Caltech's

Machine Learning

CS156

The biggest pitfall in theory is that the assumptions that are taken are divorced from the reality of how we use machine learning in practice

Machine Learning

Theory

- VC — Most Important stuff (Bounds that works well in practice)
- Bias ~ Variance theory (Learning curves...)
- Computational Complexity $O(n^2)$
  - ↳ Too away from practice!!
- Bayesian

Techniques

Models

- linear
- neural networks
- SVM
- Nearest neighbours. (Nice benchmark)
- RBF
- Gaussian Processes (Random processes) [Random Function]
- SVD
- Graphical Models (Learn joint prob dist.)

Methods

- regularization
- validation
- aggregation (putting together diff. solutions)
- Input processing PCA,...

Paradigms

- supervised
  - ↳ (Do we have enough data?)
- unsupervised
  - ↳ (How do we provide feedback?)
- reinforcement
  - ↳ (Take an action, sth tells you if you did sth good or not) Example: Chess Oracle (Blunder!)
- active
  - ↳ Instead of getting the whole dataset we query data
- online
  - ↳ Insted of getting the dataset, we get a stream of the dataset.

∅ A diminute improvement in a machine learning problem can lead to massive problem.

example:
movie recommendation

recommend(viewer, movie) → rating

LEARN

reverse engeneering knowing the movie & the rating.

∅ A problem can be tought as a machine learning problem IFF
- → We've a pattern *
- → We don't know how to pin it down mathematically. (If yes, might be better to use that)
- → We've data (A MUST)

* How do we know? We don't know! But we can try to apply methods and we can determine if we're learning or not.

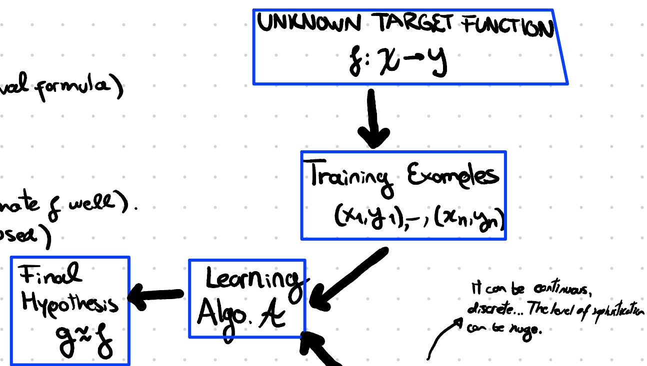
In fact we will use machine learning to see if there's a pattern

The Learning Problem.

Components of Learning

//A bank has no magical orb to know if a person is credit worthy or not.

- Input: x (customer data)
- Output: y (good/bad customer)
- Target function: $f: \mathcal{X} \to \mathcal{Y}$ (ideal credit approval formula)
- Data: $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- Hypothesis: $g: \mathcal{X} \to \mathcal{Y}$ (we hope approximate f well). (formula to be used)



(set of candidate formulas)

Ex: $\{x \to ax+b \,/\, (a,b) \in \mathbb{R} \times \mathbb{R}\}$

There is no really loss of generality. $\mathcal{H}$ can be the set of all possible functions.

Solution components

Two solution components of the learning problem: 1. The Hypothesis set $\mathcal{H} = \{h\}$, $g \in \mathcal{H}$
2. The learning algorithm.

Together, they're THE LEARNING MODEL.

Example: The Perceptron

Approve credit if $\sum_{i=1}^{d} w_i x_i >$ threshold

$h \in \mathcal{H}$ can be written as:

$$h(x) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right)$$

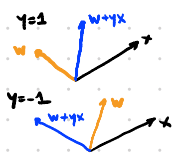
Things to learn.

rename: w_0

Add artificial coordinate $x_0 = 1$

$(w_0, \dots, w_d)$

$$h(x) = \text{sign}(w^T x)$$



Perceptron Learning Alg. → Pick a misclassified point: $\text{sign}(w^T x_n) \neq y_n$ → Update weight vector: $w \leftarrow w + y_n x_n$ → Run until no more are missclassified. (It will end if data is linearseable.

How do we know if sth is linearly sepperable?
↳ Assumme is not. We can transform data to be linearly sepperable. (At what cost?)

Statistics → We make assumptions, we have a math model.
Machine learning → Least assumptions (general as possible). Don't have a math model as in statistics.

The rate of convergence of the perceptron learning algorithm is in fact terrible. There are patological patterns where it doesn't converge.

Optimization: Tool at disposal for machine learning.

Basic Premise of Learning

"use a set of observations to undercover an underlying process"

- How much is enough data?
  - ↳ In practice is not sth you control. We'll see in theory more about that.
- The bottle neck of machine learning is its capacity of generalizing.

- **Supervised Learning** → The output data is explicitely given.
  we've (input, correct output)
  Ex: $\{(x_1,y_1), \ldots, (x_n,y_n)\}$
- **Unsupervised Learning**
  we've (input, ?) → Classify data without knowing the name of the classes.
  Example: Clustering.
  // A way of getting a high level representation (high patterns) of the input
- **Reinforcement Learning**
  we've (input, partial output, grade for this output)
  Reward for the output.
  Ex: game simulators.

Is Learning Feasible?

- Probabilistic approach:
  We pick red ballots from a blue red ballot.

  $P(\text{pick red ballot}) = \mu$

  If we do N indep exp and get a prop of ν red ballots, $\mu \approx \nu$? No and yes!
  - They can be totally diff.
  - ↳ In a big N, ν can be close to μ. (Most probably).

In fact,

$$\mathbb{P}[\underbrace{|\nu-\mu|>\varepsilon}_{\text{bad event.}}] \le 2e^{-2\varepsilon^2 N} \quad \forall N, \varepsilon > 0.$$

Hoeffding's inequality. → A law of large numbers.

So $\mu = \nu$ is P.A.C.
(Probable approximately correct.)

- The probability is bounded regardless of μ.
- $\nu \approx \mu \rightarrow \mu \approx \nu$

connection to learning

Learning: Unknown function. $f: \mathcal{X} \to y$
Each ballot is a point $x \in \mathcal{X}$,
red: right hypothesis $h(x) = f(x)$
blue: wrong hypothesis $\underbrace{h(x) \neq f(x)}_{\text{fixed } h!}$

Prob dist P on $\mathcal{X}$.
↳ data is generated from P.

Note: This example is verification, not learning.

Solution: Having multiple bins

ν is "in sample": E_{in}
μ is out-of sample: E_{out}

$E_{in}(h)$: Error of approximating my target function using h.
$E_{out}(h)$

Using Hoeffding inequality

$$\mathbb{P}[|\underset{\substack{\uparrow \\ \text{In sample performance.}}}{E_{in}(h)} - \underset{\substack{\uparrow \\ \text{out of sample performance}}}{E_{out}(h)}| > \varepsilon] \le 2e^{-2\varepsilon^2 N}$$

BUT, Hoeffding's doesn't apply to multiple bins!

Fortunately we can take the worst case. And with that, we get just a M factor which becomes meaningless.

Linear Model.

- Input rep.
- Linear Classif.
- Linear Reg.
- Non linear transf.

↱ MNIST Dataset

Input rep: $x = (x_0, x_1, \dots, x_{256})$ $\boxed{1}^{128}_{128}$

→ A linear model has 256 dim!

Sol: Features (Extract useful info)

Ex: Intensity and symmetry: $x = (x_0, x_1, x_2)$
(1s are less intense than 8)

PLA

when data is not sepparable the perception doesn't converge.

Pocket

Changes boundary only if the error is reduced.

Linear reg → real-valued output.

Example: calculate credit line based on client's features.

Regression output → $h(x) = w^T x$

• We use MSE: $(h(x) - f(x))^2 \rightarrow E_{in} = \frac{1}{N} \| \overbrace{Xw}^{\text{prediction}} - \overbrace{y}^{\text{expected outputs}} \|^2$

$$X = \begin{bmatrix} -x_1^T- \\ \vdots \\ -x_N^T- \end{bmatrix} \quad \nearrow M^{N \times m}_{\text{features}}$$

To minimize the error $E_{in}(w) = \frac{1}{N} \|Xw - y\|^2$

$\nabla E_{in}(w) = \frac{2}{N} X^T (Xw - y) = 0 \Leftrightarrow X^T X w = X^T y$

$w = X^+ y \,/\, X^+ = \underbrace{(X^T X)^{-1}}_{M \text{ d+1} \times \text{d+1}} \underbrace{X^T}_{M\ \text{d+1} \times N}$ / pseudoinverse.

The linear regression alg is just computing $w = X^+ y$

Linear reg for classification:

Binary valued functions are also real valued. We can do ↑

× However, the key problem is that it tries to fit all points into a line.

So we're just minimizing the distance. So It just works as a good starting value for the weights

$\text{Sign}(w^T x)$

Non-Linear Transformations

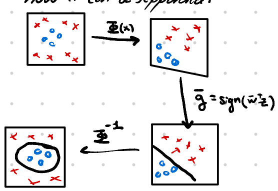
$[\![c]\!] = \begin{cases} 0, \text{if } c \text{ is false} \\ 1, \text{else} \end{cases}$

The model weight parameters is linear IN $\mathcal{W}$ Same problem!

