MACHINE LEARNING

= Pill 1: Machine Learning =

= Warm up (I)

One may measure intelligence through the number of available solutions or new approaches to problems.

= Warm up (II)

Philosophical assumptions: • that our mind is computationalism-natured
• functionalism: our mind state depends on - previous state
- perception
- ?
} ⇒ we may replicate and build IA

counter-believes

religion, "emotions aren't computational" ⇒ • Weak AI: only some parts are repicable, the rest are imitated
• Narrow AI: what we generally refer to when mentioning AI

= Warm up (III)

We don't want new 'colleagues' ⇒ we want tools we may get along with

= Why ML now?

venn diagram.

Counterfactual " "
Interventional Casual-Effects: we could model the world into casual-effects, if everthing was determined by casuality.
Stats + ML

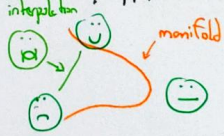
= Key elements for ml. → pattern, data, not down-pinnable mathematically

With no data, there's no ML. Without pattern, it'd be random and the result too.

= Types of ML

• Manifold learning: let's say a manifold is a subspace where we may apply the euclidean distance

Let's consider a subspace of emotions:
what's in the middle of 😊 😐
(with solution 😐)



> Embeddings. text embeddings Embeddings represent entities through vectors. (feature vectors)

• Reinforcement training: ~ "how parents teach kids" ML models that learn games.
It learns to "win", not to carry out the actions or comprehend the complexity of what it's doing, if that makes sense.

= Let's discuss the following problems

• Supervised.
• Supervised through clustering + unsupervised through TFG.
• Unsupervised, but it could be rewritten as reinforcement. (student idea)
• Reinforcement.

= Let us discuss about the following problems (slide 21)

• Could be both (Regression/classification): Learn close estimate of the model / Y/N for each possible action or which action is best
• Both. Most likely zone / Y/N for each bit (subregion of image; sliding window)

= Pragmatic ML pipeline

The model should encode the data. • Training
• Exploitation (testing)

= Knowledge representation and feature extraction =

Labels 0,1 represent each box.

More available high-level information: points of the star, radius of the ellipse...

pre domain knowledge needed

Raw Data → Feature Extraction → Machine Learning

↑ become; to have implicito.

↳ Deep Learning

↗ reduce dimensionality, capture discriminant data

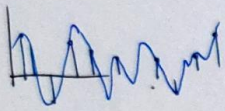
	Raw Data	Feat. Extr.
→ Domain specific knowledge	↓	↑
Complexity	↑	↓
Dimensionality	↑	↓
Redundancy	↑	↓

= Encoding a document

Bag of words: $w_1 \dots$ $w_i \dots$ w_N

1 0 0 2 ... 1 0 0 ... 0 0 1 → Grammar-free, semantic-only (podría haber un texto con las mismas palabras que signifique algo diferente)

= Shazam™



Encode the amplitude at each time

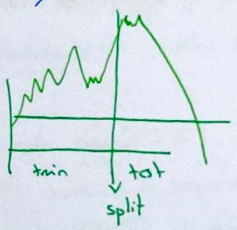
↳ Encode them in Fourier (lo veremos)

Para identificar números: interesting features number of active pixels, H/V symmetry

- A classifier is an assistant that does the boundary lines between labels.

We must scale in training:

Imagine we scaled in whole data set in stock prediction



is leaking info from the future.

→ split in test, train

scale (train)

train ($\overline{\text{train}}$)

test ($\overline{\text{test}}$)

* It might not be harmful if data is homogenous.

= Model selection { selecting model / choosing hyperparameters ⇒ "it's a form of training": you choose the best option

training → params in each model

VALIDATION

testing

Last slide?

Don't change models after testing (you would be validating again) ↳ no test data.

↓

NEVER TAKE DECISIONS ON UNSEEN DATA

= Pill 2:

- Outliers: errors in measurements, values that are very unlikely

= Handling missing data. { drop the sample / the feature / use the average / use a model (regression) / doesn't encode important information? → maybe use the average of neighboring points (in order not to change regression line too much)

= Feature extraction must be independent ⇒ non-independency splits importance among dependent features.

non-linear behaviour in linear models.

ax linear model

Feature extraction → add values

$\sum a_i x^i$; dataset $x \in \mathbb{R}$ [x] → fe → $[x, x^2, x^3, x^4, x^5]$ → reg. linear regression → Find a_i s.t. $a_1 x + a_2 x^2 + a_3 x^3 + \dots$

= Encoding data

small → 0
medium → 1
big → 2
↓ categorical ↳ ↓ order

?

red → 0
blue → 1
green → 2
↓ cat. ↳ this may not be that good

↑ RGB value could be useful

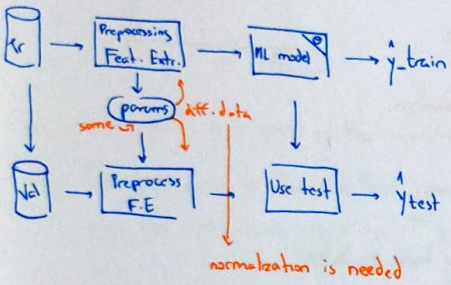
} no relationship between color and value

- One-Hot Encoding:

color	red?	blue?	green?
red	1	0	0
blue	0	1	0
green	0	0	1

redundant in this case

→ .get_dummies!.



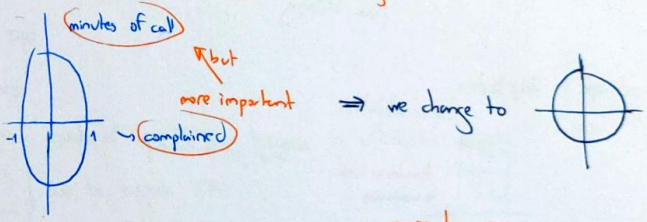
- Normalization: $\bar{x} = \frac{x - \mu_x}{\sigma_x}$ Standardization

ONLY WITH FEW OUTLIERS

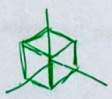
alt. $\hat{x} = \frac{x - \min(x)}{\max(x) - \min(x)}$

$\bar{x}_{mc} = \frac{x_{mc} - \mu_x^{mc}}{\sigma_x^{mc}}$

2 minute change has more weight



other examples: change to unit cube



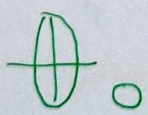
$\bar{x}_{mc} = \frac{x_{mc} - \min(x_{mc})}{\max(x_{mc}) - \min(x_{mc})}$

↑ more to origin

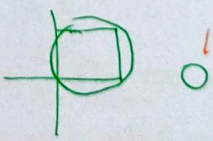
↳ squish to (0,1)

this range squish might be mostly ruined by outliers.

- Identify outliers: imagine 10% outliers



use the 90% range(s) to normalize



they will still be there "normalized"

↓ delete?

- Feature Hashing: when embedding are not usable.

HASH: deterministic f(obj) → gives always same number for an object within a customable range

important (0,1)

f(△) = |△| | |

f(□) = |△| |□| collisions

f(○) = |△○| |□| possible

↓ usable as a index to store the object.

must be pseudo-random.

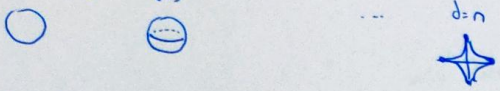
good practice for it to be in terms of the value to hash (deterministic)
and the range (to fit in the boxes → %)

We shouldn't use too much space: waste
not good encoding (as stated before) ⇒ representation vector: count the occurences of each possible hash value

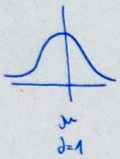
embedding ←

sparse, efficient, ↑complexity ↓ collisions, ~uniform, no metric notion, set dimensionality of embedding space

unit ball $d=2$ $d=3$ ... $d=n$

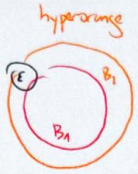


Gauss distribution



"n-dim orange" most of the volume is in the peel

d=n



$V_{B_2} - V_{B_1} = V_\varepsilon >>>>>>>>>> V_{B_1}$

=Performance=

LOO represents an intersection of ands (in this location, this gender...)

↓

small piece where the value is.

Iterative LOO may take ages ⇒ limit iterations to (K) → usually 10-fold or 2 times 5-fold (or as long as your data life is.)

split dataset into K pieces

each piece used for testing/training in each it

K-fold cross validation.

↳ buen training: usamos todos los datos para training

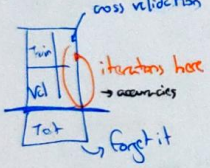
↳ K accuracies → mean... → mean...

↓

buen cálculo: usamos todos los datos.

validation? opt. 1: skip testing

opt. 2:



(used in business)

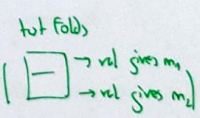
opt 3 (academic): nested cross-validation: two loops: one for testing and the other for validation.

outer-loop: decides which data is taken for test (2-fold test)

inner-loop: validates with 3-fold val...

↓

not example { doesn't work to decide which model works better / it just tells us the best acc. we may get



the model parameters are reset in each iteration (no accumulative training)

↳ we don't put the params together in the end, we just work with the accs.

then we train the model with all data and "most likely" its accuracy it's going to be at least the accuracy in testing (as it was tested with less data)

↓

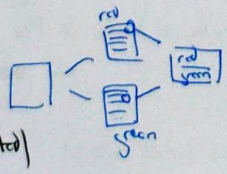
with most acc

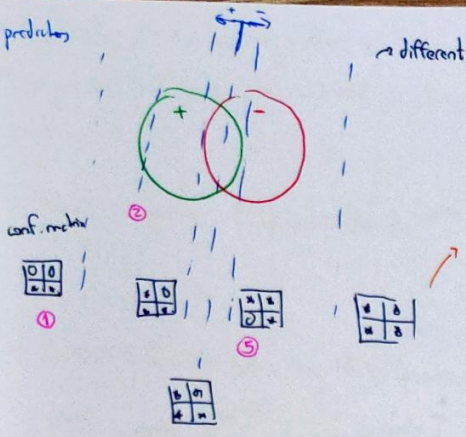
can we take it. be test

(learn mic, no he aplicado)

Stratified cross-validation

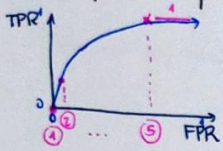
(multiclass when some classes are misrepresented)





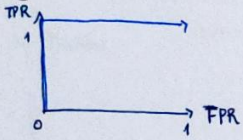
• ROC: can be represented by the curve defined by (FPR, TPR) They are not complementary
" 1-spec. " recall

↳ in case of perfect pred: (0,1).

