MACHINE LEARNING FRAMEWORK (cap. 2)

- **Learner's INPUT**:
  - Domain set $\mathcal{X}$. domain points ⇝ INSTANCES
  - Label set $\mathcal{Y}$.
  - Training data (set) $S \in \mathcal{X} \times \mathcal{Y}$. It's a sequence of labeled domain points.
- **Learner's OUTPUT**: used to predict the label of new domain points
  - Prediction rule $h: \mathcal{X} \to \mathcal{Y}$ (also called predictor or hypothesis or classifier)

  ⇒ $A(S)$ is the hypothesis A is the learning algorithm
- **Simple data-generation model**:

  Assume that the training data is generated by a probability distribution $\mathcal{D}$ (over $\mathcal{X}$).

  (Suppose that the the learner doesn't know anything about the distribution)

  Suppose that there is some "correct" labeling function $f: \mathcal{X} \to \mathcal{Y}$ ⇒ $y_i = f(x_i)\ \forall i$

  (also unknown to the learner)

  Summary: each S is generated by first sampling a point x_i according to $\mathcal{D}$ and then labeling it by f.
- **Measures of success**:

  ERROR OF A CLASSIFIER (or RISK, GENERALIZATION ERROR, TRUE ERROR OF h): probability that it does not predict the correct label on a random data point generated by $\mathcal{D}$.

  ⇒ the error of h is the probability to draw a random instance x, according to $\mathcal{D}$, such that $h(x) \neq f(x)$

  Formally:

  $A \subset \mathcal{X}$ ⇒ $\mathcal{D}(A)$ determines how likely it is to observe a $x \in A$.

  $A = \{x \in \mathcal{X} : \pi(x) = 1\}$ with $\pi: \mathcal{X} \to \{0,1\}$ (we refer to A as an event)

  $\mathcal{D}(A) \equiv \mathbb{P}_{x \sim \mathcal{D}}[\pi(x)]$

  the error of $h: \mathcal{X} \to \mathcal{Y}$ is:

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

Note that $(\mathcal{D}, f)$ means that the error is computed with respect to the probability distribution $\mathcal{D}$ and the correct labeling function f.

• Remind: the learner is blind to the $\mathcal{D}$ and f.

EMPIRICAL RISK MINIMIZATION (ERM)

TRAINING SET S $(\mathcal{D}, f)$ $\longrightarrow$ LEARNING ALGORITHM $\longrightarrow$ $h_S : \mathcal{X} \to \mathcal{Y}$

GOAL: Find h_S that minimizes the error respect to the unknown $\mathcal{D}$ and f.

The TRAINING ERROR is defined as: (or EMPIRICAL ERROR, EMPIRICAL RISK)

$$L_S(h) \equiv \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m}$$ where $[m] = \{1, \dots, m\}$

Be careful $\rightsquigarrow$ overfitting

EMPIRICAL RISK MINIMIZATION WITH INDUCTIVE BIAS

Aim: find condition that guarantee no overfitting.

The learner chose before seeing the data a set of predictors $\Rightarrow \mathcal{H}$ hypothesis class

$ERM_{\mathcal{H}}(S) \in \underset{h \in \mathcal{H}}{\operatorname{argmin}}\, L_S(h)$ $\rightsquigarrow$ gives $h \in \mathcal{H}$ lowest possible

In that way we "bias" the predictor toward a particular set.

• FINITE HYPOTHESIS CLASSES

$\mathcal{H}$ finite class

$h_S \in \underset{h \in \mathcal{H}}{\operatorname{argmin}}\, L_S(h)$

Def (Realizability Assumption)

$\exists\, h^* \in \mathcal{H}$ s.t. $L_{(\mathcal{D},f)}(h^*) = 0 \quad \rightsquigarrow \quad L_S(h^*) = 0$

Assumption: the points of S are generated by $\mathcal{D}$ independently of each other.

They are INDIPENDENTLY and IDENTICALLY distributed.

i.i.d. ASSUMPTION $\longrightarrow$ $S \sim \mathcal{D}^m$ (m size of S)

$\mathcal{D}^m$: probability over m-tuples to pick each element indipendently

$\delta \equiv$ probability of getting a non representive sample

$(1-\delta) \equiv$ CONFIDENCE PARAMETER

$\varepsilon \equiv$ ACCURACY PARAMETER $\rightsquigarrow$ $L_{(\mathcal{D},f)}(h_S) > \varepsilon$ FAILURE of the learner, $L_{(\mathcal{D},f)}(h_S) \leq \varepsilon$ SUCCESS of the learner

AIM: upper bound probability to sample m-tuple of instances that will lead to failure of the learner.

we want to bo
upperbound: $\mathcal{D}^m(\{S|x = (x_1,\dots,x_m) : L_{(\mathcal{D},f)}(h_S) > \varepsilon\}) = P_{BAD}$

set of "bad" hypothesis: $\mathcal{H}_B = \{h \in \mathcal{H} : L_{(\mathcal{D},f)}(h) > \varepsilon\}$

misleading samples: $M = \{S|x : \exists h \in \mathcal{H}_B, L_S(h) = 0\}$

namely: $\forall S|x \in M \ \exists h \in \mathcal{H}_B$ that looks like a "good" hypothesis on $S|x$.

$\Rightarrow \{S|x : L_{(\mathcal{D},f)}(h_S) > \varepsilon\} \subseteq M \quad \Rightarrow M = \bigcup_{h \in \mathcal{H}_B} \{S|x : L_S(h) = 0\}$

Hence:

$\mathcal{D}^m(\{Sx : L_{(\mathcal{D},f)}(h_S) > \varepsilon\}) \le \mathcal{D}^m(M) = \mathcal{D}^m(\bigcup_{h \in \mathcal{H}_B} \{S|x : L_S(h) = 0\})$

Lemma (Union Bound)

A, B sets, $\mathcal{D}$ distribution: $\mathcal{D}(A \cup B) \le \mathcal{D}(A) + \mathcal{D}(B)$

Apply the UNION BOUND:

$\Rightarrow \mathcal{D}^m(\{S|x : L_{(\mathcal{D},f)}(h_S) > \varepsilon\}) \le \sum_{h \in \mathcal{H}_B} \mathcal{D}^m(\{S|x : L_S(h) = 0\})$ (1)

$\mathcal{D}^m(\{S|x : L_S(h) = 0\}) = \mathcal{D}^m(\{S|x : \forall i, h(x_i) = f(x_i)\}) = \prod_{i=1}^m \mathcal{D}(\{x_i : h(x_i) = f(x_i)\})$ (1)

For each individual sampling of an element of the training set we have

$\mathcal{D}(\{x_i : h(x_i) = y_i\}) = 1 - L_{(\mathcal{D},f)}(h) \le 1 - \varepsilon$ in fact $h \in \mathcal{H}_B$ (2)

Combine (1) with (2) and use $1 - \varepsilon \le e^{-\varepsilon}$:

$\forall h \in \mathcal{H}_B$ we have $\mathcal{D}^m(\{S|x : L_S(h) = 0\}) \le (1-\varepsilon)^m \le e^{-m\varepsilon}$ (2)

Combining (1) with (2):

$\mathcal{D}^m(\{S|x : L_{(\mathcal{D},f)}(h_S) > \varepsilon\}) \le |\mathcal{H}_B| e^{-\varepsilon m} \le |\mathcal{H}| e^{-\varepsilon m}$

($|\mathcal{H}_B|$: cardinality of $\mathcal{H}_B$; $\mathcal{H}_B \subset \mathcal{H}$)

Finally:

$P_{BAD} \le |\mathcal{H}| e^{-\varepsilon m} \overset{!}{\le} \delta$

$\rightarrow e^{-\varepsilon m} \le \frac{\delta}{|\mathcal{H}|} \rightarrow -\varepsilon m \le \log\left(\frac{\delta}{|\mathcal{H}|}\right) \rightarrow m \ge -\frac{1}{\varepsilon}\log\left(\frac{\delta}{|\mathcal{H}|}\right) \Rightarrow m \ge \frac{1}{\varepsilon}\log\left(\frac{|\mathcal{H}|}{\delta}\right)$

Corollary

$\mathcal{H}$ finite hypothesis class.

$\delta \in (0,1)$, $\varepsilon > 0$, $m \geq \frac{1}{\varepsilon} \log\left(\frac{|\mathcal{H}|}{\delta}\right)$

For any f and $\mathcal{D}$, for which the realizability assumption holds (for some $h \in \mathcal{H}$, $L_{(\mathcal{D},f)}(h) = 0$), with probability of at least $1-\delta$ over the choice of iid ~~assumption~~ sample S of size m, we have that for every hypothesis h_S, it holds that:

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

A FORMAL LEARNING MODEL (cap. 3)

PAC LEARNING (Probably Approximately Correct (PAC))

Def (PAC Learability)

$\mathcal{H}$ hyphothesis class is PAC LEARNABLE if $\exists\ m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following property

$\forall\ \varepsilon, \delta \in (0,1)$, $\mathcal{D}$ distribution (over $\mathcal{X}$), $f : \mathcal{X} \to \{0,1\}$

if realizable assumption holds with respect to $\mathcal{H}, \mathcal{D}, f$

Then when running the learning algorithm on $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$, the algorithm returns a hyphotesis h such that with probability at least $1-\delta$, $L_{(\mathcal{D},f)}(h_S) \leq \varepsilon$.

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

→ how many examples are required to guarantee a probably approximately correct solution

Corollary

Every finite class is PAC learnable with sample complexity

$$m_{\mathcal{H}}(\varepsilon, \delta) \leq \left\lceil \frac{1}{\varepsilon} \log\left(\frac{|\mathcal{H}|}{\delta}\right) \right\rceil$$

Generalization:

- removing the realizability assumption
- learning problems beyond binary classification

AGNOSTIC PAC LEARNING

$\mathcal{D}$ is joint probability over $\mathcal{X} \times \mathcal{Y}$

$\Rightarrow$ $\mathcal{D}_x$ distribution over domain points (MARGINAL DISTRIBUTION)

$\Rightarrow$ $\mathcal{D}((x,y)|x)$ CONDITIONAL DISTRIBUTION over labels for each domain points

• EMPIRICAL AND TRUE ERROR

$$L_{\mathcal{D}}(h) \equiv \mathbb{P}_{(x,y)\sim\mathcal{D}}[h(x) \neq y] \equiv \mathcal{D}(\{(x,y) : h(x) \neq y\})$$ TRUE ERROR

$$L_S(h) \equiv \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m}$$ EMPIRICAL ERROR

GOAL: Find $h: \mathcal{X} \to \mathcal{Y}$ that minimizes the true risk $L_{\mathcal{D}}(h)$.

• BAYES OPTIMAL PREDICTOR

Probability distribution: $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$.

The best predictor is the BAYES OPTIMAL PREDICTOR:

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

($\mathbb{P}[y=1 \mid x]$: this quantity is not known)

Proposition

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

Def (AGNOSTIC PAC Learnability)

IF $\exists$ $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following property

$\forall \varepsilon, \delta \in (0,1)$, $\mathcal{D}$ distribution over $\mathcal{X} \times \mathcal{Y}$,

when running the learning algorithm on $m \geq m_{\mathcal{H}}(\varepsilon,\delta)$ i.i.d examples generated by $\mathcal{D}$, the algorithm returns h s.t., with probability of at least $1-\delta$

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

Idea: we drop the requirement of finding the best predictor, but we do not want to be too far from it.

SCOPE OF LEARNING PROBLEMS

Domain set and training data have the same structure for all the problems.

Learner output: $h: \mathcal{X} \to \mathcal{Y}$ ($\mathcal{Y}$ is different for the 3 problems)

1) BINARY CLASSIFICATION

Target set: $Y = \{0, 1\}$

2) MULTICLASS CLASSIFICATION WITH $K > 2$ CLASSES

Target set: $Y = \{0, 1, \dots, K-1\}$

Loss: as for the binary case.

