

# Machine Learning

## Index

Machine Learning.....	1
Index.....	1
Microsoft ML.NET.....	6
Mistake-Bounded problem: Classifying emails .....	6
Decision trees.....	9
Generalization .....	20
Introduction to PAC learning with decision trees .....	20
Basic technique: model complexity (similar to cross-validation).....	23
Basic technique: minimum description length principle.....	24
PAC model of learning.....	25
The marble game - probabilities .....	29
Bad event scenario.....	31
PAC learning .....	34
Introduction .....	34
Halfspaces .....	39
Cross-Validation .....	41
Introduction .....	41
Markov's inequality.....	42
Chernoff bound .....	43
Hold-out set.....	46
Example with decision trees .....	47
Perceptron learning.....	48
Introduction .....	48
Complicated algorithm for learning halfspace.....	49
Perceptron algorithm: mistake-bounded model .....	50
Proving perceptron algorithm claims.....	55
Polynomial threshold functions (generic halfspace functions).....	56
Kernel perceptron .....	58
Linear regression.....	63
Introduction .....	63
Regression with multiple variables .....	67
Maximum likelihood for regression .....	69
Interpretations for coefficients in linear regression .....	72

Gradient descent.....	73
Introduction .....	73
Gradient descent for complex function .....	77
Defining gradient descent problem .....	80
Applying gradient descent to linear regression .....	81
Stochastic gradient descent .....	83
Choosing “ $\eta$ ” (step-size).....	84
Boosting.....	86
Introduction .....	86
Attempt 1 to improve accuracy .....	90
Attempt 2 to improve accuracy .....	91
Adaboost .....	93
Hedge (generic algorithm for Adaboost) .....	99
Logistic regression.....	102
Introduction .....	102
Loss functions.....	106
Optimization problema associated with logistic loss.....	108
Additional considerations .....	113
PCA – Dimensional reduction.....	116
Introduction .....	116
Applications.....	122
Finding v1, v2, ... vk .....	125
Spectral theorem.....	126
Recap .....	133
SVD – Singular Value Decomposition of a Matrix .....	134
Frobenius norm of a matrix.....	141
Matrix completion.....	143
Choosing “k”.....	144
Applying SVD .....	146
Maximum likelihood estimation .....	148
Introduction .....	148
General form .....	151
Bernoulli distribution.....	152
Gaussian distribution .....	155
Uniform distribution.....	157
Linear regression .....	159

Logistic regression.....	160
Summary .....	162
Theoretical properties .....	162
Bias .....	162
Variance.....	163
Mean Square Error (MSE).....	163
Relationship between Bias, variance, MSE: Bias-variance decomposition.....	163
Consistent estimators .....	165
Example MLE consistent, biased or unbiased .....	166
MLE Consistency.....	168
Bayesian inference .....	172
Introduction .....	172
Bayes' rule/theoreme .....	172
Bayes' rule demonstration .....	175
Example Bayes' rule .....	176
Summarizing Bayes' rule .....	178
Example: predicting the commute time.....	178
Example: bayesian linear regression.....	182
Clustering – K-means.....	188
Introduction .....	188
K-means.....	190
Mathematical definition.....	192
K-means as Optimization .....	193
Limitations with the current algorithm.....	197
EM algorithm.....	198
Clustering problem with MLE.....	203
Easy example for MLE .....	206
General example for MLE.....	208
EM algorithm procedure .....	211
Derivation of EM algorithm for global optimal solution .....	215
Demonstration EM algorithm is equivalent to maximizing $LB(\theta, \rho)$ .....	219
Intepretations from demonstrations: .....	220
Multivariate distributions .....	220
Introduction .....	220
Normal distributions .....	222
Multivariate Normal Distributions .....	224

Independent Normal Distributions .....	227
Marginal Distributions.....	228
Conditional Distributions .....	229
Linear transform.....	230
Multivariate Normal Distributions and PCA.....	232
Multivariate Normal with MLE.....	235
Multivariate Normal distribution for graphical models .....	239
Natural form.....	241
Conditional Distribution via Natural form.....	243
Proving the conditional distribution via natural form.....	244
Independence and conditional independence .....	245
Proof position matrix reflects conditional independence .....	248
Graphical models.....	251
Learning Gaussian Markov Random Fields .....	254
Kernel method.....	257
Introduction .....	257
Supervised learning .....	257
Basic ideas to build flexible function classes.....	259
Kernel method.....	260
Kernel and phi .....	262
Properties of Kernel functions as basis functions .....	264
Estimating theta “ $\theta$ ”.....	264
Kernel for classification problems.....	269
Kernel as equivalent infinite amount of features .....	270
Conclusions .....	277
Neural networks.....	279
Introduction .....	279
Commonly used activation functions.....	281
Building the neural network.....	282
Optimizing the neural network: non-convexity .....	283
Solving the optimization problem.....	286
Justification of Neural networks .....	287
Representing neural network as a graph .....	291
Deep neural networks.....	294
Statistical distributions.....	298
$\chi^2$ Distribution .....	298

Usages .....	302
T-student distribution .....	303
F-distribution.....	307

## Microsoft ML.NET

Curso Machine Learning:

[https://www.youtube.com/watch?v=X0DQjfW09kA&list=PLdo4fOcmZ0oUDTvk5XMNu es09Fnub\\_D0u](https://www.youtube.com/watch?v=X0DQjfW09kA&list=PLdo4fOcmZ0oUDTvk5XMNu es09Fnub_D0u)

**Sentiment analysis:** un input, respuesta es SI o NO (True o False). Predecir si un comentario de un usuario es tóxico o no.

**Issue classification:** varios inputs, varias respuestas posibles (categorizadas). Predecir la categoría de un ticket a partir del título y descripción del ticket (bug).

**Predicting value:** varios inputs, predice una respuesta numérica. Por ejemplo, predecir la factura del taxi en función de variables (tiempo viaje, ratio ciudad, numero de pasajeros).

**Mistake-Bounded problem: Classifying emails**

Gets a list of emails and finds out if it is a spam or not.

Mistake-Bounded problema.

Learner: gets a data point (an email) and it has to guess if it is spam or not.

Teacher: evaluates if correct or mistake.

If correct next email comes and learner will give its guess.

Teacher: responds with correct or mistake.

If wrong: we update the counter the number of mistakes. The learner or computer program learns something and updates its internal state.

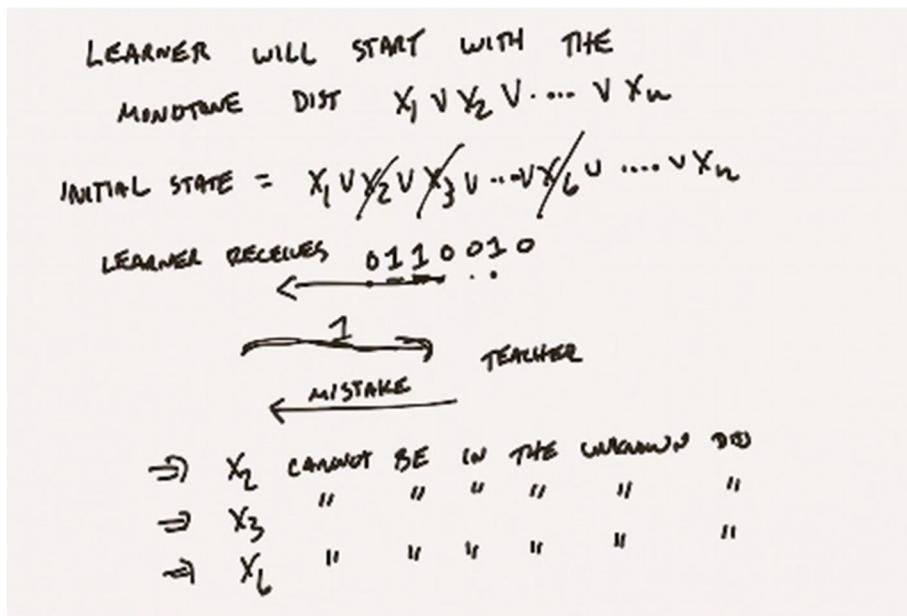
A learner has Mistake-Bound T if for every sequence of challenges the learner makes at most T mistakes. In an infinite sequence of emails it will make the most T mistakes.

Let us say we have a function C (monotone disjunction). The learner is presented with a string  $\{x_1, x_2, \dots, x_n\}$  with 0s and 1s. And the learner responds with 0 or 1. The learner is trying to learn function C.

Teacher says correct is GUESS =  $f(x)$ . Otherwise, it is a mistake.

Next challenge:  $\{x'_1, x'_2, \dots, x'_n\}$ .

Can we come up with an algorithm mistake-bound? We update our monotone disjunction with the new state and try keeping the monotone disjunction fitting for all previous values.



Learner will start with the monotone initial state:  $f(x) = x_1 \mid\mid x_2 \mid\mid \dots \mid\mid x_n$

Only 0 when  $\{0,0,\dots,0\}$

Learner receives: 0110010, learner says 1 because there is a one.

Teacher responds with mistake.

Learner learns that  $x_2$  cannot be in the monotone disjunction as well as  $x_3$  and  $x_6$ .

Every time we make a mistake we will eliminate a literal. Only way to make a mistake and learn is saying 1 and at least eliminating t most n literals.

$x_1, \dots, x_n \in \{0, 1\}^n$

"FEATURE EXPANSION"

$\underbrace{x_1, \dots, x_n}_n \rightarrow \underbrace{x_1, \dots, x_n, y_1, \dots, y_n}_{2n}$

Each  $y_i = \bar{x}_i$

$\underbrace{0110 \text{ (n=4)}}_{\text{input}} \rightarrow \underbrace{01101001}_{\text{output}}$

$f(x_1, \dots, x_n) = x_2 \vee \bar{x}_1 \vee x_7 \equiv$   
 $f(x_1, \dots, x_n, y_1, \dots, y_n) = x_2 \vee y_4 \vee \bar{x}_7$

Instead of monotone disjunctions, let's try disjunctions now:  $f(x) = x_1 \mid\mid !x_2 \mid\mid x_3 \mid\mid !x_4$

How can we use an algorithm to learn something like this?

We have an algorithm that needs to learn regular disjunctions with negations.

What we will do is taking an input:  $\{x_1, x_2, \dots, x_n\}$  and we will carry out a Feature Expansion:

$\{x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n\}$  that must be double the dimension.

Each  $y_i = !x_i$ .

Every time we have an input like:  $\{0, 1, 1, 0\}$  we transform it to  $\{0, 1, 1, 0, 1, 0, 0, 1\}$ .

In the same way the function we pursue:  $f(x) = x_2 \mid\mid !x_4 \mid\mid x_7$  will turn into:  $f(x) = x_2 \mid\mid y_4 \mid\mid x_7$ .

This function keeps the behavior. Now we can use the algorithm to learn monotone disjunctions an mistake bound to  $2n$ .

$x_1, \dots, x_n \in \{0,1\}^n$

"FEATURE EXPANSION"

$\begin{bmatrix} x_1, \dots, x_n \end{bmatrix}_n \rightarrow \begin{bmatrix} x_1, \dots, x_n, y_1, \dots, y_n \end{bmatrix}_{2n}$

Each  $y_i = \bar{x}_i$

$0110 \quad (n=4) \rightarrow 01101001$

$f(x_1, \dots, x_n) = x_2 \vee \bar{x}_1 \vee x_7 \equiv$

$f(x_1, \dots, x_n, y_1, \dots, y_n) = x_2 \vee y_4 \vee x_7$

APPLY MONOTONE DIST ALGO ON THESE CHALLENGES

USE ALGO FOR MONOTONE DIST WE HAVE A NEW ALGO WITH MISTAKE BOUND  $\leq 2n$ .

## Decision trees

A powerful way of classifying data and used a lot in practice with lot of nice theoretical properties.

Heuristics for learning decision trees and theoretical properties.

A decision tree is a boolean function that outputs 0 or 1 (two values).

Input for a decision tree is a bit string of length n:  $\{x_1, x_2, \dots, x_n\}$ .

A decision tree is composed of a root node and children. Every node is a decision: if "0" we go left and if "1" we go right.

0 5

DECISION TREE IS A BOOLEAN FUNCTION  
 (outputs  $+1$  or  $-1$ )  
 $0$  or  $1$

INPUT:  
 $x \in \{0, 1\}^n$        $f(x) \rightarrow \{0, 1\}$

SIZE OF DECISION TREE = # OF NODES IN THE TREE

DEPTH OF TREE  
 = LENGTH OF LONGEST PATH FROM ROOT TO A LEAF.

Size of the tree will be the number of nodes and the depth (length of the longest path from the root to the leaf).

Given a set of examples, build the tree with low error.

$S$  = Training set.

$m$  = number of training sets.

$n$  = number of features in each training  $\{x_1, x_2, \dots, x_n\}$

Error rate (equivalent to training error or empirical error rate) where error rate is the number of mistakes  $T$  makes on  $S$  divided the number of  $S$ .

GIVEN A SET OF LABELED EXAMPLES,  
BUILD A TREE WITH LOW ERROR.

$S = \text{TRAINING SET}$

$$\begin{pmatrix} x^1, y^1 \\ \vdots \\ x^n, y^n \end{pmatrix}$$

$y^i \in \{0, 1\}$   
 $x^i \in \mathbb{R}^n$

- ERROR RATE
- TRAINING ERROR
- EMPIRICAL ERROR RATE

=  $\frac{\text{# MISTAKES } T \text{ MAKES ON } S}{|S|}$

A natural approach in order to build decisión tres:

Given a set  $S$ .

Assume the tree is a leaf: how should we choose which literal to put at the top of the tree?

Big question i show to decide what literal o put at the root of the tree? Once we answer this question it will be easy to construct this tree because we will recurse using the same criterion at the subtree.

### NATURAL APPROACH FOR BUILDING DECISION TREES

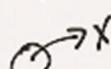
GIVEN A SET  $S$ .

$S$  — ASSUME: TREE IS A LEAF

ALWAYS +1 OR ALWAYS 0

CHOOSE MAJORITY OF LABELS

- A MORE INTERESTING TREE:



- HOW TO DECIDE WHAT LITERAL TO PUT AT THE ROOT OF TREE?

Let's define a potential function  $g(a)$ .

$g(a)$  is the min of  $(a, 1 - a)$ . We pick a literal ( $x_i$ ) and compute  $g(\text{probability } y = 0)$ .

Assume 5 negative examples (0) and 10 positive examples (1).

$$G(\text{Prob}, y=0) = g(5/15) = \min(1/3, 1 - 1/3) = 1/3.$$

Assume 10 negative examples (0) and 5 positive examples (1).

$$G(\text{Prob}, y=0) = g(10/15) = \min(2/3, 1 - 2/3) = 1/3.$$

This is the error rate for a tree with just one leaf.

DEFINE A POTENTIAL FUNCTION  $\phi(a)$

$$\phi(a) = \min(a, 1-a)$$

PICK A LITERAL  $x_i$

COMPUTE  $\phi(\Pr_{\{(x,y)\sim S\}}(y=0))$

ASSUME 5 POS EXAMPLES

ASSUME 10 NEG EXAMPLES

$$\phi\left(\Pr_{\{(x,y)\sim S\}}(y=0)\right) = \phi\left(\frac{1}{3}\right) = \min\left(\frac{1}{3}, \frac{2}{3}\right) = \frac{1}{3}.$$

$$\phi\left(\Pr_{\{(x,y)\sim S\}}(y=0)\right) = \phi\left(\frac{2}{3}\right) = \min\left(\frac{1}{3}, \frac{2}{3}\right) = \frac{1}{3}$$

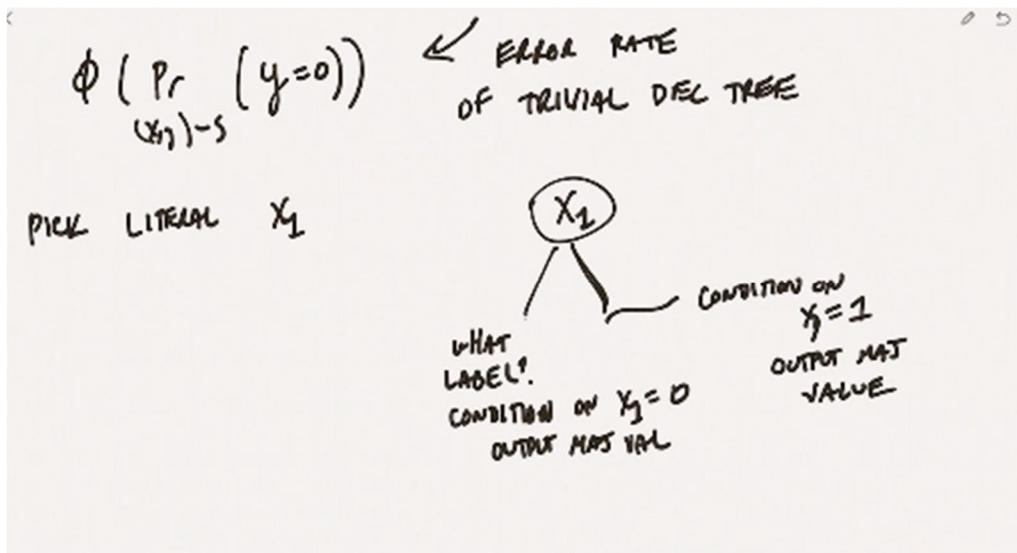
ERROR RATE FOR TREE WITH JUST 1 LEAF

If we have 10 negative examples and 5 positive examples and output 0 we will have only 1/3 error rate to make a mistake.

We have just seen the error rate of the trivial decision tree.

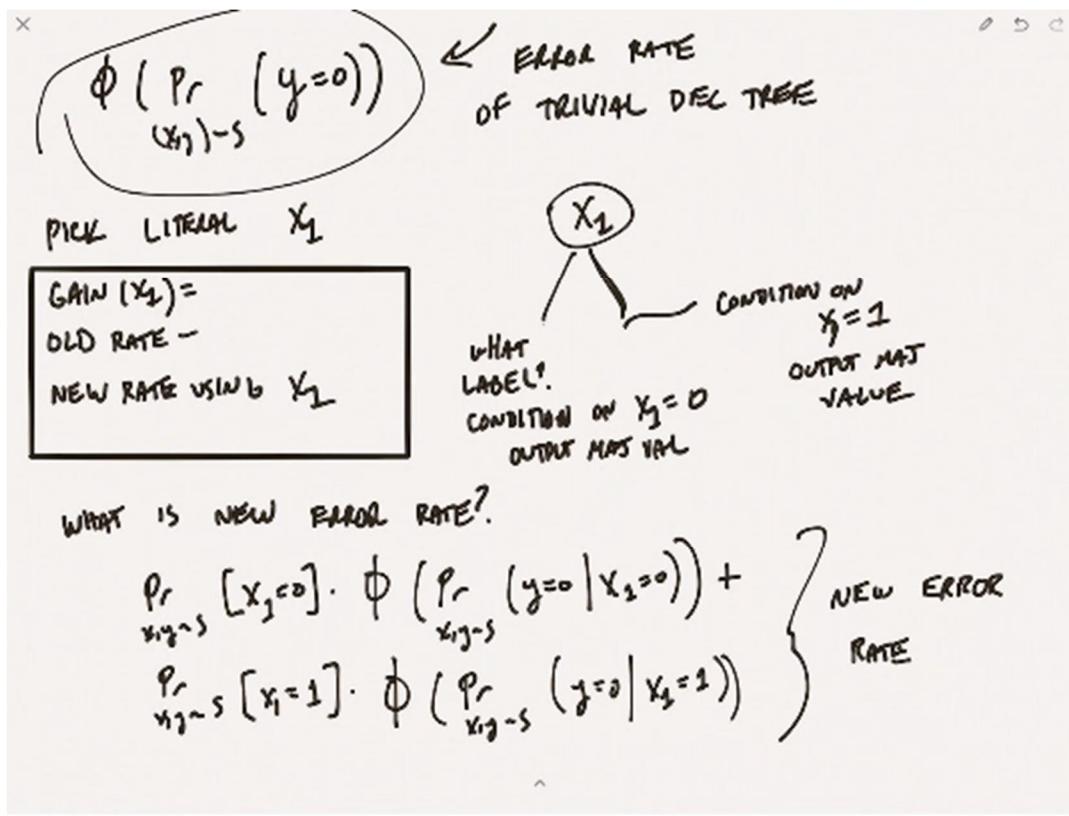
We are hoping for doing better than this.

We will place literal  $x_1$  at the top of the tree.



When building the decisión tree and we place  $x_1$  on the top then on the left we place the majority value when  $x_1 = 0$  and on the right the majority value when  $x_1 = 1$ .

What is the new error rate of this tree?



Gain of  $x_1$  is the old rate – the new rate using  $x_1$

We will go through all the  $x_i$  and compute the gain. That literal that maximizes the gain will be the one we put.

In the next training set we will take a set of trainings where either  $x_1 = 0$  or  $x_1 = 1$  to keep the training on and decide on the next literal to place in the next node.

✖ ↻ ↺

$\phi(\Pr_{x_1 \sim S}(y=0))$  ← ERROR RATE OF TRIVIAL DEC TREE

PICK LITERAL  $x_1$

$\text{GAIN}(x_1) =$   
 OLD RATE -  
 NEW RATE USING  $x_1$

WHAT IS NEW ERROR RATE?

$$\Pr_{x_1 \sim S}[x_1=0] \cdot \phi\left(\Pr_{x_1 \sim S}(y=0 \mid x_1=0)\right) + \Pr_{x_1 \sim S}[x_1=1] \cdot \phi\left(\Pr_{x_1 \sim S}(y=0 \mid x_1=1)\right)$$

} NEW ERROR RATE

You can imagine that you could start building trees that were extremely large, even exponentially large in the number of features that we have.

That's not going to be computationally feasible. So we're going to need some sort of stopping criterion.

How big does "S" need to be? That's a difficult question to answer.

Now it turns out that the structure of the tree is determined by the choice of potential function  $\phi$  which is meant to be:

$\Phi(a) = \min(a, 1-a)$  corresponds to the training error.

$\Phi(a) = 2 \cdot a \cdot (1-a)$ , which is the gini function or gini index.

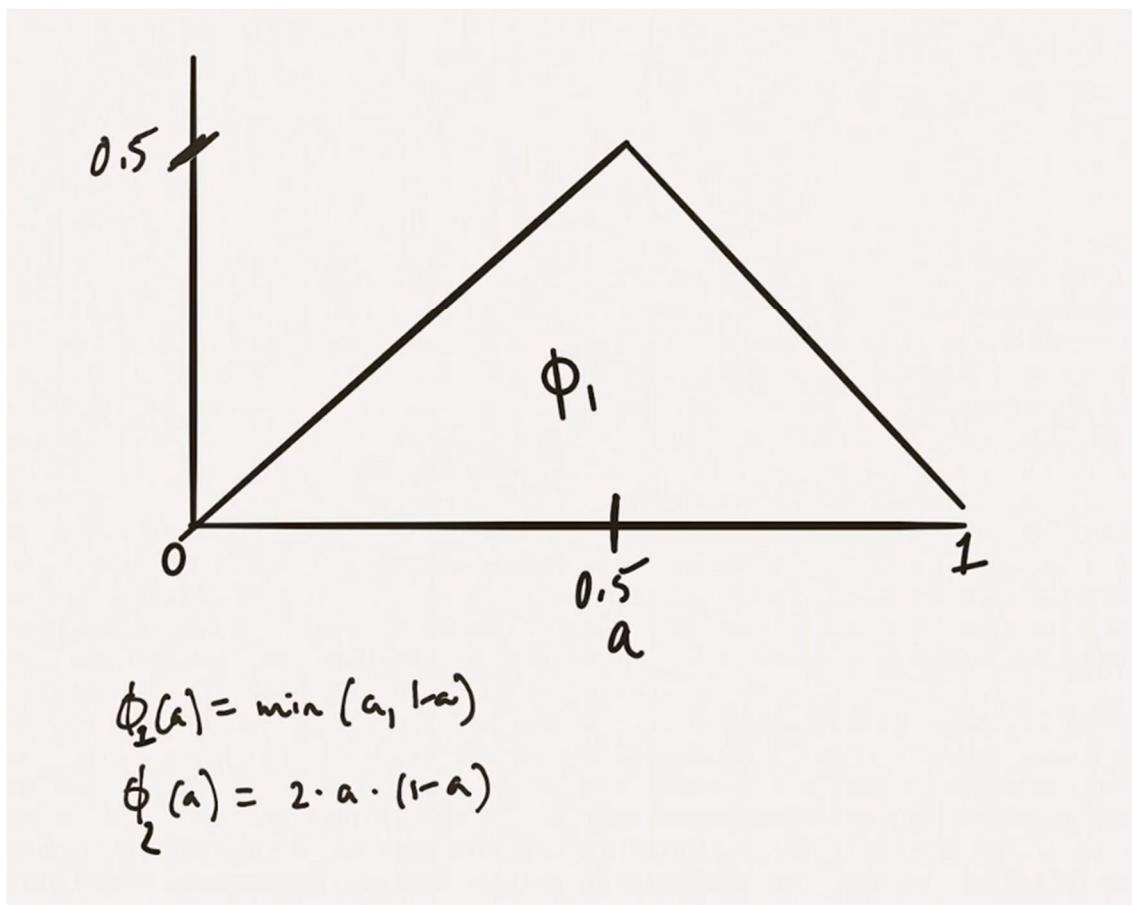
✖ ↻ ↺

STRUCTURE OF TREE IS DETERMINED BY CHOICE OF  $\phi$

$\Phi(a) = \min(a, 1-a)$  CORRESPONDS TO TRAINING ERROR

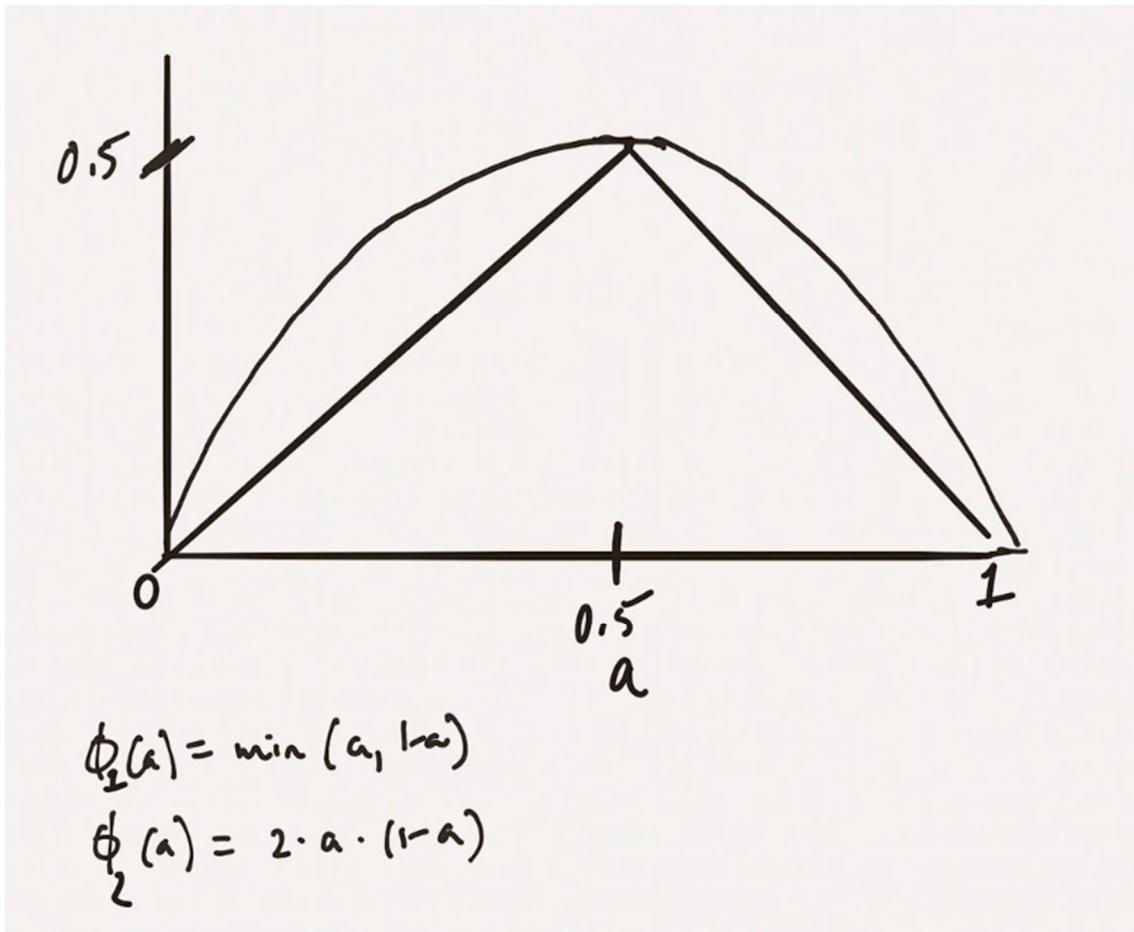
$\Phi(a) = 2 \cdot a \cdot (1-a)$  "GINI FUNCTION" OR "GINI INDEX"

Let's compare these two potential functions:



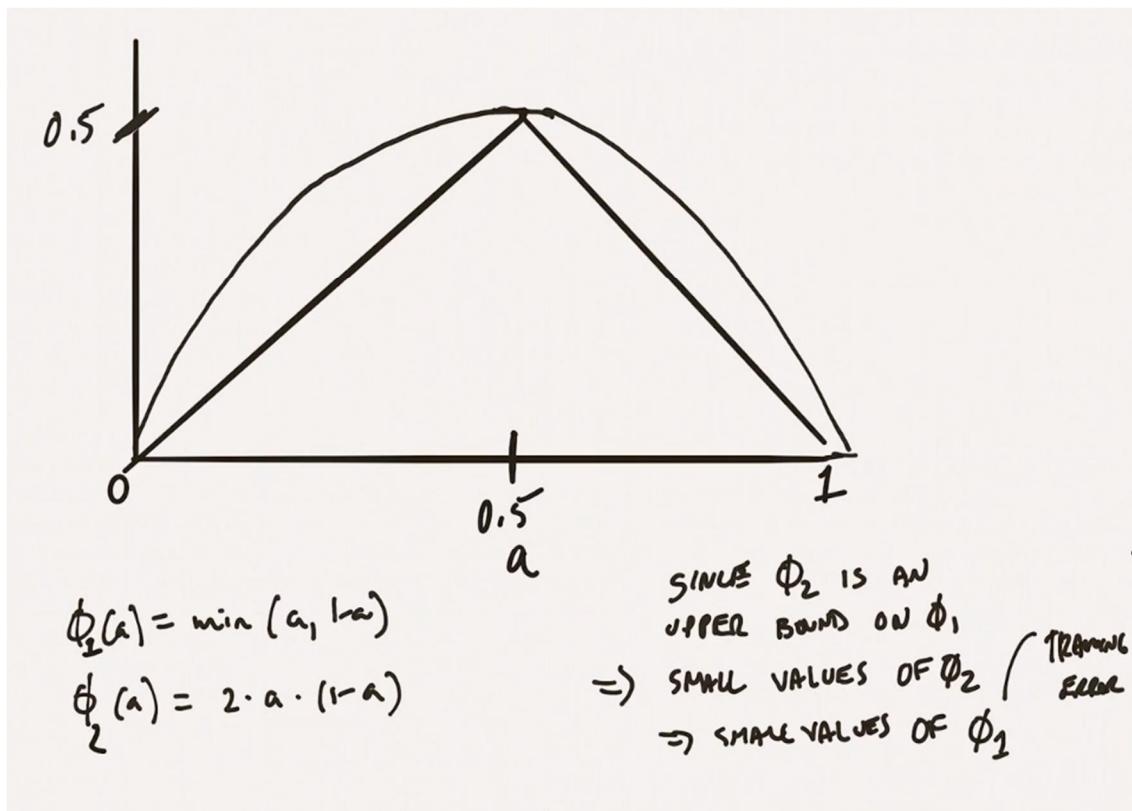
Problem is that this function cannot be derived and it is better to use smooth and continuous potential functions that can be derived.

This gini index is smooth and very close to the original one:



Since  $\Phi_2$  is upper bound on  $\Phi_1$  then Small values of  $\Phi_2$  lead to Small values of  $\Phi_1$ .

If we can build a potential function with  $\Phi_2$  where  $\Phi_2$  gets Small it also implies that  $\Phi_1$  gets Small because it is upper bound.



Next slide: When  $x_1=0$  and  $x_2=0$  we have 1 positive (1) and one negative example (0).

Total negative example = 5, total examples = 15.

$\Pr(x_1 = 0) = \text{total examples where } x_1 = 0 : 1+1+2+1 = 5 \text{ examples} / 15 \text{ total examples}$

$\Pr(\text{neg} | x_1=0) = 1 + 1 = 2 \text{ examples where result is negative (0) and } x_1 \text{ was 0 out of 5.}$

$\Phi_2(\Pr(\text{neg} | x_1=0)) = 2 \cdot 2/5 \cdot (1 - 2/5).$

$\Pr(x_1 = 1) = \text{total examples where } x_1 = 1 : 3+4+1+2 = 10 \text{ examples} / 15 \text{ total examples}$

$\Pr(\text{neg} | x_1=1) = 1 + 2 = 3 \text{ examples where result is negative (0) and } x_1 \text{ was 1 out of 10.}$

$\Phi_2(\Pr(\text{neg} | x_1=1)) = 2 \cdot 3/10 \cdot (1 - 3/10).$

$x_1$	$x_2$	Pos	Neg	$\phi(a) = 2 \cdot a \cdot (1-a)$
0	0	!	1	$\phi(\Pr_{\text{S}}[\text{Neg}]) = \frac{4}{9}$
0	1	2	1	
1	0	3	1	
1	1	4	2	$\frac{5}{15} = \frac{1}{3}$

$S =$

$$\Pr[x_1 \text{ to be at root?}]$$

$$\Pr[x_2 \text{ to be at root?}]$$

Look at  $x_1$

$$\Pr(x_1=0) \cdot \phi(\Pr(\text{Neg} | x_1=0)) + \Pr(x_1=1) \cdot \phi(\Pr(\text{Neg} | x_1=1))$$

$$\frac{1}{2} \cdot \frac{2}{5} \cdot \frac{2}{5} + \frac{1}{2} \cdot \frac{3}{5} \cdot \frac{2}{5} = \frac{7}{25} = \frac{11}{25}$$

Picking  $x_1$  at the root here have us some progress: 11/25.

Let's do the same thing for  $x_2$ :

$\Pr(x_2 = 0) = \text{total examples where } x_2 = 0 : 6 = 6 \text{ examples / 15 total examples}$

$\Pr(\text{neg} | x_2=0) = 1 + 1 = 2 \text{ examples where result is negative (0) and } x_2 \text{ was 0 out of 6.}$

$\Phi^2(\Pr(\text{neg} | x_2=0)) = 2 \cdot 2 / 6 \cdot (1 - 2/6).$

$\Pr(x_2 = 1) = \text{total examples where } x_2 = 1 : 9 = 9 \text{ examples / 15 total examples}$

$\Pr(\text{neg} | x_2=1) = 1 + 2 = 3 \text{ examples where result is negative (0) and } x_2 \text{ was 1.}$

$\Phi^2(\Pr(\text{neg} | x_2=1)) = 2 \cdot 3 / 9 \cdot (1 - 3/9).$

Picking  $x_2$  at the root here have us some progress too: 4/9.

$x_1$	$x_2$	Pos	Neg	
0	0	!	!	$\Phi(a) = 2 \cdot a \cdot (1-a)$
0	1	2	1	$\Phi(\Pr_{S \setminus \{x_1\}}[\text{neg}]) = \frac{4}{9}$
1	0	3	1	$\frac{5}{15} = \frac{1}{3}$
1	1	4	2	$2 \cdot \frac{1}{3} \cdot \frac{2}{3} = \frac{4}{9}$

PICK  $x_1$  to BE AT Root?  $\frac{11}{25}$   
 $x_2$  to BE AT Root?  $\frac{4}{9}$

Look at  $\frac{2}{5} \cdot 2 \cdot \frac{1}{3} \cdot \frac{2}{3} + \frac{3}{5} \cdot \frac{4}{9} = \frac{4}{9}$

The gain for  $x_1$  at the root is:

$$\text{Gain} = \Phi_2(\Pr(\text{neg})) - 11/25 = 4/9 - 11/25 > 0$$

The gain for  $x_2$  at the root is:

$$\text{Gain} = \Phi_2(\Pr(\text{neg})) - 4/9 = 4/9 - 4/9 = 0$$

The gain when we pick is  $x_1$  at the root is greater.

This is something that is not really difficult to code up.

One question is: when should we stop? And there are many answers to this question.

- 1) Stop when the gain is extremely Small for all literals. As we are doing this recursive procedure we could just say that if gain is less than delta (Small value) we stop.
- 2) Pruning: build an enormous tree and have some parameter indicating how many nodes we want. We start at the bottom and going up the tree and prune nodes that have the less impact on the decision tree.
- 3) Random forests: the idea is to build many Small decision trees and then take a majority of resulting trees. The way to build many decision trees needs an algorithm that takes the training set "S" and randomly subsample from "S" to create "S'". Then randomly again pick some features from  $x_1, \dots, x_n$  (we don't take all of them). Finally we build a decision tree using  $S'$  and the subset of random features from  $x_1, \dots, x_n$ .

## Generalization

### Introduction to PAC learning with decision trees

0 5

# PAC LEARNING

- WHAT IS THE "TRUE ERROR" OR GENERALIZATION ERROR OF A CLASSIFIER?
- DECISION TREES: FIX  $T$

$D$  ON NEW EXAMPLES

$(x, y)$  → CHALLENGE

LABEL  $L$  →

$\Pr_{(x,y) \sim D} [T(x) \neq y]$

OF  $T$

TRUE ERROR  
GENERALIZATION ERROR

Predictive power of a classifier. So far we have studied how to build classifier base don training sets. Now we want to know how well is that decision tree is going to do with unseen data an a classifier. Our classifier might not do well on generalization.

The PAC model is the foundational model learning that we will see.

We want to find out what is the TRUE ERROR or GENERALIZATION ERROR of a classifier.

In decision trees let's fix some tree called " $T$ ".

Let's say we have a probability distribution " $D$ " on new examples, that is, a new challenge:

$(x, y)$  where  $x$  is the challenge and  $y$  the label and we want to find ourit:  $\Pr[T(x) \neq y]$  of  $T$ .

True error should be Small.

Let's imagine we are given a training set " $S$ " and a learner is given " $S$ ". The learner uses the classifier and carries out the analysis of the training set. If we get a new point not in the training set the learner could just output "0" but that is not real training unless " $S$ " is the entire set of examples.

0 5

EACH  
 $x^i \in \{0,1\}^n$   
 $y^i \in \{0,1\}$

LET'S BUILD A DECISION TREE

$x^i$  ARE DISTINCT

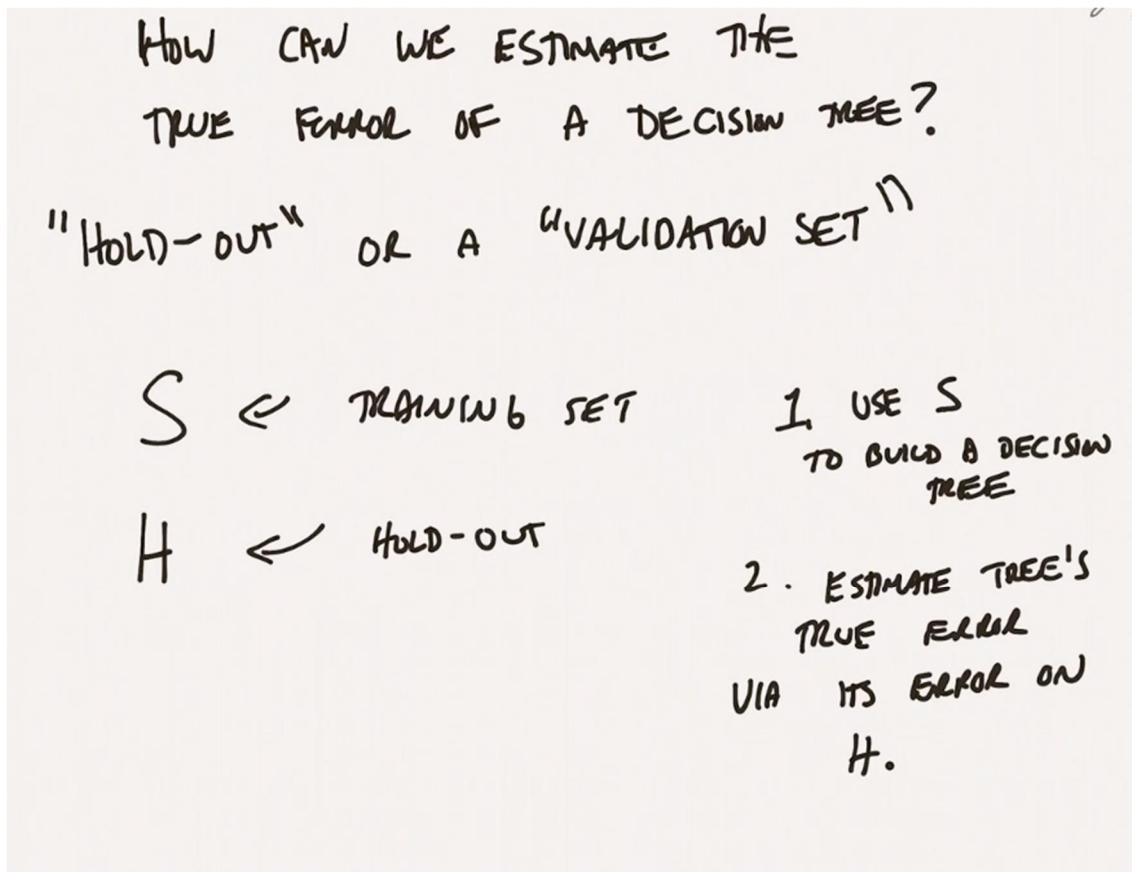
YOU CAN BUILD A DECISION TREE (SIZE  $\geq |S|$ )  
 THAT IS CONSISTENT WITH ALL THE POINTS  
 IN S.

QUESTION: How well does this tree generalize?  
 What is the true error of this tree?

Let's now consider the case that we have a training set "S" where each " $x_i$ " is binary as well as the output " $y_i$ ".

Let's build the decision tree assuming all the  $x_i$  are distinct and there are no repeats in variables.

Building this decision tree (size  $> |S|$ ) that is consistent with all the points in the training set "S". It means, we have achieved a very good model for this training set.



Question is: how well does this tree generalize for points outside the training set? In other words, how well does this tree generalize or what is the true error of this tree?

Memorizing every entry in the training set is not really learning and there is no predictive power. Now we have to worry whether we are getting predictive power or not (low generalization error) as well as good memory in the current training set.

Estimating the true error of a classifier (a decision tree in this case) requires the HOLD-OUT or a VALIDATION SET.

The idea here is having:

"S": training set

"H": hold out data

- 1) We use "S" to build a DECISION TREE (classifier)
- 2) Estimate tree's true error via its error on H. Counting the error rate of tree "T" on "H". The estimate of this TRUE ERROR will be close to the error on "H".

If the error is high we have to make some changes to build the decision tree:

- 1) Use a different algorithm.
- 2) Use a different potential function.

Then we look again at the error rate on "H" and we carry on building decision trees until the estimate for the true error is good enough for us.

The risk of doing this is that we are really training a model with two sets: "S" and "H".

The error on "H" is expensive as we need to analyze and depends on how big the set "H" is.

There is an interesting technique called cross-validation although we will continue analyzing basic techniques to estimate the true error of a tree.

Basic technique: model complexity (similar to cross-validation)

## ANOTHER APPROACH:

TRADE-OFF TRAINING ERROR WITH  
"MODEL COMPLEXITY"

DEFINE ANOTHER POTENTIAL FUNCTION  $\Phi$

$\Phi: \text{TREES} \rightarrow \mathbb{R}$  GIVEN A TRAINING SET  $S$

$$\Phi(T) = \underbrace{\text{TRAINING ERROR ON } S}_{\text{MINIMIZE } \Phi} + \alpha \cdot \frac{\text{SIZE}(T)}{|S|}$$

HYPERPARAMETER

Another approach is to trade-off training error with something we call MODEL COMPLEXITY.

We can define another type of potential function:

$\Phi$ : this time this potential function will map trees to real numbers given a training set "S".

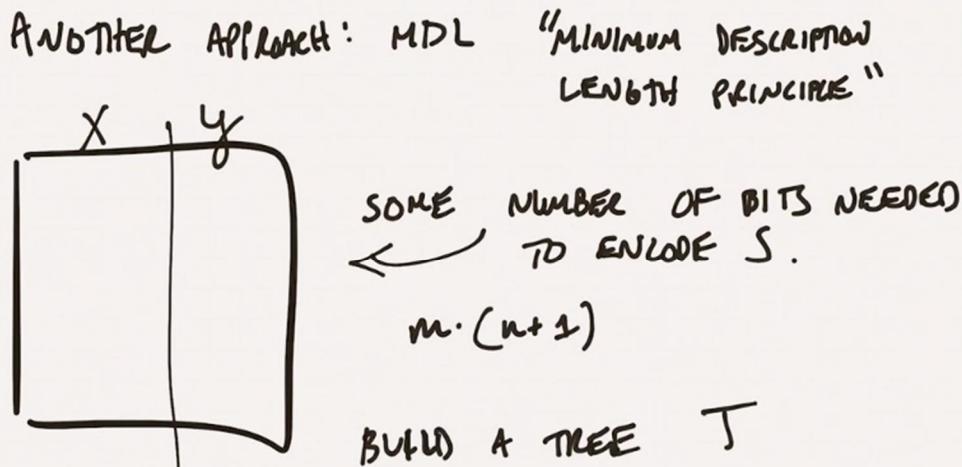
$\Phi(T) = \text{Training error on } S + \alpha \cdot (\text{Size}(T)/|S|)$ , where  $\alpha$  is the hyperparameter that we set at the beginning of the training.

The goal is to minimize  $\Phi(T)$ .

The error grows on two sides: error of the training  $S$  with  $T$  and the size of the tree itself.

This does not require the use of a hold-out set. If we're able to get a low error and a reasonable small size of the tree then we know we are looking good.

Basic technique: minimum description length principle



LET'S SAY T IS CORRECT ON 90% OF S  
AND INCORRECT ON 10%.

WE CAN ENCODE S USING #BITS(T) + #BITS TO  
ENCODE THAT 10%. WE GOT  
WRONG.

Another common approach is the "MDL" (minimum description length) which is a similar idea.

We have again a training set "S" and some number of bits needed to encode "S".

"m" is the number of examples.

"n" is the number of variables or features ( $x_1, x_2, \dots, x_n$ )

We build a tree "T".

Let's say "T" is correcto n 90% of "S" and incorrecto on 10%.

We can encode "S" using: number of bits (T) + number of bits to encode that 10% we got wrong.

The goal is to find out how well we can compress the data set we're given.

PAC model of learning

## PAC MODEL OF LEARNING

THERE IS A DISTRIBUTION  $D$  ON  $\{0, 1\}^n$  ( $\mathbb{R}^n$ )

FUNCTION CLASS  $C = \{$  DECISION TREES OF SIZE  $S\}$

LEARNER (RUNS IN POLYNOMIAL-TIME)

Fix  $c \in C$   $c$  IS THE UNKNOWN DEC. TREE  
WE WANT TO LEARN

RECEIVES

$$(x'_1, y'_1) \quad x' \in D \quad y' = c(x')$$

$$(x'_2, y'_2) \quad y'_2 = c(x'_2)$$

$$\vdots$$

$$(x'_m, y'_m)$$

GOAL: OUTPUT  $h \in C$

$$\Pr_{x \sim D} [h(x) \neq c(x)] \leq \epsilon = .01$$

LEARNER  
SHOULD BE  
EFFICIENT

$$(n, S)$$

This is going to lead us to the PAC model of learning. In this model there is a distribution "D" on  $\{0, 1\}^n$  (bits).

We are gonna use a function class "C" = { Decision trees of size S }

We also have a learner that runs in polynomial-time that receives "c" that belongs to "C". We want to learn this "c" which is the unknown decision tree.

We also receive:  $(x, y)$  where  $x$  belongs to D and  $y = c(x)$ . This is the training set.

$$(x_1, y_1)$$

$$(x_2, y_2)$$

...

$$(x_m, y_m)$$

The goal is for the learner to output "h" that belongs to "C" (a decision tree).

Then this function should be able to the following:

$\Pr[h(x) \neq c(x)] < \delta$ ; which means that the probability of the output of "h" for "x" should be very close to  $c(x)$ : less than  $\epsilon$  value, i.e low probability the value is different.

The learner should be efficient with both parameters: "n", "S". And the learner should always run in time polynomial in "n" and "S".

"OVER THE DRAWS FROM  $\pi$ "

WITH PROBABILITY AT LEAST  $1 - \delta$ ,

THE LEARNER SHOULD OUTPUT A HYPOTHESIS  $h$  S.T.  $\Pr_{\text{rand}}[h(x) \neq c(x)] \leq \epsilon.$

RUN-TIME = polynomial  $(\frac{1}{\epsilon}, \frac{1}{\delta}, n, s).$

IMAGINE LEARNER REQUESTS  $x^1, \dots, x^m$   
 $x^1 = \dots = x^m$

PAC → CORRECT      L. VALIANT [1984]

PAC  
PROBABLY → APPROXIMATELY

The real goal of many tasks in machine learning is to just output some hypothesis, some classifier "h", whose true error is at most  $\epsilon$  (Small).

In the mistake-bounded model of learning the criterion was very simple. Here, we start to talk about probabilities and we have a probability distribution.

With probability over the draws from the distribution "S" is  $1 - \delta$  the learner should output a hypothesis  $h(x)$  such that  $h(x)$  is accurate.

And the running time which includes the number of draws to take from the distribution should be polynomial. Over  $\delta$ , " $n$ " and " $S$ ".

Imagine that the learner keeps requesting new draws from the distribution and all the "x" sets are the same example over and over. We cannot expect the learner to output a classifier with small error so we have to allow some probability of failure just because we are drawing sets from some unknown probability distribution.

PAC learning: Probably Approximately Correct.

- Probability because there will be a low probability of failure:  $\delta$
- Approximately because  $\Pr[h(x) \neq c(x)] < \epsilon$
- Correct because it will be very similar.

As we demand more accuracy and  $\epsilon$  gets smaller then we require more running time to get less failure probability we also require to draw more samples.

WHEN CAN WE LEARN A FUNCTION CLASS?  
WHAT FUNCTION CLASSES CAN WE PAC LEARN?

GIVE LEARNER AN ALGORITHM A

$A : \text{TRAINING SETS} \rightarrow \text{DECISION TREES}$

$A(S)$  OUTPUT A TREE T THAT IS CONSISTENT WITH S. SIZE OF T IS GONA TO BE AT MOST s.

A ALWAYS OUTPUTS A CONSISTENT HYPOTHESIS FROM C  
GIVEN ANY TRAINING SET.

Q: GIVEN ALGORITHM A, HOW CAN WE PAC LEARN C?

We have delta and épsilon and the next question is when can we PAC learn a function class (tree, etc)?

Give a learner and Algorithm "A".

Input of A: Training sets which will turn into decision tree "T".

"A" is pretty powerful and can output a tree "T" that is consistent with the training set "S" when  $A(S)$ . The size of "T" (number of nodes) is gonna be at most "s" (number of examples in the training set).

"A" always outputs a consistent hypothesis from "C" given any training set "S".

Question is: given "A", how can we PAC learn "C"?

- DRAW SUFFICIENTLY MANY TRAINING POINTS = "S"  $\{x_i, y_i\}$
  - USE A TO FIND  $c \in C$  CONSISTENT WITH S.
  - OUTPUT  $C$ .
- Q: How large should S be?

Answer: draw sufficiently many training points for a training set "S", then use "A" to find "c" that belongs to "C" consistent with "S" and output "c" (the tree "T").

We need to determine how large the training set "S" needs to be.

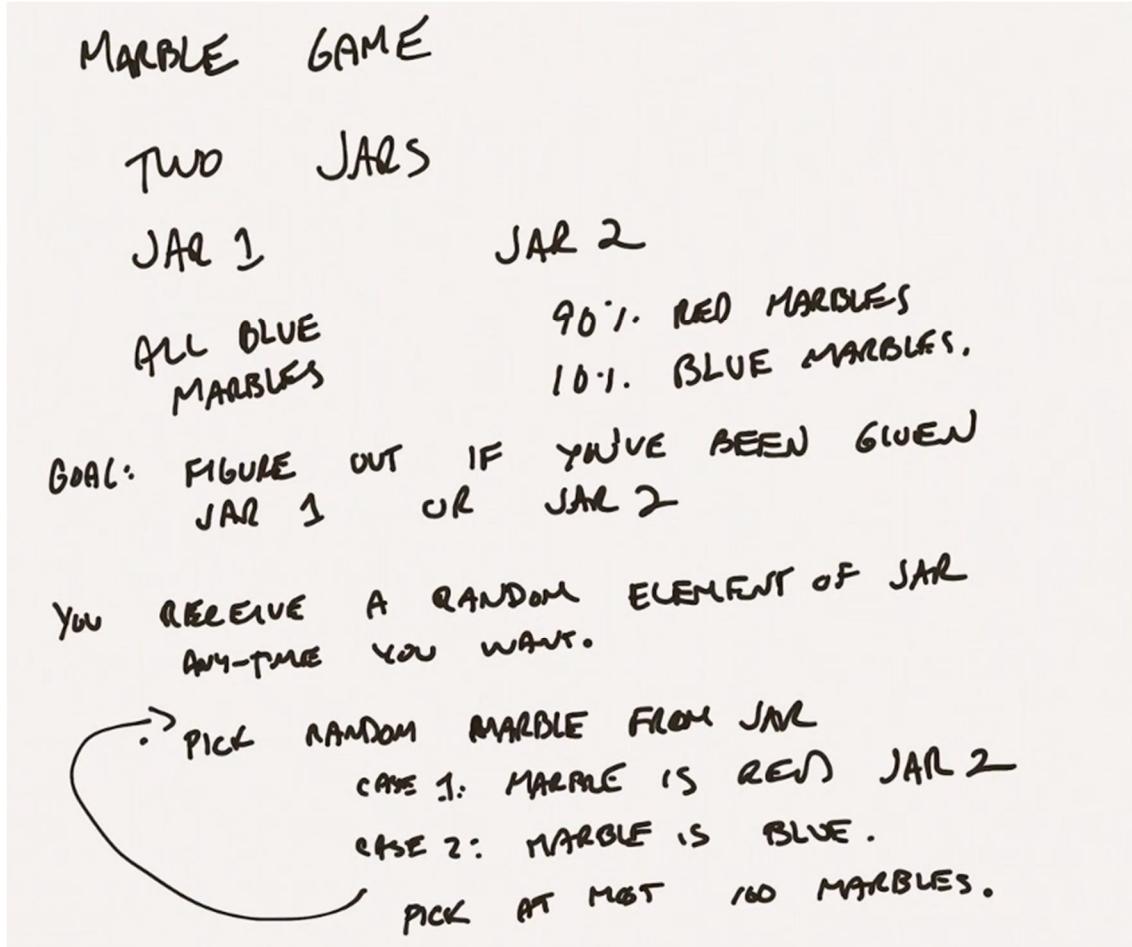
The PAC learning model needs to run in polynomial time and if we have to pick exponentially from our distribution we're in trouble to find a good model.

What we are able to do is draw a training set that is gonna be much larger than the size of the classifier we output. The size of the classifier will be smaller than our training set.

We need to assume that "A" runs in one time step (polynomial time).

The size of this tree "T" will play an important role in the PAC learning.

## The marble game - probabilities



The MARBLE game consists of two jars:

Jar 1: all blue marbles.

Jar 2: 90% red marbles and 10% blue marbles..

GOAL: figure out if we're given Jar 1 or Jar 2.

INFORMATION: we receive a random element from the Jar anytime we want.

In order to figure out if we're given Jar 1 or Jar 2 we take the next steps:

- Pick marble from Jar.
    - o Case 1: marble is red: we are Jar 2
    - o Case 2: marble is blue: probably Jar 1

Every step, the probability of being Jar 1 if we keep iterating increases.

WHAT IS THE PROBABILITY OF FAILURE?

PROB OF FAILURE IS  $(.1)^{100}$

← FAILURE  
PROB  
"δ-PARAMETER"

LET'S RETURN TO PAC LEARNING

- DRAW MANY SAMPLES  $S$
  - RUN A
  - OUTPUT CLASSIFIER  $c$  THAT IS CONSISTENT WITH  $S$  GIVEN FROM A.
- BAD EVENT:  
WE OUTPUT  $c \in C$  THAT IS CONSISTENT WITH  $S$  BUT TRUE ERROR OF  $c$

WHAT IS THE PROBABILITY THIS PROCEDURE FAILS?  $\leq \delta$

The big question is what is the probability of failure because there is some probability this algorithm will fail:

$\Pr(\text{failure}) = 10\%^{100}$ , if we draw 100 times. This corresponds to the  $\delta$  parameter in PAC learning.

Returning to PAC learning:

- Draw many samples.
- Run "A".
- Output classifier "c" (tree "T") that is consistent with "S".
- What is the probability this procedure "c" fails?  $\Pr(\text{fails}) > \delta$

What could go wrong in order to fail assuming we have a large training set "S":

- We output "c" that is consistent with "S" but the true error of "c" is greater than "E" (épsilon). Bad event.

Bad event scenario

BAD EVENT:

OUTPUT  $c$  THAT IS CONSISTENT WITH  $S$   
BUT HAS TRUE ERROR  $> \epsilon$ .

$\Pr[\text{BAD EVENT}] ?$

ENUMERATED ALL FUNCTIONS IN  $\mathcal{C} = \{c_1, \dots, c_N\}$

FIX  $c_1$  ASSUME  $c_1$  HAS TRUE ERROR  $> \epsilon$

WHAT IS  $\Pr_S[c_1 \text{ is consistent with } S] ? \leq (1-\epsilon)^{|S|}$

FIX  $c_2$  "  $c_2$  HAS " "  $> \epsilon$   
 $\Pr_S[c_2 \text{ is consistent with } S] ? \leq (1-\epsilon)^{|S|}$

The BAD EVENT is that we output "c" consistent with training set "S" but has true error > épsilon.  
=

Let's imagine we have several outputs "c" from "A":  $\{c_1, c_2, \dots, c_n\}$

Assume  $c_1$  has true error greater than épsilon.

What is:  $\Pr[c_1 \text{ is consistent with } S] < (1 - \epsilon)^{|S|}$ ,  $|S|$  = size of the training set.

Assume  $c_2$  has true error greater than épsilon.

