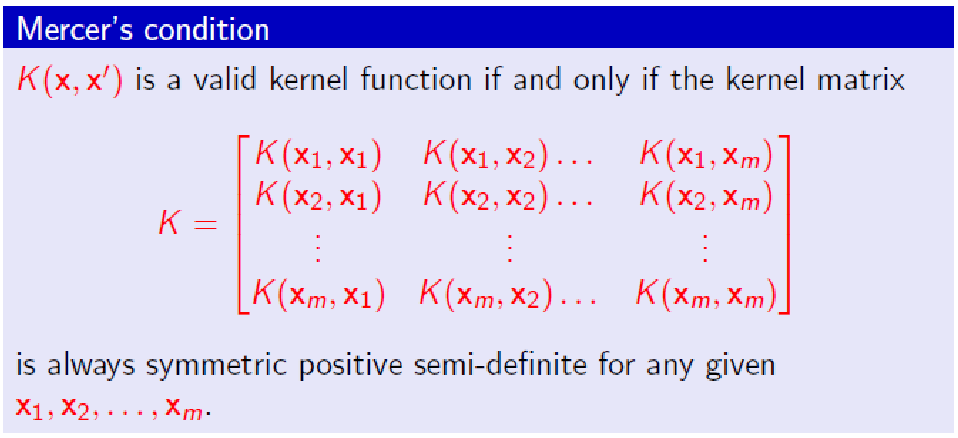


For $Q = 2$:

$$\vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} \qquad \psi(\mathbf{x}) = [\zeta, \sqrt{2\zeta\gamma}x_1, \sqrt{2\zeta\gamma}x_2, \dots, \sqrt{2\zeta\gamma}x_d, \gamma x_1 x_1, \gamma x_1 x_2, \dots, \gamma x_d x_d]^T \in \mathbb{R}^{1+d+d^2}$$

- Gaussian RBF: $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{x}'\|^2) \;\; \gamma > 0$ usuallly used with $\gamma \in [0,1]$
- Sigmoid: $K(\boldsymbol{x}, \boldsymbol{x}') = \tanh(\gamma\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + \zeta) \;\; \gamma, \zeta > 0$

Choice of kernel:



Mercer's condition

$K(\mathbf{x}, \mathbf{x}')$ is a valid kernel function if and only if the kernel matrix

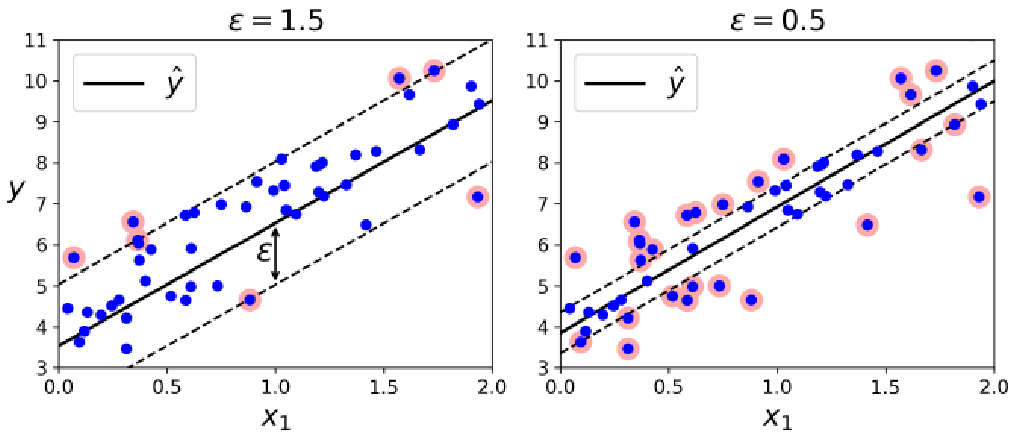
$$K = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) \dots & K(\mathbf{x}_1, \mathbf{x}_m) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) \dots & K(\mathbf{x}_2, \mathbf{x}_m) \\ \vdots & \vdots & \vdots \\ K(\mathbf{x}_m, \mathbf{x}_1) & K(\mathbf{x}_m, \mathbf{x}_2) \dots & K(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

is always symmetric positive semi-definite for any given $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$.

10.4. SVM for regression

Instead of trying to fit the largest possible street between two classes while limiting margin violations, SVM Regression tries to fit as many instances as possible on the street while limiting margin violations.

The width of the street is controlled by a **hyperparameter $\boldsymbol{\varepsilon}$**.



Function to minimize:

$$\frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{m} V_\varepsilon(y_i - \langle \boldsymbol{x_i}, \boldsymbol{w}\rangle - b)$$

$$\text{where loss function } V_\varepsilon(r) = \begin{cases} 0, & |r| < \varepsilon \\ |r| - \varepsilon, & \text{otherwise} \end{cases}$$

Solution has the form: $\boldsymbol{w} = \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)\boldsymbol{x_i}$

Final model produced: $h(\boldsymbol{x}) = \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)\langle \boldsymbol{x_i}, \boldsymbol{x}\rangle + b$ where $\alpha_i^*, \alpha_i \geq 0$

Support vector: $\boldsymbol{x_i}$ such that $\alpha_i^* - \alpha_i \neq 0$

11. Neural Networks

11.1. Basics

Neuron: function $\boldsymbol{x} \to \sigma(\langle \boldsymbol{v}, \boldsymbol{x} \rangle)$ with $\boldsymbol{x} \in \mathbb{R}^d$
Activation function: $\sigma: \mathbb{R} \to \mathbb{R}$

- sign: $\sigma(a) = \text{sign}(a)$
- threshold: $\sigma(a) = \mathbf{1}[a > 0]$
- sigmoid: $\sigma(a) = \frac{1}{1+e^{-a}}$

Neural network formalism:

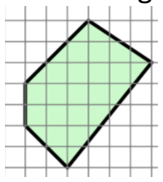
- Acyclic graph $G = (V, E)$
- Weight function $w: E \to \mathbb{R}$
- $V = \bigcup_{t=0}^{T} V_t, V_i \cap V_j = \emptyset \; \forall i \neq j$
- $e \in E$ can only go from V_t to V_{t+1}
- Input layer: V_0
- Output layer: V_T
- Hidden layers: $V_t, 0 < t < T$
- Depth: T
- Size: $|V|$
- Width: $\max_t |V_t|$

for binary classification and regression (1 variable): output layer has 1 node

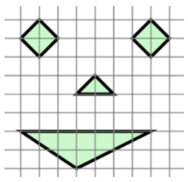
Architecture: $(\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{\sigma})$
NN represent a **function**: $h_{V,E,\sigma,w}: \mathbb{R}^{|V_0|-1} \to \mathbb{R}^{|V_T|}$
Hypothesis Set: $H_{V,E,\sigma} = \{h_{V,E,\sigma,w}: w \text{ is a mapping from } E \text{ to } \mathbb{R}\}$

Expressiveness of NN: every Boolean function can be implemented using a neural network of depth 2. NNs are universal approximators. But it's very big.
With $\sigma = sign$:

- Depth 2 (1 hidden layer): Intersections of halfspaces



- Depth 3 (2 hidden layers): Union of intersections of halfspaces



Sample Complexity: quantity of data needed to learn with NN.

- VC-dim of $H_{V,E,sign} = O(|E| \log|E|)$
- VC-dim of $H_{V,E,sigmoid} = O(|V|^2 |E|^2)$

Large NNs require a lot of data.

Runtime of Learning: applying the ERM rule with respect to $H_{V,E,sign}$ is NP hard.
So, we train NN using Stochastic Gradient Descent.

11.2. Learning NN

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:

$$\mathbf{v}^{(t)} = \begin{bmatrix} 1 \\ v_{t,1} \\ \vdots \\ v_{t,d^{(t)}} \end{bmatrix} = \begin{bmatrix} 1 \\ \sigma\left(\langle \mathbf{w}_1^{(t)}, \mathbf{v}^{(t-1)} \rangle\right) \\ \vdots \\ \sigma\left(\langle \mathbf{w}_{d^{(t)}}^{(t)}, \mathbf{v}^{(t-1)} \rangle\right) \end{bmatrix}$$

Let

$$a_{t,j} := \langle \mathbf{w}_j^{(t)}, \mathbf{v}^{(t-1)} \rangle$$

and

$$\mathbf{a}^{(t)} = \begin{bmatrix} a_{t,1} \\ \vdots \\ a_{t,d^{(t)}} \end{bmatrix} \qquad \sigma\left(\mathbf{a}^{(t)}\right) = \begin{bmatrix} \sigma\left(a_{t,1}\right) \\ \vdots \\ \sigma\left(a_{t,d^{(t)}}\right) \end{bmatrix}$$

Then

$$\mathbf{v}^{(t)} = \begin{bmatrix} 1 \\ \sigma\left(\mathbf{a}^{(t)}\right) \end{bmatrix}$$

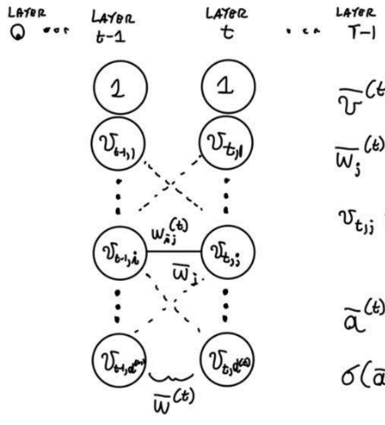
Let

$$\mathbf{w}^{(t)} = \begin{bmatrix} w_{01}^{(t)} & w_{02}^{(t)} & \dots & w_{0d^{(t)}}^{(t)} \\ w_{11}^{(t)} & w_{12}^{(t)} & \dots & w_{1d^{(t)}}^{(t)} \\ \vdots & \vdots & \dots & \vdots \\ w_{d^{(t-1)}1}^{(t)} & w_{d^{(t-1)}2}^{(t)} & \dots & w_{d^{(t-1)}d^{(t)}}^{(t)} \end{bmatrix}$$

($\mathbf{w}^{(t)}$ describes the weights of edges from layer $t-1$ to layer t)

Then

$$\mathbf{a}^{(t)} = \left(\mathbf{w}^{(t)}\right)^T \mathbf{v}^{(t-1)}$$



$$\bar{v}^{(t)} = \left(1, v_{t,1}, \dots, v_{t,d^{(t)}}\right)^T = \left(1, \sigma\left(\langle w_1^{(t)}, \bar{v}^{(t-1)} \rangle\right), \dots\right)$$

$$= \left(1, \sigma\left(\bar{a}^{(t)}\right)\right)^T$$

$$\bar{w}_j^{(t)} = \left(w_{0j}^{(t)}, \dots, w_{d^{(t-1)}j}^{(t)}\right)^T$$

$$v_{t,j} = \sigma\left(\underbrace{\langle \bar{w}_j^{(t)}, \bar{v}^{(t-1)} \rangle}_{a_{t,j}}\right)$$

$$\bar{a}^{(t)} = \left(a_{t,1}, \dots, a_{t,d^{(t)}}\right)^T = \left(\bar{w}^{(t)}\right)^T \bar{v}^{(t-1)}$$

$$\sigma\left(\bar{a}^{(t)}\right) = \left(\sigma(a_{t,1}), \dots, \sigma(a_{t,d^{(t)}})\right)^T$$

$$\bar{w}^{(t)} = \begin{bmatrix} w_{01}^{(t)} & \dots & w_{0d^{(t)}}^{(t)} \\ \vdots & & \vdots \\ w_{d^{(t-1)}1}^{(t)} & \dots & w_{d^{(t-1)}d^{(t)}}^{(t)} \end{bmatrix}$$

DESCRIBES THE WEIGHTS FROM LAYER $t-1$ TO t

Forward propagation:

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
    $\mathbf{a}^{(t)} \leftarrow \left(\mathbf{w}^{(t)}\right)^T \mathbf{v}^{(t-1)}$;
    $\mathbf{v}^{(t)} \leftarrow \left(1, \sigma\left(\mathbf{a}^{(t)}\right)^T\right)^T$;
$y \leftarrow \sigma\left(\mathbf{a}^{(T)}\right)$;
return $y$;
```

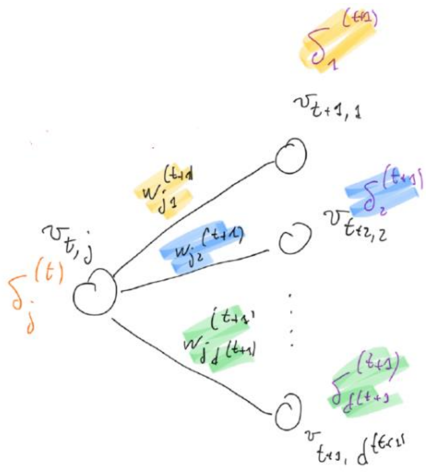
Learning the weights $w_{ij}^{(t)}$

By minimizing the training error $L_S(h) = \frac{1}{m}\sum_{i=1}^{m} l\big(h, (\boldsymbol{x}_i, y_i)\big)$ with SGD

Update rule: $\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla L_S\big(\boldsymbol{w}^{(t)}\big)$

where $\nabla L_S\big(\boldsymbol{w}^{(t)}\big)$ is the gradient

$$\forall t\colon \frac{\partial L_S}{\partial \boldsymbol{w}^{(t)}} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial l\big(h, (\boldsymbol{x}_i, y_i)\big)}{\partial \boldsymbol{w}^{(t)}}$$



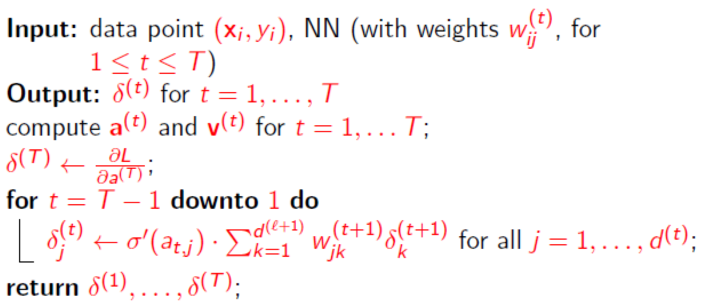
Sensitivity vector for layer t: $\delta^{(t)} = \frac{\partial l}{\partial \boldsymbol{a}^{(t)}} = \begin{bmatrix} \frac{\partial l}{\partial a_{t,1}} \\ \vdots \\ \frac{\partial l}{\partial a_{t,d^{(t)}}} \end{bmatrix} = \begin{bmatrix} \delta_1^{(t)} \\ \vdots \\ \delta_{d^{(t)}}^{(t)} \end{bmatrix}$

quantify how the training error changes with $\boldsymbol{a}^{(t)}$ (the inputs to the t layer - before the nonlinear transformation)

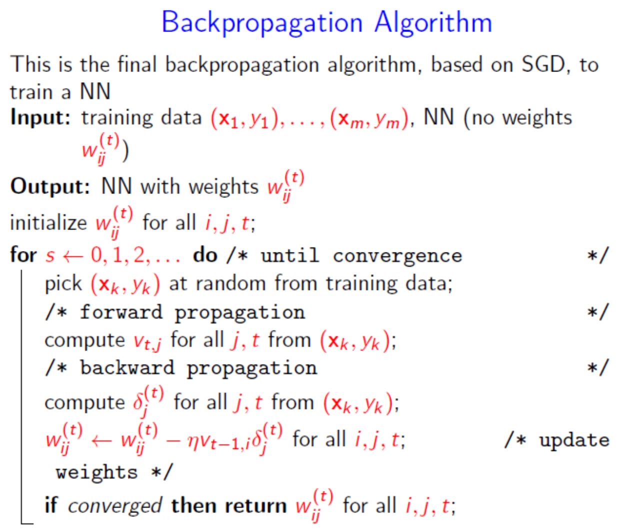
$$\delta_j^{(t)} = \sigma'(a_{t,j}) \sum_{k=1}^{d^{(t+1)}} w_{jk}^{(t+1)} \delta_k^{(t+1)}$$

where $\delta^{(T)} = \frac{\partial l}{\partial \boldsymbol{a}^{(t)}}$ (sensitivity of final layer depends on the loss L used) and σ' is the derivative of activation function.

Compute sensitivities $\delta^{(t)}$ $\forall t$ given a datapoint $(\boldsymbol{x}_i, y_i)$:



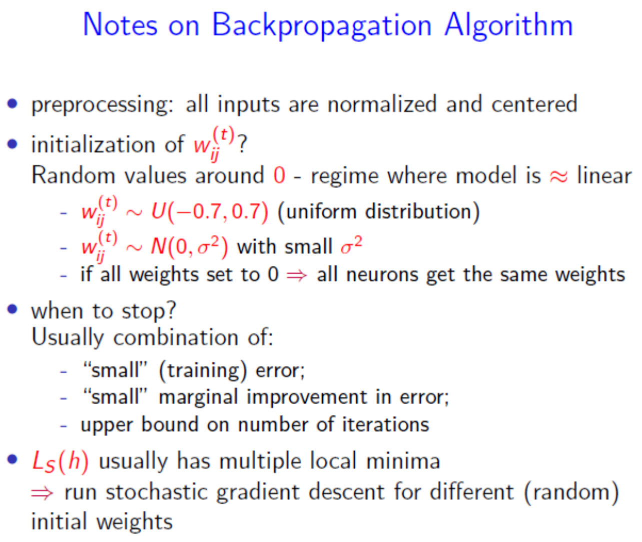
```
Input: data point $(\mathbf{x}_i, y_i)$, NN (with weights $w_{ij}^{(t)}$, for
       $1 \le t \le T$)
Output: $\delta^{(t)}$ for $t = 1, \dots, T$
compute $\mathbf{a}^{(t)}$ and $\mathbf{v}^{(t)}$ for $t = 1, \dots T$;
$\delta^{(T)} \leftarrow \frac{\partial L}{\partial a^{(T)}}$;
for $t = T - 1$ downto $1$ do
    $\delta_j^{(t)} \leftarrow \sigma'(a_{t,j}) \cdot \sum_{k=1}^{d^{(\ell+1)}} w_{jk}^{(t+1)} \delta_k^{(t+1)}$ 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: To prevent overfitting. Introduces the **hyperparameter λ**

Minimize $L_S(h) + \frac{\lambda}{2}\sum_{i,j,t}\big(w_{ij}^{(t)}\big)^2$ (squared weight decay regularizer)

General construction of NN for a given Boolean formula:

General construction: let's take an arbitrary function $f: \{-1, 1\}^d \to \{-1, 1\}$.