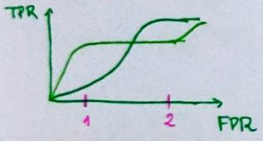


- An operating point is a point on the operating curve (which is chosen a threshold)
- prop: The best curve is the one that grows the fastest. This may be identified by calculating the area. ▶

obs: the ideal curve would be



⚠ This might not be our decision criteria: imagine two rocs



our choice would depend on how much of a FPR our business is willing to accept:
if it's ① } we would choose } — } due to higher TPR
② } — }

= Compaign exercise: } cost = all positives × 10
ben. = ↳ I guess: TP × 100

Cost: $\underset{\sim 10}{\alpha}$ units/person * (TP+FP) ppl
ppl model predicted will recover

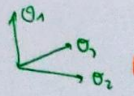
Benefício: β × TP × 100
↓
el modelo predice quien va a churnear. con la campaña: } 10% sigue
[TP] } 90% churnea

(loss would be loss = 100 * (TP+FN)
money
↓
cree que — leaving and no campaign
Falta un 0'9* aquí, ya que salvamos al 10%

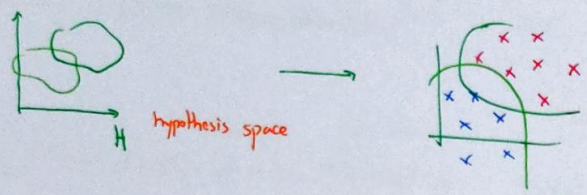
= Pill 4: Opening The Blackbox (Disections ML methods)

$\hat{y} = h(x) = h(x;\theta)$
↓ ↓ ↓ ↓
prediction (output) — model — input — set of parameters

Suppose that $\hat{y} = h(x;\theta) = \theta_1 x^2 + \theta_2 x + \theta_3$



parameter space is not generally used.



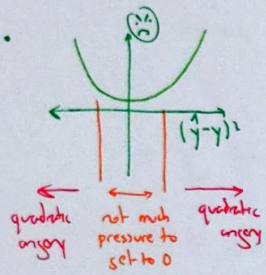
We are minimizing respect to θ the following error, in predictions notion: $\frac{1}{N}\sum_{i=1}^{N} e(\hat{y}_i, y_i) + \dots$

↓ loss generally

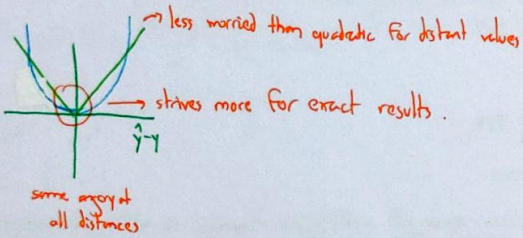
↓ loss

we may add other terms in the future

It is convenient to model an irritable function ~ make it "angry" when the value is not correct



"The more it differs, the angrier it gets" ⟶ LES $\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$



- Linear Angry

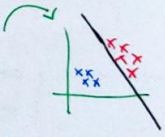
= human learning algorithm exercise:

model class: linear functions.

loss: error loss $\mathbb{1}_{[y \neq \hat{y}]} = \begin{cases} 0 \text{ if } y = \hat{y} \\ 1 \text{ if } y \neq \hat{y} \end{cases}$ ⟶ this might give a fine that is too close to one point to label

↳ indicator: classical loss in classifier.

↓ solvable by extra terms



Remember: $h(x) = w_1 x + w_0 = \begin{bmatrix} w_1 & w_0 \end{bmatrix}\begin{bmatrix} x \\ 1 \end{bmatrix} = w^T \tilde{x}$ ↳ extended data that integrates the offset.

minimize $\frac{1}{N}\sum_{i=1}^{i=N}(y_i - w^T \tilde{x}_i)^2 = \frac{1}{N}(y - w^T\tilde{x})^T(y - w^T\tilde{x})$

// suitable because it's regression in this example

where $y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$, $X = \begin{bmatrix} x_1 \dots & 1 \\ x \dots & 1 \end{bmatrix}$

↓ vectorial form | make derivative easy.

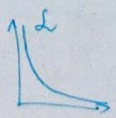
We need to get the gradient of the loss respect to parameters $\nabla_w \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_0}\right]$

$\nabla_w \mathcal{L} = -2\tilde{x}^T(y - w^T\tilde{x})$

↓ vectorial gradient, independent from dimensionality

curve of descense gradient: convergence curve.

Expected loss behaviour



= Feasibility of the learning problem =

$P(|\gamma-\mu| \geq \varepsilon) \leq \delta \searrow \phi$ → Hoeffding's Inequality (no explica de donde sale -yet-

odds of mistake

para N ha de ser muy grande para compensar una ε muy pequeña)

• Hoeffding's inequality: $P[|\gamma-\mu|>\varepsilon] \leq 2e^{-2\varepsilon^2 N} \Rightarrow P(|E_{in}-E_{out}| \geq \varepsilon) \leq e^{-\varepsilon^2 N}$

out of sample error: expected error

in sample error (freq of hypotesis being wrong)

- Markov's inequality: Let x be (R.V) (positive), $P(x \geq a) \in \frac{E[x]}{a}$

real value

proof: $E[x] = \int_{-\infty}^{\infty} x\,p(x)\,dx = \int_{0}^{\infty} x\,p(x)\,dx = \int_0^a x\,p(x)\,dx + \int_a^\infty x\,p(x)\,dx \geq \int_a^\infty x\,p(x)\,dx$

pos.

We now only need to see that $\int_a^\infty x\,p(x)\,dx \geq \int_a^\infty a\,p(x)\,dx$

the area of $x\,p(x)$ is bigger the $a\,p(x)$.

$\Rightarrow \int_a^\infty x\,p(x)\,dx \geq \int_a^\infty a\,p(x)\,dx = a\int_a^\infty p(x)\,dx = a\,P(x \geq a)$ ∎

- Chernoff bound: positive RV x and $\varepsilon>0$: $P(|x-\mu| \geq \varepsilon) \leq \min_{\lambda>0} \mathbb{E}[e^{\lambda|x-\mu|}]e^{-\lambda\varepsilon}$

proof: $P(|x-\mu| \geq \varepsilon) \leq \frac{E[|x-\mu|]}{\varepsilon}$ by Markov

the prob. is the same ⇕ (∞)

$P(e^{|x-\mu|} \geq e^{\varepsilon})$

also ⇕ $\lambda>0$

$P(|x-\mu| \geq \varepsilon) = P(e^{\lambda|x-\mu|} \geq e^{\lambda\varepsilon}) \leq \underbrace{\mathbb{E}[e^{\lambda|x-\mu|}]e^{-\lambda\varepsilon}}_{\text{for any } \lambda} \Rightarrow P(|x-\mu| \geq \varepsilon) \leq \min_{\lambda>0} \mathbb{E}[e^{\lambda|x-\mu|}]e^{-\lambda\varepsilon}$ ∎

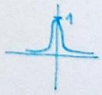
- Simplified Hoeffding's Inequality: Let z_i be independent bounded RV such that $z_i \in [a,b]\ \forall i$: $P\left(\frac{1}{N}\sum_{i=1}^N (x-\mu_i) > \varepsilon\right) \leq e^{-\frac{2N\varepsilon^2}{c}}$

where $c \stackrel{?}{=} 8(b-a)$

to proof: $P\left(\frac{1}{N}\sum_{i=1}^N x_i > \varepsilon\right) \leq e^{-\frac{N\varepsilon^2}{2}}$

proof (for a binary case): z_i are Rademacher RV: distribution s.t. $f(k) = \begin{cases} 1/2 & \text{if } k=\pm 1 \\ 0 & \text{otherwise} \end{cases}$

in terms of Dirac delta function → value 1 area del 0



$P(z_i) = \frac{1}{2}[\delta(z_i-1)+\delta(z_i+1)]$ → Fair coin (check what happens with 0,1)

$\Rightarrow P\left(\frac{1}{N}\sum_{i=1}^N z_i > \varepsilon\right) = P\left(\sum_{i=1}^N z_i > N\varepsilon\right) \leq \mathbb{E}[e^{\lambda\sum z_i}]e^{-\lambda N\varepsilon} = \mathbb{E}\left[\prod_{i=1}^N e^{\lambda z_i}\right]e^{-\lambda N\varepsilon} = \prod_{i=1}^N \mathbb{E}[e^{\lambda z_i}]e^{-\lambda N\varepsilon}$

$= \mathbb{E}[e^{\lambda z_i}]^N e^{-\lambda N\varepsilon} \leq \min_\lambda e^{\lambda^2 N/2 - \lambda N\varepsilon} = e^{-\varepsilon^2 N/2}$ ④

for any λ → for min λ

$\mathbb{E}[e^{\lambda z_i}] = \frac{1}{2}e^{\lambda} + \frac{1}{2}e^{-\lambda} = \frac{1}{2}(e^{\lambda}+e^{-\lambda}) = \frac{1}{2}\left(\sum \frac{\lambda^i}{i!} + \sum \frac{(-\lambda)^i}{i!}\right) = \frac{1}{2}\sum \frac{\lambda^{2i}}{(2i)!} \leq \sum \frac{\lambda^{2i}}{2^i i!} = \sum \frac{(\lambda^2/2)^i}{i!} = e^{\lambda^2/2}$

Taylor exp. $e^{\lambda} = \sum_{i=1}^{\infty} \frac{\lambda^i}{i!}$

odd summands are twice

$(2i)! \geq 2^i i!$

From 0...N it's always the same than 2 is always smaller than i

(From $i+1$ to $2i$: $i \to 2 \cdot \underset{i}{\sim} \cdot 2 = 2^i$)

④ $\frac{\varepsilon^2 N}{2} - \varepsilon^2 N = -\frac{\varepsilon^2 N}{2} \not\to \left(\frac{\lambda N}{N} - N\varepsilon\right) = \phi$

$x = \varepsilon$

- Modification: use max to bound the worst case scenario.

↑N, better.

↑|H|, worse. → we will give more notions of the complexity of hypothesis space.

↓ cardinality of hypothesis space.

- We may define that $P[|E_{out} - E_{in}| \geq \epsilon] \leq 2|H| e^{-2\epsilon^2 N} = \delta$ → (*)

a.k.a with $P = 1-\delta$, $|E_{out} - E_{in}| < \epsilon$ holds (complementary)

(*) $\Rightarrow \epsilon = \sqrt{\frac{-\log \delta/2|H|}{2N}} = \sqrt{\frac{\log |H| + \log 2/\delta}{2N}} \approx O\left(\frac{|H|}{N}\right) = O\left(\sqrt{\frac{C}{N}}\right)$

↑ δ has a minimum effect.

↓ $C = |H|$ ∘ $C = f(d_{vc})$ → said in posterior class (not sure)

$E_{out} > E_{in}$ → training error: we can control by optimizing (and overfitting and other terrible things)

↓ generalization error: no control ← we expect that reducing training error will reduce generalization error, we can't assure it though.

$$\Downarrow$$

$$\left\{ E_{in} \leq E_{out} \leq E_{in} + O\left(\frac{|H|}{N}\right) \right\}$$

→ one of the most important equations in training error. (equivt to $E_{out}(f) \approx E_{in}(f)$)

$O\left(\frac{|H|}{N}\right)$ = 0 in best case scenario $\Rightarrow E_{in} = E_{out}$

= Using the bound: example on how to use $|E_{out} - E_{in}| < \epsilon$ to know the needed amount of samples.