What is  $\Pr[c_2 \text{ is consistent with } S] < (1 - \epsilon)^{|S|}$ ,  $|S|$  = size of the training set

o s c

FOR EVERY  $c_i$  (WITH ERROR  $> \epsilon$ )  
 $\Pr_{S} [c_i \text{ IS CONSISTENT ON } S] \leq (1-\epsilon)^{|S|}$  ↴ BAD

Q: RANDOMLY FORM  $S$ , WHAT IS THE PROB  
 THERE EXISTS A FUNCTION  $c \in C$  WHOSE ERROR  
 $> \epsilon$  AND IS CONSISTENT WITH  $S$ ?

UNION BOUND  $A, B \quad \Pr[A \cup B] \leq \Pr[A] + \Pr[B]$

$$\Pr[\text{BAD}] \leq |C| \cdot (1-\epsilon)^{|S|} \leq \delta$$

$$\text{SOLVE FOR } |S|: \quad (1-x) = e^{-x}$$

For every,  $c_i$  with error greater than  $\epsilon$  the  $\Pr[c_i \text{ is consistent with } S] < (1-\epsilon)^{|S|}$ .

Question for BAD EVENT: what is the probability if we were to draw from a random training set "S" that "c" is consistent with "S" and "c" belongs to "C" but error is greater than  $\epsilon$ .

Answer:  $\Pr[\text{BAD EVENT}] = |C| \cdot (1-\epsilon)^{|S|}$

Size of "C":  $\{c_1, c_2, \dots, c_m\}$

Size of "S":  $\{x_1, y_1\}, \{x_2, y_2\}, \dots, \{x_n, y_n\}$

$\Pr[\text{BAD EVENT}] = |C| \cdot (1-\epsilon)^{|S|} < \delta$  (we want it less than delta)  $\rightarrow$  equivalent to the following if we want to get  $|S|$ .

$$|\mathcal{C}| \cdot (1-\epsilon)^{|S|} \leq \delta$$

$$|\mathcal{C}| \cdot e^{-\epsilon|S|} \leq \delta \quad (\ln(1+x) \approx x)$$

$$e^{-\epsilon|S|} \leq \frac{\delta}{|\mathcal{C}|}$$

IF YOU CHOOSE  
# TRAINING POINTS  
LARGER THAN  $\frac{\log(\frac{|\mathcal{C}|}{\delta})}{\epsilon}$

$$-\epsilon|S| \leq \log\left(\frac{\delta}{|\mathcal{C}|}\right)$$

$$|S| > \frac{\log\left(\frac{|\mathcal{C}|}{\delta}\right)}{\epsilon}$$

ITEM WITH PROB.  $\geq \delta$ ,  
FUNCTION OUTPUT  
 $c$  IS  $1-\epsilon$   
ACCURATE.

We will output the size of the training set "S" we need to be sure we don't have a bad event.

If we choose the number of training points to be larger than this value then with probability  $> 1 - \delta$  we know the function output "c" is  $1 - \epsilon$  accurate.

Now we know how large this training set needs to be in order to be sure accurate.

## PAC learning

Introduction

0 5

**PAC-LEARNING**    **AXIS-PARALLEL RECTANGLES.**

WE ARE WORKING IN 2-DIMENSIONS

- LABELED + IF THE POINT IS INSIDE  $c$ , AXIS- $\mu$  RECT.
- IF THE POINT IS OUTSIDE  $c$ , AXIS- $\mu$  RECT.

GOAL: GIVEN  $\epsilon, \delta$   
OUTPUT  $h$  THAT IS  $\epsilon$ -ACCURATE  
WITH PROB  $> 1 - \delta$

Let's try another example where we will PAC learn axis-parallel rectangles.

We will work in 2D with an XY plane and the rectangle sides will be parallel to the X and Y axis.

We will label the point as positive if it is inside "c" (unknown axis parallel rectangle).

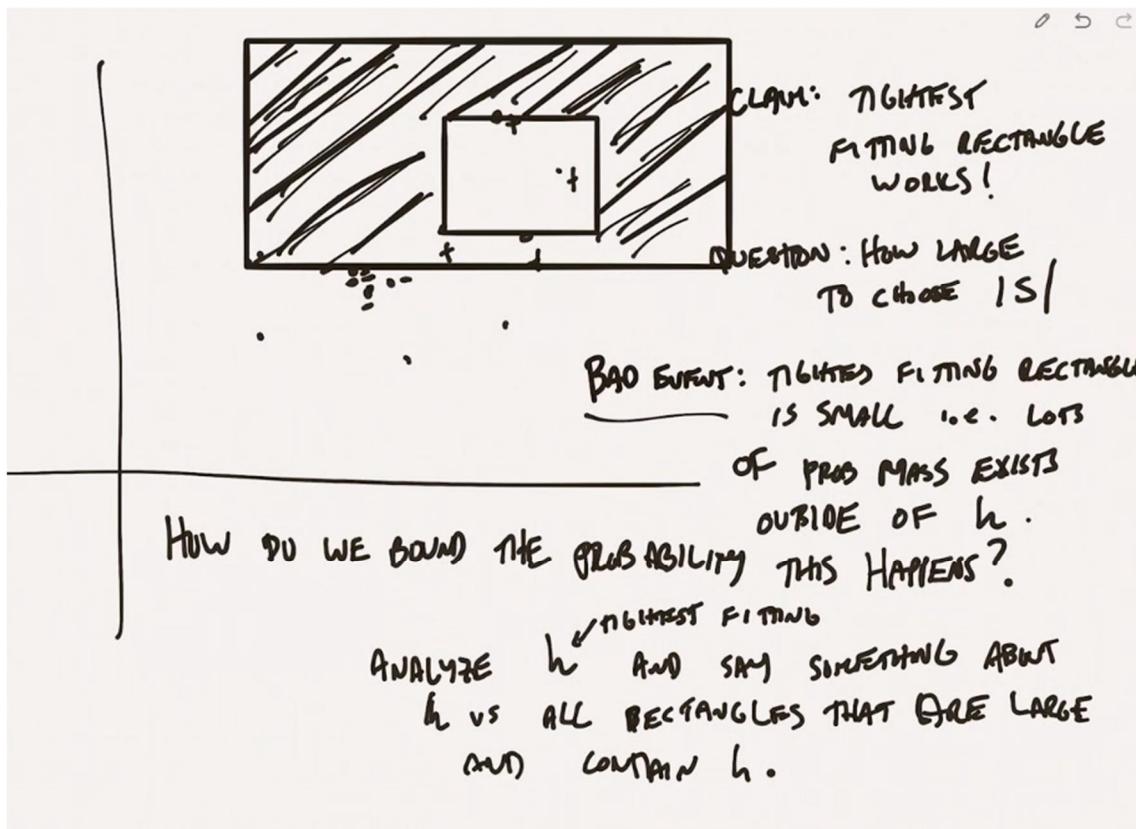
We will label the point as negative if it is outside "c" (unknown axis parallel rectangle).

Our goal is that given  $\epsilon$  and  $\delta$  our output "h" is accurate with probability  $> 1 - \delta$ .

There are many algorithms in machine learning that are based on geometry.

We are looking for a candidate algorithm that actually PAC learns axis parallel rectangles in two dimensions.

Algorithm or finding the function class: Suggestion is to take all positive points and take a rectangle that most closely covers all of our positive points. Another suggestion is to take some rectangle that is a bit larger than the tightest fitting rectangle. The tightest fitting rectangle works but how do we analyze it?



We need to specify the sample size.

Assuming tightest fitting rectangle Works the question is how large to choose our training set  $|S|$ .

We are taking random points given a distribution.

What do we consider a bad event? A bad event would be that these positive points are clustered around some tiny rectangle that's very very small with respect to the true rectangle.

Bad event: tightest fitting rectangle is small i.e lots of probability mass exists outside of output "h".

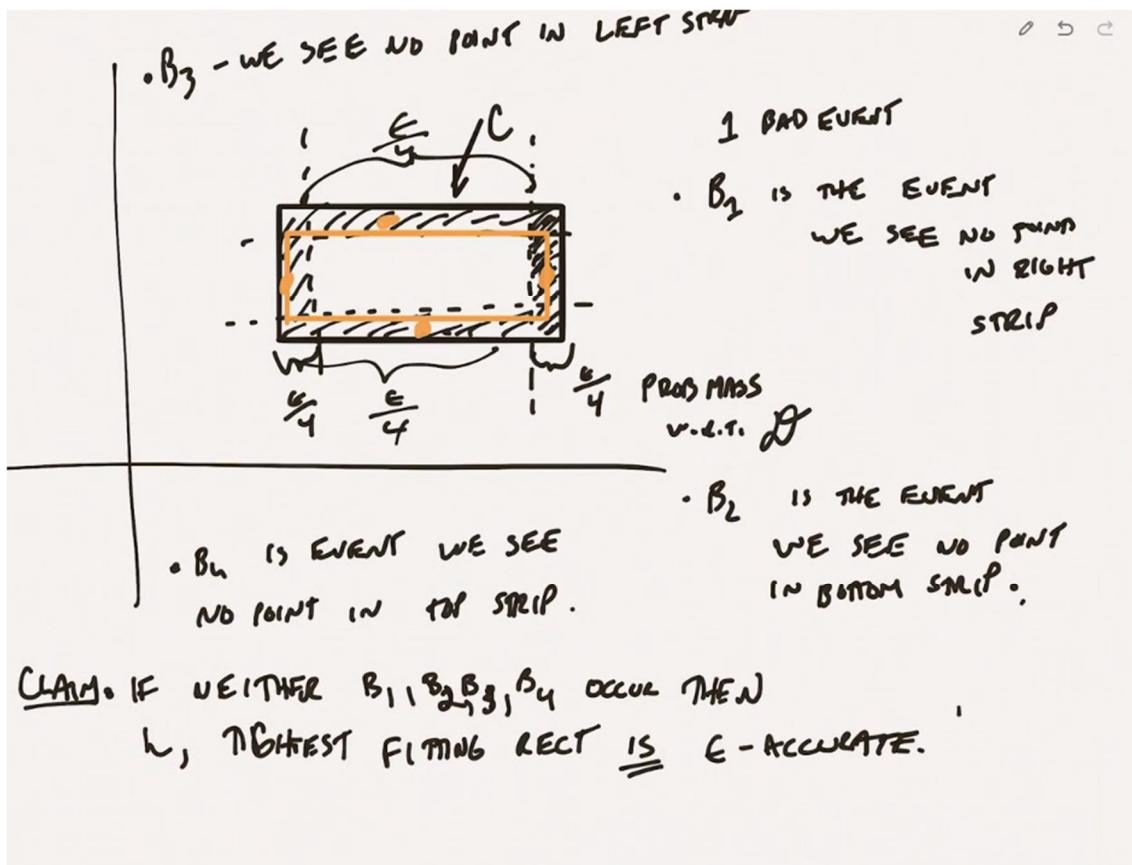
If the probability of landing in the shaded region is at least  $\epsilon$  then we are in trouble because we're going to say that this shaded area is negative.

How do we bound the probability this happens? We need to play with  $|S|$  and the number of training sets in order to prevent this from happening.

The more points we analyze the closer we are to finding the tightest fitting rectangle to all points in the XY plane.

A bad event is that there is an enormous rectangle that surrounds because we didn't take a good set of points.

Analyze "h" (our tightest fitting rectangle) and say something about "h" VS all rectangles larger and contain "h".



So the ratio of the areas between the tightest fitting rectangle and the true rectangle is going to indicate our error.

Let's assume "c" is our true rectangle.

We're gonna define our bad event: a region of the rectangle where we didn't get points:

$B_1$  is the event we see no points in right strip ( $\epsilon$ /4).

$B_2$  is the event we see no points in bottom strip ( $\epsilon$ /4).

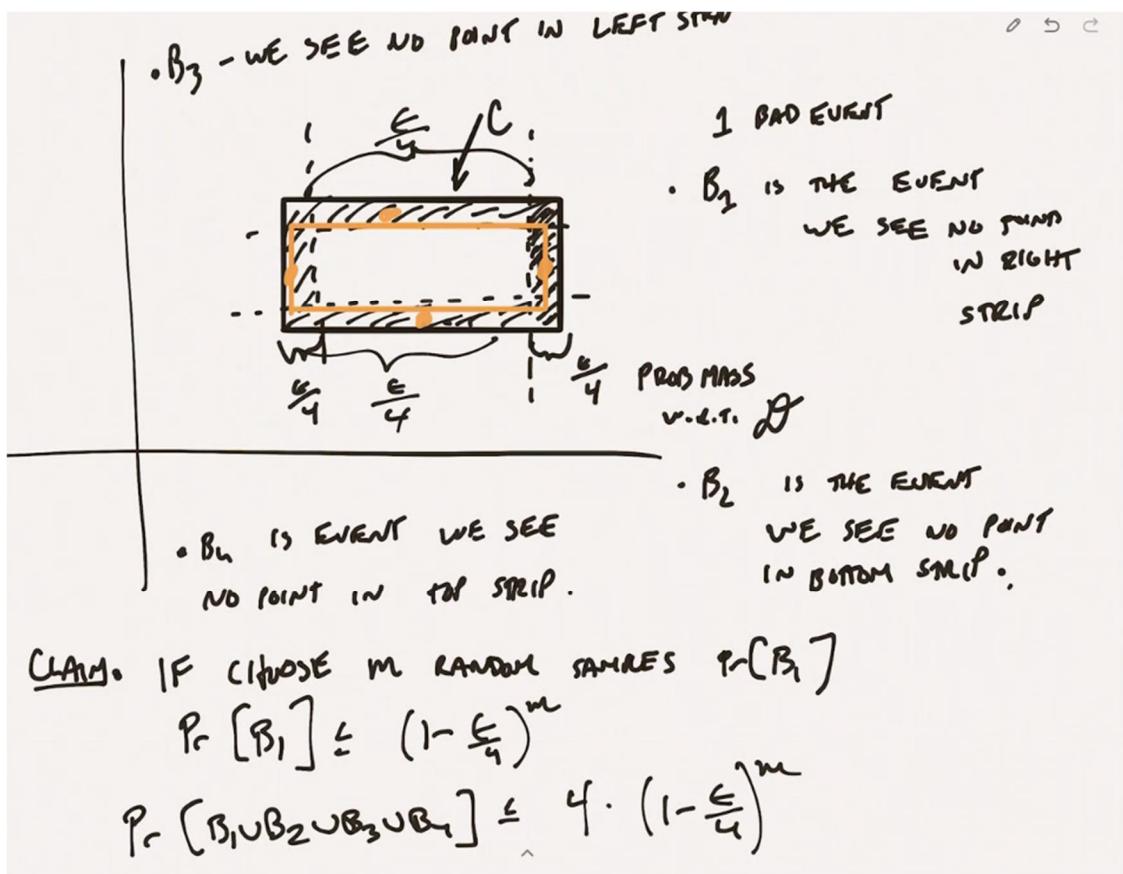
$B_3$  is the event we see no point in left strip ( $\epsilon$ /4).

$B_4$  is the event we see no point in the top strip ( $\epsilon$ /4 probability mass).

Four bad events each one corresponding to a strip of probability mass  $\epsilon$ /4.

Claim: if neither  $B_1, B_2, B_3, B_4$  occur then "h" the tightest fitting rectangle is  $\epsilon$ -accurate.

It means we see at least one point in each of the strips and then the tightest fitting rectangle is actually gonna be very close to the true rectangle.



We have to add up these  $\epsilon$ s over four to get the true error and that's why we choose  $\epsilon/4$  because we have to add up all the bad events.

Claim: if we choose "m" random samples of  $\Pr[B_1]$ ?

$$\Pr[B_1] < (1 - \epsilon/4)^m$$

$$\Pr[B_1 \cup B_2 \cup B_3 \cup B_4] < 4 \cdot (1 - \epsilon/4)^m < \delta.$$

$$\begin{aligned}
 4 \cdot \left(1 - \frac{\epsilon}{4}\right)^m &\leq \delta \\
 \left(1 - \frac{\epsilon}{4}\right)^m &\leq \frac{\delta}{4} \\
 e^{-\frac{\epsilon m}{4}} &\leq \frac{\delta}{4} \\
 -\frac{\epsilon m}{4} &\leq \log\left(\frac{\delta}{4}\right) \\
 \Rightarrow m &\geq \frac{4 \cdot \log\left(\frac{\delta}{4}\right)}{\epsilon}
 \end{aligned}$$

0 5

l+x ≈ e<sup>x</sup>  
1-x ≈ e<sup>-x</sup>

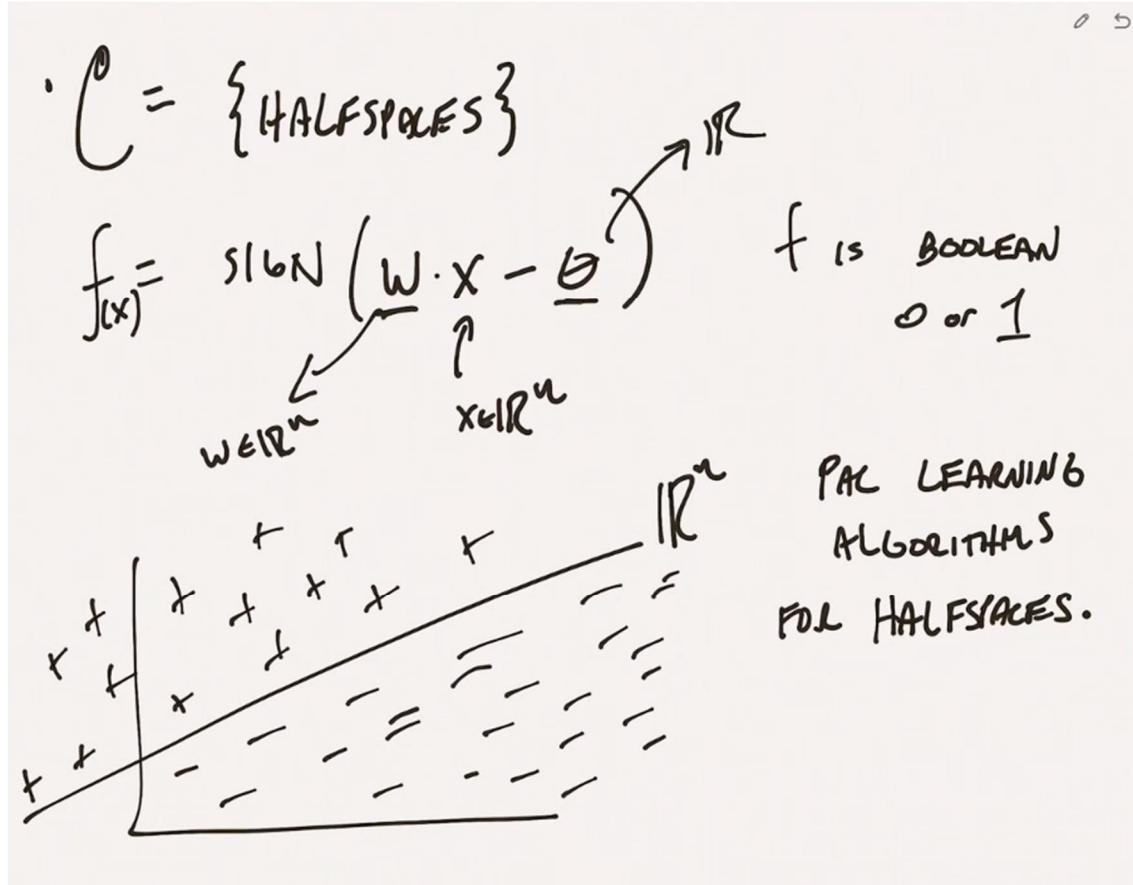
↳ highest rect, will be  $\epsilon$ -accurate w prob  $\geq 1-\delta$

If we choose "m" to be polynomial:

$$4 \cdot (1 - \epsilon/4)^4 < \delta \rightarrow e^{(-\epsilon \cdot m/4)} < \delta/4$$

$$m > 4 \cdot \log(\delta/4)/\epsilon$$

And then we know the highest fitting rectangle will be an  $\epsilon$ -accurate hypothesis with probability at least  $1 - \delta$ .

Halfspaces

Another interesting function to look at is the class "C" = {HALFSPACES}

A halfspace is a function that takes in  $\{x_1, x_2, \dots, x_n\}$  and outputs the sign of the product of  $w \cdot x - \theta$ :

$f = \text{SIGN}(w \cdot x - \theta)$ , where  $w$  is a vector in  $R^n$  and  $\theta$  is a scalar.

"f" is a boolean that outputs a boolean 0 (negative) or 1 (positive).

$w$  is an unknown weight vector and  $\theta$  is usually also unknown.

Geometrically we can think this  $n$ -dimensional euclidean space into two half spaces. So there are the points "X" where we say the inner product is positive or negative.

These halfspaces are used in countless tolos in various applications and we would like to come up with PAC learning algorithms for half-spaces.

## ONE APPROACH FOR LEARNING HALFSPACES

 $w \in \mathbb{R}^n$  $\theta \in \mathbb{R}$ IS UNKNOWN  
UNKNOWN

$$f = \text{SIGN}\left(\sum_{i=1}^n w_i x_i - \theta\right)$$

GIVEN DRAWS FROM  $D$   $(x, f(x))$   
 $x \uparrow$ 

$$(01010, \text{pos}) \rightarrow w_2 + w_4 > \theta \quad w_i \in \mathbb{Z}_{\geq 0}$$

$$(0110, \text{neg}) \rightarrow w_2 + w_3 \leq \theta \quad \text{SOME BOUNDED RANGE.}$$

EACH LABELED EXAMPLE  $\rightarrow$  LINEAR INEQUALITY.SYSTEM OF LINEAR INEQUALITIES.  
CAN WE FIND A CONSISTENT HYPOTHESIS?

GENERAL-PURPOSE TOOL CALLED LINEAR PROGRAMMING

In n-dimension things are a little bit more complicated and we are gonna find out an approach.

“w” is unknown.

“θ” is unknown.

The function we try to learn is:  $f(x) = \text{SIGN}(w \cdot x - \theta)$  given draws from “D”.

For example:

$$(01010, +) \rightarrow \text{This implies } w_2 + w_4 > \theta$$

$$(0110, -) \rightarrow \text{This implies } w_2 + w_3 < \theta$$

Each labeled example corresponds to a linear inequality. So, we get a system of linear inequalities. The big question is can we come up with some kind of consistency hypothesis although there might be infinite number of halfspaces: let's assume all of the  $w_i$  are integers in some bounded range. We need to find a consistent hypothesis.It turns out there is a general purpose tool called linear programming and linear programming can be used to solve general systems of linear inequalities and these algorithms that solve linear programs are known to run in polynomial time. Given an algorithm that will learn halfspaces by applying this sort of sophisticated tool for linear programming it would give us some solution for these  $w_i$  that would be consistent with our training set and then we could use our consistency analysis to conclude that there is at least an efficient algorithm for learning at least this class of halfspaces.

We will come up with algorithms for learning halfspaces under certain conditions that aren't really quite as complicated as algorithms for linear programming and we will begin to investigate sort of simple algorithms that we can use for concept classes.

## Cross-Validation

Introduction

# CROSS-VALIDATION

- HOLD-OUT APPROACH FOR TESTING/APPROXIMATING THE TRUE ERROR OF A CLASSIFIER
- LET'S ASSUME CLASSIFICATION; SO HYPOTHESIS  $h$  IS GOING TO OUTPUT  $\{0, 1\}$   
 $\{-1, +1\}$  VALUES.

"Hold-out"

1. LEAVE SOME PART OF TRAINING SET OUT DURING TRAIN TIME.
2. TEST CLASSIFIER ON THIS HELD OUT SET.

We're going to talk about how to test what the true error of a model classifier you've built is.

That is to say, you have a bunch of data you've run some machine learning procedure on it to build some classifier and you would like to know what is the true error of that classifier for future examples.

We are going into the detail of a technique called cross-validation.

Hold-out approach for testing/approximating the true error of a classifier.

Let's assume classification, so hypothesis " $h$ " is going to output  $\{0, 1\}$ .

"Hold-out" approach leaves some part of the training set out during train time and then we test the classifier on this held out set.

Let's look at some tools from probability to figure out how confident we are that the estimate from our hold-out is good or close to the true error.

Markov's inequality

1st INEQUALITY MARKOV'S INEQUALITY

let  $X$  be R.V. THAT TAKES ON ONLY POS VALUES.

$$\Pr[X \geq k \cdot E[X]] \leq \frac{1}{k}$$

CHEBYSHEV'S INEQUALITY

$$\text{VARIANCE}(X) = E[(X - E[X])^2] \quad \underline{E[X] = \mu}$$

$$\sqrt{\text{VARIANCE}(X)} = \text{STANDARD DEVIATION}(X) = \sigma$$

$$\Pr[|X - \mu| > t \cdot \sigma] \leq \frac{1}{t^2}$$

The first inequality of Markov's inequality: let  $X$  be a random variable (RV) that takes on only positive values.

$\Pr(X > K \cdot E(X)) < 1/K$ ; the probability of  $X$  being greater than the expected value or mean value of  $X$  is at most one over  $K$ . This is not a difficult inequality to prove. It is another way to talk about averages.

Another inequality Chebyshev's inequality:

VARIANT( $X$ ) =  $E[(X - E(X))^2]$ . How much does it deviate from its mean.

$E(X) = \mu = \text{mean}$ .

SQRT(VARIANT( $X$ )) = STANDARD DEVIATION ( $X$ ) =  $\sigma$

$\Pr[|X - \mu| > t \cdot \sigma] < 1/t^2$

The probability that  $X$  deviates from its average by more than "t" times its standard deviation is less than or equal to one over "t" squared.

We understand that " $\mu$ " is the mean and " $\sigma$ " is the variance.

Chernoff bound

## CHERNOFF BOUND

$$X_1, X_2, \dots, X_n \quad E[X_i] = p$$

$$S = \sum_{i=1}^n X_i \quad \mu = E[S] = p \cdot n$$

$$E[X_1 + \dots + X_n] = p \cdot n$$

$$\Pr[S > \mu + \delta n] \leq e^{-2n\delta^2}$$

$$\Pr[S < \mu - \delta n] \leq e^{-2n\delta^2}$$

$$\Rightarrow \Pr[|S - \mu| > \delta n] \leq 2 \cdot e^{-2n\delta^2}$$

Let's take a look at the Chernoff Bound.

Let's assume that I have a random set of independent variables and each have a mean.

$X_1, X_2, \dots, X_n$  and  $E(X_i) = p$

Then, we are going to form this quantity "S" which is going to be the sum from "i" equals 1 to n.

What is the expectation of this sum?

$$M = E(S) = p \cdot n$$

The Chernoff bound says that the probability that "S" is greater than  $\mu + \delta \cdot n$  is less or equal to  $e^{-2 \cdot n \cdot \delta^2}$ .

When we have a bunch of independent random variables each one's mean is some value and we take the sum of them, we would expect the sum to be about each mean times n.

The probability that you deviate from mu is actually exponentially small in the quantity "n".

If we look at this bound the probability we deviate by more than  $\delta \cdot n$  is exponentially small in "n". Depending on our choice of  $\delta$  we are going to set different bounds for these probabilities.

This will help us estimate the true error of a classifier.

APPLY THE CHERNOFF BOUND TO THE CASE  
OF ESTIMATING THE TRUE ERROR OF A  
CLASSIFIER

HOLD-OUT SET  $S$  WE'LL SAY  $|S|=n$   
FIX  $h$  (GENERATED USING SOME WD TRAINING SET)  
RECALL  $D$ ,  $S$  IS A SAMPLE DRAWN FROM  $D$   
INDEPENDENT OF TRAINING SET

$$Z = \Pr_{x \sim D} [h(x) \neq c(x)]$$

↓ UNKNOWN FUNCTION TRYING TO LEARN

Let  $X_i$  be r.v. EQUALS 1 if  $h$  is INCORRECT  
ON THE  $i^{\text{th}}$  element  
of  $S$ .  
0 if  $h$  is CORRECT ON THE  $i^{\text{th}}$   
element of  $S$ .

Let's apply the Chernoff bound to the case of estimating the true error of a classifier.

Again we're going to have our hold-out set and we will call it "S".

We will say the size of "S" is equal to "n".

Now we fix a classifier "h" which is generated using some training set totally independent from "S".

We're gonna define some random variables but recall that there is some distribution "D" for which we are generating training points and that "S" is a sample drawn from "D" independent of the training set.

Let's say we are trying to learn an unknown function "c" with a classifier "h" with PAC learning.

"h" is the classifier we've generated and we want to understand its true error when we choose a random "X" from entire "D".

The true error of the classifier ("Z") is the random variable that we are trying to estimate.

Let "Xi" be the random variable equals 1 if "h" is incorrect on the "i" element of "S" and equals 0 if "h" is correct on the "i" element of "S".

$$X_1, \dots, X_n \quad x_i = \begin{cases} 1 & \text{if } h \text{ is incorrect on } i^{\text{th}} \text{ element} \\ 0 & \text{otherwise.} \end{cases}$$

$$S = \sum_{i=1}^n x_i \quad E[S] = n \cdot p \leftarrow \text{true error of } h.$$

$$p = E[x_i] = E[x_1] = E[X_n]$$

$$\Pr[|S - n \cdot p| > \delta n] \leq 2e^{-2n\delta^2}$$

(RECALL  $p$  is TRUE ERROR OF CLASSIFIER  $h$ ).

$$\Pr[|S - n \cdot p| > .1n] \leq 2e^{-\frac{2n}{100}}$$

How LARGE to choose  $n$  BEFORE THIS QUANTITY BECOMES SMALL?

$$e^{-\frac{2n}{100}} < \frac{\alpha}{2}$$

$$\frac{-2n}{100} < \log\left(\frac{\alpha}{2}\right) \Rightarrow n > 50 \cdot \log\left(\frac{\alpha}{2}\right)$$

IF  $|S - n \cdot p| \leq .1n$   
 $\Rightarrow$  ERROR RATE ON  $S$  IS WITHIN .1 OF TRUE ERROR RATE.

Now we have these random variables:  $X_1, \dots, X_n$  from "S" which are actually points of the hold-out set.

$x_i = 1$  if it is correct on the "i" element of "S", 0 otherwise.

$$S = \sum(x_i)$$

$E(S) = n \cdot p$ , where "p" is actually the true error of "h" because "p" is  $E(x_i)$  and  $x_i = 1$  if incorrect and 0 otherwise and we want to output "1" if our hypothesis is incorrect on that particular point, so "p" is actually the true error of "h".

So, according to Chernoff, the probability that "S" deviates from its mean ( $n \cdot p$ ) is greater than  $\delta \cdot n$  corresponds to  $2e^{-2n\delta^2}$ .

Let's say that  $\delta = 0.1$ . Then we want to set up a large enough data set to bound the true error.

If we want the probability of failure to be less than " $\alpha$ " and we want to be that confident that our estimate is actually within 0,1 times n, then we need "n" to be 50 times  $\log(\alpha/2)$ .

If  $|S - n \cdot p| < 0.1 \cdot n$  this means that error rate on "S" is within 0,1 of true error rate.

RECAP:

We wanted our estimate to be within 0,1 and took  $\delta = 0.1$ . That means we wanted the error rate on the training set to be within 0,1 of the true error of our classifier.

We want whatever the error rate we obtain in this hold-out set "S", needs to be within 0,1 of the true error rate and I want to be alpha confident in order to do that.

Thanks to Chernoff we know that we should take "n" number of samples of our hold-out set "S" greater than  $50 \cdot \log(\alpha/2)$ . By this bound we know the probability that our true error on our hold-out set "S" is larger than 0,1 from the true error rate (the failure probability) will be at most  $\alpha$ .

In other words, with probability  $1 - \alpha$ , the error rate of our hold-out set "S" will be within the 0,1 of the true error of the classifier. So we will prove that using this hold-out set and validating we are sure that the validation of this small set "S" corresponds to the global error rate by less than 0,1.

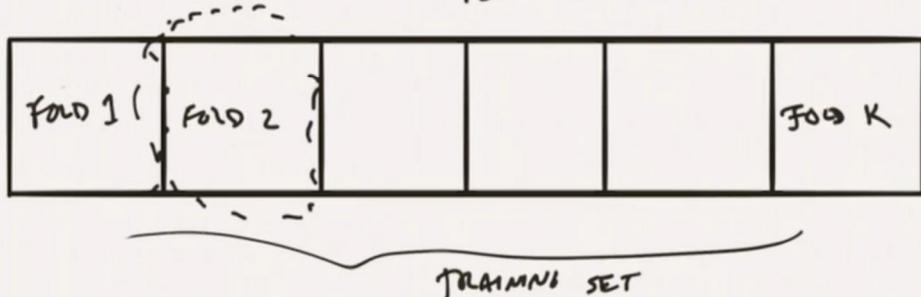
### Hold-out set

Hold-out set is somewhat expensive

- DATA IS EXPENSIVE
- IF WE WANT TO TRY OUT MULTIPLE METHODS FOR GENERATING CLASSIFIERS, WE QUICKLY LOSE CONFIDENCE IN OUR ESTIMATES.

### CROSS-VALIDATION

- DRAW USWEG FOLDS 2...FOLD K
- TEST ON FOLD 1



So the hold-out set is somewhat expensive because of several reasons:

- Labelled data is expensive or difficult to obtain. Data that we will not use to build our controller.
- If we want to try out multiple methods for generating classifiers, we lose confidence in our estimates. Every time we use a classifier and if we want to use the hold-out set to test its true accuracy we have to add up all of these probabilities of failure. As we try new multiple methods the number of "n" increases and so the error rate " $\alpha$ ".

Understanding the best way to repeatedly use a hold-out set or repeatedly use data that we've gained in order to have confidence that we understand the true error of many classifiers we generate.

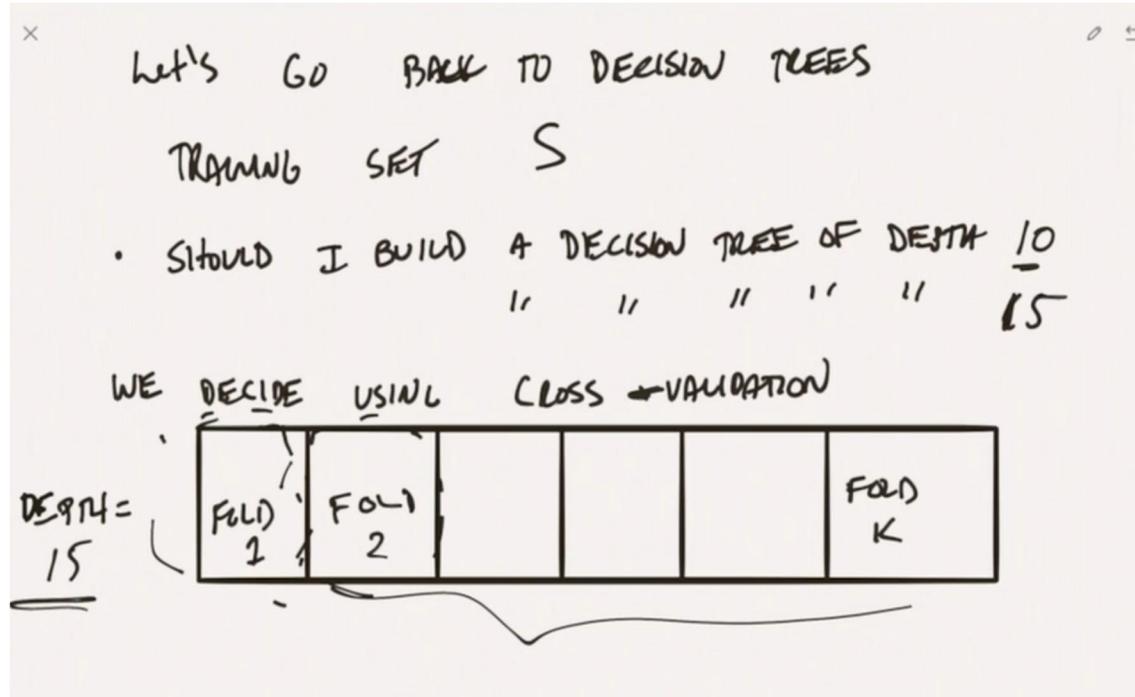
There is a technique used most often to try and solve this problem is something called cross-validation which is that it works in practice.

The idea behind cross-validation is to take our entire training set and break it up into folds: fold 1, fold 2, ..., fold k. This is our entire training set and we will use it both to train a classifier and at the same time estimate its true error.

- Train using fold 2, ..., fold k
- Test on fold 1 and get true error.

Do this all the time to test on each fold and train with the rest and the average up of all the errors will be the estimate of the true error.

### Example with decision trees



Let's go back to decision trees: we have a training set "S" and we're trying to decide whether to build a decision tree of depth 10 or depth 15.

In terms of pure training error we should use 15. On the other hand maybe with value 15 we're overfitting and its true error is going to be worse.

Using cross-validation which model is best and having our folds.

- Depth 10: Fold 1, fold 2, fold k. We will test the accuracy of the decision tree on fold 1, fold 2, fold k. And we will get the average of the true error using that depth 10.
- Depth 15: Fold 1, fold 2, fold k. We will test the accuracy of the decision tree on fold 1, fold 2, fold k. And we will get the average of the true error using that depth 15.

Whichever is smaller the true error is going to be the best tree.

We should use 5 to 10 folds the useful value.

## Perceptron learning

Introduction

o s c

# PERCEPTRON ALGORITHM

- $C = \{ \text{HALFSPACES} \}$

$$f(x) = \text{SIGN} \left( \sum_{i=1}^n w_i x_i - \theta \right)$$

$w$  is UNKNOWN       $f(x) \in \{-1, +1\}$

$$x_i \in \mathbb{R}$$

Super exciting algorithm called the Perceptron algorithm used for learning half spaces and it has probable guarantees. Many machine learning packages rely on a perceptron type update rule.

Script  $C = \{ \text{HALFSPACES} \}$

$\text{HALFSPACE} = \text{SIGN}(w \cdot X - \Theta)$

" $w$ " is an unknown vector:  $\{w_1, w_2, \dots, w_n\}$

" $\Theta$ " is a scalar.

Complicated algorithm for learning halfspace

A "COMPLICATED" ALGORITHM FOR LEARNING  
A HALFSPACE

$$(X^1, +1) \rightarrow w_1 x_1^1 + \dots + w_n x_n^1 \geq \theta$$

$$(X^2, -1) \rightarrow w_1 x_1^2 + \dots + w_n x_n^2 \leq \theta$$

⋮

m training set → m inequalities

WE CAN USE LP → FIND A WEIGHT VECTOR  
 LINEAR PROGRAMMING SOLVERS      w CONSISTENT w/  
 TRAINING SET.

It is still an inefficient polynomial algorithm.

We have several vectors:  $\{X^1, +1\}, \{X^2, -1\}$ ; where  $X^i$  is a vector.

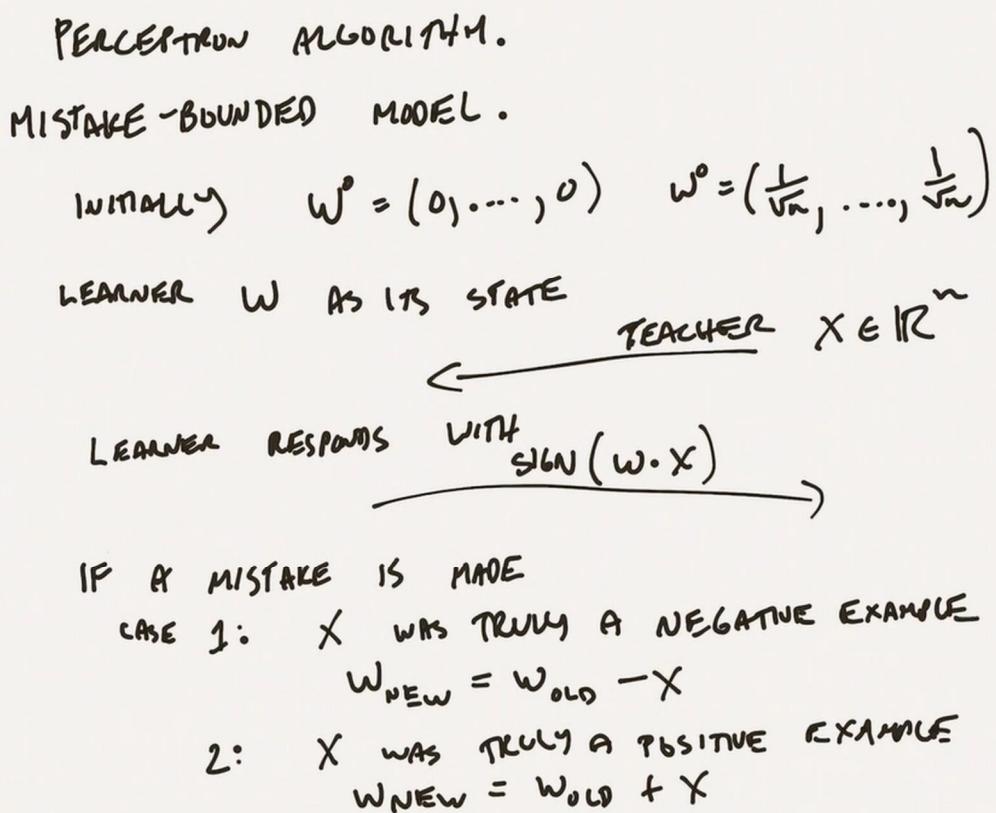
We will finally get a set of inequalities and if we have "m" points in our training set then we will get "m" inequalities where we can use linear programming to find the weight vector ("w") consistent with our training set.

This linear programming solvers are known to run in polynomial time so we can find a vector "w" consistent with all our inequalities.

Once you have an algorithm that can find a solution that is consistent with the training set we can probably prove that the solution is going to generalize well and have low true error.

We could think the case for learning half spaces is completely closed but these linear programming solvers are very expensive and complicated to run.

We're gonna learn an simpler algorithm for learning half spaces (linear programming).

Perceptron algorithm: mistake-bounded model

This algorithm is the perceptron algorithm. Let's go back to the mistake-bounded model and we will prove guarantees for this perceptron algorithm in the mistake bounded model.

We could initially choose the weight vector "w" to be all values 0s or a unit vector  $\{1, 1, 1\}$ .

So, our learner has "w" in its initial state and the teacher will present a challenge:  $\{x_1, x_2, \dots, x_n\}$ .

The learner will respond with  $\text{SIGN}(w \cdot x - 0)$ .

If a mistake is made there will be two cases:

- Case 1: "X" was truly a negative example. Update:  $w_{\text{new}} = w_{\text{old}} - x$
- Case 2: "X" was a positive example. Update:  $w_{\text{new}} = w_{\text{old}} + x$

Every time we make a mistake:  $w_{\text{new}} = w_{\text{old}} + y \cdot x$ ; where "y" is the output (the sign: +/-).

EQUIVALENT WAY TO VIEW UPDATE RULE:

EVERY TIME WE MAKE A MISTAKE

$$w_{\text{NEW}} = w_{\text{OLD}} + y \cdot x$$

↑  
LABEL

ASSUMPTIONS:

- ASSUME  $\exists w^*$ , TRUE UNKNOWN WEIGHT VECTOR  
 $\|w^*\|_2 = 1$
- ASSUME  $x$  HAS NORM 1  $\|x\|_2 = 1$
- $\theta = 0$

MAIN ASSUMPTION

: THERE EXISTS A MARGIN  $\rho$   
ALL POINTS ARE AT LEAST DISTANCE  $\rho$  FROM  $w^*$

Now we will need to make some assumptions to prove this algorithm Works:

- 1) Assume exists  $w^*$ , the true unknown weight vector to discover.  $\|w^*\| = 1$ ; it will be a unit vector. No loss as we are taking the sign and having norm 1 or not will not affect.
- 2) Assume  $X$  has norm = 1,  $\|X\| = 1$ . We will show how to handle input points with larger norm.
- 3)  $\theta = 0$ ; For simplicity but we will see how we can get rid of this assumption.

Main assumption: there exists a margin " $\rho$ ". All points are at least distance " $\rho$ " from " $w^*$ ". We have the guarantee that all the challenges that the learner gets obey this assumption even if there is a positive or negative example.

ALL POSITIVE/NEGATIVE POINTS HAVE DISTANCE  $\geq \rho$  FROM HALFSPACE.  
 (BECAUSE  $\|x\| = \|w^*\| = 1$ )  
 EQUIVALENTLY  $|\langle x, w^* \rangle| \geq \rho$

"MARGIN ASSUMPTION"

MAIN THEOREM "PERCEPTRON CONVERGENCE THEOREM"  
 IS THAT THE MISTAKE BOUND OF PERCEPTRON ALGORITHM IS  $O(\frac{1}{\rho^2})$ .

The circle corresponds to all points "X" that have norm 1. And there is a line that divides our half space.

" $w^*$ " is going to be the normal vector that separates half spaces.

All positive/negative points have distance at least " $\rho$ " from the true half space.

Because all the norms assume to be 1:  $\|X\| = \|w^*\| = 1$ .

Moreover:  $|\langle X, w^* \rangle| > \rho$ ; this is the geometric margin assumption when we carry out the inner product thanks to the points being distant.

Main theorem: PERCEPTRON CONVERGENCE THEOREM. The mistake bound of perceptron algorithm is  $O(1/\rho^2)$ . This is the order of mistakes the model will make and it is at most order  $1/\rho^2$ .

" $\rho$ " can be extremely small but in that case our perceptron algorithm could make exponentially mistakes.

This mistake bound is independent of the dimension of "X".

### INNER PRODUCT RECAP

Another example shows two vectors whose inner product is 0.

$$\vec{v} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}, \vec{w} = \begin{bmatrix} 4 \\ -4 \\ 4 \end{bmatrix}$$

$$\vec{v} \cdot \vec{w} = 0$$

$$1 \cdot 4 + 3 \cdot (-4) + 2 \cdot 4 = 0$$

Now, our vectors  $\vec{v}$  and  $\vec{w}$  are of size 3.

On the other hand, we can calculate the length of a single vector using the dot product and we apply the square root. This is sometimes called a “norm” of a vector.

$$\vec{v} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

$$\vec{v} \cdot \vec{v} = 1 + 9 + 4 = 14$$

The length is  $\|\vec{v}\| = \sqrt{14}$

This can be very useful when we need to normalize our vectors. In this case we want a vector to be of the length 1. Basically, if we have a length of our vector equal to 4 it is understandable that we need to divide it by 4. In this case, where we have  $\sqrt{14}$  and we will just divide every element of our vector with  $\sqrt{14}$  and this will be our resulting vector:

$$\vec{u} = \frac{\vec{v}}{\|\vec{v}\|} = \frac{1}{\sqrt{14}} \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

---


$$\|\vec{u}\| = 1$$

UPDATE STEP:

$$w_{\text{new}} = w_{\text{old}} + y \cdot x$$

let's say  $w$  is current state of learner  
 $w^*$  is true normal to halfspace.

CLAIM 1: ON EVERY MISTAKE  $w \cdot w^*$  INCREASES BY AT LEAST  $\rho$

CLAIM 2:  $\|w\|^2$  INCREASES AFTER EVERY MISTAKE BY AT MOST 1

QUESTION: How to obtain the  $O(\frac{1}{\rho^2})$  mistake bound given claims 1 and 2? Let  $t$  be # mistakes we've made at some point during execution

$$\begin{aligned} t \cdot \rho &\leq w \cdot w^* \leq \|w\| \cdot \|w^*\| \\ t \cdot \rho &\leq \sqrt{t} \Rightarrow t \leq \frac{1}{\rho^2} \end{aligned}$$

This is not complicated to prove, but let's remind the update step:

- $w_{\text{new}} = w_{\text{old}} + y \cdot x$

" $w$ " is the current state of the learner.

" $w^*$ " is the true normal to this halfspace.

Claim 1: On every mistake  $w \cdot w^*$  increases by at least " $\rho$ ".

Claim 2:  $\|w\|^2$  increases after every mistake by at most 1.

How do we obtain the  $O(1/\rho^2)$  mistake bound given claims 1 and 2? Let " $t$ " be the number of mistakes we made at some point during execution.

After " $t$ " mistakes the product " $w \cdot w^*$ " has increased at least " $t \cdot \rho$ ".

Proving perceptron algorithm claims

CLAIM 1  $w \cdot w^*$  INCREASES ON EVERY MISTAKE BY AT LEAST  $\rho$

$$w_{\text{new}} = w_{\text{old}} + y \cdot x$$

$$w_{\text{new}} \cdot w^* = (w_{\text{old}} + y \cdot x) \cdot w^* = \underline{w_{\text{old}} \cdot w^*} + \underbrace{y \cdot x \cdot w^*}_{\geq \rho} \quad \blacksquare$$

We reduced our problem to proving claim 1 and claim 2.

**Claim 1:** On every mistake  $w \cdot w^*$  increases by at least " $\rho$ ".

$w_{\text{new}} = w_{\text{old}} + y \cdot x$ ; being "x" the point where we made our mistake.

$$W_{\text{new}} \cdot w^* = (w_{\text{old}} + y \cdot x) \cdot w^* = w_{\text{old}} \cdot w^* + y \cdot x \cdot w^*.$$

We can assume  $y \cdot x \cdot w^* > \rho$ ; as per previous requirements of the problem as all of the points "x" were at least " $\rho$ " distance from  $w^*$ . "y" can only be +1 or -1 as this is the sign or output. Regardless as to whether "x" is a positive or negative example this quantity is going to be at least " $\rho$ " and once we update  $w_{\text{old}}$  we have increased the inner product by at least " $\rho$ ".

Remember:  $|\langle X, w^* \rangle| > \rho$ ; this is the geometric margin assumption.

Now if "x" was a negative example then  $x \cdot w^*$  would be negative, but we're multiplying it by "y" which will be negative in the mistake case ( $y=-1$ ). This way, we always increase by at least " $\rho$ ".

CLAIM 2  $\|w\|^2$  INCREASES BY AT MOST 1 ON EVERY MISTAKE.

$$\begin{aligned} w_{\text{new}} &= \|w_{\text{old}} + y \cdot x\|^2 \\ &= \|w_{\text{old}}\|^2 + 2 \cdot y \cdot \langle x, w_{\text{old}} \rangle + \|x\|^2 \end{aligned}$$

NEGATIVE

ENDS PROOF OF  
CLAIM 2.  $\square$

**Claim 2:**  $\|w\|^2$  increases after every mistake by at most 1 on every mistake.

$$W_{\text{new}} = \|w_{\text{old}} + y \cdot x\|^2 = \|w_{\text{old}}\|^2 + 2 \cdot y \cdot \langle x, w_{\text{old}} \rangle + y^2 \cdot \|x\|^2 \rightarrow$$

$$\|w_{\text{old}}\|^2 + 2 \cdot y \cdot \langle x, w_{\text{old}} \rangle + +1 \cdot +1 \quad (y \cdot y = -1 \cdot -1 \text{ and } \|x\| = 1, \text{ we are taking unit vectors}).$$

Moreover,  $2 \cdot y \cdot \langle x, w_{\text{old}} \rangle < 0$  as we made a mistake in "x" and if "y" is positive then  $x \cdot w_{\text{old}}$  should be negative and if "y" is positive then  $x \cdot w_{\text{old}}$  should be negative as we made a mistake.

$\|w_{\text{new}}\| = \|w_{\text{old}}\| + 1 - \text{negative\_value}$ ; increase is at least 1, end of proof of claim.

LOOK BACK AT SOME ASSUMPTIONS

'  $\theta = 0$  ADD A NEW FEATURE CALL IT  $x_{n+1}$

$(x_1, \dots, x_n) \rightarrow (x_1, \dots, x_n, x_{n+1})$  ← assumes set  $x_{n+1} = 1$

Let's look back at some of these assumptions:  $\theta = 0$  (the constant value). We could just subtract a bias instead of using " $\theta$ ". We add a new feature and this will allow us to add or subtract some constant term in the unknown  $w^*$ .

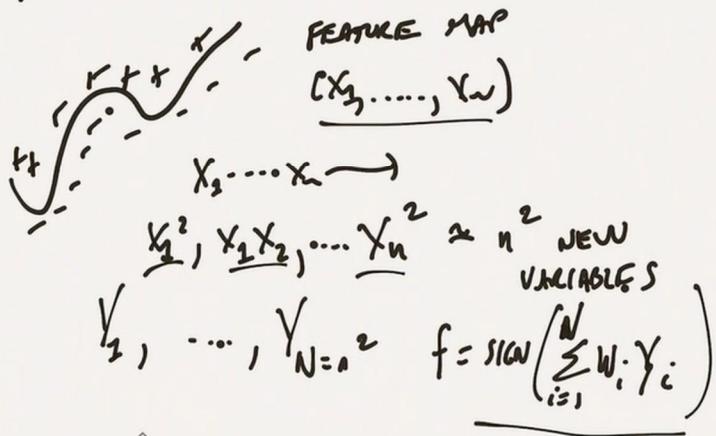
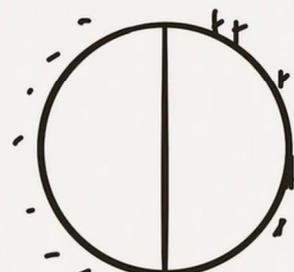
$\|x\| = 1$ ; if it is  $\|x\| = R$  then  $O(R^2/p^2)$  instead of  $O(1/p^2)$  mistake bound.

Polynomial threshold functions (generic halfspace functions)

CONSIDER POLYNOMIAL THRESHOLD FUNCTIONS  
(PTFs)

$f = \text{SIGN}(p(x))$  ←  $p$  IS A MULTIVARIATE  
POLYNOMIAL OF, LET'S  
SAY DEGREE  $d$ .

HOW CAN WE USE PERCEPTRON TO LEARN THIS  
FUNCTION CLASS?



Let's consider some polynomial threshold functions (PTFs) of the form:  $f = \text{SIGN}(p(x))$ ; where "p" is a multivariate polynomial of degree "d".

We're trying to learn "f" using the perceptron algorithm. How can we use the perceptron to learn this function?

Imagine that our data was not necessarily linear. In the linear separable case we know we have some half space. But we could envision a scenario where we have some curve and we have a bunch of positive points on one side of the curve and a bunch of negative points on the otherside.

Then, we would have a feature map (variables) that takes in "x<sub>1</sub>" through "x<sub>n</sub>" (x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub>) and we would be only interested in degree 2 polynomials ( $ax^2 + bx + c$ ) then we would map to n<sup>2</sup> new variables and "f" can be written as:

$f = \text{SIGN}(w_i \cdot y_i)$ ; this is actually a halfspace in n dimensión. So basically our polynomial of degree 2 in "n" dimensión can be analyzed as a halfspace of dimensión n<sup>2</sup>. We could actually run the perceptron algorithm to learn this halfspace in higher dimensions applying the transformarions to the feature map.

LEARNING PTFs OF DEGREE d IS EQUIVALENT  
TO LEARNING HALFSPACES IN n<sup>d</sup> DIMENSIONS.

- RUN-TIME  
JUST CONSIDER THE FEATURE MAP TAKES  
TIME n<sup>d</sup>.
- WHAT IS Margin IN THIS n<sup>d</sup> DIMENSIONAL SPACE?  
(MAY BE COSTLY)

Learning PTFs of degree "d" is equivalent to learning halfspaces in "n<sup>d</sup>" dimensions.

We have to consider the margin and error we will get now.

LEARNING PTFs OF DEGREE  $d$  IS EQUIVALENT  
TO LEARNING HALFSPACES IN  $n^d$  DIMENSIONS.

- RUN-TIME JUST COMPUTING THE FEATURE MAP TAKES TIME  $n^d$ .

- WHAT IS MARGIN IN THIS  $n^d$  DIMENSIONAL SPACE?  
(MAY BE COSTLY)

WE CAN SAVE ON THE RUNNING TIME

WE CAN SAVE DRAMATICALLY USING SOMETHING  
CALLED THE KERNEL TRICK!

It turns out for the running time we can really win saving on the running time dramatically using the "KERNEL TRICK".

We're gonna describe the Kernel perceptron and in some sense, we will bring the running time back to linear time despite working in this much higher dimensional space.

### Kernel perceptron

#### KERNEL PERCEPTRON

$x \leftarrow$  INPUT  $\varphi(x)$  TO BE THE IMAGE OF  $x$   
IN THE FEATURE SPACE.

$$x \in \mathbb{R}^n \quad \varphi(x) \in \mathbb{R}^d$$

$K(\underline{x}^1, \underline{x}^2)$  AND OUTCOMES  $\langle \varphi(x^1), \varphi(x^2) \rangle$

AND LET'S ASSUME  $K(x^1, x^2)$  IS EASY TO COMPUTE.

✓ KERNEL FUNCTION.

If we have an input "x", and we will denote:

$\varphi(x)$  to be the image of "x" in the feature space.

$x \in \mathbb{R}^n$ , then  $\varphi(x) \in \mathbb{R}^{n^d}$ .

Now imagine that we had a function "K" that takes in point " $x^1$ " and point " $x^2$ " and outputs the inner product of  $\varphi(x^1)$  and  $\varphi(x^2)$ :

$$K(x^1, x^2) = \langle \varphi(x^1), \varphi(x^2) \rangle$$

And now let's assume that "K" is easy to compute. "K" is the Kernel function.

KERNEL PERCEPTRON (WANT TO WORK IN  $\mathbb{R}^{n^d}$ )  
 $w = 0^{n^d}$

LET'S ASSUME WE MADE A MISTAKE ON POINT  $\underline{x^1}$

$\underline{w_{\text{new}}} = \underline{w_{\text{old}}} + \underline{y \cdot \underline{\varphi(x)}}$

WE NEED TO EVALUATE  $\underline{w_{\text{new}}} \cdot \underline{\varphi(x^2)}$

$K(x^1, x^2) = \langle \underline{\varphi(x^1)}, \underline{\varphi(x^2)} \rangle = y \cdot \underline{K(x^1, x^2)}$

Kernel perceptron description:

Remember in the perceptron algorithm we started off with some "w" (first iteration the all zeros vector). So now we want to work in  $\mathbb{R}^{n^d}$ .

Our initial point "w" is the all zeros vector and we make a mistake in " $x^1$ ".

$$w_{\text{new}} = w_{\text{old}} + y \cdot \varphi(x) = 0 + y \cdot \varphi(x)$$

Let's keep running this perceptron algorithm some time, and we made a mistake in point " $x^2$ ". We need to evaluate " $w_{\text{new}} \cdot \varphi(x^2)$ " in order to know if we are now making a mistake or not.

But remember this  $w_{\text{new}} = w_{\text{old}} = y \cdot \varphi(x^1)$  in " $x^1$ ", so eventually we need to compute:

$$\langle y \cdot \varphi(x^1), \varphi(x^2) \rangle, \text{ and by definition of the Kernel function:}$$

$$K(x^1, x^2) = \langle \varphi(x^1), \varphi(x^2) \rangle$$

We can reduce the inner product above as:  $y \cdot K(x^1, x^2)$ .