3) REGRESSION

Target set: $Y = \mathbb{R}$

Loss: need a new loss function

- GENERALIZED LOSS FUNCTION

H hypothesis class

Z domain $(X \times Y)$

$\ell : H \times Z \to \mathbb{R}_+$ LOSS FUNCTION

RISK FUNCTION: expected loss of an hypothesis $h \in H$ with respect to D over Z

$$L_D(h) \equiv \mathbb{E}_{z \sim D}[\ell(h, z)]$$

EMPIRICAL RISK: expected loss over a given training set $S = (z_1, \dots, z_m) \in Z^m$

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

- COMMON LOSS FUNCTION

- 0-1 loss: used in BINARY and MULTICLASS CLASSIFICATION

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

- square loss (L2): used in regression, penalize few large errors

$$\ell_{sq}(h, (x, y)) \equiv (h(x) - y)^2$$

- absolute value loss (L1): used in regression, penalize many small errors

$$\ell_{abs}(h, (x, y)) \equiv |h(x) - y|$$

DEF (Ag

H hypot

function

learning

$\forall \varepsilon,$

learnin

D, th

$1 - \delta$

L_D (

where

Remark

value

LEARN

H hypo

ERM le

- rec

- th

th

Def (Agnostic PAC learnability for general loss function)

H hypothesis class, is PAC learnable with respect to a set Z and a loss function $\ell : H \times Z \to \mathbb{R}_+$, if $\exists$ function $m_H : (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following property

$\forall\, \varepsilon, \delta \in (0,1)$ and $\forall\, D$ distribution over Z, when running the learning algorithm on $m \geq m_H(\varepsilon, \delta)$ i.i.d. examples generated by D, the algorithm returns $h \in H$ such that, with probability at least $1-\delta$:

$$L_D(h) \leq \min_{h' \in H} L_D(h') + \varepsilon$$

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

Remark $\ell(h, \cdot) : Z \to \mathbb{R}_+$ random variable and $L_D(h)$ is the expected value of this random variable.

LEARNING VIA UNIFORM CONVERGENCE (cap. 4)

H hypothesis class

ERM learning algorithm:

- receive a training sample S
- the learner evaluates the risk of each h in H on S
- the learner outputs $h^* \in H$, that minimizes this empirical risk.
- Hope: an h that minimizes the empirical risk w.r.t. S is a risk minimizer w.r.t. the true data probability distribution.

  $\Rightarrow$ SUFFICES THAT: to ensure that the empirical risk of all $h \in H$ is a good approximation of their true error

Def (ε-representative sample)

S training set is called ε-representative (w.r.t. Z, H, ℓ, D) if

$\forall\, h \in H, \; |L_S(h) - L_D(h)| \leq \varepsilon$

Lemma

Assume: S is $\frac{\varepsilon}{2}$ representative.

Then, any output of $ERM_H(S)$, namely any $h_S \in \operatorname{argmin}_{h \in H} L_S(h)$, satisfies

$$L_D(h_S) \leq \min_{h \in H} L_D(h) + \varepsilon$$

⇒ Consequence: if with probability $1-\delta$, a random training set S is ε-representive, then the ERM's rule is an agnostic PAC learner.

Def (uniform convergence)

H hypothesis class has the uniform convergence property (w.r.t. Z and ℓ) if there exists a function $m_H^{UC} : (0,1)^2 \to \mathbb{N}$, such that $\forall \varepsilon, \delta \in (0,1)$ and $\forall D$ over Z, if S is a sample of $m \geq m_H^{UC}(\varepsilon, \delta)$ examples drawn i.i.d. according to D, then with probability of at least $1-\delta$, S is ε-representative.

Corollary

If H has the uniform convergence property with m_H^{UC}, then the class is agnostically PAC learnable with the sample complexity $m_H(\varepsilon, \delta) \leq m_H^{UC}(\frac{\varepsilon}{2}, \delta)$

Furthermore, in that case the ERM_H paradigm is a successful agnostic PAC learner for H.

Corollary

Let H be a finite hypothesis class, Z a domain, $\ell : H \times Z \to [0,1]$ be a loss function. Then:

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

Proof.

1. $D^m(\{S : \forall h \in H, |L_S(h) - L_D(h)| \leq \varepsilon\}) \geq 1-\delta$
2. $P_{BAD} = D^m(\{S : \exists h \in H, |L_S(h) - L_D(h)| > \varepsilon\}) \leq \delta$

3. $\{S : \exists h \in H, |L_S(h) - L_D(h)| > \varepsilon\} = \bigcup_{h \in H} \{S : |L_S(h) - L_D(h)| > \varepsilon\}$

4. Union Bound:

$\Rightarrow D^m(\{S : \exists h \in H, |L_S(h) - L_D(h)| > \varepsilon\}) \leq \sum_{h \in H} D^m(\{S : |L_S(h) - L_D(h)| > \varepsilon\})$

5. Demonstrate that for any fixed h, the difference $|L_S(h) - L_D(h)|$ is small enough.

6. Recall that: $L_D(h) = \underset{z \sim D}{\mathbb{E}}[\ell(h, z)]$ and $L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h, z_i)$.

By the linearity of expectation, it follows that $L_D(h)$ is also the expected value of $L_S(h)$. Hence, the quantity $|L_D(h) - L_S(h)|$ is the deviation of the random variable $L_S(h)$ from its expectation.

→ Need to show that the measure of $L_S(h)$ is concentrated around its expected value.

7. Law of large numbers: if m is large, the average converges to the expectation.

8. Lemma (Hoeffding's inequality)

$\theta_1, \ldots, \theta_m$ is a sequence of i.i.d. random variables.

Assume that $\forall i$, $\mathbb{E}[\theta_i] = \mu$ and $\mathbb{P}[a \leq \theta_i \leq b] = 1$.

Then for any $\varepsilon > 0$:

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

$[0,1]$

9. $\Rightarrow D^m(\{S : |L_S^{(h)} - L_D(h)| > \varepsilon\}) = \mathbb{P}\left[\left|\frac{1}{m}\sum_{i=1}^{m} \theta_i - \mu\right| > \varepsilon\right] \leq 2\exp(-2m\varepsilon^2)$

10. $\sum_{h \in H} D^m(\{S : |L_S(h) - L_D(h)| > \varepsilon\}) \leq |H| \, 2\exp(-2m\varepsilon^2)$

$= \sum_{h \in H} 2\exp(-2m\varepsilon^2)$

11. Force:

$|H| \, 2e^{-2m\varepsilon^2} \overset{!}{\leq} \delta \Rightarrow m \geq \log\left(\frac{2|H|}{\delta}\right) \frac{1}{2\varepsilon^2}$

$\Rightarrow \sum_{h \in H} D^m(\{S : |L_S(h) - L_D(h)| > \varepsilon) \leq \delta$

□

Remark (The discretization trick)

simple trick that allows us to get a good estimate of the practical sample complexity of infinite hypothesis class

- Consider an hypothesis class determined by d real number parameters.
- Im principle: hypothesis class of infinite size
- Im practice: real numbers represented with 64 bits double precision variables.
- For d parameters: $|H| = 2^{64d}$ $\rightsquigarrow$ $|H|$ is larger but finite
- $m_H(\varepsilon, \delta) \leq m_H^{uc}\left(\frac{\varepsilon}{2}, \delta\right) \leq \frac{2\log(2\,2^{64d}/\delta)}{\varepsilon^2}$
- $\Rightarrow$ the bound depends on the chosen number representation.

THE BIAS COMPLEXITY TRADE-OFF (cap. 5)

A learning algorithm

S training set

$\mathcal{H}$ hypothesis set $\Rightarrow$ $\hat{h}$ from $\mathcal{H}$ for which $L_D(\hat{h})$ is small.

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

NO FREE-LUNCH THEOREM

Theorem (NO FREE-LUNCH)

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

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

$\Rightarrow$ meaning: there is no universal learner. Im fact, no learner can succeed on all learning tasks. For every learner, there exist a task on which it fails, even though that task can be successfully learned by another learner.

Corollary

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

Idea of the proof of the theorem: our training set is smaller than half of the domain, so we have no information on what happens on the other half. There exist some target function f that works on the other half in a way that contradicts our estimated labels.

Remark Class $\mathcal{H}$ of all possible functions from $\mathcal{X}$ to $\{0,1\}$ (i.e. assuming no prior knowledge) is not a good idea. → it is not PAC LEARNABLE.

Proof of the corollary.

proceed by contradiction:

1. Assume PAC learnable.
   - choose $\varepsilon < 1/8$, $\delta < 1/7$ (NB. $\mathcal{H}$ includes all functions)
   - By definition of PAC: $\exists$ an algorithm A s.t. For every distribution $\mathcal{D}$, if realizable, when running the algorithm on m i.i.d. examples generated by $\mathcal{D}$, the algorithm returns a hypothesis h such that, with probability $\geq 1-\delta$, $L_{\mathcal{D}}(A(S)) \leq \varepsilon$.
2. Apply no free-lunch theorem to A.
   - $|\mathcal{X}| > 2m$: for any ML algorithm (including A) there exist a distribution $\mathcal{D}$ for which with probability $\geq 1/7 > \delta$, $L_{\mathcal{D}}(A_S) \geq 1/8 > \varepsilon$.
3. Contradiction!

Remark (Choose a good hypothesis set)

- We need to use our prior knowledge about $\mathcal{D}$, to pick a good hypothesis set.
- We would like $\mathcal{H}$ to be large, so that it may contain a function h with small $L_S(h)$ and hopefully a small $L_{\mathcal{D}}(h)$.

- No Free-lunch: $\mathcal{H}$ cannot be too large!

$\mathcal{H}$ → SMALL: large L_S and $L_D \sim L_S$

$\mathcal{H}$ → LARGE: small L_S and $L_D \gg L_S$ (risk of overfitting)

ERROR DECOMPOSITION

h_S $\underset{\mathcal{H}}{\text{ERM}}$ hypothesis

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

where

ε_{app} = approximation error = $\min_{h \in \mathcal{H}} L_D(h)$

Minimum risk achievable by a predictor in the hypothesis class. Measures how much risk we have because we restrict ourselves to a specific class, namely, how much INDUCTIVE BIAS we have.

- depends on the choice of $\mathcal{H}$.
- does not depend on the sample size m.
- under the realizability assumption $\Rightarrow \varepsilon_{app} = 0$!
- larger $\mathcal{H}$ $\Rightarrow$ smaller ε_{app}.

ε_{est} = estimation error = $L_D(h_S) - \varepsilon_{app}$

Difference between the approximation error and the error achieved by the ERM predictor. The estimation error results because the empirical risk (training error) is only an estimate of the true risk, and so the predictor minimizing the empirical risk is only an estimate of the predictor minimizing the true risk.

- depends on the training set size m and on the size (complexity) of the hypothesis class:

  ε_{est} increase with $|\mathcal{H}|$ and decrease with m.