Build a NN that corresponds to f (if the input of the NN is $\vec{x} \in \{-1, 1\}^d$, then the output of the NN is $f(\vec{x})$):

- consider $\vec{x}$ such that $f(\vec{x}) = 1$: for each such $\vec{x}$, there is a neuron in the (only) hidden layer that "corresponds" to $\vec{x}$. The corresponding neuron "implements":

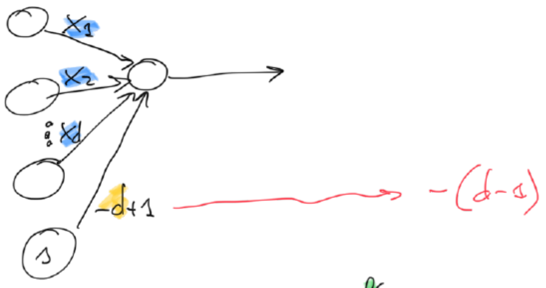
$$g_i(\vec{x}') = \text{sign}(\langle \vec{x}, \vec{x}' \rangle - d + 1)$$

weight on (incoming edges)

input to the neuron (and to the NN)

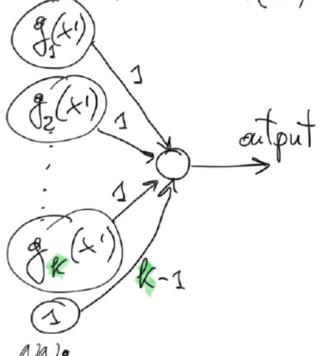
neuron $g_i(\vec{x}')$:

corresponds to $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$

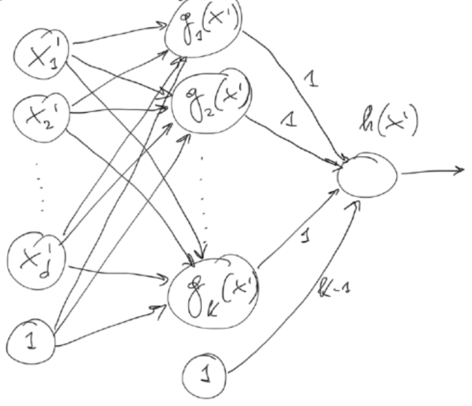


- output node: "implements" $h(\vec{x}') = \text{sign}\left(\sum_{i=1}^{k} g_i(\vec{x}') + k - 1\right)$

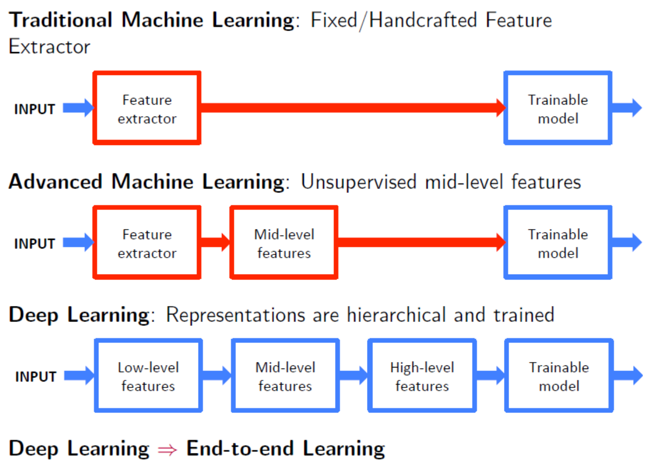
where k = # of vectors $\vec{x}$ s.t. $f(\vec{x}) = 1$



The overall NN:



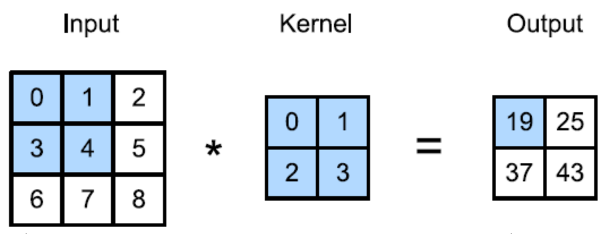
11.3. Convolutional Neural Networks


Traditional Machine Learning: Fixed/Handcrafted Feature Extractor
INPUT
Feature extractor
Trainable model
Advanced Machine Learning: Unsupervised mid-level features
INPUT
Feature extractor
Mid-level features
Trainable model
Deep Learning: Representations are hierarchical and trained
INPUT
Low-level features
Mid-level features
High-level features
Trainable model
Deep Learning ⇒ End-to-end Learning

Traditional NN requires a huge number of edges and the domain structure is not taken into account.

CNN: Are neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Prodotto di Convoluzione 2D: ogni cella della matrice risultante è data dalla somma dei prodotti elemento per elemento sovrapponendo la matrice filtro alla matrice di ingresso + bias (1 solo bias per filtro).



$$out_{1,1} = \big((0*0)+(1*1)+(3*2)+(4*3)\big)+b = 19+b$$

$\boldsymbol{S(i,j) = (I*K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)}$ where I is 2d input and K is 2D function (kernel)

Convolution Properties:

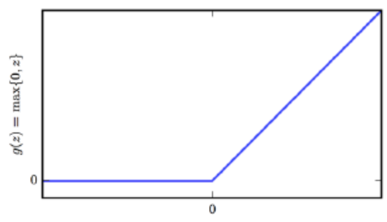
- sparse interactions: many edges do not exist in the network
- parameter sharing: using the same parameter for more than one function in a model.
- equivariant representations.

After a convolutional layer:

- nonlinear function: ReLU (Rectified Linear Unit)
- pooling layer, often combined with subsampling

ReLU (rectified linear unit) $\sigma(z) = \max\{0, z\}$

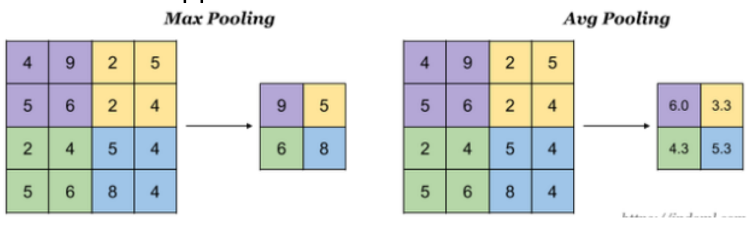
- Helps convergence (almost linear)
- Does not hurt expressiveness
- Prevent the vanishing gradient issue (for sigmoid, gradient can be 0 and so weights do not change)



Pooling function: replaces the output of the network at a certain location with a summary statistic of the nearby outputs.