We are taking our kernel function "K" and we are evaluating it using these inputs and multiplying by a scalar. This is efficient to calculate as opposed to computing the inner product of "n" dimensional vectors in " $n^d$ ".

$$w = \sum_{i=1}^t y^i \cdot \phi(x^i) \in \mathbb{R}^{n^d}$$

NEED TO COMPUTE  $\langle w_{t+1}, \phi(x^{t+1}) \rangle$

$$\sum_{i=1}^t y^i \cdot \langle \phi(x^i), \phi(x^{t+1}) \rangle$$

$\boxed{K(x^i, x^{t+1})} \rightarrow$  EFFICIENTLY COMPUTABLE.

We can keep going like this (always assuming we start with all-zeroes vector), and "w", after "t" mistakes, will end up being:

$$w_{t+1} = \sum_{i=1}^t y^i \cdot \phi(x^i) = \sum_{i=1}^t y^i \cdot \langle \phi(x^i), \phi(x^{t+1}) \rangle = \sum_{i=1}^t y^i \cdot K(x^i, x^{t+1})$$

And "K" is efficiently computable.

We can run the perceptron implicitly via this kernel function. We can run this perceptron update in this much higher dimensional space and the only thing that we need to keep track of is our inner products of vectors in this higher dimensional space via a kernel function.

# EXAMPLE OF A SIMPLE KERNEL FUNCTION

(CONSIDER DEG-2 POLYNOMIAL THRESHOLD FUNCTIONS)

$$\varphi(x_1, \dots, x_n) = (\underbrace{x_1 x_2, x_1 x_3, \dots, x_{n-1} x_n}_{x_i, x_j \in \mathbb{R}^n}, x_n^2).$$

$$K(x, z) = \langle \varphi(x), \varphi(z) \rangle = (z_1^2 + x_1 z_1, z_1 z_2, \dots, z_n^2)$$

$$= \sum_{i,j} x_i x_j z_i z_j = \underbrace{\left( \sum_{i=1}^n x_i z_i \right)}_{(x \cdot z)^2} \cdot \underbrace{\left( \sum_{j=1}^n x_j z_j \right)}_{K(x, z)}$$

Let's see an example of a Kernel function.

Consider the case  $d = 2$  polynomial threshold ( $a \cdot x^2 + b \cdot x + c$ ).

$$\varphi(x_1, x_2, \dots, x_n) = (x_1 \cdot x_1, x_1 \cdot x_2, \dots, x_{n-1} \cdot x_n, x_n \cdot x_n)$$

$$\varphi(z_1, z_2, \dots, z_n) = (z_1 \cdot z_1, z_1 \cdot z_2, \dots, z_{n-1} \cdot z_n, z_n \cdot z_n)$$

$K(x, z) = \langle \varphi(x), \varphi(z) \rangle$  = Inner product of "x" and "z" squared.

$$K(x, z) = (\langle x, z \rangle)^2$$

## OTHER KERNELS

$$K(x, z) = \underline{(x \cdot z + c)}^2$$

$$\mathcal{L}(x) = (\underline{x_1^2}, \dots, \underline{x_n^2}, \underline{\sqrt{2c} x_1}, \dots, \underline{\sqrt{2c} x_n}, \underline{c})$$

$$\text{GAUSSIAN KERNEL} \approx K(x, z) \approx e^{-\|x - z\|_2^2}$$

RADIAL BASIS KERNEL

Understanding a very simple example of something called Kernel methods. We could teach a whole course on Kernel methods. It's a huge area of study and there are many interesting types of kernel functions you can see to get different types of Kernel perceptron.

We can see some other examples of kernel method. The more expressive our kernel maps get the more expressive feature space.

## Linear regression

Introduction

# LINEAR REGRESSION

- CLASSIFICATION

$$(x, f(x)) \uparrow \{0, 1\}$$

- HALFSPACES
- DECISION TREES

- REAL-VALUED LABELS  $(x, y) \quad y \in \mathbb{R}$

Linear regression involves fitting a line to data and it is a core task in multiple fields including machine learning.

We are going to start with the basics of linear programming.

Previously we've been analyzing learning algorithms for classification.

Now we are given examples "x" and we want to learn "f(x)" where it has taken values {0, 1}.

We've studied learning function classes such as half-spaces and decision-trees.

We are now getting examples (X, Y) and Y is a Real value, no longer a bool.

o s c

$X, Y$  two RANDOM VARIABLES

WE WANT TO PREDICT THE VALUE/LABEL  
WE GET TO SEE  $X$ .

- WE WANT TO PREDICT  $Y$ ; WE DON'T SEE  $X$   
 $(X, Y)$   $\sim D$  OPTIMAL GUESS FOR  $Y$  IS  $E[Y]$
- MEASURE OUR LOSS USING SQUARE-LOSS:  $(\text{PREDICTION} - Y)^2$
- WE OBSERVE  $X$  WE WANT TO PREDICT  $Y$   
OPTIMAL PREDICTION  $\hat{E}[Y|X] = f(X)$   
REGRESSION FUNCTION

OBSTACLE:  $f(x)$  COULD BE UNKNOWN OR VERY HARD TO COMPUTE!

Let's imagine now we have two random variables. Let's say we want to predict the value "Y" (label) and we get to see "X".

Let's say we want to predict:  $(x, y)$  that belong to a Distribution "D". We don't see X.

The optimal guess for "Y" is the average of "Y" or  $E[Y]$ .

Measure our loss using Square-loss:  $(\text{PREDICTION} - Y)^2$ .

A more reasonable scenario is that we get to see "X" and using that information we want to predict "Y". Optimal prediction is going to be average "Y" condition the expected value of "X" or  $E[Y|X] = f(x)$  this is the regression function we should output. The only obstacle in this guess is that  $f(cx)$  could be unknown or very hard to compute.

LINEAR REGRESSION ASKS THE FOLLOWING QUESTION:

GIVEN  $X$  WHAT LINEAR FUNCTION OF  $X$   
SHOULD WE USE TO PREDICT  $Y$ ?



• WE WANT TO LEARN  
COEFFICIENTS  $\beta_0$  AND  $\beta_1$

TO MINIMIZE

$$\mathbb{E} [(y - (\beta_0 + \beta_1 x))^2]$$

$(x, y) \sim d$

Linear regression asks the following question: Given "X" what linear function of "X" should we use to predict "Y"? Which line should we use so it is simple and easy to compute? Which line does fit the data with respect to square loss?

We really want to learn coefficients  $B_0$  and  $B_1$  to minimize:  $E[(Y - (B_0 + B_1 \cdot X))^2]$ . How do we find  $B_0$  and  $B_1$ ?

$$\times \frac{1}{m} \sum_{j=1}^m (y^j - \beta_0 - \beta_1 x^j) = 0$$

$$\frac{1}{m} \sum_{j=1}^m (y^j - \beta_0 - \beta_1 x^j)(x^j) = 0 \quad \text{SOLVE FOR } \beta_0$$

$\beta_0$  IN TERMS OF  $\beta_1$ ,  $\bar{y}$ ,  $\bar{x}$

$\beta_0 = \bar{y} - \beta_1 \bar{x}$

$\beta_1$  WILL NOT INVOLVE  $\beta_0$

$$\beta_1 = \frac{\bar{xy} - \beta_0 \bar{x} - \beta_1 \bar{x^2}}{\bar{x^2} - (\bar{x})^2}$$

$$\bar{xy} - \bar{x} \bar{y} + \beta_1 (\bar{x})^2 - \beta_1 \bar{x^2} \quad \beta_1 = \frac{\text{cov}(x, y)}{\text{var}(x)}$$

$$\text{cov}(x, y) = E[x \cdot y] - E[x] \cdot E[y]$$

So, we are going to draw a training set "S" of size "m".

$(X_1, Y_1), \dots, (X_n, Y_n)$  "Simple linear regression" scenario.

We want to find  $B_0$  and  $B_1$  that minimize our square loss.

From Calculus we know that taking the derivative with respect to  $B_0$ ,  $B_1$  and set them equal to 0 we are searching for the optimal values.

$$L = \frac{1}{m} \cdot \sum_{j=1}^m (Y^j - B_0 + B_1 \cdot X^j)^2$$

Let's compute the partial derivative  $dL/dB_0$ :

$$(1) \frac{dL}{dB_0} = \frac{1}{m} \cdot \sum_{j=1}^m (Y^j - B_0 + B_1 \cdot X^j) \cdot (2 \cdot -1) = 0$$

$$(2) \frac{dL}{dB_1} = \frac{1}{m} \cdot \sum_{j=1}^m (Y^j - B_0 + B_1 \cdot X^j) \cdot (2 \cdot -X^j) = 0$$

Let's solve these equations for  $B_0$  using (1).

But let's consider  $E[Y]$  and  $E[X]$  as they appear in formulas.

$$B_0 = E[Y] - B_1 \cdot E[X]$$

Taking equation (2) and replacing with the above result we can isolate B1.

The result of B0 nad B1 are outstanding.

### Regression with multiple variables

REGRESSION WITH MULTIPLE VARIABLES.

$X \in \mathbb{R}^n$   $y \in \mathbb{R}$  FITTING A LINE TO  $n$ -dimensional DATA.

$X \leftarrow$  Matrix  $X$  is going to be an  $m \times n$  matrix  

- $m$  rows  $\leftarrow$  EACH ROW IS EQUAL TO  $X^i$  DRAWN FROM D
- $n$  columns  $\leftarrow$  EACH POINT IS IN  $\mathbb{R}^n$

$y \in \mathbb{R}^m$   $\leftarrow$  LABELS FOR THESE  $m$  POINTS.

GOAL: FIND A VECTOR  $w \in \mathbb{R}^n$   $\|X \cdot w - y\|_2^2$

$$X^T = X_1^T, \dots, X_n^T \quad y^T \\ (y - (X_1^T w_1 + \dots + X_n^T w_n))^2$$

Let's look at regression with multiple variables. Unfortunately there is no really nice analogy we can make with this formula in the general setting world.

Now "X" is a vector and "Y" is a scalar.

We want to fit a line to n-dimension data.

Granted from calculus taking the derivative and making it equal to zero you're going to get a critical point. As we are getting closer and closer to the result we can take for granted that the local minimum is the good one.

Let's consider "X" as a Matrix where "X" is going to be an  $m \times n$  matrix ("m" rows based on the training set, "n" columns based on the n-dimension of variables).

- Each row will be equal to  $X^i$  drawn from D.
- Each column is each point in our space  $\mathbb{R}^n$ .

Our goal is to define a vector W that belongs to  $\mathbb{R}^n$  that minimizes:

$$\| [X] \cdot W - \bar{Y} \|^2$$

Here we are looking for  $W = \{w_1, w_2, \dots, w_n\}$ , "n" variables.

$\min_w \|Xw - y\|_2^2$

$Xw$  is a vector in the span of the columns of  $X$ .

VECTOR  $y - Xw$  is orthogonal to  $X$

$$\underline{X^T(y - Xw) = 0}$$

$$X^Ty - X^TXw = 0 \quad \text{1st ISSUE:}$$

$$X^Ty = X^TXw \quad \text{WHAT IF}$$

$$(X^TX)^{-1} X^Ty = w \quad \text{WHAT IS THE RUNNING}$$

$$\text{TIME FOR COMPUTING } w? \quad \text{"PSEUDO-INV"}$$

NORMAL EQUATIONS  $\rightarrow O(n^3 + m \cdot n^2)$

Now we have a nice compact problem to minimize the square-loss finding values for "W".

This time using derivatives can be expensive and we will apply a simple geometric property to get to the same result.

" $X \cdot W$ " is a vector in the span of the columns of " $X$ ". Which points should we pick in the span of " $X$ " to best approximate " $Y$ "?

The optimal point in our span is the one closer in distance to " $Y$ ".

Vector:  $Y - X \cdot W$  is orthogonal to " $X$ ".

Another way of writing that  $Y - X \cdot W$  is orthogonal to " $X$ " is:  $X^T(Y - X \cdot W) = 0$  (inner product is 0).

From this equation above we get to the final form and find out the optimal point ( $X \cdot W$ ) in our span:

$$(X^T \cdot X)^{-1} \cdot X^T \cdot Y = W$$

There are several issues:

- 1) What if  $(X^T \cdot X)^{-1}$  is not invertible? This is OK and we will end up with something called pseudo-inverse instead of a true inverse. It means there are multiple choices of " $W$ " that minimize square loss.
- 2) Running time for computing " $W$ " as calculating the inverse is time-consuming. We need a linear or polynomial running times and this might not be the case.

The order of this solution is  $O(n^3 + m \cdot n^2)$ .

This algorithm for solving linear regression is good as we don't need to take actually derivatives but we will need special algorithms like the gradient descent to improve it.

### Maximum likelihood for regression

MAXIMUM LIKELIHOOD

ASSUMPTION "SIMPLE LINEAR REGRESSION CASE"

X ; ASSUME  $y = \beta_0 + \beta_1 x + \epsilon$  ← RANDOM NOISE VARIABLE  
 $\epsilon \sim N(0, \sigma^2)$

DRAWN  $x^1, \dots, x^m$  AND  $y^1, \dots, y^m$

WE WANT TO UNDERSTAND: FOR A FIXED CHOICE OF  $\beta_0$  AND  $\beta_1$  ( $\sigma^2$  IS KNOWN)  
 WHAT IS THE PROBABILITY THAT WE SEE  $(x^1, y^1) \dots (x^m, y^m)$

The last topic to cover in linear regression is the MAXIMUM LIKELIHOOD.

In this scenario we are gonna make an assumption back in the "SIMPLE LINEAR REGRESSION CASE".

We're gonna have a random variable "X" scalar and assume  $Y = \beta_0 + \beta_1 \cdot X + \epsilon$ , where  $\epsilon$  is always going to be chosen independently from a Gaussian Distribution with mean 0 and variance  $\sigma^2$ :  $N(0, \sigma^2)$ .

So imagine we draw  $X_1$  to  $X_m$  so we have  $Y_1$  to  $Y_m$ .

And we want to understand for a fixed choice of  $\beta_0$  and  $\beta_1$  and we know  $N(0, \sigma^2)$ . What is the probability so for a choice of  $\beta_0$  and  $\beta_1$  that we see  $X_1$  to  $X_m$  and  $Y_1$  to  $Y_m$ . In other words, how sure are we that we found the right function for our training set and therefore  $\epsilon = 0$ .

LET'S WRITE DOWN LIKELIHOOD FUNCTION

PROBABILITY OF SEEING TRAINING SET GIVEN A CHOICE  $\beta_0$  AND  $\beta_1$  OF OUR PARAMETERS

$$\prod_{i=1}^m p(Y^i | X^i; \beta_0, \beta_1) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(y^i - (\beta_0 + \beta_1 X^i))^2}{2\sigma^2}}$$

← LIKELIHOOD  
OF  
OUR TRAINING  
SET.

CHOOSE  $\beta_0$  AND  $\beta_1$  THAT  
MAXIMIZES THIS LIKELIHOOD

Let's write the likelihood function: it's the probability of seeing training set given choice B0 and B1 of our parameters.

$$Likelihood = \prod_{i=1}^m P(Y^i | X^i; B_0, B_1)$$

Assuming Gaussian distribution of the error  $\epsilon$ , then PDF of the Gaussian can be obtained.

We would like to choose B0 and B1 that maximizes this likelihood so we know we have the "true" line that fits all points.

INSTEAD OF DIRECTLY MAXIMIZING LIKELIHOOD

WE WILL MAXIMIZE LOG-LIKELIHOOD

$$\text{LOG } (L(B_0, B_1)) = \log \prod_{i=1}^m p(y^i | X^i; \beta_0, \beta_1)$$

$$= \sum_{i=1}^m \log(p(y^i | X^i; \beta_0, \beta_1))$$

$$= \underbrace{-\frac{m}{2} \log 2\pi - m \log \sigma}_{\text{---}} - \underbrace{\frac{1}{2\sigma^2} \sum_{i=1}^m (y^i - (\beta_0 + \beta_1 X^i))^2}_{\text{---}}$$

LEAST-SQUARES ESTIMATE  
FOR SIMPLE LINEAR REGRESSION

Instead of directly maximizing likelihood, we will maximize the logarithm likelihood, taking all advantages of logarithms properties given the Gaussian is exponential:

$$\text{Log}(L(B_0, B_1)) = \log \left( \prod_{i=1}^m P(Y^i | X^i; B_0, B_1) \right) = \left( \sum_{i=1}^m \log(P(Y^i | X^i; B_0, B_1)) \right)$$

Notice this quantity is negative so to maximize this negative quantity we need to be as close to zero as possible. No matter how we choose  $B_0$  and  $B_1$  it's not going to affect this term. But the least-square estimate for simple linear regression appears in this expression and we have two ways to interpret the coefficients in order to solve.

Interpretations for coefficients in linear regression

TWO INTERPRETATIONS FOR COEFFICIENTS IN LINEAR REGRESSION:

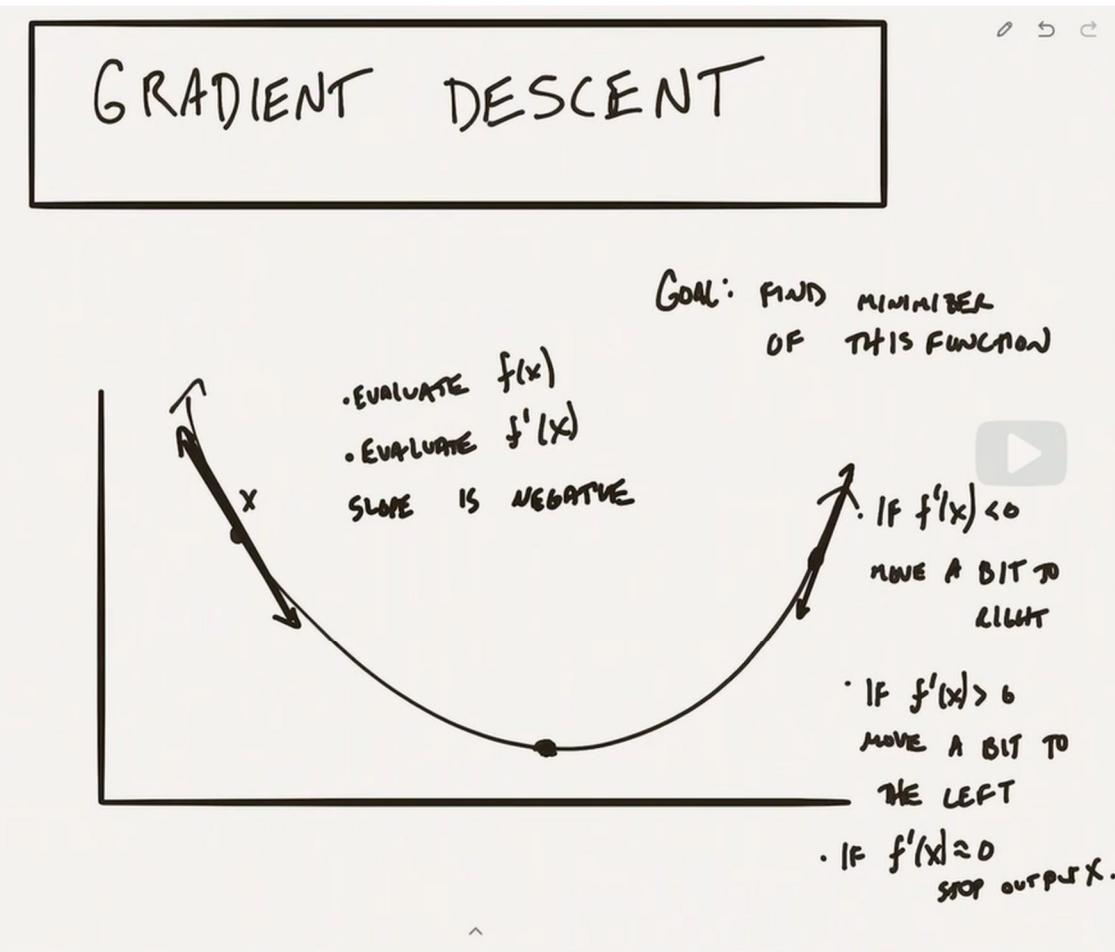
- GEOMETRIC: COEFFICIENTS OF THE LINE THAT MINIMIZES SQUARED DISTANCE FROM LINE TO OUR LABELS
- STATISTICAL: COEFFICIENTS GIVE YOU THE MAXIMUM LIKELIHOOD ESTIMATOR FOR A TRAINING SET GENERATED  $y \sim N(\beta_0 + \beta_1 x, \epsilon)$

## Interpretations:

- Geometric. Coefficient of the line that minimizes squared distance from the line to our labels (seen some slides above).
- Probabilistic/Statistical. Coefficients that give us the maximum likelihood estimator for a training set generated per the assumption so we had "Y" and "Y" was generated according to a normal distribution  $N(\beta_0 + \beta_1 \cdot X, \epsilon)$

## Gradient descent

### Introduction



Gradient descent is a algorithm and the most important tool for training complicated models today in Machine Learning.

For example, if we would like to train a neural network with thousands of parameters it results in a very complicated and high dimensional optimization problema.

If we want to find the setting of the parameters for our neural network that with fir our data pretty well, we can use an algorithm that's really simple like gradients descent, it's a simple iterative update rule, and it Works incredibly well in practice.

We're gonna see some simple examples of gradient descent and motivating theories behind it.

Imagine that we want to minimixe the above simple parábola and our goal is to find the minimizer of this function. We learners have some limited function of this particular function and we randomly pick a point somewhere in the parábola "X".

We can then evaluate  $f(x)$  and  $f'(x)$ .

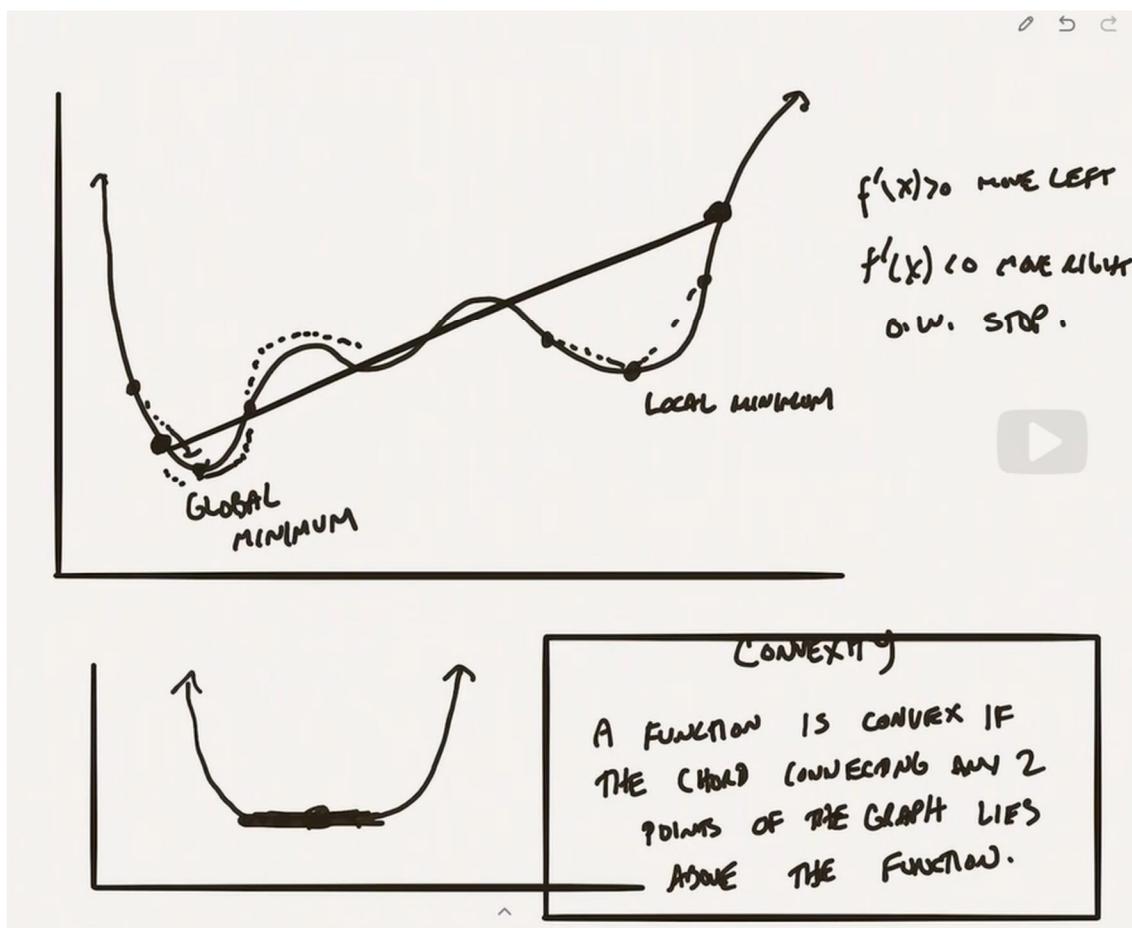
In the example we see that the derivative in point "X" will be negative as the slope is decent.

If  $f'(x) < 0$ , move a bit to the right.

If  $f'(x) > 0$ , move a bit to the left.

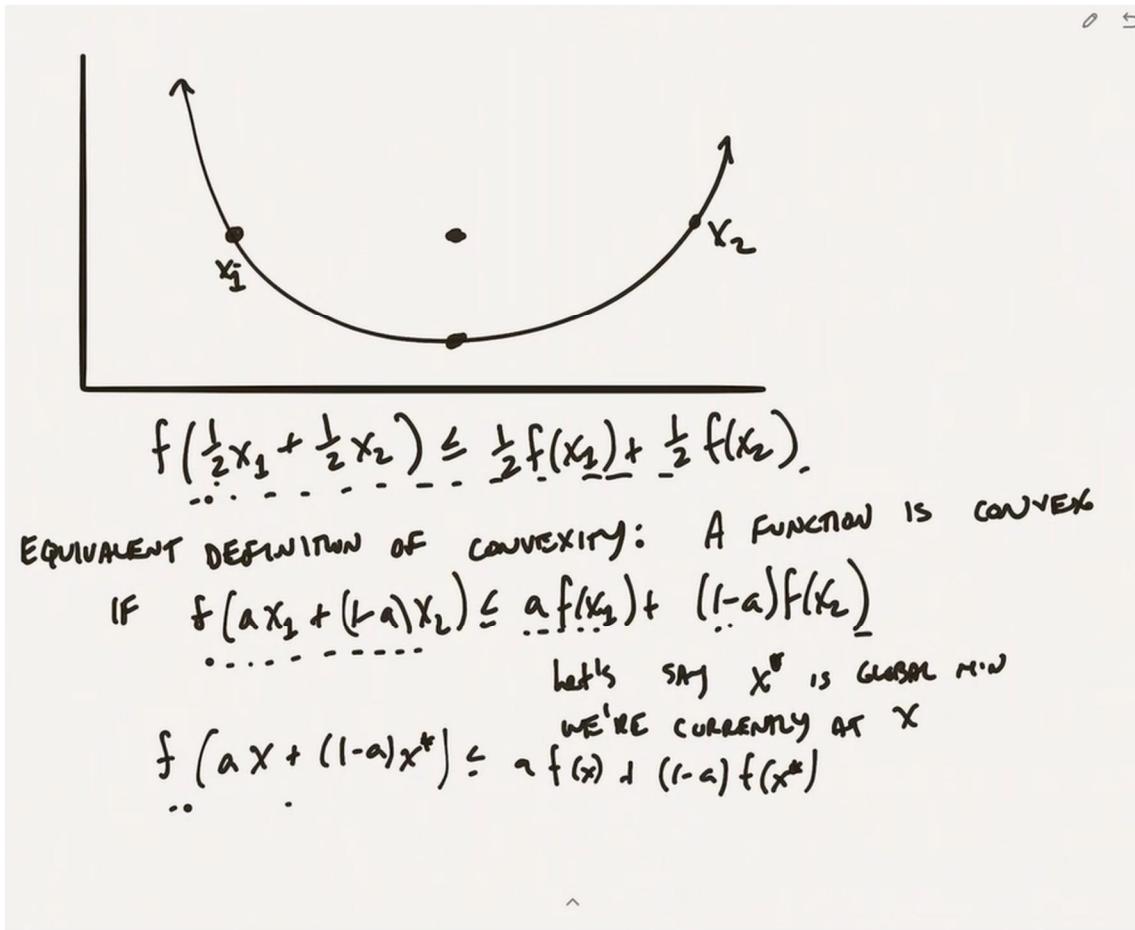
If  $f'(x) = 0$ , stop output X.

Eventually we will make our way down to the minimum, the global minimum regardless where we started.



Now, let's see a more complicated function. We want to use the same update rule and we are trying to minimize  $f(x)$ .

Depending on where we start we could find a local minimum but not the minimizer. Qualitatively what's the difference between these two functions where one works and the other one? Convexity is the difference: a function is Convex if the CHORD connecting any two points of the graph of the function lies above the function. If we take 2 points and connect them the chord lies above the graph in the connecting area. If some points lie above the chord connecting our points this is not a convex function.



So, generally we can say, in terms of the convexity of functions, let's say we have 2 points:  $x_1$ ,  $x_2$  and we know  $f(\frac{1}{2} \cdot X_1 + \frac{1}{2} \cdot X_2) < \frac{1}{2} \cdot f(X_1) + \frac{1}{2} \cdot f(X_2)$  is like we describe convexity.

Alternatively: a function is convex if:  $f(a \cdot X_1 + (1-a) \cdot X_2) < a \cdot f(X_1) + (1-a) \cdot f(X_2)$ . With this condition we know the algorithm will find the global minimum.

Demonstration is:

Let's say  $X^*$  is global minimum and we're currently at  $X$ :

$$f(a \cdot X + (1-a) \cdot X^*) < a \cdot f(X) + (1-a) \cdot f(X^*)$$

So, this sort of iterative process where we're moving towards  $X^*$  based on derivatives is going to find the global minimum as long as the function is convex.

ANOTHER EXAMPLE:

$$f(x) = w^T x + b$$

(LINEAR FUNCTIONS IN d dimensions)

CURRENTLY AT POINT X. We want to know what direction should we move in to minimize f?

By "direction" we mean unit vector.

(u will be unit vector)

$$\underline{f(x+u)} = \underline{w^T x} + \underline{w^T u} + \underline{b} \quad \text{correct choice of } u = \frac{-w}{\|w\|}$$

IF WE MOVE IN DIRECTION  $\frac{-w}{\|w\|}$  THEN f DECREASES

BY  $\|w\|_2$

Let's take another example, we have a linear function:  $f(x) = w \cdot x + b$ , where "w" is a vector and this linear function is in d-dimension.

Currently we're at point "X" and we want to know what direction should we move in to minimize "f(x)" and by direction we mean the unit vector ("u").

For this particular case because we're a linear function we know exactly which direction to move in. In other scenarios we will take the derivative with respect to different dimensions of "X".

$f(x+u) = w^T x + w^T u + b$ ; correct choice of "u" in linear functions:  $u = -w/\|w\|$ . In linear functions it makes sense to go the opposite direction of the slope.

Gradient descent for complex function

Thus far: our idea has been to look at <sup>tangent</sup> lines and this idea works for say linear functions and simple convex functions.

Even if we want to minimize more complicated functions, assume they are "locally" linear.

$$f \text{ at fixed } x$$

$$f(x+\epsilon) = f(x) + \epsilon \cdot f'(x) + \frac{\epsilon^2}{2!} f''(x) + \frac{\epsilon^3}{3!} f'''(x) + \dots$$

*expression w terms of  $\epsilon$ .*

*LINEAR FUNCTION OF  $\epsilon$*

*WHEN  $\epsilon$  IS SMALL, THESE TERMS ARE NEGIGIBLE.*

This far: our idea has been to look at lines and this idea Works for linear functions and simple convex functions.

Even if we want to minimize more complicated functions, assume they are "locally linear". Locally we are going to be they are locally linear so it means for a functions to be differentiable, this is in a very Small region dx.

Imagine we have a function "f(x)" at point "x" and we will use an alternative theoreme: Taylor.

$$f(x + \epsilon) = f(x) + \epsilon \cdot f'(x) + \epsilon^2/2! \cdot f''(x) + \epsilon^3/3! \cdot f'''(x) + \dots$$

This is a linear function in terms of épsilon ( $\epsilon$ ) and when " $\epsilon$ " is very Small these terms are going to be negligible.

So, according to Taylor when " $\epsilon$ " is very Small we can totally asume that our super complicated function "f" is actually a linear function:

$$f(x + \epsilon) = f(x) + \epsilon \cdot f'(x)$$

Taylor's Thm ALSO Holds IN d DIMENSIONS

INSTEAD OF TAKING DERIVATIVES (UNIVARIATE CASE)

FOR HIGHER DIMENSIONS WE MUST LOOK AT GRADIENTS.

THE GRADIENT OF  $f$  AT POINT  $x$

$$\nabla f(x) = \left( \frac{\partial f}{\partial x_1}(x), \dots, \frac{\partial f}{\partial x_d}(x) \right) \quad \begin{matrix} \swarrow \\ \text{d-dimensional} \end{matrix} \quad \begin{matrix} \searrow \\ \text{vector.} \end{matrix}$$

$$f = w^T x + b \quad \frac{\partial f}{\partial x_i} = w_i \quad \nabla f = w$$

Fortunately, Taylor's theorem also holds in d-dimensions and it still makes sense assuming the function looks linear in local neighborhood.

Instead of taking derivatives, as we were doing, for higher dimensions we must look at gradients.

The gradient of "f" at point "x":

$$\text{Grad}(f(x)) = \left( \frac{\partial f}{\partial x_1} \cdot x_1, \frac{\partial f}{\partial x_2} \cdot x_2, \dots, \frac{\partial f}{\partial x_d} \cdot x_d \right)$$

Simplest function to show as example:

$$f = w \cdot x + b$$

$$df/dx_i = w_i$$

$$\text{Grad}(f) = w.$$

$$f(x) = x^T A x - b^T x \quad (\text{n-dimensional})$$

o s c

$$f(x) = \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j - \sum_{i=1}^n b_i x_i$$

(\*)  $\dots$

$$\frac{\partial f}{\partial x_k} = \sum_{j=1}^n \underbrace{a_{kj} x_j}_{\substack{\text{ith row of } A \\ \text{inner prod with } x}} + \sum_{i=1}^n \underbrace{a_{ik} x_i}_{\substack{\text{kth col of } A \\ \text{inner prod with } x}} - b_k$$

CASE 1:  $i=k$   $\frac{\partial f}{\partial x_k} = \sum_{j=1}^n a_{kj} x_j + \sum_{i=1}^n a_{ik} x_i - b_k$   
 (when  $i=k$  and  $j \neq k$ )  $a_{kk} x_k^2 \frac{\partial (a_{kk} x_k^2)}{\partial x_k} = 2a_{kk} x_k$

CASE 2:  $i \neq k$  we won't have  $a_{ik} x_k$  term because  $i \neq k$   
 ↙ GRADIENT AT POINT X  
 IF A IS SYMMETRIC  
 $2A_x - b$

ANSWER  $Ax + A^T x - b$

Now another example:

$$f(x) = x^T A x - b^T x, \text{ in n-dimension}$$

We would like to compute the gradient of  $f(x)$  with a more complicated expression.

Defining gradient descent problem

o s c

# DEFINE GRADIENT DESCENT

INITIALLY WE'LL CHOOSE  $w$  RANDOMLY  
 (WANT TO MINIMIZE  $f(w)$ )

IF  $\|\nabla f(w)\|_2 < \epsilon$  STOP OUTPUT  $w$

OTHERWISE  $\underline{w_{\text{new}} = w_{\text{old}} - \eta \nabla f(w)}$   
 ↴ STEP-SIZE PARAMETER

$$w_j^{\text{new}} = w_j^{\text{old}} - \eta \frac{\partial f}{\partial w_j}(w)$$

IS USUALLY  
 SET TO BE  
 RELATIVELY SMALL.

We're finally ready, so far:

- Built up some intuition about using derivative information to minimize a function.
- Done some examples where we've taken partial derivatives.

We're ready to define the gradient descent.

Initially, we will choose "w" when trying to minimize  $f(w)$ , where "w" can be an arbitrary point, a random "w" or the zero vector.

We will then compute the gradient of "f" at point "w".

**Note:** In particular, the Euclidean distance of a vector from the origin is a norm, called the Euclidean norm, or 2-norm, which may also be defined as the square root of the inner product of a vector with itself.

So if finally the 2-norm of  $\text{grad}(f(w)) < \epsilon$ , we will stop and output "w" as the incline is already close to 0 (global minimum).

Otherwise:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \text{grad}(f(w))$$

This " $\eta$ " is referred to the step-size parameter. This is usually set to be relatively small. The reason for it to be small we want our linear approximation to hold locally as we only have guarantee the linear approximation works locally.

Applying gradient descent to linear regression

APPLY GRADIENT DESCENT TO LINEAR REGRESSION

$$h(x) = w^T x + b \quad (\text{SEARCHING FOR THIS FUNCTION})$$

(WE HAVE A TRAINING SET OF SIZE  $m$ )

$$\text{M.S.E. } (w) = \frac{1}{m} \sum_{j=1}^m \underbrace{(w^T x^j + b - y^j)^2}_{g_j}$$

$$\frac{\partial g_j}{\partial w_i} = 2 \cdot (w^T x^j + b - y^j) x_i^j$$

$$\nabla g_j(w) = 2 (w^T x^j + b - y^j) x^j$$

Apply gradient descent to linear regression we've seen in the lesson before.

$h(x) = w^T x + b$  (searching for this function "h").

We're assuming we have a training set of size "m" and we are trying to find the linear function that best fits.

Recall our notion of loss was mean squared error:

$$\text{Mean Squared Error } (w) = \frac{1}{m} \cdot \sum_{j=1}^m (w^T \cdot x^j + b - y^j)^2$$

So this is the quantity we're trying to minimize, we're not doing gradient descent of the function "h" but gradient descent of this MSE of "w". This is our actual "f" function.

$$w^T \cdot x^i + b - y^i = g_j$$

o s c

### APPLY GRADIENT DESCENT TO LINEAR REGRESSION

$$h(x) = w^T x + b \quad (\text{SEARCHING FOR THIS FUNCTION})$$

(WE HAVE A TRAINING SET OF SIZE  $m$ )

$$\underline{\text{M.S.E.}(w)} = \frac{1}{m} \sum_{j=1}^m \underbrace{(w^T x^j + b - y^j)^2}_{g_j}$$

$\text{M.S.E.}(w)$  IS  
A CONVEX FUNCTION.

$$\frac{\partial g_j}{\partial w_i} = 2 \cdot (w^T x^j + b - y^j) x_i^j$$

$$\nabla g_j(w) = 2 (w^T x^j + b - y^j) x^j$$

$$\underline{\nabla \text{M.S.E.}(w)} = \boxed{\frac{2}{m} \cdot \sum_{j=1}^m \underbrace{(w^T x^j + b - y^j) \cdot x^j}_{\ddots \ddots}}$$

Let's compute each:

$\frac{\partial g_j}{\partial w_i}$ , as these partial derivatives can work pretty well taking their sums.

We have performed the gradient of the MSE. And it is not hard to prove that the  $\text{MSE}(w)$  is a convex function. As we have a linear function in here, squaring is only going to make it more convex, we know that parabolas are convex. Then taking the positive sums of the convex functions is going to result in a convex function.

So if we do gradient descent on the MSE (Mean Squared Error) and the size is Small (to ensure linearity) we're guaranteed to find something very close to the global minimum.

Every time we need to compute a gradient update step we just need to compute the quantity highlighted in a rectangle.

Imagine we are building a model and we want to solve some linear regression running gradient descent.

$\{X_i\}, \{Y_i\} \rightarrow$  Known to us as they are part of the training set.

" $w$ "  $\rightarrow$  Known to us (current estimate for our linear function).

So we can compute the gradient descent quite easily.

The running time for computing this gradient will be:  $O(m \cdot n)$ . There is no need to run matrix inverses which increased the running time (recap).

This task is also easily parallelizable and each " $g_i$ " ( $\frac{\partial g_i}{\partial w_i}$ ) can be sent to a different processor. Each point in our training set is giving us some sort of vote to which direction we should move

in: some points give positive contributions and some other negative contribution and the gradient is taking an average of all those votes.

### Stochastic gradient descent

- o s c
- ### STOCHASTIC GRADIENT DESCENT
- PREVIOUSLY IN LINEAR REGRESSION EXAMPLE  
WE SUMMED OVER ALL POINTS IN TRAINING SET.

- CHOOSE AN INDEX  $j$  AT RANDOM; COMPUTE GRADIENT w.r.t. THIS POINT ONLY

$$w_{\text{new}} = w_{\text{old}} - 2 \cdot \eta \cdot (w^T \cdot x^j + b - y^j) \cdot x^j$$

$$E[w_{\text{new}}] = w_{\text{old}} - 2 \eta \cdot \frac{1}{m} \sum_{j=1}^m (w^T \cdot x^j + b - y^j) \cdot x^j$$

- USE "BATCHES" TO INTERPOLATE BTWN GRADIENT DESCENT AND PURE S.G.D. (SGD)

There is a popular variant of the gradient descent: stochastic descent.

Previously in the linear regression we summed overall points in the training set.

The gradient is the sum of those partial " $g_j$ " points-

One trick is to choose an index " $j$ " at random and compute the gradient with respect to this point only.

This will result in the following update rule:

$$w_{\text{new}} = w_{\text{old}} + 2 \cdot \eta \cdot (w^T \cdot x^j + b - y^j) \cdot x^j$$

Previously we had seen:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \text{grad}(f(w)) = w_{\text{old}} - \eta \cdot \text{grad}(\text{MSE}(w))$$

Now we just pick one point and we wonder how this update rule makes sense: there is something about the randomness of this point that will help us. In expectation our update will use the entire gradient as each point will be chosen with equal probability.

$$E[w_{\text{new}}] = w_{\text{old}} - 2 \cdot \eta \cdot \frac{1}{m} \cdot \sum_{j=1}^m (w^T \cdot x^j + b - y^j) \cdot x^j$$

See, because we made the choice uniformly at random the expected value of the update actually is the entire gradient. In expectation, we are just doing gradient descent. There is the issue what does the variance good like.

We have gradient descent where we use the entire training set and pure stochastic gradient descent where we choose one example at random. We could interpolate between gradient descent and pure Stochastic Gradient Descent (SGD) using "batches" and this would help us reduce the variance of the random variable.

### Choosing " $\eta$ " (step-size)

- HOW TO CHOOSE  $\eta$  THE STEP-SIZE
- MORE ART THAN SCIENCE; USE CROSS VALIDATION TO PICK  $\eta$
- MANY TECHNIQUES FOR ADAPTIVELY CHOOSING  $\eta$
- MOMENTUM

We should discuss now how we choose " $\eta$ " (step-size). We can use our favourite tool to decide:

- Cross-validation to pick " $\eta$ ".
- Many techniques for adaptively choose " $\eta$ " very successful in practice-
- Momentum is one of these techniques.

The idea of "momentum" is based on the following idea: Imagine our gradient is zigzagging and it would be really great to get this momentum or inertia to force see the actual direction. It is like a velocity variable.

MOMENTUM HAS A "VELOCITY" VARIABLE ✓

$$V_0 = 0$$

$$V_i = -\eta g_i$$

$$V_i = \alpha \cdot V_{i-1} - \eta g_i$$

This TAKES A WEIGHTED MOVING AVERAGE OF  $-\eta g_i$ 'S  
EXponential " "

$$w_{\text{new}} = w_{\text{old}} + V_i$$

### ACCELERATED GRADIENT DESCENT

Momentum has a velocity variable "v". Initially our velocity is "0" and in the "i"th iteration the velocity is:

$$V_0 = 0$$

$$V_i = \alpha \cdot V_{i-1} - \eta \cdot g_i$$

$$\text{The first iteration } V_1 = -\eta \cdot g_1$$

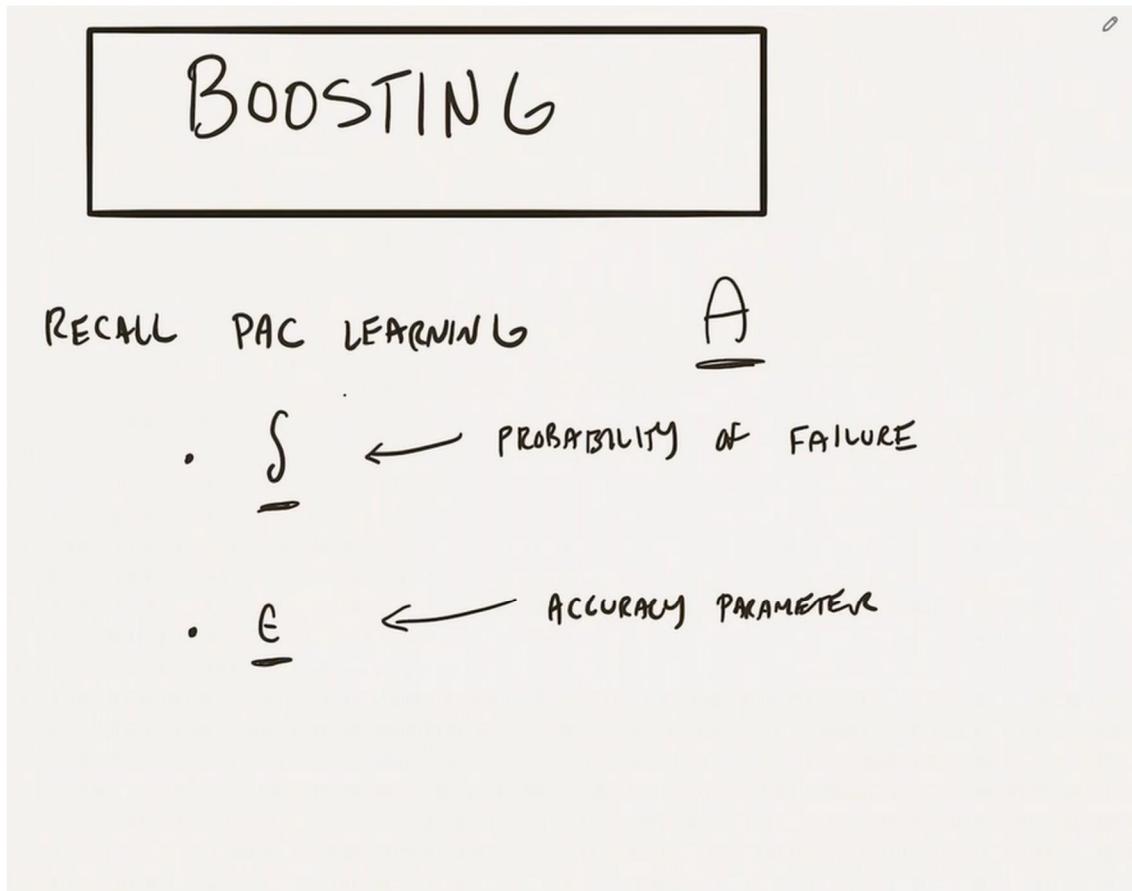
$$w_{\text{new}} = w_{\text{old}} + V_i$$

The first operation will be exactly like the gradient descent.

But then it is going to start subtracting off these weighted moving averages of our gradients and by doing this we are taking into account the average of gradients so far and we end up moving in the relevant direction.

There is something called the accelerated gradient descent is a more sophisticated type of momentum but it is advanced matter.

## Boosting

Introduction

This is a new technique called “Boosting”, this is an algorithm where we take a bunch of classifiers that are only sort of weak (i.e. 51% accurate on a training set), and we combine them in such a way that we end up with one classifier that is extremely accurate on our training set.

The technique changes the way people did machine learning and made the problem of coming up with one single classifier with high accuracy easier just running several type of classifiers in various software packages and come up with a classifier that has really high accuracy.

As long as the classifier is just doing a little bit better than random guessing, we can use this technique to boost the accuracy.

We will see a very simple boosting algorithm.

Let's go back to PAC learning:

Two key parameters in PAC learning.

- $\delta$ : probability of failure. Some algorithm “A” fails to output a good hypothesis.
- $\epsilon$ : accuracy parameter. The smaller the parameter the smaller the error of the classifier that “A” outputs.

REQUIREMENT:

FOR ANY CHOICE OF  $\epsilon, \delta$  A SHOULD  
OUTPUT WITH PROBABILITY  $> 1 - \delta$ , AN  $\epsilon$ -ACCURATE  
CLASSIFIER.

A IS ALLOWED TO RUN IN TIME  $\text{poly}\left(\frac{1}{\epsilon}, \frac{1}{\delta}\right)$   
TAKE # OF SAMPLES  $\text{poly}\left(\frac{1}{\epsilon}, \frac{1}{\delta}\right)$ .

QUESTION: WHAT IF WE HAVE AN ALGORITHM

A THAT WITH PROBABILITY 5%. OUTPUTS AN  
 $\underline{\epsilon}$ -ACCURATE CLASSIFIER. HOW CAN WE USE A  
TO OBTAIN A STANDARD PAC LEARNER?

In PAC learning the requirement:

For any choice of  $\epsilon, \delta$  then "A" should output with probability  $> 1 - \delta$ , an  $\epsilon$ -accurate classifier.

"A" is allowed to run in time polynomial( $1/\epsilon, 1/\delta$ ) and is allowed to take a number of samples that are polynomial in size polynomial( $1/\epsilon, 1/\delta$ ). This means if we are trying to run more and more accurate classifiers then we should allow "A" to run longer.

Question: What if we have an algorithm "A" that with probability 5% outputs an  $\epsilon$ -accurate classifier?

This algorithm sort of looks like a PAC learner and we give it an  $\epsilon$  as input and produces an  $\epsilon$ -accurate hypothesis or result. We can make  $\epsilon$  small but it does not really have a  $\delta$  parameter as input. So no matter what we set  $\delta$  to be, it is only going to output with probability 5% an  $\epsilon$ -accurate hypothesis.

WE WANT TO INCREASE THAT 5% PROB  
OF SUCCESS TO  $1-\delta$ .

THE SOLUTION IS TO RUN A A LARGE # OF TIMES  
SAY  $t$ .

$$\Pr [A \text{ fails to output an } \epsilon\text{-ACCURATE CLASSIFIER}] = (0.95)^t$$

RUN A  $t$  times

WE CAN MAKE  $(0.95)^t$  VERY SMALL BY CHOOSING  $t$  TO  
BE  $\approx O(\log \frac{1}{\delta})$  THEN WE CAN "TEST" EACH  
(CLASSIFIER GENERATED DURING THESE  $t$  TRIALS TO  
SEE IF ANY OF THEM ARE GOOD CLASSIFIERS.

So, we want to increase that 5% probability of success to  $1-\delta$ . The solution is to run "A" a large number of times " $t$ ".

The probability that now "A" fails to output an  $\epsilon$ -accurate classifier is at most  $0.95^t$ .

So we can make  $(0.95)^t$  very small by choosing " $t$ " to be of Order  $O(\log(1/\delta))$  and then we can test each classifier generated during these " $t$ " trials to see if any of them are good classifiers.

We have an algorithm that only succeeds with probability 5%. If we run it " $t$ " times, the probability that it fails is at most  $(0.95)^t$ . So we can take " $t$ " to be  $\log(1/\delta)$  and then this probability to fail is going to be smaller than  $\delta$ .

With reasonable probability one of these classifiers is actually going to be  $\epsilon$ -accurate and test every single classifier generated to see which one is the good one.

BOTTOM LINE: AMPLIFYING THE PROBABILITY OF SUCCESS

IS NOT TOO DIFFICULT.  $S \cdot 1 \rightarrow 1 - \delta$

TRICKIER QUESTION: WHAT IF  $\epsilon$  IS FIXED TO SAY

• 49.

IMAGINE A WITH PROBABILITY  $\geq 1 - \delta$  OUTPUTS A  
CLASSIFIER WITH  $\epsilon = 0,49$ .

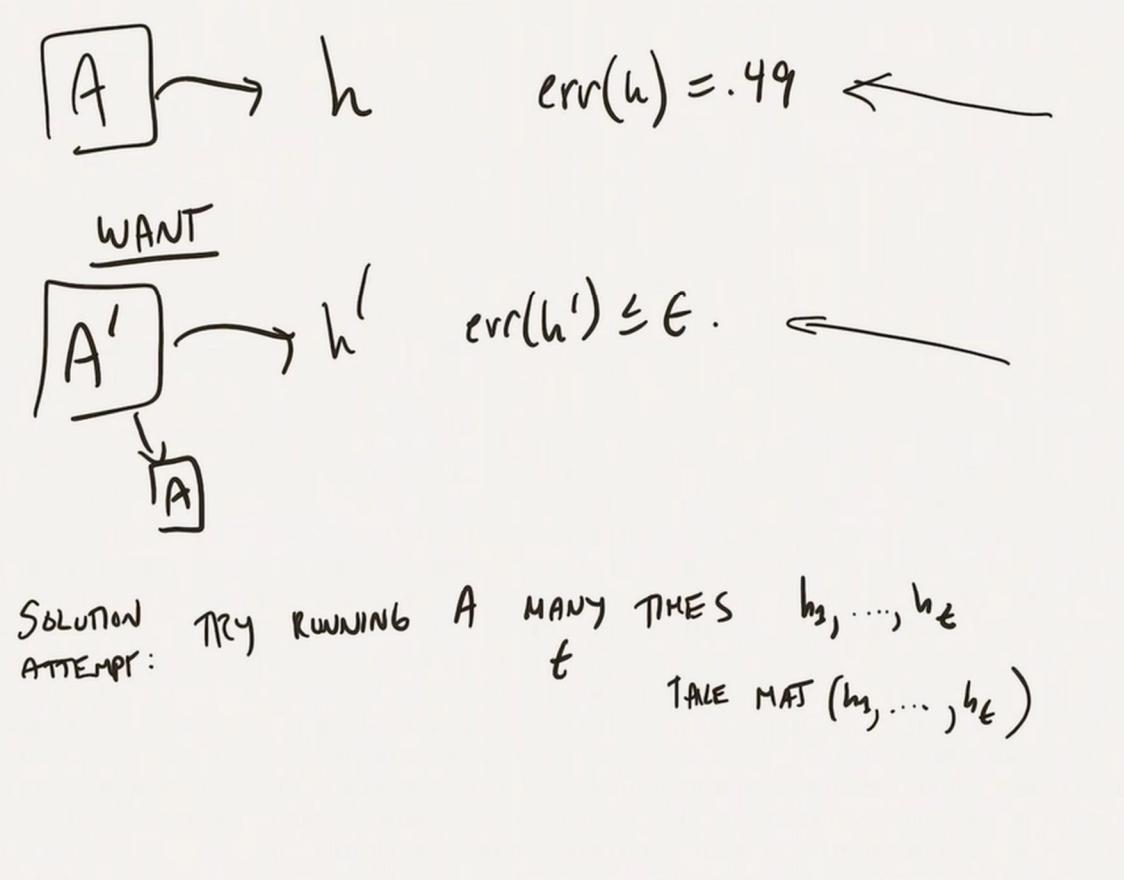
NATURAL QUESTION: HOW DO WE AMPLIFY / IMPROVE THE  
ACCURACY PARAMETER?

The bottom line is amplifying the probability of success by running the algorithm "A" "t" times taking new samples each time with "t" not too large one of the classifiers is going to be good.

This way our 5% algorithm will eventually be the  $1 - \delta$ .

Tricky question: What if  $\epsilon$  is fixed 0,49. Imagine "A" with probability  $1 - \delta$  outputs a classifier with  $\epsilon = 0,49$ . Si the classifier output is doing a little better than random guessing with 0,49 error accuracy.

How do we amplify now the accuracy parameter ( $\epsilon$ )?

Attempt 1 to improve accuracy

To summarize we can see the picture above. This "A" uses "A" as a sub-routine and our classifier "h" with error 0,49 will output a new classifier "h'" with error  $\epsilon$ .

Attempt 1: try running "A" many times "t" and the the majority of our classifiers " $h_i$ " ( $h_1, h_2, \dots, h_t$ ) that the algorithm outputs. This might not work as the majority of "A" could output the same classifier over and over and there is no gain.

Attempt 2 to improve accuracy

x

o s c

QUESTION WASPOSED BY VALIANT ON PAC LEARNING

SOLVED BY R. SCHAPIRE.

SOLUTION DUE TO FREUND &amp; SCHAPIRE

"BOOSTING ALGORITHMS"

ALGORITHM I'LL PRESENT: ADABOOST

HIGH-LEVEL IDEA

A outputs

 $\epsilon = .49$ FRACTION OF  
CORRECT POINTS

IS .51



TRAINING SET

RUN A ON UNIFORM  
DIST ON POINTS IN  
TRAINING SET.

Attempt 2: question posed by Valiant and solved by Ronb Schapire. This solution is called boosting algorithm. The “boosting” comes from the fact that we’re boosting the accuracy parameter ( $\epsilon$ ).

The algorithm that we’ll be presenting is called “AdaBoost”.

The high level idea is imagine we’ve taken a training set with all our points displayed, and we run “A” on the unifrom distribution on points in the training set. We give “A” all the points in the training set.

“A” outputs a classifier “h” with  $\epsilon = 0.49$  and the fraction of correct points is 51%. Inside the régión the classifier is getting a little more than half of the points correct.

We gave all the points equal weight, but now we are gonna put a little more weight to the points wrong and little less weight to the points we got right.

The nice property of this new probability distribution is that it is going to have a little more probability on points we got wrong last time.

Now, when we call “A” on this new distribution that’s been re-weighted, it’s going to forcé “A” to do a little bit better on the points we got wrong last time.

We’ve given a new distribution to our training set and putting some little weight on the points we got wrong. This way the algorithm it’s not going too bad on the points that we were still getting right.

CORE IDEA:

- RE-WEIGHT POINTS WE GET WRONG TO HAVE MORE WEIGHT
- POINTS WE GET RIGHT: HAVE LESS WEIGHT
- RUN A AGAIN TO OBTAIN CLASSIFIER W.R.T. NEW WEIGHTING

MAJ (CLASSIFIERS GENERATED DURING THIS PROCESS)

Core idea: re-weight points we get wrong to have more weight., put less weight to points we get right. Run "A" to obtain classifier "h" with respect to new weighting. Finally we take majority of classifiers generated during this process (the majority result).

We have the assumption that no matter what distribution we give our learner it will be 51% accurate.