We just apply non-linear transf to the data.

$(x_1, x_2) \xrightarrow{\Phi} (x_1^2, x_2^2)$

• Data that wasn't linearly sepparable now It can be sepparated!



$g(x) = \tilde{g}(\Phi(x))$
$= \text{Sign}(\tilde{w}^T \Phi(x))$

$x = (x_0, x_1, \dots, x_d) \xrightarrow{\Phi} z = (z_0, z_1, \dots, z_{\tilde{d}})$

// Powerful, but generalization can be terribly poor

$x_1, \dots, x_n \xrightarrow{\Phi} z_1, \dots, z_N$

$y_1, \dots, y_N \xrightarrow{\Phi} y_1, \dots, y_N$

No the weights are only in $\mathcal{Z}$

$\tilde{w} = (w_0, w_1, \dots, w_{\tilde{d}})$

Error Measures

$E(h, f)$ // error function

Almost pointwise def: $e(h(x), f(x))$ $\overset{\text{examples}}{\rightleftarrows}$ $= (h(x) - f(x))$ Squared error
$= [\![h(x) \neq f(x)]\!]$ Binary error

Insample error: $E_{in}(h) = \frac{1}{N} \sum e(h(x_n), f(x_n))$

Outsample error $E_{out}(h) = \mathbb{E}_x [e(h(x), f(x))]$

How to choose error measure → Depends on the problem. (Application Domain question)

If we don't know what to do, use PLAUSIBLE measures or FRIENDLY MEASURES
squared error (Gaussian Noise) — convex optimization

Noisy Targets

// Target function is not a function $f(x_1) = 0$, $f(x_2) = 2$, where $x_1 = x_2$.

Sol: use "target distribution" $P(y|x)$

(x, y) is now def. by a joint distribution.

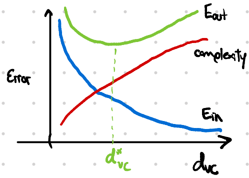
Noisy target: $\underset{\text{deterministic}}{f(x)} + \underset{\text{noise}}{\varepsilon} \Rightarrow f(x) = \mathbb{E}(y|x)$

$P(x) \neq P(y|x)$

$P(x)$: quantifies relative importance of X
$P(y|x)$: ↳ what are we trying to learn

Merging $P(x)P(y|x)$ as $P(x, y)$ mixes two concepts

We need a metric to measure the sophistication of the model $\rightarrow d_{vc}$



model complexity ++ $\rightarrow$ Ein ++ but
Eout - Ein ++

Ein is kinda a proxy to Eout.

$E_{out}(g) \approx 0$ is achieved through:
$E_{out}(g) \approx E_{in}(g)$
$E_{in}(g) \approx 0$
Learning is reduced to two questions
- Can we assure that $E_{out}(g)$ is close to $E_{in}(g)$?
- Can we make $E_{in}(g)$ small enough?

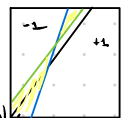
Obs: Sometimes its impossible to have $E_{out}(g) \approx 0$. Ex: stock market. Having an error of 45% would make you rich already.

Training vs Testing.

Testing:
$$\mathbb{P}[|E_{in}-E_{out}|>\varepsilon] \leq 2\exp(-2\varepsilon^2 N).$$
Training
$$\mathbb{P}[|E_{in}-E_{out}|>\varepsilon] \leq 2M\exp(-2\varepsilon^2 N)$$
↳ Objective: Find a better bound than M

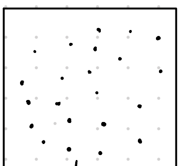
M comes from
Bad events: $P[B_1 \text{ or} \dots B_m] \leq \underbrace{P[B_1] \dots P[B_m]}_{\text{no overlaps: } M}$

In practice, Bad Events overlaps!
↳ ΔE_{out}: change in ± 1 areas
ΔE_{in}: change of labels in data points



$|E_{in}(h_1) - E_{out}(h_1)| \approx |E_{in}(h_2) - E_{out}(h_2)|$

If there is data here, the Ein changes. If h's are almost the same, there is almost no change in ΔEin!



↳ From this constellation of points, how many patterns of red & blue can I get?
↳ Number of dichotomies

$h: \mathcal{X} \rightarrow \{\pm 1\}$ //hypothesis
$h: \{x_1, \dots, x_N\} \rightarrow \{\pm 1\}$ //dichotomy
The number of hypothesis $|\mathcal{H}|$ can be Infinite, but the number of dichotomies is at most 2^N

The growth function

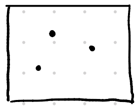
> I give you a budget N, choose where to place N points so the dichotomies are maxed.

The growth function counts the most dichotomies on any N points

$$m_{\mathcal{H}}(N) = \max_{x_1,\dots,x_N \in \mathcal{X}} \underbrace{|\mathcal{H}(x_1,\dots,x_N)|}_{\text{set of dichotomies}}$$

It satisfies:
$$m_{\mathcal{H}}(N) \leq 2^N$$

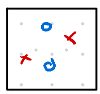
Apply $m_{\mathcal{H}}(N)$ to perceptrons



N=3

We have 2^3 ways to separate it with a line.
$m_{\mathcal{H}}(3) = 2^3$

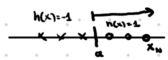
And for $m_{\mathcal{H}}(4)$?



- we can't make this dichotomy with a line!

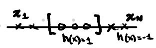
In fact, $m_{\mathcal{H}}(4) = 14$
↳ perceptrons

Illustration of the growth function:



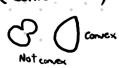
Example
(Positive Rays) $\mathcal{H}$: $h: \mathbb{R} \rightarrow \{\pm 1\}$
$h(x) = \text{sgn}(x-a)$
$m_{\mathcal{H}}(N) = N+1$

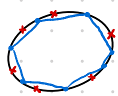
(Positive Intervals)



$m_{\mathcal{H}}(N) = \binom{N+1}{2} + 1$
// Ways of assigning N+1 points in two separate sets.

(Convex sets) $h: \mathbb{R}^2 \rightarrow \{\pm 1\}$
$h(x) = +1$ is convex.
$h(x) = 0$ not convex.





Sol: put the N points in a circle, we get 2^N dichotomies (max bound!)

$m_{\mathcal{H}}(N) = 2^N$

↳ When this happens: we say $\mathcal{H}$ shatters the points

$\mathbb{P}[|E_{in} - E_{out}| > \epsilon] \leq 2Me^{-2\epsilon^2 N}$

- Let's replace M with $m_{\mathcal{H}}(N)$.
- If $m_{\mathcal{H}}(N)$ has polynomial order. I've won.

Just proving that $m_H(N)$ is polynomial is enough to prove that learning is possible.

Breakpoint

→ The point where you fail to get all possible dichotomies (We can't shatter the data anymore).

If no data set of size K can be shattered by $\mathcal{H}$, then K is a break point of $\mathcal{H}$

lowest k / $m_{\mathcal{H}}(k) < 2^k$

Breakpoint examples
- positive rays: $K=2$
- positive intervals: $K=3$
- convex sets: $K=+\infty$

Le main point

$\nexists$ break point → $m_H(N) = 2^N$

$\exists$ break point → $m_H(N) \in P(N)$

↳ polynomials in N

We want to bound $m_{\mathcal{H}}(N)$

tactic: $m_{\mathcal{H}}(N) \leq \ldots \leq$ a polynomial.

Key quantity: $B(N,k)$: Max num of dichotomies on N points, with break point k

↳ BIN

Derived from this.

$$m_{\mathcal{H}}(N) \leq \underbrace{\sum^{k-1} \binom{N}{i}}_{\text{max power is } N^{k-1} \to \text{ constant! (A polynomial}}$$

Upper bound for anything with breakpoint K.

> $m_{\mathcal{H}}(N)$ is a polynomial

How to replace M in hoeffdings inequality?

$$\boxed{\mathbb{P}[|E_{in}(g) - E_{out}(g)| > \epsilon] \leq 4m_H(2N)e^{-\frac{1}{8}\epsilon^2 N}}$$

- This bound can be improved. But this is enough to prove that learning is possible.

C'est Vapnik-Chervonenkis Inequality.

The VC dimension

Def: The VC dimension of an hypothesis set $\mathcal{H}$ denoted by $d_{vc}(\mathcal{H})$ is the largest value of N for which $m_H(N) = 2^N$

"the most points $\mathcal{H}$ can shatter"

$N \leq d_{vc}(\mathcal{H}) \to \mathcal{H}$ can shatter N points.

$k > d_{vc}(\mathcal{H}) \to k$ is a breakpoint of $\mathcal{H}$

$$\to m_{\mathcal{H}}(N) \leq \underbrace{\sum^{d_{vc}} \binom{N}{i}}_{\text{poly of order } N^{d_{vc}}}$$

Ex: Positive rays: $d_{vc} = 1$

2D perceptrons: $d_{vc} = 3$

Convex sets: $d_{vc} = \infty$

VC dimension and Learning

$d_{vc}(\mathcal{H}) < \infty \longrightarrow g \in \mathcal{H}$ will generalize.