- **max pooling:** pick the maximum of the region
- **average pooling:** pick the average of the region

Max pooling introduces **invariance to local translation:** many neurons have the same output value even if input values are "shifted" a bit. Useful if we care more about whether some feature is present than exactly where it is or how it appears.



Subsamplig (stride): Invece di fare 1 passo se ne fanno di più. Non viene fatto il prodotto se il filtro va fuori dal bordo.
Reduce the size of the representation.

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 => controls overfitting
- reduces computation

CNN Last Layer: fully connected NN, which learns from the features extracted at the previous hidden Layers.

ADAM (Adaptive Moment Estimation Iterative method)

$\mathbf{g}^{(t)} \in \nabla f(\mathbf{x})$. 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)

β_1, β_2, η: parameters

How to set them?

Validation, etc...

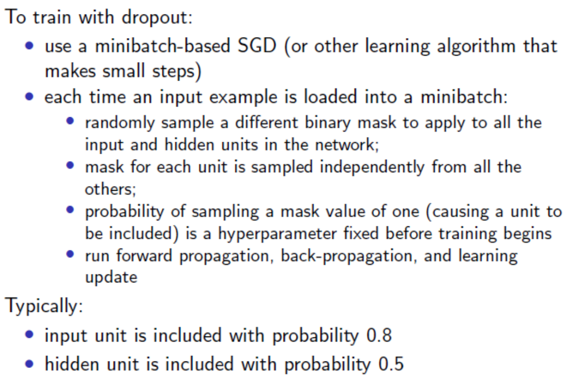
Good starting point:

- $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- $\eta = 0.001$ or $\eta = 5 \times 10^{-4}$

11.4. Techniques to Help Avoiding Overfitting

Regularization! (same as for other models!), dropout, early stopping, data augmentation

Dropout: method of regularizing a broad family of models.



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:

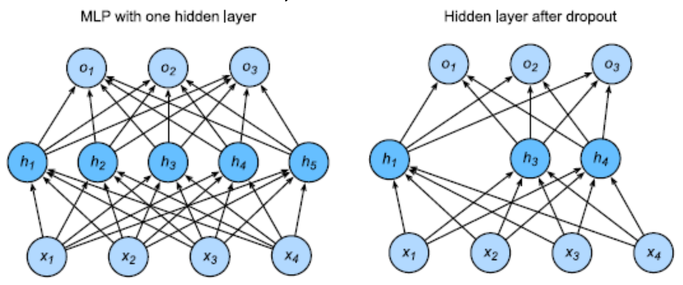
- input unit is included with probability 0.8
- hidden unit is included with probability 0.5

Dropout

Usato **quando** si ha un **training set piccolo**. Aspetto negativo è che la cost function non è più ben definita. Ogni layer ha associata una probabilità di eliminare i suoi nodi, se ne eliminano e si ripete per ogni esempio. Quindi ogni esempio ha associata una rete diversa, che viene allenata.

Non si usa drop out sul test set.

The idea behind drop-out is that at each iteration, you train a different model that uses only a subset of your neurons. With dropout, your neurons thus become less sensitive to the activation of one other specific neuron, because that other neuron might be shut down at any time.



Early Stopping: Use validation error to decide when to stop training. Stops when loss has not improved after n subsequent epochs. Parameter n is called patience

Data Augmentation:

Quando il dataset ha pochi dati, si può utilizzare qualche trucco per aumentarli.

Esempi:

- Mirroring
- Random cropping
- Less common (perhaps due to their complexity): Rotation, Shearing (3D rotation), Local warping
- Color shifting (R +20, G -20, B +20)

Loss Function:

Classification: cross entropy (instead of 0-1 loss) $l\big(h,(\boldsymbol{x},y)\big) = -y \log h(\boldsymbol{x}) - (1-y)\log\big(1 - h(\boldsymbol{x})\big)$

- is a convex function, SGD works better
- best hypothesis $h(\boldsymbol{x}) = \Pr[y = 1|\boldsymbol{x}]$
- Hypotheses set: functions with (prediction) value in 0 and 1. use sigmoid activation function for output.

Multiclass classification: k classes (0, 1, . . . , k)

- output is vector $\boldsymbol{y}$ with $y_i = 1$ if correct class is i, 0 otherwise
- $h(\boldsymbol{x}) \in (0,1)^d$ with $h_i(\boldsymbol{x})$ = probability label of $\boldsymbol{x}$ is i
- $l\big(h, (\boldsymbol{x}, \boldsymbol{y})\big) = -\sum_{i=k}^{k} y_i \log h_i(\boldsymbol{x})$

Regression: squared loss

12. Clustering

Unsupervised Learning: We have the input features X, but we do not have the labels y.
We are interested in finding some interesting structure in the data, or, equivalently, to organize it in some meaningful way.

Clustering: 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.
Problems:

- **Similarity (or proximity) is not a transitive relation**: Example, sequence of objects, $x_1, \dots, x_m$ such that each x_i is very similar to its two neighbors, x_{i-1} and x_{i+1}, but x_1 and x_m are very dissimilar.
- **Lack of "ground truth":** a given set of objects can be clustered in various different meaningful ways.

Model for Clustering:

- **Input**: X and d
  - set of elements X
  - distance function $d: X \times X \to \mathbb{R}_+$ symmetric
    - $d(x, x) = 0 \ \forall x$
    - often satisfy the triangle inequality: $d(x, x') \leq d(x, z) + d(z, x')$
  - OR similarity function $s: X \times X \to \mathbb{R}_+$ symmetric
    - $s(x, x) = 1 \ \forall x$
  - Additionally, k: number of clusters
- **Output**: partition of X $C = (C_1, \dots, C_k)$
  - $\bigcup_{i=1}^{k} C_i = X$
  - $\forall i \neq j: C_i \cap C_j = \emptyset$
  - (sometimes) dendrogram

Classes of Algorithms for Clustering:

- Cost minimization algorithms
- Linkage-based algorithms

12.1. Cost Minimization Clustering

Approach:

- Define a cost function over possible partitions of the objects
- find the partition (=clustering) of minimal cost

Assumptions:

- Datapoints $\boldsymbol{x} \in X$ come from a larger space X', often $X \subseteq X' = \mathbb{R}^d$
- A distance function between each pair of datapoints is defined.