Adaboost

ADABoost (Simplified Version)

- TRAINING SET OF SIZE  $m$
- INITIALLY  $D_0 = \text{UNIFORM DIST.}$  CORRESPONDS TO  $w_i = 1 \forall i$ .  
DIST IS OBTAINED BY DIVIDING BY  $W$ , SUM OF WEIGHTS.
- $E = \text{error rate} \quad A = \text{accuracy} = 1 - E \quad \beta = \frac{E}{A}$
- CONCRETELY  $E = \frac{1}{2} - \gamma \quad \beta = \frac{1 - \gamma}{\frac{1}{2} + \gamma}$
- HOW TO UPDATE WEIGHTS: AT ITERATION  $t$ , RUN  $A$  TO OBTAIN  $\underline{h_t}$   
FOR EACH  $x_i$  s.t.  $h_t(x_i)$  IS CORRECT  $w_i^{\text{NEW}} = \beta \cdot w_i^{\text{OLD}}$   
" IS INCORRECT  $w_i^{\text{NEW}} = w_i^{\text{OLD}}$
- REPEAT FOR  $T$  STEPS OUTPUT  $\text{MAJ}(\underline{h_1}, \dots, \underline{h_T})$ .

AdaBoost is the algorithm we are gonna study now (simplified versión).

Assume we have a training set of size "m".

Initially the first distribution  $D_0$  is going to be the uniform distribution:  $w_i = 1$  for all points "i".  
The distribution is obtained by "W", sum of weights.

$E$  = error rate.

$A$  = accuracy =  $1 - E$

$\beta = E/A$  (update factor)

Error at every iteration (assuming error is approximately the random guessing - 50%):  $E = \frac{1}{2} - \gamma$   
 $\rightarrow A = 1 - E = 1 - (1/2 - \gamma) = \frac{1}{2} + \gamma \rightarrow \beta = (1/2 - \gamma)/(1/2 + \gamma)$ .

How do we update the weights?

At iteration "t" run "A" to obtain  $h_t$ .

Then: for each  $x_i$  such that  $h_t(x_i)$  is correct:  $w_i^{\text{new}} = \beta \cdot w_i^{\text{old}}$ .

If  $h_t(x_i)$  is incorrect  $w_i^{\text{new}} = w_i^{\text{old}}$ . Same weight.

Repeat for  $T$  steps and output majority vote  $h_1, h_2, h_t$ .

This corresponds to a new distribution for "t" steps" and output the majority of "H" through 1 to "t".

CLAIM: AFTER  $T$  iterations error  $h_{final} = \text{MAJ}(h_1, \dots, h_T)$

$$\leq e^{-2T\gamma^2} \Rightarrow \text{CHOOSE } T \approx \frac{1}{\gamma^2} \log\left(\frac{1}{\epsilon}\right)$$

then error of  $h_{final} \leq \epsilon$ .

CLAIM: after "t" iterations error  $h_{final} = \text{MAJORITY}(h_1, h_2, \dots, h_t)$  is at most:

$$\text{Error}(h_{final}) = e^{-2 \cdot t \cdot \gamma^2}$$

If we choose "t" to be roughly:

$$t = \frac{1}{\gamma^2} \cdot \log\left(\frac{1}{\epsilon}\right)$$

Then error of  $h_{final}$ :

$$\text{Error}(h_{final}) = e^{-2 \cdot \left(\frac{1}{\gamma^2} \cdot \log\left(\frac{1}{\epsilon}\right)\right) \cdot \gamma^2} < \epsilon$$

Let's prove it.

✖ o s c

**TOTAL WEIGHT AFTER AN ITERATION**

$W$  is wt of all points before iteration  $t$

WT OF CORRECT POINTS AFTER ITERATION  $t = \frac{(\frac{1}{2} + \gamma) \cdot \beta \cdot W}{(\frac{1}{2} - \gamma) \cdot W}$

" INCORRECT POINTS AFTER ITERATION  $t = \frac{(\frac{1}{2} - \gamma)}{(\frac{1}{2} + \gamma) \cdot \beta \cdot W}$

RECALL  $(\beta = \frac{\frac{1}{2} - \gamma}{\frac{1}{2} + \gamma})$

NEW SUM OF ALL WEIGHTS?  $W \left( \frac{1}{2} \beta + \gamma \beta + \frac{1}{2} - \gamma \right)$

$W \left( \frac{1}{2} + \gamma \right) \left( \beta + \frac{1}{2} - \gamma \right)$

$W (1 - 2\gamma) = W \cdot (2 \cdot (\frac{1}{2} - \gamma))$

AFTER  $i$  ITERATIONS  $\underset{\wedge}{\text{SUM OF WEIGHTS}} = \underline{W} \cdot \left( 2 \left( \frac{1}{2} - \gamma \right) \right)^i$

Let's look at the total weight after an iteration of our procedure.

" $W$ " is weight of all points before iteration " $t$ " ( $w_{\text{old}}$ ).

Weight of correct points after iteration " $t$ "?

$$w_{\text{correct\_new}} = (1/2 + \gamma) \cdot \beta \cdot W \quad (51\% \text{ total points are correct times the update factor})$$

$$w_{\text{incorrect\_new}} = (1/2 - \gamma) \cdot W \quad (49\% \text{ total points are incorrect times the previous weight}).$$

Recalling  $\beta$ :

$$W_{\text{correct}} + W_{\text{incorrect}} = W_{\text{total}} = (1/2 + \gamma) \cdot \beta \cdot W + (1/2 - \gamma) \cdot W = W \cdot (1 - 2\gamma)$$

If that " $W$ " was the sum of the weights before its iteration, what is the sum of the weight after " $i$ " iterations now?

$$w_{\text{total}, i} = W \cdot \left( 2 \cdot \left( \frac{1}{2} - \gamma \right) \right)^i, \text{ where } W \text{ is the original weight}$$

$\Rightarrow$  AFTER  $T$  ITERATIONS SUM OF ALL WEIGHTS  $\leq \underbrace{\beta \cdot (\frac{1}{2} - \gamma)}^{\theta} \cdot W_0$

CONSIDER A POINT  $X_i$  THAT  $h_{\text{FINAL}}$  GETS WRONG

$$w_T(X_i) \geq \beta^{\frac{T}{2}}$$

$\Rightarrow$  IF  $h_{\text{FINAL}}$  HAS ERROR  $\epsilon$ , THEN WT OF POINTS  $h_{\text{FINAL}}$  MISCLASSIFIES  $\geq \underbrace{\epsilon \cdot m \cdot \beta^{\frac{T}{2}}}$

This implies that after "t" iterations the sum of all weights is at most (sum of all points):

$$w_{\text{total}, t} = W_0 \cdot (2 \cdot (\frac{1}{2} - \gamma))^t$$

Where  $W_0$  is the initial weight.

Consider a point  $X_i$  that  $h_{\text{final}}$  gets wrong. This means the majority vote failed to give us a good estimate. Which implies:

$$w_T(X_i) > \beta^{t/2}$$

If  $h_{\text{final}}$  has error  $\epsilon$ , then  $w_T$  of all points  $h_{\text{final}}$  missclassifies (sum of all wrong points) is at least:

$$w_t > \epsilon \cdot m \cdot \beta^{\frac{t}{2}}$$

$$\begin{aligned}
 \underline{\epsilon \cdot m \beta^{\frac{t}{2}}} &\leq \underline{\left(2 \cdot \left(\underbrace{\frac{1}{2} - \gamma}_{\epsilon}\right)\right)^{\frac{t}{2}} \cdot m} \\
 \epsilon &\leq \left(\frac{4 \cdot E^2}{\beta}\right)^{\frac{t}{2}} \\
 \epsilon &\leq (1 - 4\gamma^2)^{\frac{t}{2}} \quad (1+x \approx e^x) \\
 &\leq e^{-2\gamma^2 \cdot t}
 \end{aligned}$$

A very basic inequality can be built now:

$$\begin{aligned}
 \epsilon \cdot m \cdot \beta^{\frac{t}{2}} &< m \cdot \left(2 \cdot \left(\frac{1}{2} - \gamma\right)\right)^t \quad (\text{summ of all wrong points} < \text{sum of all points}) \rightarrow \\
 \epsilon &< e^{-2\gamma^2 \cdot t}
 \end{aligned}$$

This proves the claim.

We said this was a simplified version of AdaBoost because we assumed that each iteration the accuracy of our classifier was  $\frac{1}{2} + \gamma$ .

This algorithm can really take advantage in case we take a very good accurate classifier.

More precisely, in AdaBoost we will set:

$$\beta_t = E_t / A_t$$

We set the beta parameter the multiplicative factor to be equal to the error of the "t"th iteration. So, this beta parameter could change throughout the course of the boosting algorithm depending on what the accuracy and error are.

Then output, instead of the strict majority of votes of our classifiers, the **sign** of the following sum:

$$SIGN\left(\sum_t \alpha_t \cdot h_t - \frac{1}{2}\right); \quad \alpha_t = \frac{\log\left(\frac{1}{\beta_t}\right)}{\sum_t \log\left(\frac{1}{\beta_t}\right)}$$

IF  $\gamma$  (Accuracy) IS INDEPENDENT FROM  $m$ , SIZE OF TRAINING SET THEN WE CAN CHOOSE  $m$  TO BE SUFFICIENTLY LARGE.

ADABoost IS ACTUALLY A SPECIAL CASE OF AN ALGORITHM DUE TO FREUND AND SCHAPIRE CALLED "HEDGE."

This is a slightly technical expression but it shows with some minor modifications how we can take advantage of classifiers that have hight accuracy during our boosting procedure.

QUESTION: how do we guarantee that  $h_{\text{final}}$  generalizes when taking the majority of classifiers is going to have a Small error?

Cross-validation is an option but it is sort of hard to prove anything regarding cross-validation.

We need to make sure that our number of training points "m" in the training set is large enough.

Size of final classifier ( $h_1, h_2, \dots, h_t$ ) VS size of training set.

We have the freedom to choose a training size "m" large enough to make sure that after "t" iterations of boosting, the number of classifiers times the size of the classifier is going to be less than "m".

- If accuracy ( $\gamma$ ) is independent from "m" (size of the training set) then we can choose "m" to be sufficiently large.

What the choice of "m" should be to guarantee it's sufficiently large?

AdaBoost is actually a special case of an algorithm due fo Freund and Shapire in the same paper called: "HEDGE".

Hedge (generic algorithm for Adaboost)

X HEDGE: "BEST EXPERTS" SETUP.

O ↗

$c_1, \dots, c_m$  AT EACH ITERATION EXPERT  $t$   
 $c_i$  SUFFERS A LOSS  $l_i \in [0, 1]$

$l^t$  A VECTOR OF LOSSES SUFFERED BY ALL EXPERTS AT  $t^{th}$  ITERATION.

INTUITION: WE WANT TO HAVE A MIXED STRATEGY OF EXPERTS  
WEIGHTED AVG OF "

GOAL: SUM OF OUR LOSSES AFTER  $T$  ITERATIONS  
SHOULD BE "CLOSE" TO BEST EXPERT IN HINDSIGHT.

AT EACH ITERATION WE MAINTAIN A SET OF WEIGHTS

$w_1, \dots, w_m$  WEIGHTED AVERAGE =  $\frac{w_i}{\sum_i w_i} = p_i$

To understand to "HEDGE" we have to move back to the sort of "BEST EXPERTS" set up.

Let's assume that we have experts  $\{C\} = \{C_1, C_2, \dots, C_m\}$ .

Each expert ( $C_i$ ) is trying to determine what the Price of the stock market is going to be tomorrow.

At each iteration (day) the expert  $C_i$  suffers a loss ( $L_i$ ). At iteration "t": " $L_t$ ".

Each expert is trying to pick the Price of the stock market and depending on how it does we could say the loss it's "0" (perfect) or it could suffer a large loss that we will call "1" (maximum loss). We are sort of normalizing the losses.

" $L_t$ " is going to be a vector of losses suffered by all experts at the "t"th iteration.

The goal here is to have a mixed strategy of experts so instead of having to pick an expert we're allowed to have a mixed strategy of experts and then accordingly suffer a weighted average of their losses.

Goal: sum of our losses after "t" iterations should be close to best expert in hindsight. Our total loss should be close to the loss to the best expert in hindsight (after all iterations if one of our experts was doing super well, then we should keep doing pretty well too).

At each iteration we maintain a set of weights:  $w_1, w_2, \dots, w_m$  (being "m" the size of training set or number of points).

$$P_i = \text{Weighted average} = \frac{w_i}{\sum_i w_i}$$

o s c

$w_1^t, \dots, w_m^t \rightsquigarrow$  PROBABILITY DISTRIBUTION  $p^t$

$$p_i^t = \frac{w_i^t}{\sum_i w_i^t}$$

LOSS WE SUFFER AT  $t$  ITERATION IS  $\frac{p^t \cdot l^t}{\text{WT AVERAGE OF LOSS OF EXPERTS}}$

TOTAL LOSS WE SUFFER AFTER  $T$  ITERATIONS =  $\sum_{t=1}^T p^t \cdot l^t$

HEDGE:  $\underline{w_i^{\text{NEW}}} = \underline{w_i^{\text{OLD}}} \cdot \beta^{l_i^t}$

We have:  $w_1, w_2, \dots, w_m$  that corresponds to a probability distribution  $p^t$ . This is the weighted average of experts encoded by  $P_t$ .

$$P_i^t = \text{Weighted average} = \frac{w_i^t}{\sum_i w_i^t}$$

Loss we suffer at the "t"th iteration:

$$L_{t,\text{weighted}} = P^t \cdot L^t$$

Total loss we suffer after "t" iterations:

$$L_{t,\text{total}} = \sum_{j=1}^t P^j \cdot L^j$$

"HEDGE" says that:

$$w_i^{\text{new}} = w_i^{\text{old}} \cdot \beta^{L_i^t}, \text{ where } i \text{ is each expert}$$

If loss is "0" then  $\beta^0 = 1$ , and the weight stays the same.

If loss is "1" then  $w_{\text{new}} = w_{\text{old}} \cdot \beta$

This is similar to the boosting procedure. The "HEDGE" algorithm actually tells us how we should update weights and we have to be careful choosing beta.

0 5 2

$w_1^t, \dots, w_m^t \rightsquigarrow$  PROBABILITY DISTRIBUTION  $p^t$

$$p_i^t = \frac{w_i^t}{\sum_i w_i^t}$$

LOSS WE SUFFER AT  $t$  ITERATION IS  $\frac{p^t \cdot l^t}{\text{WT AVERAGE OF LOSS OF EXPERTS}}$

TOTAL LOSS WE SUFFER AFTER  $T$  ITERATIONS =  $\sum_{t=1}^T p^t \cdot l^t$

HEDGE:  $\underline{w_i^{\text{NEW}}} = \underline{w_i^{\text{OLD}}} \cdot \underline{\beta^{l_i^t}}$

CLAIM:

YOUR LOSS  $\leq \frac{\min_i \sum_{t=1}^T l_i^t + O(\sqrt{T} \log n)}{T}$

CLAIM: Our loss is at most:

$$L_{t,\text{total}} = \sum_{j=1}^t p^j \cdot L^j \leq \min_i \left( \sum_{j=1}^t L_i^j + O(\sqrt{t} \cdot \log(N)) \right),$$

The minimum of all experts which has less loss is our global loss after "t" iterations. In other words, our global loss will be at most the loss of the best expert.

As we add the square root of "t"  $\log(N)$  the loss is getting larger.

As "t" gets larger our average loss (dividing by "t") converges to the average loss of the very best expert.

"HEDGE" is an abstract framework to get the best expert in hindsight.

AdaBoost is derived from this "HEDGE".

This is the tip the iceberg of what boosting algorithms are able to do.

## Logistic regression

Introduction

# LOGISTIC REGRESSION

- CLASSIFICATION

- REGRESSION

$$\min_w \frac{1}{m} \sum_{i=1}^m (w^T x^i - y^i)^2$$

(\*)  $E[Y|X] = w^T X$

EVEN IF  
 (\*) DIFS NOT  
 HOLD

Logistic regression combines two techniques we already know: half space (perceptron) and linear regression.

Sometimes we would like to do classification mixed with some measure of confidence and we would like to understand how confident we are in our prediction.

Logistic regression is going to take that into account.

We talked about classification and regression in previous chapters.

In regression we ended up with an optimization problem:

$$\min_w \left( \frac{1}{m} \cdot \sum_{j=1}^m (w^T \cdot x^j + b - y^j)^2 \right)$$

We want to find the vector "w" that minimizes the error (MSE) o mean squared error.

We also discussed that if we're in a scenario where we have random variables "X", "Y" and the conditional mean " $E[Y|X]$ " is actually equal to " $w^T \cdot X$ " then our data is essentially a line and we know that in reality we could perform linear regression and output "w" that minimizes the MSE.

$$(1) E[Y|X] = w^T \cdot X$$

Another thing to notice is that even if (1) does not fit as a line and the random variables "X", "Y" do not satisfy this relationship maybe finding a line still does okay. We can always still write this down as a optimization problema and run gradient decent and output a line that fits reasonable well.

The image shows handwritten notes on classification loss functions. At the top, it says "CLASSIFICATION" and "WHAT IS LOSS FUNCTION FOR CLASSIFICATION?". Below this, it states that our guess for label  $y^i \in \{0, 1\}$  is going to be  $\text{SIGN}(w^T x^i)$  for some vector  $w$ . It then defines a "PENALIZED" loss function where if  $\text{SIGN}(w^T x^i) \neq y^i$ , there is a penalty, but if  $\text{SIGN}(w^T x^i) = y^i$ , there is no penalty. This leads to the "EXAMINE QUANTITY"  $y^i \cdot (w^T x^i)$ . A horizontal line under this quantity indicates that if the quantity is positive, there is no penalty; if it is negative, there is a negative penalty. To the left, it shows the "0-1 LOSS" function  $\varphi(z) = \begin{cases} 1 & \text{if } z \leq 0 \\ 0 & \text{if } z > 0 \end{cases}$  and the "SQ-LOSS" function  $(w^T x^i - y^i)^2$ . To the right, it shows the "CLASSIFICATION" loss function  $\underline{\varphi_{0-1}(y^i \cdot w^T x^i)}$ .

When it comes to classification, what is the loss function for classification? It is not the MSE but our guess for label  $y^i \in \{0, 1\}$  is going to be  $\text{SIGN}(w^T \cdot X)$  for some vector " $w$ ". We should penalize if sign of " $w^T \cdot X \neq Y^i$ ".

We should examine " $Z = Y^i \cdot (w^T \cdot X^i)$ " and check the SIGN:

- $Z > 0$ : No penalty, all good.
- $Z < 0$ : Penalty, SIGN is different.

Our function loss adds a penalti of value "1" when the SIGN expected is not equal to the actual SIGN:

$$\varphi(Z) = \{ 1 \text{ if } Z < 0; 0 \text{ if } Z > 0 \}$$

Let's again remember the formula for MSE from regression:

$$\text{SQUARE - LOSS} = (w^T \cdot x^j + b - y^j)^2$$

For classification:

$$0 - 1 \text{ LOSS} = \underline{\varphi_{0-1}(Y^i \cdot w^T \cdot X^i)}$$

So, we have two type of losses. For regression we had the least square loss and for classification we have something more appropriate because we just care if we are getting the sign right or not.

**OPTIMIZATION PROBLEM ASSOCIATED WITH CLASSIFICATION**

$$\min_w \frac{1}{m} \sum_{i=1}^m \ell_{0-1}(y_i \cdot w^T x^i)$$

WHEN DOES PERCEPTRON FIND A  $w$  WITH SMALL LOSS?

WHAT IS THE COMPUTATIONAL COMPLEXITY OF THIS OPTIMIZATION PROBLEM?

BAD NEWS: THIS PROBLEM IS NP-HARD RECALL THAT PERCEPTRON "AGNOSTICALLY LEARNING A HALFSPACE" REQUIRED  $\exists w$  s.t.  $\forall x$  AGNOSTIC LEARNING

QUESTION: WHAT IF THERE IS NO MARGIN?

THERE MIGHT NOT EXIST A  $w$  s.t.  $\text{SIGN}(w^T x^i) = y^i \forall i$

$y \cdot w^T x > \rho \Rightarrow$   
CONVERGENCE + MISTAKES  $< \frac{1}{\rho^2}$

Let's look back at our optimization problem associated with classification that we want to solve.

$$\min_w \left( \frac{1}{m} \cdot \sum_{j=1}^m (\phi_{0-1}(Y^j \cdot w^T \cdot X^j)) \right)$$

Under what condition does perceptron find a " $w$ " with Small loss?

Let's recall perceptron required the existence of a " $w$ " such that for all " $X$ ", the label " $Y$ " associated with " $X$ " was always at least row: " $Y \cdot w^T \cdot X > \rho$ " (margin condition). This margin condition implies convergence or a number of mistakes at most " $1/\rho^2$ "

After these many mistakes it is going to find a " $w$ " consistent with our training set as long as we have a margin.

What if there is no margin and no half space " $w$ " consistent with all the labels for our training set so our training set is not linearly separable?

Even though we could be in this tricky situation where there might not exist a " $w$ " that classifies all our training points, this optimization problema that we wrote still makee sense and we want to try and find the best fitting " $w$ ".

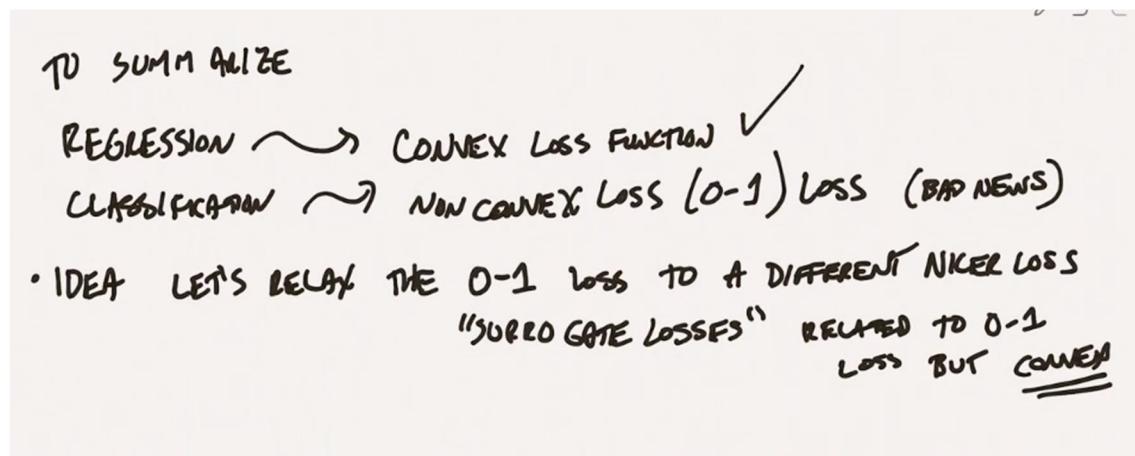
We could try and find the best fitting “w” with respect to the “0-1 LOSS”. This means that we want to find a half space with the SIGN of some linear function that makes as few mistakes as possible in our training set. That's what it means to minimize “w” with respect to the “0-1 LOSS”.

What is the computational complexity of this optimization problem compared with the MSE problem? Now the function is no longer convex and differentiable and we cannot use gradient descent.

Problem is NP-HARD (unlikely to admit a polynomial time solution). This problem is also referred to as agnostically learning half space.

We talked about PAC learning but there is another model called AGNOSTIC LEARNING which is a generalization of PAC learning.

If we knew that our training set was labeled perfectly by some unknown half space then we could talk about PAC learning the half space because we would know that every training point is perfectly labeled by some fixed half space. In agnostic learning, the training set may have no functional relationship with any half space but we can still ask for the best fitting half space.



To summarize: regression corresponds to a convex loss function classification. Non-convex loss namely the “0-1 LOSS”. Solving this optimization problems will lead to something called LOGISTIC REGRESSION.

The idea is to relax the 0-1 LOSS to something different or nicer that we can differentiate and convex.

Loss functions

$$\varphi_{LOGISTIC}(z) = \log(1 + e^{-z})$$

$$\varphi_{LOGISTIC}(y^i \cdot w^T x^i) = \log(1 + e^{-(y^i \cdot w^T x^i)})$$

If  $\underbrace{y^i \cdot w^T x^i}_{\text{MARGIN}} < 0 \Rightarrow \varphi_{LOGISTIC}(y^i \cdot w^T x^i)$  is LARGE

$\gg 0 \Rightarrow \varphi_{LOGISTIC}(y^i \cdot w^T x^i)$  is SMALL (moving to  $\rightarrow 0$ )

$$\varphi_{HUGE}(z) = \max\{1 - z, 0\}$$

$$\varphi_{HUGE}(y^i \cdot w^T x^i) \quad \begin{array}{l} \text{LARGE WHEN } y^i \cdot w^T x^i \text{ IS NEG} \\ \text{SMALL WHEN } y^i \cdot w^T x^i \text{ IS POS} \end{array}$$

$$\varphi_{EXP}(z) = e^{-z}$$

Let's introduce a few LOSS FUNCTIONS.

$$\varphi_{LOGISTIC}(Z) = \log(1 + e^{-Z})$$

$$\varphi_{LOGISTIC}(Y^i \cdot w^T \cdot X^i) = \log(1 + e^{-Y^i \cdot w^T \cdot X^i})$$

If  $Z < 0 \rightarrow$  Wrong answer then  $\varphi_{LOGISTIC}(Z)$  is large.

If  $Z > 0 \rightarrow$  Right answer then  $\varphi_{LOGISTIC}(Z)$  is Small and moving to "0".

This is what we want: large values when wrong and Small values when right.

$$\varphi_{HUGE}(Z) = \max\{1 - Z, 0\}$$

$$\varphi_{HUGE}(Y^i \cdot w^T \cdot X^i) = \max\{1 - Y^i \cdot w^T \cdot X^i, 0\}$$

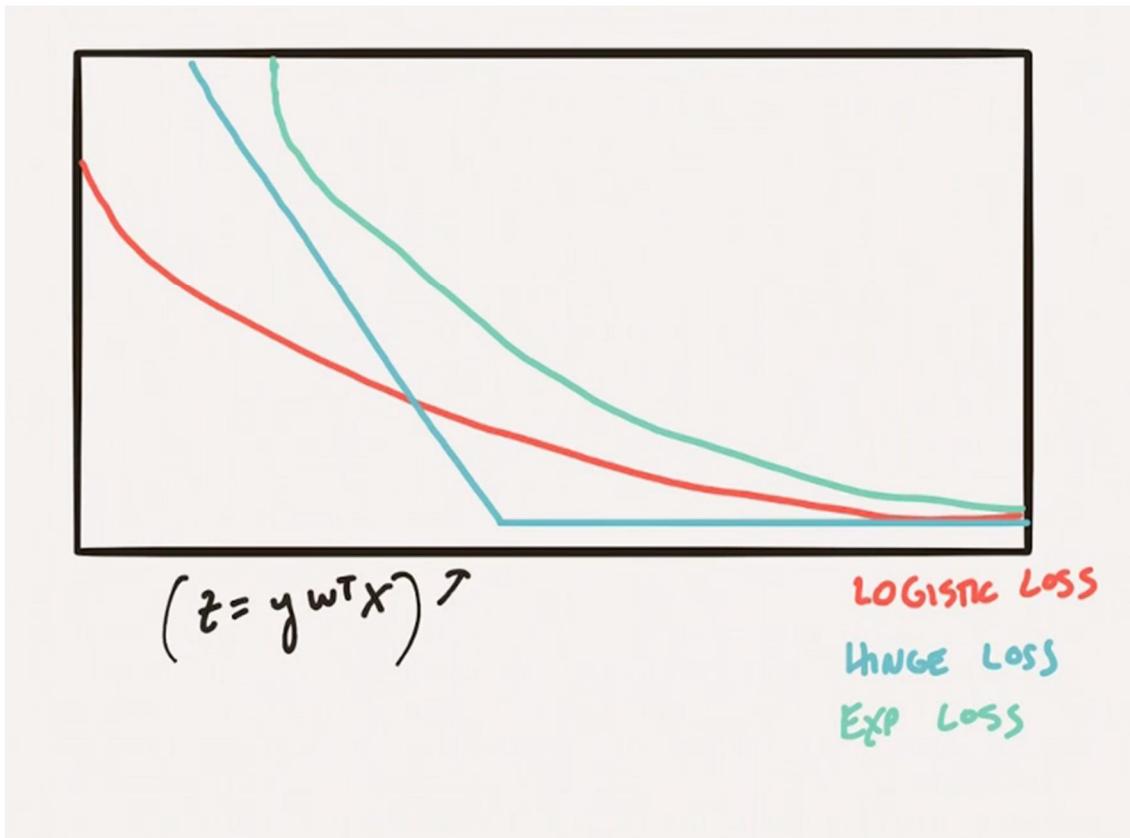
If  $Z < 0 \rightarrow$  Wrong answer then  $\varphi_{LOGISTIC}(Z)$  is large.

If  $Z > 0 \rightarrow$  Right answer then  $\varphi_{LOGISTIC}(Z)$  is Small and moving to "0".

Even when the SIGN is correct there is going to be some penalty if the answer is "half right" ( $Z = 0,5$ ).

$$\varphi_{EXP}(Z) = e^{-Z}$$

Let's graph some of these functions.



We're gonna focus in the logistic loss.

Optimization problema associated with logistic loss

OPTIMIZATION PROBLEM ASSOCIATED WITH LOGISTIC LOSS? o s c

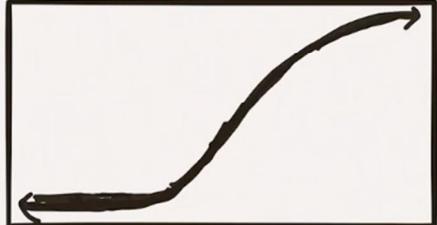
$$L(w) = \frac{1}{m} \sum_{i=1}^m \log(1 + \exp(-y^i \cdot w^T x^i))$$

$\min_w L(w)$

ENTER THE SIGMOID FUNCTION:

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

As  $z$  gets large  $g(z) \rightarrow 1$   
 $z$  gets small  $g(z) \rightarrow 0$



What is the optimization problema associated with logistic loss? We want to use the 0-1 loss but it is NP-hard so we want to try the logistic loss which is continuous, convex and differentiable.

$$\text{Loss}(w) = \frac{1}{m} \cdot \sum_{i=1}^m \log(1 + e^{(-y^i \cdot w^T \cdot x^i)})$$

So our problema is to figure out the vector "w" that minimizes the total loss ("Loss(w)"):

$$\min_w \text{Loss}(w)$$

There is one function that we need to introduce: Sigmoid Function.

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

It's a symmetric function that

- as "z" gets large  $\rightarrow g(z) = 1$
- as "z" gets Small  $\rightarrow g(z) = 0$ .

✖ ∅ ⊕ ⊲

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

$$\mathbb{E}[Y|X] = g(Y \cdot w^T X)$$

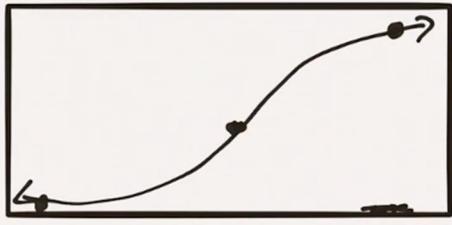
$Y \in \{-1, +1\}$

FACT:  $g(z) + g(-z) = 1$

$\frac{e^z}{e^z} \left( \frac{1}{1+e^{-z}} \right) + \frac{1}{1+e^z} \Rightarrow \Pr[Y=1|X] = g(w^T x)$

$\frac{e^z}{1+e^z} + \frac{1}{1+e^z} = 1.$

Given  $X$  if  $w^T x$  is large  
If  $w^T x$  is neg small



Taking into account:  $g(z) + g(-z) = 1$  (based on its symmetry)

Also take into account that the expected value of "y" ("y"  $\in [-1, 1]$ ) is obtained based on "x":

$$\mathbb{E}[Y|X] = g(y \cdot w^T x) \text{ for some } w.$$

This implies, that the  $\Pr[Y=1|X] = g(w^T x)$ .

Given "x" if " $w^T x$ " is large  $\rightarrow$  Sigmoid outputs "1".

Given "x" if " $w^T x$ " is small  $\rightarrow$  Sigmoid outputs "0".

In logistic regression we will assume that the label associated to the point "x" is proportional to this " $g(w^T x)$ ". Where "w" is unknown.

If  $g(w^T x)=1$  it implies " $w^T x$ " was a large value in a half space scenario and this implies our label was positive.

$\Pr[y = y^i \mid x^i; w] = g(y^i \cdot w^T x^i)$

"MODEL FOR LOGISTIC REGRESSION"

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

GIVEN A TRAINING SET  $S = \{(x^1, y^1), \dots, (x^m, y^m)\}$

WHAT IS THE MOST LIKELY  $w$  GIVEN THE TRAINING SET?

$L(w) = \prod_{i=1}^m p(y = y^i \mid x^i, w) = \prod_{i=1}^m g(y^i \cdot w^T x^i)$

$\text{LOG-LIKELIHOOD}(w) = \sum_{i=1}^m \log g(y^i \cdot w^T x^i)$

$= -\sum_{i=1}^m \log (1 + \exp(-y^i \cdot w^T x^i)) = -m \cdot L(w)$

LOGISTIC  
LOSS

So, we have to change half-space perspective from  $[-,+]$  to  $[-1,1]$ .

Remember that if:

- $y^i \cdot w^T \cdot x^i < 0 \rightarrow$  Then wrong result because signs are different (negative-positive).
- $y^i \cdot w^T \cdot x^i > 0 \rightarrow$  Then good result because signs are equal.

This means:

- $g(y^i \cdot w^T \cdot x^i) \approx 1 \rightarrow y^i \cdot w^T \cdot x^i > 0$ , then probability this is a good result is high ( $\approx 1$ )
- $g(y^i \cdot w^T \cdot x^i) \approx 0 \rightarrow y^i \cdot w^T \cdot x^i < 0$ , then probability this is a good result is low ( $\approx 0$ )

With this in mind:

$\Pr[y = y^i \mid x^i; w] = g(y^i \cdot w^T \cdot x^i)$ , this is the model for logistic regression.

When learning half spaces we assumed there was a classifier that perfectly separated the positive points from the negative points.

Now, we are going to assume that the probability that  $y = y^i$  (0 or 1) conditioned on "x" and "w" is equal to " $g(y^i \cdot w^T \cdot x^i)$ ".

Then we could ask, given the training set "S": what is the most likely "w" given the training set?

$$S = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$$

Which is the most likely "w" to occur?

$$\text{Likelihood}(w) = \prod_{i=1}^m P(y = y^i \mid x^i, w) = \prod_{i=1}^m g(y^i \cdot w^T \cdot x^i)$$

We want to **maximize this likelihood** and we will take the logarithm to transform the products in sums again and will use the sigmoid function:

$$\begin{aligned} \text{Log - Likelihood } (w) \\ = \sum_{i=1}^m \log(g(y^i \cdot w^T \cdot x^i)) = - \sum_{i=1}^m \log(1 + e^{-y^i \cdot w^T \cdot x^i}) = -m \cdot \text{Loss}(w) \end{aligned}$$

The log likelihood will turn into something that is exactly the Logistic Loss except for the "m" that we make appear as product.

Now our goal will be to minimize Logistic Loss  $L(w)$ .

How should we minimize Logistic Loss ( $w$ )?

Idea: Run gradient descent on Logistic Loss.

 This is the algorithm for performing logistic regression!

Let's say we find  $w^*$ ; for future examples we label them "+1" with prob

$$g(w^{*T} \cdot x)$$

So, in order to find the "w" that maximizes the likelihood, we have to minimize this logistic Loss quantity.

Now we see where the logistic loss function comes into play: we had this model that was based on a classification problem and we tried to find the maximum likelihood "w". We took logarithms and ended up with a logistic loss function.

Our goal now is to minimize the logistic loss ("Loss(w)").

Minimizing the Logistic loss is again easy as the function is convex and differentiable.

Idea: Run gradient descent on logistic loss to find some particular "w".

Let's see we find "w", for future examples from now on we label them "+1" (correct answer with probability "g( $w^T \cdot x$ )").

✖ ↗ ↘ ↙

LET'S COMPUTE THE GRADIENT OF  $L(w)$

$\varphi_{LOGISTIC}(z) = \log(1 + e^{-z})$

1.  $\varphi'_{LOGISTIC}(z) = \frac{-e^{-z}}{1 + e^{-z}} = -\frac{1}{1 + e^z} = -\underline{\underline{g(-z)}}$

2. COMPUTE

$$\frac{\partial \varphi_{LOGISTIC}(y \cdot w^T x)}{\partial w_k} = -\underline{\underline{g(-y \cdot w^T x)} \cdot \underbrace{y \cdot x_k}_{\text{CHAIN RULE}}}.$$

WITH THIS FORMULA WE CAN DIRECTLY APPLY GRADIENT DESCENT

Let's compute de Gradient descent of "Loss(w)". Remember some concepts:

$$\varphi_{LOGISTIC}(Z) = \log(1 + e^{-Z})$$

What is the derivative applying the chain rule we get to the sigmoid function:

$$\frac{d\varphi_{LOGISTIC}(Z)}{dz} = \frac{-e^{-Z}}{1 + e^{-Z}} = -\frac{1}{1 + e^Z} = -g(-Z)$$

Let's compute:

$$\frac{d\varphi_{LOGISTIC}(y \cdot w^T \cdot x)}{dw_k} = -g(-y \cdot w^T \cdot x) \cdot y \cdot x_k$$

Let's remember SIGMOID:

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

And apply gradient descent to the derivative. This Loss(w) function tells us precisely how to find "w" with maximum likelihood.

We could think of logistic regression as a model where the probability or likelihood or log likelihood are true and we can rely on these functions to find the best "w".

Additional considerations

WHAT HAPPENS IF WE HAVE MULTIPLE LABELS FOR  $y$ ?  $y \in \{0, 1\}$

WHAT IF  $y \in \{1, \dots, k\}$

MULTINOMIAL LOGISTIC REGRESSION:  $w^1, \dots, w^{k-1}$

$$\Pr[y=1|x] \propto e^{w^1 \cdot x}$$

$$\Pr[y=j|x] \propto e^{w^j \cdot x}$$

$$\Pr[y=k] = 1 - \sum_{i=1}^{k-1} \Pr[y=i]$$

What happens if we have multiple labels for "y":

- $y \in [0, 1]$ , normal scenario.
- $y \in [1, 2, \dots, k]$ , where "y" can take several values or labels rather than 2 (0 or 1). This is called multinomial logistic regression.

In this case of **multinomial logistic regression** we have several "w" vectors instead of 1:  $\{w^1, w^2, \dots, w^{k-1}\}$ .

The probability that "y" takes value "1" (as an example) is:

$$\Pr[y=1|x] = \frac{e^{w^1 \cdot x}}{\sum_{i=1}^{k-1} e^{w^i \cdot x}}$$

And then in general:

$$\Pr[y=j|x] = \frac{e^{w^j \cdot x}}{\sum_{i=1}^{k-1} e^{w^i \cdot x}}$$

When we do multinomial logistic regression we have "k-1" hidden weight vectors and the probability that you take any particular value is going to be proportional to " $e^w$ ".

WHAT IS THE ASSOCIATED LOSS?

CROSS-ENTROPY LOSS

GENERALIZATION OF LOGISTIC LOSS

IMAGINE  $y$  IS A VECTOR OF LENGTH  $K$   
WITH A 1 IN THE  $j^{\text{th}}$  POSITION IF  
(CORRECT LABEL IS  $j$ ) (ONE-HOT ENCODING  
OF LABELS).

LET'S SAY OUR GUESS FOR THE PROB  $y$  HAS LABEL  $i$   
IS  $\underline{P_i}$

$$-\sum_{i=1}^k y_i \log(P_i)$$

What loss should we use in a problem like this? This is called the cross-entropy-loss. And it consists in a generalization of the logistic loss.

Imagine “y” is a vector of length “k” with a “1” in the “j”th position if correct label is “j”.

Let's say our guess for the probability that “y” has label “i” is “ $P_i$ ”.

This result is the cross-entropy-loss:

$$-\sum_{i=1}^k y_i \cdot \log(P_i)$$

For example:

$Y^i = \{y_1, y_2, y_3, y_4, \dots, y_k\} = \{0, 0, 1, 0, \dots, 0\} \rightarrow P_3$  is gonna take a large value.

So we are fairly confident (large probability) that our label or value is in fact “3”.

The other way round, if some labels obtained with “wi” are far off from the true labels then we're going to suffer for this loss.

This is the introduction how the multiple label scenario should work when we get away from the binary scenario.

SUMMARY  $\rightsquigarrow$  TURNS REAL-VALUES INTO PROBABILITIES

$$w^T x \text{ via SIGMOID } w^T x \rightsquigarrow [0, 1]$$

(INTO A PROBABILITY)

$$\underbrace{(z_1, \dots, z_k)}_{K \text{ COORDINATES}} \rightsquigarrow \underbrace{\left( \frac{e^{z_1}}{Z}, \frac{e^{z_2}}{Z}, \dots, \frac{e^{z_k}}{Z} \right)}_{\text{PROBABILITIES}}$$

$$Z = \sum_{i=1}^k e^{z_i}$$

There is something called SOFTMAX which turns real values into probabilities.

We've transformed " $w^T \cdot x$ " via sigmoid to something that outputs between [0,1] (a probability).

So when we have a vector with "k" coordinates taking real values  $Z_1$  through  $Z_k$  and we want to transform this vector into probabilities, for what we think the correct label is, then we can transform this into " $\{e^{z_1}, e^{z_2}, \dots, e^{z_k}\}$ " and again normalize all of these.

With this, the "k" real values will be mapped to "k" probabilities between "0" to "1" and the sum will turn into 1.

SOFTMAX corresponds to taking some "K" long vector of real values and mapping them to probabilities or guesses for what the class label "w" should be.

One nice thing about logistic regressions was to learn the SIGMOID function to obtain the probability that our label was "1" corresponding to a value coming from a SIGMOID.

This is going to lead us into a discussion about more complicated models for data, namely neural networks where we may have different types of nonlinear activations such as SIGMOIDS but composed in very interesting ways.

And algorithms for learning neural networks have really transformed machine learning, artificial intelligence and data science.

## PCA – Dimensional reduction

Introduction

# PCA

- DIMENSIONALITY REDUCTION
- COMPARE TO "RANDOM PROJECTION" "JL-LEMMA"
 

RANDOMLY PICKED VECTORS  $r_1, \dots, r_k$   
 PROJECTED  $x \rightarrow \underbrace{\langle x, r_1 \rangle, \dots, \langle x, r_k \rangle}$

  - $r_i$ 's were ~~not~~ meaningful with respect to  $S$
  - PRESERVE EUCLIDEAN DISTANCE BTWN POINTS
  - IN PRACTICE  $k > 100$  FOR RANDOM PROJECTION TO WORK.
  - FOR PCA WE CAN CHOOSE  $k = 2$
  - PCA LOOKS AT  $S$  TO COME UP WITH A NEW REPRESENTATION

This is probably the most important technique for reducing the dimensionality of our data in machine learning. Is a simple and super effective way to come up with a different representation of our data but keeping it really meaningful and not loose too much information.

PCA stands for Principal Component Analysis and it helps us reducing the dimension of our training set but forms the basis for a bunch of other sophisticated techniques in machine learning.

PCA compared to "random projection" method or "JL-Lemma".

When talking about random projection:

- we randomly pick vectors  $\{r_1, r_2, \dots, r_k\}$
- we Project a random data point " $x$ " onto these random vectors (new base ?).
- we obtain:  $\langle x, r_1 \rangle, \langle x, r_2 \rangle, \dots, \langle x, r_k \rangle$ , this is the new representation of " $x$ " in a new base.
- " $r_i$ " are not meaningful with respect to " $S$ " but rather random.
- The only guarantee we have picking these random vectors is that we preserve the Euclidean distance.
- In practice  $k > 100$  for random projection to work.
- In contrast, for PCA we can choose  $k = 2$  and we can still get meaningful results.
- Moreover, PCA looks at the data to come up with a new representation that reduces our dimension and is data-dependent.

With PCA we are going to examine the training data and look at some structure to try and figure out how to best represent our training set "S" in a meaningful way.

HIGH LEVEL GOAL OF PCA IS TO FIND VECTORS

$$v_1, \dots, v_k \text{ s.t. } \forall x \in S \quad x \approx \sum_{j=1}^k a_j v_j$$

NOTE ABOUT PRE-PROCESSING OF S

- SUBTRACT THE MEAN OR CENTER OF MASS FROM EACH DATA POINT.
- NORMALIZE THE STANDARD DEVIATION OF EACH FEATURE BY

$$\forall i \text{ compute } \sqrt{\sum_{j=1}^m (x_i^j)^2} = \sigma_i$$

DIVIDE ALL THE  $i^{th}$  FEATURES BY  $\sigma_i$

At high level, the goal of PC1 is to find vectors  $\{v_1, v_2, \dots, v_k\}$  such that for all vectors  $\{x\}$  in our training set "S", we are going to be able to represent all vectors as projections of vectors  $\{v_1, v_2, \dots, v_k\}$

$$v_1, v_2, \dots, v_k \text{ and } \forall x \in S \rightarrow x \approx \sum_{j=1}^k a_j \cdot v_j$$

NOTE: before applying PCA to our training set we will do a little bit pre-processing of our training set.

- Subtract the mean or center of mass from each data point. Mean zero.
- Normalize by the standard deviation of each feature. We want the features to have relatively similar ranges. Let's say that one coordinate corresponds to length and we rescale this length. This way the features contribute to the calculations and cannot impact more or less.

For this, we need to compute:

$$\forall i \text{ compute } \sqrt{\sum_{j=1}^m (x_i^j)^2} = \sigma_i$$

Then, divide all the " $i$ "th features by " $\sigma_i$ ".

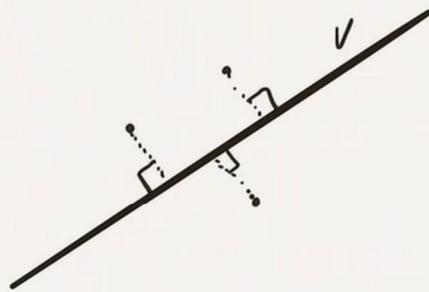
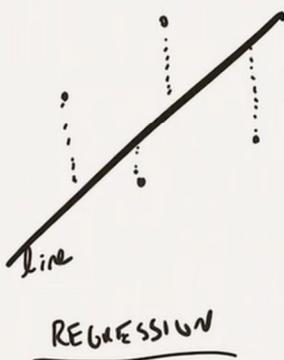
This sort of standardizes the magnitude of the features.

How to begin? FIND  $v_1$

LOOK FOR A VECTOR THAT MINIMIZES SQ-DISTANCE

$$\min_{\|v\|_2=1} \frac{1}{m} \sum_{j=1}^m (\text{distance BTWN } x^j \text{ AND } v)^2$$

PICTURE



How to proceed:

1. Find "v1". We need to look for a vector that minimizes squared distance to each point "j" in the training set "S".

$$\min(v, \|v\| = 1) \frac{1}{m} \cdot \sum_{j=1}^m (\text{Distance between } x^j \text{ and } v)^2$$

In regression we are trying to find a line that best fits the points and our loss (that is the vertical distance between the label and the line squared) will be the minimum overall for all points (left picture).

In PCA we're going to measure the distance from each point via its orthogonal projection and we will take squares (right picture). The distance will be computed via the orthogonal projection of our point "x" to that line "v".

$\min_{\|v\|_2=1} \frac{1}{m} \sum_{j=1}^m (\text{DISTANCE between } x^j \text{ and } v)^2$

$\langle x, v \rangle^2 + (\text{DIST from } x \text{ to } v)^2 = \|x\|^2$

- LOOKING FOR  $v$  SMALL
- LOOK FOR A  $v$  TO MAKE  $\langle x, v \rangle^2$  LARGE.

EQUIVALENTLY, WE WANT TO FIND A  $v$  THAT MAXIMIZES  $\langle x, v \rangle^2$

FIND  $v$   $\max_{\|v\|_2=1} \frac{1}{m} \sum_{j=1}^m \langle x^j, v \rangle^2$

DIRECTION OF  
MAXIMAL VARIANCE.

VAR  $E[x] - (E[x])^2$

Remember our goal:

$$\min(v, \|v\| = 1) \frac{1}{m} \cdot \sum_{j=1}^m (\text{Distance between } x^j \text{ and } v)^2$$

The orthogonal projection of "x" in "v" is the distance from "x" to "v". Thanks to the Pythagorean Theorem we can find that:

$$(\langle x, v \rangle)^2 + (\text{Distance}_v^x)^2 = (\|x\|)^2$$

$\|x\|$  is a value fixed by our training set "S".

We are looking for "v" to make distance Small or  $\langle x, v \rangle^2$  large.

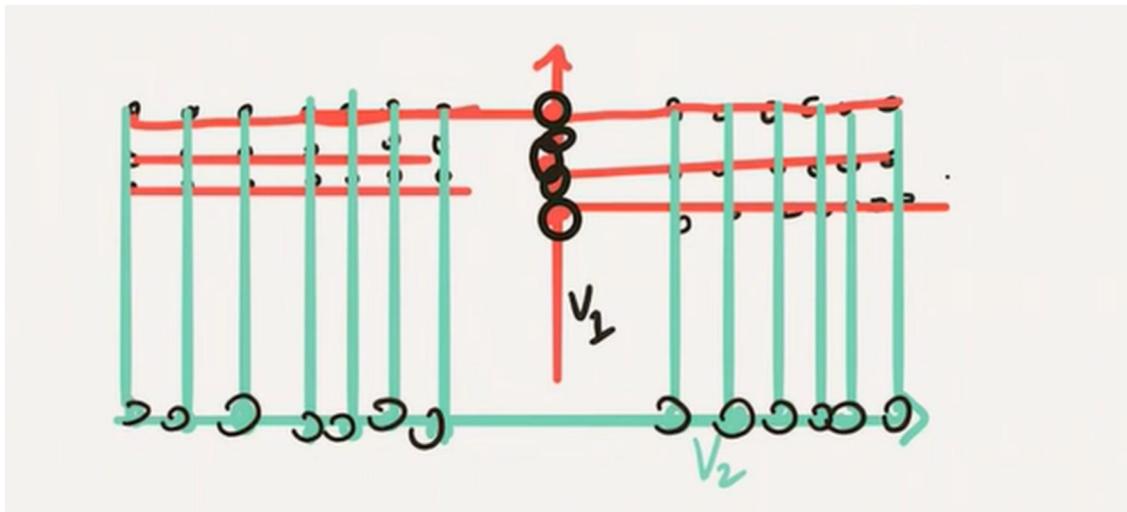
Equivalently, we want to find a "v" that maximizes the inner product  $\langle x, v \rangle^2$ .

$$\max(v, \|v\| = 1) \frac{1}{m} \cdot \sum_{j=1}^m \langle x^j, v \rangle^2$$

This is often referred as the direction of maximal variance.

Remember that the mean of our training point is "0".

Remember variance: variance =  $E[x^2] - (E[x])^2$ . Because our data is normalized and mean is "0" then variance: variance =  $E[x^2] - 0$ . Therefore, the projection of our training points in "v" maximum is equivalent to looking for the direction of maximum variance.



Let's try to understand this with a picture.

We had roughly 50 points but with  $v_1$  and  $v_2$  we reduce it to a few 20 points.  $v_1$  reduces everything to 5 points whereas  $v_2$  reduces to 10 points. Also if we Project along the direction " $v_2$ " we end up with much more variance in our training set "S" that if we Project with respect to direction " $v_1$ " where everything is clumped together and closer to each other.

This is why we prefer directions that have a lot of variance because we keep things spread out as our training set rather than collapsing everything into a single point where we don't really have a good projection.

In this case we would prefer vector " $v_2$ " to Project.

This was for 1 vector what about finding  $k$ -vectors  $k$ -components

MAX SUBSPACES  $S$  OF DIMENSION  $K$

$$\frac{1}{m} \sum_{j=1}^m (\text{LENGTH OF } x^j \text{ PROJECTED ONTO } S)^2$$

A REALLY NICE/PREFERRED BASIS WOULD BE AN ORTHONORMAL BASIS  $v_1, \dots, v_k$

$$(\text{DISTANCE FROM } X \text{ TO } S)^2 = \langle x_1, v_1 \rangle^2 + \dots + \langle x_k, v_k \rangle^2$$

$$\max_{\substack{v_1, \dots, v_k \\ \text{ORTHONORMAL}}} \frac{1}{m} \sum_{j=1}^m \sum_{i=1}^k \langle x^j, v_i \rangle^2$$

PCA OBJECTIVE

How about finding " $k$ " vectors or " $k$ " components?

The optimization problem is:

$$\text{Max}(subspaces S \text{ of dimension } k) \frac{1}{m} \cdot \sum_{j=1}^m (\text{Length of } x^j \text{ projected onto } S)^2$$

We are going to Project a subspace of dimension " $k$ " (a new vectorial basis).

A preferable basis would be an orthonormal basis  $\{v_1, v_2, \dots, v_k\}$ .

The advantage of using a orthonormal basis is that we can Project our point "x" into each vector in our basis and take the sum of the squares directly to get the length from "S" to "x".

$$(Distance_x^S)^2 = \langle x_1, v_1 \rangle^2 + \langle x_2, v_2 \rangle^2 + \dots + \langle x_k, v_k \rangle^2 \quad (S \text{ is orthonormal})$$

PCA ultimate goal is:

$$\text{Max}(v_1, v_2, \dots, v_k \text{ orthonormal}) \frac{1}{m} \cdot \sum_{j=1}^m \sum_{i=1}^k \langle x^j, v_i \rangle^2$$

In PCA we look for an orthonormal basis  $\{v_1, v_2, \dots, v_k\}$  that maximizes this variance along all of these directions as long as they are orthogonal.

Assume we have  $v_1, \dots, v_k$

$$x = \underbrace{\langle x, v_1 \rangle \cdot v_1}_{\text{projection}} + \underbrace{\langle x, v_2 \rangle \cdot v_2}_{\text{projection}} + \dots + \underbrace{\langle x, v_k \rangle \cdot v_k}_{\text{projection}}$$

$x$  can be written as a vector in  $\mathbb{R}^k$  corresponding to these projections.

Let's assume that we have  $\{v_1, v_2, \dots, v_k\}$ .

How do we rewrite "x":  $x = \langle x, v_1 \rangle \cdot v_1 + \langle x, v_2 \rangle \cdot v_2 + \dots + \langle x, v_k \rangle \cdot v_k$ .

So, our training set is just going to be the projection of the training set "S" into these bases's vectors.

Assume we have  $v_1, \dots, v_k$

$$\underline{x} = \underbrace{\langle x, v_1 \rangle \cdot v_1}_{\text{projection}} + \underbrace{\langle x, v_2 \rangle \cdot v_2}_{\text{projection}} + \dots + \underbrace{\langle x, v_k \rangle \cdot v_k}_{\text{projection}}$$

$\underline{x}$  can be written as a vector in  $\mathbb{R}^k$  corresponding to these projections.

### APPLICATION 1: UNDERSTANDING GENOMES

Took 1400 people from Europe

Each person was represented according 200,000 genetic markers in their genome.

Corresponds to matrix of dim  $1400 \times 200,000$

- Ran PCA on this data to find vectors  $v_1$  and  $v_2$

- Each person corresponds to 2 numbers.

They plotted these 2 numbers; color code each point according to country of origin.

### Applications

Let's talk about some interesting applications.

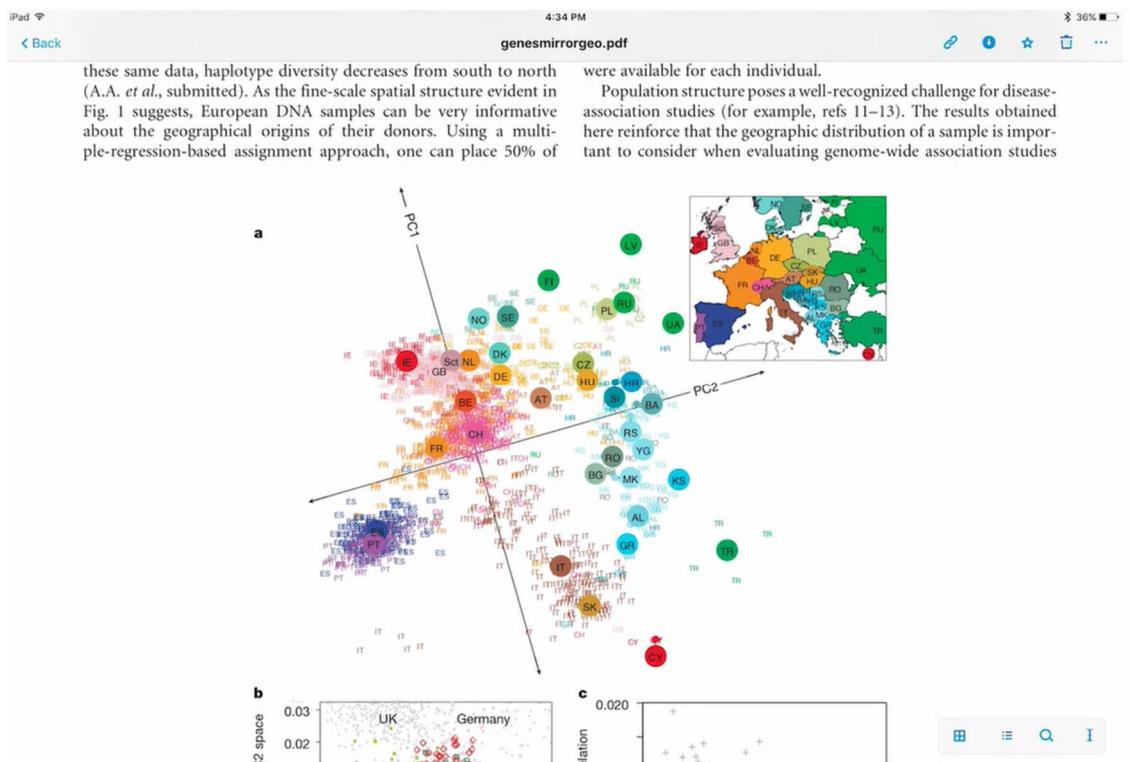
#### 1. Understanding genomes.

Somebody took 1400 people from Europe and each person was represented according 200.000 genetic markers (features) in their genome.

This corresponds to a matrix of dimension 1400x200.000.

They decided to run PCA on this data to find vectors  $v_1$  and  $v_2$  (two principal components).

Now each person corresponds to 2 numbers. They then plotted everyone and color coded each point according to the country of origin.



It matches the countries of Europe. Genetic similarities between people and neighbours.

## 2. Image data compression.

Strategy for compressing data is: each data point is an image (vector of pixels). Each image has 65.000 pixels (65.000 features).

The data set is a lot of images of faces with 65.000 pixels each.

They ran PCA on this data set with  $k = 150$ . So now each image is a linear combination of 150 vectors of length 65.000 (which is an image).

So we will build all images as a linear combination of 150 base images.