> This is independently from the Learning alg & Input distribution & target function.

> You will generalize with high probability wrt the input data.

Example: For d-dim perceptrons $\boxed{d_{vc}=d+1}$ ⟶ Proof: Let $N=d+1$, $X=\underbrace{\begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_{d+1}^T \end{bmatrix}}_{M_{d+1\times d+1}} = \overbrace{\begin{bmatrix} 1 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 1 & & \\ \vdots & & & \ddots & 0 \\ 1 & 0 & \cdots & 0 & 1 \end{bmatrix}}^{\text{invertible}}$. For any $y=\begin{bmatrix} y_1 \\ \vdots \\ y_{d+1} \end{bmatrix} = \begin{bmatrix} \pm 1 \\ \vdots \\ \pm 1 \end{bmatrix}$

Lets find $w / \text{sign}(Xw)=y$. We can do $Xw=y$. Because X inv, $\underline{w=X^{-1}y}$!

Thus, we can shatter these $d+1$ points ⟶ $d_{vc} \geq d+1$

For $d+2$, more points than dim → $x_j = \sum a_i x_i$ (Lin. dep), not all the $a_i = 0$.

The non-zero a_i get $y_i = \text{sign}(a_i)$ and x_j gets $y_j = -1$. No perceptron can implement this dichotomy!

$x_j = \sum a_i x_i \rightarrow w^T x_j = \sum a_i w^T x_i$. If $y_i = \text{sign}(w^T x_i) = \text{sign}(a_i)$, thus $a_i w^T x_i > 0$

So $w^T x_j = \sum a_i w^T x_i > 0 \rightarrow y_j = \text{sign}(w^T x_j) = +1$ ∎

d+1 is also the number of parameters in the perceptron.

Interpreting the VC dimension.

\# of params in model ~ degrees of freedom

d_{vc} ~ binary degrees of freedom (If you can separate data or not)

Obs: Parameters may not contribute degrees of freedom.

$x \rightarrow (S) \xrightarrow{\text{perceptrons}} (S) \rightarrow \overset{\infty}{\cdots} \rightarrow (S) \rightarrow y$

Here there are ∞ params, but 1 degree of freedom.

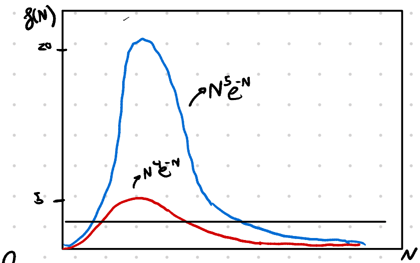
→ d_{vc} measures the EFFECTIVE number of parameters.

2_ How many data points we need

VC ineq: $\mathbb{P}[|E_{in}(g) - E_{out}(g)| > \epsilon] \leq \underbrace{4 m_{\mathcal{H}}(2N) e^{-\frac{1}{8}\epsilon^2 N}}_{\delta}$

If we want certain ϵ and δ, how does N depend on d_{vc}?

simplification of δ ↳ $f(N) = N^d e^{-N}$



$\mathbb{P}[|E_{out} - E_{in}| > \epsilon] \leq \overbrace{4 m_{\mathcal{H}}(2N) e^{-\frac{1}{8}\epsilon^2 N}}^{\delta}$

Get $\epsilon(\delta)$: $\epsilon = \sqrt{\frac{8}{N} \ln\left(\frac{4 m_{\mathcal{H}}(2N)}{\delta}\right)} = \Omega$

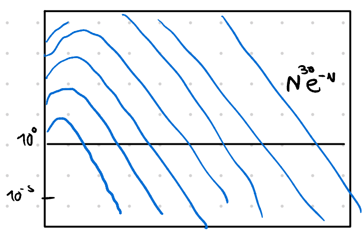
With prob $1-\delta$, $|E_{out} - E_{in}| \leq \Omega(N, \mathcal{H}, \delta)$

$\leftrightarrow E_{out} - E_{in} \leq \Omega(N, \mathcal{H}, \delta) \rightarrow E_{out} \leq E_{in} + \Omega$ (With prob $1-\delta$)

goes down with bigger $\mathcal{H}$

goes ↑ with bigger $\mathcal{H}$

Rescale:



> Bigger VC dimension ⟶ Need for most samples

$d_{vc} \propto$ number of samples needed to obtain a certain performance.

Rule of thumb: $N \geq 10 d_{vc}$ ⟶ $N \propto d_{vc}$

↳ Number of samples needed is proportional to d_{vc}

Bias Variance Tradeoff

Approximation - Generalization tradeoff.

Small E_{OUT}: good approx of f out of sample

More complex $\mathcal{H} \to$ Better chance of approximating f
Less complex $\mathcal{H} \to$ Better chance of generalizing out of sample

$f \in \mathcal{H}$ never happens...

The biasvariance is a new approach

VC analysis: $E_{out} \leq E_{in} + \Omega$

Biasvariance : Decompose E_{out} into → How well $\mathcal{H}$ approximates f / How well we can zoom in on a good $h \in \mathcal{H}$

↳ Applies to real-valued targets and uses square error

$$E_{out}(g^{(D)}) = \mathbb{E}_x\left[\left(g^{(D)}(x) - f(x)\right)^2\right]$$

I want to get rid of (D), my dataset.

$$\mathbb{E}_D\left[E_{out}(g^{(D)})\right] = \mathbb{E}_D\left[\mathbb{E}_x\left[\left(g^{(D)}(x) - f(x)\right)^2\right]\right]$$ → just integration! (Def $\mathbb{E}[\cdot]$)

$$= \mathbb{E}_x\left[\underbrace{\mathbb{E}_D\left[\left(g^{(D)}(x) - f(x)\right)^2\right]}\right]$$

Lets define $\bar{g}(x) := \mathbb{E}_D\left[g^{(D)}(x)\right] \to$

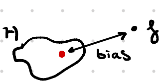
$$\mathbb{E}_D\left[\left(g^{(D)}(x) - \bar{g}(x) + \bar{g}(x) - f(x)\right)^2\right] = \mathbb{E}_D\left[\left(g^{(D)}(x) - \bar{g}(x)\right)^2 + \left(\bar{g}(x) - f(x)\right)^2 + 2\left(g^{(D)}(x) - \bar{g}(x)\right)\left(\bar{g}(x) - f(x)\right)\right]$$

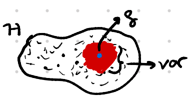
$$= \underbrace{\mathbb{E}_D\left[\left(g^{(D)}(x) - \bar{g}(x)\right)^2\right]}_{var(x)} + \underbrace{\left(\bar{g}(x) - f(x)\right)^2}_{bias(x)}$$

var(x) → Variance of my model

bias(x) → This is my best candidate "from" my hyp set. We differ from f because

$$\longrightarrow \mathbb{E}_D\left[E_{out}(g^{(D)})\right] = \mathbb{E}_x\left[bias(x) + var(x)\right]$$



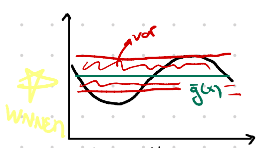


Example: $f: [-1, 1] \to \mathbb{R}$, $f(x) = \sin(\pi x)$

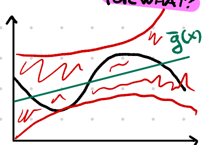
Our training dataset: $N = 2$ (Lol!)

Hyp. set: $H_0: h(x) = b$
$H_1: h(x) = ax + b$

who is better? → FOR WHAT?



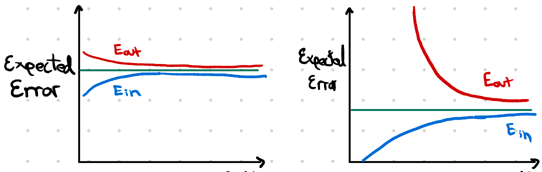
bias = 1/2
var = 1/4
+ = 3/4



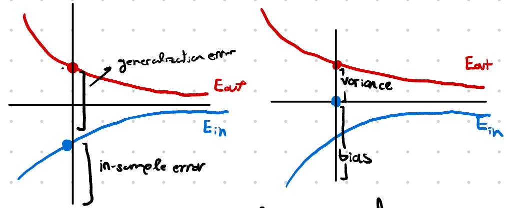
bias = 0.21
var = 1.69

Learning Curves

Dataset D of size N. How does E_{in} and E_{out} vary with N?



VC Analysis / Bias Varience



careful when you do the transformations!

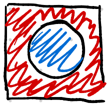
Looking at the data BEFORE choosing the model can be hazardous to your E_{OUT}

You just "snooped" at the data.
↳ The dataset with the d_{vc} guarantees is the one you had before.
DATA SNOOPING

Non-linear Transformations

$Z = \Phi(x)$

with this we can do:



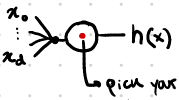
ex: $Z = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2) = (z_0, \dots, z_{\tilde{d}})$

Obs: Find hyp on $\mathcal{X}$.