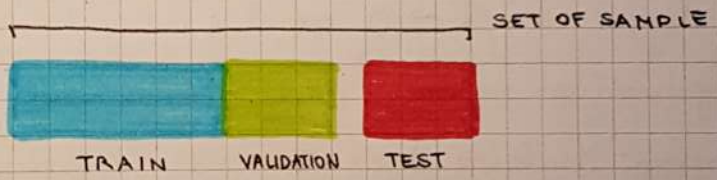
BIAS COMPLEXITY TRADE-OFF

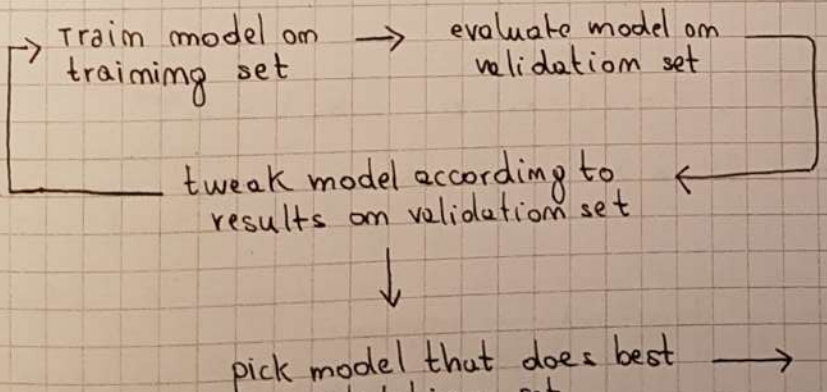
$\mathcal{H}$ large $\Rightarrow$ decrease ε_{app}, increase ε_{est} $\rightsquigarrow$ overfitting

$\mathcal{H}$ small $\Rightarrow$ decrease ε_{est}, increase ε_{app} $\rightsquigarrow$ underfitting

TRAIN, TEST AND VALIDATION SETS

- Estimate $L_D(h)$ (h selected with ERM)
- use a test set: a new set of samples not used for picking h
  - Different from the training set!
  - More reliable estimation of $L_D(h)$.
  - The test must not be looked at until we have picked our final hypothesis!
  - In practice: 1 set of sample splitted into training set and test set.
- Sometimes: training set divided into training set and validation set.
  - The validation set is used for selecting the hyper-parameters of the algorithm.





LINEAR PREDICTORS (cap. 9)

ERM approach.

hypothesis class → halfspaces (classification)
→ linear regression (regression)
→ logistic regression (classification modeled as a regression problem)

algorithms → linear programming (for halfspaces)
→ perceptron (for halfspaces)
→ least squares (for regression)

Class of AFFINE FUNCTIONS:

$$L_d = \{ h_{\bar{w},b}, \ \vec{w} \in \mathbb{R}^d, \ b \in \mathbb{R} \}$$

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

each member of L_d is a function $\bar{x} \to \langle \bar{w}, \bar{x} \rangle + b$.

⇒ dimensionality of parameters space: $d+1$

hypothesis class: $\phi : \mathbb{R} \to \mathcal{Y}$ on L_d.

- Binary classification: $\mathcal{Y} = \{-1, 1\} \to \phi(z) = \text{sign}(z)$
- Regression: $\mathcal{Y} = \mathbb{R} \to \phi(z) = z$

More convenient to write:

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

⇒ $h_{\bar{w},b}(\bar{x}) = \langle \bar{w}, \bar{x} \rangle + b = \langle \bar{w}', \bar{x}' \rangle$ becomes a linear function

~~⇒ $h_{\bar{w}} = \langle \bar{w}, \bar{x} \rangle$~~

HALFSPACES Designed for binary classification problems.

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

The class of halfspaces is defined as:

$$HS_d = \text{sign} \circ L_d = \{ \bar{x} \mapsto \text{sign}(h_{w,b}(\bar{x})) : h_{w,b} \in L_d \}$$ hypothesis class

Realizability: linearly separable.

- LINEAR PROGRAMMING

Maximize a linear function subject to linear inequalities.

TARGET:
Find $\max_{\vec{w} \in \mathbb{R}^d} \langle \bar{u}, \bar{w} \rangle$ subject to $A\bar{w} \geq \bar{v}$

$\bar{w} \in \mathbb{R}^d$ unknowns, $A \in m \times d$ matrix, $\bar{v} \in \mathbb{R}^m$, $\bar{u} \in \mathbb{R}^d$

Exist efficient linear programmming solvers.

Empirical risk Minimization (ERM) for halfspaces in the realizable case can be expressed as a linear program. In Fact:

$S = \{(\bar{x}_i, y_i)\}_{i=1}^{m}$ training set

- realizable assumption $\Rightarrow$ $L_S(h_S) = 0$ $\Rightarrow$ we are looking for $\bar{w}$ such that $y_i \langle \bar{w}, \bar{x}_i \rangle > 0 \quad \forall i = 1, \dots, m$
- There exist also a vector that satisfy $y_i \langle \bar{w}, \bar{x}_i \rangle \geq 1 \quad \forall i = 1, \dots, m$:

  proof. w^* satisfy $y_i \langle \bar{w}^*, x_i \rangle > 0 \quad \forall i$, $\hat{\bar{w}} = \frac{w^*}{\gamma}$ with $\gamma = \min_i (y_i \langle w^*, x_i \rangle)$

  $\Rightarrow \forall i \quad y_i \langle \hat{\bar{w}}, \bar{x}_i \rangle = \frac{1}{\gamma} y_i \langle \bar{w}^*, \bar{x}_i \rangle \geq 1$

  exist a vector such that

  $\Rightarrow y_i \langle \bar{w}, \bar{x}_i \rangle \geq 1 \quad \forall i$
- Model ERM with linear programming:

  $A \in m \times d$ matrix such that $A_{ij} = y_i x_{ij}$ $\quad x_{ij} = j$-element of the vector x_i

  $y_i \langle \bar{w}, \bar{x}_i \rangle \geq 1 \implies A\bar{w} \geq \bar{v}$ LP constraint $\bar{v} = (1, \dots, 1) \in \mathbb{R}^m$

  $\bar{u} = (0, \dots, 0) \in \mathbb{R}^d$ "dummy target" for $\max_{\bar{w} \in \mathbb{R}^d} \langle \bar{u}, \bar{w} \rangle$

- PERCEPTRON

Iterative algorithm.

TARGET: Find separating hyperplane.

→ vector $\bar{w}$ represents separating hyperplane

At each step focus on a missclassified sample and guide the algorithm to be "more correct" on it.

INPUT: training set $(x_1, y_1), \dots, (x_m, y_m)$

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

For $t = 1, 2, \dots$ do

  if $\exists i$ s.t. $y_i \langle w^{(t)}, x_i \rangle \leq 0$ then $w^{(t+1)} = w^{(t)} + y_i x_i$

  else return $w^{(t)}$

Note that: $y_i \langle w^{(t+1)}, x_i \rangle = y_i \langle w^{(t)} + y_i x_i, x_i \rangle = y_i \langle w^{(t)}, x_i \rangle + \|x_i\|^2$

Realizability assumption corresponds to linearly separable data.

Theorem

$(x_1, y_1), \dots, (x_m, y_m)$ is linearly separable

$B = \min \{ \|w\| : y_i \langle w, x_i \rangle \geq 1 \quad \forall i = 1, \dots, m \}$

$R = \max_i \|x_i\|$

The Perceptom algorithm stops after at most $(RB)^2$ iterations, and when it stops it holds that $\forall i \in [m]$, $y_i \langle w^{(t)}, x_i \rangle > 0$.

Proof.

1. Define:

   w^* vector achieving the min in B

   T number of iteration before stopping

2. Consider:

$$\frac{\langle w^*, w^{(T+1)} \rangle}{\|w^*\| \, \|w^{(T+1)}\|} \leq 1$$

3. We need to demonstrate: $\frac{\sqrt{T}}{RB} \leq \frac{\langle w^*, w^{(T+1)} \rangle}{\|w^*\| \, \|w^{(T+1)}\|} \leq 1 \Rightarrow T < (RB)^2$

4. Proceed in two parts:

   a) Numerator: demonstrate $\langle w^*, w^{(T+1)} \rangle \geq T$

   - first iteration: $w^{(1)} = (0, \dots, 0) \rightarrow \langle w^*, w^{(1)} \rangle = 0$
   - at each step: $\langle w^*, w^{(t+1)} \rangle - \langle w^*, w^{(t)} \rangle \geq 1$
   - after T iterations: $\langle w^*, w^{(T+1)} \rangle \geq T$

   in fact $\langle w^*, w^{(T+1)} \rangle = \sum_{t=1}^{T} \left(\langle w^*, w^{(t+1)} \rangle - \langle w^*, w^{(t)} \rangle \right) =$

   $= \sum_{t=1}^{T} \langle w^*, w^{(t+1)} - w^{(t)} \rangle = \sum_{t=1}^{T} \langle w^*, y_i x_i \rangle \geq T$

   b) Denominator: demonstrate $\|w^*\| \, \|w^{(T+1)}\| \leq \sqrt{T} \, RB$

   - $\|w^{(t+1)}\|^2 = \|w^{(t)} + y_i x_i\|^2 = \|w^{(t)}\|^2 + 2 y_i \langle w^{(t)}, x_i \rangle + y_i^2 \|x_i\|^2$

     $\leq \|w^{(t)}\|^2 + R^2$

   $\Rightarrow \|w^{(T+1)}\| = \sum_{t=1}^{T} \left(\|w^{(t+1)}\|^2 - \|w^{(t)}\|^2 \right) = \sum_{t=1}^{T} \left(\|w^{(t)} + y_i x_i\|^2 - \|w^{(t)}\|^2 \right) =$

   $= \sum_{t=1}^{T} \left(2 y_i \langle w^{(t)}, x_i \rangle + \|x_i\|^2 \right) \leq T R^2$

5) Combining we obtain:

$$\frac{\langle w^{(T+1)}, w^* \rangle}{\|w^*\| \, \|w^{(T+1)}\|} \geq \frac{T}{B\sqrt{T}R} = \frac{\sqrt{T}}{BR}$$

□

Remark

- Convergence is guaranteed!
- Convergence depends on B, which can be exponential in d

LINEAR REGRESSION

Estimate the relation between some explanatory variables and some real valuated outcome.

- $\mathcal{X} \in \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}$
- Find $h \in H_{reg} : \mathbb{R}^d \to \mathbb{R}$ that best approximates the relation between input and output, where

hypothesis class $H_{reg} = L_d = \{ \bar{x} \mapsto \langle \bar{w}, x \rangle + b : \bar{w} \in \mathbb{R}^d, b \in \mathbb{R} \}$

- Loss Function: L2 commonly used

$$\ell(h, (\bar{x}, y)) = (h(x) - y)^2$$

- Empirical risk Function: $L_S(h) = \frac{1}{m} \sum_{i=1}^{m} (h(x_i) - y_i)^2$ MEAN SQUARED ERROR

• LEAST SQUARE

It's the algorithm that solves the ERM problem for linear regression predictors with the squared loss.

ERM problem is to Find

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

Solve the problem: calculate gradient w.r.t. $\bar{w}$ and set to $\emptyset$:

$$\frac{\partial L}{\partial \bar{w}} = \frac{2}{m} \sum_{i=1}^{m} (\langle \bar{w}, \bar{x}_i \rangle - y_i) \bar{x}_i = 0$$

$$\Rightarrow A\bar{w} = \bar{b} \quad \text{with} \quad A = \left(\sum_{i=1}^{m} x_i x_i^T \right), \quad \bar{b} = \sum_{i=1}^{m} y_i \bar{x}_i$$

The solution is: $\bar{w} = A^{-1}\bar{b}$

• POLYNOMIAL REGRESSION

Find the one dimensional polynomial of degree m that better predicts the data

$$p(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_m x^m$$

$(a_0, \dots, a_m) \in \mathbb{R}^{m+1}$

$\mathcal{H}^m_{poly} = \{ x \mapsto p(x) \}$ hypothesis class

N.B. $\mathcal{X} = \mathbb{R}$, $\mathcal{Y} = \mathbb{R}$ ⇝ regression problem

⇒ reduce the problem to a m-dimensional linear regression using the map

$\psi : \mathbb{R} \to \mathbb{R}^{m+1}$ s.t. $\psi(x) = (1, x, x^2, \dots, x^m)$

we obtain

$$\langle \vec{a}, \psi(x) \rangle = a_0 + a_1 x + a_2 x^2 + \dots + a_m x^m$$

⇝ Find the optimal vector of coefficients $\vec{a}$ by using the Least Squares algorithm.

LOGISTIC REGRESSION

Reframe a classification problem as a regression one.