= The last consideration: we must move from function to probabilities some blue → {sky, sea} we now will consider the odds of that specific value to be {sky, sea}

- $P(y^* | x^*, \bar{X}, \bar{Y})$ is the target distribution (what we want to learn)

  probability of the prediction ← $\bar{X}, \bar{Y}$: known data ← unseen input

  - We may consider that $P(y^* | x^*) = \underbrace{\mathbb{E}[y|x]}_{\text{expected value}} + \underbrace{\eta(x)}_{\text{noise}}$ where the noise is $\eta(x) = y - S(x)$

    $\| \quad \| = S(x)$

    y

  - $P(x)$ is the input distribution that quantifies the relative importance of x.

  - note that $P(x,y) = P(x)P(y|x) \neq P(y|x)$

    ↓ most meth. models aspire to learn this. Harder, though, because you need $P(x)$ explicitly.

- Learning is to minimize the generalization error ($E_{out} \searrow 0$) ⇝ done through our control on E_{in}

  - conditions: ① $E_{out} \approx E_{in}$

    ② $E_{in} \searrow 0$: achieved with a training model on the data I have.

  bad news: in linear models: $|H| = \infty$

= Vapnik-Chervonenkis dimension

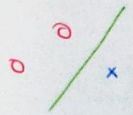
- Shatter: in a binary classification problem, if a model of a given family is able to classify it correctly

- VC-dim of a perceptron in 2D?

  | + 2 entries ⇒ o | x

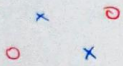
  | + 3 entries ⇒ o x o unpassable test ⇒ VC-dim of | = 2

Now let's add tilt to the classifier:



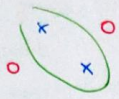
... any label combination will work: ⇒ VC-dim ≥ 3

4 entries:



impossible test ⇒ VC dim of 1 + tilt = 3

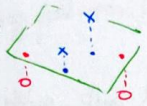
Intuitively, an ellipsis will have higher VC:



- As an intuition, $|H| \approx d_{vc}$ ⇒ Final result formula (substitute H)
- $d_{vc} \approx 10\, df$, where df := degrees of freedom.

3D hyperplane: "if you can do it in 2D you can do it in 3" → 3 entries + ⇒ VC-dim ≥ 3

4 entries:



In fact VC-dim = 4.

- Note that

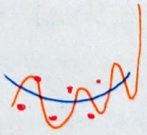
dim		hyperplane d_{vc}
2	→	3
3	→	4
⋮		⋮
{ N	→	d+1 }

= Pill 6: Overfitting =

- Remember: you can regress from $x_1, \dots, x_n$ or from $x_1, \dots, x_1^m, x_2, \dots, x_2^m, \dots, x_n, \dots, x_n^m$ where m is lower than the dim of the model. ≤
  ↳ training set extension
  ↓ $x_1^0, \dots, x_n^0$ should be included $(1, \dots, 1)$ bias representation.
- It is not a good idea always to increase the order (degree of our model) → we need enough data for that. ⇒ overfitting.*

* $E_{in}^{10} \le E_{in}^{3} \le E_{out}^{3} \le E_{out}^{10}$ instead of $E_{in}^{10} \le E_{in}^{3}$ (with $E_{out}^{10} \le E_{out}^{3}$) because at some point E_{out}^{10} stopped following E_{in}^{10}

Happens when there is not enough data for the complexity of the model:



easy model
samples
complex model

not enough data to shape these "curve" (complexities) of the model properly ⇒ it does whatever.

In terms of classification, 0 errors ⇒ forcing to do so, will cause to bad generalization.

"purple islands" ⇒ reduce complexity. to avoid forcing
add data to be "closer to infinity" (sense of generalization)
↳ but with 0 error is horrible. ⇒ the bands are very small ⇒ no influence area to take correct decisions.

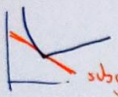
- bias: difference between the limits of training and testing error rates.
  more complexity ⇒ the curves get closer together with less bias.

Attention: $E_{in}[g] + O\left(\frac{C}{N}\right)$ we make it converge to a solution with worse errors.

- We want: - $E_{out} \approx E_{in} \Rightarrow$ balance between C and N at $O\left(\frac{C}{N}\right)$
  - $E_{in} \searrow 0 \Rightarrow \uparrow\uparrow C$ ← ↑↑C not always a good solution.

= Pill 7: Stochastic Subgradient Methods = ④ (next page)

Subgradient for



subgradient that supports (closest hyperplane to the point)

When we choose one of the violated constraints, we do it in order to move towards the feasible set.
But these steps are not guaranteed to be always forward.

If we find $f_0(x^{t+1}) > f_0(x^t)$ → Keep going but keep track best solution POCKET STRATEGY

Stochastic methods let you get past saddle points and local minima (there's some randomness)
↳ most robust strategy so far.

momentum → imitate the movement of letting a ball go.

Adagrad: avoid not arriving ever to the minimum (as it happens to desc. grad) by increasing the step in plain areas
(flat)

if gradient was small in past it. → D^{-1} will increase the speed.

= Pill 8: Regularization - Combatting Overfitting =

We know that $E_{out} \leq E_{in}(C) + O\left(\sqrt{\frac{C}{N}}\right)$

And so far, we minimize $\frac{1}{N}\sum_{i=1}^{N} e(x_i, y_i)$ ⟹ (let's add a term related to complexity) minimize $\frac{1}{N}\sum_{i=1}^{N} e(x_i, y_i) + \lambda C$

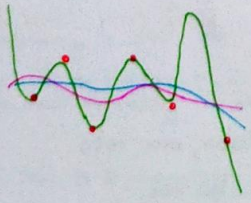
We should be worry if adaptative and analytical solutions are not close.

- Getting the exact solution causes overfitting ⇒ it forces to pass through all the entries. ⇒ EARLY STOPPING ④
  ④ Iterate during few epochs so the method does not memorize.

x = optimum → "solves the problem" → error = 0 → overfitting.

Stoch. Grad. Methods

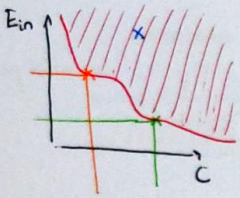
learns / trains →



Early stopping model.

Other alternative: Multi Objective Problem

$\left(\frac{1}{N}\sum_{i=1}^{N} e(x_i, y_i),\ C\right) \rightarrow$ minimize both at the same time



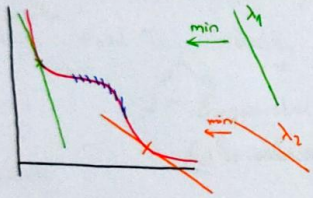
Pareto's optimal surface

We will choose optimal points (they don't have adequate points in their 3rd quad: × ×)

× is not on optimal point.

We should return the whole surface (you may choose from there a balance between training error and complexity) but that's not feasible:

- Scalarization: plot a line that describes the optimal points. minimize $f_1 + \lambda f_2$ ↓ choosing lambda (slope)



"!!" → method has no access to concave zones.

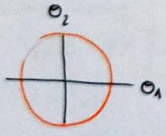
We know now about the λ we added to the minimization. What about C?

Actually, let's consider $\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} e(y_i, h(x_i,\theta)) + \lambda C(\theta)$

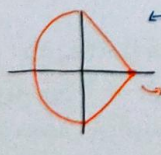
↳ $h(x,\theta) = \sum_{j=1}^{M} \theta_j h_j(x)$ ↗ sparse approx. $\|\theta\|_1$ ↘ "smooth" approx $\|\theta\|_2$

$|\theta_1| + |\theta_2| + \varepsilon + \delta = 1$ ($|\theta_2|$ ↑, $|\theta_1|$ ↑, $\varepsilon+\delta$ ↓, $\varepsilon+\delta$ ↓, $\emptyset$)

- Regularization:



set a problem where $\min \mathcal{L}$ st $\|\theta\|_1 = 1$ ↳ con esta restricción, lo que le sumamos a una coordenada se lo quitamos a la otra. Si unas suben mucho → muchas serán 0.



← changes (constraints) reduce the angles.

↳ many problems have this as solution.

↳ problem is $\min_{\theta} \mathcal{L}(y - \theta^T x) + \lambda\|\theta\|_1$ $+ \gamma\|\theta\|_2$ ↳ add this for a lasso.

$h(x,\theta) = \theta^T x = \sum_{i=1}^{d} \theta_i x^{(i)}$ → d dimensions; → features

↓ many zeros → we are selecting features with impact on the output. TEST-WORTHY

- Data augmentation: (remember it also works to avoid overfitting) - add noise: $\tilde{x} = x + \eta$; $\tilde{x} : x\eta$ with $\eta \sim N$ (normal dist.) (one single kind)
- Dropout: set to 0 features at random (changes pixels to 0 in an image) $\tilde{x} = x \cdot 0$ (feature wise)

= ^ midterm exam contents =

= Decision Theory =

- An agent is someone or something choosing options (actions) based on context evaluation and consequences.

- state - consequence matrix:

Actions / State of the world	OK	Traffic-Jam
A1. Take the bus	arrive early	very late
A2. Walk	arrive late	arrive late

state of the world might not be certain. → in general it won't be

we expect preferences to give us an ordering based on a numerical value (utility): $A \succeq B$, $A \to B$

A) Garlic soup
B) Carrot soup
C) Pumpin cream

$ABC \leftarrow A \Rightarrow \begin{cases} AB \leftarrow A \\ AC \leftarrow A \end{cases}$ ①

$BC \leftarrow ?$ we have to say something about it, e.g. $BC \leftarrow BC$ "tie/idc"

① Irrationalism: not respecting transitivity: $ABC \leftarrow A \Rightarrow AB \leftarrow B$ or

we won't consider this: it should not happen.

› completeness vs incompleteness: completeness shows us if there's irrationalism or not

→ could be rational or not.

- utility: $u(A) \geq u(B) \Leftrightarrow A \succeq B$ (it is a function that respects the order)

Let's take into account the state of the world's: A / S_1 ... N | $P(S_i)$ $P(S_j)$

- Lottery: $L_1 = \langle (p_1, x_1) \rangle$ (50%, cost) where p_j are probabilities and x_j are prizes.