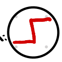
↳ $d_{vc} \leq \tilde{d} + 1$
↳ we're limited to Φ.

Logistic Regression

$s = w^T x$:



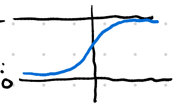
↳ pick your function

linear classification: 

linear regression: 

logistic regression: 

Logistic function θ:



$$\theta(s) = \frac{e^s}{1+e^s}$$

// soft threshold.
↳ models uncertainty

This is also called sigmoid

↳ We pick this because is great for optimization

signal: $w^T x$ (remains linear)

$h(x) = \theta(s)$ is interpreted as a probability.

Example: (x, y) . y binary, y is noisy

$$\mathbb{P}(y \mid x) = \begin{cases} f(x), & \text{for } y=+1 \\ 1-f(x), & \text{for } y=-1 \end{cases}$$

$\theta(-s) = 1 - \theta(s)$

$$P(y \mid x) = \theta(y w^T x)$$

My target is $f : \mathbb{R}^d \to \{0,1\}$ probability

Learn $g(x) = \theta(\underline{w^T x}) \approx f(x)$

How to train weights.

$$\prod_{n=1}^{N} P(y_n \mid x_n)$$

$$\prod_{n=1}^{n} \theta(y_n w^T x_n) \quad \text{(Likelihood)}$$

→ we want to maximize this wrt w.

How we build an error measure in this context?
→ Plausible error measure based on likelihood.

↳ We max the log likelihood instead (Equivalent). And we minimize with -1 instead:

$$\min \; \frac{-1}{N} \ln\left(\prod \theta(y_n w^T x_n)\right) = \frac{1}{N} \sum \ln\left(\frac{1}{\theta(y_n w^T x_n)}\right)$$

$$\hookrightarrow E_{in}(w) = \frac{1}{N} \sum^{N} \underbrace{\ln\left(1 + e^{-y_n w^T x_n}\right)}_{e(h(x_n), y_n)}$$

To minimize this, there is no close form solution

"cross-entropy" error

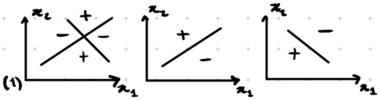
Iterative Method!
↳ Gradient descent.

Neural Networks

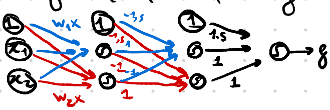
// features from features

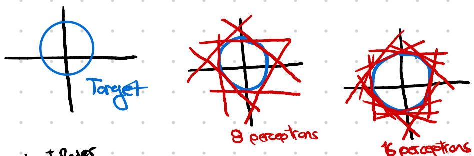
biological function ⇝ biological structure

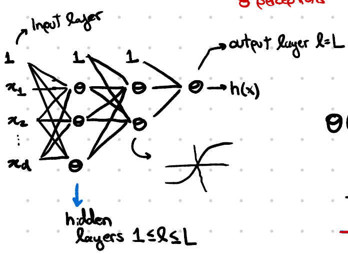
Key: combining perceptrons



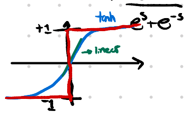
A multilayered perceptron implementing this (1)







$\theta(s) = \tanh(s) = \dfrac{e^s - e^{-s}}{e^s + e^{-s}}$



$$w_{ij}^{(l)} \begin{cases} l, & 1 \le l \le L \text{ layers} \\ i, & 0 \le i \le d^{(l-1)} \text{ inputs} \;/\text{We always have } x_0 \text{ constant!} \\ j, & 1 \le j \le d^{(l)} \text{ outputs} \end{cases}$$

$$x_j^{(l)} = \theta(\underbrace{s_j^{(l)}}_{\text{signal}}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}\right)$$

Apply x to $x_1^{(0)} \dots x_{d^{(0)}}^{(0)} \to\to x_1^{(L)} = h(x)$

> **Algo: Backpropagation.**
> 1- Init weights $w_{ij}^{(l)}$ RANDOMLLY.
> 2- for $t = 0,1,2 \dots$ do
>   Pick $n \in \{1 \dots N\}$
>   Forward: compute ALL $x_j^{(l)}$
>   Backward: compute ALL $\delta_j^{(l)}$
>   Update $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$
>   Iterate until criteria stop
> return $w_{ij}^{(l)}$

SGD

$E_{in}(w) = \frac{1}{N} \sum \ln(1 + e^{-y_n w^T x_n})$ // logistic regression

We pick a small subset (batchsize) of the dataset and do gradient descent.

// Theory: $E_n(\nabla g(x_n, y_n)) = E_{in}$!

Advantages of SGD:
1. less order of comp.
2. randomization (makes you escape of silly local minima)
3. simple

Rule of thumb: $\eta = 0.1$ is ok (learning rate)

SGD @ NN

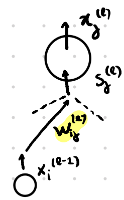
Error on sample: $e(h(x_n), y_n) = e(w)$

We need the gradient:

$$\nabla e(w) : \frac{\partial e(w)}{\partial w_{ij}^{(l)}}$$

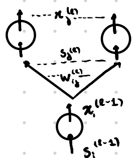
We can evaluate $\frac{\partial e(w)}{\partial w_{ij}^{(l)}}$ analytically or numerically.

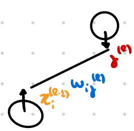
$$\to \frac{\partial e(w)}{\partial w_{ij}^{(l)}} = \frac{\partial e(w)}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}$$



$$x_i^{(l-1)} = \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}} =$$

What we need: $\delta_j^{(l)} = \frac{\partial e(w)}{\partial s_j^{(l)}}$





$$\delta_i^{(l-1)} = \frac{\partial e(w)}{\partial s_i^{(l-1)}} = \sum_{j=1}^{d^{(l)}} \frac{\partial e(w)}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial x_i^{(l-1)}} \times \frac{\partial x_i^{(l-1)}}{\partial s_i^{(l-1)}}$$

$$\delta_i^{(l-1)} = \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)})$$

$$\Longrightarrow \delta_i^{(l-1)} = (1 - (x_i^{(l-1)})^2) \sum_{j=1}^{d^{(l)}} w_{ij}^{(l)} \delta_j^{(l)} \quad \text{// For tanh}$$

δ for the final layer

$$\delta_1^{(L)} = \frac{\partial e(w)}{\partial s_1^{(L)}}$$

$$e(w) = e(\underbrace{h(x_n)}_{x_1^{(L)}}, y_n)$$

The value of the output!

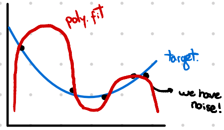
$x_1^{(L)} = \theta(s_1^{(L)})$

$\to \theta'(s) = 1 - \theta^2(s)$ for tanh.

Overfitting

> Overfitting is what separate ML amateurs from professionals.

Example:



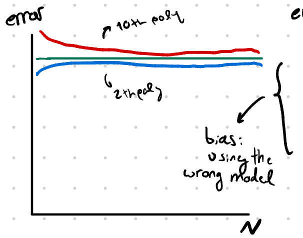
> Using a 4-order polynomial is an overfit.

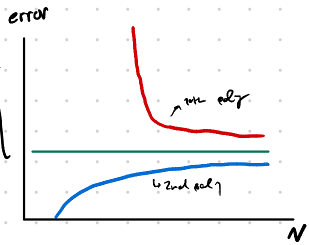
<u>Overfitting</u> $\neq$ Bad generalization

↳ You tried too hard

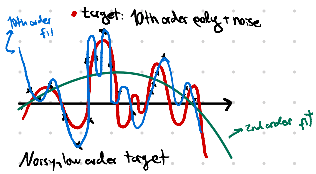
Def: Overfitting: fitting the data more than is warranted

→ Culprit: fitting the noise (harmful)





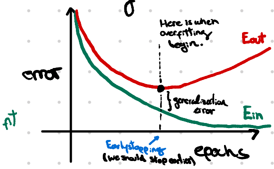
> With enough N, the $\mathcal{H}_{10}$ model is a better fit. But before, we had smaller error out of sample with $\mathcal{H}_2$.



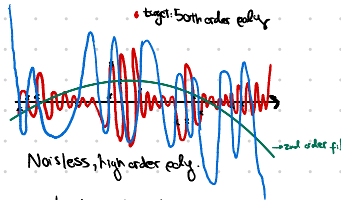
Noisy, low order target

	2nd order	10th order
E_{in}	0.05	0.034
E_{out}	0.127	9.00

Training a NN:



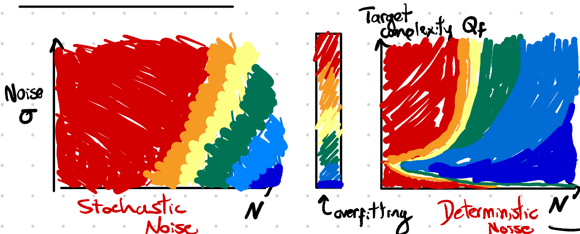
overfitting: $E_{in}\downarrow$ $E_{out}\uparrow$



Noisless, high order poly.