TARGET: learn a function $h : \mathbb{R}^d \to [0;1]$

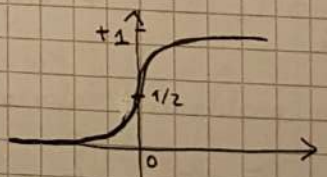
It's used for classification tasks: we can interpret $h(\bar{x})$ as the probability that the label of $\bar{x}$ is 1.

hypothesis class: $H : \phi_{sig} \circ L_d = \{ \bar{x} \mapsto \phi_{sig}(\langle \bar{w}, \bar{x} \rangle) : \bar{w} \in \mathbb{R}^d \}$

where

$\phi_{sig} : \mathbb{R} \to [0,1]$ sigmoid function over linear function class L_d

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



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

ERM problem: $\underset{\bar{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} \log(1 + \exp(-y_i \langle \bar{w}, \bar{x}_i \rangle))$

MLE

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.

⇒ is equivalent to ERM solution for logistic regression

- $S = ((\bar{x}_1, y_1), \dots, (\bar{x}_m, y_m))$ with each $(\bar{x}_i, y_i)$ i.i.d. From some probability distribution.
- $P[S(\theta)]$ (likelihood of data given parameters)
- log-likelihood: $L(S;\theta) = \log(P[S(\theta)])$
- maximum likelihood estimator: $\hat{\theta} = \arg\max_{\theta} L(S;\theta)$

THE VC-DIMENSION (cap.6)

AIM: answer at the question: which hypothesis class are PAC learnable?

Simplification: Focus on binary classification and 0-1 loss.

if $|\mathcal{H}| < \infty \Rightarrow \mathcal{H}$ is PAC learnable.

if $|\mathcal{H}| = \infty$?

⇒ we'll demonstrate that the finite size is a sufficient but not necessary condition

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

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

The restriction $\mathcal{H}_C$ of $\mathcal{H}$ to C is:

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

namely, the set of functions from C to $\{0,1\}$ that can be derived from $\mathcal{H}$. We represent each function from C to $\{0,1\}$, as a vector in $\{0,1\}^{|C|}$.

Def (Shattering)

A hypothesis class shatters a finite set $C \subset \mathcal{X}$ if the restriction of $\mathcal{H}$ to C is the set of <u>all</u> functions from C to $\{0,1\}$. That is, $|\mathcal{H}_C| = 2^{|C|}$.

Corollary (of NO-FREE LUNCH)

Let H be a hypothesis class of functions from $\mathcal{X}$ to $\{0,1\}$. Let m be a training size. Assume that there exists a set $C \subset \mathcal{X}$ of size $2m$ that is shattered by H. Then, for any learning algorithm, A, there exist a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$ and a predictor $h \in H$ such that $L_{\mathcal{D}}(h) = 0$ but with probability of at least $1/7$ over the choice of $S \sim \mathcal{D}^m$ we have that $L_{\mathcal{D}}(A(S)) \geq 1/8$.

Remark The corollary tell us that if H shatters some set C of size $2m$, then we cannot learn H using m samples.

Definition (VC-dimension)

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

NOTE: if H can shatter sets of arbitrarily large size, then we say that

$$VCdim(H) = +\infty.$$

Theorem

Let H be a class of infinite VC-dimension. Then, H is not PAC learnable.

Proof. consequence of previous NFL corollary, there will always exist a subset of size $2m$ that is shattered for any m.

Remark In the case of finite class hypothesis: $|H| < +\infty \Rightarrow VCdim(H) \leq \log_2 |H|$.

EXAMPLES

To show that $VCdimension(H) = d$ we need to show that:

1. $VCdim(H) \geq d$: there exists a set C of size d which is shattered by H.
2. $VCdim(H) < d+1$: every set of size $d+1$ is not shattered by H.

- THRESHOLD FUNCTIONS

$H = \{h_a : a \in \mathbb{R}\}$ with $h_a : \mathbb{R} \to \{0,1\}$ $h_a(x) = \mathbb{1}[x < a] = \begin{cases} 1 & \text{if } x < a \\ 0 & \text{if } x \geq a \end{cases}$

$VCdim(H) = 1$

- INTERVALS

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

$VCdim(H) = 2$

• AXIS ALIGNED RECTANGLES

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

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

$VCdim(H) = 4$

Remark. In general for K-dimension space: $VCdim\, H = K+1$

↓

• HALFSPACE HYPOTHESIS CLASS

• VC dim of homogeneous halfspaces in $\mathbb{R}^d$ is d.

• VC dim of non-homogeneous halfspaces in $\mathbb{R}^d$ is d+1.

FUNDAMENTAL THEOREM OF PAC LEARNING

Theorem (The Fundamental theorem of statistical learning)

Let H be a hypothesis class of functions from a domain X to $\{0,1\}$ and let the loss function be the 0-1 loss. Then, the following are equivalent:

1. H has the uniform convergence property.
2. Any ERM rule is a successful agnostic PAC learner for H.
3. H is agnostic PAC learnable.
4. H is PAC learnable.
5. Any ERM rule is successful PAC learner for H.
6. H has a Finite VC-dimension.

Notes on the demonstration:

1. We've already seen that $1 \Rightarrow 2 \Rightarrow 3$.
2. $3 \Rightarrow 4$ is trivial
3. $2 \Rightarrow 5$ also trivial
4. $4 \Rightarrow 6$ and $5 \Rightarrow 6$ (infinite VC is not PAC learnable)
5. The key part is how to close the loop $6 \Rightarrow 1$!

The proof of $6 \Rightarrow 1$ can be divided in two main parts:

- If $VCdim(H) = d$, then even though $|H|$ might be infinite, when restricting $|H|$ to a finite set C, it's "effective size" $|H_C|$, is only $O(|C|^d)$. That is, $|H_C|$ grows polynomially rather than exponentially with $|C|$. ← SAUER'S LEMMA

- Recall that infinite hypothesis classes enjoy the uniform convergence property. This result can be generalized by showing that uniform convergence holds whenever the hypothesis class has a "small effective size" (i.e., classes for which $|H_C|$ grows polynomially with $|C|$).

Theorem (The Fundamental theorem of statistical learning - quantitative)

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

1. H has the uniform convergence property with sample complexity

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

2. H is agnostic PAC learnable with sample complexity

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

3. H is PAC learnable with sample complexity

$$C_1 \frac{d + \log(1/\delta)}{\varepsilon} \leq m_H(\varepsilon, \delta) \leq C_2 \frac{d \log(1/\varepsilon) + \log(1/\delta)}{\varepsilon}$$

Definition (Growth function)

Let H be a hypothesis class. Then the growth function of H, denoted $\tau_H : \mathbb{N} \to \mathbb{N}$, is defined as

$$\tau_H(m) = \max_{C \subset \mathcal{X} : |C| = m} |H_C|$$

In words, $\tau_H(m)$ is the number of different functions from a set C of size m to $\{0,1\}$ that can be obtained by restricting H to C.

Notes:

- $\tau_H(m)$ is monotonically nondecreasing.
- $\tau_H(m)$ is distribution free (does not depend on D)
- what is the upper bound to $\tau_H(m)$?

Remark. If $VCdim(H) = d$, then for any $m \leq d$ we have $\tau_H(m) = 2^m$, therefore H induces all possible functions from C to $\{0,1\}$.

Lemma (Sauer–Shelah–Perles)

Let H be a hypothesis class with $VCdim(H) \leq d < \infty$. Then, for all m,

$$\tau_H(m) \leq \sum_{i=0}^{d} \binom{m}{i}.$$

N.B. $\binom{m}{i} = \frac{m!}{i!(m-i)!}$

In particular, if $m > d+1$, then

$$\tau_H(m) \leq \left(\frac{em}{d}\right)^d$$

N.B. In this case, it grows polynomially with m!

Remark. For $m > d+1$ the bound is much smaller than 2^m!

Theorem

Let H be a class and let τ_H be its growth function. Then, for every $\mathcal{D}$ and every $\delta \in (0,1)$, with probability of at least $1-\delta$ over the choice of $S \sim \mathcal{D}^m$ we have

$$|L_{\mathcal{D}}(h) - L_S(h)| \leq \frac{4 + \sqrt{\log(\tau_H(2m))}}{\delta\sqrt{2m}}$$

Proof idea. If we take two independent samples S, S' of size m call them, then $|L_S(h) - L_{S'}(h)|$ should look like an estimate for a hypothesis class of size $\tau_H(2m)$...

A FEW NOTES ON NON-UNIFORM LEARNABILITY (cap. 7) no exam

Def

A hypothesis class H is nonuniformly learnable if there exist a learning algorithm, A, and a function $m_H^{NUL} : (0,1)^2 \times H \to \mathbb{N}$ such that, for every $\varepsilon, \delta \in (0,1)$ and for every $h \in H$, if $m \geq m_H^{NUL}(\varepsilon, \delta, h)$ then for every distribution $\mathcal{D}$, with probability of at least $1-\delta$ over the choice of $S \sim \mathcal{D}^m$, it holds that

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

Key difference: m depends on h!

Recall the definition of AGNOSTIC PAC learnability.

→ we note that this definition implies that $\forall h \in H : L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(h) + \varepsilon$

⇒ AGNOSTIC PAC learnability ⇒ non-uniformly learnable

Theorem

A hypothesis class $\mathcal{H}$ of binary classifiers is nonuniformly learnable if and only if it is a countable union of agnostic PAC learnable hypothesis classes ($H = \bigcup_{m \in \mathbb{N}} H_m$, each H_m agnostic PAC).

Theorem

Let H be a hypothesis class that can be written as a countable union of hypothesis classes $H = \bigcup_{m \in \mathbb{N}} H_m$, where each H_m enjoys the uniform convergence property. Then H is non uniformly learnable.

Non-uniform learning can be achieved through the STRUCTURAL RISK MINIMIZATION (SRM) paradigm where we jointly minimize the empirical risk and a cost function depending on the complexity of the selected hypothesis

$$h \in \operatorname*{argmin}_{h \in H} \left[L_S(h) + \varepsilon_{m(h)}(m, w(m(h)) \cdot \delta) \right]$$

Assume a description language for the hypothesis class $\mathcal{H}$. Denote with $|h|$ the length of description of hypothesis $h \in H$

Theorem (Minimum description length) (MDL)

Let H be a hypothesis class, let $d : H \to \{0,1\}$ be a prefix-free description language for H. Then:

- For every sample size m
- For every $\delta > 0$
- For every probability distribution D

with probability $\geq 1-\delta$ (over the choice $S \sim D^m$) we have

$$\forall h \in H : L_D(h) \leq L_S(h) + \sqrt{\frac{|h| + \log(2/\delta)}{2m}}$$

The minimum description length (MDL) represents an alternative learning paradigm for $\mathcal{H}$ (in place of ERM):

prior Knowledge:

- $\mathcal{H}$ is countable hypothesis class
- $\mathcal{H}$ is described by a prefix-free language over $\{0,1\}$
- for every $h \in \mathcal{H}$: $|h|$ = length of representation of h

input:

training set $S \sim \mathcal{D}^m$, confidence δ

output:

$$\text{MDL}(S): h \in \operatorname*{argmin}_{h \in \mathcal{H}} \left[\underbrace{L_S(h)}_{\text{"bias"}} + \sqrt{\frac{\overbrace{|h|}^{\text{complexity}} + \ln(2/\delta)}{2m}} \right]$$

Occam's Razor:

"A short explanation (that is, a hypothesis that has a short length) tends to be more valid than a long explanation."

MODEL SELECTION AND VALIDATION (cap. 11)

Aim: choose the best algorithm for the particular problem.

There are different algorithms. Algorithms have parameters / design choices.

How to select the best?