$L_2 = \langle (p_2, x_j) \rangle$ (40%, $45)

compounded lotteries: $\langle (q_1, x_1), \dots, (q_n, x_n) \rangle$ where $\sum_i q_i = 1$

$\langle (90\%, \$10), (10\%, \$100) \rangle$
$\langle (50\%, \$10), (50\%, \$100) \rangle$ → assesment → expected value (prize): $\sum_i x_i q_i = E(L)$

$L_1 \succeq L_2 \Leftrightarrow E(L_1) \geq E(L_2)$

We want to assess $A_1 > A_2$:

A/SoW	$P(S_1)$	$P(S_2)$	probabilities
A_1	+1	-2	prizes
A_2	-1	-1	prizes

→ $A_1 \Rightarrow L_1 = \langle (P(S_1), 1), (P(S_2), -2) \rangle$

→ $A_2 \Rightarrow L_2 = \langle (P(S_1), -1), (P(S_2), -1) \rangle$

Lottery

$\Rightarrow E(U(A_j)) = \sum_{i=1}^{n} u(c_{ij}) P(S_i)$: expected value of the utility of action A_j

apply to conf. matrix

state of the world (let's suppose if it's P or N)

not confusion matrix ← / predicts ↓	P	N
Classifier chooses P	TP (A)	FP (B)
" " N	FN (C)	TN (D)

→ we could add utilities here

preferences (default): $A \sim D$, $B \sim C$, $T \succeq F$: $A = D = 1$, $B = C = 0$

adding our understanding of the SoW: Act/SoW | $P(y=P|x)$ | $P(y=N|x)$

A_1: P — ↓ prob of our classifier to say Pos given data — ↓ prob of it saying Neg.

A_2: N

using lotteries: $L_1 = \langle [P(y=+|x), 1] \rangle = EU(+) = P(y=+|x)$

$L_2 = \langle [P(y=-|x), 1] \rangle = EU(-) = P(y=-|x)$ $\Rightarrow$ If $EU(+) \geq EU(-) \Rightarrow$ classifier uses + $\Rightarrow$

- Assumption 1 (A1): $\exists$ set Ω with $X \times Y \subseteq \Omega$ a σ-algebra $\mathcal{F} \subseteq 2^\Omega$ $(x,y) \sim P_{X\times Y}$ and a probabilistic measure $P: \mathcal{F} \to [0,1]$
  That is $\exists$ probabilistic space $(\Omega, \mathcal{F}, P)$ such that $D \subseteq \Omega$.

Let $D = \{(x_i, y_i)\}$ $i = 1, \dots, N$

$P(y^*|x^*, D)$ —discriminative ML→ Model $P(y|x, D)$ directly —A2→ $P(y^*|x^*, D) = \int_M P(y^*, M|x^*, D)\, dM = \int_M P(y^*|x^*, D, M) P(M|x^*, D)\, dM \Rightarrow$ ④

(A2 over $P(y^*|x^*, M)$; A3 over $P(M|D)$)

↳ I can compress data into a model (basis of physics)

generative ML ↓ A1

$P(x,y)$: Probabilistic

non-parametric models: KNN, Kernels, Gaussian Processes.
↳ the more samples, the more parameters they need.

- Assumption 2: $P(y^*|x^*, M, D) = P(y^*|x^*, M)$ compression (we can get rid of data when we have our model)

- Assumption 3: $P(M|x^*, D) = P(M|D)$

④ $\rightarrow \int_M P(y^*|x^*, M) P(M|D)\, dM$ —classical ML→ A4, A5 ↓ substituting $P(M|D)$ by $\delta(M - M^*)$ (Test)

$P(y^*|x^*, D) = P(y^*|x^*, M)$

↓ Ensemble learning

$\sum_{i=1}^{K} P(y^*|x^*, M_i) \underbrace{\omega_i}_{P(M_i|D)}$

- Assumption 4: winner takes it all $P(M|D) = \delta(M - M^*)$
- Assumption 5: "Maximum A Posteriori" model. $M^* = \underbrace{\operatorname{argmax}_M P(M|D)}_{\text{training}} = \operatorname{argmax}_M \frac{P(D|M) P(M)}{P(D)} =$

$\Rightarrow \delta(M - M^*) = \begin{cases} 1 & \text{iff } M = M^* \\ 0 & \text{otherwise} \end{cases}$

↓

- A6: iid data

$= \prod_{i=1}^{N} P(|x_i, y_i|_M) \cdot P(M) = \operatorname{argmax}_M \prod_{i=1}^{N} P(y_i|x_i, M) \frac{P(x_i|M)}{P(x_i)} P(M) = \operatorname{argmax}_M P(M) \prod_{i=1}^{N} P(y_i|x_i, M) =$ ⑤

- Assumption 7: Exponential family: $P(y_i|x_i, M) = e^{-\lambda_1 f_1(y_i, x_i, M)} \rightarrow P(M) = e^{-\lambda_2 f_2(M)}$

⑤ $= \operatorname{argmax}_M e^{-\lambda_2 f_2(M)} \prod_{i=1}^{N} e^{-\lambda_1 f_1(y_i, x_i, M)} = \operatorname{argmax}_M \underbrace{-\lambda_2 f_2(M)}_{\|\theta_M\|} - \underbrace{\sum_{i=1}^{N} \lambda_1 f_1(y_i, x_i, M)}_{\text{loss funct.}} = \operatorname{argmin}_M \lambda_1 \sum_{i=1}^{N} \mathcal{L}(y_i, x_i, M) + \lambda_2 f(M)$

A7 ↓

regularization

Empirical risk minimization extended

Let's say now: $M^* = \underset{M}{\operatorname{argmax}}\ P(M|_D) \overset{iid}{=} \underset{M}{\operatorname{argmax}} \prod_i P(M|_{(x_i,y_i)}) =$ (missing)

= Pill 10: Linear Models =

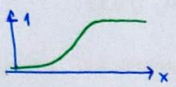
• Logistic regression:

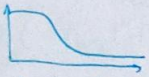
We start from $M^* = \underset{M}{\operatorname{argmax}}\ P(M|_D)$ as we want to maximize $\overset{iid}{\Rightarrow}$ $M^* = \underset{M}{\operatorname{argmax}} \prod_i P(M|_{D_{(i)}})$

$= \underset{M}{\operatorname{argmax}} \prod_i P(D_{(i)}|_M) * P(M) = \underset{M}{\operatorname{argmax}} \prod_i P(y_i|_{x_i,M}) P(x_i|_M) P(M) = \textcircled{*}$

"we can drop this model's prob." → why?

I want to model $P(y_i=1|_{x,M})$ as

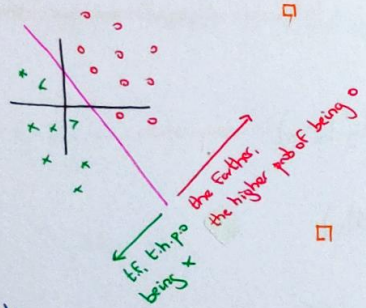




$1 - P(y_i=1|_{x,M}) = P(y_i=0|_{x,M})$

our data is {0,1} → remember "linear classifier"
↳ we will use a regression in order to be able to get the middle part.

The classifier we are building will be



what about these samples?
very high probs in zones we don't know anything about: ASSUMPTION

It makes sense: our model is too simple

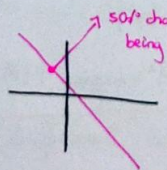
$\textcircled{*} = \underset{M}{\operatorname{argmax}} \prod_i P(y_i=1|_{x_i,M})^{y_i} P(y_i=0|_{x_i,M})^{(1-y_i)} = \underset{M}{\operatorname{argmax}} \prod_i P(y_i=1|_{x_i,M})^{y_i} P(y_i=0|_{x_i,M})^{(1-y_i)} =$

up to this point: it is true for any binary classifier.

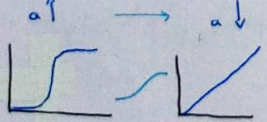
$= \underset{M}{\operatorname{argmax}} \prod_i P(y_i=1|_{x_i,M})^{y_i} \left|1 - P(y_i=1|_{x_i,M})\right|^{(1-y_i)} = \underset{M}{\operatorname{argmax}} \sum y_i \log P(y_i=1|_{x_i,M}) + (1-y_i) \log\left(1 - P(y_i=1|_{x_i,M})\right)$

skipping a few steps

$\Rightarrow P(y_i=1|_{x_i,M}) = \dfrac{1}{1+e^{-(a^T x+b)}}$



- b moves the boundaries "left to right"
- a defines how sharp the curve is

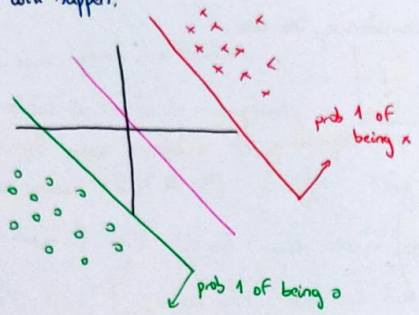


it defines how confident we are (less intermediate zone)

- we will be using the logit function for prediction: $p(x_i|y) = P(x_i) = \dfrac{1}{1+e^{-(a^T x+b)}}$ → decided thru $\hat{y} = \underset{y}{\operatorname{argmax}}\ P(y^*|_{x^*,D})$

optimization: gradient descent with its loss (in notebook)

If the sets are separable, this will happen:



as we are maximizing (the number of predictions with high confidence). Of course, this might be zoomed in/out.

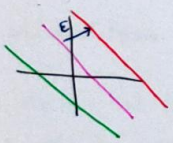
- First approach to modeling the separating hyperplane.

minimize 1

subject to $a^Tx+b>0$ if $y=1$

$a^Tx+b<0$ if $y=0$

Adding a threshold ε to



we may get $\varepsilon(a^Tx+b)>0 \Rightarrow \varepsilon(a^Tx+b) \geq \varepsilon \Rightarrow a^Tx+b \geq 1$

Therefore: minimize 1

subject $y_i(a^Tx_i+b) \geq 1$ with $y_i \in (-1,1)$ ← subject to.

$f(x_i)$

we want to maximize $d(p \rightarrow \Pi) = \frac{a^Tx+b}{\|a\|}$

[...]

- Soft-margin SVM: minimize $\|a\| + C \cdot (\sum \xi_i)$ s.t. $a^Tx_i+b \geq 1-\xi_i$ if $y_i='x'$ → labels.

(error)

$a^Tx_i+b \leq -1+\xi_i$ if $y_i='o'$

So now our problem is minimize $\|a\|_2 + C \cdot \sum \xi_i$

subject to $y_i(a^Tx_i+b) \geq 1-\xi_i \Rightarrow \xi_i \geq 1-y_i(a^Tx+b)$ $\quad \xi_i \geq 0$

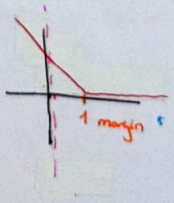
$\Downarrow$

$\xi_i \geq \max(0, 1-y_i(a^Tx_i+b))$

Turns out then that the solution should satisfy the equality

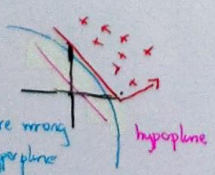
unconstrained problem solved by constrained methods. ←

minimize $\|a\|_2 + C \sum_i \max(0, 1-y_i(a^Tx_i+b))$ → Hinge Loss



* Only points that irritate me:

aka those points that are wrong or those close to hyperplane



separable problem ⇒ can C be 0? NO.