	2nd order	10th order
E_{in}	0.029	10^{-5}
E_{out}	0.120	7680

Why? If there's no noise!
↳ There is another type of noise...



Overfitting Heatmap

aka bias → Deterministic noise: the part of $\mathcal{H}$ cannot capture.

- It depends on $\mathcal{H}$ and is fixed for a given x.

number of data points $N\uparrow$: Overfitting $\downarrow$
stochastic noise $\uparrow$: Overfitting $\uparrow$
deterministic noise $\uparrow$: Overfitting $\uparrow$

Bias variance decomposition with noise:

$$\underbrace{\mathbb{E}_{\mathcal{D},x}\left[\left(g^{(\mathcal{D})}(x)-\bar{g}(x)\right)^2\right]}_{\text{var}} + \underbrace{\mathbb{E}_{x}\left[\left(\bar{g}(x)-f(x)\right)^2\right]}_{\text{bias}} + \underbrace{\mathbb{E}_{\epsilon,x}\left[\left(\epsilon(x)\right)^2\right]}_{\sigma^2}$$

Fitting the noise is like fitting to a pattern that doesn't exist.
The deterministic noise comes from the limitations of our hypothesis set $\mathcal{H}$.

Regularization

Instead of having wild lines, we want mild lines.

↓

We greatly reduce the variance at the cost of mildly increasing the bias.

Two approaches
- → Math: Ill-posed problems In function approximations. → Problem: The assumptions don't hold in practice sometimes !!
- → A tool to put a break on overfitting.

EXAMPLE

Def: Let L_k the kth Legendre polynomial of order k.
They are orthogonal to each other.
$L_1 = x,\ L_2 = \frac{1}{2}(3x^2-1),\ L_3 = (5x^3-3x)$

$$Z = \begin{bmatrix} 1 \\ \vdots \\ L_Q(x) \end{bmatrix} \qquad \mathcal{H}_Q = \left\{ \sum_{q=0}^{Q} w_q L_q(x) \right\}$$

> Legendre polynomials for an hypothesis set.

Given $(x_1,y_1),\dots,(x_n,y_n) \to (z_1,y_1),\dots,(z_n,y_n)$

$$\text{Minimize } E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N}(w^T z_n - y_n)^2 = \frac{1}{N}(Zw-y)^T(Zw-y)$$

$\to w_{lin} = (Z^TZ)^{-1}Z^Ty$ // GG.

However, If we constrain the weights:

Obs: $\mathcal{H}_2$ is a constrained version of $\mathcal{H}_{10}$ // Hard constraint: $w_q = 0\ \forall q > 2$.

a softer constraint: $\sum_{q=0}^{Q} w_q^2 \le C \leftrightarrow \min \frac{1}{N}(Zw-y)^T(Zw-y)$ st $w^Tw \le C$

- We call this sol w_{reg}
- This is easly solvable using KKT multipliers.

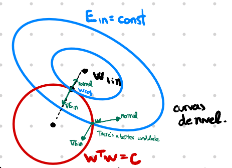
$\nabla E_{in}(w_{reg}) \propto -w_{reg}$

$= -2\frac{\lambda}{N} w_{reg}$

$E_{aug}(w)$

$\shortparallel$

$\nabla E_{in}(w_{reg}) + 2\frac{\lambda}{N} w_{reg} = 0 \to \text{minimize } E_{in}(w) + \frac{\lambda}{N} w^Tw$



Compromiso cuando w_{in} no está en O.

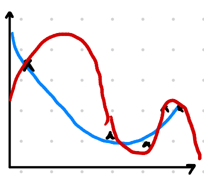
> Minimizing $E_{aug}(w)$ unconditionally is equiv to $\begin{cases} \min & E_{in}(w) \\ \text{st} & w^Tw \le C \end{cases}$

The final solution is $Z^T(Zw-y) + \lambda w = 0$

$\leftrightarrow w_{reg} = (Z^TZ + \lambda I)^{-1} Z^T y$ // with reg.

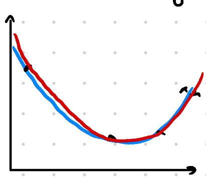
$w_{lin} = (Z^TZ)^{-1}Z^Ty$ // without reg.

One shot learning with regularization!

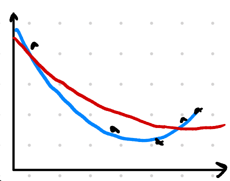


$\lambda = 0$

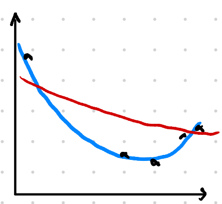
Overfit →



$\lambda = 1 \times 10^{-4}$ →



$\lambda = 0.01$ →



$\lambda = 1$

Underfit

This regularizer is called WEIGHT DECAY.

Gradient descent $w(t+1) = w(t) - \eta \nabla E_{in}(w(t)) - 2\eta\frac{\lambda}{N}w(t)$

$= w(t)(1 - 2\eta\frac{\lambda}{N}) - \eta\nabla E_{in}(w(t))$

Variations of weight decay:

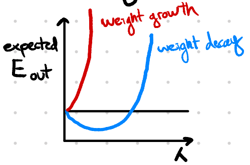
1_ Emphasize certain weights: $\sum_{q=0}^{Q} \gamma_q w_q^2$

↳ Examples: $\gamma_q = 2^q$ // low-order fit.
$\gamma_q = 2^{-q}$ // high order fit

> In Neural Networks: Different γs for each layer.

↳ Tikhonov regularizer: $w^T \Gamma^T \Gamma w$

2_ We play the inverse: Weight growth!



X Terrible Idea.
Best $\lambda = 0$.

Rule of thumb: stochastic noise is "high frequency"
deterministic noise is also non-smooth ⟹ constrain learning towards smoother hypothesis

In general smaller weights → smoother hypothesis

> We want to penalize the noise, not the signal.

General form of Augmented Error

Call reg $\Omega(h) = \Omega$, we minimize $E_{aug}(h) = E_{in}(h) + \frac{\lambda}{N}\Omega(h)$
$\Downarrow$
$E_{out}(h) \leq E_{in}(h) + \Omega(\mathcal{H})$

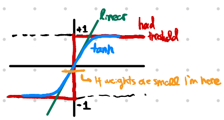
E_{aug} is better than E_{in} as a proxy for E_{out}.

Choosing A Regularizer

> We move to smoother postures because the noise is not smooth.

> If Ω is bad, we have λ to clean If we have it wrong.

Neural Network regularizers



Weight decay: If all weights are small we end up with a linear function camuflaged (f linear, g linear → $f \circ g$ linear). As weights increase we get a full logical function

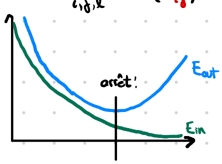
Weight elimination: fewer weights → smaller VC dimension

Soft weight elimination: $\Omega(w) = \sum_{i,j,l} \frac{\left(w_{ij}^{(l)}\right)^2}{\beta^2 + \left(w_{ij}^{(l)}\right)^2}$

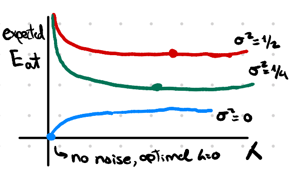
// Big weights are left alone, small weights are pushed toward zero.

Early stopping as a Regularizer

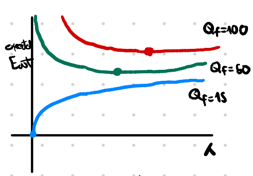
> Regularization through the optimizer!
> when to stop?



The optimal λ



STOCHASTIC NOISE



Deterministic Noise

> Deterministic noise behaves almost exactly as if it were stochastic noise.

Validation

Validation $\neq$ Regularization

$E_{out}(h) = E_{in}(h) + \text{overfit-penalty}$

validation estimates this — reg. estimates this

$\hookrightarrow E_{out}(h) = E_{in}(h) + \text{overfit-penalty}$

On out of sample point (x,y) the error is $e(h(x),y)$

$\rightarrow \mathbb{E}[e(h(x),y)] = E_{out}(h)$

$\text{var}[e(h(x),y)] = \sigma^2$

$\Longrightarrow$ On a validation set $(x_1,y_1),\dots,(x_K,y_K)$ the error is $E_{val}(h) = \frac{1}{K}\sum^{K} e(h(x_K),y_K)$

$\rightarrow \mathbb{E}[E_{val}(h)] = E_{out}(h)$

$\text{Var}[E_{val}(h)] = \text{var}\left[\frac{1}{K}\sum e(h(x_K),y_K)\right]$

$= \frac{1}{K^2}\sum \text{var}[e(h(x_K),y)] = \sigma^2/K$ ↗ Big K var↓!

$\Longrightarrow E_{val}(h) = E_{out}(h) \pm \underbrace{O(1/\sqrt{K})}_{\text{error}}$

Is K is too big you are giving a reliable estimate of the error for a very poor model.