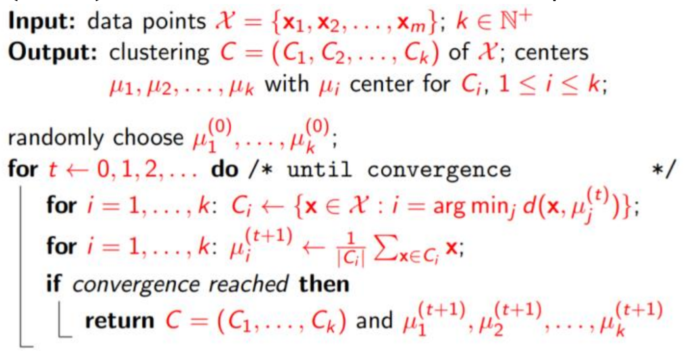
Data is partitioned into disjoint sets $C_1, \dots, C_k$ where each C_i is represented by a **centroid** μ_i

Objective(cost) functions:

- **k-means:** $\min_{\mu_1,\dots,\mu_k \in X'} \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} d(\boldsymbol{x}, \mu_i)^2$ measures the squared distance between each point in X to the centroid of its cluster.
- **k-medoids:** $\min_{\mu_1,\dots,\mu_k \in X} \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} d(\boldsymbol{x}, \mu_i)^2$ requires the cluster centroids to be members of the input set (X not X').
- **k-median:** $\min_{\mu_1,\dots,\mu_k \in X} \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} d(\boldsymbol{x}, \mu_i)$

To solve K-Means:

- **brute force (NP-Hard):** Try all possible partitions of the m points into k clusters, evaluate each partition, and find the best one. $O\left(\frac{k^m}{k!}\right)$
- **Lloyd's Algorithm** $O(tkmd)$, where t=#of iterations, m=#datapoints



```
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 $\mu_i$ center for $C_i$, $1 \le i \le k$;

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 $\mu_1^{(t+1)}, \mu_2^{(t+1)}, \dots, \mu_k^{(t+1)}$
```

Common Stop conditions (convergence):

- The k-means objective for the cluster at iteration t is not lower than the k-means objective for the cluster at iteration $t-1$. (Always terminates with this condition)
- $\sum_{i=1}^{k} d(\mu_i^{(t+1)}, \mu_i^{(t)}) \le \varepsilon$
- $\max_{1 \le i \le k} d(\mu_i^{(t+1)}, \mu_i^{(t)}) \le \varepsilon$

In practice it stops after just m iterations.

[Solution: Exercise 4]

Proof: let consider the cost function as a function of $\vec{\mu}_1, \vec{\mu}_2, \dots, \vec{\mu}_k$: $f(\vec{\mu}_1, \vec{\mu}_2, \dots, \vec{\mu}_k) = \sum_{i=1}^{k} \sum_{\vec{x} \in C_i} d(\vec{x}, \vec{\mu}_i)^2$

At the optimum, the gradient is equal to $\vec{0}$. Let's compute a part of the gradient, in particular $\partial f / \partial \vec{\mu}_j$: $\partial\left(\sum_{i=1}^{k} \sum_{\vec{x} \in C_i} d(\vec{x}, \vec{\mu}_i)^2\right) / \partial \vec{\mu}_j =$

$$= \sum_{i=1}^{k} \frac{\partial\left(\sum_{\vec{x} \in C_i} d(\vec{x}, \vec{\mu}_i)^2\right)}{\partial \vec{\mu}_j} = \frac{\partial\left(\sum_{\vec{x} \in C_j} d(\vec{x}, \vec{\mu}_j)^2\right)}{\partial \vec{\mu}_j}$$

$$= \sum_{\vec{x} \in C_j} \frac{\partial\, d(\vec{x}, \vec{\mu}_j)^2}{\partial \vec{\mu}_j} = \sum_{\vec{x} \in C_j} \frac{\partial (\vec{x} - \vec{\mu}_j)^T (\vec{x} - \vec{\mu}_j)}{\partial \vec{\mu}_j}$$

$$= \sum_{\vec{x} \in C_j} \left(-2\vec{x} + 2\vec{\mu}_j\right) = 2|C_j|\vec{\mu}_j - 2\sum_{\vec{x} \in C_j} \vec{x}.$$

At the optimum $2|C_j|\vec{\mu}_j - 2\sum_{\vec{x} \in C_j} \vec{x} = \vec{0}$

$$\Leftrightarrow \vec{\mu}_j = \frac{1}{|C_j|} \sum_{\vec{x} \in C_j} \vec{x}. \qquad \square$$

To improve the results of k-means it is often recommended to repeat the procedure several times with different randomly chosen initial centroids (e.g., we can choose the initial centroids to be random points from the data).

Kmeans++: provides a good solution, but it’s used to initialize centers in Lloyd’s.

$\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$, with $\mathbf{x}_i \in \mathbb{R}^d$ for $1 \leq i \leq m$; $k \in \mathbb{N}^+$

Given a point $\mathbf{x} \in \mathcal{X}$ and a set F, let $d(\mathbf{x}, F) = \min_{\mathbf{f} \in F} d(\mathbf{x}, \mathbf{f})$

The algorithm to compute the initial set F of centers is the following:

```
$\mu_1 \leftarrow$ random point from $\mathcal{X}$ chosen uniformly at random;
$F \leftarrow \{\mu_1\}$;
**for** $i \leftarrow 2$ **to** $k$ **do**
    $\mu_i \leftarrow$ random point from $\mathcal{X} \setminus F$, choosing point $\mathbf{x}$ with
     probability $\frac{(d(\mathbf{x},\mathbf{F}))^2}{\sum_{\mathbf{x'} \in \mathcal{X} \setminus F} (d(\mathbf{x'},\mathbf{F}))^2}$;
    $F \leftarrow F \cup \{\mu_i\}$;
**return** $F$;
```

12.2. Linkage-Based Clustering

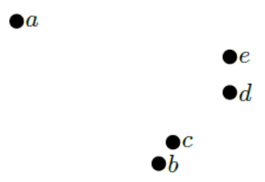
Algorithm:

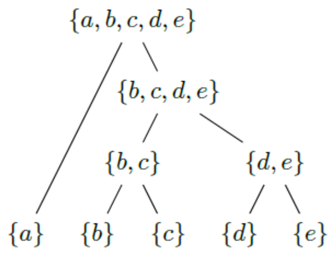
- Each data point is a single-point cluster.
- Then, repeatedly merge the "closest" clusters of the previous clustering.