Two approaches:

- STRUCTURAL RISK MINIMIZATION (SRM) → limited practical applicability → not part of the course
- Use a validation test

MODEL SELECTION WITH SRM (no exam)

- $\mathcal{H} = \bigcup_{m \in \mathbb{N}} \mathcal{H}_m$
- $\mathcal{H}_m$: uniform convergence property.
- sample complexity of $m^{UC}_{\mathcal{H}_m}(\varepsilon, \delta) \leq \frac{g(m)\log(1/\delta)}{\varepsilon^2}$

where $g: \mathbb{N} \to \mathbb{R}$ monotonically increasing

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

$$L_{\mathcal{D}}(h) \leq L_S(h) + \sqrt{\frac{g(d)(\log(1/\delta) + 2\log d + \log(\pi^2/6))}{m}}$$

SRM rule: pick d and $h \in H_d$ minimizing

$$L_S(h) + \sqrt{\frac{g(d)\,(\log(1/\delta) + 2\log(d) + \log(\pi^2/6))}{m}}$$

In many practical cases the upper bound is pessimistic.

VALIDATION SET

Idea: divide the training set in 2 parts, use the first to pick an hypothesis, and the second (not used to train) to estimate its true error.

Assume we have picked a predictor h (e.g. by ERM rule on H_d):

- V: set of m_v samples from D not used for training (validation set)
- L_V: loss computed on V (loss in $[0,1]$)

Theorem (1)

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, we have

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

Proof. using Hoeffding's inequality.

Remark. The bound in the last theorem does not depend on the algorithm or the training set used to construct h.

The bound based on the validation set is more accurate:

- there is no constant term
- does not depend on vc dimension (because validation samples have not been used for training).
- choose Final hypothesis by ERM over the validation set

To illustrate this points:

- suppose that h was obtained by applying an ERM predictor with respect to a hypothesis class of vc-dimension d, over a training set of m samples. Then, from the fundamental theorem of learning we obtain the bound

$$L_D(h) \leq L_S(h) + \sqrt{C\,\frac{d + \log(1/\delta)}{m}}$$

- In contrast, from the last theorem we obtain the bound

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

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

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

Remark. The empirical risk on the validation set is not an estimate of the true risk, in particular if we choose among many models!

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

Summary

We have to choose among multiple possible hypothesis set $\mathcal{H}_i$.

Approach: split the data in 3 parts.

1. TRAINING SET: used to learn the best model h_i inside each class $\mathcal{H}_i$.
2. VALIDATION SET: used to pick one hypothesis h^* from $h_1, h_2, \dots$.
3. TEST SET: used to estimate the true risk $L_{\mathcal{D}}(h^*)$.

Note that:

- the estimate from the test set has the guarantees provided by the proposition on estimate of $L_{\mathcal{D}}(h^*)$ for one class.
- The test set is not involved in the choice of h^*.
- If after using the test set to estimate $L_{\mathcal{D}}(h^*)$ we decide to choose another hypothesis (because we have seen $L_{\mathcal{D}}(h^*)$) we cannot use the test set again to estimate $L_{\mathcal{D}}(h^*)$!

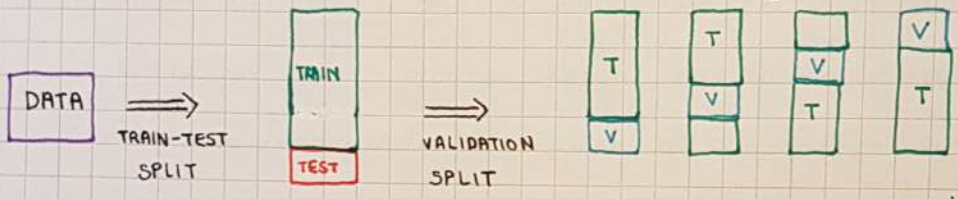
CIAO <3

K-FOLD CROSS VALIDATION

When data is not plentiful, we cannot afford to drop part of it to build the validation set. We use the K-fold cross validation.

K-fold cross validation:

1. Partition (training) set of m samples into K folds of size m/K.
2. For each fold:
   - train on union of the other folds
   - estimate error (for learned hypothesis) on the selected folds.
3. Estimate of the true error as the average of the estimated errors.



4-FOLD cross validation

Remark If K = m ⇒ leave-one-out (LOO) case.

K-fold cross validation is often used for model selection, and once the best parameter is chosen, the algorithm is retrained using this parameter on the entire training set.

Pseudocode:

input:

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

set of parameter values Θ

learning algorithm A

integer K

partition: S into $S_1, S_2, \ldots, S_K$

for each $\theta \in \Theta$

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:

$\theta^* = \text{arg min}_\theta [\text{error}(\theta)]$

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

WHAT TO DO IF LEARNING FAILS

Case: good training/validation errors, but results on test set are bad!

Main approaches to solve the problem:

- Get a larger sample.
- Change the hypothesis class by:
  - enlarging it
  - reducing it
  - completely changing it
  - changing the parameters you consider
- Change the feature representation of the data.
- Change the optimization algorithm used to apply your learning rule.

In order to find the best remedy, it is essential first to understand the cause of the bad performance.

Recall: h_S = learned predictor

- $L_D(h^*)$ APPROXIMATION ERROR (true error of best hypothesis in H).
- $L_D(h_S) - L_D(h^*)$ ESTIMATION ERROR
  ($L_D(h_S)$ → difference between the true error of best hypothesis in H and true error of ERM solution)
- $L_S(h_S)$ TRAINING ERROR (empirical error of ERM solution on training set S)
- $L_V(h_S)$ VALIDATION ERROR (error on validation set V of ERM solution)

We decomposed the true error of the learned predictor into:

$$L_D(h_S) = L_D(h^*) + (L_D(h_S) - L_D(h^*))$$

using train and validation errors

$$L_D(h_S) = \underbrace{(L_D(h_S) - L_V(h_S))}_{\text{can be bounded by theorem (1)}} + \underbrace{(L_V(h_S) - L_S(h_S))}_{\text{if is large} \rightarrow \text{overfitting}} + \overbrace{L_S(h_S)}^{\text{if is large} \rightarrow \text{underfitting}}$$

Remark

- The approximation error of the class does not depend on the sample size or on the algorithm being used. It only depends on the distribution D and on the hypothesis class H.
- The estimation error of the class does depend on the sample size.

The following steps should be applied:

1. If learning involves parameter tuning, plot the model-selection curve to make sure that you tuned the parameters appropriately.
2. If the training error is excessively large consider enlarging the hypothesis class, completely change it, or change the feature representation of the data.
3. ⇒ If the training error is small, plot learning curves and try to deduce from them whether the problem is estimation error or approximation error.
4. If the approximation error seems to be small enough, try to obtain more data. If this is not possible, consider reducing the complexity of the hypothesis class.
5. If the approximation error seems to be large as well, try to change the hypothesis class or the feature representation of the data completely.

$$L_S(h_S) = (L_S(h_S) - L_S(h^*)) + (L_S(h^*) - L_D(h^*)) + L_D(h^*)$$

Remark For (3): if the validation error seems to decrease (the error is large but the two curves get closer):

- get more data (if possible)
- otherwise reduce complexity of $\mathcal{H}$

If the validation error remains large:

- change $\mathcal{H}$
- change feature representation of the data

REGULARIZATION AND STABILITY (cap. 13)

The new learning paradigm we introduce in this chapter is called REGULARIZED LOSS MINIMIZATION (RLM).

Key idea: jointly minimize empirical risk and a regularization function.

- hypothesis h: defined by a vector $\vec{w} = (w_1, \dots, w_d)^T \in \mathbb{R}^d$ (e.g. coefficients of a linear model, weights in a neural network, etc.)
- Regularization function: $R : \mathbb{R}^d \to \mathbb{R}$, function of $\vec{w}$.

- Regularized loss minimization (RLM): select h from

$$\operatorname{argmin}_{\vec{w}} \left(L_S(\vec{w}) + R(\vec{w}) \right)$$

- $L_S(\vec{w})$: standard loss as previously
- $R(\vec{w})$: regularization term (measure in some way the "complexity" of the found solution)

The regularization term balances between low empirical risk and aiming at less complex hypothesis.

It is possible to view the extra term as a "stabilizer".

TIKHONOV REGULARIZATION

Define a function R using the ℓ_2 norm:

$$R(\vec{w}) = \lambda \|\vec{w}\|^2 = \lambda \sum_{i=1}^{d} w_i^2 \quad \text{where } \lambda > 0 \text{ is a scalar}$$

The output of the function R is a real positive number.

This yields the learning rule:

$$A(S) = \operatorname{argmin}_{\vec{w}} \left(L_S(\vec{w}) + \lambda \|\vec{w}\|^2 \right)$$

Remark $\|\vec{w}\|^2$ measures the "complexity" of the hypothesis defined by $\vec{w}$.
Instead, λ controls the trade-off between low empirical risk or high complexity (risk of overfitting).

RIDGE REGRESSION

Linear regression with squared loss + Tikhonov regularization

- Linear regression with squared loss:
  find $\vec{w}$ that minimizes the squared loss

$$\vec{w} = \operatorname{argmin}_{\vec{w}} \sum_{i=1}^{m} \left(\langle \vec{w}, \vec{x}_i \rangle - y_i \right)^2$$

- Ridge regression: find $\vec{w}$ that minimizes

$$\vec{w} = \operatorname{argmin}_{\vec{w}} \left(\lambda \|\vec{w}\|^2 + \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(\langle \vec{w}, \vec{x}_i \rangle - y_i \right)^2 \right)$$

Closed form solution:

aim: find optimal $\vec{w}$

1. Compute gradient w.r.t. $\vec{w}$ and set to $\emptyset$

$$2\lambda \|\vec{w}\| + \frac{1}{m} \sum_{i=1}^{m} \left(\langle \vec{w}, \vec{x}_i \rangle - y_i \right) x_i = \emptyset$$

2. Define

$$A = \left(\sum_{i=1}^{m} \vec{x}_i \vec{x}_i^T \right) \qquad \vec{b} = \sum_{i=1}^{m} y_i \vec{x}_i$$

3. The solution is

$$\vec{w} = (2\lambda m \mathbb{1} + A)^{-1} \vec{b}$$

Remark Tikhonov regularization makes the learner stable w.r.t. small perturbations of the training set. This in turns leads to small bounds on generalization error.

STABILITY

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

Let A be a learning algorithm, let $S = (z_1, \dots, z_m)$ be a training set of m samples and let $A(S)$ denote the output of A.

Let $S^{(i)}$ be the training set obtained by replacing the i'th example of S with z': namely, $S^{(i)} = (z_1, \dots, z_{i-1}, z', z_{i+1}, \dots, z_m)$.

In our informal definition:

- "a small change of the input" means that we feed A with $S^{(i)}$ instead of with S. That is, we only replace one training sample.
- "a small change of its output hypothesis" means a small change in the loss → ON-AVERAGE-REPLACE-ONE-STABLE (OAROS) algorithms.

Def (ON-AVERAGE-REPLACE-ONE-STABLE)

Let $\varepsilon : \mathbb{N} \to \mathbb{R}$ be a monotonically decreasing function. We say that a learning algorithm A is on-average-replace-one-stable (OAROS) with rate $\varepsilon(m)$ if for every distribution $\mathcal{D}$:

$$\mathbb{E}_{(S, z') \sim \mathcal{D}^{m+1}, i \sim U(m)} \left[\ell(A(S^{(i)}), z_i) - \ell(A(S), z_i) \right] \leq \varepsilon(m)$$

Remark $U(m)$ is an uniform distribution over $[m]$

Proposition

If algorithm A is OAROS with rate $\varepsilon(m)$ then:

$$\mathbb{E}_{S\sim\mathcal{D}^m}\left[L_{\mathcal{D}}(A(S)) - L_S(A(S))\right] \leq \varepsilon(m)$$

$L_{\mathcal{D}}(A(S))$ = true risk of output
$L_S(A(S))$ = empirical risk of output

Proof. Since S and z' are both drawn i.i.d. from $\mathcal{D}$, we have that for every i:

$$\mathbb{E}_S\left[L_{\mathcal{D}}(A(S))\right] = \mathbb{E}_{S,z'}\left[\ell(A(S), z')\right] = \mathbb{E}_{S,z'}\left[\ell(A(S^{(i)}), z_i\right]$$

On the other hand we have:

$$\mathbb{E}_S\left[L_S(A(S))\right] = \mathbb{E}_{S,i}\left[\ell(A(S), z_i)\right] \quad (?)$$

The proof follow from the definition of stability. □

The proposition tell us that a learning algorithm does not overfit if and only if it is on-average-replace-one-stable —

→ Stable rules do not overfit.