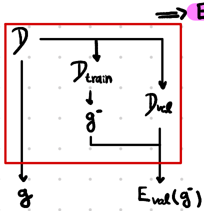
Problem: K is not given. Each point in validation is a point less for training.

K points validation // D_{val}

N-K points training // D_{train}

$D \rightarrow \overset{N-K}{D_{train}} \cup D_{val}$

$D \rightarrow g$, $D_{train} \Longrightarrow g^-$



$E_{val} = E_{val}(g^-)$

Large K → bad estimate.

Rule of thumb: $K = \frac{N}{5}$

> D_{val} is used to make learning choices

If an estimate of E_{out} affects learning:

The set is no longer a TEST set.

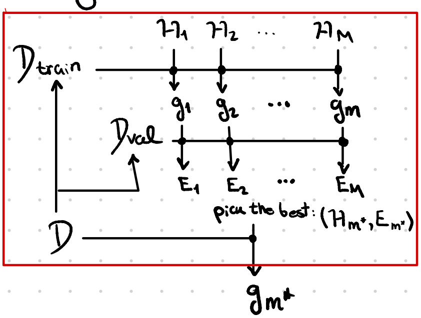
It becomes a VALIDATON set!

- Test set is unbiased; validation set has optimistic bias.
- Example: Two hyp h_1,h_2 / $E_{out}(h_1) = E_{out}(h_2) = 1/2$

  Pick $h \in \{h_1,h_2\}$ / $e = \min(e_1,e_2)$

  $\mathbb{E}(e) < 1/2$ (optimistic bias)

Using D_{val} many times



M models $\mathcal{H}_1,\dots,\mathcal{H}_M$

Use D_{train} to learn g_m^- from each model.

$E_m = E_{val}(g_m^-)$

we pick $m = m^*$ with the smallest E_m.

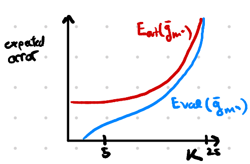
D_{val} is used for "training" on the finalists model

$\mathcal{H}_{val} = \{\bar{g}_1,\dots,\bar{g}_M\}$

$E_{out}(\bar{g}_{m^*}) \leq E_{val}(\bar{g}_{m^*}) + O\left(\sqrt{\frac{\ln M}{K}}\right)$

We selected the model $\mathcal{H}_{m^*}$ using D_{val}

$E_{val}(\bar{g}_{m^*})$ is a biased estimate of $E_{out}(\bar{g}_{m^*})$



regularization λ, early-stopping T

→ each give a degree of freedom: d_{VC}

Data contamination $E_{in}, E_{test}, E_{val}$

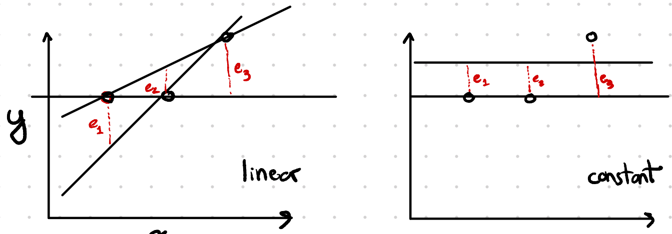
> Optimistic (deceptive) bias in estimating E_{out}

Training set: totally contaminated

Test set: totally clean

Validation set: "slightly contaminated"

Cross-Validation



$E_{cv} = \frac{1}{3}(e_1 + e_2 + e_3)$ $\qquad$ $E_{cv} = \frac{1}{3}(e_1 + e_2 + e_3)$

> constant model has less E_{cv}!

$E_{out}(g) \approx E_{out}(g^-) \approx E_{val}(g^-)$

(small K) (large K)

> We want small and big K.

N-1 points for training, 1 for validation

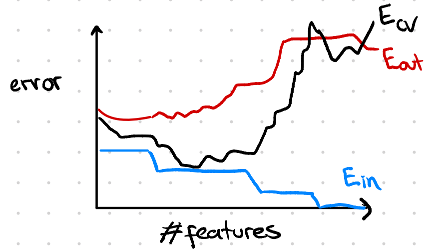
$D_n = (x_1, y_1) \dots \cancel{(x_n, y_n)} \dots (x_N, y_N)$

Final hyp learned from D_n: g_n^-

$e_n = E_{val}(g_n^-) = e(g_n^-(x_n), y_n)$

cross val error: $E_{cv} = \frac{1}{N} \sum_{n=1}^{N} e_n$

catch: e_ns aren't indep!
but not that far away...

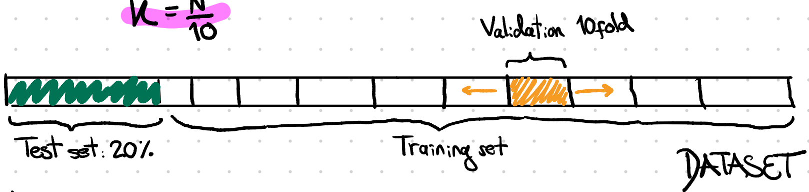


In practice the leave one out is inefficient as N increases

Sol: Take a chunk! $\longrightarrow$ $\frac{N}{K}$ training sessions on N-K points each.

Rule of thumb: 10fold cross validation

$K = \frac{N}{10}$

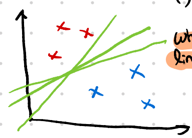


Validation is not data snooping as we have accounted for it.

SVM

Support Vector machine.

- Max the margin.
- The solution.
- Nonlinear transforms. (When data is not lin sep.)



which is the best line for classification?

↳ BIGGER MARGIN → BETTER

> Best model before deep learning.

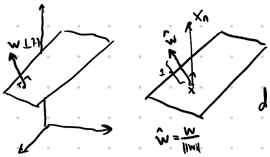
Dichotomies with fat margin

- If we restrict the margin, we get a smaller VC dimension. (Restrict num. of dichotomies)

Let x_n be the nearest point to the plane $w^Tx+b=0$

Technicalities
- Normalize w: $|w^Tx_n+b|=1$
- Take out w_0 from w (as we did here) $w^Tx+b=0$ (no x_0)

To compute the distance:



$$\text{distance} = \frac{1}{\|w\|}\Big|\underbrace{w^Tx_n - w^Tx}_{=1}\Big| = \frac{1}{\|w\|}$$

$w^Tx_n \to =1$, $w^Tx \to =0$

(The bias is omitted as: $+b-b=0$)

The optimization problem

$$(1)\begin{cases}\max \frac{1}{\|w\|} \\ \text{st} \min\limits_{n=1,\dots,N} |w^Tx_n+b|=1\end{cases} \xRightarrow{\text{problema análogo.}} (1.\text{Alt})\begin{cases}\text{Min } \frac{1}{2}w^Tw \\ \text{st} \quad y_n(w^Tx_n+b)\geq 1\ \forall n\end{cases}$$

this is friendlier.

Obs: $|w^Tx_n+b| = y_n(w^Tx_n+b)$ // we only take w that classify correctly.

This is a Constrained Optimization problem!

- But we need equalities... ⍨ → Add slack variables!

- This is similar to the regularization problem

	TO OPTIMISE	CONSTRAINT
Regularization	E_{in}	w^Tw
SVM	w^Tw	E_{in}

We thus formulate the lagrangian of our problem

$$(1)\begin{cases}\min \frac{1}{2}w^Tw \\ \text{st} \\ y_n(w^Tx_n+b)\geq 1\end{cases} \xrightarrow{\mathcal{L}} \min\left[\mathcal{L}(w,b,\alpha) = \frac{1}{2}w^Tw - \sum^N \alpha_n(y_n(w^Tx_n+b)-1)\right]$$

w.r.t to w and b and maximise w.r.t each $\alpha_n \geq 0$

$$\hookrightarrow \nabla_w \mathcal{L} = w - \sum^N \alpha_n y_n x_n = 0 \leftrightarrow w = \sum^N \alpha_n y_n x_n \quad (1)$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{n=1}^N \alpha_n y_n = 0 \leftrightarrow \sum^N \alpha_n y_n = 0 \quad (2)$$

We write the dual formulation of the problem

Substitute (1) & (2) in $\mathcal{L}(w,b,\alpha)$

$$\hookrightarrow \mathcal{L}(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2}\sum_{n=1}^N\sum_{m=1}^N y_n y_m \alpha_n \alpha_m x_n^T x_m$$

where maximize wrt α subject to $\alpha_n \geq 0\ \forall n=1,\dots,N$ and $\sum^N \alpha_n y_n = 0$

↳ with respect to

The solution to the problem (quadratic programming)

↳ prog lineal pero con funciones cuadráticas.