Parameters:

- **Distance** between clusters, commons:
  - **Single linkage**: $D(A,B) = \min\{d(\boldsymbol{x},\boldsymbol{x}'): \boldsymbol{x} \in A, \boldsymbol{x}' \in B\}$ minimum distance between members of the two clusters.
  - **Average linkage:** $D(A,B) = \frac{1}{|A||B|}\sum_{\boldsymbol{x}\in A,\boldsymbol{x}'\in B} d(\boldsymbol{x},\boldsymbol{x}')$ average distance between a point in one of the clusters and a point in the other
  - **Max Linkage**: $D(A,B) = \max\{d(\boldsymbol{x},\boldsymbol{x}'): \boldsymbol{x} \in A, \boldsymbol{x}' \in B\}$ maximum distance between their elements
- **Termination** condition, common:
  - **Fixed number of clusters** k: stop merging clusters as soon as the number of clusters is k.
  - **Distance upper bound**: Stop merging as soon as all the between-clusters distances are $> r$.
  - all points are in a cluster => output is a dendrogram

Dendrogram: tree with input points $\boldsymbol{x} \in X$ as leaves, shows the arrangement/relation between clusters.





Finding the Optimal Number of Clusters

- Run the algorithm for various values of k
- Use a score S to evaluate each clustering
- Pick the value of k that has the maximum score.

Silhouette Coefficient: $s(\boldsymbol{x}) = \frac{B(\boldsymbol{x})-A(\boldsymbol{x})}{max(A(\boldsymbol{x}),B(\boldsymbol{x}))}$ measures if $\boldsymbol{x}$ is closer to points in its "nearest cluster" than to the cluster it is assigned to.

- $A(\boldsymbol{x}) = \frac{\sum_{x'\neq x, x'\in C(x)} d(x,x')}{|C(\boldsymbol{x})|-1}$ is the mean distance to the other instances in the same cluster
- $B(\boldsymbol{x}) = \min_{C_i \neq C(\boldsymbol{x})} d(\boldsymbol{x}, C_i)$ is the mean nearest-cluster distance, that is the mean distance to the instances of the next closest cluster.

Silhouette Score: $S(C) = \frac{\sum_{x\in X} s(x)}{|X|}$ the higher, the better the clustering quality.

Clustering for Predictions:

- goal: find centers to be used as representatives for predictions
- label of center = most frequent label in its cluster
- to predict label for a new point $\boldsymbol{x}$ predict the most frequent label of the labels of the k centers that are closest to $\boldsymbol{x}$ (K-nearest neighbors)
- choice of k: cross-validation

13. Dimensionality Reduction - PCA

Change of Basis: …..

Dimensionality reduction is the process of taking data in a high dimensional space and mapping it into a new space whose dimensionality is much smaller.
The reduction is performed by applying a linear transformation to the original data.
Find a good r-dimensional representation of D, with $r \ll d$

Given a base $\boldsymbol{u_1}, \dots, \boldsymbol{u_d} \in \mathbb{R}^d$ we consider the first r basis vectors $\boldsymbol{u_1}, \dots, \boldsymbol{u_r}$ and project $\boldsymbol{x}$ on those.

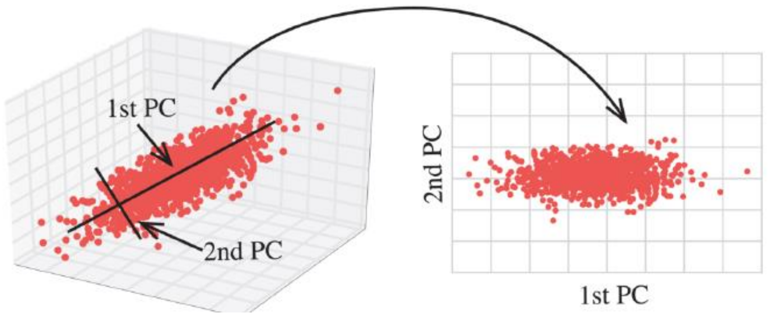
$$\boldsymbol{x}' = \sum_{i=1}^{r} a_i \boldsymbol{u}_i = [\boldsymbol{u_1} \dots \boldsymbol{u_r}] \begin{bmatrix} a_1 \\ \vdots \\ a_r \end{bmatrix} = U_r \boldsymbol{a_r} \in \mathbb{R}^d$$

$$\boldsymbol{a_r} = U_r^T \boldsymbol{x}$$

$$\boldsymbol{x}' = U_r U_r^T \boldsymbol{x} = P_r \boldsymbol{x}$$

Projection matrix $P_r = U_r U_r^T$, symmetric, $P_r^2 = P_r$
Error vector $\varepsilon = \sum_{i=r+1}^{d} a_i \boldsymbol{u}_i = \boldsymbol{x} - \boldsymbol{x}'$**,** orthogonal to $\boldsymbol{x}'$

Principal Component Analysis (PCA): seeks a r-dimensional basis that best captures the variance in the data. Identifies the hyperplane that lies closest to the data, and then it projects the data onto it.
1° principal component = direction with largest projected variance
2° principal component = orthogonal direction with largest projected variance
…



Data is given in a matrix D, where i-th row = vector $\boldsymbol{x}_i \in \mathbb{R}^d$ and is centered (mean=0)
Computing the principal components:

- compute the (sample) covariance matrix of (centered) data $\Sigma = D^T D$
- compute eigenvalues $\lambda_1 \geq \dots \geq \lambda_d \geq 0$ of Σ
- Let $\boldsymbol{u_1}, \dots, \boldsymbol{u_d}$ are the eigenvectors associated to the eigenvalues
- The principal components are the first r eigenvectors $\boldsymbol{u_1}, \dots, \boldsymbol{u_r}$