TIKHONOV REGULARIZATION AS A STABILIZER

Def (strongly convex functions)

A function f is λ-strongly convex if for all $\vec{w}, \vec{u}$, and $\alpha \in (0,1)$ we have

$$f(\alpha\vec{w} + (1-\alpha)\vec{u}) \leq \underbrace{\alpha f(\vec{w}) + (1-\alpha) f(\vec{u})}_{\text{convex}} - \underbrace{\frac{\lambda}{2}\alpha(1-\alpha)\|\vec{w}-\vec{u}\|^2}_{\text{margin ("strongly")}}$$

Lemma

1. The function $f(\vec{w}) = \lambda\|w\|^2$ is 2λ-strongly convex.
2. If f is λ-strongly convex and g is convex, then $f+g$ is λ-strongly convex.
3. If f is λ-strongly convex and $\vec{u}$ is a minimizer of f, then, for any $\vec{w}$,

$$f(\vec{w}) - f(\vec{u}) \geq \frac{\lambda}{2}\|\vec{w}-\vec{u}\|^2$$

Def (LIPSCHITZNESS)

Let $C \subset \mathbb{R}^d$, a function $f: \mathbb{R}^d \to \mathbb{R}^k$ is ρ-Lipschitz over C if $\forall \vec{w}_1, \vec{w}_2 \in C$ we have that $\|f(\vec{w}_1) - f(\vec{w}_2)\| \leq \rho\|\vec{w}_1 - \vec{w}_2\|$

Remark Intuitively it means that the function cannot change too fast. Moreover, for derivable functions corresponds to bound on derivative: if derivative bounded by ρ at any point, the function is ρ-Lipschitz.

Corollary

Assume that the loss function is convex and ρ-Lipshitz continuous. Then, the RLM rule with regularizer $\lambda \|\vec{w}\|^2$ is OAROS with rate $\frac{2\rho^2}{\lambda m}$. It follows that for the RLM rule:

$$\mathbb{E}_{S \sim \mathcal{D}^m} [L_{\mathcal{D}}(A(S)) - L_S(A(S))] \leq \frac{2\rho^2}{\lambda m}$$

CONTROLLING THE FITTING-STABILITY TRADE-OFF

We can rewrite the expected risk of a learning algorithm as

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

- $\mathbb{E}_S [L_S(A(S))]$: reflects how well A fits the training set S.
- $\mathbb{E}_S [L_{\mathcal{D}}(A(S)) - L_S(A(S))]$: reflects the difference between the true and empirical risks of A(S). It is equivalent to the stability of A.

In Tikhonov regularization, λ controls tradeoff between the two terms

- How do $L_S(A(S))$ and $\|\vec{w}\|^2$ vary as a function of λ?
  → Larger λ leads to higher empirical risk $L_S(A(S))$.
- How may $\mathbb{E}_S [L_{\mathcal{D}}(A(S)) - L_S(A(S))]$ change as a function of λ?
  → On the other side increasing λ, the stability term decreases.
- How to set λ?
  → theoretical bound in the book.
  → In practice, validation error is used.

→ Corollary

Assume that the loss function is convex and ρ-Lipshitz. Then the RLM rule with the regularization function $\lambda \|\vec{w}\|^2$ satisfies

$$\forall \vec{w}^*, \ \mathbb{E}_S [L_{\mathcal{D}}(A(S))] \leq L_{\mathcal{D}}(\vec{w}^*) + \lambda \|\vec{w}^*\|^2 + \frac{2\rho^2}{\lambda m}$$

This bound is often called "oracle inequality".

→ If w^* is an hypothesis with low risk, the bound tell us how many examples we need to get close to w^*.

→ But there is a problem...

- need to know $\|w^*\|$: in practice not known.
- it is more practical to use a validation set!

STOCHASTIC GRADIENT DESCENT (cap. 14)

Aim: need a general approach to minimize a differentiable convex function $f(\vec{w})$.

Recall: the gradient $\nabla f(\vec{w})$ of a differentiable function $f: \mathbb{R}^d \to \mathbb{R}$ is

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

Idea: the gradient points in the direction of the largest increase of f close to $\vec{w}$.

- Move in the opposite direction until you find a minima.
- Gradient correspond to first order Taylor approximation.
  → It is good for small steps → need to move step by step.
- The theory can be extended to non-differentiable functions using subgradients.

GRADIENT DESCENT (GD)

General approach to minimize a differentiable convex function $f(\vec{w})$.

GD ALGORITHM:

$\vec{w}^{(0)} \leftarrow 0$

For $t \leftarrow 0$ to $T-1$ do

$\vec{w}^{(t+1)} \leftarrow \vec{w}^{(t)} - \eta \nabla f(\vec{w}^{(t)})$;

return $\vec{w} = \vec{w}^{(T)}$

GD iterations

- start from an initial point
- at each step move in direction opposite to the gradient
- stop when solution does not improve or max iterations reached
- get the final point or the one corresponding to minimum value of the objective functions.

Remark. $\eta > 0$.

Hypothesis:

- $f(\vec{w})$ is a convex ρ-Lipschitz function
- $\vec{w}^* \in \operatorname{argmin}_{\{\vec{w}: \|\vec{w}\| \le B\}} f(\vec{w})$

We have the following Corollary:

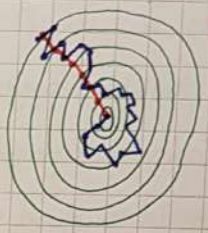
Corollary Let f be a convex, ρ-Lipschitz function, and let $\vec{w}^* \in \operatorname{argmin}_{\{\vec{w}:\|\vec{w}\|\le B\}} f(\vec{w})$. If we run the GD algorithm on f for T steps with $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$, then the output vector $\vec{w}$ satisfies

$$f(\vec{w}) - f(\vec{w}^*) \le \frac{B\rho}{\sqrt{T}}$$

Furthermore, for every $\varepsilon > 0$, to achieve $f(\vec{w}) - f(\vec{w}^*) \le \varepsilon$ it suffices to run the GD algorithm for a number of iterations that satisfies $T \ge \frac{B^2\rho^2}{\varepsilon^2}$.

STHOCASTIC GRADIENT DESCENT (SGD)

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



SGD algorithm:

$\vec{w}^{(0)} \leftarrow 0$

For $t \leftarrow 0$ to $T-1$ do

choose $\vec{v}_t$ random from a distribution such that $\mathbb{E}[\vec{v}_t \mid \vec{w}^{(t)}] \in \nabla f(\vec{w}^{(t)})$;

$\vec{w}^{(t+1)} \leftarrow \vec{w}^{(t)} - \eta \vec{v}_t$;

return $\bar{\vec{w}} = \frac{1}{T}\sum_{t=1}^{T} \vec{w}^{(t)}$; or $\vec{w} = \vec{w}^{(T)}$

SGD VS GD

Why should we use SGD instead of GD?

Consider the ERM setting: Find $\vec{w}$ that minimizes $L_S(\vec{w})$, i.e., $f(\vec{w}) = L_S(\vec{w})$.

- Using GD:
  - $\nabla f(\vec{w})$ depends on <u>all</u> the m pairs $(\vec{x}_i, y_i) \in S$.
  - Need to process all the training set at each iteration.
  - Very long computation time if training set is large.
- Using SGD:

need to pick $\vec{v}_t$ such that $\mathbb{E}[\vec{v}_t \mid \vec{w}^{(t)}] = \nabla f(\vec{w}^{(t)})$

- pick a random $(\vec{x}_i, y_i) \in S \Rightarrow \vec{v}_t = \nabla \ell(\vec{w}^{(t)}, (\vec{x}_i, y_i))$ (?)
- satisfies the requirement
- can be computed from just a single sample! → much faster!

Remark. For SGD we have less stable trajectory:

- More "noisy" but could jump out of local minima.
- Advanced approaches to stabilize, e.g., momentum.
- Sometimes the final point is computed as average of a set of samples to account for fluctuations.
- Better to average only a set of final iterations, not all
- Alternative approach to get a stable result: use an adaptive step size.

GRADIENT DESCENT: VARIANTS

1. BATCH GRADIENT DESCENT (standard GD): compute the gradient over the complete training set.
2. MINI-BATCH GRADIENT DESCENT: compute the gradient over a small set of K samples.
   - K: parameter, mini-batch size.
   - Trade-off between the two "extreme" cases GD and SGD.
   - Used to train deep neural networks.
3. STOCHASTIC GRADIENT DESCENT (SGD): use a single sample to estimate the gradient.

SGD: APPLICATIONS

Use SGD to solve ML problems:

1. RISK MINIMIZATION (ERM).
2. REGULARIZED LOSS MINIMIZATION (RLM) ← details not part of the course
3. Support vector machines (SVM).
4. Neural networks.

SGD FOR RISK MINIMIZATION

- SGD allows us to minimize $L_D(\vec{w})$ directly
- All we need is to find an unbiased estimate of the gradient of $L_D(\vec{w})$, that is a random vector whose conditional expected value is $\nabla L_D(\vec{w}^{(t)})$
- Sample a single fresh sample and estimate the gradient with it.

· can be applied to RLM solving its target.

Algorithm: SGD for minimizing $L_D(\vec{w})$

parameters: scalar $\eta > 0$, integer $T > 0$

initialize: $w^{(1)} = 0$

For $t = 1, 2, \ldots, T$

sample $z \sim D$

pick $\vec{v}_t \in \vec{\nabla} \ell(\vec{w}^{(t)}, z)$

update $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \vec{v}_t$

output $\overline{\vec{w}} = \frac{1}{T} \sum_{t=1}^{T} \vec{w}^{(t)}$

We shall now use our analysis of SGD to obtain a sample complexity analysis for learning convex-Lipschitz-bounded problems:

Corollary

Consider a convex-Lipschitz-bounded learning problem with parameters ρ, B. Then, for every $\varepsilon > 0$, if we run the SGD method for minimizing $L_D(\vec{w})$ with a number of iterations (i.e. number of samples)

$$T \geq \frac{B^2 \rho^2}{\varepsilon^2}$$

and with $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$, then the output of SGD satisfies

$$\mathbb{E}[L_D(\vec{w})] \leq \min_{\vec{w} \in H} L_D(\vec{w}) + \varepsilon$$

· SGD: ISSUES

· The selection of the learning rate η is a critical point
  → if it is too small the convergence can be very slow;
  → if it is too large the optimization can be very unstable.

· Simple solution: use adaptive learning rates, e.g.,
  ⇝ progressively reducing the learning rate according to a pre-defined schedule
  ⇝ reduce it when the change in the loss becomes too small
  ⇝ however, these approaches requires rules and thresholds to be defined in advance and thus are difficult to adapt to different problems.

- Additionally, the same learning rate applies to all parameters updates
  → the various parameters have different behaviours and the learning rate could be too fast for some and too slow for others.
  → sometimes better not to update all parameters to the same extent, but perform a larger update for some and smaller for others.

MOMENTUM

SGD has troubles (i.e. it oscillates) in areas where the surface curves much more steeply in one dimension than in another (which are common around local optima).