We have $\max_{\alpha} \sum^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m$

$$\Downarrow$$

$$\min_{\alpha} \frac{1}{2}\sum^{N}\sum^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m - \sum^{N} \alpha_n$$

$$\Downarrow$$

$$\begin{cases} \min_{\alpha} \frac{1}{2}\alpha^T \begin{bmatrix} y_1 y_1 x_1^T x_1 & \cdots & y_1 x_N x_1^T x_N \\ \vdots & \ddots & \\ y_N y_1 x_N^T x_1 & & y_N y_N x_N^T x_N \end{bmatrix} \alpha + (-1^T)\alpha \\ \text{st } y^T\alpha = 0 \end{cases}$$

$0 \le \alpha \le \infty$

It's just
$$\begin{cases} \min_{\alpha} \frac{1}{2}\alpha^T Q \alpha - 1^T\alpha \\ \text{st } y^T\alpha = 0, \alpha \geq 0 \end{cases}$$

Obs: It's a convex function. No problem in optimising.

→ Obs: When N is too large ($N > 10000$) we need to use other heuristics to solve this.

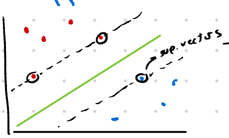
Using Quadratic programming, we get the solution $\alpha = \alpha_1, -, \alpha_N$

$$W = \sum^{N} \alpha_n y_n x_n$$

KKT conditions:
For $n = 1, -, N$
$$\alpha_n(y_n(w^T x_n + b) - 1) = 0$$

There are going to be some $\alpha_n > 0$ ⟹ Their respective X_n are the support vectors

Support Vectors



$$W = \sum_{X_n \in SV} \alpha_n y_n x_n$$

W has a lot less parameters than a linear model ($\#\alpha_n > 0$)

Now we solve for b using any SV
$$y_n(W^T x_n + b) = 1$$

non-Linear Transformations $\mathcal{X} \to \mathcal{Z}$

We work with z instead of x: $z_1 =$

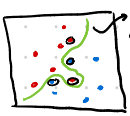
$$\to \mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum\sum y_n y_m \alpha_n \alpha_m z_n^T z_m$$

Obs: $\#\alpha_n$ has nothing to do with the dimension of x or z, only N: the datapoints.

The support vectors in "$\mathcal{X}$" space

$$\boxed{\mathbb{E}[E_{OUT}] \leq \frac{\mathbb{E}[\# \text{ of SV's}]}{N-1}}$$

- Support vectors live in $\mathcal{Z}$ space
- In $\mathcal{X}$ space, "pre images of SV



→ only 4 SV: 0 → 4 params.

SVM allows you to go very sophisticated without fully paying the price of it.

Kernel Methods

1- Kernel trick
2- Soft margin

> Introducing: Infinite dim. spaces!

→ The kernel trick.

$$\mathcal{L}(\alpha) = \sum^{N} \alpha_n - \frac{1}{2} \sum^{N} \sum^{N} y_n y_m \alpha_n \alpha_m z_n^T z_m$$

st $\alpha_n \geq 0, \sum_{n=1}^{N} \alpha_n y_n = 0$

$$g(x) = \text{sign}(w^T z + b), \text{ where } w = \sum_{z_n \text{ is SV}} \alpha_n y_n z_n \text{ and } b: y_m(w^T z_m + b) = 1$$

Given two points x and $x' \in \mathcal{X}$, get $z^T z'$?

$z^T z' = K(x, x')$ (the kernel)
↳ Idk what's that.

Example: $x = (x_1, x_2) \to$ 2nd order poly.
$z = \Phi(x) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$
$K(x,x') = z^T z' = 1 + x_1' x_1 + x_2 x_2' + x_1^2 x_1'^2 + x_2^2 x_2'^2 + x_1 x_1' x_2' x_2$

Le trick: can we compute $K(x,x')$ without transforming x and x'?

Ex: Consider: $K(x,x') = (1 + x^T x')^2 = (1 + x_1 x_1' + x_2 x_2')^2$
$= 1 + x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_1' x_2 x_2'$

This is an inner p.
$(1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2)$
$(1, x_1'^2, x_2'^2, \sqrt{2}x_1', \sqrt{2}x_2', \sqrt{2}x_1' x_2')$

The polynomial kernel

$\mathcal{X} = \mathbb{R}^d$, $\Phi: \mathcal{X} \to \mathcal{Z}$ is a polynomial of order Q

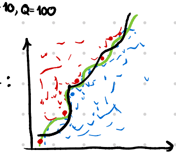
The equiv. kernel $K(x,x') = (1 + x^T x')^Q = (1 + x_1 x_1' + x_2 x_2' + \ldots + x_d x_d')^Q$

It can be very ugly for $d = 10, Q = 100$

We can adjust coefficients: $K(x,x') = (a x^T x' + b)^Q$

Now, let's try without knowing what $\mathcal{Z}$ is.

Kernel n action:



Let $K(x,x')$ be the inner product in some space $\mathcal{Z}$

Ex: $K(x,x') = \exp(-\gamma \|x - x'\|^2)$

The corresponding $\mathcal{Z}$ of this kernel is Infinite dimensional.

We commute a finite inner product from an infinite dimensional space!

Take an example: $K(x,x') = e^{-(x-x')^2} = e^{-x^2} e^{-x'^2} \underbrace{\sum_{k=0}^{\infty} \frac{2^k (x)^k (x')^k}{k!}}_{\exp(2xx')}$ (Taylor series for exp)

Factor to divide

How can we know, given a kernel candidate, if its respective $\mathcal{Z}$ exists.

3 approaches:
- → Construction. (when we want to test a new kernel.)
- → Math prop of the kernel (Mercer's conditions)
- → Who cares? I never visit it.
  ↳ sometimes you succeed xd.

How do we use It?

Instead of passing $y_i y_j x_i^T x_j$ to the quadratic programming, we pass $y_i y_j K(x_i, x_j)$

Express $g(x) = \text{sign}(w^T z + b)$ in terms of K

$$w = \sum_{z_n \text{ in SV}} \alpha_n y_n z_n \to g(x) = \text{sign}\left(\sum_{\alpha_n > 0} \alpha_n y_n K(x_n, x) + b\right)$$

Depends on x_n.

where $b = y_m - \sum_{\alpha_n > 0} \alpha_n y_n K(x_n, x_m)$ for any $\alpha_m > 0$.

Design your own kernel

$K(x, x')$ is a valid kernel $\longleftrightarrow$ it is symmetric $\wedge$ $\begin{bmatrix} K(x_1,x_1) & \cdots & K(x_1,x_N) \\ \vdots & & \\ K(x_N,x_1) & & K(x_N,x_N) \end{bmatrix}$ is positive semi-definite $\forall\, x_1, \dots, x_N$

C'est les conditions de Mercer.

Soft Margin SVM

Data can be SLIGHTLY non separable to inseparable.

 → sol: soft margin

 → sol: kernel.

Margin Violation: A point may violate the margin ↳ the cushion...

Quantify: $y_n(w^T x_n + b) \geq 1 - \xi_n$, $\xi_n \geq 0$
↳ "slack"

Total violation: $\sum_{n=1}^{N} \xi_n$

Constant that defines the importance between $w^T w$ and $\sum \xi_n$

<u>The new optimization</u>: $\begin{cases} \min\ w^T w + C \sum^{N} \xi_n \\ \text{st } y_n(w^T x_n + b) \geq 1 - \xi_n \\ \xi_n \geq 0 \end{cases}$

We go to the Lagrangian again...

New multiplier for parameters ξ_i.

$$\mathcal{L}(w, b, \xi, \alpha,\ ) = \frac{1}{2} w^T w + C \sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} \alpha_n (y_n(w^T x_n + b) - 1 + \xi_n) - \sum^{n} \beta_n \xi_n$$

$\nabla_w \mathcal{L} = w - \sum^{N} \alpha_n y_n x_n = 0$

$\frac{\partial \mathcal{L}}{\partial b} = -\sum \alpha_n y_n = 0$

$\frac{\partial \mathcal{L}}{\partial \xi_n} = C - \alpha_n - \overset{\geq 0}{\beta_n} = 0$

$\alpha_n \leq C$

So the solution is: $\begin{cases} \max\ \mathcal{L}(\alpha) = \sum \alpha_n - \frac{1}{2} \sum^{N} \sum^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m \\ \text{st} \quad 0 \leq \alpha_n \leq C, \ \sum \alpha_n y_n = 0 \end{cases}$

$\longrightarrow w = \sum \alpha_n y_n x_n$

which minimises: $\frac{1}{2} w^T w + C \sum^{N} \xi_n$

Types of support vectors

- margin support vectors $(0 < \alpha_n < C)$
  ↳ $y_n(w^T x_n + b) = 1$ $(\xi_n = 0)$
- non-margin support vectors $(\alpha_n = C)$
  $y_n(w^T x_n + b) < 1$ $(\xi_n > 0)$

C is defined using cross-validation

Two technical observations

↳ 1 – What if data non-linearly sep?

↳ 2 – Z: what about the bias (w_0)?
All goes to b and $w_0 \to 0$

Radial Basis Functions (RBF)

Each $(x_n, y_n) \in D$ influences $h(x)$ based on $\overbrace{\|x - x_n\|}^{\text{radial}}$

↳ closer → + influence.

Standard form: $$h(x) = \sum_{n=1}^{N} w_n \underbrace{e^{-\gamma\|x-x_n\|^2}}_{\text{basis function.}}$$

The Learning Algorithm.