**Figure 3.** An original face image and its projection onto the face space defined by the eigenfaces of Figure 2.

Finding  $v_1, v_2, \dots, v_k$

QUESTION:  
How do we find THESE  $v_1, \dots, v_k$ 's?

MAX  
 $v_1, \dots, v_k$      $\frac{1}{m} \sum_{j=1}^m \sum_{i=1}^k \langle x^j, v_i \rangle^2$  IS MAXIMIZED  
 ORTHONORMAL.

Let  $X$  be an  $m$  by  $n$  matrix     $\underbrace{\frac{1}{n} X^T X}_{(n \text{ by } n \text{ MATRIX})}$

$v$  ← a column vector  
 $v^T$  ← a row vector  
 $v^T v$  ← INNER PRODUCT (SCALAR)  
 $v v^T$  ← OUTER PRODUCT (MATRIX)

SAMPLE COVARIANCE MATRIX  
 $(i,j)^{th}$  entry of  $X^T X$   
 CORRESPONDS TO "How similar is FEATURE  $i$  to FEATURE  $j$ ?"

The big question is: how do we find this basis:  $\{v_1, v_2, \dots, v_k\}$ .

Remember the function to maximize is:

$$\text{Max}(v_1, v_2, \dots, v_k \text{ orthonormal}) \frac{1}{m} \cdot \sum_{j=1}^m \sum_{i=1}^k \langle x^j, v_i \rangle^2$$

Let " $X$ " be an "mxn" matrix where "n" is the dimension and "m" the size of the training set.

" $v$ ": a column vector.

" $v^T$ ": a row vector

" $v^T \cdot v$ ": inner product (scalar).

" $v \cdot v^T$ ": outer product (matrix).

Once we have this matrix " $X$ " that encodes our training set " $S$ ", we are going to look at the " $X^T$ ".

" $X^T \cdot X$ ": "nxn" matrix.

" $1/n \cdot X^T \cdot X$ ": sample covariance matrix.  $(i,j)^{th}$  entry of " $X^T \cdot X$ " corresponds to how similar is feature "i" to feature "j".

NOTE THAT  $X^T X$  IS A SYMMETRIC MATRIX

$$X^T X \begin{pmatrix} ij \end{pmatrix} = \begin{array}{l} \text{INNER PRODUCT OF ROW } i \text{ OF } X^T \text{ WITH COLUMN } j \text{ OF } X \\ " \quad \text{COLUMN } i \text{ OF } X \text{ WITH COLUMN } j \text{ OF } X \\ = \text{COLUMN } j \text{ OF } X \text{ WITH " } i \text{ OF } X \end{array}$$

FACT: ALL EIGENVALUES OF SYMMETRIC MATRICES  $\geq 0$ .

FOR A MATRIX  $A$ ,  $v$  IS AN EIGENVECTOR IF  $A \cdot v = \lambda \cdot v$   $\lambda \in \mathbb{R}$   
 ↴  
 EIGENVALUE.

DEFINITION: AN ORTHOGONAL MATRIX IS ONE WHERE ALL COLUMNS ARE ORTHONORMAL.  $\Leftrightarrow$

$$A^T A = I \quad (A A^T = I)$$

NOTE: " $X^T \cdot X$ ": it is a symmetric matrix based on the inner product equivalents.

FACT: all eigenvalues of symmetric matrices are greater or equal to "0".

For a Matrix "A", "v" is an eigenvector if " $A \cdot v = \lambda \cdot v$ ", where " $\lambda$ " is the eigenvalue.

DEFINITION: an orthogonal matrix is one where all the columns are orthonormal  $\rightarrow "A^T \cdot A = I"$  and " $A \cdot A^T = I$ "

Spectral theorem

**SPECTRAL THM:** EVERY <sup>SYMMETRIC</sup> MATRIX  $A$  HAS AN EIGENDECOMPOSITION:  $A = Q \cdot D \cdot Q^T$

$\downarrow$                              $\downarrow$   
 ORTHONORMAL MATRIX      DIAGONAL MATRIX

THE ENTRIES OF  $D$  ARE THE EIGENVALUES OF  $A$ .

SPECTRAL THEOREM: every symmetric matrix "A" has an eigendecomposition: " $A = Q \cdot D \cdot Q^T$ " where "Q" is an orthogonal matrix and "D" a diagonal matrix where entries correspond to the eigenvalues.

$X$  is  $m$  by  $n$

$$Xv = \begin{bmatrix} \langle x_1, v \rangle \\ \vdots \\ \langle x_m, v \rangle \end{bmatrix} \quad \text{← Rows of } X, v$$


---


$$v^T X^T X v = \underbrace{(x_1)^T \cdot (x_1)}_{\text{FIND } A} \sum_{i=1}^m \langle x_i, v \rangle^2 \quad \text{FIND } v$$

FIND  $A$  THAT MAXIMIZES  
THAT MAXIMIZES  $v^T \underbrace{(X^T X)}_A v$   
"MAXIMIZING A QUADRATIC FORM"

Let's try to compute "v1" now that we have all of these algebra tools.

"X" is the "mxn" matrix that corresponds to "S".

Find a "v" that maximizes the " $\langle x_i, v \rangle^2$ " or a quadratic form.

MAXIMIZE  $v^T A \cdot v$  "MAX QUADRATIC FORM"  
 $\|v\|_2 = 1$

Let's look at a simple case:  $A$  is DIAGONAL.

$A = \begin{pmatrix} \lambda_1 & & & \\ 0 & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}$

$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \geq 0$ .

PICK  $v = (1, \dots, 0)$

$A = X^T X$

$v^T A v = v_1, \dots, v_n \cdot \begin{pmatrix} \lambda v_1 \\ \vdots \\ \lambda v_n \end{pmatrix} = \sum_{i=1}^n v_i^2 \cdot \lambda_i$

WE DON'T KNOW IF  $A$  IS DIAGONAL IN GENERAL  
 $A$  IS "ALMOST" DIAGONAL.

$e_1 = (1, 0, \dots, 0)$

$A = Q \cdot D \cdot Q^T$

CHOOSE  $v = (Q \cdot e_1)$

DIAOGONAL

HIGHLIGHTS  
 TOP EIGENVECTOR OF  $A$

Finally we reduce the problem to maximize a quadratic form: " $v^T \cdot A \cdot v$ ", where " $A=X^T \cdot X$ ".

First assumption: " $A$ " is diagonal and all eigenvalues are greater or equal than "0" with  $\lambda_1 > \lambda_2 > \dots > \lambda_n > 0$ .

Which " $v$ " should we take considering norm should be "1"? We should pick  $v = \{1, 0, \dots, 0\}$  to maximize the result.

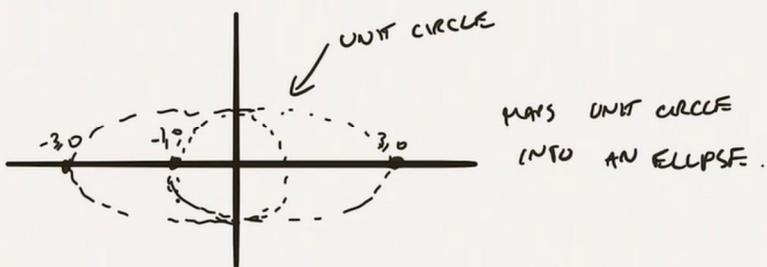
Unfortunately, " $A=X^T \cdot X$ " (covariance matrix). And we don't know if " $A$ " is a diagonal matrix or not. But we actually know that: " $A=Q \cdot D \cdot Q^T$ ". " $A$ " is actually "almost" diagonal.

So renaming:  $e_1 = \{1, 0, \dots, 0\} \rightarrow v = Q \cdot e_1$ " and this corresponds to the top eigenvector of " $A$ " or the eigenvector corresponding to the largest eigenvalue of " $A$ ".

LAST TIME: WE ALSO DISCUSSED THE "EASY CASE"

WHEN A was A DIAGONAL MATRIX

$$\begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$$



RECALL FROM LINEAR ALGEBRA: ROTATION MATRICES

ORTHOGONAL MATRICES

FOR EXAMPLE

$$\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

ROTATE  
θ degrees  
COUNTERCLOCKWISE

$$\begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix}$$

ROTATE  
YUK  
ATES θ degrees  
CLOCKWISE

We will start now talking about another algorithm related to PCA.

Remember we were studying the following optimization problem:

$$\max(v, \|v\| = 1) v^T \cdot A \cdot v, \text{ where } A \text{ is the covariance matrix } (X^T \cdot X \text{ or } \frac{1}{n} \cdot X^T \cdot X)$$

We discussed an easy case when "A" is a Diagonal matrix.

From Algebra, this "A" matrix or diagonal matrix is going to turn a unit circle into an ellipse. From Algebra, we also remember rotation matrices which are orthogonal matrices which rotate our coordinate axis:

$$\text{Rotation matrix } \theta \text{ degrees counterclockwise} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

$$\text{Rotation matrix } \theta \text{ degrees clockwise (transpose)} = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix}$$

$\begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$  THE solution to  $\nabla_{\mathbf{v}} \frac{1}{2} \mathbf{v}^T \mathbf{A} \mathbf{v}$  s.t.  $\|\mathbf{v}\|_2 = 1$  is  $\mathbf{v}^T \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{v}$   
 $\mathbf{v} = (1, 0)$

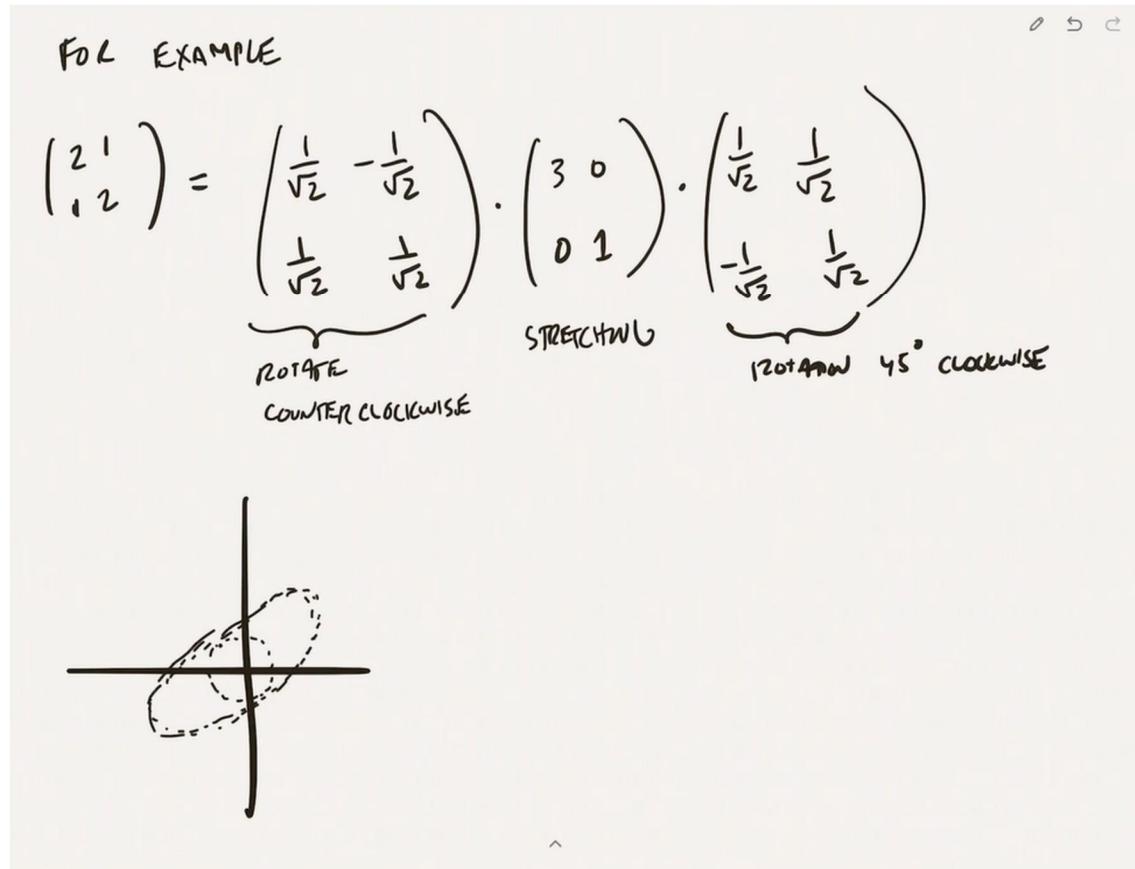
$\underline{\mathbf{v}^T \mathbf{A} \mathbf{v}} = \sum_{i=1}^n v_i^2 d_i$        $\sum v_i^2 = 1$  corresponds to choosing 3  
 $\mathbf{A}$  is diagonal  $\mathbf{v} = (1, 0)$   
 $\lambda_1, \dots, \lambda_n \geq 0$

For the diagonal matrix below:

$$\begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$$

The "v" solution with max variance with  $\|\mathbf{v}\| = 1$  corresponds to direction:

$\mathbf{v} = \{1, 0\}$  as it corresponds to the element where eigenvalue takes the maximum value "3".



We have these rotation matrices and we would like to make sure that for any covariance matrix we're given (that is to say any " $X^T \cdot X$ " matrix) we can actually write that covariance matrix as a diagonal matrix that has been rotated ( $Q \cdot D \cdot Q^T$  decomposition).

For example:

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \cdot \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} = Q \cdot D \cdot Q^T$$

Where "Q" rotates counterclockwise 45°, "Q<sup>T</sup>" rotates 45° clockwise and "D" stretches the ellipse.

0 5 2

SPECTRAL THM: ANY SYMMETRIC MATRIX CAN BE WRITTEN AS  $Q D Q^T$  WHERE  $Q$  IS ORTHOGONAL AND  $D$  IS DIAGONAL WITH REAL VALUES ON THE DIAGONAL, EIGENVALUES.

FURTHERMORE: IF  $A = \underline{X^T X}$  THEN ALL EIGENVALUES  $\geq 0$ .

CLAIM 1: FOR ANY  $v$ ,  $v^T A v \geq 0$

BECAUSE  $A = X^T X$   $v^T A v = (\underline{Xv})^T \cdot \underline{Xv} \geq 0$ .

CLAIM 2  $A$  CANNOT HAVE NEGATIVE EIGENVALUES. (PROOF IS BY CONTRADICTION)

Let's assume by contradiction that  $\lambda_i < 0$  ( $i$ th eigenvalue is negative)

$A = Q D Q^T$  Let's consider vector  $Q \cdot e_i$   $(\underbrace{0 0 0 0 0}_{i\text{th}} \underbrace{1 0 0 0 0}_i) = e_i$

$v = Q \cdot e_i$   $v^T A v$

$$\cancel{e_i^T Q^T} \cancel{Q D} \cancel{Q^T Q} e_i = \cancel{e_i^T} \cancel{D e_i} < 0$$

Let's restructure the spectral theorem:

Any symmetric matrix can be written as " $Q \cdot D \cdot Q^T$ " where " $Q$ " is orthogonal and " $D$ " is diagonal with real values on the diagonal and "0" elsewhere which correspond to the eigenvalues.

Furthermore: if " $A = X^T \cdot X$ " then all eigenvalues are greater or equal than "0".

CLAIM 1: For any " $v$ " then " $v^T \cdot A \cdot v > 0$ "

$v^T \cdot A \cdot v = (X \cdot v)^T \cdot X \cdot v > 0$ , it cannot be negative as it is the product with itself

CLAIM 2: " $A$ " cannot have negative eigenvalues. Proof by contradiction: let's assume that  $\lambda < 0$  and we take " $v$ " the maximum variance in the " $i$ "th element where  $\lambda_i < 0$  ( $e_i = \{0, 0, 1, 0, 0\}$ ).

$$\begin{aligned} \lambda_i < 0 \rightarrow A = Q \cdot D \cdot Q^T: Q \cdot e_i = v \rightarrow v^T \cdot A \cdot v = v^T \cdot Q \cdot D \cdot Q^T \cdot v > 0 \\ \rightarrow e_i^T \cdot Q^T \cdot Q \cdot D \cdot Q^T \cdot Q \cdot e_i = e_i^T \cdot I \cdot D \cdot I \cdot e_i = e_i^T \cdot D \cdot e_i = 1 \cdot \lambda_i \cdot 1 > 0 \end{aligned}$$

Therefore,  $\lambda_i > 0$  otherwise claim 1 is not possible.

Recap

TO RECAP:

- PCA:
- 1) SUBTRACT THE MEAN FROM YOUR DATA
  - 2) NORMALIZE THE COLUMNS OF YOUR DATA
  - 3) COMPUTE EIGENVALUE/EIGENVECTOR DECOMPOSITION OF YOUR MATRIX

$$Q \cdot D \cdot Q^T$$

- 4) THE FIRST K ROWS OF  $Q^T$  ARE THE K EIGENVECTORS WE ARE LOOKING FOR

TO GET K PRINCIPAL COMPONENTS.

PCA is:

1. Subtract the mean from our data.
2. Normalize the columns of our data to have similar range or variance.
3. Compute the eigenvalue/eigenvector decomposition of our matrix: " $Q \cdot D \cdot Q^T$ ".
4. The first "k" rows of " $Q^T$ " are the "k" eigenvectors we are looking for: top "k" principal components.

Prove that "i"th row of " $Q^T$ " is an eigenvector of " $A = Q \cdot D \cdot Q^T$ ":

Let's consider a vector " $v = Q \cdot e_i$ " where " $e_i$ " is a null vector except in the "i"th element where it takes value "1".

$$\begin{aligned} A \cdot v &= A \cdot (Q \cdot e_i) = (Q \cdot D \cdot Q^T) \cdot (Q \cdot e_i) = Q \cdot D \cdot I \cdot e_i \\ &= \lambda_i \cdot Q \cdot e_i, \text{ therefore } Q \cdot e_i \text{ must be an eigenvector of } A \end{aligned}$$

There is another problem when trying to decompose matrix "A". There are polynomial running algorithms for computing SVD and QDQ decomposition.

## SVD – Singular Value Decomposition of a Matrix

SVD

- "NETFLIX CHALLENGE PROBLEM"  
EQUIVALENT TO A "MATRIX COMPLETION" PROBLEM

PREDICT WHICH USERS WILL LIKE CERTAIN MOVIES

This is probably the most important method for decomposing a matrix into simpler structures so that we can really understand some structural properties about the data matrix we're given.

It is used in a lot of different areas in machine learning.

Let's see an introduction: "Netflix challenge problem" equivalent to "matrix completion" problem.

In Netflix we would like to predict which users will like certain movies.

We have a giant matrix where:

Rows: Netflix subscribers.

Columns: Movies.

Entry in this matrix ( $i, j$ ) corresponds to the rating that user " $i$ " would give to movie " $j$ " (it is a Real number).

This is a giant matrix as you can guess but we only know some of the entries. But Netflix knows some information: some entries of this matrix are known and they can guess your rate.

Netflix wants to take advantage of some information here and given movies you liked(movieA, movieB, movieC) and movies you disliked it wants to predict whether you will like or not other movies.

At a high

Before:

**HIGH LEVEL**

$$\begin{pmatrix} 1 & ? & ? \\ ? & 2 & ? \\ ? & 6 & 9 \\ ? & ? & 3 \\ 4 & 4 & ? \end{pmatrix}$$

ADDITIONAL INFORMATION: EACH ROW IS A MULTIPLE OF OTHER ROWS

SOME ENTRIES ARE KNOWN; OTHERS ARE MISSING / ?'S

GOAL: REPLACE '?' WITH NUMBERS THAT REPRESENT TRUE PREFERENCES.

After:

**HIGH LEVEL**

$$\begin{pmatrix} 1 & 1 & \frac{3}{2} \\ 2 & 2 & 3 \\ 6 & 6 & 9 \\ 2 & 2 & \frac{3}{2} \\ 4 & 4 & 6 \end{pmatrix}$$

ADDITIONAL INFORMATION: EACH ROW IS A MULTIPLE OF OTHER ROWS  
 $\equiv$  THE MATRIX HAS RANK-1

RANK-0 MATRIX  $\equiv$  ALL ZEROS MATRIX  
RANK-1 MATRIX  $\equiv$  ALL ROWS ARE MULTIPLES OF EACH OTHER  
(COLUMNS ARE MULTIPLES OF EACH OTHER)

At a high level we have:

- A giant matrix.
- Some  $(i,j)$  entries are known (number known).
- Some other entries  $(i,j)$  are missing "?".

Goal: replace entries where we don't know the preference and replace the "?" with a number.

In general this is a really hard problema.

With just a matrix and a bunch of entries it is really difficult to solve the problema without knowing anything else.

If we are given some more information about the matrix there are certain circumstances where we can begin to fill in the entries ourselves:

- Each row is a multiple of the other rows  $\approx$  the matrix has Rank 1.

Then we're able to complete the matrix.

Rank-0 matrix  $\approx$  all zeros matrix

Rank-1 matrix  $\approx$  all rows/columns are multiples of each other.

EQUIVALENTLY IF WE HAVE A RANK-1 MATRIX

$$A = u \cdot v^T$$

)      u and v are vectors  
 ↘      ↗  
 mxn      m×1      nx1  
 MATRIX      VECTOR      VECTOR

ij<sup>th</sup> entry of A  
 $= u_i \cdot v_j$

OUTER PRODUCT

$$A = u \cdot v^T$$

$$A = \begin{bmatrix} u_1 \cdot v^T \\ u_2 \cdot v^T \\ \vdots \\ u_m \cdot v^T \end{bmatrix} \quad \begin{bmatrix} v_1 \cdot u \\ v_2 \cdot u \\ \vdots \\ v_n \cdot u \end{bmatrix}$$

Equivalently if we have a Rank-1 matrix "A" then it can be written as: " $A = u \cdot v^T$ " ( $m \times n$ ) where " $u$ " ( $m \times 1$ ) and " $v$ " ( $n \times 1$ ) are vectors and " $\cdot$ " is the outer product.

$$A(i, j) = u_i \cdot v_j$$

Now consider case where  $A$  is a rank-2 matrix  
 This means  $A$  is the sum of 2 rank-1 matrices  
 (and  $A$  is not rank-1)

$$A = u \cdot v^T + w \cdot z^T$$

Now let's consider a case where "A" is a Rank-2 matrix:

This means "A" is the sum of 2 Rank-1 matrices and "A" is not Rank-1.

$$A = u \cdot v^T + w \cdot z^T$$

$$A(i, j) = u_i \cdot v_j + w_i \cdot z_j$$

There is an additional way to write a Rank-2 matrix:

$$A = \begin{pmatrix} u_1 & w_1 \\ u_2 & w_2 \end{pmatrix} \cdot \begin{pmatrix} v_1 & v_2 \\ z_1 & z_2 \end{pmatrix}$$

DEFINE THE SINGULAR VALUE DECOMPOSITION OF A MATRIX

EVERY MATRIX  $A = U \cdot S \cdot V^T$

columns of  $U$  ARE LEFT SINGULAR VECTORS

$U$  IS  $m \times n$  ORTHOGONAL MATRIX

$S$  IS  $m \times n$  DIAGONAL MATRIX

$V^T$  IS  $n \times n$  ORTHOGONAL MATRIX

ROWS OF  $V^T$  ARE RIGHT SINGULAR VECTORS

ENTRIES OF  $S$   $s_1 > s_2 > \dots > 0$

$A = \sum_{i=1}^{\min(n,m)} s_i u_i v_i^T$  OVER PRODUCT

SINGULAR VALUES ARE UNIQUE  
SINGULAR VECTORS ARE NOT UNIQUE  
SVD CAN BE COMPUTED IN TIME  $O(m^2n)$  or  $O(n^2m)$  (WHICHEVER IS SMALLER)

We're ready to define the Singular Value Decomposition (SVD) of a matrix.

So every matrix "A" can be written as:

$$A = u \cdot S \cdot v^T$$

Where "u" is a  $m \times m$  orthogonal matrix where columns of "u" are "left singular values".

"S" is a  $m \times n$  diagonal matrix where all entries are positive and orders in decreasing way:  $s_1 > s_2 > \dots > s_n$ . " $s_i$ " are the singular values.

"v" is a  $n \times n$  orthogonal matrix where rows of " $v^T$ " are "right singular vectors".

$$A = \sum_{i=1}^{\min(n,m)} s_i \cdot \bar{u}_i \cdot \bar{v}_i^T$$

The singular values (" $s_i$ ") are unique.

The singular vectors are not unique (we can have several orthogonal matrices "u" and "v").

The SVD can be computed in time  $O(m^2 \cdot n)$  or  $O(n^2 \cdot m)$  whichever is smaller.

We won't be going into the algorithm that computes SVD but it is interesting to know the time complexity.

WE COULD DEFINE A MATRIX TO HAVE RANK K

$$\text{IF } \underbrace{\begin{pmatrix} n \\ m \\ A \end{pmatrix}}_{\text{m}} = \underbrace{\begin{pmatrix} k \\ m \\ Y \end{pmatrix}}_{\text{m}} \cdot \underbrace{\begin{pmatrix} n \\ Z^T \end{pmatrix}}_{k} \text{ (AND } A \text{ IS NOT RANK } 0, \dots, k-1)$$

GIVEN A MATRIX A, IT WOULD BE GREAT IF WE COULD

FIND A FACTORIZATION  $Y \cdot Z^T$  WHERE  $K$  IS REALLY SMALL

$A$   $\leftarrow$  m·n ENTRIES CAN BE WRITTEN USING ONLY  $k(n+m)$  NUMBERS

We could define a matrix "A" to have Rank "k" if "A" can be written as:

$$(A)_{mxn} = (Y)_{mxk} \cdot (Z^T)_{kxn}$$

The resulting matrix "A" has orden "k" but not 0,1,...,k-1. Just "k"

Given a matrix "A" it would be great if we could find a factorization of the form above: "A=Y·Z" where "k" is the smallest value possible.

This would mean that "A" can be written using only " $k(n+m)$ " numbers. By representing this way we are compressing something storing the most information.

ONE WAY WE COULD REWRITE A / represent A

$$A = U S V^T$$

$m \times n$

$m \times n$  matrix  $\xrightarrow{\quad}$   $m \times k$  matrix  $\xrightarrow{\quad}$   $k \times n$  matrix, diagonal

$S$   $\xrightarrow{\quad}$   $k \times k$  matrix  $\xrightarrow{\quad}$   $k \times k$  matrix, diagonal

$A \approx U' \cdot S \cdot V^T$

$m \times n$   $\xrightarrow{\quad}$   $m \times k$   $\xrightarrow{\quad}$   $k \times n$

$\downarrow$   $\downarrow$   $\downarrow$

$m \times k$   $\quad$   $k \times k$   $\quad$   $k \times n$

TAKEN ONLY  $k$  columns from  $U$   $\quad$   $\quad$   $\quad$   $k$  rows from  $V^T$

There is one way we could rewrite or represent "A" taking a look at its SVD:

$$A = u \cdot s \cdot v^T$$

Taking a look at its singular values  $\{s_1, s_2, \dots, s_n\}$  we could decide to zero out some of them:

i.e.  $\{100, 99, 98, \dots, 0.4, 0.3, 0.2\} \rightarrow \{100, 99, 98, \dots, 0, 0, 0\}$

In this case, we could take "S" which was a  $m \times n$  matrix and zero out everything except some  $k \times k$  matrix that is diagonal but just keeps the biggest "k" singular values that account for a small chunk of the matrix but 99% of the original matrix values.

$$(A)_{m \times n} = (u)_{m \times k} \cdot (S)_{k \times k} \cdot (v^T)_{k \times n}$$

Bear in mind "A" is still a  $m \times n$  matrix but that is less accurate and the ones that correspond to the hopefully smaller and less meaningful singular values.

Frobenius norm of a matrix

X

o s

DEFINIE FROBENIUS NORM OF A MATRIX TO BE

$$\sqrt{\sum_{i,j} A_{i,j}^2}$$

GIVEN: A MATRIX A

GOAL: FIND A MATRIX A' SUCH THAT A' HAS RANK K

AND MINIMIZES  $\|A - A'\|_F$  OVER ALL RANK K MATRICES.

ANSWER: COMPUTE SVD OF A AND TAKE TOP K SINGULAR VECTORS AND VALUES.

$$\text{Frobenius norm} = \sqrt{\sum_{i,j} A_{i,j}^2}$$

**Remember our goal:**

Given a matrix "A" we want to find a matrix "A'" with Rank "k" and minimizes  $\|A - A'\|_F$  over all Rank "k" matrices.

- In other words, we want to find the closest matrix A' with Rank "k" such as we get a minimum Frobenius norm (so we can say that "A" and "A'" are the same matrix).
- $A - A'$  is the subtraction of matrices.

**Solution:**

Compute SVD of "A" and take top "k" singular vectors and values.

$\times$  $\theta$ 

$$A = USV^T$$

Answer is  $A' = u' s' v'^T$

$$\left( \underbrace{\begin{array}{|c|} \hline u' \\ \hline \end{array}}_{U} \right) \cdot \left( \begin{array}{|c|} \hline s' \\ \hline \end{array} \right) \cdot \left( \begin{array}{|c|} \hline v^T \\ \hline \end{array} \right)$$

rows of  $v^T$

$A'$  is still  $m \times n$

Remember:

$$A = u \cdot S \cdot v^T; A' = u' \cdot S' \cdot v'^T, \text{ where } \dim(u') < \dim(u)$$

Matrix completion

## MATRIX COMPLETION

A

- REPLACE  $\text{?}$  WITH EITHER 0  
AVG VALUE OF KNOWN ENTRIES  
AVG VALUE IN THAT COLUMN  
OR ROW.
- FIND BEST RANK K APPROXIMATION TO  
A AFTER FILLING IN THE  $\text{?}$ 's.
- OUTPUT THIS BEST RANK K APPROXIMATION.



Given matrix "A" with a lot of unknown entries (i,j).

- Replace unknown entries (i, j) with either "0", average value on the known entries, average value in that column or row, suggested to subtract the average of all known entries and setting unknown cells to "0" (new average), etc.
- Find the best Rank "k" approximation of "A" after filling in the unknown entries (i, j).
- Output this best Rank "k" approximation of "A".
- Hope there is good information in the entries we observe so that this best Rank "k" approximation is a good approximation to the rest of those values.

Choosing "k"

✖ ⌂ ⌃

ONE TYPICAL HEURISTIC FOR CHOOSING K  
IS TO TAKE ENOUGH SINGULAR VALUES SO  
THAT THE SUM OF REMAINING VALUES  $\leq \frac{1}{10}$  OF VALUES  
YOU DONT TAKE.

APPLICATION: LINEAR REGRESSION

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|^2$$

$$A \text{ is } mxn \text{ matrix}$$

$$b \in \mathbb{R}^m$$

EASY CASE  
 $A = D$  (DIAGONAL)

$$D = \begin{pmatrix} d_1 & & \\ & d_2 & \\ & & \ddots \\ & & & 0 \end{pmatrix}$$

$$x_1 = \frac{b_1}{d_1}, \quad x_2 = \frac{b_2}{d_2}$$

$$d_j = 0 \Rightarrow x_j = 0$$

$D^+ \leftarrow \begin{pmatrix} \frac{1}{d_1} & 0 & \dots & 0 \\ 0 & \frac{1}{d_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$

$D^+ \rightarrow$  PSEUDOINVERSE OF D.

SUMMARIZE:  $x = D^+ \cdot b$

"k" is a hyperparameter and we need to experiment with different values.

For choosing "k" is to take enough singular values so that the Frobenius norm is within a range or the sum of the remaining singular values we didn't take is less than a fraction of the values we took (i.e. 1/10).

**Application 1:** linear regression.

Goal:

$$\min(x \in R) \|Ax - b\|^2, A \text{ is } mxn \text{ matrix}$$

Easy case:  $A = D$  (diagonal matrix)

In this case, the optimum value for "x1":

$$x_1 = \frac{b_1}{d_1}, x_2 = \frac{b_2}{d_2}$$

"x" would be the optimal vector here for this case.

Solution:

$$\text{Solution} = \begin{pmatrix} \frac{1}{d_1} & 0 & 0 \\ 0 & \frac{1}{d_2} & 0 \\ 0 & 0 & \frac{1}{d_n} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_n \end{pmatrix} = D^\dagger \cdot b$$

" $D^\dagger$ " is the pseudo-inverse of " $D$ ".

Summary: " $x = D^\dagger \cdot b$ ".

$$\begin{aligned}
 \|Ax - b\|^2 &\equiv \min_x \|USV^T x - b\|^2 \\
 \|Ux\| = \|x\| &\quad \equiv \min_x \|SV^T x - U^T b\|^2 \\
 \underline{y = V^T x} &\quad \equiv \underline{V y = x} \\
 &\quad = \min_x \|Sy - U^T b\|^2 \\
 \Rightarrow y &= S^+ \cdot U^T b \\
 \Rightarrow x &= VS^+ U^T b
 \end{aligned}$$

General solution:

$$\min(x \in R) \|A \cdot x - b\|^2 = \min(x \in R) \|u \cdot S \cdot v^T \cdot x - b\|^2$$

Where  $u$  is an orthogonal matrix with "norm( $u$ ) = 1" that rotates the vector but preserves the norm.

The previous expression is equivalent to the following multiplying by " $u^T$ " as it does not affect the norm of any vector:

$$\min(x \in R) \|u \cdot S \cdot v^T \cdot x - b\|^2 = \min(x \in R) \|S \cdot v^T \cdot x - u^T \cdot b\|^2$$

If now we make a new variable substitution: " $y = v^T \cdot x$ " (1)  $\rightarrow$  " $v \cdot y = x$ " (2):

$$\min(x \in R) \|S \cdot y - u^T \cdot b\|^2, \text{ where } S \text{ is a diagonal matrix} \rightarrow$$

$$\min(x \in R) \|S \cdot y - u^T \cdot b\|^2$$

This expression is similar to the "easy case" expression where the solution was: " $x = D^T \cdot b$ " for a problem  $\|A \cdot x - b\|^2$ .

Applying the substitution equations (1) and (2):

$$y = S^T \cdot u^T \cdot b \xrightarrow{v \cdot y = x} x = v \cdot S^T \cdot u^T \cdot b$$

This is the solution for linear least squares.

Using the normal equations we had this annoying issue we had to take the matrix inversion. Using SVD to find directly a simple expression for the optimal solution. Remember "A" might not be invertible.

### Applying SVD

RECALL PCA: FIND EIGENDECOMPOSITION OF COVARIANCE MATRIX

$$X^T X = \underbrace{(U S V^T)^T \cdot U S V^T}_{V S U^T \cdot U \cdot S V^T}$$

SINGULAR VALUES VS EIGENVALUES? . . .

SINGULAR VALUES ARE THE SQUARE ROOT OF EIGENVALUES OF  $X^T X$

$V S^2 V^T$  ORTHO ORTHO

EIGENDECOMPOSITION  
RIGHT SINGULAR VECTORS OF  $X$  (ROWS OF  $V^T$ )  
ARE THE PRINCIPAL COMPONENTS (TOP EIGENVECTORS OF  $X^T X$ ).)

**Recall PCA:** Find eigen decomposition of a covariance matrix.

$$\begin{aligned} X^T \cdot X &= (U \cdot S \cdot V^T)^T \cdot (U \cdot S \cdot V^T) = (V \cdot S \cdot U^T) \cdot (U \cdot S \cdot V^T) = (V \cdot S) \cdot \cancel{U^T \cdot U} \cdot (S \cdot V^T) \\ &= v \cdot S^2 \cdot v^T \end{aligned}$$

Because "v" is orthogonal and therefore "v<sup>T</sup>" is orthogonal too we got it.

This is the Eigendecomposition of  $X^T \cdot X$ .

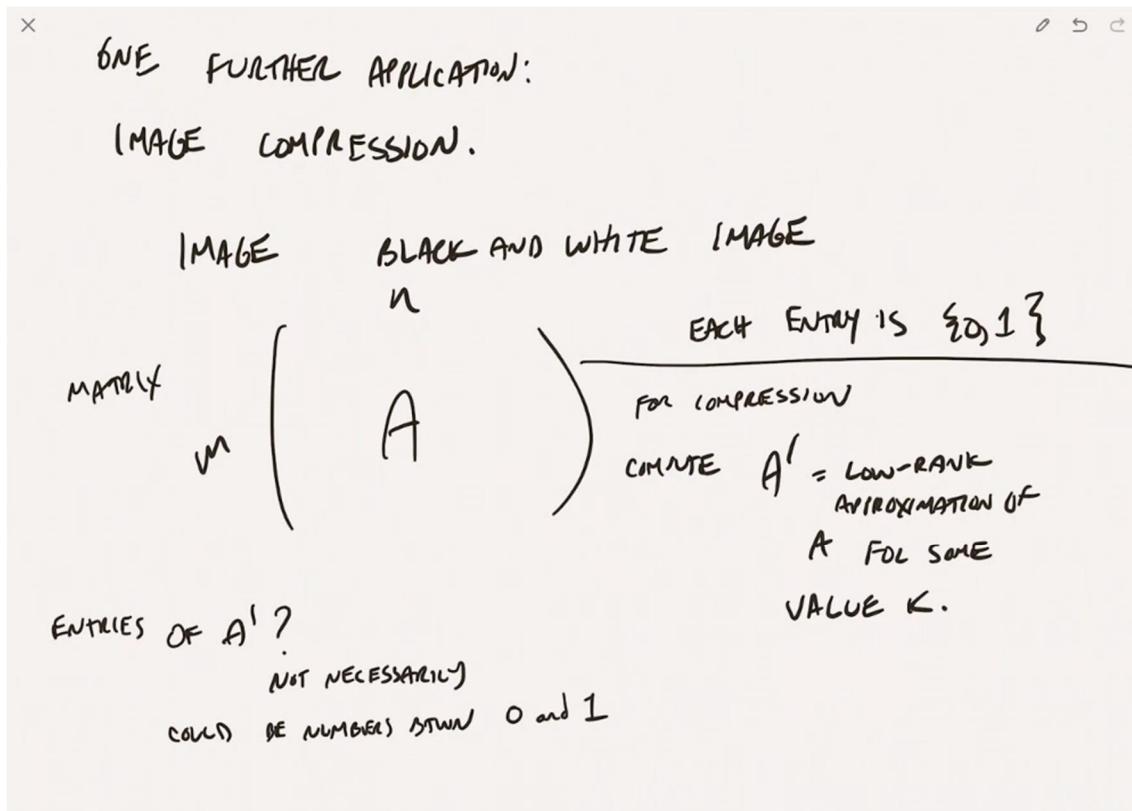
Right singular vectors of "X" ARE the principal components (top eigenvectors of "X<sup>T</sup>·X").

SVD can give you more information than the PCQ or eigendecomposition of our matrix. When we compute SVD we can read off the rows of " $V^T$ ".

Singular values vs eigenvalues?

$$\lambda_i = s_i^2 \rightarrow s_i = \sqrt{\lambda_i}$$

So the SVD of " $X^T \cdot X$ " is a lot more powerful than the PCA of " $X$ ".



**One further application:** Image compression

We have a black and white image so the image is a  $m \times n$  "A" matrix where each entry is {0,1}.

For compression:

Compute " $A'$  = low-rank approximation of  $A$ " for some value " $k$ " to experiment with.

What will the entries of " $A'$ " look like if we apply SVD: it will be an approximation of " $A$ ".

Could be numbers between "0" and "1" that could be associated to a shade of gray.

It will just come up with some new image.

## Maximum likelihood estimation

Introduction

$$\hat{\theta} = \frac{4}{5}$$

$\Theta = 0.9$ , vs.  $0.1$

If  $\underline{\Theta = 0.1}$ ,  $\text{Prob}(\text{HHHHT} | \Theta = 0.1) = \Theta^4(1-\Theta)^1$   
 $= \underline{(0.1)^4}(0.9)^1$   
 $\approx 0.00009 \downarrow$

If  $\underline{\Theta = 0.9}$ ,  $\text{prob}(\text{HHHHT} | \Theta = 0.9) = \Theta^4(1-\Theta)^1$   
 $= \underline{(0.9)^4}(0.1)^1 \uparrow$

It is a way to estimate parameters for giving probabilistic distribution from observation.

Let's imagine an example where we have a biased coin so we can model the result.

X = random variable every time we draw a coin defined by a probability distribution.

$$P[X = \text{tail}] = 1 - \theta$$

$$P[X = \text{head}] = \theta$$

The parameter “θ” is unknown but what we do know is that we observe a sequence of draws from its distribution.

The question to estimate the parameter “θ” through the random observation drawn.

This is a very simple question and intuition tells us that we can easily guess that the estimation of “θ” based on observation could be:

Observation: HHHHT = 4 heads, 1 tail

$$\theta = 4/5$$

This is a very simple intuitive justification.

Problem: is there a mathematical and principle way for estimating theta for complex parameters.

Maximum likelihood estimation does exactly this task.

Let's simplify the problem: we have two choices of “θ”:

$$\theta = 0.9, \theta = 0.1$$

We want to decide which one is more likely.

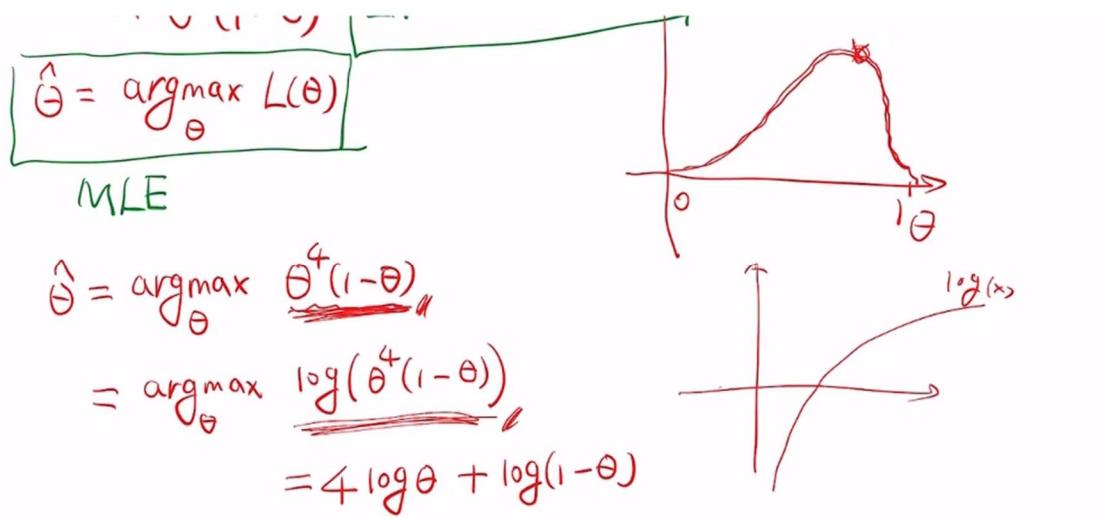
Let's assume  $\theta = 0.9$ . Which is the probability that we can say that the given data set  $\theta = 0.1$ .

$\text{Prob}(\text{HHHHT}|\theta=0,1) = \theta^4 \cdot (1-\theta)^1$  = Probability of seeing head 4 times and tails 1 time = 0.00009.

$\text{Prob}(\text{HHHHT}|\theta=0,9) = \theta^4 \cdot (1-\theta)^1$  = Probability of seeing head 4 times and tails 1 time = 0.009.

The likelihood that  $\theta = 0.9$  is higher.

We can actually get a function (the likelihood function) that we can evaluate for each parameter and we can analyze its maximum.



Let's implement this idea of obtaining a function that we can analyze to find our optimal value “ $\theta$ ”.

$$L(\theta) = \text{Prob}(\text{HHHHT}|\theta) = \theta^4 \cdot (1 - \theta), \text{This is the Likelihood function}$$

We want to get the function to estimate maximum “ $\theta$ ” and this is provided by the Maximum Likelihood Function.

We can maximize this by taking logs:

$$\text{MLE}(\theta) = \log(\theta^4 \cdot (1 - \theta))$$

If a value of “ $\theta$ ” maximizes the original function it also maximizes the log of that function as “log” operator is monotonically increasing.

$$\text{MLE}(\theta) = \log(\theta^4 \cdot (1 - \theta)) = \log(\theta^4) + \log(1 - \theta) = 4 \cdot \log(\theta) + \log(1 - \theta)$$

Now, we could take the gradient and find the points where the gradient is zero.

Applying logs we are obtaining a log-likelihood function.

$$\begin{aligned}
 \hat{\theta} &= \underset{\theta}{\operatorname{argmax}} \quad \underline{\theta^4(1-\theta)} \\
 &= \underset{\theta}{\operatorname{argmax}} \quad \underline{\log(\theta^4(1-\theta))} \\
 &= 4 \log \theta + \log(1-\theta) \\
 &= \underline{\ell(\theta)} \triangleq \underline{\log L(\theta)}
 \end{aligned}$$

log-likelihood function

Taking the gradient of "L" (derive):

$$\text{grad}(\log L(\theta)) = \frac{4}{\theta} - \frac{1}{1-\theta} = 0 \rightarrow \theta = 4/5$$

This is exactly the point that we obtained applying logic.

$\rightarrow \nabla L(\theta)$

$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} L(\theta)$

MLE

$$\begin{aligned}
 \hat{\theta} &= \underset{\theta}{\operatorname{argmax}} \quad \underline{\theta^4(1-\theta)} \\
 &= \underset{\theta}{\operatorname{argmax}} \quad \underline{\log(\theta^4(1-\theta))} \\
 &= 4 \log \theta + \log(1-\theta) \\
 &= \underline{\ell(\theta)} \triangleq \underline{\log L(\theta)}
 \end{aligned}$$

$\nabla \ell(\theta) = \frac{4}{\theta} + \frac{-1}{1-\theta} = 0 \Rightarrow \boxed{\hat{\theta} = \frac{4}{5}}$

General form**Parameter Estimation by MLE**

- Given a set of observation  $\{x_i\}_{i=1}^n$ .
- Independent and identically distributed (iid)** following an unknown distribution  $p_*$ , from a **parametric family of distributions**:

$$\{p(\cdot | \theta) : \theta \in \Theta\}.$$

- Estimate the parameter  $\theta$ .

We're given a ser of observations  $\{x_1, x_2, \dots, x_n\}$ , in our examples either Head or Tails.

We asume each observation is independently and identically distributed for some distribution  $P^*$ .

This unknown distribution  $P^*$  belongs to a parametric set of distributions for different values of a parameter.

The problema is to estimate the parameter " $\theta$ ".

- Estimate the parameter  $\theta$ .

Likelihood function:

$$L(\theta) = P(x_1, \dots, x_n | \theta) = \prod_{i=1}^n P(x_i | \theta)$$

log-likelihood function:

$$\begin{aligned} l(\theta) &= \log L(\theta) = \log \left( \prod_{i=1}^n P(x_i | \theta) \right) \\ &= \sum_{i=1}^n \underline{\log P(x_i | \theta)} \end{aligned}$$

The idea behind is to write the Likelihood function of seeing our entire set of observation for a particula value of " $\theta$ ":

$$L(\theta) = P(x_1, x_2, \dots, x_n | \theta) = \prod_{i=1}^n P(x_i | \theta)$$

It is more convenient to use the log-likelihood function:

$$\log(L(\theta)) = l(\theta) = \log \left( \prod_{i=1}^n P(x_i | \theta) \right) = \sum_{i=1}^n \log(P(x_i | \theta))$$

Maximum Likelihood estimation: finding maximum “ $\theta$ ”.

$$\hat{\theta} = \operatorname{argmax}_{\theta}(l(\theta)) = \operatorname{argmax}_{\theta} \left( \sum_{i=1}^n \log(P(x_i | \theta)) \right)$$

Let's solve this optimization problema using the gradient or numerical algorithms.

**Maximum Likelihood Estimation :**

$$\hat{\theta} = \operatorname{argmax}_{\theta} l(\theta) = \operatorname{argmax}_{\theta} \left\{ \sum_{i=1}^n \log P(x_i | \theta) \right\}$$

Solve this optimization problem .

$\left\{ \begin{array}{l} \text{Closed form} \\ \text{Numerical Method} \end{array} \right.$

Bernoulli distribution

Let's go over more examples:

### Example: Biased Coin Revisited

- Given  $\{x_i\}_{i=1}^n$ , where  $x_i \in \{0, 1\}$ , iid drawn from Bernoulli distribution:

$$\begin{cases} \Pr(X = 1) = \frac{\exp(w)}{1 + \exp(w)}, \\ \Pr(X = 0) = \frac{1}{1 + \exp(w)}. \end{cases}$$

- Problem:** Estimate  $w$ .

In this case  $\{x_1, x_2, \dots, x_n\}$  takes binary values following a Bernoulli distribution.

Estimate “ $w$ ”. We prefer this because we don't need to constrain “ $w$ ” between 0 and 1.

This probability function enriches the realism of the problema.

- Given  $\{x_i\}_{i=1}^n$ , where  $x_i \in \{0, 1\}$ , iid drawn from Bernoulli distribution:  $\theta \in [0, 1]$
- $$\begin{cases} \Pr(X=1) = \frac{\exp(w)}{1+\exp(w)}, \\ \Pr(X=0) = \frac{1}{1+\exp(w)}. \end{cases} = \theta$$
- 
- $w \in \mathbb{R}$ .

Let's solve this problem using maximum likelihood estimation:

$$P(x_1, x_2, \dots, x_n | w) = \prod_{i=1}^n P(x_i | w)$$

Let's use logarithms:

$$\log(P(x_1, x_2, \dots, x_n | w)) = \sum_{i=1}^n \log(P(x_i | w))$$

SIDE NODE: turns out we can write probability of "x" like just taking a look at the probability function screenshot when  $X=1$  and  $X=0$ :

$$P(X) = \frac{e^{X \cdot w}}{1 + e^w}$$

$$\log P(x_1, \dots, x_n | w) = \sum_{i=1}^n \log P(x_i | w)$$

$w \in \mathbb{R}$ .

$P(x) = \frac{\exp(xw)}{1 + \exp(w)}$   
 If  $x=1$ ,  $\frac{\exp(w)}{1 + \exp(w)}$   
 If  $x=0$ ,  $\frac{1}{1 + \exp(w)}$

We have a simple formula for different values of "X" and "P" and plug in the formula:

$$\log(P(x_i | w)) = \log\left(\frac{e^{x \cdot w}}{1 + e^w}\right) = x \cdot w - \log(1 + e^w)$$

Then the log-likelihood function:

$$l(w) = \sum_{i=1}^n \log(P(x_i | w)) = \sum_{i=1}^n (x_i \cdot w - \log(1 + e^w)) = \sum_{i=1}^n x_i \cdot w - n \cdot \log(1 + e^w)$$

Introducing the average value thanks to the sum of values:

$$l(w) = n \cdot (\bar{x} \cdot w - \log(1 + e^w))$$

$$\begin{aligned} \underline{l(w)} &= \sum_{i=1}^n \log P(x_i | w) \\ &= \sum_{i=1}^n [x_i w - \log(1 + \exp(w))] \\ &= \underbrace{\left(\sum_{i=1}^n x_i\right) w}_{n \bar{x}} - n \log(1 + \exp(w)). \\ &= \cancel{n \bar{x} w} - \cancel{\log(1 + \exp(w))} \quad \boxed{\bar{x} = \frac{\sum_{i=1}^n x_i}{n}} \\ \nabla l(w) &= n \bar{x} - \frac{\exp(w)}{1 + \exp(w)} \end{aligned}$$

Let's take the gradient to evaluate the optimal values:

$$\text{grad}(l(w)) = n \cdot \bar{x} - \frac{e^w}{1 + e^w} = 0$$

At this point we can actually already estimate the parameters.

$$\begin{aligned} \nabla l(w) &= n \left( \underbrace{\bar{x} - \frac{\exp(w)}{1 + \exp(w)}}_{} \right) = 0 \\ &\quad x_i \in \{0, 1\} \\ \hat{\theta} &= \frac{\exp(\hat{w})}{1 + \exp(\hat{w})} = \bar{x} = \frac{\#(1)}{\#(1) + \#(0)} \end{aligned}$$

This leads back to the head and tails problema and we get to the same result previously seen.

Comparing these formula with the results we obtained previously and some probabilistic concepts we see that some of the values of the optimization problema are identical to some expressions we see logically.

Bear in mind we introduced a sigmoid function to be able to transfrom a {0,1} parameter into a parameter that takes all real values.

Gaussian distribution

Let's solve another problem for a gaussian distribution with two parameters.

**Example: Gaussian Distribution**

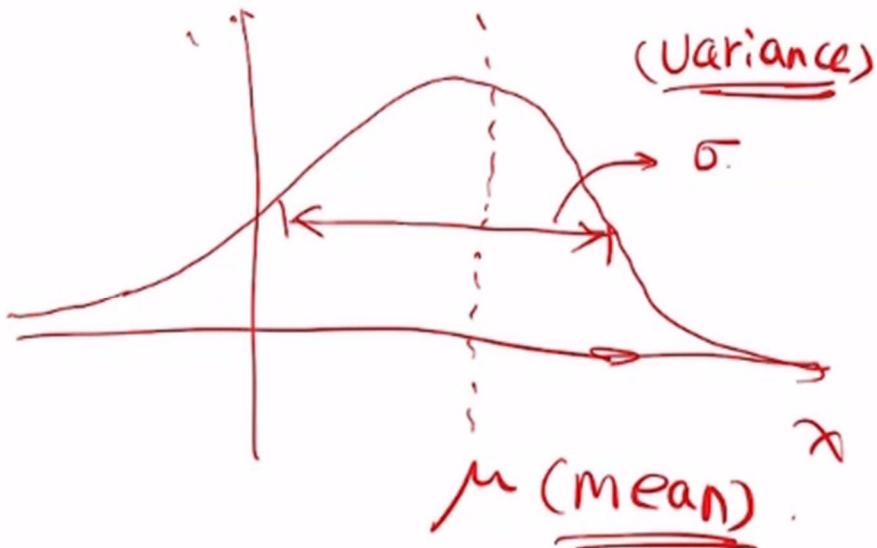
- Given  $\{x_i\}_{i=1}^n$  iid drawn from Gaussian distribution  $\mathcal{N}(\mu, \sigma^2)$ :

$$p(x | \theta) = \frac{1}{(2\pi)^{1/2}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right), \quad \theta = \{\mu, \sigma\}.$$

We again have a set of observations and our observations are drawn from a Gaussian distribution with 2 parameters:

$\mu$  = average/ mean.

$\sigma$  = deviation or variance.



We have a Gaussian distribution where " $\mu$ " and " $\sigma$ " are unknown but we do observe samples from the distribution.

Assuming the observations follow a Gaussian distribution let's practice the same procedure as before:

$$l(\theta) = \sum_{i=1}^n \log(P(x_i | \theta)) = \sum_{i=1}^n \log\left(\frac{1}{(2\pi)^{1/2} \cdot \sigma} \cdot e^{-\frac{1}{2\sigma^2}(x_i - \mu)^2}\right)$$

Applying logarithms to transform multiplications and divisions into sums and subtractions:

$$\begin{aligned}
 l(\theta) &= \sum_{i=1}^n \log \left( \frac{1}{(2\pi)^{\frac{1}{2}} \cdot \sigma} \cdot e^{-\frac{1}{2\sigma^2}(x_i - \mu)^2} \right) \\
 &= -\frac{n}{2} \cdot \log(2\pi) - n \cdot \log(\sigma) - \frac{1}{2\sigma^2} \cdot \sum_{i=1}^n (x_i - \mu)^2
 \end{aligned}$$

We need to maximize this function finding both “ $\mu$ ” and “ $\sigma$ ”.

Let's fix “ $\sigma$ ” and try to solve “ $\mu$ ” for a given value of “ $\sigma$ ”, in this case we just need to focus on analyzing this part of the equation as “ $1/2\cdot\sigma^2$ ” is constant:

$$\hat{\mu} = \operatorname{argmax}_{\mu} \left( -\sum_{i=1}^n (x_i - \mu)^2 \right) = \operatorname{argmin}_{\mu} \left( \sum_{i=1}^n (x_i - \mu)^2 \right)$$

So, the optimal estimation of “ $\mu$ ” is going to minimize the square distance from “ $x_i$ ” to “ $\mu$ ”.

$$\begin{aligned}
 l(\theta) &= \sum_{i=1}^n \log P(x_i | \theta) \\
 &= \sum_{i=1}^n \left( \log \left( \frac{1}{(2\pi)^{\frac{1}{2}} \sigma} \right) - \frac{1}{2\sigma^2} (x_i - \mu)^2 \right) \\
 &= -\frac{n}{2} \log(2\pi) - n \log \sigma - \frac{1}{2\sigma^2} \left( \sum_{i=1}^n (x_i - \mu)^2 \right) \quad \text{max } \mu, \sigma \\
 \hat{\mu} &= \operatorname{argmin}_{\mu} \left( \sum_{i=1}^n (x_i - \mu)^2 \right) \\
 &= \frac{1}{n} \sum_{i=1}^n x_i \\
 &\boxed{\sum_{i=1}^n 2(\hat{\mu} - x_i) = 0} \quad \Rightarrow \quad \hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i
 \end{aligned}$$

$$\operatorname{grad}(l(\theta)) = \frac{d}{d\mu} \left( \sum_{i=1}^n (x_i - \mu)^2 \right) = \sum_{i=1}^n 2 \cdot (x_i - \mu) \cdot 1 = 0 \rightarrow \mu = \frac{1}{n} \cdot \sum_{i=1}^n x_i$$

It means that if we derive the maximum likelihood estimation for “ $\mu$ ” it is exactly equals to the mean value of all the “ $x_i$ ” observations.

With respect to “ $\sigma$ ” when fixing “ $\mu$ ”. Remember the part “ $-n/2 \cdot \log(2\pi)$ ” is constant and we don't analyze it.

Note we are already introducing “ $\mu$ ” optimal and this is the reason the hat appears.

$$\hat{\sigma} = \operatorname{argmax}_{\sigma} \left( -n \cdot \log(\sigma) - \frac{1}{2 \cdot \sigma^2} \cdot \sum_{i=1}^n (x_i - \hat{\mu})^2 \right)$$

Taking the gradient:

$$\begin{aligned} \text{grad}(l(\theta)) &= \frac{d}{d\sigma} \left( -n \cdot \log(\sigma) - \frac{1}{2 \cdot \sigma^2} \cdot \sum_{i=1}^n (x_i - \hat{\mu})^2 \right) = -\frac{n}{\sigma} - \frac{(-2) \cdot 1}{2 \cdot \sigma^3} \cdot \sum_{i=1}^n (x_i - \hat{\mu})^2 \\ &= 0 \rightarrow \hat{\sigma}^2 = \frac{1}{n} \cdot \sum_{i=1}^n (x_i - \hat{\mu})^2 \end{aligned}$$

You will find that the estimation is actually equal to the variance (" $\sigma^2$ ").

$$\begin{aligned} \hat{\sigma} &= \underset{\sigma}{\operatorname{argmax}} \left\{ -n \log \sigma - \frac{1}{2 \sigma^2} \sum_{i=1}^n (x_i - \hat{\mu})^2 \right\} \triangleq f(\sigma) \\ \nabla f(\sigma) &= -\frac{n}{\sigma} - \frac{(-2) \cdot 1}{2 \sigma^3} \sum_{i=1}^n (x_i - \hat{\mu})^2 \\ \Rightarrow \hat{\sigma}^2 &= \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu})^2 \end{aligned}$$

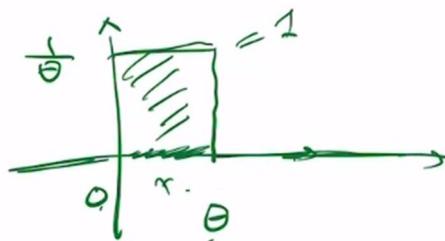
So, when it is a Gaussian distribution the maximum likelihood distribution allows us to find the best " $\sigma$ " and " $\mu$ " that fits for our observed data with the empirical functions.