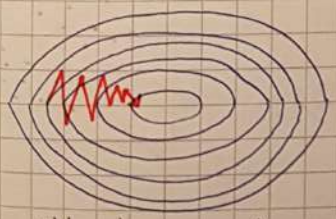
→ Therefore, we consider the MOMENTUM: in that way, the update is the linear combination of previous gradient and new one.

The momentum parameter γ is usually set to 0.9 or a similar value

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

It helps accelerate SGD in relevant direction and dampens oscillations.

Remark. The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.



SGD without momentum

SGD with momentum

ADVANCED SGD SCHEMES & algorithms

- ADAGRAD adapts the learning rate for each parameter independently
  → it performs smaller updates (i.e. low learning rates) for parameters associated with ~~infrequent features~~ frequently occurring features.
  → it performs larger updates (i.e. high learning rates) for parameters associated with infrequent features.
- ADADELTA (improved version of ADAGRAD)

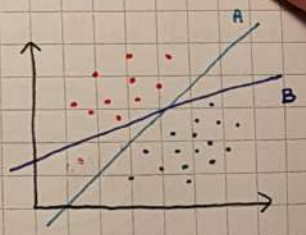
- RMSSprop (improved version of ADAGRAD).
- ADAM (Adaptive Moment Estimation)
  → it also computes adaptive learning rates for each parameter.
  → it combines ideas from ADAGRAD and momentum.
  → whereas momentum can be seen as a ball running down a slope, ADAM behaves like a heavy ball with friction, which thus prefers flat minima in the error surface.

SUPPORT VECTOR MACHINES (cap. 15)

Aim: learning linear predictors in high dimensional feature spaces.
The SVM algorithmic paradigm tackles the sample complexity challenge by searching for "large margin" separators. Roughly speaking, a halfspace separates a training set with a large margin if all the examples are not only on the correct side of the separating hyperplane but also far away from it. Restricting the algorithm to output a large margin separator can yield a small sample complexity even if the dimensionality of the feature space is high (even infinite).

CLASSIFICATION MARGIN

Consider a classification problem with two classes:
Training data: $S = ((\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m))$
with $\vec{x}_i \in \mathbb{R}^d$ and $y_i \in \{\pm 1\}$



Hypothesis set: $\mathcal{H}$ = halfspaces
Assumption: the data is linearly separable; there exist a halfspace that perfectly classify the training set.
Find a solution: there are multiple separating hyperplanes that correctly classify the training set, which one is the best?
To answer, we use the concept of "margin".
MARGIN: minimum distance from an example in the training set.
Idea: best separating hyperplane is the one with the largest margin
↝ can tolerate more "noise".

LINEARLY SEPARABLE TRAINING SET:

A traiming set $S = ((\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m))$ is linearly separable if there exists a halfspace $(\vec{w}, b)$ such that $y_i = \text{sign}(\langle \vec{w}, \vec{x}_i \rangle + b) \; \forall i=1,\dots,m$

- i.e., it perfectly separate all data im the traiming set.
- or, equivalently $\forall i : y_i (\langle \vec{w}, \vec{x}_i \rangle + b) > 0$

MARGIN:

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

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

Claim

The distance between a point $\vec{x}$ and the hyperplane defined by $(\vec{w}, b)$ if $\|\vec{w}\| = 1$ is $d(\vec{x}, L) = |\langle \vec{w}, \vec{x} \rangle + b|$

Im this case $(\|\vec{w}\| = 1)$ the margim is:

$$\min_i |\langle \vec{w}, \vec{x}_i \rangle + b|$$

The closest examples are called support vectors.

SUPPORT VECTOR MACHINE (SVM)

HARD-SVM: seek for the separating hyperplane with largest margim.

(NB. works only for linearly separable data)

Computational problem:

$$\arg\max_{(\vec{w},b) : \|\vec{w}\|=1} \min_i |\langle \vec{w}, \vec{x}_i \rangle + b| \quad \text{s.t.} \quad \forall i, \; y_i (\langle \vec{w}, \vec{x}_i \rangle + b) > 0$$

(linearly separable)

equivalent formulation:

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

An equivalent formulation of the HARD-SVM rule is expressing it as a quadratic formulation problem:

ALGORITHM: HARD-SVM

INPUT: $S = ((\vec{x}_1, y_1), \dots, (\vec{x}_m, \vec{y}_m))$

SOLVE:

$$(\vec{w}_0, b_0) = \underset{(\vec{w}, b)}{\operatorname{argmin}} \|\vec{w}\|^2 \quad \text{s.t.} \quad \forall i,\ y_i(\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1 \quad (2)$$

OUTPUT: $\hat{\vec{w}} = \frac{\vec{w}_0}{\|\vec{w}_0\|}$, $\hat{b} = \frac{b_0}{\|\vec{w}_0\|}$

The objective of that algorithm is a convex quadratic function, constraints are linear inequalities: can be solved with quadratic programming solvers.

Lemma The output of Hard-SVM is a solution of equation (1).

Proof. Let $(\vec{w}^*, b^*)$ be a solution of (1) and define the margin achieved by $(\vec{w}^*, b^*)$ to be $\gamma^* = \min_{i \in [m]} y_i(\langle \vec{w}^*, \vec{x}_i \rangle + b)$.

Therefore, for all i we have:

$$y_i(\langle \vec{w}^*, \vec{x}_i \rangle + b^*) \geq \gamma^*$$

or equivalently

$$y_i\left(\left\langle \frac{\vec{w}^*}{\gamma^*}, \vec{x}_i \right\rangle + \frac{b^*}{\gamma^*}\right) \geq 1$$

Hence, the pair $\left(\frac{\vec{w}^*}{\gamma^*}, \frac{b^*}{\gamma^*}\right)$ satisfies the conditions of the quadratic optimization problem given in equation (2). Therefore, $\|\vec{w}_0\| \leq \left\|\frac{\vec{w}^*}{\gamma^*}\right\| = \frac{1}{\gamma^*}$.

It follows that for all i,

$$y_i(\langle \hat{\vec{w}}, \vec{x}_i \rangle + \hat{b}) = \frac{1}{\|\vec{w}_0\|} y_i(\langle \vec{w}_0, \vec{x}_i \rangle + b_0) \geq \frac{1}{\|\vec{w}_0\|} \geq \gamma^*$$

Since $\|\hat{\vec{w}}\| = 1$ we obtain that $(\hat{\vec{w}}, \hat{b})$ is an optimal solution of (1).

HOMOGENEOUS REPRESENTATION → pass through the origin and are thus defined by $\text{sign}(\langle \vec{w}, \vec{x} \rangle)$, where the bias term b is set to zero

Formulation with homogeneous halfspaces:

- Assume first component of $x \in \mathcal{X}$ is 1, then

$$\vec{w}_0 = \underset{\vec{w}}{\operatorname{argmin}} \|\vec{w}\|^2 \quad \text{s.t.} \quad \forall i,\ y_i\langle \vec{w}, \vec{x}_i \rangle \geq 1 \quad (3) \quad \text{(HARD-SVM)}$$

- Notice that is similar but not exactly the same as the non-homogeneous one (the bias now also goes inside the regularization)

→ in practice, no big difference.

↑ case in which we reduce non homogeneous halfspaces by increasing the dimension to d+1 (homogeneous one)

SUPPORT VECTORS

The SUPPORT VECTORS are the vectors at minimum distance from $\vec{w}_0$. They are the only training vectors that matter for defining $\vec{w}_0$!

Theorem

Let $\vec{w}_0$ be as defined in **(3)** and let $I = \{i : |\langle \vec{w}_0, \vec{x}_i \rangle| = 1\}$.

Then, there exist coefficients $\alpha_1, \dots, \alpha_m$ such that

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

The examples $\{\vec{x}_i : i \in I\}$ are called "support vectors".

Remark. Solving HARD-SVM is equivalent to find α_i for the support vectors ($\alpha_i \neq 0$ only for support vectors).

DUALITY

Consider the function

$$g(\vec{w}) = \max_{\vec{\alpha} \in \mathbb{R}^m : \vec{\alpha} \geq 0} \sum_{i=1}^{m} \alpha_i (1 - y_i \langle \vec{w}, \vec{x}_i \rangle) = \begin{cases} 0 & \text{if } \forall i,\ y_i \langle \vec{w}, \vec{x}_i \rangle \geq 1 \\ \infty & \text{otherwise} \end{cases}$$

We can therefore rewrite equation **(3)** as

$$\min_{\vec{w}} \left(\|\vec{w}\|^2 + g(\vec{w}) \right)$$

$$\Rightarrow \min_{\vec{w}} \max_{\vec{\alpha} \in \mathbb{R}^m : \vec{\alpha} \geq 0} \left(\frac{1}{2} \|\vec{w}\|^2 + \sum_{i=1}^{m} \alpha_i \left(1 - y_i \langle \vec{w}, \vec{x}_i \rangle\right) \right)$$

It is called the "dual" problem.

Key property: only requires the inner product between instances but not the direct access to instances. It will be very useful for the "Kernel trick".

SOFT-SVM

Key issue: hard-SVM needs the data to be linearly separable → almost never true in practical problems.

We need an approach that can work also with non linearly separable data

→ SOFT-SVM.

↓

Relax the constraints of HARD-SVM but take into account the violations of the separation into the objective function.

A natural relaxation is to allow the constraint to be violated for some of the examples in the training set.

- Relax the constraint:
  - introduce slack variables: $\vec{\xi} = (\xi_1, \dots, \xi_m)$, $\xi_i \geq 0$
  - for each $i = 1, \dots, m$: $y_i (\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1 - \xi_i$
  - ξ_i: how much the constraint is violated.
- Soft-SVM jointly minimizes:
  - the norm of $\vec{w}$ (→ maximize margin)
  - the average of ξ_i (→ minimize constraint violation from misclassified points)

The trade-off between the two objectives is controlled by a parameter $\lambda > 0$. This leads to the SOFT-SVM optimization problem:

Input: $(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)$, parameter $\lambda > 0$

Solve:

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

$$\text{s.t. } \forall i, \; y_i (\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

Output: $\vec{w}$, b

REFORMULATE WITH HINGE LOSS

HINGE-LOSS:

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

The problem can be reformulated with the Hinge loss:

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

It is often more convenient to consider SOFT-SVM for learning a homogeneous halfspace, where the bias term b is set to be zero, which yields the following optimization problem:

$$\min_{\vec{w}} \left(\lambda \|\vec{w}\|^2 + L_S^{hinge}(\vec{w}) \right)$$

where

$$L_S^{hinge}(\vec{w}) = \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y\langle \vec{w}, \vec{x}_i\rangle\}$$

Approaches to solve the problem:

→ use standard solvers for optimization problems

→ use stochastic gradient descent (SGD)

SGD FOR SOFT SVM

HINGE LOSS:

$$f^{hinge}(\vec{w}) = \max\{0, 1 - y\langle \vec{w}, \vec{x}\rangle\}$$

Subgradient of f^{hinge} at $\vec{w}$:

$$\vec{v}^{hinge} = \begin{cases} 0 & \text{if } 1 - y\langle \vec{w}, \vec{x}\rangle \leq 0 \\ -y\vec{x} & \text{if } 1 - y\langle \vec{w}, \vec{x}\rangle > 0 \end{cases}$$

Update rule (for the complete soft-SVM optimization):

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \vec{v}^{(t)} \quad \text{or} \quad \vec{w}^{(t+1)} = -\frac{1}{\lambda t}\sum_{j=1}^{t} \vec{v}_j$$

↑ standard SGD

↑ variant of SGD for λ-strongly convex functions

We want to solve:

$$\min_{\vec{w}} \left(\frac{\lambda}{2}\|\vec{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y\langle \vec{w}, \vec{x}_i\rangle\} \right)$$

Remark. It's standard to add a 1/2 in the regularization term to simplify some computations.