Because we are maximizing the distance without considering the error.

rewriting: minimize $\|a\|_2 + C\left(\sum_i \xi_i + \sum_j \xi_j\right)$ → separating the validation errors for neg and pos. value.

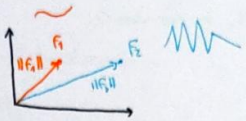
subject to $(a^T x_i + b) \geq 1 - \xi_i$ if $y_i = +1$

$(a^T x_i + b) \leq -1 + \xi_j$ if $y_i = -1$

$\|a\|_2 + C\left(\sum_i \xi_i + \sum_j \xi_j\right)$ to give more 'anger' to one of the errors (FP/FN)

= Kernel Methods = P.II 11 (16)

We will consider a space where each point is a function.
We will assume that the norm is related to the function complexity.



Functional Analysis → function spaces $(\mathbb{R} \mid f:\mathbb{R}\to\mathbb{R}),\ \mathcal{C}^1(D))$. They are vector spaces.
We are looking to bound the norm of $\|f\|$, for all f with similar behaviour.

- Hilbert Space: → $\mathcal{H} = (V, \langle\cdot,\cdot\rangle)$ also: norm concept is linked to Hilbert Spaces. $\|v\| = \sqrt{\langle v, v\rangle}$

Also, Cauchy-Schwartz: $|\langle v, v\rangle| = \|v\|\|v\|$

- A Hilbert Space $\mathcal{H}$ is a reproducing Kernel Hilbert space (RKHS)

If $\mathcal{H}$ is a RHKS: $\mathcal{F}_x[f] = f(x) = \langle K_x, f\rangle$ → notion of complexity.

Now we may introduce this notion into regularization: $f^* = \underset{f\in\mathcal{H}}{\operatorname{argmin}} \frac{1}{n}\sum_i \mathcal{L}(f(x_i), y_i) + \lambda\|f\|^2_{\mathcal{H}}$ → $\mathcal{L} = (f(x_i) - y_i)^2$

↓ complexity penalty

- Representer's Theorem: the solution of former problem has the following form: $f^*(\cdot) = \sum_i \alpha_i K(x_i, \cdot)$

(So far, all we know about Kernels is that they are functions) → 2 arg, as input. Represents similarity.

proof: $S = \text{span}\{K(x_1,\cdot), K(x_2,\cdot), \dots, K(x_n,\cdot)\}$ → basis of this space → $K(x_1,\cdot)$, $K(x_2,\cdot)$, $K(x_n,\cdot)$

① $F \perp S \Leftrightarrow f(x_i) = 0 \quad i = 1,\dots,N$

proof: $f(x) = \langle K(x,\cdot), f(\ )\rangle = \phi \quad i = 1,\dots,N$ → if it is orthogonal to the whole space, all its evaluations must be 0.

② $f(x_i) = f_S(x_i)$ ↑ a function can be decomposed into a hyperplane and its perpendicular.

proof: $f(x_i) = f_\perp(x_i) + f_S(x_i)$ → minimizing $f(x_i) \equiv \min f_S(x_i)$ ④
($f_\perp(x_i) = 0$)

③ $\|f\|_{\mathcal{H}} \geq \|f_S\|_{\mathcal{H}}$

proof: $\|f\|_{\mathcal{H}} \geq \|f_\perp\| + \|f_S\| + \langle f_\perp, f_S\rangle$

④ → $f^* = \underset{f\in\mathcal{H}}{\operatorname{argmin}} \frac{1}{n}\sum_i \mathcal{L}(f_S(x_i), y_i) + \lambda\|f_S\|^2_{\mathcal{H}}$ (② ↑)

$f(z) = \alpha_1 K(x_1, z) + \alpha_2 K(x_2, z) + \dots \longrightarrow f^*_S(z) = \sum_{i=1}^{N} \alpha_i K(x_i, z)$

↳ we passed from smth very abstract to smth in function of x_i.

(...)

- RHKS needs positive semidefinite Kernel functions.

What are Kernels?

- Gram Matrix: → blocks. → evaluate Kernel on whole dataset $G = (K(x_i, x_j))_{i,j}$

$\in \mathbb{R}^{N\times N}$

• Kernel (similarity matrix): $K(x_i, x_j) = 1$ iif $y_i = y_j$

first half of elements is class 1, the rest class 2.

It allows us to classify any kind of data in any form (not only vectors)

Gaussian: $K(x_i, x_j) = e^{\frac{-\|x_i - x_j\|^2}{\sigma^2}}$ → you can input anything as long as it is inside a distance (graph, e.g.)

* The idea comes from $y^T y$ and from the goal to form something close to the similarity matrix.

- Kernel are linear ($K = aK$; $K = K_1 + K_2$) and also, $K = K_1 \cdot K_2$

(Kernel trick)

$x_i^T x_j \xrightarrow{\text{Kernel trick}} K(x_i, x_j)$

math method → Kernelize

SVM → Kernel SVM

} it can be proved that $f(x)$ substituted by its representer and it will work as KT does.

= Decision Trees = (Pill 12: Ensembles)

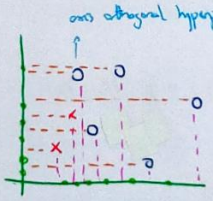
In decision trees, the top nodes encode the most information.

↳ the deeper you go, the more 'specific' the nodes are.

- What's the best decision tree? Many ways to answer this: 1. Less amount of rules (complexity)

In. Nodes = [ ] → contains feature + threshold (a value of a feature is tested)

1. N ← retrieve first (pop)
2. data ← N
3. c ← find splitting point.



• = possible splitting points all defined by a threshold.

↳ choose it (by error minimization, e.g.)

4. After taking a step, the next one to be chosen will be the one that maximizes Gain = Parent − Avg(Children)

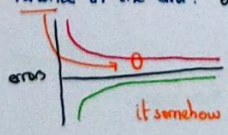
error — errors (2)

5. Add 2 more nodes to the list.
6. Repeat.

Out. Decision Tree (Classification)

• Avoiding overfitting: ensemble learning → reduces the variance in the error. by calculating the average of randoms.

↓ train a set of models aggregate



it somehow tells us how bad generalization is going to be.

notion: if one classifier is unlucky at classifying a sample → maybe the rest is luckier.

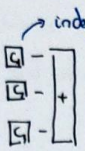
it makes little sense to have 1 model for huge datasets → submodels.

divide + conquer (have many classifiers)

it makes no sense either to have one classifier in cases of data fusion (heterogenuous sources)

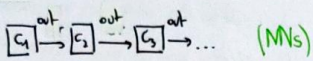
- diversity: "as how you choose 2nd opinions from people with different backgrounds"

Ensemble through: 1) Bagging, (Bootstrapping aggregation)



2) Boosting relies on waste — they all depend on each other. link through: residual / statistical resampling — gradient boosting, adoptive boosting

3) Stacking output → input



(MVs)

↳ should be vertical

① ⊕ aggregation done through majority voting

③ Stage wise approximation: $H(x_i) = \sum_{t=1}^{T} \alpha_t h_t(x_i)$; with classic LSE ⇒ minimize $\frac{1}{N}\sum_{i=1}^{N}(y_i - H(x_i))^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \sum_{t=1}^{T}\alpha_t h_t(x_i))$

linear comb. of models

we will minimize model by model, aka, at time T; minimize $\alpha_T h_T$ $\frac{1}{N}\sum_{i=1}^{N}(y_i - \sum_{t=1}^{T-1}\alpha_t h_t(x_i) - \alpha_T h_T(x_i))^2$

already fixed

The new added model T is trying to decrease the residue of the rest.
↳ Specialized in correcting the rest of models.

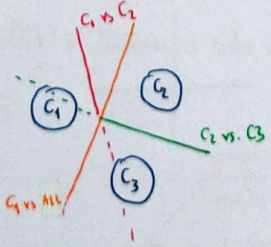
- reductionist frameworks for multi-class problems

	H MClass	Naturally? (scalation ability)
SVM	N	
NN	Y	'NO' { Kinda. Last layer changes dim. / Bad. Linear (you need to model one prob per class)
Probabilistic	Y	
Dec. Tree	Y	Yes
K-NN	Y	Yes ↳ we only care about the labels of close samples to the query samples.
Log Regression	N	

'1 against all': divide-n-conquer in multiclass

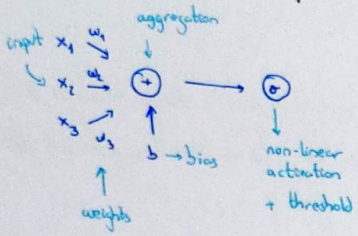
⇓

1 vs 1 but aggregated.



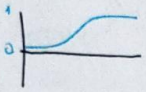
'1 against 1': consider all pairs of classes and then aggregate.

= Pill 13b: A story of deep learning =



If threshold is exceeded → neuron 'fires'

- sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$



(Fully connected layers of neurons)

- To define a NN: 1) Architecture (Hypothesis space)
  2) Loss function
  3) Optimization algorithm

> Number problem: 10 class problem

Remember from DL: CNN (c= conv.) are a subtype of FCNN. = dense in Keras.

- architecture definition: 1) Sequential: limited to stacking layers.
  2) Graph RESNet (the input of a layer is not always the output of the latter)

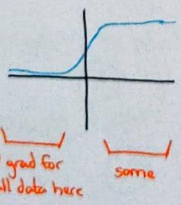
We want a probability of the input being a specific number ⇒ we need normalization: SOFTMAX → at the end.
($p_i \in [0,1]$ $\sum_i p_i = 1$)

- it is called a softmax layer because it emphasizes big values and desemphasizes small values.
  It is not needed for the NN to work, but it is handy for the exponentials in the loss function. → easier computation.

- cross-entropy loss function.

We expect more layers to lead to better results. Spoiler: it doesn't. At some cases it gets even worse. / overfitting / vanishing gradients.

- Vanishing gradients: due to sigmoid



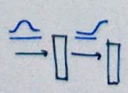
, we get dead neurons that do nothing in learning process:

ReLU: fastest 'elU'

many elUs: SeLU, LeaKy ReLU, ...
recomendación del chef

Can we add now 10^{300} layers? No.

- Internal covariate shift: The distribution moves through each layer:



We need Batch Normalization (normalize data according to stats in the batch) → 0 mean ≡ centering the data

'approx' → we only using a batch.

Will it work to add 30 sigmoid layers with batch normalization? YES. Also, 100% accuracy.