### Uniform distribution

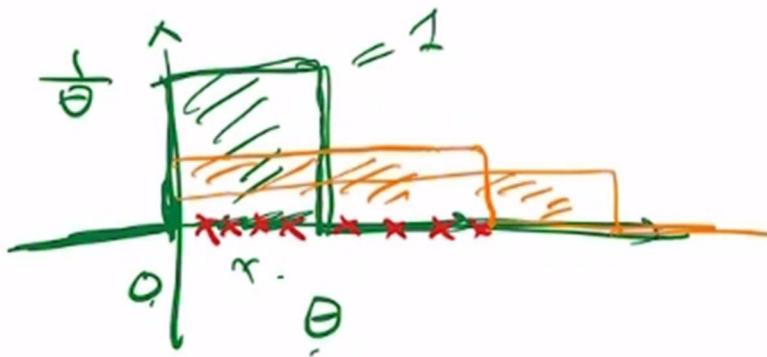
#### Example: Uniform Distribution

- Given  $\{x_i\}_{i=1}^n$  iid drawn from a uniform distribution Uniform([0,  $\theta$ ]):

$$p(x | \theta) = \begin{cases} \frac{1}{\theta} & \text{if } x \in [0, \theta] \\ 0 & \text{if otherwise} \end{cases} \quad \theta > 0$$



We have a uniform distribution between "0" and " $\theta$ " where " $\theta$ " takes positive values. We want to make sure that the density of this distribution (the integral) is "1".



Turns out that as we use larger and larger values of " $\theta$ " the rectangle gets less height. This is intuitive and can explain some random variable we observe.

Let's see the maximum likelihood estimation for this case.

$$P(x|\theta) = \frac{1}{\theta} \cdot 1, \forall x \in [0, \theta]$$

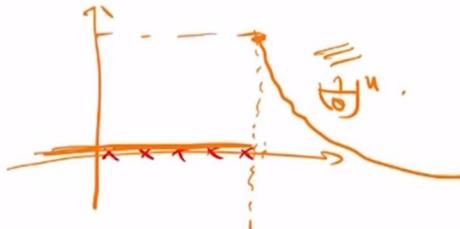
In this case the maximum likelihood estimation function:

$$L(\theta) = \prod_{i=1}^n P(x_i|\theta) = \prod_{i=1}^n \frac{1}{\theta} \cdot 1(x_i \in [0, \theta]) = \left(\frac{1}{\theta}\right)^n \cdot 1(x_i \in [0, \theta])$$

$$\begin{aligned} L(\theta): \\ \left(\frac{1}{\theta}\right)^n & \text{ if } x_i \in [0, \theta], \forall i \\ 0 & \text{ otherwise} \end{aligned}$$

Let's calculate the optimal values:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^n P(x_i|\theta) = \prod_{i=1}^n \left(\frac{1}{\theta}\right) 1(x_i \in [0, \theta]) \\ &= \underbrace{\left(\frac{1}{\theta}\right)^n}_{\frac{1}{\theta^n}} \underbrace{\prod_{i=1}^n 1(x_i \in [0, \theta])}_{\text{if } x_i \in [0, \theta]} \\ &= \begin{cases} \frac{1}{\theta^n} & \text{if } x_i \in [0, \theta] \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$



The optimum value is achieved optimizing these two conditions:

- 1) " $\theta$ " is larger or equal than the datapoints (otherwise the product gets "0").
- 2) " $\theta$ " is as Small as possible as the MLE decreases as " $\theta$ " increases.

This is only achieved taking " $\hat{\theta}$ " =  $\max(x_1, x_2, \dots, x_n)$ .

$$\hat{\theta} = \max(x_1, x_2, \dots, x_n)$$

For instance:

Draws ( $n = 6$ ) = 1,1,2,4,5,3

$$\max(1,1,2,4,5,3) = 5$$

with  $\theta = 4 \rightarrow L(\theta=4) = 0$ , as for  $x=5 \rightarrow "x"$  does not belong to " $\theta$ " and it cascades "0" for all the non-zero products when " $x < 5$ ".

$$L(\theta=5) = 0,000064$$

$$L(\theta=6) = 2,14335E-05, \text{ which is smaller than } L(\theta=5).$$

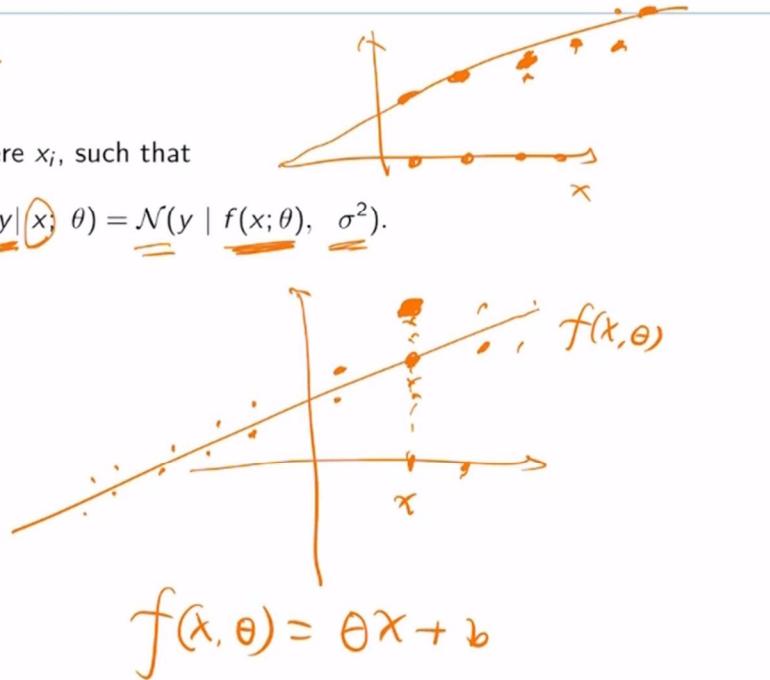
### Linear regression

#### MLE for Regression

Given  $\{(x_i, y_i)\}_{i=1}^n$ , where  $x_i$ , such that

$$p(y|x; \theta) = \mathcal{N}(y | f(x; \theta), \sigma^2).$$

Estimate  $\theta$  and  $\sigma^2$ .



MLE can be also used for more complex cases like linear regression or classification and we would like to understand the relation between observed " $x_i$ " and estimated " $y_i$ ".

We want to predict the " $y_i$ " value given " $x_i$ ". And it sounds reasonable to assume that " $y_i$ " is generated for some distribution (biased algorithm) because the generation process of " $y_i$ " is typically noisy.

We can assume conditioning of " $x_i$ " the " $y_i$ " is following some Gaussian distribution whose mean is a function and then we have a variance.

For instance, let's assume there is a simple linear line that given an observed value "x" then the "y" is generated from some Gaussian noise whose mean and variance that can return a value within a range.

The problem is to estimate a parameter " $\theta$ " mean as well as the noise magnitude " $\sigma^2$ ".

Instead of maximizing a single likelihood we need to maximize the conditional likelihood:  $x|y$ .

$$l(\theta) = \sum_{i=1}^n \log(P(y_i|x_i, \theta)) = \sum_{i=1}^n \left( -\frac{n}{2} \cdot \log(2 \cdot \pi) - n \cdot \log(\sigma) - \frac{1}{2 \cdot \sigma^2} \cdot (y_i - f(x_i, \theta))^2 \right)$$

Where " $f(x_i, \theta) = \mu = y_i$ " is the mean if you remember the original gauss distribution function. In this case this is the mean deviation.

In this case we want to maximize:

$$\max(l(\theta)) = \max \left( \sum_{i=1}^n -(y_i - f(x_i, \theta))^2 \right) = \min \left( \sum_{i=1}^n (y_i - f(x_i, \theta))^2 \right)$$

This is equivalent to the least square estimator.

$$\begin{aligned} l(\theta, \sigma) &= \sum_{i=1}^n \log P(y_i | x_i, \theta) \\ &= \sum_{i=1}^n \left[ -\frac{n}{2} \log(2\pi) - n \log \sigma - \sum_{i=1}^n \frac{1}{2\sigma^2} (y_i - \hat{f}(x_i, \theta))^2 \right] \\ \max_{\theta} l(\theta, \sigma) \Rightarrow \min_{\theta} &\sum_{i=1}^n (y_i - \hat{f}(x_i, \theta))^2 \end{aligned}$$

*Least Square Estimation*

Let's now estimate the best " $\sigma$ ": empirical squared-loss:

$$\sigma^2 = \frac{1}{n} \cdot \sum_{i=1}^n (y_i - f(x_i, \theta))^2$$

### Logistic regression

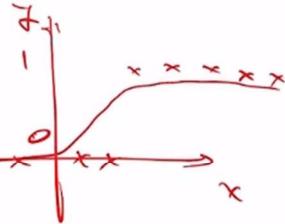
This can also be used to solve classification problems.

MLE for Logistic Regression

- Given  $\{(x_i, y_i)\}_{i=1}^n$ , where  $y_i \in \{0, 1\}$ , such that

$$p(y|x; \theta) = \frac{\exp(yf(x; \theta))}{1 + \exp(f(x; \theta))}.$$

Estimate  $\theta$ .



$$f(x, \theta)$$



In this case " $y_i = [0, 1]$ " and we know the probability function:

$$P(y=1|x, \theta) = \frac{e^{f(x, \theta)}}{1 + e^{f(x, \theta)}}$$

$$P(y=0|x, \theta) = \frac{1}{1 + e^{f(x, \theta)}}$$

We want to maximize the generic distribution function for all observed values 1 to "n":

$$l(\theta) = \sum_{i=1}^m \log(P(y_i|x_i, \theta)) = \sum_{i=1}^n \log\left(\frac{e^{y_i f(x_i, \theta)}}{1 + e^{f(x_i, \theta)}}\right) = \sum_{i=1}^n (y_i \cdot f(x_i, \theta) - \log(1 + e^{f(x_i, \theta)}))$$

This is a complicated formula to derive and typically we solve this optimization using numerical methods like the gradient descent.

$$P(y=1|x, \theta) = \frac{\exp(f(x, \theta))}{1 + \exp(f(x, \theta))}$$

$$P(y=0|x, \theta) = \frac{1}{1 + \exp(f(x, \theta))}.$$

$$\max \sum_{i=1}^n \log P(y_i|x_i, \theta)$$

$$= \sum_{i=1}^n (y_i f(x_i, \theta) - \log(1 + \exp(f(x_i, \theta))))$$

$$\stackrel{\Delta}{=} l(\theta)$$

Numerical Methods.

Summary

- MLE estimator is random variable (as a function of the random data):

$$\hat{\theta} = \hat{\theta}(x_1, \dots, x_n), \quad \{x_i\} \stackrel{iid}{\sim} p(\cdot | \theta_*).$$

- Evaluation metrics: Bias, variance, mean square error (MSE).

This MLE estimation is a random variable because it depends from some “xi” points obtained from a distribution function of the random variables.

We can try understanding the statistical behavior understanding the variance of this MLE analysis.

Theoretical properties

Understanding the mathematical likelihood as a mathematical cool.

- MLE estimator is random variable (as a function of the random data):

$$\hat{\theta} = \hat{\theta}(x_1, \dots, x_n), \quad \{x_i\} \stackrel{iid}{\sim} p(\cdot | \theta_*).$$

- Evaluation metrics: Bias, variance, mean square error (MSE).
- Unbiased estimators vs. consistent estimators.

The first thing to note is that the MLE is actually a random variable following sort of a distribution: when performing MLE estimation the data is drawn from a distribution and therefore the data is a random variable picked from all potential points in the space.

Because data is a random variable therefore the MLW is a random variable too.

If we look at this function is implicitly defined by the maximization process of MLE but it is still a function.

Because MLE is a random variable we want to understand its statistical properties: variance, mean square error.

We will define three key properties and show the relation between them. Then we will define the unbiased estimators and consistent estimators that give us reliable maximum likelihood estimations.

Bias

This is the difference of the expectation of the MLE based on the random data drawn minus the actual MLE ( $\theta^*$ ).

$$Bias(MLE) = Bias(\hat{\theta}) = E[\hat{\theta}(x_1, x_2, \dots, x_n)] - \theta^*$$

This can be written with the definition of the expectation as an integration:

$$Bias(MLE) = Bias(\hat{\theta}) = \int \hat{\theta}(x_1, x_2, \dots, x_n) \cdot \prod_{i=1}^n P(x_i | \theta^*) \cdot dx - \theta^*$$

Variance

We also have a property variance:

$$var(\hat{\theta}) = E[(\hat{\theta}(x_1, x_2, \dots, x_n) - E[\hat{\theta}(x_1, x_2, \dots, x_n)])^2]$$

Where value of the " $\hat{\theta}(x_1, x_2, \dots, x_n)$ " has already been calculated in the MLE bias.

The variance represents the fluctuation around the mean while the bias measures the difference between the mean and the true value.

Mean Square Error (MSE)

$$MSE(\hat{\theta}) = MSE(MLE) = E[(\hat{\theta}(x_1, x_2, \dots, x_n) - \theta^*)^2]$$

This is almost the same as the variance except that instead of subtracting the mean, we directly subtract the true value. This way, the Mean Square Error is actually a very direct estimation of the quality of the MLE estimation.

We want to minimize MSE or find a MLE that has minimum MSE.

Relationship between Bias, variance, MSE: Bias-variance decomposition

$$MSE(\hat{\theta}) = (Bias(\hat{\theta}))^2 + var(\hat{\theta})$$

$$\begin{aligned} \underline{Bias}(\hat{\theta}) &= \underbrace{E_{\theta^*}[\hat{\theta}(x_1, \dots, x_n)]}_{\hat{\theta}} - \underline{\theta^*} \\ &= \int \hat{\theta}(x_1, \dots, x_n) \prod_{i=1}^n P(x_i | \theta^*) dx - \underline{\theta^*} \\ \underline{Var}(\hat{\theta}) &= E_{\theta^*}[(\hat{\theta}(x_1, \dots, x_n) - \underline{E_{\theta^*}(\hat{\theta}(x_1, \dots, x_n))})^2] \\ \underline{MSE}(\hat{\theta}) &= E_{\theta^*}[(\hat{\theta}(x_1, \dots, x_n) - \underline{\theta^*})^2] \end{aligned}$$

Bias-Variance Decomposition

$$MSE(\hat{\theta}) = (Bias(\hat{\theta}))^2 + Var(\hat{\theta})$$

Both bias and variance contribute to the MSE.

Let us prove this statement:

$$\begin{aligned}
 MSE(\hat{\theta}) &= MSE(MLE) = E[(\hat{\theta} - \theta^*)^2] = E[(\hat{\theta} + (E(\hat{\theta}) - E(\hat{\theta})) - \theta^*)]^2 \\
 &= E[(\hat{\theta} - E(\hat{\theta}))^2 + (E(\hat{\theta}) - \theta^*)^2 + 2 \cdot (\hat{\theta} - E(\hat{\theta})) \cdot (E(\hat{\theta}) - \theta^*)] \\
 &= \text{var}(\hat{\theta}) + (\text{bias}(\hat{\theta}))^2 + 0
 \end{aligned}$$

The third term has a constant as:

$$E[\hat{\theta} - E(\hat{\theta})] = E(\hat{\theta}) - E(\hat{\theta}) = 0$$

This is a very important formula as this explains a tradeoff between bias and variance.

- Sometimes we have larger variance but Small bias.
- Sometimes we have Small bias but larger variance.

By trading these terms, we can find an optimal point.

Bias-Variance Decomposition

$$\boxed{MSE(\hat{\theta}) = (\text{Bias}(\hat{\theta}))^2 + \text{Var}(\hat{\theta})}$$

Proof:  $MSE(\hat{\theta}) = E[(\hat{\theta} - \theta^*)^2]$

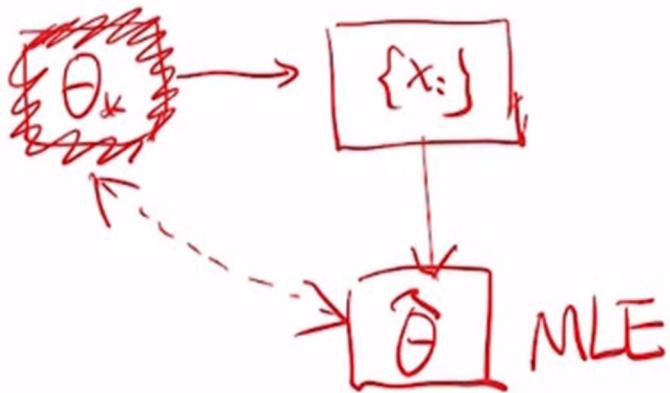
$$\begin{aligned}
 &= E[(\hat{\theta} - E(\hat{\theta}) + E(\hat{\theta}) - \theta^*)^2] \\
 &= E[(\hat{\theta} - E(\hat{\theta}))^2 + (E(\hat{\theta}) - \theta^*)^2 \\
 &\quad + 2(\hat{\theta} - E(\hat{\theta}))(E(\hat{\theta}) - \theta^*)] \\
 &= \text{Var}(\hat{\theta}) + (\text{bias}(\hat{\theta}))^2 = 0
 \end{aligned}$$

$$\begin{aligned}
 E[(\hat{\theta} - E(\hat{\theta}))(E(\hat{\theta}) - \theta^*)] &= E[\hat{\theta} - E(\hat{\theta})](E(\hat{\theta}) - \theta^*) \\
 &= (E(\hat{\theta}) - E(\hat{\theta}))(E(\hat{\theta}) - \theta^*).
 \end{aligned}$$

If we look at this framework once more time we see that as we estimate some unknown parameter (i.e. " $\theta^*$ ") generated from a bunch of observations  $\{x_1, x_2, \dots, x_n\}$ .

From the data  $\{x_i\}$  we estimate " $\theta$ " which is our MLE (the most likely value for the unknown parameter we are observing). As we compare " $\theta$ " and " $\theta^*$ " we hope on average they are very close to each other.

The magnitude of the difference is what the MSE gives us.



Turns out if an estimator has “0” bias then we call “ $\hat{\theta}$ ” unbiased estimator.

If means if we perform the estimation many times every time from a different training set all the estimation we get the parameter value “ $\hat{\theta}$ ” should be equal to the actual value “ $\theta^*$ ”.

#### Consistent estimators

This indicates that the MSE of our parameter goes to zero as we have increasingly more data available and “n” goes to infinite.

$$MSE(\hat{\theta}) \rightarrow 0, \text{ as } n \rightarrow \infty$$

In this case, we can say that “ $\hat{\theta}$ ” is consistent.

In other words, if we have an infinite number of points then there should not be error.

Bear in mind being unbiased does not imply being consistent, because even when the bias term is “0” we may have a large variance or a variance that does not go to zero as “n” increases.

On the other hand, even if we are consistent this does not imply unbiased: consistency just says that with a infinite “n” (number of data) the MSE goes to “0” but we still have some amount of bias even if it is increasingly Small (concept of **asymptotic unbiased**):

$$Bias(\hat{\theta}) \rightarrow 0, \text{ as } n \rightarrow \infty$$

Being consistent implies asymptotic unbiased

Being unbiased implies asymptotic unbiased.

Being unbiased does not imply being consistent.

If  $\text{Bias}(\hat{\theta}) = 0$ , we call  $\hat{\theta}$  "unbiased".

MSE( $\hat{\theta}$ )  $\rightarrow 0$ , as  $n \rightarrow \infty$ ,  $\hat{\theta}$  "consistent".

$\text{Bias}(\hat{\theta}) \rightarrow 0$ , as  $n \rightarrow \infty$ , "Asymptotic Unbiased"

2

$$\text{MSE}(\hat{\theta}) = (\text{Bias}(\hat{\theta}))^2 + \text{Var}(\hat{\theta})$$

Example MLE consistent, biased or unbiased

• **Example:** For  $\{x_i\}_{i=1}^n \sim \mathcal{N}(\mu, \sigma^2)$ , MLE is

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i, \quad \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu})^2.$$

The Gaussian distribution where we can see that "xi" (data) is drawn from a Gaussian.

In the last lecture, we showed that the MLE of mean and variance are simply the empirical value.

$$\text{Bias}(\hat{\mu}) = E[\hat{\mu}] - \mu = E\left[\frac{1}{n} \cdot \sum_{i=1}^n x_i\right] - \mu = \frac{1}{n} \cdot \sum_{i=1}^n E[x_i] - \mu = \frac{1}{n} \cdot \sum_{i=1}^n \mu - \mu = \mu - \mu = 0$$

It means the bias of the mean estimation is "0".

$$\begin{aligned} \text{var}(\hat{\mu}) &= E[(\hat{\mu} - \mu)^2] = E\left[\left(\frac{1}{n} \cdot \sum_{i=1}^n x_i - \mu\right)^2\right] \\ &= E\left[\frac{1}{n^2} \cdot \left(\sum_{i=1}^n (x_i - \mu)^2 + \sum_{i \neq j} (x_i - \mu) \cdot (x_j - \mu)\right)\right] \\ &= \frac{1}{n^2} \cdot \sum_{i=1}^n E[(x_i - \mu)^2] + \sum_{i \neq j} E[(x_i - \mu) \cdot (x_j - \mu)] = \frac{1}{n} \cdot \sigma^2 + 0 \end{aligned}$$

On the one hand, the first term is the definition of " $\sigma^2$ ", and on the other hand, the second term is the covariance, which should be equal to "0":

$$\sum_{i \neq j}^n E[(x_i - \mu) \cdot (x_j - \mu)] = \sum_{i \neq j}^n (E[(x_i - \mu)] \cdot E[(x_j - \mu)])$$

As "xi" and "xj" are independent results, then we can push the expectation inside the product and because "xi and xj" are independent draws, there is no dependency among them and the covariance should be "0".

• Example: For  $\{x_i\}_{i=1}^n \sim \mathcal{N}(\mu, \sigma^2)$ , MLE is

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i, \quad \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu})^2.$$

$$\text{Bias}(\hat{\mu}) = E[\hat{\mu}] - \mu = E\left[\frac{1}{n} \sum_{i=1}^n x_i\right] - \mu = \frac{1}{n} \sum_{i=1}^n E[x_i] - \mu$$

$$\begin{aligned} \text{var}(\hat{\mu}) &= E[(\hat{\mu} - \mu)^2] \\ &= E\left[\left(\frac{1}{n} \sum_{i=1}^n x_i - \mu\right)^2\right] = E\left[\frac{1}{n^2} \left( \sum_{i=1}^n (x_i - \mu)^2 + \sum_{i \neq j} (x_i - \mu)(x_j - \mu) \right)\right] \\ &= \frac{1}{n^2} \sum_{i=1}^n E[(x_i - \mu)^2] + \underbrace{\sum_{i \neq j} E[(x_i - \mu)(x_j - \mu)]}_{= E[(x_i - \mu)] E[(x_j - \mu)]} \\ &= \frac{1}{n} \sigma^2 + 0 \end{aligned}$$

Finally, the Mean Square Error (MSE) can be obtained:

$$MSE(\hat{\mu}) = (\text{Bias}(\hat{\mu}))^2 + \text{var}(\hat{\theta}) = 0 + \frac{\sigma^2}{n}$$

Is this estimator **consistent**? Yes, as  $n \rightarrow \infty$  then the MSE tends to "0".

In addition, this estimator is unbiased as  $\text{bias}(\hat{\mu}) = 0$ .

Then the mean is both unbiased and consistent.

What about the variance ("σ")?

It is biased what means:

$$E[\hat{\sigma}^2] \neq \sigma^2$$

The variance is a biased estimator but it is actually asymptotically unbiased meaning that:

$$\text{Bias}[\hat{\sigma}^2] \rightarrow 0, n \rightarrow \infty$$

$$\text{MSE}(\hat{\mu}) = (\text{Bias}(\hat{\mu}))^2 + \text{Var}(\hat{\mu}) = \frac{\sigma^2}{n}$$

If  $n \rightarrow \infty$ ,  $\text{MSE}(\hat{\mu}) \rightarrow 0 \Rightarrow \text{consistent}$

$$E[\hat{\sigma}^2] \neq \sigma^2, \quad \hat{s}^2 = \left( \frac{1}{n-1} \right) \sum_{i=1}^n (x_i - \hat{\mu})^2$$

"Biased"

$$E[\hat{s}^2] = \sigma^2$$

$\text{Bias}(\hat{\sigma}^2) \rightarrow 0 \quad n \rightarrow \infty \quad (\text{Asymp. Unbiased})$

$\text{var}(\hat{\sigma}^2) \rightarrow 0 \quad n \rightarrow \infty$

$\text{MSE}(\hat{\sigma}^2) \rightarrow 0 \quad n \rightarrow \infty \quad (\underline{\text{consistent}})$

### MLE Consistency

In most cases, MLE is usually consistent except for weird cases where regularity does not hold.

What is the mathematical principle that makes MLE always consistent?

### Why MLE?

- MLE is equivalent to minimizing Kullback-Leibler (KL) Divergence.

$$\text{KL}(q \parallel p) = \mathbb{E}_q[\log q(x) - \log p(x)].$$

- $\text{KL}(q \parallel p) \geq 0$  for any  $q$  and  $p$ .
- $\text{KL}(q \parallel p) = 0$  if and only if  $q = p$ .

**KL divergence is connected to the MLE:**

Let us say "p" and "q" are two different distributions.

Therefore, for "p" and "q" we can define their divergence: KL Divergence.

KL Divergence is the expectation for "q" for the difference between "p" and "q". We can apply the integration definition of "q" for the Expectation:

$$\text{KL}(q \parallel p) = E_q[\log q(x) - \log p(x)] = \int q(x) \cdot \log \left( \frac{q(x)}{p(x)} \right) \cdot dx$$

Generic Expectation formula, where “ $f(x)$ ” is a frequency or probabilistic function for the value “ $x$ ”:

$$E(x) = \int f(x) \cdot x \cdot dx$$

In our case “ $E(x)$ ” turns out to be “ $E_q(\log(q(x)/p(x)))$ ”  $\rightarrow$  “ $x = q(x)$ ” and “ $f(x) = \log(q(x)/p(x))$ ”.

Results:

$KL(q|p) \neq KL(p|q)$ , it is not symmetric.

This definition KL divergence is always larger or equal to zero.

$KL(q|p) = KL(p|q) = 0$  if “ $p$ ” and “ $q$ ” are the same.

Therefore, the KL divergence is actually a good measure of difference between “ $p$ ” and “ $q$ ” when we want to minimize the divergence so we can approximate one distribution to the other and trying to make it close to “0”.

### Jensen's inequality:

If “ $f(x)$ ” is convex function (function with bow shape) then we can show that:

- Expectation of “ $f(x)$ ” under distribution “ $q$ ”  $\rightarrow$  “ $E_q[f(x)]$ ”. Take “ $x$ ”, evaluate “ $f(x)$ ”, compute expectation “ $E_q(f(x))$ ” for several “ $x$ ”s drawn from distribution “ $q$ ”.
- Expectation of “ $x$ ” under distribution “ $q$ ” and function  $f(x)$   $\rightarrow$  “ $f(E_q(x))$ ”. Take “ $x$ ”s drawn from distribution “ $q$ ”, compute expectation “ $E_q(x)$ ”, evaluate “ $f(E_q(x))$ ”.

Then:

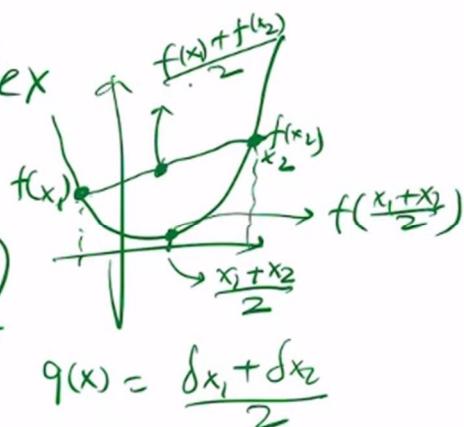
$$E_q[f(x)] \geq f(E_q[x])$$

Jensen's Inequality:

If  $f(x)$  is convex

then

$$\underline{E_q[f(x)]} \geq \underline{f(E_q[x])}$$



$$q(x) = \frac{dx_1 + dx_2}{2}$$

We can generalize this idea to arbitrary distribution “ $q$ ”.

The non-negativity of MLE can be proved with Jensen's inequality:

$$KL(q|p) = E_q \left[ \log \left( \frac{q}{p} \right) \right]$$

Log is not a convex but a concave function. To address the problem we adjust the formula:

$$KL(q|p) = E_q \left[ -\log \left( \frac{p}{q} \right) \right]$$

Reversing "p" and "q" and making it negative is equivalent. Now the negative log function is convex and we can use Jensen's inequality to show that:

$$KL(q|p) = E_q \left[ -\log \left( \frac{p}{q} \right) \right] \geq -\log \left( E_q \left[ \frac{p(x)}{q(x)} \right] \right)$$

Working with the right-hand side expression:

$$-\log \left( E_q \left[ \frac{p(x)}{q(x)} \right] \right) = -\log \left( \int q(x) \cdot \frac{p(x)}{q(x)} \cdot dx \right) = -\log(1) = 0$$

The result of integrating a single distribution (i.e. "f(x)" or "q(x)") function between " $-\infty$ " and " $+\infty$ " should be always "1".

Which implies that divergence "KL(q|p)" is always greater than "0" for 2 distributions "p" and "q" and only "0" when "p = q" as it can be seen in the picture below:

$$\text{if } -\log \left( E_q \left[ \frac{p(x)}{q(x)} \right] \right) = -\log(1) \rightarrow E_q \left[ \frac{p(x)}{q(x)} \right] = 1 \rightarrow \frac{p(x)}{q(x)} = \text{const}$$

$$\underline{\underline{KL(q|p)}} = E_q \left[ \underline{\underline{\log \left( \frac{q}{p} \right)}} \right] = E_q \left[ \underline{\underline{-\log \left( \frac{p(x)}{q(x)} \right)}} \right]$$

$$\Rightarrow \frac{p(x)}{q(x)} = \text{const}$$

$$\Rightarrow p(x) = \text{const} \cdot q(x)$$

$$\Rightarrow \sum_x p(x) = (\text{const}) \cdot \sum_x q(x)$$

$$\Rightarrow 1 = \text{const} \cdot 1 \Rightarrow \text{const} = 1$$

$$\Rightarrow p = q$$

$$\begin{aligned} & \approx -\log \left( E_q \left[ \frac{p(x)}{q(x)} \right] \right) \\ & = -\log \left( \sum_x q(x) \frac{p(x)}{q(x)} \right) \\ & = -\log \left( \sum_x p(x) \right) = -\log(1) \\ & = 0 \end{aligned}$$

How do we connect this now with MLE?

*MLE  $\leftrightarrow$  KL Divergence*

$$KL(q|p) = E_q [\log q(x) - \log p(x)]$$

Assume "q" is the underlying data distribution. It is given through the noisy observations. It is fixed.

Assume "p" is the model we try to estimate. The model we want to optimize.

We want to estimate the model and minimize the divergence:

$$\min_{\theta} KL(q||p_{\theta}) = \min_{\theta} E_q[\log q(x) - \log p_{\theta}(x)]$$

This problem is equivalent to minimizing the second term that is the model we want to improve as "q(x)" cannot be changed or improved:

$$\min_{\theta} E_q[\log q(x) - \log p_{\theta}(x)] = \min_{\theta} E_q[-\log p_{\theta}(x)] = \max_{\theta} E_q[\log p_{\theta}(x)]$$

Working with observed "x"s  $\{x_1, x_2, \dots, x_n\}$  values coming from distribution "q":

$$\max_{\theta} E_q[\log p_{\theta}(x)] \approx \max_{\theta} \frac{1}{n} \cdot \sum_{i=1}^n \log p_{\theta}(x_i)$$

This quantity can be approximately the empirical average as they are similar. And this quantity is the log-likelihood.

Maximizing the likelihood function is actually equivalent to minimizing the KL divergence between the data distribution "q" and the model "p<sub>θ</sub>".

Assume  $q$  is data distribution (Given)

$p_{\theta}$  is "model"  $\theta \in \Theta$

$$\min_{\theta} (KL(\underline{q} || \underline{p_{\theta}})) \Leftrightarrow \min_{\theta} E_{\underline{q}} [\log \underline{q(x)} - \log \underline{p_{\theta}(x)}]$$

$$\Leftrightarrow \min_{\theta} -E_{\underline{q}} [\log \underline{p_{\theta}(x)}]$$

$$\Leftrightarrow \boxed{\max_{\theta} E_{\underline{q}} [\log \underline{p_{\theta}(x)}]}$$

$$\approx \max_{\theta} \underline{\frac{1}{n} \sum_{i=1}^n \log p_{\theta}(x_i)} \quad \{x_i\} \sim q$$

Avg log-likelihood.

## Bayesian inference

### Introduction

This is a very powerful technique for parameter estimation.

In the last lectura we talked about maximum likelihood estimation, Bayesian inference is another technique and it can also quantify uncertainty.

---

### Bayesian Inference: Main Idea

- Maximum likelihood estimation (MLE):

$$\hat{\theta} = \arg \max_{\theta} p(D | \theta).$$

Parameter  $\theta$  is unknown but deterministic (frequentist view).

*data*  
*Unknown variable*

- Bayesian inference:

- $\theta$  is viewed as a random variable (even when it is actually deterministic).
- Use Bayes' rule to calculate the posterior distribution:

$$p(\theta | D) = \frac{p(D | \theta)p(\theta)}{p(D)}.$$

In the previous lecture the idea was to maximize the likelihood function for some observed data "D"  $\{x_1, x_2, \dots, x_n\}$  and some parameter " $\theta$ " which we try to make the most likely value for the actual parameter " $\theta^*$ ".

We assume that there is a " $\theta^*$ " value that explains the observed values but it is unknown.

Hopefully we can use the information from "D" to recover the underlying value of " $\theta$ ".

The approach of estimating something unknown but with a fixed value from randomized data sets is something called the "Frequentist" or objective view.

Bayesian inference is significantly different. Now our parameter is not something we can determine but a random variable.

In practice " $\theta^*$ " could be something we can determine (deterministic). From a bayesian perspective we will treat it as a random variable and we only have some limited, partial information about " $\theta^*$ ". This approach is "subjective" and this idea is behind Bayesian inference.

In Bayesian inference we are going to explicitly calculate the potential distribution of " $\theta^*$ " using the given observation "D" via Bayes' Rule.

### Bayes' rule/theoreme

The probability of parameter " $\theta$ " given the observed data "D" is the "Posterior distribution": " $p(\theta | D)$ ":

$$p(\theta | D) = \frac{p(D | \theta) \cdot p(\theta)}{p(D)}$$

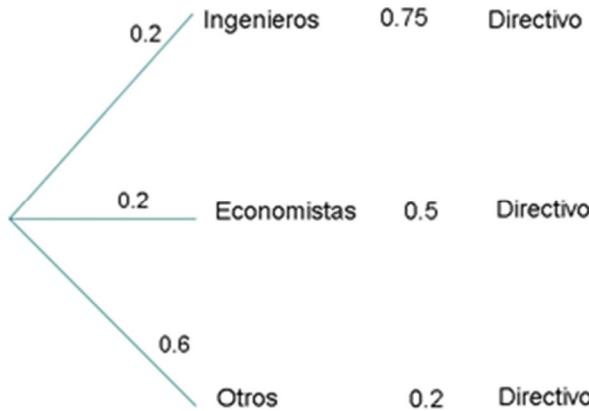
Bear in mind now " $\theta$ " and "D" are both random variables drawn from random distribution functions.

Prior distribution: " $p(\theta)$ "

Likelihood: “ $p(D|\theta)$ ”

### Ejemplo 1

El 20% de los empleados de una empresa son ingenieros y otro 20% son economistas. El 75% de los ingenieros ocupan un puesto directivo y el 50% de los economistas también, mientras que los no ingenieros y los no economistas solamente el 20% ocupa un puesto directivo.



A = Probabilidad de ser ingeniero

B = Probabilidad de ser directivo

¿Cuál es la probabilidad de que un empleado directivo elegido al azar sea ingeniero? O alternativamente: en el supuesto de que seamos directivos, ¿cuál es la probabilidad de ser ingeniero?

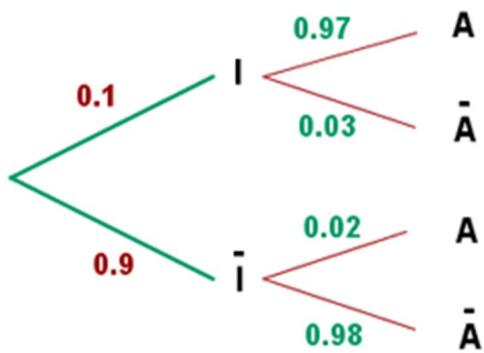
$$p(A|B) = \frac{p(B|A) \cdot p(A)}{p(B)}$$

En nuestro caso:

$$\begin{aligned} p(\text{ingeniero } | \text{directivo}) &= \frac{p(\text{directivo } | \text{ingeniero}) \cdot p(\text{ingeniero})}{p(\text{directivo})} \\ &= \frac{0,75 \cdot 0,2}{(0,75 \cdot 0,2 + 0,5 \cdot 0,2 + 0,2 \cdot 0,6)} = 0,405 \end{aligned}$$

### Ejemplo 2

La probabilidad de que haya un accidente en una fábrica que dispone de alarma es 0,1. La probabilidad de que suene esta sí se ha producido algún incidente es de 0,97 y la probabilidad de que suene si no ha sucedido ningún incidente es 0,02.



A = Probabilidad de incidente

B = Probabilidad de que suene la alarma

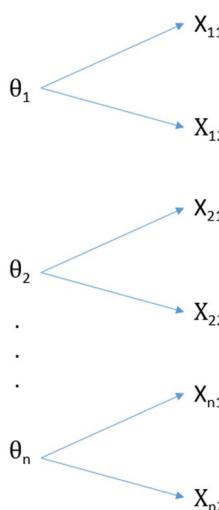
En el supuesto de que haya funcionado la alarma, ¿cuál es la probabilidad de que no haya habido ningún incidente?

$$p(A | B) = \frac{p(B | A) \cdot p(A)}{p(B)}$$

En nuestro caso:

$$p(\bar{A} | B) = \frac{p(A|\bar{I}) \cdot p(\bar{I})}{p(A)} = \frac{0,02 \cdot 0,9}{(0,02 \cdot 0,9 + 0,97 \cdot 0,1)} = 0,157$$

### Ejemplo 3: our case



$$p(\theta | D) = \frac{p(D | \theta) \cdot p(\theta)}{p(D)}$$

Assuming we have observations "D= {x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub>}" what is the probability of " $\theta^*$ " being " $\theta$ "?  
Posterior:  $p(\theta|D)$ ?

- We need to get the probability of "D" being the observed values for certain " $\theta$ ". Likelihood: " $p(D|\theta)$ ".
- We also need to know the probability for " $\theta^*$ " being our " $\theta$ ". Prior distribution: " $p(\theta)$ ".
- Finally we need the probability value for our observed collection "D = {x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub>} regardless of " $\theta$ ".

### Recall:

"P(D)" this is a constant: the marginal distribution of the data and can be obtained:

$$p(D) = \int p(D|\theta) \cdot p(\theta) \cdot d\theta$$

For our purpose we don't need to look into this "p(D)" because this is just a normalization constant to make sure the posterior distribution is normalized. Actually in our examples we were all the time applying already implicitly " $p(D|\theta) \cdot p(\theta)$ " for all " $\theta$ ".

We can re-write this posterior or rule to:

$$p(\theta|D) = \frac{p(D|\theta) \cdot p(\theta)}{\int p(D|\theta) \cdot p(\theta) \cdot d\theta}$$

In practice we only care about " $\theta$ " and Data "D" is fixed as a constant we can rewrite it:

$$p(\theta|D) \propto \frac{p(D|\theta) \cdot p(\theta)}{k}$$

#### • Bayesian inference:

- $\theta$  is viewed as a random variable (even when it is actually deterministic).
- Use Bayes' rule to calculate the posterior distribution:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$$

$p(\theta|D)$  = Posterior

$p(D|\theta)$  = likelihood

$p(\theta)$  = prior

$$\underline{p(D)} = \int p(D|\theta)p(\theta)d\theta$$

$$\begin{aligned} p(\theta|D) &= \frac{p(D|\theta)p(\theta)}{\int p(D|\theta)p(\theta)d\theta} \\ &\propto p(D|\theta)p(\theta) \end{aligned}$$



### Bayes' rule demonstration

Both " $\theta$ " and "D" are random variables and they have joint distributions.

We can use the chain rule to prove this rule:

$$p(\theta|D) = p(\theta) \cdot p(D|\theta) \text{ (chain rule)} = p(D) \cdot p(\theta|D)$$

The first part (" $p(D|\theta)$ ") is equivalent to say that conditioned on " $\theta$ " we generate data "D".

The second part (" $p(\theta | D)$ ") is equivalent to the reverse.

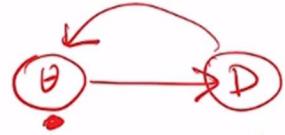
This allows us to require the previous formula:

$$p(\theta) \cdot p(D|\theta) = p(D) \cdot p(\theta|D)$$

And dividing each part by " $P(D)$ " we get to our expression:

$$p(\theta) \cdot p(D|\theta) = p(D) \cdot p(\theta|D) \rightarrow \frac{p(\theta) \cdot p(D|\theta)}{p(D)} = \frac{p(D) \cdot p(\theta|D)}{p(D)}$$

Bayes' Rule

$$\begin{aligned} P(\theta, D) &= \cancel{P(\theta)} \cancel{P(D|\theta)} \quad (\text{Chain Rule}) \\ &= P(D) \cancel{P(\theta|D)} \\ \cancel{\frac{P(\theta)P(D|\theta)}{P(D)}} &= \frac{P(D) \cancel{P(\theta|D)}}{P(D)} \\ &= \underline{\underline{P(\theta|D)}} \end{aligned}$$


Example Bayes' rule

### Example: Did the Sun Just Explode?

- We have a device that detects if the sun explodes with high accuracy:

$$\begin{aligned} p(x = \theta | \theta) &= 1 - \alpha, \\ p(x = 1 - \theta | \theta) &= \alpha. \end{aligned}$$

$\theta \in \{0, 1\}$ : if the sun explode;  $x \in \{0, 1\}$ : if the alarm fires.

- If the alarm fires ( $x = 1$ ). Should we believe sun has exploded or not?

With MLE we trust the alarm because we trust that the obtention of " $\theta$ " is deterministic and therefore the sun must have exploded. With MLE we determined that the " $\theta$ " that maximizes probability (most likely scenario) is when the sun has exploded.

## Example: Did the Sun Just Explode?

- We have a device that detects if the sun explodes with high accuracy:

$$P(x = \theta | \theta) = 1 - \alpha, \\ P(x = 1 - \theta | \theta) = \alpha.$$

 $\alpha$ : error

known, fixed

 $\theta \in \{0, 1\}$ : if the sun exploded;  $x \in \{0, 1\}$ : if the alarm fires.• If the alarm fires ( $x = 1$ ). Should we believe sun has exploded or not?  $\alpha = 0.0001$ 

① MLE:  $\hat{\theta} = \underset{\theta \in \{0, 1\}}{\operatorname{argmax}} P(x=1 | \theta) = \begin{cases} \alpha & \theta=0 \\ 1-\alpha & \theta=1 \end{cases}$

 $= 1$

## ② Bayesian Inference:

Step 1: prior  $P(\theta) = \begin{cases} 10^{-100k} \approx \beta, & \theta=1 \\ \approx 1-\beta & \theta=0 \end{cases}$

Step 2: posterior:

$$P(\theta | x=1) = \frac{P(x=1 | \theta) P(\theta)}{P(x=1)} \propto P(x=1 | \theta) P(\theta)$$
 $= \begin{cases} (1-\alpha)\beta & \theta=1 \\ \alpha(1-\beta) & \theta=0 \end{cases}$

 $\theta \stackrel{?}{=} 0 / 1 ?$ 

$$\underbrace{P(\theta=1 | x=1)}_{(1-\alpha)\beta} \leq \underbrace{P(\theta=0 | x=1)}_{\alpha(1-\beta)}$$

$$\Rightarrow \text{predict } \theta=1 \text{ if } \underbrace{P(\theta=1 | x=1)}_{(1-\alpha)\beta} < \underbrace{P(\theta=0 | x=1)}_{\alpha(1-\beta)}$$

predict  $\theta=0$ 

$$\Rightarrow \frac{\beta}{1-\beta} < \frac{\alpha}{1-\alpha} \quad (\times)$$
 $\beta \approx 0 \quad \alpha = 0.0001$

Summarizing Bayes' rule**Bayesian Inference**

$$p(\theta | D) = \frac{p(D | \theta)p(\theta)}{p(D)} \propto p(D | \theta)p(\theta).$$

Parameter  
Data.  
 $p(\theta)$ : Prior

We have seen that the Bayesian inference relies on a very simple Bayes' rule where we want to estimate the posterior distribution of " $\theta$ " (i.e. " $p(\theta | D)$ ") given the observed data "D".

As seen before this can be obtained as:

$$p(\theta | D) = \frac{p(D | \theta) \cdot p(\theta)}{p(D)}$$

Where we need to obtain:

- " $p(D | \theta)$ ": **likelihood**, probability of the data "D" (observations) given " $\theta$ ".
- " $p(\theta)$ ": **prior** information about " $\theta$ ". It is the distribution that we have about " $\theta$ " when we observe nothing.

We also saw that this posterior distribution " $p(\theta | D)$ " will be proportional to the likelihood times prior as the  $p(D)$  is a constant value:

$$p(\theta | D) \propto p(D | \theta) \cdot p(\theta)$$

Example: predicting the commute time**Example: Predicting the Commute Time**

- You move to a new apartment.
- Your friend told you the commute time is  $30 \pm 10$  min.
- You also drove yourself a few times, and found the times are 25, 45, 30, 50, ...
- How should you predict the commute time?

We want to get the actual commute time.

We have some observations of the commute time when carried out by “me”.

We also have some prior information from a “friend”.

How do we combine this information from a friend with our experiments?

$\theta$ : time. Unknown parameter we want to obtain.

**Prior** -  $p(\theta)$ : will be a Gaussian distribution with average 30 minutes and variance 10 minutes.

$$p(\theta) \sim N(\eta, \sigma^2) = N(30, 10^2)$$

**Observation (D)**:  $\{x_1, x_2, \dots, x_n\} = \{25, 45, 30, 50\}$

**Assumption** - every time we carry out the test this is sort of a noisy observation:

$$x_i = \theta + \sigma_1 \cdot \psi_i, \text{ where } \psi_i \sim N(0, 1)$$

So each observation is the “ $\theta$ ” plus some noise given by a “ $\sigma_1$ ” which represents a variance that can fluctuate in a unit Gaussian.

**Final assumption**:  $\sigma_1 = 5$  (this is a data given in our problem).

Based on this assumption we can therefore establish another normal distribution:

$$p(x_i | \theta) \sim N(\theta, \sigma_1^2)$$

Including now our current data:

- Prior
- Observation
- “ $p(x|\theta)$ ”

Let's obtain the posterior:

$$p(\theta | D) \propto p(D | \theta) \cdot p(\theta) = \left( \prod_{i=1}^n p(x_i | \theta) \right) \cdot p(\theta)$$

Note that “ $p(D | \theta)$ ” is equal to the product of probability of individual data points.

The probability of each individual points is as follows:

$$p(x_i | \theta) = \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma_1} \cdot e^{-\frac{(x_i - \theta)^2}{2 \cdot \sigma_1^2}} \propto e^{-\frac{(x_i - \theta)^2}{2 \cdot \sigma_1^2}}$$

As the first part is constant we just take the proportional part.

Rewriting now the posterior formula:

$$p(D | \theta) \cdot p(\theta) \propto \left( \prod_{i=1}^n e^{-\frac{(x_i - \theta)^2}{2 \cdot \sigma_1^2}} \right) \cdot p(\theta)$$

" $p(\theta)$ " can also be expressed as a normal distribution with mean and variance proportional (which allows us to again remove the constant part):

$$p(D | \theta) \cdot p(\theta) \propto \left( \prod_{i=1}^n e^{-\frac{(x_i - \theta)^2}{2 \cdot \sigma_1^2}} \right) \cdot p(\theta) \propto \left( \prod_{i=1}^n e^{-\frac{(x_i - \theta)^2}{2 \cdot \sigma_1^2}} \right) \cdot (e^{-\frac{(\theta - \eta)^2}{2 \cdot \sigma^2}})$$

Rewriting this:

$$\left( \prod_{i=1}^n e^{-\frac{(x_i - \theta)^2}{2 \cdot \sigma_1^2}} \right) \cdot \left( e^{-\frac{(\theta - \eta)^2}{2 \cdot \sigma^2}} \right) = e^{\left( \sum_{i=1}^n \left( e^{-\frac{(x_i - \theta)^2}{2 \cdot \sigma_1^2}} \right) - e^{-\frac{(\theta - \eta)^2}{2 \cdot \sigma^2}} \right)}$$

$$\begin{aligned} \overline{P(\theta | D)} &= \frac{\overline{P(D | \theta) P(\theta)}}{\overline{P(D)}} \propto \overline{P(D | \theta) P(\theta)} \\ &= \underbrace{\left[ \prod_{i=1}^n \overline{P(x_i | \theta)} \right]}_{\overline{P(D | \theta)}} \overline{P(\theta)} \\ &\propto \left[ \prod_{i=1}^n \exp\left(-\frac{(x_i - \theta)^2}{2\sigma_1^2}\right) \right] \exp\left(-\frac{(\theta - \mu_0)^2}{2\sigma_0^2}\right) \\ &\propto \exp\left(-\sum_{i=1}^n \frac{(\theta - x_i)^2}{2\sigma_1^2} - \frac{(\theta - \mu_0)^2}{2\sigma_0^2}\right) \end{aligned}$$

We can further simplify this as a quadratic function of " $\theta$ :

$$p(D | \theta) \cdot p(\theta) \propto e^{-\frac{1}{2}(A \cdot \theta^2 - 2 \cdot B \cdot \theta + C)}$$

Where:

$$A = \left( \sum_{i=1}^n \frac{1}{\sigma_1^2} \right) + \frac{1}{\sigma^2} = \frac{n}{\sigma_1^2} + \frac{1}{\sigma^2}$$

$$B = \sum_{i=1}^n \frac{x_i}{\sigma_1^2} + \frac{\eta}{\sigma^2}$$

$C = \text{constant, we don't care}$

Further simplification can be achieved in the main formula:

$$p(D | \theta) \cdot p(\theta) \propto e^{-\frac{1}{2} A \left( \theta - \frac{B}{A} \right)^2}$$

This is equivalent to an equivalent normal Gaussian distribution taking a look at the formula obtained:

$$p(\theta | D) \propto p(D | \theta) \cdot p(\theta) \propto N\left(\frac{B}{A}, \frac{1}{A}\right) = N(\eta_p, \sigma_p^2)$$

Let's calculate the value of this normal distribution:

$$\eta_p = \frac{B}{A} = \frac{\sum_{i=1}^n \frac{x_i}{\sigma_i^2} + \frac{\mu_0}{\sigma_0^2}}{\frac{n}{\sigma_0^2} + \frac{1}{\sigma_0^2}}$$

$$\sigma_p^2 = \frac{1}{A} = \left( \frac{n}{\sigma_0^2} + \frac{1}{\sigma_0^2} \right)^{-1}$$

$$\begin{aligned}
 & \propto \exp\left(-\sum_{i=1}^n \frac{(\theta - x_i)^2}{2\sigma_i^2} - \frac{(\theta - \mu_0)^2}{2\sigma_0^2}\right) \\
 & = \exp\left(-\frac{1}{2}(A\theta^2 - 2B\theta + C)\right) \\
 & = \exp\left(-\frac{1}{2}A(\theta - \frac{B}{A})^2 + \text{const}\right) \\
 & \sim N\left(\frac{B}{A}, \frac{1}{A}\right)
 \end{aligned}
 \quad \left| \begin{array}{l}
 A = \sum_{i=1}^n \frac{1}{\sigma_i^2} + \frac{1}{\sigma_0^2} \\
 = \frac{n}{\sigma_0^2} + \frac{1}{\sigma_0^2} \\
 B = \sum_{i=1}^n \frac{x_i}{\sigma_i^2} + \frac{\mu_0}{\sigma_0^2} \\
 C = ??
 \end{array} \right.$$

$$\mu_p = \frac{B}{A} = \frac{\sum_{i=1}^n \frac{x_i}{\sigma_i^2} + \frac{\mu_0}{\sigma_0^2}}{\frac{n}{\sigma_0^2} + \frac{1}{\sigma_0^2}}$$

$$\sigma_p^2 = \frac{1}{A} = \left( \frac{n}{\sigma_0^2} + \frac{1}{\sigma_0^2} \right)^{-1}$$

Let's check some cases:

- If "n=0" (no data)  $\rightarrow \eta_p = \eta$ , if we don't have any data then the posterior mean will be the prior mean (the one our "friend" gave us).  
The same happens for the variance:  $\sigma_p = \sigma$
- If "n>0" (we have data): the posterior mean (" $\eta_p$ ") will be the weighted sum of the data and the prior.
- If " $n \rightarrow \infty$ " we have lots of data (infinite data), in this case the importance of the "prior" values decrease.

In this case when "n" is infinite the prior is not so important compared with the biggest values (undetermination):

$$\eta_p = \frac{\sum_{i=1}^n \frac{x_i}{\sigma_i^2} + 0}{\frac{n}{\sigma_1^2} + 0} = \frac{1}{n} \cdot \sum_{i=1}^m x_i$$

And we obtain the empirical values for the mean for the provided data.

$$\sigma_p^2 = \frac{1}{A} = \left( \frac{n}{\sigma_1^2} + 0 \right)^{-1} = \frac{\sigma_1^2}{n}$$

As we increase "n" the posterior variance will fluctuate less and less and tending to "0". Therefore the observed mean will be the actual value for " $\theta$ " we are looking for (the actual commute time).

Bayesian inference provides a good framework for combining prior and posterior information.

$\mu_p = \frac{B}{A} = \frac{\sum_{i=1}^n \frac{x_i}{\sigma_i^2} + \frac{\mu_0}{\sigma_0^2}}{\frac{n}{\sigma_1^2} + \frac{1}{\sigma_0^2}}$

$\sigma_p^2 = \frac{1}{A} = \left( \frac{n}{\sigma_1^2} + \frac{1}{\sigma_0^2} \right)^{-1}$

If  $n=0$  (No data),  $\mu_p = \frac{\mu_0}{\sigma_0^2} / \frac{1}{\sigma_0^2} = \mu_0$ .

If  $n>0$  (have data)

If  $n \rightarrow \infty$  (Infinite data)

$\sigma_p^2 = \sigma_0^2$

$\mu_p \approx \frac{\sum_{i=1}^n \frac{x_i}{\sigma_i^2} + 0}{\frac{n}{\sigma_1^2} + 0} \approx \frac{\sum_{i=1}^n x_i}{n}$

$\sigma_p^2 \approx \left( \frac{n}{\sigma_1^2} \right)^{-1} = \frac{\sigma_1^2}{n}$

Example: bayesian linear regression

## Bayesian Linear Regression

- Given data points  $\{x_i, y_i\}_{i=1}^n$ , want to find  $\theta$ , such that  $y \approx x^\top \theta$ .

Normally we solve linear regression with least squares estimation.

Using bayesian linear regression has the advantage it provides a consistent and coherent way in order to quantify the uncertainty in the data and the prompt estimation.

Problem:

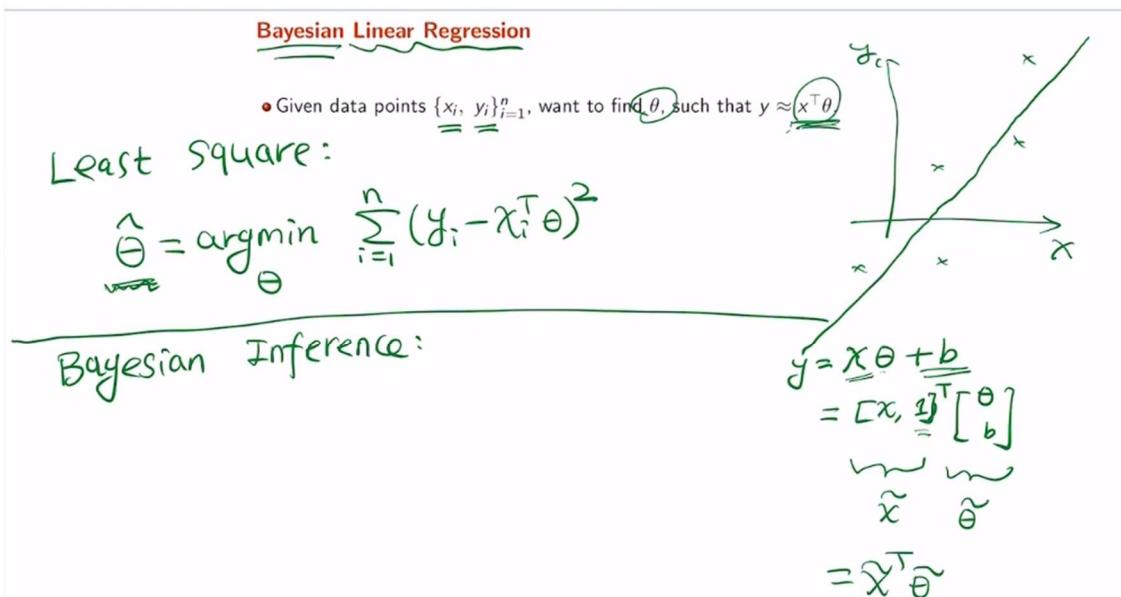
- **Data:** pair of data points:  $\{\{x_1, y_1\}, \{x_2, y_2\}, \dots, \{x_n, y_n\}\}$ . Observed values.
- **Goal:** Obtain a linear function depending on “ $\theta$ ” that allows us to predict “ $y_i$ ” values provided “ $x_i$ ” values.