(Finding w_i).

* I have N datapoints, I'm trying to learn N params

We want $E_{in} = 0$, $h(x_n) = y_n$

$$\to \sum_{m=1}^{N} w_n e^{-\gamma\|x_n - x_m\|^2} = y_n$$

↳ N eq, N unkn!

$$\underbrace{\begin{bmatrix} \exp(-\gamma\|x_1-x_1\|^2) & \cdots & \exp(-\gamma\|x_1-x_N\|^2) \\ \vdots & \ddots & \\ \exp(-\gamma\|x_N-x_1\|^2) & \cdots & \exp(-\gamma\|x_N-x_N\|^2) \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix}}_{W} = \underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}}_{y}$$

- If Φ is inv $\to W = \Phi^{-1} y$ (perfect interpolation)

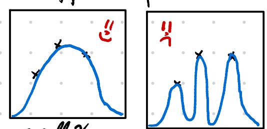
- Often RBF is used for classification.

$$h(x) = \text{sgn}\Big(\underbrace{\sum_{n=1}^{N} w_n e^{-\gamma\|x_n - x\|^2}}_{s}\Big)$$

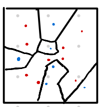
↳ min $(s-y)^2$ on D $(y = \pm 1)$

$h(x) = \text{sign}(s)$

Effect of γ



RBF ~ KNN



- Instead of using a gaussian, for the basis function. Lets use a function that is like a cylinder. $(k=1)$

The bigger k, the smoother should be the basis function

RBF with K centers

N parameters, $w_0, \ldots, w_N$ based on N datapoints.

Sol: Use $K \ll N$ centers: $\mu_1, \ldots, \mu_k$ instead of $x_1, \ldots, x_N$

$$h(x) = \sum_{k=1}^{K} w_k \exp(-\gamma\|x - \mu_k\|^2)$$

1- How can I choose μ_k? ⟶ Min distance between each x_n and its closest centroid μ_k.

2- How can I choose the weights?

↳ C'est K-means clustering.

Split $x_1, \ldots, x_N$ into $S_1, \ldots, S_k$

minimise $$\sum_{k=1}^{K} \sum_{x_n \in S_k} \|x_n - \mu_k\|^2$$

- Example of unsupervised learning ☺
- NP-hard problem ☹

Find the minimum is NP-hard... we need an algorithm to get a decent sol

↳ Lloyd's Algorithm: Iteratively minimise: $$\sum_{k=1}^{K} \sum_{x_n \in S_k} \|x_n - \mu_k\|^2 \text{ wrt } \mu_k, S_k$$

$$\mu_k \leftarrow \frac{1}{|S_k|} \sum_{x_n \in S_k} x_n$$

$$S_k \leftarrow \{x_n \,/\, \|x_n - \mu_k\| \le \text{all } \|x_n - \mu_\ell\|\}$$

We converge to a local minimum, which depends on the initial clustering.

$$\sum_{k}^{K} w_k e^{-\gamma\|x_n - \mu_k\|^2} \approx y_n$$

N eq, $K < N$ unk

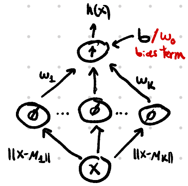
$$\underbrace{\begin{bmatrix} \exp(-\gamma\|x_1-\mu_1\|^2) & \cdots & \exp(-\gamma\|x_1-\mu_k\|^2) \\ \vdots & \ddots & \\ & & \exp(-\gamma\|x_N-\mu_k\|^2) \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} w_1 \\ \vdots \\ w_k \end{bmatrix}}_{W} \approx \underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}}_{y}$$

If $\Phi^T\Phi$ inv $\to W = (\Phi^T\Phi)^{-1}\Phi^T y$

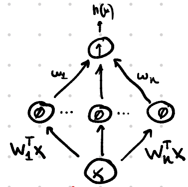
RBF Network

Features: $e^{-\gamma\|x-\mu_k\|^2}$

Nonlinear transform depends on D



RBF Network



Neural Network

Choosing γ

$$h(x) = \sum_{k=1}^{K} w_k e^{-\gamma\|x-\mu_k\|^2}$$

Iterative approach (~EM algoritm in mixture of Gaussians) → Expectation maximisation

1. Fix γ, solve for W
2. Fix W, minimize error wrt γ

Obs: We can have a different γ_k for each μ_k.

RBF & Regularization

RBF can be derived purely from regularization.

$$\sum^{N}(h(x_n)-y_n)^2 + \lambda\sum_{k=1}^{\infty} a_k \int_{-\infty}^{+\infty}\left(\frac{d^k h}{dx^k}\right)^2 dx$$

This is the smoothest interpolation. Which solution is RBF. aprox.

SVM kernel implements:

$$\text{sign}\left(\sum_{\alpha_n>0} \alpha_n y_n e^{-\gamma\|x-x_n\|^2} + b\right)$$

RBF implements

$$\text{sign}\left(\sum_{k=1}^{K} w_k e^{-\gamma\|x-\mu_k\|^2} + b\right)$$

Three Learning Principles

- Occam's razor
- Sampling bias
- Data snooping

> An explanation of the data should be as simple as possible, but no simpler (A. Einstein)

> The razor of occam: The simplest model that fits all the data is also the most plausible.

- What is simple?
- Why simple → better?

Measures of complexity

- Complexity of h → MDL (minimum desc length), Order of a polynomial
- Complexity of $\mathcal{H}$ → Entropy, VC-dimension

Kolmogorov Complexity → describe with as few bits as possible.

Simpler → better E_{out} performance

Fewer simple hyp than complex ones are less likely to fit the data: $\frac{m_{\mathcal{H}}(N)}{2^N}$

So when it happens its more significant.

When we think of simple, it's in terms of h. Proof use simple in terms of $\mathcal{H}$...

They are related to the principle of counting:

ℓ bits specify h → h is one of 2^ℓ elements of $\mathcal{H}$

Exception: SVM; looks complex but is one of few.

- **Sampling bias:** If the data is sampled in a biased way, learning will produce a similarly biased outcome.

If you have a population that was not used in training, we have a sampling bias if they appear on testing.

- **Data snooping** → If the data has affected any step in the learning process, its ability to asses the outcome has been COMPROMISED

* Most common trap - many ways to slip.

Examples of Snooping

1.→ Looking at the Dataset: $z=(1,x_1,x_2,x_1x_2,x_1^2,x_2^2)$. Why not $z=(1,x_1^2+x_2^2)$? The data fits good (just a circle)
Note: you can take into account Info you have of f (sym in -2,2 for ex), while you not look at $\mathcal{D}$.

1.5: Example: $\underbrace{\Delta_{r-20} \dots \Delta_{r-1}}_{\text{from the past}} \rightarrow \underbrace{\Delta_{r_0}}_{\text{value of now.}} \Rightarrow$ Normalize data, split $\mathcal{D}_{train}, \mathcal{D}_{test}$

Wrong order!
You give the mean and var of the test set to the algo.

2.→ Trying one model after the other ON THE SAME DATASET, you will eventually succeed.

> If you torture the data long enough, It will confess.

- VC dimension of the total learning model.
- May include what others tried! // This paper says that SVM with $\beta = \frac{1}{2}$ is perfect!
- The problem is matching for a PARTICULAR dataset

Avoid data snooping
discipline
Datasnooping
Account for data snooping.
> how much data contamination.

Bayesian Learning

We extend probability to calculate $P(\mathcal{D} \mid h=f)$

$$\rightarrow \underbrace{P(h=f \mid \mathcal{D})}_{\text{posterior}} = \frac{P(\mathcal{D} \mid h=f)\,P(h=f)}{P(\mathcal{D})} \propto \underbrace{P(\mathcal{D} \mid h=f)}_{\text{likelihood}}\,\underbrace{P(h=f)}_{\text{prior}}$$

- Key: The prior is an assumption that can be wrong → Bayesian learning is justified when
  - The prior is valid — trumps all other methods
  - The prior is irrelevant
    ↳ $\pi_0 \dots \pi_{100000}$ (doesn't matter π_0 that much)

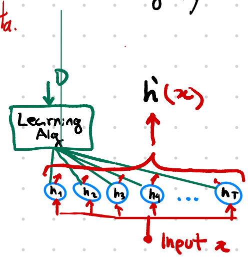
Knowing the prior we can find the most probable h given the data.
Can we really calculate $P(h=f)$?

Aggregation

Combining many models $h_1, \dots, h_T$ that were trained on $\mathcal{D}$.

Tactics:
- Take an average. (Reg.)
- Take a vote. (Classif.)

> ensemble learning



Aggregation can be done

1. After the fact → combines existing solutions
2. Before the fact → creates solutions to be combined
   ↳ Boosting (AdaBoost): create $h_1, \dots, h_T$ which are decorrelated with the previous h that were created at each step

We care about missclasifications in $\mathcal{D}$
Each model is weighted:

$$g(x) = \sum^{T} \alpha_t h_t(x)$$

We choose α_ts to minimize the error on the aggregated dataset