No training and test so far ⟹ (add split) shit results. And we only changed the data distribution. → overfitting / few samples → data augmentation *
\ high complexity

* - change the split ratio
- transformation (img, flipping)
- random noise → dropout layers. some values are randomly set to 0.
↳ only in images is good to drop some pixels due to high correlation between neighbours.

• Layers in deep learning is all about Knowledge representation. The last layer is a linear classifier, the rest of layers transform for the last layer.

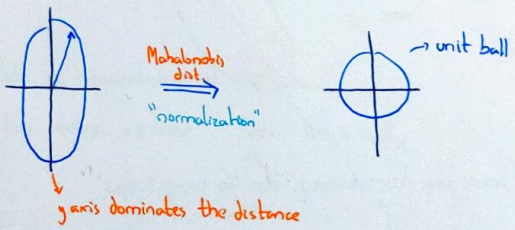
↓ can be changed for a real classifier. (random forest)

increase in accuracy → the transformation was good.

(...) → Autoencoders, whole pill.

= Pill-12 Gausian discrimination and processes =

We can compensate axes with large variance.



We will suppose we have quadratic forms ⟹ all eigvals ≥ 0

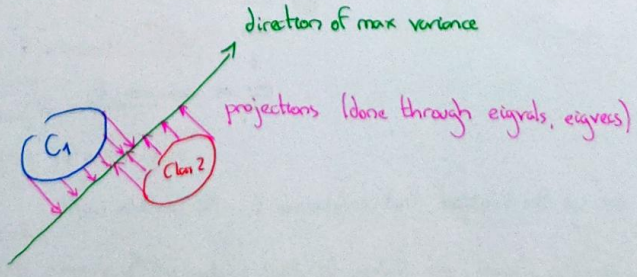
PCA
The eigenvalues will appear in order of maximum variance.

⟹ let's represent all the dataset as a normal distribution (assumption) and get the direction with the most variation

we want to use a linear transformation to select d' dimensions onto which project our d-dimensional dataset

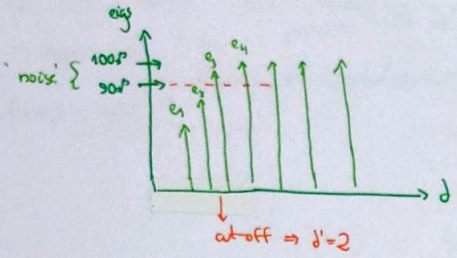
with d'<<d

d' is preset or depending on 'energy' (keep a % of variance of data)



- how to fix d'? As a hyperparameter, in relation to the ratio dim/noise.

Imagine we want to recover 90% of the energy:



note: remember to center by substracting the mean

$\Sigma = \Lambda B$ → d×d (eigvals, eigvecs)

↓ selection evecs of max eigvals ⇒

B' (d'×d')

$x \in \mathbb{R}^{35}$

$B' \in \mathbb{R}^{30\times 5}$

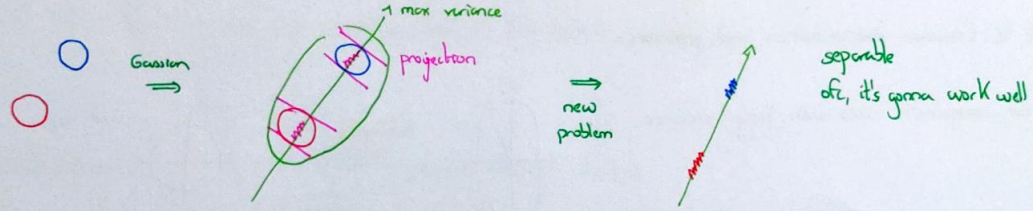
$x' \in \mathbb{R}^5$

$\Big\}$ $\boxed{xB' = x'}$

it results that working with the selection is the same as working with the whole dataset.

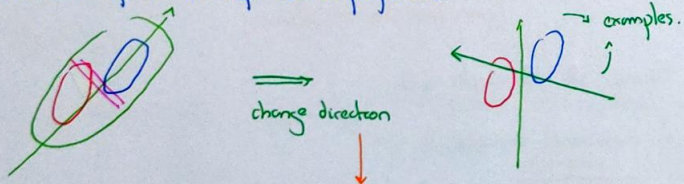
note: the cut-off for $d'=5$ can be seen with the graph in cell 37

Problem 1:



A problem can turn from sep. to non-sep. due to projections.

Problem 2:



↓ new problem: Linear discriminating analysis.

- LDA: our new B is gonna be: $B = \frac{S_b}{S_w}$ → between classes scatter matrix: difference between the mean of both classes (we want to max. dist. line from center of both classes)
  → within " " " " : distance from class instances to class center (minimum → compression)

$\frac{\max}{\min}$ ⇒ maximize B

LDA ⇒ line in problem 2 of this color.

note: we get the direction, that maximizes B. All parallels work.

How many clusters do we need to compute?

We may get less or more clusters than data sources (e.g.)

There are automatic ways to determine the number of clusters (we will see that)

We can also find correct solutions that do not fit: • we see '2' clusters and we know there are 3 the solution may split the 'wrong' cluster.

• divide horizontally when it's vertical, or viceversa.

- This happens because K-means assumes all directions weigh the same (Gaussian)

$r_{ik} = \begin{cases} 1 \\ 0 \end{cases}$ is too black or white ⟹ grays. make all elements be involved in responsibility: all elements will eventually pull if there are several together.

↓ change it to something that relates distance and similarity.

SOFT K-MEANS $e^{-\gamma d(x_i, c_k)}$ → actually a Gaussian*

weight → it represents the 'zoom effect' previously mentioned.

- Imagine we want way too many clusters. Depending on the zoom, centroids will eventually collapse.
Ĉ use this as a toy and discover number of clusters (high computational needs)

• Mixture of Gaussians: learn $\beta = \frac{1}{\sigma^2}$

minimize $\sum_i \sum_k e^{-\beta \|x_i - \mu_k\|_2^2} \|x_i - c_k\|_2^2$

maximize r $e^{-(x-\mu_k)^T \Sigma^{-1} (x-\mu_k)}$ ↓ covariance matrix

we can use Mahalanobis distance (not all directions weigh the same)

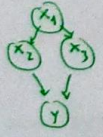
• Maximum likelihood

Letting each Gaussian have different covariance matrices allows us to have concentrical clusters.

= Pill 14b: Unsupervised learning + Clustering =

• PGM: ⟶ $P(y, X)$ ⟶ $P(y|X)$

joint



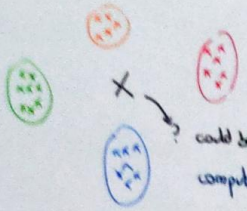
$$P(y, x_1, x_2, x_3, x_4) = P(y|x_1, x_2, x_3, x_4) = P(y|x_2, x_3) P(x_1|x_2 x_3) P(x_2|x_3) P(x_3)$$

most complex term

no edge (independency)

$P(x_1)$ $P(x_2)$

Assume all data comes from distributions:



could be detected as an outlier

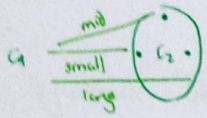
compute P(x in each dist) with mixture of Gaussians.

- Clustering (K-means): novel << outliers
  ↓ in terms of 'unusual'

Control countries may make relevant data less relevant if they are too many: 1) one may be an outlier
2) they could be in a cluster of only control countries.

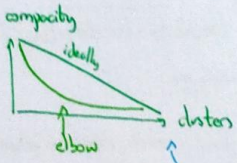
solutions: - delete from dataset and reinsert it on top. (after clustering)

It might be interesting to locate where in the cluster an elements is (checking the distance to other clusters' centroids)



- K-means depends on the K number of clusters. How to get K? Elbow/Knee technique (notion of compactness is related to the average distance to the centroids)

corner case: all entries in a point → 1 cluster



corner case: everything is isolated.

It only gives an idea of a number of clusters that may be good.

remember: don't use K-means (soft K-means, etc.)

- Hierarchical clustering: dividir maximizando distancia uniendo clusteres más cercano — #inicial de cluster: 1 / #entries } costful

- DBSCAN (density-based S.C.A.N.) → gets fancy shaped clusters.
  It uses local measure, rather than global ones.
  It can detect points in high density areas, low density and outliers.
  (high density areas: core; low density: boundaries)

Select a point, consider a ε-neighbourhood, share label with all ε-neighbour. Select a point from the neighbourhood and repeat.

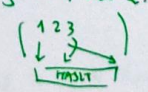
- ε and minpoints. (...)

- Spectral clustering:
  - eval/vecs. of a similarity matrix

= Pill 2: Dirty Data =

- Split columns `df[cols] = df[col].str.split(sep, expand=True)`
  - cols: list of new column names
  - col: str: colName to split
  - sep: str.
- Drop useless columns `df.drop`
- Replace all missing values for NaN (*will be extended) `df = df.replace( ... , np.Nan )`
  - opt: restrict by columns
- Delete waste from numerical columns: `df.replace(regEx, '', regex=True)`
- Handling missing data: inputing
  - ↳ add pair wise deletion to notes: if we are analysing a subset, only delete samples from that subset.
  - ↳ average + dummy variable indicating if value was originally missing.
- More info ⇒ more dimensionality ⇏ Better results
  - Especially in ML
- Approaches to outliers:
  - Statistical description: assume data follows a distribution. Those samples that do not → outliers.
  - Geometric: drawing in notes
  - Distance to neighbours: KNN may be used.
  - Percentile funct: $\hat{x} = \frac{x - pc(x,\theta)}{pc(x,100-\theta) - pc(x,\theta)}$ — θ: percentile we consider as outliers (10-/-)
    - pc: np.percentile

Questions:

1) In general it is good idea to drop samples w/ missing data (only not when dataset is small)
2) You can capture non-linear behaviour with linear regressors using dataset extension.
3) Only use standarization with few outliers.
4) Normalization is useful to compare different ranges.
5) Categorical variables ⇒ One HOT (don't do green=1, blue=2...)
6) Hashing trick does not maintain distance 

7) Hashing is ~ uniform.
8) In distance based classification algorithms, it is a bad idea to add random features.

= Pill 3: Performance =

- Cross-validation: all points will be used for testing. (LOO with n it).
- K-fold cross validation es L"Many"O ≈ LKO. with K it.
- Unbalanced datasets with unbalance between positives and negatives.

- Confusion Matrix:

Prediction \ Truth	Positive	Negative	
Positive	TP	FP	→ precision: $TP/(TP+FP)$
Negative	FN	TN	→ negative predicted value: $TN/(TN+FN)$
	↓ sensitivity/recall $TP/(TP+FN)$	↓ specifity $TN/(TN+FP)$	

accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$

estructura: T de la fila/columna / suma de la fila/columna

Si churn = +. Correctly predicted churned ppl? Recall

From the ppl that predictedly churn, how many actually do? Precision.