There is also a bias term “ $b$ ” that we can include in this form:

$$y = x \cdot \theta + b = [x \quad 1] \cdot [\begin{matrix} \theta \\ b \end{matrix}] = [\tilde{x}] \cdot [\tilde{\theta}]$$

How to estimate “ $\theta$ ”?

The typical approach is the least square method that estimates the optimal value minimizing the square difference. This is a deterministic estimation of “ $\theta$ ”.



Sometimes we also want to get an uncertainty estimation about how accurate our estimation about “ $\theta$ ” actually is by means of Bayesian inference.

With Bayesian inference we assume “ $\theta$ ” is a random variable and this implies to **assume a prior** value: a Gaussian prior:

$$p(\theta) \sim N(\eta, \sigma^2)$$

Step without information enough we can assume the following:

$$\eta = 0, \sigma^2 = \text{large number}$$

Step with some sort of prior information about these values: assume what will be the expected mean “ $\eta$ ” and variance around this mean “ $\sigma$ ”.

Bayesian Inference:

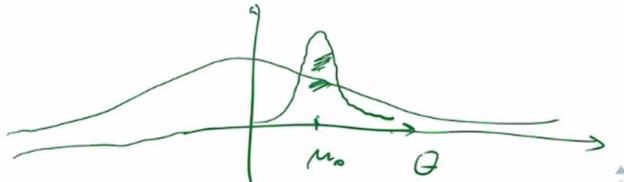
Treat  $\theta$  as a Random Variable.

$$\begin{aligned} y &= \underline{x}\theta + \underline{b} \\ &= [\underline{x}, \underline{1}]^T \begin{bmatrix} \theta \\ b \end{bmatrix} \end{aligned}$$

Assume Prior:

$$P(\theta) \sim \mathcal{N}(\mu_0, \sigma_0^2)$$

$\mu_0 = 0$ .  $\sigma_0^2 \rightarrow$  Large Number.



Assume **likelihood** is the next step, and it will again follow a Gaussian distribution with noisy observation of values “ $\{x_i, y_i\}$ ”:

$y_i = x_i \cdot \theta + \sigma_1 \cdot \psi_i$ , where  $\psi_i \sim N(0, 1)$

“ $\sigma^2$ ”: is a variance that we can decide ourselves or learn from data.

“ $\Psi$ ”: is the standard Gaussian noise.

Applying the chain rule we can decompose the likelihood:

$$p(\{\gamma_i, x_i\} | \theta) = p(\gamma_i | x_i, \theta) \cdot p(x_i) \text{ (chain rule)}$$

Where " $p(x_i)$ " is a likelihood and can be expected to be constant with respect to " $\theta$ " as it should be completely random and independent from " $\theta$ " the way we draw points.

Posterior distribution will be as follow:

$$p(\theta | \{x_i, y_i\}_{i=1}^n) = p(\theta | D) \propto p(D, \theta) \cdot p(\theta) = \left( \prod_{i=1}^n p(\{x_i, y_i\} | \theta) \right) \cdot p(\theta)$$

Again using the chain rule we can expand this (assuming the data points are generated independently and that we can expand the joint probability of  $\{x_i, y_i\}$  with the chain rule):

$$p(D, \theta) \cdot p(\theta) = \left( \prod_{i=1}^n p(\{x_i, y_i\} | \theta) \right) \cdot p(\theta) = \left( \prod_{i=1}^n p(y_i | x_i, \theta) \cdot p(x_i) \right) \cdot p(\theta)$$

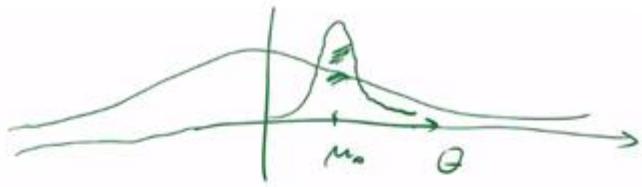
Because we said that " $p(x_i)$ " is a constant and independent with respect to " $\theta$ ":

$$p(D, \theta) \cdot p(\theta) \propto \left( \prod_{i=1}^n p(y_i | x_i, \theta) \right) \cdot p(\theta)$$

$$\underline{y}_i = \underline{x}_i^T \theta + \sigma_i \xi_i$$

$\sigma$ : Variance

$$\xi_i \sim N(0, 1)$$



$$P(\{y_i, x_i\} | \theta) = \underbrace{P(y_i | x_i, \theta)}_{\text{Conditional Likelihood}} \underbrace{P(x_i)}_{\text{Prior}}$$

$$\begin{aligned} P(\theta | D) &\propto \underbrace{P(D | \theta)}_{\text{Posterior}} P(\theta) \\ &= \left[ \prod_{i=1}^n P(y_i, x_i | \theta) \right] P(\theta) \\ &= \left[ \prod_{i=1}^n P(y_i | x_i, \theta) P(x_i) \right] P(\theta) \\ &\propto \left[ \prod_{i=1}^n P(y_i | x_i, \theta) \right] P(\theta) \end{aligned}$$

This regression problem or supervised learning problem, the posterior is proportional the product of the conditional likelihood times the prior information.

Plugging in what we know about the Gaussian distributions:

$$p(\theta | D) \propto \left( \prod_{i=1}^n e^{-\frac{(y_i - x_i \cdot \theta)^2}{2 \cdot \sigma_1^2}} \right) \cdot e^{-\frac{(\theta - \eta)^2}{2 \cdot \sigma^2}} = e^{\left( -\left( \sum_{i=1}^n \frac{(y_i - x_i \cdot \theta)^2}{2 \cdot \sigma_1^2} \right) - \frac{(\theta - \eta)^2}{2 \cdot \sigma^2} \right)}$$

$$\begin{aligned}
 p(\theta | D) &\propto \underline{P(D | \theta)} P(\theta) \\
 &= \left[ \prod_{i=1}^n P(y_i | x_i, \theta) \right] P(\theta) \\
 &= \left[ \prod_{i=1}^n P(y_i | x_i, \theta) \underline{P(x_i)} \right] P(\theta) \\
 &\propto \left[ \prod_{i=1}^n \underbrace{P(y_i | x_i, \theta)}_{=} \right] \underline{P(\theta)} \\
 &\propto \left[ \prod_{i=1}^n \exp\left(-\frac{(y_i - x_i^T \theta)^2}{2\sigma_i^2}\right) \right] \exp\left(-\frac{(\theta - \mu_0)^2}{2\sigma_0^2}\right) \\
 &= \exp\left(-\sum_{i=1}^n \frac{(y_i - x_i^T \theta)^2}{2\sigma_i^2} - \frac{(\theta - \mu_0)^2}{2\sigma_0^2}\right)
 \end{aligned}$$

Similar to the example earlier is a quadratic form. It means the posterior is going to be a Gaussian.

We can rewrite this following a quartic function:

$$p(\theta | D) \propto e^{\left(-\frac{1}{2}(\theta^T \cdot A \cdot \theta - 2 \cdot B^T \cdot \theta + Constant)\right)}$$

Where "p(θ|D)" will take a Gaussian distribution of the form:

$$p(\theta | D) \sim N(A^{-1} \cdot B, A^{-1})$$

$$\begin{aligned}
& \propto \left[ \prod_{i=1}^n P(y_i | x_i; \theta) \right] P(\theta) \\
& \propto \left[ \prod_{i=1}^n \exp\left(-\frac{(y_i - x_i^\top \theta)^2}{2\sigma_i^2}\right) \right] \exp\left(-\frac{(\theta - \mu_0)^2}{2\sigma_0^2}\right) \\
& = \exp\left(-\underbrace{\sum_{i=1}^n \frac{(y_i - x_i^\top \theta)^2}{2\sigma_i^2}}_{\text{constant}} - \underbrace{\frac{(\theta - \mu_0)^2}{2\sigma_0^2}}_{\text{constant}}\right) \\
& = \exp\left(-\frac{1}{2}(\theta^\top A \theta - 2B^\top \theta + \text{const})\right) \\
& \sim N(A^{-1}B, A^{-1})
\end{aligned}$$

Expanding and deriving we find out that:

$$A = \left( \sum_{i=1}^n \frac{x_i \cdot x_i^T}{\sigma_i^2} \right) + \frac{I}{\sigma^2}, \text{ where } A \text{ is a matrix and } I \text{ is the identity matrix.}$$

If “ $\theta$ ” is a  $d \times 1$  vector then “ $A$ ” is a  $d \times d$  vector.

$$B = \left( \sum_{i=1}^n \frac{y_i \cdot x_i}{\sigma_i^2} \right) + \frac{\eta}{\sigma^2}$$

Finally we find out that:

$$p(\theta|D) \sim N(\eta_p, \sigma_p^2) = N(A^{-1} \cdot B, A^{-1})$$

Where:

$$\eta_p = A^{-1} \cdot B$$

$$\sigma_p^2 = A^{-1}$$

$$\begin{aligned}
 & \propto \left[ \prod_{i=1}^n \exp\left(-\frac{(y_i - x_i^\top \theta)^2}{2\sigma_i^2}\right) \right] \exp\left(-\frac{(\theta - \mu_0)^2}{2\sigma_0^2}\right) \\
 & = \exp\left(-\sum_{i=1}^n \frac{(y_i - x_i^\top \theta)^2}{2\sigma_i^2} - \frac{(\theta - \mu_0)^2}{2\sigma_0^2}\right) \\
 \theta & : \begin{bmatrix} ] \\ d \times 1 \end{bmatrix} \\
 A & = \begin{bmatrix} ] \\ d \times d \end{bmatrix} \quad \mu_p = \tilde{A}^{-1} B \\
 & \sim \mathcal{N}(\tilde{A}^{-1} B, \tilde{A}^{-1}) \quad \begin{cases} A = \sum_{i=1}^n \frac{x_i x_i^\top}{\sigma_i^2} + \frac{I}{\sigma_0^2} \\ B = \sum_{i=1}^n \frac{y_i x_i}{\sigma_i^2} + \frac{\mu_0}{\sigma_0^2} \end{cases}
 \end{aligned}$$

We can actually use Bayesian inference to solve the linear regression problem.

$$\eta_p = A^{-1} \cdot B = \theta$$

$$\sigma_p^2 = A^{-1} = \text{variance (confidence)}$$

We can get posterior distribution provided prior information, data and prior information to follow a Gaussian distribution.

With this we obtain that the posterior distribution is a Gaussian distribution too where we can calculate mean and variance.

With this posterior mean and posterior variance we can make optimal prediction and quantify the uncertainty using the variance.

## Clustering – K-means

### Introduction

We're gonna see an important algorithm for clustering which is called K-means.

## Clustering

- Clustering: Partition the dataset into groups based on their similarity.



Clustering is a very important unsupervised learning task. The problem we are given is a set of data points and each of them represent features or variables.

The goal is to partition the data set into groups so that similar data points are grouped into the same cluster.

We need an algorithm that is able to group these points into sub-groups so we can sort of think of each of these groups represent some particular structure.

### Example 1: Google News

**Classification**

**Top Stories**

**Howard's end: Slugger NLCS MVP**  
MLB.com - **Corey Brock** - 41 minutes ago  
The Phillies are on their way back to the World Series. Ryan Howard's bat helped them get there. Howard was named the MVP of the National League Championship Series on Wednesday after helping the Phillies beat the Dodgers 4-1.

**Phillies 10, Dodgers 4: Phillies Slug Way Back to Series** New York Times  
Dodgers vs. Phillies Game 5 inning-by-ingning recap Los Angeles Times  
Chicago Tribune - **James City Star - Newsday**  
[Email this story](#)  
[all 1,937 news articles](#)

**Despite 1,100 cases, many worry about vaccination**  
Washington Post - **Bob Stein, Jennifer Aspinwall** - 1 hour ago  
Americans have become increasingly alarmed about the swine flu, but many are wary about getting vaccinated against the disease, according to a new Washington Post-ABC News poll.

**Walgreen Planning Key as Marion Co. prepares to open flu clinic** WISH-TV  
The quest for an H1N1 vaccine has been good business for some firms Boston Globe  
Chicago Tribune - **Harold Young - San Bernardino Sun - Tampa Tribune**  
[Email this story](#)  
[all 4,091 news articles](#)

**US 'Attentive' to Having Fair Afghan Runoff Vote, Gates Says**  
Bloomberg - **Viola Gienger** - 31 minutes ago  
Oct. 22 (Bloomberg) -- The US and its allies are "going to be clearly attentive" to ensuring that the Nov. 20 runoff election in Afghanistan is as free and fair as possible, Defense Secretary Robert Gates said.

**Video: Afghanistan Runoff Election Welcomed By Obama** The UpTake  
Gates says all eyes on Afghan runoff election The Associated Press  
Associated Press - **Steve Lohr - Post-Bulletin - Times Online**  
[Email this story](#)  
[all 9,547 news articles](#)

**In The News**

<b>Brooke Bundy</b>	<b>Kanye West</b>
<b>Steve Phillips</b>	<b>Creigh Deeds</b>
<b>Windows 7</b>	<b>Morgan Stanley</b>
<b>Somer Thompson</b>	<b>Sun Microsystems</b>
<b>Tarek Mehanna</b>	<b>Marc Savard</b>

Google News gives you a bunch of classified actually stories into different categories.

### Example 2: marketing Company

- Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs



We want to make different advertisements and we want to target different groups with similar behaviors sending the same type of advertisement.

Other examples: machine learning, computer vision, machine language processing.

In these examples clustering plays a fundamental role.

### K-means

#### **K-means Algorithm**

---

- **Inputs:**  $n$  objects (data points)  $\{x_i\}_{i=1}^n$  and a number  $K$  of clusters.

- **K-means:**

- Idea:

- Alternatively optimize

- i) the **assignment** of the data points different clusters.
    - ii) the **centroids** of the clusters.

- Algorithm:

- Initialize the centroids or assignments randomly;

- Iterate until convergence:

- i) **Assign each data point** ( $x_i$ ) to the cluster that has the closest centroid.
    - ii) **Update the centroids** of all the clusters.

We are given a set of data points:  $\{x_1, x_2, \dots, x_n\}$ . Each “ $x_i$ ” represent information about a particular entity (customer, person, etc).

Each “ $x_i$ ” then can have multiple dimensions and become a vector depending on which information we want to treat.

“K” is the number of clusters we want to partition the data. This is a given value and we have to assume a value. Sometimes we can try several values and pick the best.

With the data points and “K” we want to decide how to partition the data.

Goals:

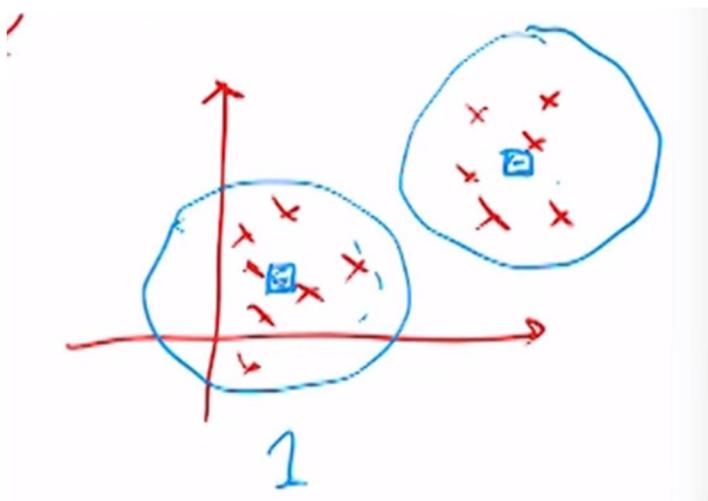
- Decide which cluster each data point belongs to.
- Decide a centroid of each of the clusters. It is the representative data point for each of the cluster and represents the average of the geometric center.

The goal of K-means is therefore to jointly decide both the assignment and the centroids.

The idea for this algorithm is to start from random initialization either for the centroid and the assignment and then iteratively update so we can improve sequentially the clusters.

We will repeat these steps until it converges and the iterative process changes nothing in the assignments in the current update.

Each iteration chooses a centroid and assigns the data point to the cluster that has the closest centroid.

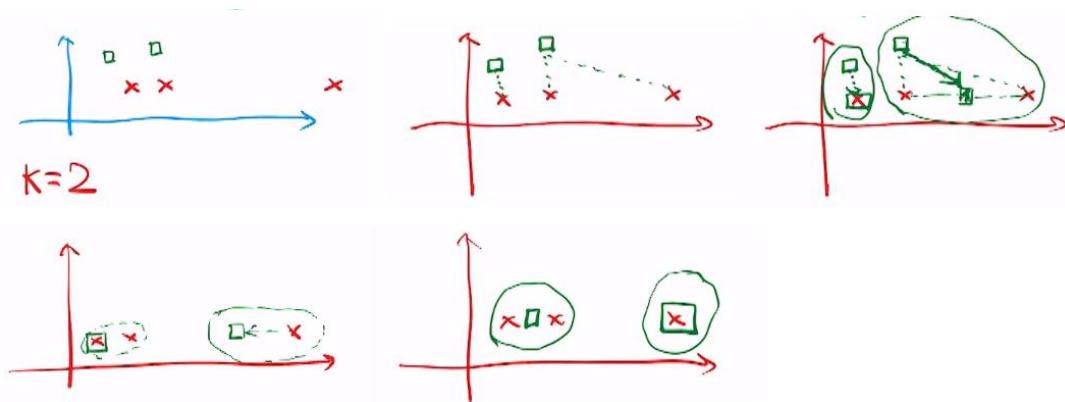


We compare the data points using the distance from each centroid.

After each update we need to realize if the centroid is reasonable or not, otherwise a more reasonable centroid needs to be found.

We then move and update the centroids averaging the middle point between the data points.

As long as the centroids aren't reasonable we need to continue moving the centroids using the average distance and taking the middle point between 2 data points in the cluster.



The algorithm stops as soon as we don't need to update at all.

The idea is that:

1. We start from random initialization assignments in centroids and then we iteratively improve the centroid and the assignment alternatively.
2. We assign the data point into the closest centroid.
3. Update the centroid so it becomes the more reasonable point given all the points in the cluster.
4. Repeat the process until at some point the assignment and the centroid is consistent.

Let's see the mathematical way to see the algorithm.

### Mathematical definition

#### K-means Algorithm

- **Inputs:**  $n$  objects (data points)  $\{x_i\}_{i=1}^n$  and a number  $K$  of clusters.

- **K-means Algorithm:**

- Initialization: randomly place  $K$  points (as the centroids)
- Iterate until convergence:
  - i) Assign each object to the group that has the closest centroid

$$z_i = \arg \min_{k=1, \dots, K} \|x_i - \mu_k\|$$

- ii) Recalculate the position of the  $K$  centroids

$$\mu_k = \frac{1}{|S_k|} \sum_{i \in S_k} x_i, \quad S_k = \{i : z_i = k\}.$$

“zi” belongs to  $\{1, 2, \dots, K\}$  represents the assignment of each data point: it is the distance of each “xi” to each centroid.

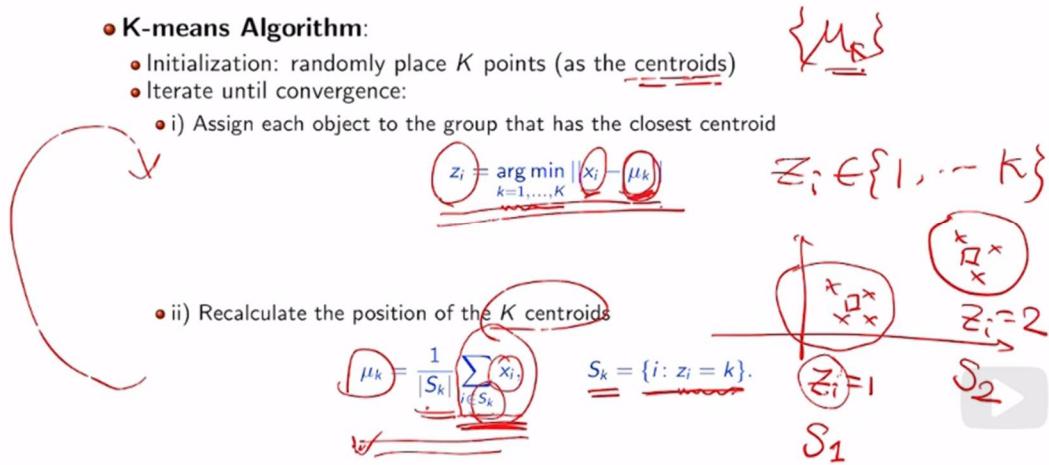
“ $\mu_k$ ” represents the centroid.

Update the centroids by calculating the mean of each data points that belongs to each of the centroids.

“Sk” is the size of the centroid: number of points in this cluster.

The new centroid is the average of all the data points in the cluster.

Once a new centroid is chosen, we repeat the process until the algorithm converges.



### K-means as Optimization

- 
- K-means can be viewed as a “coordinate descent” algorithm for optimizing an objective function of the centroids  $\{\mu_k\}$  and assignments  $\{z_i\}$ .

Now we need to understand the mathematical principle of this algorithm. In other words, it turns out that our k-means procedure can actually be viewed as some sort of coordinate descent algorithm in order to minimize the objective function that defines how well the data is partitioned.

Let's try using this framework to find out that k-means is actually an optimization algorithm which is very elegant as convergence is automatically guaranteed.

Given  $\{x_1, x_2, \dots, x_n\}$  data points.

We want  $\{\mu_1, \mu_2, \dots, \mu_K\}$  (the centroids) and  $\{z_1, z_2, \dots, z_n\}$  the assignments of each data point to one of the centroids.

We need to define an objective function (Loss function):

$$L(\mu, z) = \sum_{i=1}^n \|x_i - \mu_{z_i}\|^2$$

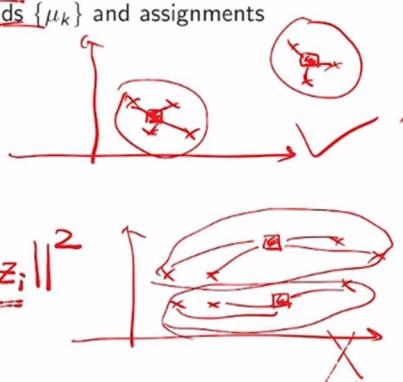
Goal is to minimize this Loss function getting to a point where we minimize the distance between the centroids and each of the data points.

- K-means can be viewed as a "coordinate descent" algorithm for optimizing an objective function of the centroids  $\{\mu_k\}$  and assignments  $\{z_i\}$ .

Given  $\{x_i\}_{i=1}^n$

Want  $\{\mu_k\}_{k=1}^K$ ,  $\{z_i\}_{i=1}^n$

Define:  $L(\mu, z) = \sum_{i=1}^n \|x_i - \mu_{z_i}\|^2$



We want to find the best centroids but also we want to find the best choice of assignments.

$$\min_{\mu, z} L(\mu, z) = \sum_{i=1}^n \|x_i - \mu_{z_i}\|^2$$

This is actually a tricky optimization because we have mixes optimization problema of " $\mu$ " and " $z$ " where  $\mu=\{\mu_1, \mu_2, \dots, \mu_K\}$  is a continuous function and  $z=\{z_1, z_2, \dots, z_n\}$  is a discrete variable that assigns each point to one cluster.

mixed optimization

$\mu \in \mathbb{R}^{d \times K}$        $z \in \{1, \dots, K\}^n$

However, the k-means algorithm is a very simple algorithm that actually tries to minimize this objective function using an idea called "coordinate descent".

The idea of coordinate descent is to update " $\mu$ " and " $z$ " alternatively:

1. Initialize " $\mu$ ".
2. Repeat:
  - a. update " $z$ " with fixed " $\mu$ ": start from random " $\mu$ " and update " $z$ " finding the optimal when " $\mu$ " is fixed.
  - b. Update " $\mu$ " with fixed " $z$ ".

## Coordinate descent.

Initialize  $\mu_0$

Repeat : (t-iteration)

① update  $Z$ , with fixed  $\mu$

$$\underline{z}_t = \operatorname{argmin}_z L(\underline{\mu}_t, z)$$

② update  $\mu$ , with  $Z$  fixed.

$$\underline{\mu}_{t+1} = \operatorname{argmin}_\mu L(\mu, \underline{z}_t)$$

We keep on repeating until the process converges.

If we implement this algorithm we will obtain the k-means algorithm we discussed earlier.

Remember the Loss function:

$$L(\mu, z) = \sum_{i=1}^n |x_i - \mu_{zi}|^2$$

In this case assume we are in step "t" and " $\mu_t$ " is fixed. We want to find the optimal "z" to minimize the Loss function when " $\mu$ " is a fixed value.

In order to find the optimal "zi":

$$z_i \leftarrow \operatorname{argmin}_{z_i} |x_i - \mu_{zi}|^2 \text{ (assignment step)}$$

In order to find optimal " $\mu_k$ " with "z" fixed:

$$\mu_k \leftarrow \operatorname{argmin}_{\mu_k} \sum_{i \in S_k} |x_i - z_k|^2, \text{ where } S_k = \{i: Z_i = k\}$$

So, "Sk" is the subset of data points that was assigned to "k" centroids from the last iteration. When we optimize " $\mu$ " we only care about the data points in "Sk".

$$L(\underline{\mu}, \underline{z}) = \sum_{i=1}^n \|x_i - \underline{\mu}_{\underline{z}_i}\|^2$$

Find optimal  $\underline{z}_i \leftarrow \operatorname{argmin}_{\underline{z}_i} \|x_i - \underline{\mu}_{\underline{z}_i}\|^2$  (Assignment Step)

Find optimal  $\underline{\mu}_k \leftarrow \operatorname{argmin}_{\underline{\mu}_k} \sum_{i \in S_k} \|x_i - \underline{\mu}_k\|^2$

$$S_k = \{i : z_i = k\}$$

If we work out this optimization, it's really just a quadratic optimization and we can show that the solution is exactly going to be the mean of "xi":

$$\begin{aligned} \mu_k &\leftarrow \operatorname{argmin}_{\mu_k} \sum_{i \in S_k} |x_i - z_k|^2 = \sum_{i \in S_k} (|x_i|^2 - 2 \cdot x_i^T \cdot \mu_k + |\mu_k|^2) \\ &= \text{constant} - 2 \cdot \left( \sum_{i \in S_k} x_i \right)^T \cdot \mu_k + |S_k| \cdot |\mu_k|^2 \end{aligned}$$

Turns out that the last term gets " $\mu_k$ " gets multiplied by " $S_k$ " (the number of items in the centroid as we have a sumatory).

If we solve this optimization problem we don't consider the constant part and we just make this term equal to "0" and then solve to get the optimal " $\mu_k$ ":

$$\mu_k = \frac{\sum_{i \in S_k} x_i}{|S_k|}$$

This is exactly the centroid update step of the k-means algorithm.

Find optimal  $z_i \leftarrow \underset{z_i}{\operatorname{argmin}} \|x_i - \mu_k\|^2$  (Assignment Step)

Find optimal  $\mu_k \leftarrow \underset{\mu_k}{\operatorname{argmin}} \sum_{i \in S_k} \|x_i - \mu_k\|^2$

$$S_k = \{i : z_i = k\}$$

$$= \sum_{i \in S_k} \|x_i\|^2 - 2x_i^T \mu_k + \|\mu_k\|^2$$

$$= \text{const} - 2 \left( \sum_{i \in S_k} x_i \right)^T \mu_k + \|S_k\| \|\mu_k\|^2$$

$$\Rightarrow \mu_k \leftarrow \frac{\sum_{i \in S_k} x_i}{\|S_k\|}$$

(Centroid update step)

### Limitations with the current algorithm

#### Local optimal vs Global optimal

This Loss function is going to be a non-convex optimization meaning it will have lots of local optimals.

Every time we start from some random initialization points we might just end up in a random local optimal. This is not guarantee we will find the global optimal centroids.

Finding the global optimal points is going to be a harder problem.

K-means algorithm is very sensitive to the initializations and in practice we try several different initializations and pick the solution that finds the minimum Loss function value.

It is guaranteed that the coordinate descent procedure always keeps on decreasing the Loss function in every update to find the local optimal. This is at least good, because maybe we cannot find the best global solution but we can find a local optimal.

Let's claim about monotonic decrease and coordinate descent:

" $\mu_t$ ": is the centroid that we found in iteration "t".

" $z_t$ ": is the assignment we made in iteration "t".

$$L(\mu_{t+1}, z_{t+1}) \leq L(\mu_t, z_t)$$

To prove this let's get back to the formula:

$$L(\mu_t, z_t) \geq L(\mu_{t+1}, z_t)$$

Alternatively:

$$L(\mu_t, z_t) \geq L(\mu_{t+1}, z_{t+1})$$

From each iteration the Loss function can never update and we should be able to find the local optimal.

EM algorithm

This is the Expectation Maximization algorithm. This is another clustering algorithm that generalizes the K-means algorithm which we introduced previously.

Let's remember the K-means algorithm:

---

### K-means vs. EM

- **Inputs:**  $n$  objects (data points)  $\{x_i\}_{i=1}^n$  and a number  $K$  of clusters.

- **K-means Algorithm:**

- Initialization: randomly place  $K$  points (as the centroids)
- Iterate until convergence:
  - i) Assign each object to the group that has the closest centroid

$$z_i = \arg \min_{k=1, \dots, K} \|x_i - \mu_k\|^2$$

- ii) Recalculate the position of the  $K$  centroids

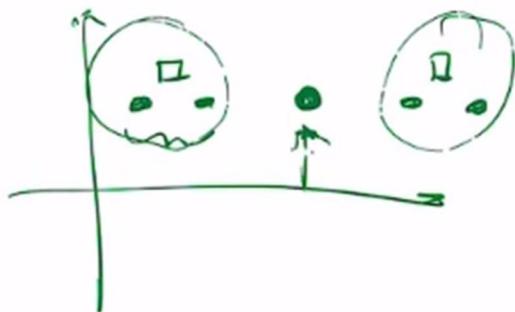
$$\mu_k = \frac{\sum_{i=1}^n \mathbb{I}(z_i = k) x_i}{\sum_{i=1}^n \mathbb{I}(z_i = k)}$$

It is a very simple clustering algorithm.

The idea is to start from some random centroids and iteratively update assignments by finding the closest centroids for each data point and also updating the centroids based on the assignments.

The problem is that for each data point we are forced to give an assignment and sometimes this is difficult to do.

With the example below we can see that assigning the middle point we are forced to make a decision:



EM algorithm is a sort of generalization of K-means that allows us to make soft or probabilistic assignments of the data points.

We can sort of introduce the procedure very quick and the idea is that instead of deterministic assignments we make probabilistic assignments:

$$p(z_i = k) = \frac{e^{-\frac{|x_i - \mu_k|^2}{\lambda}}}{\sum_{i=1}^k e^{-\frac{|x_i - \mu_k|^2}{\lambda}}}$$

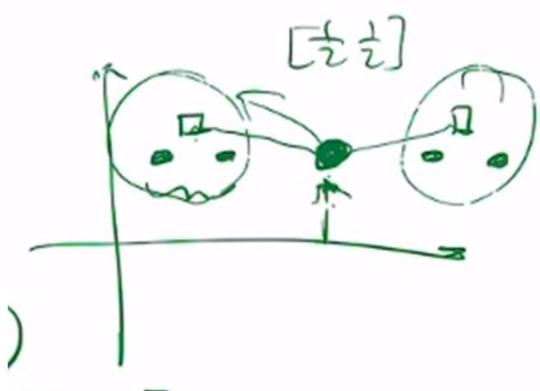
This probability is actually proportional to the distance but in a probabilistic way taking a kind of sigmoid function.

“ $\lambda$ ” is a parameter that scales that we can decide in the complete version of EM algorithm.

The idea is that instead of deterministically assigning each data point to the centroids that is close to it we actually maintain our probability about “ $z_i$ ” belonging to any of the centroids and we know that the probability is proportional to minus the square distance.

The division appears as a way to normalize the probability so that we reflect all the possible clusters.

Going back to the example above, the middle point that is exactly equally distanced from the two centroids, if we calculate the probability we will have something close to  $\frac{1}{2}, \frac{1}{2}$  for both centroids.



For other points we will see that one probability is much larger than the other centroids.

Summarizing: we maintain probabilities instead of making hard assignments and then after we calculate the probabilities we can update centroids.

In K-means the centroids are updated by averaging over the data points that are assigned to each centroid.

Now, defining randomized assigments we only know probabilities. And the mean needs to be a weighted average.

We also need to introduce an indicative function ("I") that indicates whether our point belongs or not to our cluster.

In K-means this indicative function had this form:

$$\begin{aligned} I(z_i = k) &= 1, \text{when } z_i = k \\ I(z_i = k) &= 0, \text{when } z_i \neq k \end{aligned}$$

In EM algorithm thsi indicative is replaced with the actual probability:

$$\mu_k = \frac{\sum_{i=1}^n p(z_i = k) \cdot x_i}{\sum_{i=1}^n p(z_i = k)}$$

We just replaced the indicative function with the actual soft probabillity or expectation.

**K-means Algorithm:**

- Initialization: randomly place  $K$  points (as the centroids)
- Iterate until convergence:
  - Assign each object to the group that has the closest centroid

$$z_i = \arg \min_{k=1,\dots,K} \|x_i - \mu_k\|^2$$

prob( $Z_i = k$ ) =  $\frac{\exp(-\|x_i - \mu_k\|^2 / \lambda)}{\sum_{l=1}^K \exp(-\|x_i - \mu_l\|^2 / \lambda)}$

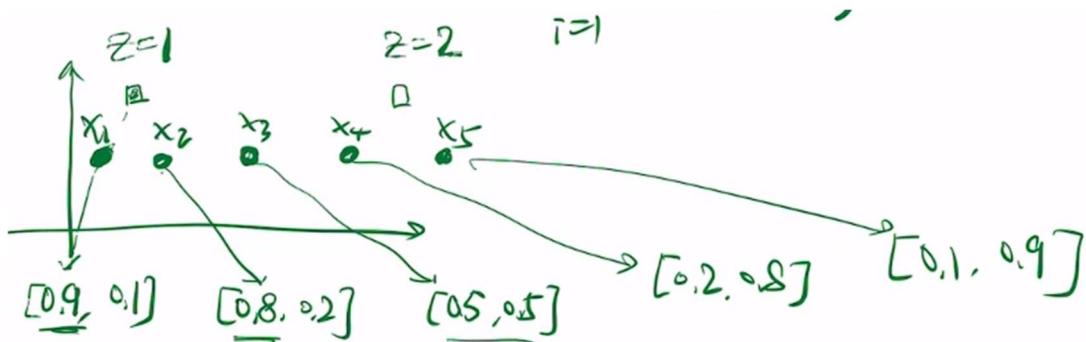
i) Recalculate the position of the  $K$  centroids

$$\mu_k = \frac{\sum_{i=1}^n \mathbb{I}(z_i = k) x_i}{\sum_{i=1}^n \mathbb{I}(z_i = k)}$$

$\mathbb{I}(Z_i = k)$  = 1 .  $z_i = k$   
0 .  $z_i \neq k$

$$\mu_k = \frac{\sum_{i=1}^n \text{prob}(z_i = k) x_i}{\sum_{i=1}^n \text{prob}(z_i = k)}$$

Let's go over the example we had and re-draw the graph:



1. Take random centroids.
2. For each point calculate the probability they belong to each of the centroids.
3. Update the centroids: " $\mu_k$ ": we actually see that each point contributes partially (given by its probability value) to the updated new centroid.
  - a. for " $\mu_1$ ":

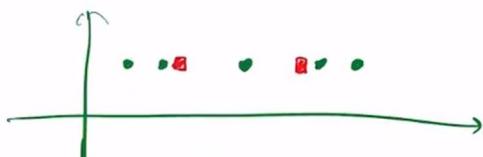
$$\mu_1 = \frac{0.9 \cdot x_1 + 0.8 \cdot x_2 + 0.5 \cdot x_3 + 0.2 \cdot x_4 + 0.1 \cdot x_5}{0.9 + 0.8 + 0.5 + 0.2 + 0.1}$$

1. for " $\mu_2$ ":

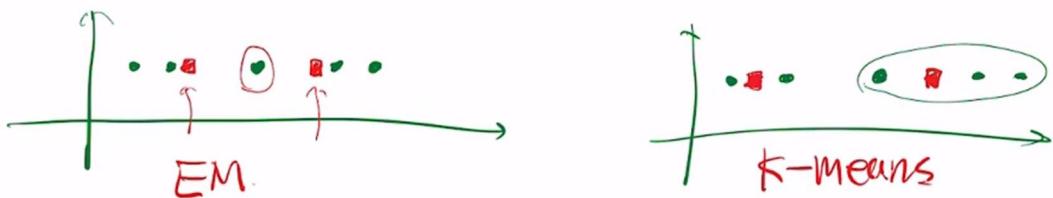
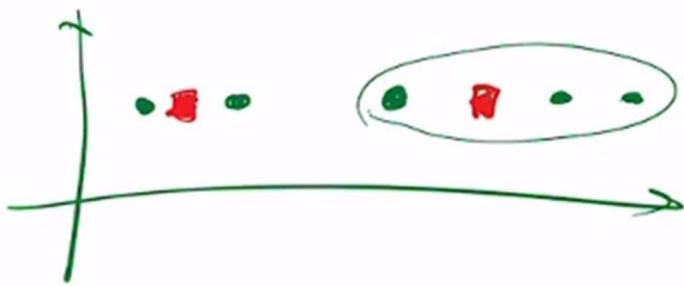
$$\mu_2 = \frac{0.1 \cdot x_1 + 0.2 \cdot x_2 + 0.5 \cdot x_3 + 0.8 \cdot x_4 + 0.9 \cdot x_5}{0.1 + 0.2 + 0.5 + 0.8 + 0.9}$$

The updated decision now moves these centroids:

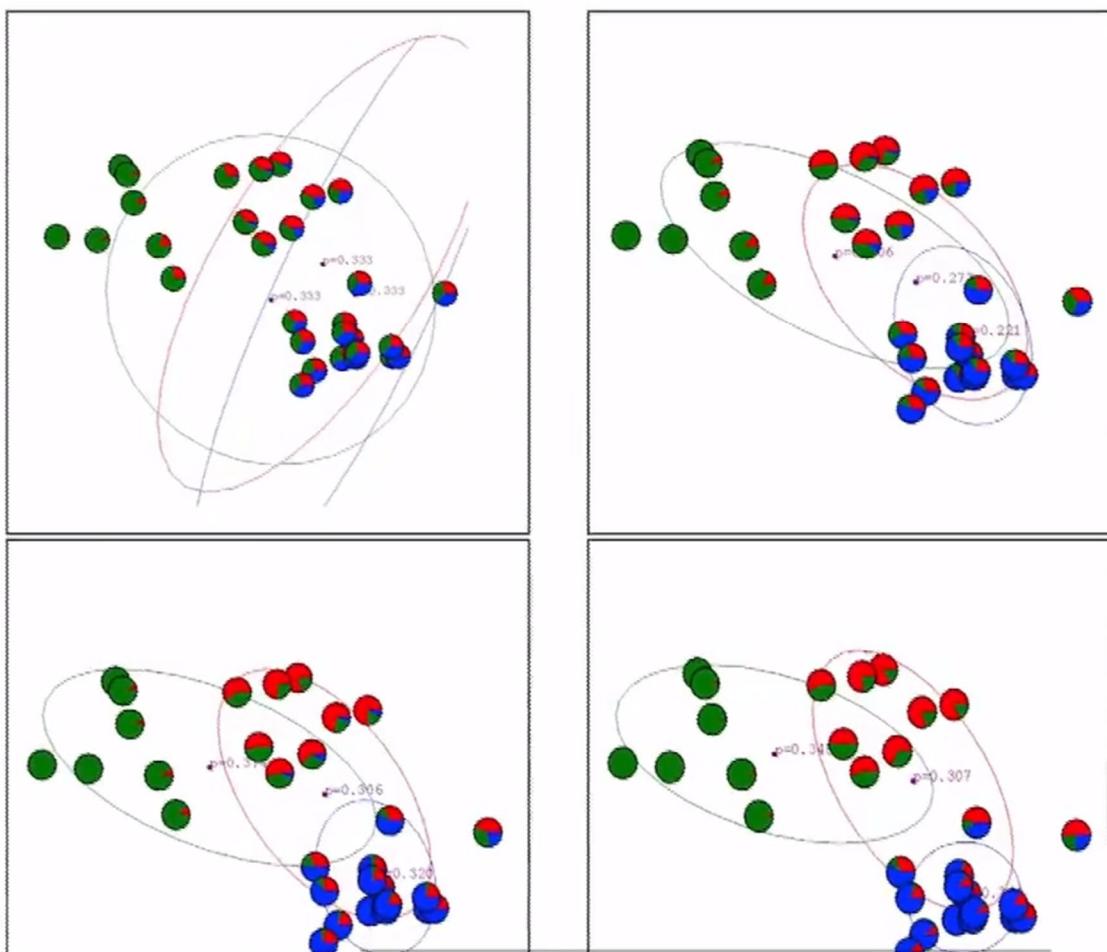
$$\left\{ \begin{array}{l} \text{For } \mu_1 = \frac{0.9x_1 + 0.8x_2 + 0.5x_3 + 0.2x_4 + 0.1x_5}{0.9 + 0.8 + 0.5 + 0.2 + 0.1} \\ \mu_2 = \frac{0.1x_1 + 0.2x_2 + 0.5x_3 + 0.8x_4 + 0.9x_5}{0.1 + 0.2 + 0.5 + 0.8 + 0.9} \end{array} \right.$$



Doing the K-means algorithm, we get something different because we are forced to assign the middle point to one of the clusters and depending on the decision the centroids will be different:



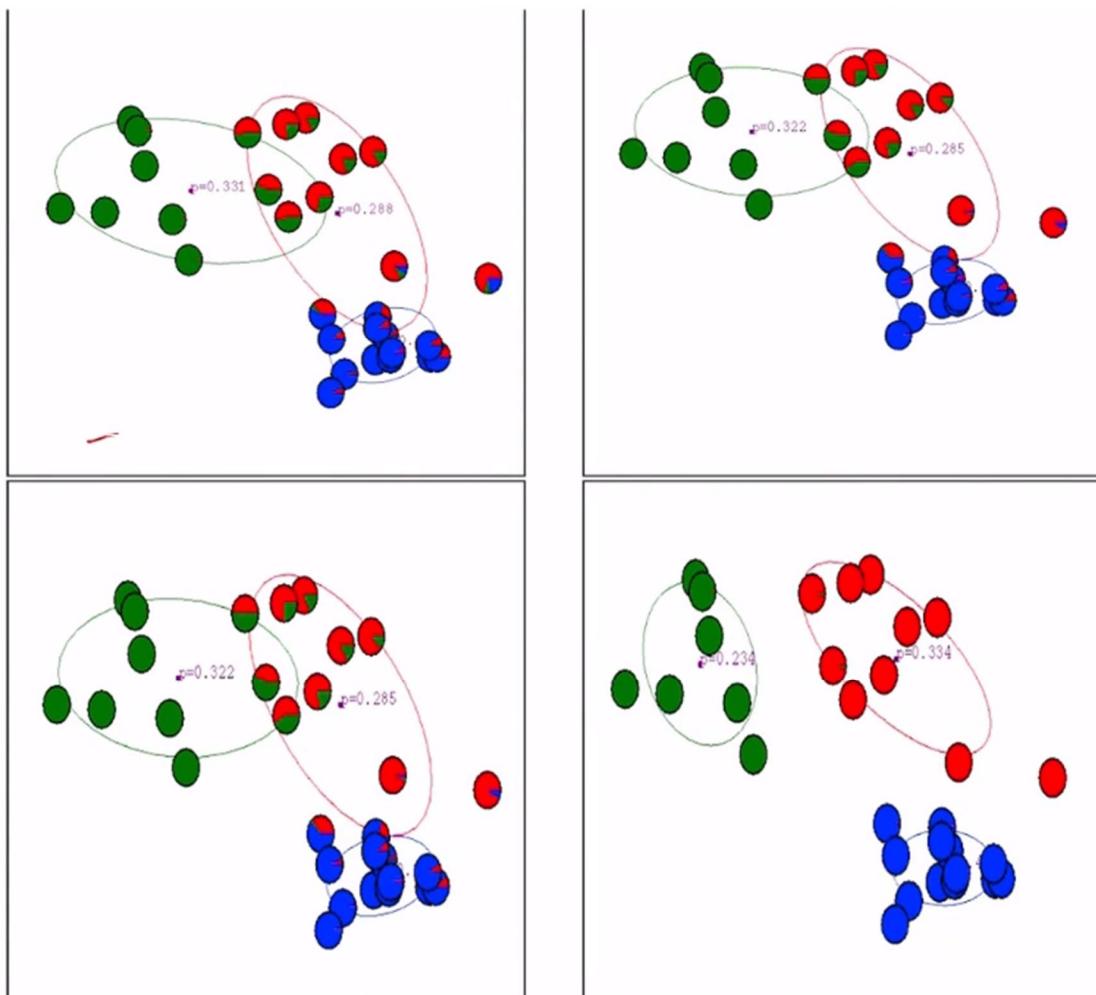
The intuition is replacing the deterministic assignment with the probability and then we update the centroids using a weighted sum of all the data points.



For each data point we have different probabilities that it is assigned to different centroids.

In the initialization have very mixed probabilities and all the centers have significant overlap.

As we perform EM updates we can sort of see the centroids move and the probability updates too.



As we repeat the process gradually the different centroids are separating from each other and the probability becomes deterministic.

In the next lecture, we are going to introduce the mathematical idea behind EM algorithm and we will show that the EM algorithm is actually an optimization algorithm for performing Maximum Likelihood Estimation of Gaussian mixture models.

#### Clustering problem with MLE

Let's define the clustering problem in detail.

## Clustering

- **Inputs:**  $n$  objects (data points)  $\{x_i\}_{i=1}^n$  and a number  $K$  of clusters.
- **Goal:** Group the points into several groups
  - Deciding a group ID,  $z_i \in \{1, \dots, K\}$ , for each data point  $x_i$ .

Input: a set of points “ $x_i$ ”.

Goal: partition these points into a few number of groups. “ $k$ ” represents the number of groups.

Formerly this problem could be viewed as assigning “ $z_i$ ” labels to the data points where label is an integer number between 1 and “ $k$ ”.

The actual number of the labels can be permuted arbitrary.

K-means was a simple algorithm for performing clustering.

EM algorithm is another more principled and more powerful way that quantifies the uncertainty.

In addition, we will introduce the mathematical ideas behind EM algorithm.

## Probabilistic Modeling of Clustering

### □ Probabilistic Approach:

- Assume a joint distribution  $p(x, z | \theta)$  that generates data and labels  $(x, z)$ .
- Estimate parameter  $\theta$ .
- Infer the labels from the posterior distribution  $p(z | x; \theta)$ .

The idea behind to solve this problem via MLE is to assume that there is a joint distribution “ $p(x,z)$ ” for points and labels so for each data point “ $x_i$ ” we also have an associated “ $z_i$ ”.

Despite not observing the actual labels “ $z_i$ ” (we only observe “ $x_i$ ” and “ $z_i$ ” is latent) we can assume labels are drawn from the same distribution as points “ $x_i$ ”.

We want to estimate a parameter “ $\theta$ ” that best fits the observed data “ $x_i$ ” as well as the labels “ $z_i$ ”: “ $p(x,z | \theta)$ ”.

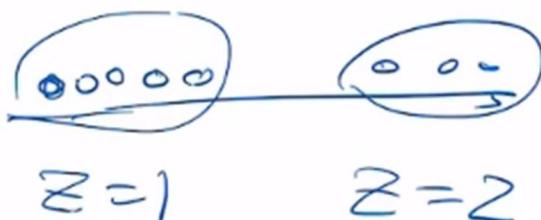
The approach to follow is similar to the estimation problem where we want to estimate the parameter “ $\theta$ ”.

Once we estimate the parameter “ $\theta$ ” we can estimate the label using the posterior distribution: “ $p(z | x, \theta)$ ”.

In order to specify the joint distribution “ $p(x,z)$ ”, we can break it into two parts using the chain rule.

$$p(x, z) = p(z) \cdot p(x|z)$$

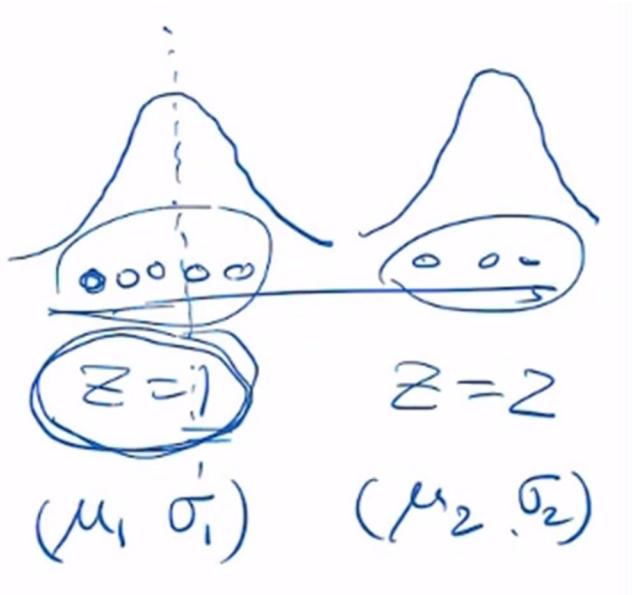
In order to establish the probability let us decide randomly to which cluster each data points belongs to:



$$p(x, z) = p(z) \cdot p(x|z)$$

$$(I) p(z = k) = \pi_k (\text{probability of } z = k)$$

Let us assume that the data points are grouped along a centroid following a Gaussian distribution with their mean and variance:



The conditional distribution for "x" when belonging to certain centroid and following a Gaussian distribution is:

$$(II) p(x | z = k) = N(x | \mu_k, \sigma_k^2) (\text{probability of point } x \text{ belonging to group } k)$$

We can assume that in these normal distributions the mean " $\mu_k$ " will be the center of the cluster "k".

$$(III) p(x, z = k) = p(z = k) \cdot p(x | z = k) = \pi_k \cdot N(x | \mu_k, \sigma_k^2)$$

This means that the joint distribution of "x" and "z" will exactly be equal to the probability of "z=k" times the Gaussian distribution of points around the centroid.

We actually have introduced a bunch of parameters " $\theta$ ":

$$\theta = \{\pi_k, \mu_k, \sigma_k\}_{k=1}^K$$

We do not observe these parameters directly:

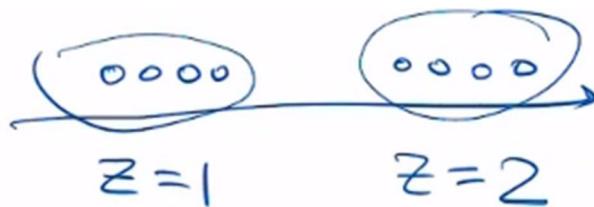
$$p(x, z | \theta) = p(x, z | \pi, \mu, \sigma)$$

Easy example for MLE**MLE with Complete Information**

- If we observe both data and labels  $\{x_i, z_i\}_{i=1}^n$  (**complete information**), we can estimate the parameter  $\theta$  by maximizing the **joint probability**:

$$\max_{\theta} \sum_{i=1}^n \log p(x_i, z_i | \theta)$$

In the case we have complete information, let us estimate the best parameter “ $\theta$ ” that maximizes the joint probability.



In this easy case:

$$\pi_1 = \frac{\text{number of cases where } z = 1}{\text{all cases}} = \frac{4}{8} = \frac{1}{2} = 0,5$$

This is easy to follow as the probability of choosing a point randomly that belongs to “ $z=1$ ” is 50% as there are 8 total points and 4 belong to “ $z=1$ ”.

$$\mu_1 = \frac{\sum_{i=1}^n I(z_i = 1) \cdot x_i}{\sum_{i=1}^n I(z_i = 1)}$$

Remember the indicator function:

$$\begin{aligned} I(z_i = k) &= 1, \text{when } z_i = k \\ I(z_i = k) &= 0, \text{when } z_i \neq k \end{aligned}$$

This is equivalent to:

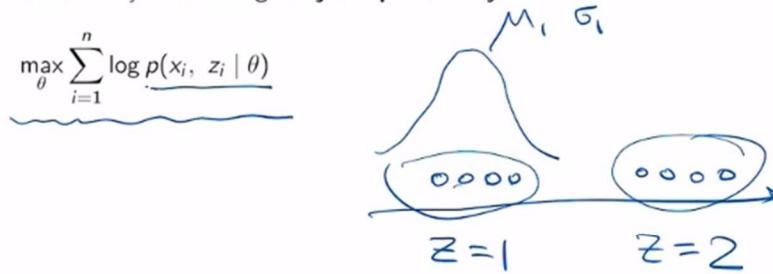
$$\mu_1 = \frac{\sum_{i \in z=1} x_i}{\#(z = 1)}$$

It is the average value of “ $x$ ” for all points in the first cluster.

Similarly we calculate the empirical variance of data points where “ $z=1$ ”.

$$\sigma_1^2 = \text{empirical variance of data points where } z = 1$$

- If we observe both data and labels  $(x_i, z_i)_{i=1}^n$  (**complete information**), we can estimate the parameter  $\theta$  by maximizing the **joint probability**:



$$\pi_1 = \frac{\#(z=1)}{n}$$

$$\mu_1 = \frac{\sum_{i=1}^n I(z_i=1)x_i}{\sum_{i=1}^n I(z_i=1)}$$

$$\sigma_1^2 = \text{Var}(\{x_i | z_i=1\})$$

If we think about this problem mathematically, we are trying to find a maximum joint likelihood.

We know that:

$$(IV) p(x_i, z_i = k | \theta) = p(z = k) \cdot p(x | z = k) = \pi_k \cdot N(x | \mu_k, \sigma_k^2)$$

Introducing the previous formula provided in the statement of the problem and generalizing for all the observed points in the given clusters:

$$(V) p(x_i, z_i = k | \theta) = \sum_{i=1}^n \log P(x_i, z_i | \theta) = \sum_{i=1}^n \sum_{k=1}^K I(z_i = k) \cdot \log P(x_i, z_i = k | \theta)$$

This is really a rewrite of the joint likelihood that we have seen in the problem statement above.

This just plug the formulas normal distribution of points in the cluster (IV) and (V):

$$\begin{aligned} p(x_i, z_i = k | \theta) &= \sum_{i=1}^n \sum_{k=1}^K I(z_i = k) \cdot \log P(x_i, z_i = k | \theta) \\ &= \sum_{i=1}^n \sum_{k=1}^K I(z_i = k) \cdot \log(\pi_k \cdot N(x | \mu_k, \sigma_k^2)) \\ &= \sum_{i=1}^n \sum_{k=1}^K (z_i = k) \cdot (\log(\pi_k) + \log N(x | \mu_k, \sigma_k^2)) \end{aligned}$$

This is easily prove that in order to maximize this function we need to maximize both terms:

$$\pi_k = \frac{\sum_{i=1}^n I(z_i = k)}{n}$$

This matches the empirical heuristic idea we had earlier. This can be easily obtained with MLE theory.

The optimal mean “ $\mu$ ” (center) and variance “ $\sigma^2$ ” (radius) of the centroids is going to be the empirical mean and variance of the data points where “ $z_i=k$ ”.

If we actually observe both “ $x_i$ ” and “ $z_i$ ” we can estimate the parameters fairly easily using this heuristic argument of empirical mean and empirical variance.

□ If we observe both data and labels  $(x_i, z_i)_{i=1}^n$  (complete information), we can estimate the parameter  $\theta$  by maximizing the joint probability:

$$\max_{\theta} \sum_{i=1}^n \log p(x_i, z_i | \theta)$$

$$P(x_i, z_i = k | \theta) = \pi_k N(x_i | \mu_k, \sigma_k^2)$$

$$\sum_{i=1}^n \sum_{k=1}^K I(z_i = k) \log P(x_i, z_i = k | \theta)$$

$$= \sum_{i=1}^n \sum_{k=1}^K I(z_i = k) \log (\pi_k N(x_i | \mu_k, \sigma_k^2))$$

$$= \sum_{i=1}^n \sum_{k=1}^K I(z_i = k) (\log \pi_k + \log N(x_i | \mu_k, \sigma_k^2))$$

$$\pi_k = \frac{\sum_{i=1}^n I(z_i = k)}{n}$$

$$\pi_1 = \frac{\#(z=1)}{n} = \frac{1}{2}$$

$$\mu_1 = \frac{\sum_{i=1}^n I(z_i = 1) x_i}{\sum_{i=1}^n I(z_i = 1)}$$

$$\sigma_1^2 = \text{Var}(\{x_i | z_i = 1\})$$

### General example for MLE

### Clustering and Mixture Models

□ If we only observe data  $\{x_i\}_{i=1}^n$  (incomplete information), we shall estimate  $\theta$  by maximizing the marginal probability:

$$\sum_{i=1}^n \log p(x_i | \theta) = \sum_{i=1}^n \log \sum_{z_i} p(x_i, z_i | \theta)$$

$z_i$ : missing information.

Unfortunately, in our problem, we do not observe the clusters “z” and therefore we have a latent variable. We only observe the data points “xi” and then “zi” is the missing information.

In this case, we want to estimate “θ” despite having incomplete information.

Fortunately, we can estimate the “θ” parameter using MLE except that now the likelihood to optimize should be defined based on what we observe: in this case {x1, x2, ..., xn}.

The maximization problem now is defined as:

$$(now) \sum_{i=1}^n \log(x_i | \theta) = \sum_{i=1}^n \log \left( \sum_{z_i} p(x_i, z_i | \theta) \right)$$

Remember the previous problem:

$$(V) \sum_{i=1}^n \log p(x_i, z_i | \theta)$$

In this case, both “xi” and “zi” were observed values and we just needed to maximize the probability. Actually, in this case, the optimal values resulted from the marginal probability of each “xi” and “zi”.

Now, in the missing “zi” case, we only observe “xi” and the MLE is going to maximize the likelihood of “xi”. We have to consider all the cases of possible values of “zi” for each point; this is why we have now two summation parts.

This sum over all the possible labels “zi” makes the problem much more difficult. Previously, it was enough maximizing the joint distribution. Now maximizing this joint distribution, we actually don’t have a nice close form solution and we need numerical algorithms.

Remember the marginal distribution of “x” (“p(x|θ)”) and let’s apply the rule of total probability and the formulas seen previously (III):

$$p(x|\theta) = \sum_{k=1}^K p(x, z=k | \theta) = \sum_{k=1}^K \pi_k \cdot N(x|\mu_k, \sigma_k^2)$$

In order to maximize and then solve this problem let us take the log-probability:

$$\sum_{i=1}^n \log(p(x_i | \theta)) = \sum_{i=1}^n \log \left( \sum_{k=1}^K \pi_k \cdot N(x_i | \mu_k, \sigma_k^2) \right)$$

This is the objective function we want to maximize with respect to each group:

$$\theta = \{ \pi_k, \mu_k, \sigma_k \}_{k=1}^K$$

The summation over “k” is inside the log and pushing the summation outside the log would then lead the problem to a reduction of the earlier joint probability.