Projected Variance $\operatorname{var}(A) = \frac{1}{m}\sum_{i=1}^{m} \|\boldsymbol{a_i} - \boldsymbol{0}\|^2 = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x_i^T} P_r \boldsymbol{x_i} = \dots = \sum_{j=1}^{r} \lambda_j$ where A is the dataset projected
Mean Squared Error (MSE) for PCA: $\mathrm{MSE} = \operatorname{var}(D) - \operatorname{var}(A) = \sum_{j=1}^{d} \lambda_j - \sum_{j=1}^{r} \lambda_j$
PCA finds the basis of r vectors that minimizes MSE

Choice of $\boldsymbol{r}$: Pick smallest r such that $f(r) = \frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \geq \alpha$ (usually $\alpha \geq 0.9$)

Review-Questions

Those questions are not taken from exams, they are made by me just to study, but they turn out to be pretty effective.
You can study them for free in flashcard format at: https://www.brainscape.com/p/3YF5P-LH-B89RE

Chapter 2 - A gentle start

- What are the 6-key point of the Formal Model? What is the loss function? Draw the learning process.
- Define the training error and the ERM paradigm.
- ERM with inductive bias and finite hypothesis classes.
- (Simplied) PAC learning. With proof.

Chapter 3 - A formal model

- What is PAC learnability?
- What is sample complexity and what is its value for the PAC?
- What is Agnostic PAC learning? What are its empirical and true error?
- What is the Bayes optimal predictor?
- Definition of Agnostic PAC learnability.
- What is the general loss function? general risk function? general empirical risk?
- What are some common loss functions?
- What is Agnostic PAC Learning (for generalized loss functions)?

Chapter 4 - Uniform Convergence

- What is an e-representative training set?
- When ERM return a good hypothesis?
- When do we have uniform convergence?
- Which hypotheses classes have uniform convergence?
- What about finite H classes?

Chapter 5 - Bias tradeoff (slide 7)

- What the no free lunch theorem states? Informally and Formally.
- When H is not PAC learnable? How to avoid those failing distributions? How do we choose H?
- How we decompose the error?
- What is the bias complexity tradeoff?
- How can we estimate the generalization error?

Chapter 9 - Linear models

- Definition of Linear (affine) functions and its hypothesis classes. And equivalent notation.
- Halfspaces:
    - Definition of halfspace. When it is used?
    - When data are linearly separable?
    - Perceptron for halfspaces. When the algorithms stop?
- Linear regression:
    - What is, hypothesis class, loss function and empirical risk.
    - What is least squares?
    - How to find the solution of the ERM problem? What if the matrix is not invertible? (slide 8.4-11)

- Logistic regression:
    - What is? For what is used for?
    - What is its hypothesis class?
    - What are the differences with halfspaces?
    - What is the loss function?
    - What is the ERM problem? how can be solved?

Statistics: (Slide 9)

- What is a confidence interval? How it should be?
- What is MLE? (UML 24.1)
- How to find the MLE of θ?
- How is the smallest confidence set for θ?

Chapter 6 - VC dimension:

- What is the goal?
- What is the Restriction of H to C? What is shattering? Definition of VC-dimension of H
- What are the VCdim of Threeshold function, intervals, Axis aligned rectangles, Convex set,
- What The Fundamental Theorems of Statistical Learning states?
- When H is not pac learnable?
- If H is the class of halfspaces, VCdim=? (Dim)

Chapter 11 - Model Selection and Validation

- What is Model selection? What are the 2 approaches and their basic idea?
- Validation. What if V is large? How it is validation compared to VcDim?
- How does validation for model selection works? What the model selection curve plots?
- how can we estimate the true risk after model selection? Train-Validation-Test Split (slide 12.1)
- How k cross validation work? And for why is used?
- How to improve a model (what if learning fails)?

Chapter 13 - Regularization and Stability

- What is RML? What does?
- What is thikonov regularization function
- Ridge regression, derivation of optimal solution
- Fitting-Stability Tradeoff. How to pick λ?
- L1, LASSO; LASSO vs Ridge Regression.

Chapter 25 - Feature selection (25.1.2)

- What is feature selection? Why we use it?
- How Subset Selection works? Algorithm with training and with validation.
- How Forward Selection works? Algorithm with training and with validation.
- How Backward Selection works? Algorithm with training and with validation.

Chapter 15 - SVM

- When a training set is linearly separable? What is a margin? What is Hard-SVM? Equivalent formulations.
- How Soft-SVM works? Optimization problem, hinge formulation, homogeneous halfspaces.
- How to solve Soft-SVM? What is gradient descent? What is SGD? How to use SGD to solve Soft-SVM?
- What is the dual problem for hard-SVM? When we use this notation?
- How we can use SVM if we have non linearly separable data?
- What are the 2 issues generated by this procedure? How we can solve it?

- What is a kernel function? What is the kernel trick? (slide 16.8, 16.10) What are the most common kernels? How do we choose the kernel? What the mercer condition says?
- How to use SVM for regression? What is the function to minimize? What is the final model produced?

Chapter 20 - Neural networks

- What is a neuron? What is an activation function? What are the most common? What is the hypothesis class of a NN?
- What type of functions can be implemented using a neural network? Sample complexity. What is the runtime of learning a nn? What is the solution?
- Matrix notation. Forward propagation algorithm.
- How do we learn the parameters?
- What is a sensitivity vector?
- Backpropagation algorithm
- How we can regularize NN?
- What are the issues of traditional NN?
- What are CNN?
- How the Convolutional product works? What are its properties?
- What is padding? Relu? Pooling?
- How LeNet works?
- What is ADAM? How it works?
- What are the techniques to avoid overfitting? And how they works?
- What are RNN?
- Why DeepLearning works?

Chapter 22 - Clustering

- What is clustering (definition)?
- What are the difficulties of clustering?
- Model for clustering
- What are the 2 Classes of Algorithms for Clustering?
- Cost Minimization Clustering
    - What are the 3 main objective functions?
    - What is a brute force algorithm to sole K-means? What is its problem?
    - Lloyd's Algorthm, its convergence criteria, its complexity.
- Linkage-Based Clustering
    - How the algorithm works?
    - What are the 2 parameters that it needs?
    - What is a dendrogram? What it represents?
- How can we choose the number k of clusters? What is and how Silhouette works?

Chapter 23 - Dimensionality Reduction

- What is dimensionality reduction?
- What is PCA?
- How to compute the principal components?
- What does PCA on the projected dataset?
- How to choose the number of components?