- F1 Score = $2\frac{\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}} = \frac{2TP}{TP+FN+TP+FP} = \frac{TP}{TP+\frac{1}{2}(FP+FN)}$

- Other alternatives: undersample majority class
  oversample minority → interpolator.
  class weights.
  change performance metric.
  split majority in subclasses → train a classifier with each. → aggregation.

= Pill 4: Supervised ML =

- Hypothesis space/model class: Family of mathematical models to consider. → Liner: $h(w_0,w_1) = w_1x + w_0$
- Problem model: optimization, minimization of err function → minimize $L(y_i,\hat{y}_i) = \frac{1}{N}\sum_{i=0}^{N}(y_i-\hat{y}_i)^2 = \frac{1}{N}\sum_{i=0}^{N}(y_i-(w_1x_i+w_0))^2$
- Learning algorithm: optimization method or algorith that fits the model to the data. → Gradient descent.
  ↳ target: simplicity/speed/interpretability ↑↑ / accuracy ↓ (trade-off de todos por la última)

Linear regression model: $\tilde{x} = \begin{pmatrix}1\\x\end{pmatrix} \Rightarrow \hat{y} = \tilde{x}^T w$ → minimize $\underbrace{\frac{1}{N}(y-\tilde{X}^Tw)^T(y-\tilde{X}^Tw)}_{LSE}$ with $\nabla_w L = -\frac{2}{N}\tilde{X}(y-\tilde{X}^Tw)$

= Pill 5: Feasibility =

Is learning possible if f is unknown? No, we can't know if f gives a certain value in unseen data.
But yes, through probability → red/green balls take-out might give more green while there are more red in bin

⇒ All combinations are possible, but not equally probable. We hope the sample to be similar to the total.

- r = E_{in} 'in sample error' → # times hypothesis getting it wrong.
- μ = E_{out} 'out of sample error' → expected

  ↓

  Learning is assuming E_{in} indicates E_{out} and finding the hypothesis with least E_{in}. ($E_{out} \searrow 0$ through $E_{in} \approx E_{out}$)

- Hoeffding taking into account worst case scenario among hypothesis as in terms of complexity of hyp. space.

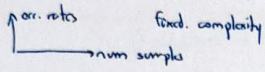
We move to Probabilities as the target may not always be a function (sea, sky detection ← 2 vals for y|x)

- VP-dim: measure of complexity

Extras on questions: 1) Overfitting when $E_{in}\downarrow$ $E_{out}\uparrow$ ($E_{in}\neq E_{out}$)
2) Overfitting depends on complexity and samples (complexity ↓ and samples ↑ to avoid)

= Pill 6: Overfitting =

- We have to match the data complexity, not the underlying true model complexity (always start from data)
- Increasing complexity to reduce bias and make test and train error rates converge closer generally causes worse error rates.



bias seen in

given for a fixed complexity.

= Pill 7: Stochastic Subgradient Methods =

They satisfy that at x for all z

$$f(z) \geq f(x) + g^T(z-x) \Rightarrow \text{hyperplane with normal } (g,-1) \text{ supports } f \text{ at } (x,f(x))$$

- f convex and differentiable ⇒ its gradient is subgradient.
- The subgradient may exist even if f is not differentiable.
- x^* is a minimizer of f, convex ⇔ f subdifferentiable at x^* (there is at least 1 subgradient) and $0 \in \partial f(x^*)$ (set of subgradients) ⇒ $f(z) \geq f(x)$.
  - If f is differentiable at x^*, the equation is reduced to $\nabla f(x^*) = 0$.
- They are useful to solve problems like minimize $f_0(x)$ subject to $f_i(x) \leq 0$ $i=1,\dots,m$ with $x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$

where $g^{(k)} = \begin{cases} \partial f_0(x^{(k)}) & \text{if } f_i(x^{(k)}) \leq 0 \quad i=1,\dots,m \\ \partial f_j(x^{(k)}) & \text{if } f_j(x^{(k)}) > 0 \end{cases}$

⇒ There is a version with noise.

- momentum: $x^{(k+1)} = x^{(k)} + m^{(k+1)}$ with $m^{(k+1)} = \underset{\in[0,0.9]}{\gamma} m^{(k)} - \alpha_k g^{(k)}$
- autograd: $x^{(k+1)} = x^{(k)} - \alpha_k D^{-1} g^{(k)}$ with $D = \sqrt{\sum_{i=1}^{k} \text{diag}(g^{(i)})}$
  also could be $D^{(k+1)} = D^{(k)} + \text{diag}(g^{(k)})^2 \rightarrow x^{(k+1)} = x^{(k)} - \alpha_k [D+\varepsilon]^{-1/2} g^{(k)}$
- adam: adaptative momentum (combination of both) → I wouldn't even bother...

= Pill 1: First Steps =

- Machine Learning: - Improve SW performance based on previous experience.
  - Methods that detect patterns in data to predict future data or take decisions under uncertainty.

• Types of ML
- Predictive / supervised: training labelled (with target) dataset ⇒ regression, classification
- Descriptive / unsupervised: given data set find information about structure ⇒ density, clustering, dim reduction, manifolds.
- Reinforcement

• Curse of dimensionality: too complex

④ Quiz questions:
1) Subgradient generaliza gradiente para no diferenciales.
2) Subgradients don't always decrease.
3) Allow approximations with ~~loss~~. noise.
4) Subgradient = Gradient si convexa (+ differentiable?)
5) Adagrad and momentum don't give better solutions, just faster convergency.
6) Advantadge of adagram over momentum: solution adaptation coordinate-wise.

= Pill 8: Regularization - Combatting overfitting =
1) Keeping overfitting at check: regularization, build an ensemble, validation set
2) Regularization → controls complexity of model
3) Correct ways of adding regularization term to objective function: - control # coeff. ≠ 0 - control max value coeff.
- control size of coeff. values.
4) Regularization techniques: adding norm to loss function, adding noise to input samples, early stopping.

= Pill 10: Linear Models = • Least Squares regression: $h(x) = a^T x + b$; $\min_{a,b} \sum_i (y_i - h(x_i; a,b))^2$

• Logistic regression: linear classifier based on maximum likelihood to model classification boundary and certainty of the fit. Fits logistic function to data.

$$p(x_i|y) = \frac{1}{1 + e^{-(a^T x + b)}}$$

• Support Vector Machines (SVM): discriminative learning

↓

focuses on the decision boundary. works with labeled data (the more the better)

discriminative vs. generative → prob. model of each class; unlabeled; boundary = where a model is more likely.

We assume a function model for the boundary: linear (classical)

Other discriminative linear models: SVM, perceptron, logistic classifier...

It finds the boundary with maximum distance/margin to both classes.

- Support vectors: points right in the margins (remember OPT lab) If they disappear, boundaries change.

critical subset of points.

It implicitly models noise through boundaries. We expect it to be robust against tiny perturbations.

- Maximum margin classifier → unique solution in separable case.

- Distance to a hyperplane: $d(x, \pi) = \frac{a^T x + b}{\|a\|_2}$ goal: $a^T x + b \geq 1$ or $a^T x + b \leq -1$

$\max \frac{2}{\|a\|_2}$ s.t. $y_i(a^T x_i + b) \geq 1$ $\equiv$ $\min \frac{2}{\|a\|_2}$ s.t. ... ↑ al revés

↓

closest different classes points are $2 \cdot \frac{1}{\|a\|_2}$ apart

• Soft-Margin SVM (non-separable): introduce a new set of variables that measure amount of violation in i-th constraint. (ξ_i)

We want $\xi_i \searrow 0$.

→ $\min \left(\frac{2}{\|a\|_2} \right) + C \sum_i \xi_i$ trade-off between margin and misclassification rate.

s.t. $y_i(a^T x_i + b) \geq 1 - \xi_i$

$\xi_i \geq 0$

Quizz:
- Logistic regression commonly used for regression ✗ → classification.
- " " probability assignment to each data point ✓
- Concept of margin related to distance (samples, boundary) ✓
- SVM maximizes margins ✓
- Non-separable SVM's C balances robustness and training error ✓
- " " " " $= 0$ ≡ Hard margin SVM ✗

= Pill 11: Kernels =

- Linear evaluation functional over the Hilbert space of functions $\mathcal{H}$ $\quad \mathcal{F}_x : \mathcal{H} \longrightarrow \mathbb{R}$, $f \longmapsto \mathcal{F}_x[f] = f(x)$
- RKHS: if evaluation functionals are bounded, $\exists M \,/\, |\mathcal{F}_x[f]| = |f(x)| \leq M \|f\|_{\mathcal{H}} \quad \forall f \in \mathcal{H}$.
  - By the Riesz representation theorem, for each $x \in X$, $\exists$ function $K_x \in \mathcal{H}$ (called representer) that satisfies:

$$\mathcal{F}_x[f] = f(x) = \langle K_x, f \rangle = \langle K(x, \cdot), f(\cdot) \rangle$$

Examples of positive semidefinite Kernel function K:
- linear: $K(x_i, x_j) = x_i^T x_j$
- polynomial: $K(x_i, x_j) = (x_i^T x_j + 1)^d$
- gaussian: $K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma^2}\right)$

- Gram matrix: shows the degree of shared information among training samples.
  - interpretation:
    1) bad Kernel: mostly diagonal → most points are orthogonal to each other, no clusters/structure.
    2) good Kernel: structure + clusters.
    3) ideal: $K = yy^T$ 

- Kernels: tips
  - use your fav. distance $d(x_i, x_j) \Rightarrow K(x_i, x_j) = e^{-d(x_i, x_j)}$
  - " " non-linear transform: $K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$

- Kernel trick: regularized Least Squares but using a mapping $\phi(x): \mathbb{R}^d \to \mathbb{R}^{d'}$ instead of the original input space.

$$\Rightarrow \alpha^* = \underset{\alpha}{\operatorname{argmin}} \frac{1}{2} \sum_i (y_i - \alpha_i \phi(x_i))^2 + \frac{\lambda}{2} \|\alpha\|^2$$

$$\Rightarrow \text{model is } \hat{y} = \Phi w = \frac{1}{\lambda} \Phi \Phi^T \alpha$$

↳ same solution as Kernel method with $K = \Phi \Phi^T$

pros:
- using Kernels can be seen as mapping data in an arbitrarily high dimensional space spanned by $\bar{\Phi}(x)$
- using Kernels allows a compact implicit description of the space generated by $\bar{\phi}(x)$ by means of the Gram matrix $K(x, \cdot) = \bar{\phi}(x) \bar{\phi}(\cdot)$

- $\dim(\phi(x)) = \infty$ for a gaussian kernel.