Algorithm: SGD for solving SOFT-SVM (algorithm 1) → see Kernel section

parameter: T

initialize: $\vec{\theta}^{(1)} = 0$

for $t = 1, \dots, T$:

$\vec{\theta}^{(t)} = -\sum_{j=1}^{t-1} \vec{v}_j$

let $\vec{w}^{(t)} = \frac{1}{\lambda t}\vec{\theta}^{(t)}$

choose i uniformly at random from $\{1, \dots, m\}$;

if $y_i \langle \vec{w}^{(t)}, \vec{x}_i\rangle < 1$ then $\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} + y_i \vec{x}_i$

else $\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)}$

output: $\bar{\vec{w}} = \frac{1}{T}\sum_{t=1}^{T} \vec{w}^{(t)}$

hinge loss: gradient is 0 if correctly classified and $-y_i x_i$ if error

KERNEL METHODS (cap. 16)

In the previous chapter we tackle the sample complexity issue using the concept of margin. In this chapter we tackle the computational complexity challenge using the method of Kernels.

SVM is a powerful algorithm, but still limited to linear models, and linear models cannot always be used (directly!)

→ Idea: • apply a nonlinear transformation to each point in training;
• learn a linear predictor in the transformed space;
• make a prediction for a new instance.

We start the chapter by describing the idea of embedding the data into a high dimensional feature space. We then introduce the idea of Kernels. A Kernel is a type of a similarity measure between instances. The special property of Kernel similarities is that they can be viewed as inner products in some Hilbert space to which the instance space is virtually embedded.

EMBEDDINGS INTO FEATURE SPACES

Define a non-linear mapping ψ from the input space to a new (larger) space:

1. Given a domain set $\mathcal{X}$ and a learning task, find a mapping to a new <u>feature space</u> F:
   $\psi: \mathcal{X} \to F$
   F is usually $\mathbb{R}^n$ for some n, but can be an arbitrarly Hilbert space (even of infinite size).
2. Given a sequence of labeled examples $S = ((\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m))$ map them to $\hat{S} = ((\psi(\vec{x}_1), y_1), \dots, (\psi(\vec{x}_m), y_m))$
3. Train a linear predictor h over $\hat{S}$.
4. Predict the label of $\vec{x}$ as $h(\psi(\vec{x}))$.

The success of this learning paradigm depends on choosing a good ψ for a given learning task: that is, a ψ that will make the image of the data

distribution linearly separable in the feature space, thus making the resulting algorithm a good learner for a given task.

THE KERNEL TRICK

. The learning over the new highly dimensional space makes halfspaces more expressive.

. On the other side, the computational complexity can become huge.

The solution: Kernel-based learning.

- Kernel: inner product in the feature space.
- Kernel function $K(\vec{x}, \vec{x}') = \langle \psi(\vec{x}), \psi(\vec{x}') \rangle$
- $K(\vec{x}, \vec{x}')$ represents similarity of the samples in a space where the similarities are realized as inner products.
- Key result: machine learning algorithms for halfspaces can be carried out just on the basis of the values of the Kernel function without explicitly representing the points in the feature space.
- Sometimes we can compute $K(\vec{x}, \vec{x}')$ without computing $\psi(\vec{x})$ and $\psi(\vec{x}')$.

Consider $x \in \mathbb{R}^d$:

$$\psi(\vec{x}) = (1, x_1, x_2, \dots, x_d, x_1 x_1, x_1 x_2, x_1 x_3, \dots, x_d x_d)^T$$

example with 2nd degree polynomial

The dimension of $\psi(\vec{x})$ is $1 + d + d^2$

$$\langle \psi(\vec{x}), \psi(\vec{x}') \rangle = 1 + \sum_{i=1}^{d} x_i x_i' + \sum_{i=1}^{d} \sum_{j=1}^{d} x_i x_j x_i' x_j' \qquad O(d^2)$$

Note that

$$\sum_{i=1}^{d} \sum_{j=1}^{d} x_i x_j x_i' x_j' = \left(\sum_{i=1}^{d} x_i x_i' \right) \left(\sum_{j=1}^{d} x_j x_j' \right) = (\langle \vec{x}, \vec{x}' \rangle)^2$$

therefore

$$K_\psi(\vec{x}, \vec{x}') = \langle \psi(\vec{x}), \psi(\vec{x}') \rangle = 1 + \langle \vec{x}, \vec{x}' \rangle + (\langle \vec{x}, \vec{x}' \rangle)^2 \qquad O(d)$$

Remark. Computing $\psi(\vec{x})$ requires $O(d^2)$ time, computing $K_\psi(\vec{x}, \vec{x}')$ From the last formula requires $O(d)$ time.

When $K_\psi(\vec{x}, \vec{x}')$ is efficiently computable, we don't need to explicitly compute $\psi(\vec{x})$ → Kernel trick

KERNEL TRICK APPLIED TO SVM

Note that all versions of the SVM optimization problem we have derived in the previous chapter are instances of the following general problem:

$$\min_{\vec{w}} \left(f\left(\langle \vec{w}, \psi(\vec{x}_1) \rangle, \dots, \langle \vec{w}, \psi(\vec{x}_m) \rangle \right) + R\left(\|\vec{w}\| \right) \right) \quad (1)$$

where $F: \mathbb{R}^m \to \mathbb{R}$ arbitrary function

$R: \mathbb{R}_+ \to \mathbb{R}$ monotonically non decreasing function

Theorem (Representer theorem)

Assume that ψ is a mapping from $\mathcal{X}$ to a Hilbert space. Then, there exists a vector $\vec{\alpha} \in \mathbb{R}^m$ such that $\vec{w} = \sum_{i=1}^{m} \alpha_i \psi(\vec{x}_i)$ is an optimal solution of equation (1).

→ Consequence: we can optimize the problem with respect to the coefficients α_i getting a problem that depends only on $K(\vec{x}, \vec{x}') = \langle \psi(\vec{x}), \psi(\vec{x}') \rangle$ without explicitly computing $\psi(\vec{x})$ or $\psi(\vec{x}')$.

Note that:

$$\langle \vec{w}, \psi(\vec{x}_i) \rangle = \left\langle \sum_j \alpha_j \psi(\vec{x}_j), \psi(\vec{x}_i) \right\rangle = \sum_{j=1}^{m} \alpha_j \langle \psi(\vec{x}_j), \psi(\vec{x}_i) \rangle$$

similarly,

$$\|\vec{w}\| = \left\langle \sum_j \alpha_j \psi(\vec{x}_j), \sum_i \alpha_j \psi(\vec{x}_j) \right\rangle = \sum_{i,j=1}^{m} \alpha_i \alpha_j \langle \psi(\vec{x}_i), \psi(\vec{x}_j) \rangle$$

Let $K(\vec{x}, \vec{x}') = \langle \psi(\vec{x}), \psi(\vec{x}') \rangle$ be a function that implements the kernel function with respect to the embedding ψ. Instead of solving equation (1) we can solve the equivalent problem

$$\min_{\vec{\alpha} \in \mathbb{R}^m} f\left(\sum_{j=1}^{m} \alpha_j K(\vec{x}_j, \vec{x}_1), \dots, \sum_{j=1}^{m} \alpha_j K(\vec{x}_j, \vec{x}_m) \right) + R\left(\sqrt{\sum_{i,j=1}^{m} \alpha_i \alpha_j K(\vec{x}_j, \vec{x}_i)} \right)$$

To solve the last equation, we solely need to know the value of the $m \times m$ matrix G such that $G_{i,j} = K(\vec{x}_i, \vec{x}_j)$, which is often called the GRAM MATRIX G.

In particular, specifying the preceding to the Soft-SVM problem we can rewrite the problem as

$$\min_{\vec{\alpha} \in \mathbb{R}^m} \left(\lambda \vec{\alpha}^T G \vec{\alpha} + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i (G\vec{\alpha})_i\} \right)$$

Once we [illegible] coefficients $\vec{\alpha}$ we can calculate the prediction on a new instance by

$$\langle \vec{w}, \psi(\vec{x}) \rangle = \sum_{j=1}^{m} \alpha_j \langle \psi(\vec{x}_j), \psi(\vec{x}) \rangle = \sum_{j=1}^{m} \alpha_j K(\vec{x}_j, \vec{x})$$

The advantage of working with Kernels rather than directly optimizing $\vec{w}$ in the feature space is that in some situations the dimension of the feature space is extremely large while implementing the Kernel function is very simple.

→ POLYNOMIAL KERNELS

$$K(\vec{x}, \vec{x}') = (1 + \langle \vec{x}, \vec{x}' \rangle)^K$$

K is the degree of the polynomial

- it is a Kernel function, i.e., $K(\vec{x}, \vec{x}') = \langle \psi(\vec{x}), \psi(\vec{x}') \rangle$
- $\psi: \mathbb{R}^m \to \mathbb{R}^{(m+1)^K}$ contains all the monomials up to degree K.
- Halfspace over ψ corresponds to a polynomial predictor of order K in the original space.
- Complexity of computation is $O(m)$ while the dimension of feature space is $O(m^K)$.

→ GAUSSIAN KERNELS (radial basis functions, RBF)

$$K(\vec{x}, \vec{x}') = \exp\left\{ -\|\vec{x} - \vec{x}'\|^2 / 2\sigma \right\}$$

- it is a Kernel function, i.e., $K(\vec{x}, \vec{x}') = \langle \psi(\vec{x}), \psi(\vec{x}') \rangle$
- The feature space is of infinite dimension, but computing the Kernel is simple and fast!
- Product is close to ø if instances are far and close to 1 if they are close.
- Parameter σ controls what we mean by "close".
- We can learn any polynomial predictor in the original space by using a Gaussian Kernel.
- VC-dimension is infinite (sample complexity depends on the margin in the feature space).

SOFT SVM WITH KERNELS

We want to solve :

$$\min_{\vec{w}} \left(\lambda \|\vec{w}\|^2 + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i \langle \vec{w}, \psi(\vec{x}_i) \rangle\} \right) \quad (1)$$

Let :

- K be the Kernel Function : $K(\vec{x}, \vec{x}') = \langle \psi(\vec{x}), \psi(\vec{x}') \rangle$
- We shall maintain two vectors in $\mathbb{R}^m$, corresponding to two vectors $\vec{\theta}^{(t)}$ and $\vec{w}^{(t)}$ defined in the SGD procedure (algorithm 1). That is $\vec{\beta}^{(t)}$ will be a vector such that :

$$\vec{\theta}^{(t)} = \sum_{j=1}^{m} \beta_j^{(t)} \psi(\vec{x}_j) \quad ,$$

and $\vec{\alpha}^{(t)}$ be such that

$$\vec{w}^{(t)} = \sum_{j=1}^{m} \alpha_j^{(t)} \psi(\vec{x}_j)$$

The vectors $\vec{\beta}$ and $\vec{\alpha}$ are updated according to the following procedure.

Algorithm : SGD for solving SOFT-SVM with Kernels

Goal : solve equation (1)

parameter : T

initialize : $\vec{\beta}^{(1)} = 0$

for $t = 1, \dots, T$

let $\vec{\alpha}^{(t)} = \frac{1}{\lambda t} \vec{\beta}^{(t)}$

choose i uniformly at random from $[m]$

For all $j \neq i$ set $\beta_j^{(t+1)} = \beta_j^{(t)}$

If $\left(y_i \sum_{j=1}^{m} \alpha_j^{(t)} K(\vec{x}_j, \vec{x}_i) < 1 \right)$

set $\beta_i^{(t+1)} = \beta_i^{(t)} + y_i$

else

set $\beta_i^{(t+1)} = \beta_i^{(t)}$

output : $\bar{\vec{w}} = \sum_{j=1}^{m} \bar{\alpha}_j \psi(\vec{x}_j)$ where $\bar{\vec{\alpha}} = \frac{1}{T} \sum_{i=1}^{T} \vec{\alpha}^{(t)}$