The Kernel trick replaces any $\langle x_i, x_j \rangle$ by a kernel $\langle x_i, x_j \rangle = K(x_i, x_j)$

Also, in RKHS, $f(x) = \langle K(x_i, \cdot), f(\cdot) \rangle$ and we can replace $f(x) = \sum_i \alpha_i K(x, x_i) \quad \forall x_i \in D_{train}$

Online Kernel perceptron: perceptron: linear model that solves $\min 1$ s.t. $y_i(w^T x_i) > 0$

algorithm. 1) select random sample pair (x_i, y_i)

2) If $y_i \sum_{k=1}^{l} a_k \langle x_k, x_i \rangle < 0$: $a_i \leftarrow a_i + y_i$

$\Rightarrow$ model: $f(x) = \sum_{k=1}^{l} a_k \langle x_k, x \rangle$

$\Rightarrow$ generalized alg: 2) turned into if $y_i f(x_i) < 0$: $a_i \leftarrow a_i + y_i$ with model $f(x)$ in a RKHS $f(x) = K(X, x)^T \alpha$, thus, if $y_i \sum_{j=1}^{N} K(x_j, x_i) a_j < 0$: $a_i \leftarrow a_i + y_i$

Online Kernel SVM: $\min \|w\|_2^2 + C \sum_i (1 - y_i \underbrace{w^T x_i}_{f(x_i)})$

$a^T K(x_i, X)$

Quizz:
- Kernels allow to use linear models in non-linear problems with success. ✓
- Kernel trick = replace inner product by a Kernel evaluation. ✓
- Represent Theorem: minimal solution of a regularized loss objective function in RKHS is a linear combination of Kernel evaluations in the dataset. ✓
- $K(x, y) = x^T y \equiv$ linear Kernel ✓
- Increasing the sigma parameter in a RBF Kernel reduces the non-linear capacity of modelling boundaries. ✓
- A good Kernel has a diagonal matrix ✗

= Pill 12: Ensembles =

- Decision Tree: - easy to interpret.
  - All data might be boxed.
  - There can be more than one solution that feeds the data.
- Splitting criteria: 1) Misclassification error
  2) Gini index
  3) Cross-entropy / Information gain / Mutual information } maximize 'role purity'
- stop overfitting: early stopping or post-prune

- **Diversity**: it is needed because we need the errors to happen in different samples.
  - how to get it:
    1) Different training sets.
    2) Different training parameters.
    3) Combining different architectures (SVM, decision trees)
    4) Training on different features random subspaces or projections.

- **Examples of ensemble models:**

1) Bootstrapping aggregation: resampling the training set with replacement. → bagging
   one classifier per dataset copy + majority voting.

2) Random forest: randomization on the feature selected for building each tree in the ensemble.

3) Incremental learning boosting

gradient boosting:

our model is $H_t(x) = \sum_{t=1}^{T} a_t h_t(x)$

- t: time
- h_t: models; once they are added they remain constant.

Increment ⟹ select h_{t+1} and optimize a_{t+1} → H_{t+1}, $\mathcal{L}_{t+1}$

$$\mathcal{L}_t(x) = \sum_{i=1}^{N} \underbrace{(y_i - H_t(x_i))^2}_{\text{LSE}}$$

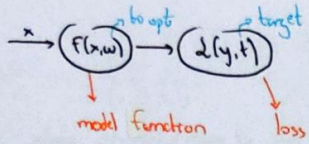
In $\mathcal{L}_{t+1} = \sum_{i=1}^{N} (\underbrace{y_i - H_t(x_i)}_{=r_t(x_i)} - h_{t+1}(x_i)\, a_{t+1})$, $r_t(x_i)$ represents error or residue of the model at time t on x_i.

adaboosting: stage-wise approach also but with $\mathcal{L}(x,y) = \sum_i e^{-y_i H(x_i)}$

Quizz:
- Decision trees use axis orthogonal hyperplanes ✓
- Naturally handles multi-class: Decision Tree, KNN, NN, Prob. Classifier, SVM
  ✓ ✓ ✗ ✗ ✗
- Decision trees can fit any dataset ✓
- Ensemble learning = field of ML studying stacking and aggregation of models. ✓
- Diversity is key to ensembles. It means that the output of models should be as uncorrelated as possible disregarding accuracy ✗
- Combining the same model trained in different unrelated datasets is an ensemble producing technique. ✗
- Boosting is based on the concept of residual: reducing the difference between true label and our current approximation ✓
- Error correcting otput is an ensemble technique that extends methods to multiclass ✓
- 1-vs-1 and winner-takes-all convert multiclass problems to binary problems. ✗
- DL is an ensemble technique ✓

= Pill 13a (18): NNs and DL =

- Model class:

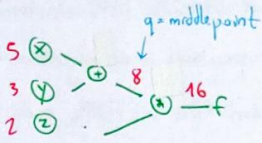


- Loss function

- Learning algorithm: $\min_w \mathcal{L}(f(x,w), t)$

- The flow of the chain rule: basis of backpropagation algorithm and automatic differentiation.
  used in DL training

- computational graph: graphical representation of operations. $f(x,y,z) = (x+y)\cdot z$



Partial derivative of each node respect to its inputs

$f = q * z$

$\frac{\partial f}{\partial f} = 1 \quad \frac{\partial f}{\partial z} = q \quad \frac{\partial f}{\partial q} = z$

$q = x + y$

$\frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1$

Gradient respect to any input: (chain rule)

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\cdot\frac{\partial q}{\partial y} = z \cdot 1$$

$$\frac{\partial f}{\partial z} = q$$

main ide behind automatic differentiation.

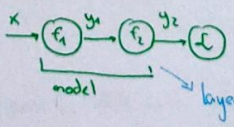
differentials can be computed as we parse the graph.

Given the following chain $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial v}\frac{\partial v}{\partial x}$

Forward: $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\left(\frac{\partial q}{\partial v}\frac{\partial v}{\partial x}\right)$ useful for vectorial functions

Backward: $\frac{\partial f}{\partial x} = \left(\frac{\partial f}{\partial q}\frac{\partial q}{\partial v}\right)\frac{\partial v}{\partial x}$ good for gradients

- Deep models. composition or stacking of functions. $y_2 = f_2(f_1(x))$



Use of standard chain rule:

$$\frac{\partial \mathcal{L}}{\partial \bar{w}_2} = \frac{\partial \mathcal{L}}{\partial y_2}\frac{\partial y_2}{\partial \bar{w}_2} \quad ; \quad \frac{\partial \mathcal{L}}{\partial \bar{w}_1} = \frac{\partial \mathcal{L}}{\partial y_2}\frac{\partial y_2}{\partial \bar{y}_1}\frac{\partial \bar{y}_1}{\partial \bar{w}_1}$$

$$\frac{\partial \mathcal{L}}{\partial \bar{w}_1} = \frac{\partial \mathcal{L}}{\partial \bar{y}_1}\frac{\partial \bar{y}_1}{\partial \bar{w}_1}$$

For N layers: $\frac{\partial \mathcal{L}}{\partial \bar{w}_m} = \frac{\partial \mathcal{L}}{\partial \bar{y}_N}\prod_{i=m}^{N-1}\frac{\partial \bar{y}_{i+1}}{\partial \bar{y}_i}\cdot\frac{\partial \bar{y}_m}{\partial \bar{w}_m}$

both gradients respect to parameters and layer input are needed.

= Pill 13: a story of DL =

- A layer defines its output through neurons. Layer with 100 units → 100-dimensional vector.
- Inner layers use non-linearity. Outer does not.
- Batch. Approximate true loss by a noisy approximation by considering a subset of data at each parameter update.
- Softmax: $\sigma(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$

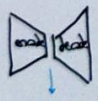
Quiz:

- Automatic differentiation can be regarded as a programatical way of differentiating ✓
- 'Flavors of automatic differentiation': back- and forward ✓
- Backpropagation is an algorithm that sequencially uses chain rule to update different layers in a compositional model ✓
- In standard NNs or FCNNs, each layers is composed of two ops: - non-linear
  - weights x output previous layer. ✓

= Manifold Learning =

- Manifold learning: non-linear dimensionality reduction (S-curve) basic hypothesis: data lies in a manifold of intrinsic dimensionality smaller than the original space.
- Multi Dimensional Scaling: constructs constellation of points using info on the distances between samples.
  - Isomap (extension): performs MDS in the geodesic space of the non-linear manifold (not in original space)
    shortest distances along the curved surface represented by geodesic distances.
- Locally Linear Embedding: approximates manifold around each sample and finds the best low-dim representation to reconstruct that sample. weights of the embedding vector of a sample are computed from its neighbors.
- Stochastic Neighbor Embedding: (SNE) changes the notion of distance to that of conditional probabilities. (of x_i selecting x_j as neighbors under a Gaussian centered in that sample) R_{ij}
  T-SNE uses a T-distribution.

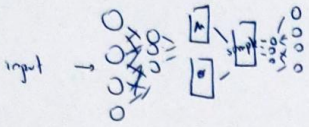
= Unsupervised learning in Keras =

- Autoencoders: 

  only the relevant info is kept, very comprised
  physical bottleneck
  - the middle representation can be used for feature extraction
  - logical bottleneck:
- Variational autoencoders: although autoencoders generate samples, the space of these samples seems to be composed of bubbles of information with bad interpolation properties among them. → VAE solution

x (input) —encoding→ latent dist. $p(z|x)$ —sampling→ sampled latent representation $z \sim p(z|x)$ —decoding→ input reconstruction. $d(z)$

It needs Kullback-Leibler divergence loss function. Latent-space ~ Gaussian. 2 vectors should be returned



based on Variational Inference

- GANs: Generative Adversarial Networks.

Clever trick for guiding a network into mimmicking sampled data. Two step process:

1) The generator: a network plays this role. Input comes from Normal distribution and the output is expected to be the type of data we desire. Same as decoder
2) The discriminator: (trick) it is trained with data from the true source and from the generator (mixed). Needs to learn to tell them apart.

Quizz:

- Isomap preserves Euclidean distance ✓
- Preserving Euclidean distance in manifold learning is generally a bad idea ✓
- LLE approximates the neighbourhood of a sample with a hyperplane ✓
- LLE in test: data x projection matrix (in order to apply) ✗
- Manifold → space where Euclidean intuition holds ✓
- Autoencoders need to memorize training data points ✗
- DL learns a good feature set that fits the problem at hand ✓
- Key for a good DL classifier: powerful classifier on last layer ✓
- Autoencoders need a physical or logical bottleneck in order to be effective ✓
- Autoencodes try to learn the identity. → unsupervised ✓
- To get a representation in DL: cut at one layer and get the result ✓
- Representations can be transferred to other domains ✓

= Pill 12: Gaussian discrimination and processes =

- Quadratic forms: all eigvals ≠0 → ellipsoid, hyperboloid, imaginary ellipsoid.
  all >0 all ≠0 all <0
- Gaussian distribution analysis model: assume each class comes from a Gaussian dist.

Quizz:

- The classification boundary in Gaussian distribution is the same as KNN with Mahalanobis distance ✓
- PCA uses info of the classes in a supervised problem ✗
- Eigvect with longest eigval of the covariance matrix points to the direction with max variance ✓

= P.11 14 : Mixture models =

- Clusters put together similar points and separate different ones.