But, in this case, this is a much more difficult problem. In this case the marginal distribution here is a sum of Gaussian distributions. This combined distribution based on several Gaussians is called a “mixture of Gaussian”:

$$p(x|\theta) = \sum_{k=1}^K \pi_k \cdot N(x|\mu_k, \sigma_k^2)$$

This is a mixture model. Plotting this is like having several Gaussians: several means and variances.

$$\begin{aligned} p(x|\theta) &= \sum_k p(x, z=k|\theta) \\ &= \sum_k \pi_k N(x|\mu_k, \sigma_k^2) \end{aligned}$$

mixture  
of Gaussian  
distributions

$$\sum_{i=1}^n \log p(x_i|\theta) = \sum_{i=1}^n \log \left( \sum_{k=1}^K \pi_k N(x_i|\mu_k, \sigma_k^2) \right)$$

In our case, we know there are half the points in "z=1" and half the points in "z=2", therefore:

$$\pi_1 = \frac{\text{number of cases where } z=1}{\text{all cases}} = \frac{4}{8} = \frac{1}{2} = 0,5$$

$$\pi_2 = \frac{\text{number of cases where } z=2}{\text{all cases}} = \frac{4}{8} = \frac{1}{2} = 0,5$$

With this the marginal distribution "p(x|θ)" takes this shape in our example:

$$p(x|\theta) = 0,5 \cdot \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma_1} \cdot e^{-\frac{(x-\mu_1)^2}{2 \cdot \sigma_1^2}} + 0,5 \cdot \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma_2} \cdot e^{-\frac{(x-\mu_2)^2}{2 \cdot \sigma_2^2}}$$

The problem is how to maximize the likelihood of this mixture Gaussian distribution.

If we want to solve this problem using some simple naïve algorithm we can just do grade descent over this objective function.

$$\begin{aligned}
 &= \sum_{k=1}^K \pi_k N(x | \mu_k, \sigma_k^2) \\
 \sum_{i=1}^n \log P(x_i | \theta) &= \sum_{i=1}^n \log \left( \sum_{k=1}^K \pi_k N(x | \mu_k, \sigma_k^2) \right) \\
 &\quad \text{Mixture of Gaussian distributions} \\
 &\quad \text{MAX } \{\pi_k, \mu_k, \sigma_k^2\}_{k=1}^K
 \end{aligned}$$
$$\begin{aligned}
 P(x | \theta) &= \frac{1}{2} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) \left(\frac{1}{\sqrt{2\pi}\sigma_1}\right) \\
 &\quad + \frac{1}{2} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right) \left(\frac{1}{\sqrt{2\pi}\sigma_2}\right)
 \end{aligned}$$

EM algorithm procedure**Expectation Maximization: Algorithm Procedure**

**Expectation Maximization (EM)** is an iterative method for maximizing the marginal likelihood.

- Initialize parameter  $\theta_0$ .
- For iteration  $t$ :
  - Given  $\theta_t$ , "impute" the missing labels by drawn samples from the posterior distribution

$$z_i \sim p(z | x_i; \theta_t).$$

- Update  $\theta$  by maximizing the expected joint likelihood:

$$\theta^{t+1} = \arg \max_{\theta} \sum_{i=1}^n \mathbb{E}_{z_i \sim p(\cdot | x_i; \theta_t)} [\log p(x_i, z_i | \theta)].$$

Turns out that EM algorithm, is an iterative algorithm very similar to grade descent and allows us to maximize the marginal distribution for this mixture likelihood.

1. We start from some initialization point and a initial " $\theta_0$ " value.

2. In each iteration "t", we input the missing labels "z\_i" based on estimation from the EM algorithm using the posterior distribution of the current parameter " $\theta_t$ ": given parameter " $\theta_t$ " we get the posterior distribution of " $z_i$ " and we impute the missing labels.
3. Finally, we update " $\theta_{t+1}$ " by maximizing the expected joint likelihood. The joint distribution of the expectation of " $z_i$ " from the posterior distribution summing over all the data points " $x_i$ ".

It turns out this procedure can be implemented very easily for simple models such as Gaussian mixture meaning that we can actually solve both of the previous steps very easily and we can actually show that this procedure monotonically decrease the last function.

Let us try implementing this algorithm:

1. Given some " $\theta_t$ " obtain the posterior distribution " $p(x|\theta)$ ".

$$\theta_t = [\pi_k^t, \mu_k^t, \sigma_k^t]_{k=1}^K$$

2. Now we have the probability distribution for us to impute the labels " $z_i$ ".

Let us calculate each posterior distribution for each cluster using conditional probability rule:

$$p(z_i = k|x_i, \theta_t) = \frac{p(x_i, z_i = k|\theta_t)}{\sum_{j=1}^K p(x_i, z_i = j|\theta_t)} = \frac{\pi_k \cdot N(x_i|\mu_k^t, \sigma_k^t)}{\sum_{j=1}^K \pi_j \cdot N(x_i|\mu_j^t, \sigma_j^t)}$$

$\square$  Initialize parameter  $\theta_0$ .  
 $\square$  For iteration  $t$ :  
 ○ Given  $\theta_t$ , "impute" the missing labels by drawn samples from the posterior distribution

$$p(z_i = 1 | x_i, \theta_t) = \frac{\pi_1 \cdot \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma_1} \cdot e^{-\frac{(x_i - \mu_1)^2}{2 \cdot \sigma_1^2}}}{\sum_j \pi_l \cdot \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma_j} \cdot e^{-\frac{(x_i - \mu_j)^2}{2 \cdot \sigma_j^2}}}$$

3. Finally, we update “ $\theta_{t+1}$ ”, but first let us introduce a “ $\gamma_{ik}$ ” variable (posterior distribution):

$$\gamma_{ik}^t = p(z_i = k | x_i, \theta_t)$$

Let us define the joint likelihood function, as in the statement above, but now using this “ $\gamma_{ik}$ ” variable:

$$L(\theta) = \sum_{i=1}^n E_{z_i \sim p(\cdot | x_i, \theta_t)} \cdot [\log p(x_i, z_i = k | \theta)] = \sum_{i=1}^n \sum_{k=1}^K \gamma_{ik}^t \cdot \log p(x_i, z_i = k | \theta)$$

Where the expectation if the sum of “ $\gamma_{ik}$ ” over all clusters. We need to maximize the above function. This function to maximize

We can easily show that in order to maximize this function parameters take the following form:

$$\pi_k^{t+1} = \frac{\sum_{i=1}^n \gamma_{ik}^t}{n}$$

$$\mu_k^{t+1} = \frac{\sum_{i=1}^n \gamma_{ik}^t \cdot x_i}{\sum_{i=1}^n \gamma_{ik}^t}$$

$\sigma_k^{t+1} = \text{similarly as done for the mean empirically}$

$$\theta^{t+1} = \arg \max_{\theta} \sum_{i=1}^n \mathbb{E}_{z_i \sim p(\cdot | x_i; \theta_t)} [\log p(x_i, z_i | \theta)]. \triangleq L(\theta)$$

Define  $\gamma_{ik}^t = P(z_i=k | x_i, \theta_t)$

$$L(\theta) = \sum_{i=1}^n \sum_{k=1}^K \gamma_{ik}^t \log P(x_i | z_i=k, \theta)$$

$$\Rightarrow \pi_k^{t+1} = \frac{\sum_{i=1}^n \gamma_{ik}^t}{n}$$

$$\mu_k^{t+1} = \left( \sum_{i=1}^n \gamma_{ik}^t x_i \right) / \left( \sum_{i=1}^n \gamma_{ik}^t \right)$$

$$\sigma_k^{t+1} = \dots$$



Then, we just repeat this process iteratively. Every time we calculate the posterior distribution, which is summarized as " $\gamma_{ik}$ ". And then given the posterior distribution we want to maximize the expected joint distribution updating " $\pi$ ", " $\mu$ " and " $\sigma$ " in every step " $t$ ".

Iterating it is guaranteed to actually converge into the global optimal solution but this can only be achieved under some scenarios which will be discussed in the next section.

Derivation of EM algorithm for global optimal solution

## Expectation Maximization: Derivation

- Consider general “imputation distributions”  $\rho(z | x)$ .
- We can construct a **tight lower bound**  $LB(\theta, \rho)$  of the marginal loglikelihood function:

$$L(\theta) \geq LB(\theta, \rho), \quad \forall \rho, \quad \text{and} \quad L(\theta) = \max_{\rho} LB(\theta, \rho).$$

- Maximizing  $L(\theta)$  is then equivalent to maximizing  $LB(\theta, \rho)$ .

$$\max_{\theta} L(\theta) = \max_{\theta, \rho} LB(\theta, \rho).$$

- Optimizing  $\theta$  and  $\rho$  alternatively (coordinate descent) yields EM algorithm.

Remember the problem we have: we want to maximize the marginal likelihood (i.e maximize “ $\theta=\{\pi_k, \mu_k, \sigma_k\}_{k=1}^K$ ”) such that:

$$\max_{\theta} \sum_{i=1}^n \log P(x_i | \theta) = L(\theta)$$

Turns out that there is a EM algorithm that follows an iterative process to monotonically increase the likelihoos function above.

The mathematical idea hebind is the following: we can show that for any imputation distribution “ $\rho(z|x)$ ” that allows us to guess the label “z” given “x”. Given this imputation distribution the EM procedure is equivalent to constructing some Lower Bound (“ $LB(\theta, \rho)$ ”) that depends on both “ $\theta$ ” and “ $\rho$ ” and provides a lower bound for the true likelihood function “ $L(\theta)$ ”:

$$L(\theta) \geq LB(\theta, \rho), \forall \rho$$

Moreover, the “LB” is also “tight” which means that if we mazimize over all possible “imputation distributions” it gives us the exact marginal likelihood we want to estimate:

$$L(\theta) = \max_{\rho} LB(\theta, \rho)$$

Therefore, maximizing “ $LB(\theta, \rho)$ ” for a given “ $\theta$ ” is equivalent to maximizing “ $L(\theta)$ ”:

$$\max_{\theta} L(\theta) = \max_{\theta, \rho} LB(\theta, \rho)$$

It turns out that EM algorithm is actually equivalent to maximizing this “ $LB(\theta, \rho)$ ” optimizing “ $\theta$ ” and “ $\rho$ ” in separate steps doing a coordinate descent.

Let us introduce this idea into detail and get more confidence about the Lower Bound.

Let us write down again the likelihood function:

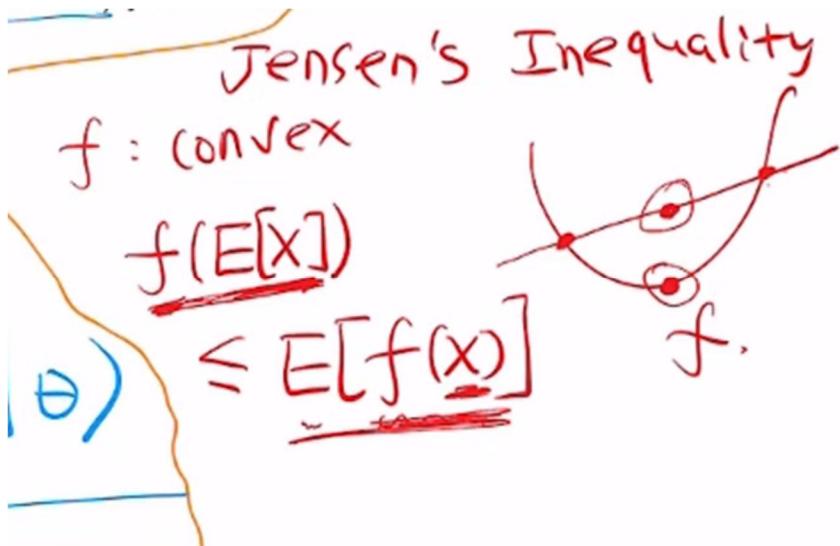
$$L(\theta) = \sum_{i=1}^n \log \left( \sum_{z_i} p(x_i, z_i | \theta) \right)$$

Remember the difficult part was to get rid of the summation of “zi”. We need to somehow push the summation outside the log and we will use something called Jensen’s inequality.

#### Jensen's inequality

For any convex function “f(x)” we can show that:

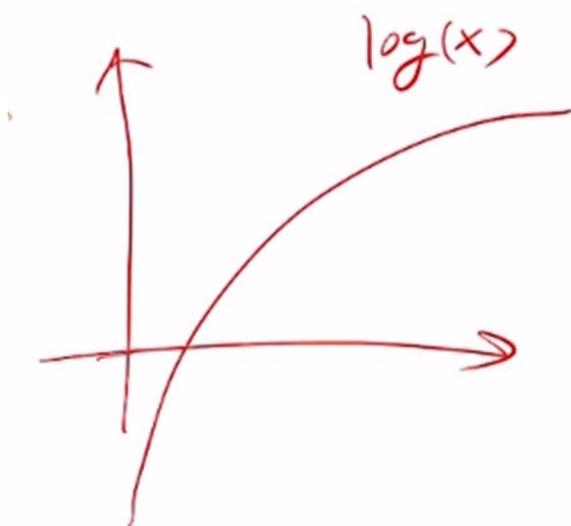
$$f(E[x]) \leq E[f(x)]$$



We need to apply Jensen's inequality to this likelihood in order to get a lower bound.

On the other hand, the above “log” function is concave:

$$\log \left( \sum_{z_i} p(x_i, z_i | \theta) \right)$$



However, if we take the “ $-\log$ ” function we will get a convex function and we will be able to apply Jensen’s inequality and get a bound.

The only gap that would be left is that we need to turn the part inside the “ $\log$ ” function an expectation proceeding like this and introducing the imputation distribution “ $\rho(z|x)$ ”:

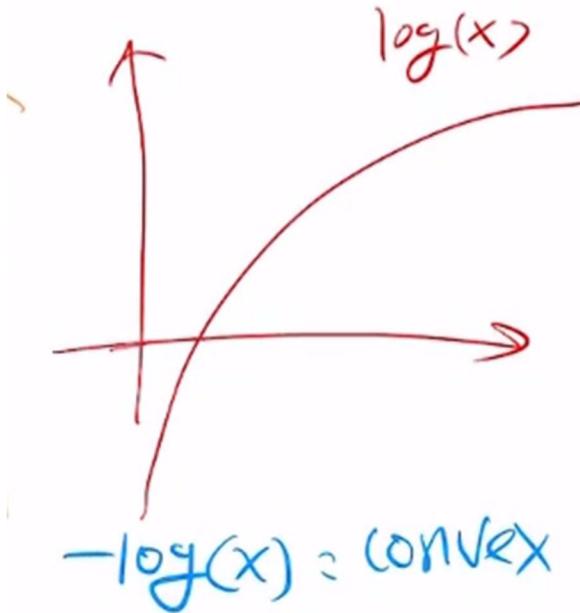
$$\begin{aligned} L(\theta) &= \sum_{i=1}^n \log \left( \sum_{z_i} p(x_i, z_i | \theta) \right) = \sum_{i=1}^n \log \left( \sum_{z_i} p(x_i, z_i | \theta) \cdot \frac{\rho(z_i | x_i)}{\rho(z_i | x_i)} \right) \\ &= \sum_{i=1}^n \log \left( E_{z_i \sim \rho(\cdot | x_i)} \left[ \frac{p(x_i, z_i | \theta)}{\rho(z_i | x_i)} \right] \right) \end{aligned}$$

We have rewritten the summation as an expectation:

$$E[x] = \sum_{i=1}^n x_i \cdot p_i \rightarrow E_{z_i \sim \rho(\cdot | x_i)} \left[ \frac{p(x_i, z_i | \theta)}{\rho(z_i | x_i)} \right] = \sum_{z_i} \left( \frac{p(x_i, z_i | \theta)}{\rho(z_i | x_i)} \right) \cdot (\rho(z_i | x_i))$$

Log is a concave function, but a negative log is actually a convex function. We can apply Jensen’s inequality but in opposite direction:

$$f(E[x]) \geq E[f(x)]$$



Therefore, we obtain:

$$L(\theta) = \sum_{i=1}^n \log \left( E_{z_i \sim \rho(\cdot | x_i)} \left[ \frac{p(x_i, z_i | \theta)}{\rho(z_i | x_i)} \right] \right) \geq \sum_{i=1}^n E_{z_i \sim \rho(\cdot | x_i)} \cdot \log \left( \frac{p(x_i, z_i | \theta)}{\rho(z_i | x_i)} \right) = LB(\theta, \rho)$$

We pushed the expectation outside the log function.

Choosing the “ $\rho$ ” properly we could match the upper bound which actually is the marginal distribution.

If " $p(z|x,\theta) = p(z|x,\theta)$ " then the inequality becomes an equality because the ratio is going to be a constant:

$$\text{Ratio} = \frac{p(x_i, z_i | \theta)}{p(z_i | x_i)} = \frac{p(x_i, z_i | \theta)}{p(x_i | z_i)} = p(x_i | \theta)$$

Where " $p(x_i | \theta)$ " is actually the marginal distribution that we want to optimize. And, if we plug this into the formula:

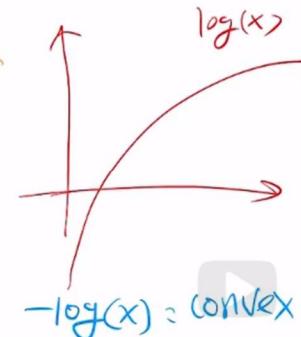
$$LB(\theta, \rho) = \sum_{i=1}^n E_{z_i \sim \rho(\cdot | x_i)} \cdot \log(p(x_i | \theta))$$

Because the marginal distribution " $p(x_i | \theta)$ " does not depend on " $z_i$ " anymore the expectation is vacuum and we can just remove it:

$$LB(\theta, \rho) = \sum_{i=1}^n E_{z_i \sim \rho(\cdot | x_i)} \cdot \log(p(x_i | \theta)) = \sum_{i=1}^n \log(p(x_i | \theta)) = L(\theta)$$

Which is then the original margin of likelihood in the statement of the problem.

$$\begin{aligned}
 \underline{L}(\theta) &= \sum_{i=1}^n \log \left( \frac{P(x_i, z_i | \theta)}{P(z_i | x_i)} \right) - \cancel{\log P(z_i | x_i)} \\
 &= \sum_{i=1}^n \log \sum_{z_i} \left( \frac{P(x_i, z_i | \theta)}{P(z_i | x_i)} \right) \cancel{P(z_i | x_i)} \\
 &= \sum_{i=1}^n \log \underbrace{E_{z_i \sim P(\cdot | x_i)}}_{\text{Marginal}} \left( \frac{P(x_i, z_i | \theta)}{P(z_i | x_i)} \right) \\
 &\geq \sum_{i=1}^n E_{z_i \sim P(\cdot | x_i)} \log \left( \frac{P(x_i, z_i | \theta)}{P(z_i | x_i)} \right) \\
 &\triangleq \underline{LB}(\theta, P)
 \end{aligned}$$

  
 $\log(x)$   
 $-\log(x) = \text{convex}$

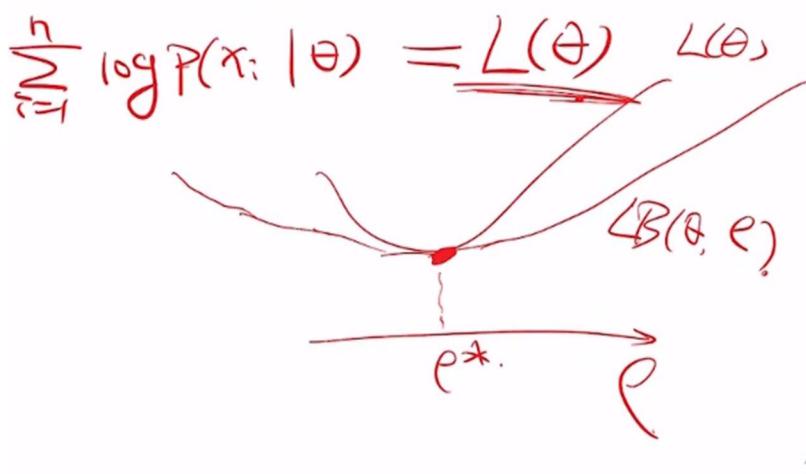
If  $P(z | x, \theta) = \underline{P}(z | x, \theta)$

$$P(x_i, z_i | \theta) = P(x_i | z_i | \theta) \quad P(x_i, z_i | \theta) \approx 1 \Rightarrow$$

If  $\hat{P}(z|x, \theta) = \underline{P}(z|x, \theta)$

$$\frac{\underline{P}(x_i, z_i | \theta)}{\hat{P}(z_i | x_i)} = \frac{\underline{P}(x_i, z_i | \theta)}{\underline{P}(z_i | x_i, \theta)} = \underline{\underline{P}(x_i | \theta)}$$

$$\begin{aligned} \Rightarrow LB(\theta, \rho^*) &= \sum_{i=1}^n \underline{E}_{z_i \sim \hat{P}(\cdot | x_i)} \log \underline{\underline{P}(x_i | \theta)} \\ &= \sum_{i=1}^n \log \underline{\underline{P}(x_i | \theta)} = L(\theta) \end{aligned}$$



Equivalencies obtained:

$$L(\theta) = \max_{\rho} LB(\theta, \rho)$$

Demonstration EM algorithm is equivalent to maximizing  $LB(\theta, \rho)$

$$L(\theta) = \max_{\theta, \rho} LB(\theta, \rho)$$

EM algorithm suggests to solve this problem in two steps:

1. Initialize “ $\theta_0$ ”.
2. Fix “ $\theta_t$ ”:

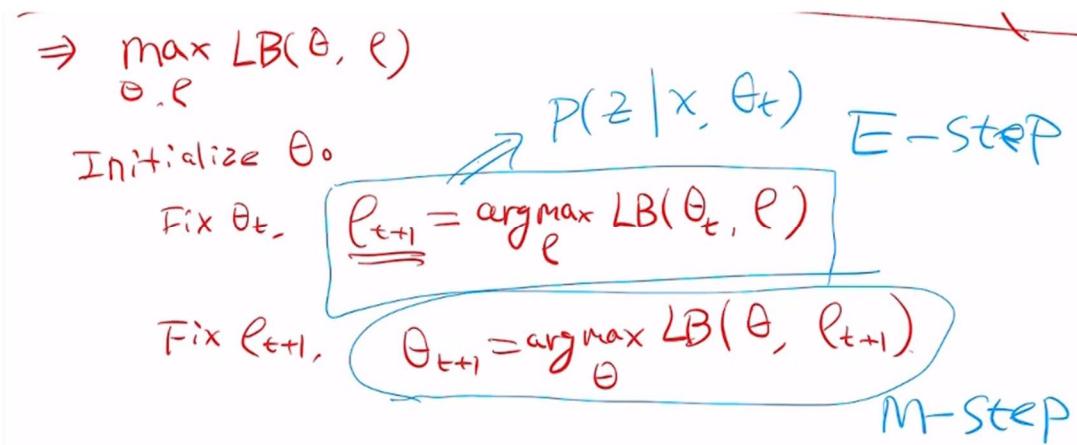
$$\rho_{t+1} = \max_{\rho} LB(\theta_t, \rho)$$

3. Fix “ $\rho_{t+1}$ ”:

$$\theta_{t+1} = \max_{\theta} LB(\theta, \rho_{t+1})$$

Step (2) is equivalent to calculating the procedural distribution in the EM algorithm (“E” step).

Step (3) is equivalent to the “M” step, which is maximizing the expected joint likelihood.



We can determine that:

$$\begin{aligned}
 LB(\Theta, \rho_{t+1}) &= \sum_{i=1}^n E_{z_i \sim p(z_i|x_i)} [\log P(x_i, z_i | \Theta) - \log p(z_i | x_i)] \\
 &= \sum_{i=1}^n E_{z_i \sim p(z_i|x_i)} [\log P(x_i, z_i | \Theta)] + \text{const.} \\
 &\quad \text{---} \\
 &\quad \text{Expected joint likelihood}
 \end{aligned}$$

This maximization procedure is equivalent to the EM procedure.

Interpretations from demonstrations:

1. EM algorithm (or procedure to optimize “ $\Theta$ ” and “ $\rho$ ” alternatively) provides a procedure that guarantees to decrease the lower bound monotonically.
2. Guarantee to converge to local optimum of the Loss function “ $L(\Theta)$ ”.
3. No guarantee to converge to global optimum because the marginal likelihood.

## Multivariate distributions

Introduction

In this lecture, we are going to introduce the concept of multivariate distributions. Later we will introduce a specific case called multivariate normal distributions which is very useful model to capture correlations between different variables.

Let us review some basic concepts of multivariate distributions.

We have seen so far univariate distributions which are distributions for a one-dimensional variable: for example the normal distribution that captures the mean and variance of a random variable called “weight”.

Multivariate distributions are really just distributions of vectors, random vectors.

In our case we have a random variable “ $x$ ” which is a vector  $\{x_1, x_2, \dots, x_d\}$  with real values (continuous vector).

$$\mathbf{x} = \begin{Bmatrix} x_1 \\ \dots \\ x_d \end{Bmatrix}$$

In order to capture the distribution of probability for this random vector we will define the density function: "probability density function".

$$p(\mathbf{x}) = p([x_1 \ \dots \ x_d])$$

Probability density functions always meet two criteria:

$$p(x) \geq 0$$

$$\int p(x) \cdot dx = 1$$

Given a multivariate distribution like the above one, we can define a mean vector and a covariance matrix.

Expectation of " $\mathbf{x}$ " under distribution " $p$ ":

$$\mu = E_p[\mathbf{x}] = \begin{Bmatrix} E_p[x_1] \\ \dots \\ E_p[x_d] \end{Bmatrix}$$

Where:

$$E_p[x_1] = \int p(x) \cdot x_1 \cdot dx$$

Covariance matrix " $\Sigma$ ":

$$\Sigma = [cov(x_i, x_j)]_{ij} = \begin{bmatrix} cov(x_1, x_1) & \dots & cov(x_1, x_d) \\ \dots & \dots & \dots \\ cov(x_d, x_1) & \dots & cov(x_d, x_d) \end{bmatrix}_{d \times d}$$

Which turns out to be a " $d \times d$ " matrix.

Where " $cov(x_i, x_j)$ ":

$$cov(x_i, x_j) = E[(x_i - \mu_i) \cdot (x_j - \mu_j)]$$

$$\Sigma = [\text{Cov}(x_i, x_j)]_{ij}$$

$\text{Cov}(x_i, x_j) = E[(x_i - \mu_i)(x_j - \mu_j)]$

Covariance matrix  
(dxd)

$$\Sigma = E[(x - \mu)(x - \mu)^T]$$

||      ——————  
⇒      [ ]

---

Normal distributions**Normal Distributions**

- A univariate distribution is called  $\mathcal{N}(\mu, \sigma^2)$  if

$$p(x) = \frac{1}{Z} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right), \quad Z = \sqrt{2\pi\sigma^2},$$

A Normal distribution is a very basic and powerful distribution.

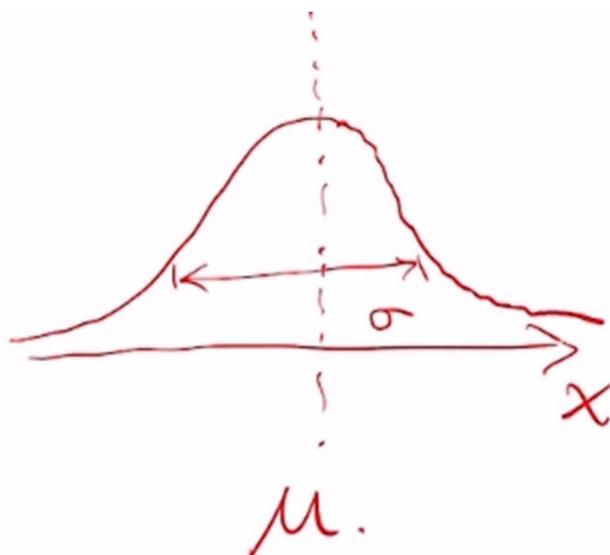
A Normal distribution has the following parameters:

- $\mu$ : mean.
- $\sigma$ : variance.

The density function is defined as:

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

Plotting this density function it gets the classical normal curve:



Where the variance defines the waist of this curve and the mean make it more peaky.

“ $\mu$ ” is also equal to the expectation of “ $x$ ” under density distribution “ $p(x)$ ”:

$$\mu = E_p[x] = \int x \cdot p(x) \cdot dx$$

“ $\sigma^2$ ” is also equal to the variance square:

$$\sigma^2 = E_p[(x - \mu)^2]$$

In practice we may observe samples from a Gaussian distribution with unknown “ $\mu$ ” and “ $\sigma$ ” and we can use the empirical formulas to fit the best “ $\mu$ ” and “ $\sigma$ ” from observations.

Given  $\{x_1, x_2, x_n\}$  observations which is drawn from a Normal distribution “ $N(\mu, \sigma^2)$ ” then the empirical values that best fit correspond to:

$$\hat{\mu} = \frac{1}{n} \cdot \sum_{i=1}^n x_i$$

$$\hat{\sigma}^2 = \frac{1}{n} \cdot \sum_{i=1}^n (x_i - \hat{\mu})^2$$

$$p(x) = \frac{1}{Z} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right), \quad Z = \sqrt{2\pi\sigma^2},$$

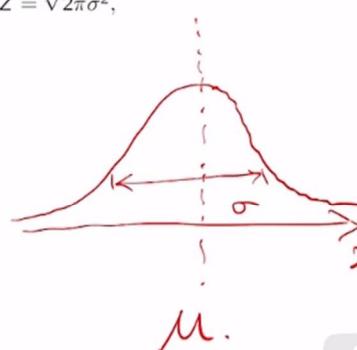
$$\mu = E_p[x] = \int x p(x) dx$$

$$\sigma^2 = E_p[(x - \mu)^2]$$

Given  $\{x_i\}_{i=1}^n \sim \mathcal{N}(\mu, \sigma^2)$

Estimate  $\mu, \sigma$  by:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i \quad \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu})^2$$

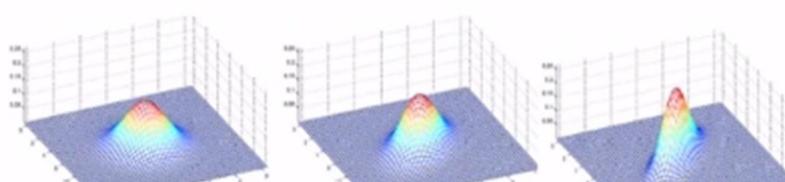


In fact, these empirical values are exactly the solution we get maximizing the likelihood of this Gaussian distributions.

### Multivariate Normal Distributions

- A  $d$ -dimensional multivariate distribution is called  $\mathcal{N}(\mu, \Sigma)$  if

$$p(\mathbf{x}) = \frac{1}{Z} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^\top \Sigma^{-1}(\mathbf{x} - \mu)\right), \quad Z = \sqrt{(2\pi)^d \det(\Sigma)}$$



- |                         |                             |                             |
|-------------------------|-----------------------------|-----------------------------|
| • $\mu = [0; 0]$        | • $\mu = [0; 0]$            | • $\mu = [0; 0]$            |
| • $\Sigma = [1 0; 0 1]$ | • $\Sigma = [1 0.5; 0.5 1]$ | • $\Sigma = [1 0.8; 0.8 1]$ |

In this case “ $\mathbf{x}$ ” is a  $d$ -dimensional random vector:

$$\mathbf{x} = \begin{Bmatrix} x_1 \\ \dots \\ x_d \end{Bmatrix}$$

Follows a multivariate Normal distribution with: "N( $\mu, \Sigma$ )" where:

$$\mu = E_p[\mathbf{x}] = \begin{Bmatrix} E_p[x_1] \\ \dots \\ E_p[x_d] \end{Bmatrix}$$

$$\Sigma = [cov(x_i, x_j)]_{ij} = \begin{bmatrix} cov(x_1, x_1) & \dots & cov(x_1, x_d) \\ \dots & \dots & \dots \\ cov(x_d, x_1) & \dots & cov(x_d, x_d) \end{bmatrix}_{d \times d}$$

Only if its density function can be written this way:

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d \cdot \det(\Sigma)}} \cdot e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \cdot \Sigma^{-1} \cdot (\mathbf{x}-\mu)}$$

Note: we need to know the determinant of " $\Sigma$ " (covariance matrix) and its inverse where we used to take the covariance itself or its inverse ("1/ $\sigma$ ") in the univariate case.

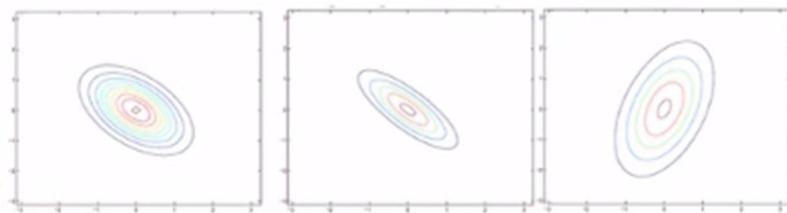
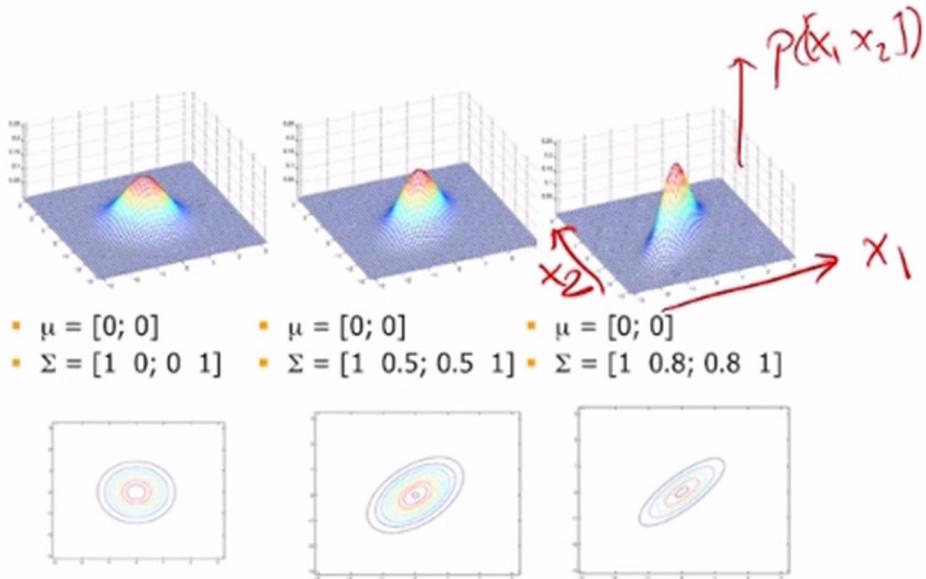
This function is a classical quadratic form that returns a number (probability) and creates a 3D ellipse in the two-dimensional case.

$$p(\mathbf{x}) = \frac{1}{Z} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) \right), \quad Z = \sqrt{(2\pi)^d \det(\Sigma)}$$

$(\mathbf{x} - \mu)^T$        $\Sigma^{-1}$        $(\mathbf{x} - \mu)$

$$\mathbf{x} = \begin{Bmatrix} x_1 \\ \vdots \\ x_d \end{Bmatrix}$$

Let us see a few number of examples for two-dimensional variables:



$\mu = [0; 0]$	$\mu = [0; 0]$	$\mu = [0; 0]$
$\Sigma = [1 \ -0.5; -0.5 \ 1]$	$\Sigma = [1 \ -0.8; -0.8 \ 1]$	$\Sigma = [3 \ 0.8; 0.8 \ 1]$

$$\Sigma = [cov(x_i, x_j)]_{ij} = \begin{bmatrix} \sigma_{11} & \dots & \sigma_{1d} \\ \dots & \dots & \dots \\ \sigma_{d1} & \dots & \sigma_{dd} \end{bmatrix}_{d \times d}$$

In our example in a two-dimensional set:

$$\Sigma = [cov(x_i, x_j)]_{ij} = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix} = [\sigma_{11}, \sigma_{12}; \sigma_{21}, \sigma_{22}]$$

The covariance quantifies the correlation between "x1" and "x2" in each independent observation "x={x1, x2}".

Remember the term below is the normalization constant that appears in the Gaussian distribution function:

$$Z = \sqrt{(2 \cdot \pi)^d \cdot \det(\Sigma)}$$

Independent Normal Distributions**Independent Normal Distributions**

- Let  $x_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ ,  $i = 1, \dots, d$ .
- Assume  $\{x_i\}$  are independent with each other ( $x_i \perp x_j$  for  $i \neq j$ ).

□ Then their concatenation  $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$  follows a multivariate normal...

How do we actually multivariate Normal distributions?

A very simple way is by putting univariate Normal distributions together drawn from a bunch of univariate different normal distributions.

We assume all “xi” are independent from each other (i.e. “xi”  $\perp$  “xj”, for “i”  $\neq$  “j”). This implies that:

$$x_i \perp x_j \leftrightarrow p(x_i, x_j) = p(x_i) \cdot p(x_j)$$

Now, if “xi” are independent form each other and we put all the “xi” in a vector:

$$x = \begin{Bmatrix} x_1 \\ \dots \\ x_d \end{Bmatrix}$$

Then the resulting “x” will be a multivariate Normal distribution.

Let us prove the mean:

$$\mu = E[x] = \begin{Bmatrix} \mu_1 \\ \dots \\ \mu_d \end{Bmatrix}$$

However, let us observe the covariance matrix takes this form:

$$\Sigma = E[(x_i - \mu_i) \cdot (x_j - \mu_j)^T] = \begin{bmatrix} \sigma_{11}^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_{dd}^2 \end{bmatrix}$$

It is a diagonal matrix as “xi” values are independent from each other there is no covariance or covariance is “0” because there is no correlation between different random variables when “i≠j”.

Marginal Distributions

## Marginal Distributions

□ Assume  $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ . We can divide the vectors into two blocks:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}, \quad \boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \quad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}.$$

□ Then the marginal distribution of  $\mathbf{x}_1$  is also Gaussian:

$$\mathbf{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}).$$

In this case, let's say we have a random vector "x" that follows a multivariate Normal distribution " $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ ".

Let us divide this vector into two subvectors/submatrices such as " $\mathbf{x}_1$ " taking the first half and " $\mathbf{x}_2$ " taking the second half of vector "x".

Similarly, we partition accordingly vector " $\boldsymbol{\mu}$ " and matrix " $\boldsymbol{\Sigma}$ ":

$$\mathbf{x} = \begin{Bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{Bmatrix} \rightarrow \boldsymbol{\mu} = \begin{Bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{Bmatrix} \rightarrow \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}$$

Let us make both " $\mathbf{x}_1$ " and " $\mathbf{x}_2$ " take real values (continuous).

Because " $\boldsymbol{\Sigma}$ " is a symmetric matrix:

$$\boldsymbol{\Sigma}_{12} = \boldsymbol{\Sigma}_{21}^T$$

It is an obvious result that each of the partitions are also Gaussian distributions:

$$\mathbf{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11})$$

$$x_1 \in \mathbb{R}^{d_1}$$

$$x_2 \in \mathbb{R}^{d_2}$$

$$d_1 + d_2 = d$$

$$\Sigma_1 \in \mathbb{R}^{d_1 \times d_1}$$

$$\Sigma_{12} \in \mathbb{R}^{d_1 \times d_2}$$

$$\Sigma_{12} = \Sigma_{21}^T$$

### Conditional Distributions

#### **Conditional Distributions**

~~~

□ Assume  $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ . We can divide the vectors into two blocks:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}, \quad \boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \quad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}.$$

□ The conditional distribution  $p(\mathbf{x}_1 | \mathbf{x}_2 = \mathbf{a})$  is also Gaussian:

$$\mathbf{x}_1 | (\mathbf{x}_2 = \mathbf{a}) \sim \mathcal{N}(\boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2})$$

where

$$\begin{aligned} \boldsymbol{\mu}_{1|2} &= \mathbb{E}[\mathbf{x}_1 | \mathbf{x}_2 = \mathbf{a}] = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{a} - \boldsymbol{\mu}_2) \\ \boldsymbol{\Sigma}_{1|2} &= \text{cov}(\mathbf{x}_1 | \mathbf{x}_2 = \mathbf{a}) = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}. \end{aligned}$$

$\boldsymbol{\Sigma}_{1|2}$  is called the Schur complement of  $\boldsymbol{\Sigma}_{22}$  in  $\boldsymbol{\Sigma}$ .

Based on the above results we can talk about conditional distributions.

For example, what is the probability “ $x_1$ ” given “ $x_2=a$ ” (“ $p(x_1|x_2=a)$ ”)?

Turns out, although less obvious, that “ $x_1$ ” conditioning on “ $x_2$ ” will also be a multivariate Normal distribution:

$$x_1|(x_2=a) \sim N(\mu_{1|2}, \Sigma_{1|2})$$

Provided that:

$$\mu_{1|2} = E[x_1|x_2=a] = \mu_1 + \Sigma_{12} \cdot \Sigma_{22}^{-1} \cdot (a - \mu_2)$$

$$\Sigma_{1|2} = cov(x_1|x_2=a) = \Sigma_{11} - \Sigma_{12} \cdot \Sigma_{22}^{-1} \cdot \Sigma_{21}$$

“ $\Sigma_{1|2}$ ” is called the **Schur complement** of the “ $\Sigma$ ” matrix. We will likely get back to the derivation of this result later on.

For now, it is important to note that the conditional distribution of a multivariate Gaussian is also a Gaussian.

### Linear transform

## Linear Transform

- Let  $z \sim \mathcal{N}(\mu, \Sigma)$ .
- Applying linear transform:

$$x = Az + b.$$

Then  $x$  is also a multivariate normal...

Basic but very useful result is that the linear transform of a multivariate Gaussian is also a multivariate Gaussian.

Assume “ $z$ ” is a random vector, which follows a multivariate Gaussian with “ $\mu$ ” and “ $\Sigma$ ”.

If we construct a new random vector “ $x$ ” (where “ $x$ ” is a vector with the same dimension or not as “ $z$ ”: “ $x_1, x_2, \dots, x_{d_1}$ ”) which is the linear transform of “ $z$ ” (where “ $z$ ” is a vector “ $\{z_1, z_2, \dots, z_d\}$ ”), then “ $x$ ” is also a multivariate Gaussian.

**Note:** “ $A$ ” and “ $b$ ” are deterministic values. “ $x$ ” does not need to have the same dimensions as “ $z$ ”.

**Linear Transform**

- Let  $\underline{z} \sim \mathcal{N}(\mu, \Sigma)$ .
- Applying linear transform:

$$\underline{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_d \end{bmatrix} \in \mathbb{R}^d$$

$$\underline{x} = \underline{A}\underline{z} + \underline{b}$$

Then  $\underline{x}$  is also a multivariate normal...

$$\underline{A} \in \mathbb{R}^{d_1 \times d}$$

$$\underline{b} \in \mathbb{R}^{d_1 \times 1}$$

$$\underline{x} = \underline{A}\underline{z} + \underline{b} \in \mathbb{R}^{d_1 \times 1}$$

Let us calculate the multivariate Gaussian resulting from this linear transform:

We know that “ $x$ ” follows a multivariate Gaussian:

$$x \sim N(\mu_x, \Sigma_x)$$

Where mean of “ $x$ ”:

$$\mu_x = E[x] = E[A \cdot z + b]$$

Using the linear property of expectation, we can move the constants outside of the expectation:

$$\mu_x = E[x] = E[A \cdot z + b] = A \cdot E[z] + b = A \cdot \mu + b$$

And where covariance of “ $x$ ”:

$$\begin{aligned} \Sigma_x &= E[(x - \mu_x) \cdot (x - \mu_x)^T] = E[(A \cdot z + b - (A \cdot \mu + b)) \cdot (A \cdot z + b - (A \cdot \mu + b))^T] \\ &= E[(A \cdot (z - \mu)) \cdot (A(z - \mu))^T] = E[(A \cdot (z - \mu)) \cdot ((z - \mu))^T \cdot A^T] \end{aligned}$$

Remember “ $A$ ” and “ $b$ ” are deterministic values and we can place them outside the expectation term:

$$\Sigma_x = E[(A \cdot (z - \mu)) \cdot ((z - \mu))^T \cdot A^T] = A \cdot E[(z - \mu) \cdot (z - \mu)^T] \cdot A^T$$

**Note:** the highlighted expression is the covariance matrix “ $\Sigma$ ” of “ $x$ ”, therefore:

$$\Sigma_x = A \cdot \Sigma \cdot A^T$$

**Linear Transform**

- Let  $\underline{z} \sim \mathcal{N}(\mu, \Sigma)$ .
- Applying linear transform:

$$\underline{x} = A\underline{z} + b$$

Then  $\underline{x}$  is also a multivariate normal...

$$\underline{x} \sim \mathcal{N}(\underline{\mu}_x, \underline{\Sigma}_x)$$

$$\begin{aligned} A &\in \mathbb{R}^{d_1 \times d} \\ b &\in \mathbb{R}^{d_1 \times 1} \end{aligned}$$

$$\underline{x} = A\underline{z} + b \in \mathbb{R}^{d_1 \times 1}$$

$$\begin{aligned} \underline{\mu}_x &= E[\underline{x}] = E[A\underline{z} + b] \\ &= A E[\underline{z}] + b = \underline{A}\underline{\mu} + b \end{aligned}$$

$$\begin{aligned} \underline{\Sigma}_x &= E[(\underline{x} - \underline{\mu}_x)(\underline{x} - \underline{\mu}_x)^T] \\ &= E[(A\underline{z} + b - \underline{A}\underline{\mu} - b)(A\underline{z} + b - \underline{A}\underline{\mu} - b)^T] \\ &= E[A(\underline{z} - \underline{\mu})(A(\underline{z} - \underline{\mu}))^T] \\ &= E[A(\underline{z} - \underline{\mu})(\underline{z} - \underline{\mu})^T A^T] \\ &= A E[(\underline{z} - \underline{\mu})(\underline{z} - \underline{\mu})^T] A^T = A \Sigma A^T \end{aligned}$$

Turns out this is a very useful result. Let's take advantage for the next example.

Multivariate Normal Distributions and PCA**Multivariate Normal and PCA**

- We can construct a probabilistic interpretation of principle component analysis (PCA) using multivariate normal distributions.

PCA stands for Principle Component Analysis. Turns out that linear transform has a very close relation to PCQ.

Sing multivariate Gaussian we can actually have a probabilistic interpretation of PCA.

Assume "z" is a random variable d-dimensional following a random Gaussian "N(0, Σ)" where "Σ" is a diagonal matrix.

$$z \sim N(0, \Sigma), \Sigma = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_d \end{bmatrix}$$

"Σ" being diagonal implies we have independent variables in "z={z<sub>1</sub>, z<sub>2</sub>, ..., z<sub>d</sub>}".

Let us assume we have a new random variable "x" obtained as:

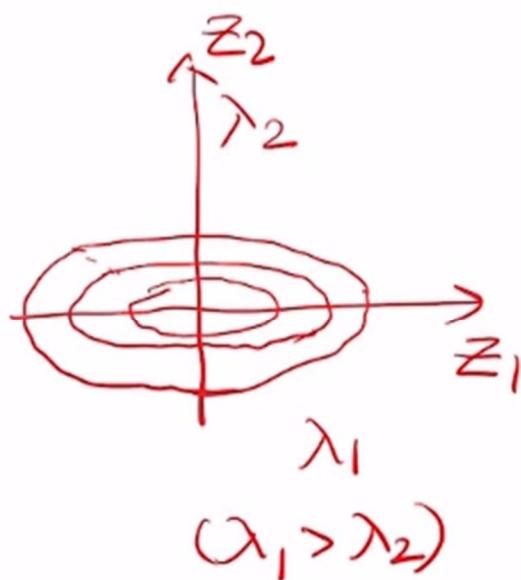
$$x = A \cdot z + b$$

We observe a bunch of samples from "x={x<sup>1</sup>, x<sup>2</sup>, ..., x<sub>d</sub>}" where each "x<sup>i</sup>" is an independent observation of "x<sup>i</sup>={x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>d</sub><sub>i</sub>}".

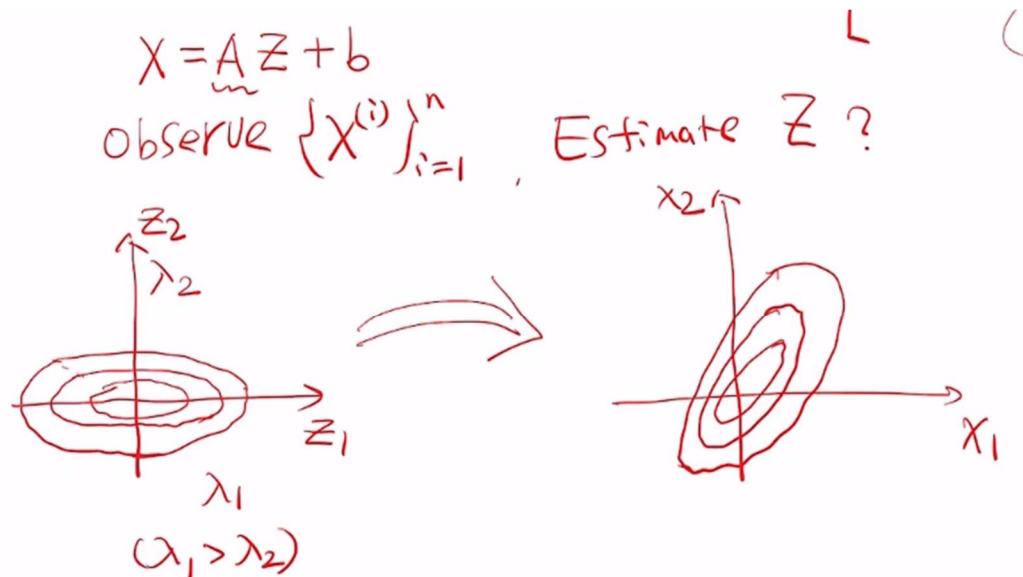
Estimate "z"?

In this case, we are assuming "z" has a diagonal covariance matrix "Σ".

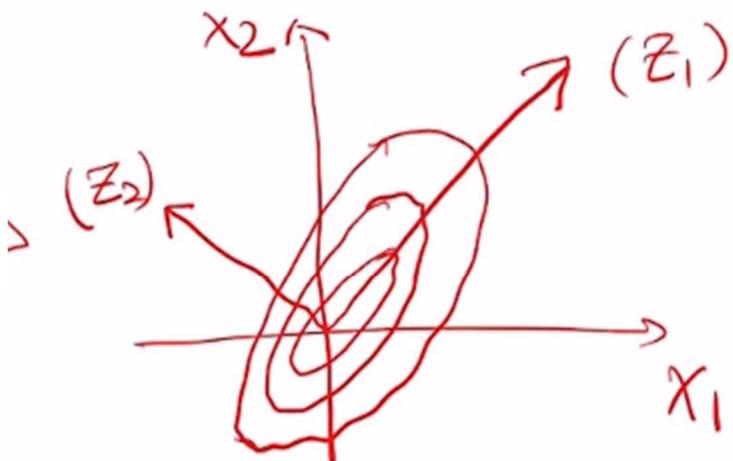
If we plot the contour of the "z" density it will look like a flat ellipse with radios "λ<sub>1</sub>" and "λ<sub>2</sub>":



Now, "x" is going to look different as "A" matrix can rotate and transform the vector in the space:



In the "x" space we will have a different shape of the contour. When "A" matrix is normal it rotates the vectors without scaling them:



This makes a connection with PCA: in PCA we observe a bunch of data which is "x" and we want to find the direction of maximum variation. We can assume that there is a class of random variables "z" that are independent from each other (covariance is "0").

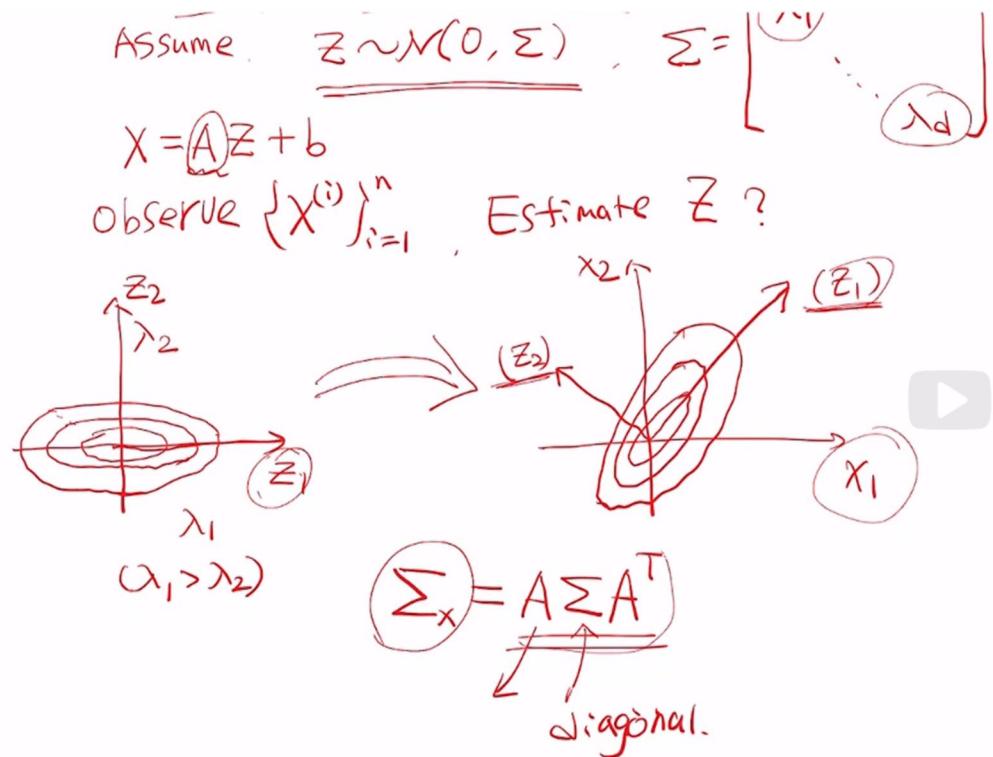
In PCA we want to identify the direction that has the larger list of variation, which corresponds to direction "z1".

This is related to the eigen decomposition of the covariance matrix "z":

$$\Sigma_x = A \cdot \Sigma \cdot A^T$$

Where " $\Sigma$ " is a diagonal matrix and "A" is a normal matrix (unit matrix that only rotates: formed with cosines and sines of certain angle).

With the above, then " $\Sigma_x$ " is the eigen decomposition of the covariance matrix of "x".



**In conclusion**, when we observe a bunch of data "x" we can actually calculate its covariance matrix " $\Sigma_x$ ". Then, perform the eigen decomposition that allows to estimate both the "A" matrix (which corresponds to rotating the directions in "z") as well as the original variance of each dimension of "z" (that corresponds to the eigen values of " $\Sigma$ " that must be place in a diagonal matrix). "A" corresponds to the eigenvectors.

### Multivariate Normal with MLE

#### Maximum Likelihood Estimation

- Given an observation  $\{x_i\}_{i=1}^n$  drawn independently from  $\mathcal{N}(\mu, \Sigma)$ .
- We can show that the maximum likelihood estimation of  $\mu$  and  $\Sigma$  equals the empirical mean and covariance.

For multivariate Gaussian we can use MLE to demonstrate that the value for the mean vector " $\lambda$ " and the covariance matrix " $\Sigma$ " is actually equal to the empirical values already introduced.

The goal of the MLE is to maximize the mean vector " $\lambda$ " and the covariance matrix " $\Sigma$ :

$$\max_{\mu, \Sigma} \sum_{i=1}^n \log p(x_i | \mu, \Sigma) = \sum_{i=1}^n \log \left( \frac{1}{\sqrt{(2\pi)^d \cdot \det(\Sigma)}} \cdot e^{-\frac{1}{2}(x_i - \mu)^T \Sigma^{-1} (x_i - \mu)} \right)$$

Making use of log properties:

$$\begin{aligned} \max_{\mu, \Sigma} \sum_{i=1}^n \log p(x_i | \mu, \Sigma) \\ = -n \cdot \log(2 \cdot \pi)^{\frac{d}{2}} - \frac{n}{2} \cdot \log(\det(\Sigma)) - \frac{1}{2} \cdot \sum_{i=1}^n ((x_i - \mu)^T \cdot \Sigma^{-1} \cdot (x_i - \mu)) \end{aligned}$$

The highlighted part becomes the likelihood function “ $L(\theta)$ ” as the remaining part is a constant and derivation of a constant does not affect the maximum or minimum value this function can take.

**Note:** sum over all values in a part that does not have “ $x_i$ ” dependent value is equivalent to removing the summation and placing “ $n$ ” times.

$$L(\theta) = -\frac{n}{2} \cdot \log(\det(\Sigma)) - \frac{1}{2} \cdot \sum_{i=1}^n ((x_i - \mu)^T \cdot \Sigma^{-1} \cdot (x_i - \mu))$$

When it comes to “ $\mu$ ”, in order to optimize the likelihood function “ $L(\theta)$ ” we only need to care about the second term as the first term does not depend on “ $\mu$ ” and it is a constant.

For “ $\mu$ ”:

$$L(\theta) = -0 - \frac{1}{2} \cdot \sum_{i=1}^n ((x_i - \mu)^T \cdot \Sigma^{-1} \cdot (x_i - \mu))$$

It is a quartic form equivalent:

$$L(\theta) = -\frac{1}{2} \cdot \left( n \cdot \mu^T \cdot \Sigma^{-1} \cdot \mu - 2 \cdot \left( \sum_{i=1}^n x_i \right) \cdot \Sigma^{-1} \cdot \mu + const \right)$$

Taking the derivative and making it “0” we can find out that our optimal “ $\mu$ ”:

$$\mu = (n \cdot \Sigma^{-1})^{-1} \cdot \left( \Sigma^{-1} \cdot \left( \sum_{i=1}^n x_i \right) \right)$$

Because “ $\Sigma$ ” cancels “ $\Sigma^{-1}$ ” as “ $\Sigma \cdot \Sigma^{-1} = I$ ”:

$$\mu = \frac{1}{n} \cdot \sum_{i=1}^n x_i$$

$$\begin{aligned}
 & \mu \cdot \Sigma^{-1} = \dots \\
 &= \sum_{i=1}^n \log \left( \frac{1}{(2\pi)^{\frac{d}{2}} \det(\Sigma)^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x_i - \mu)^T \Sigma^{-1} (x_i - \mu) \right) \right) \\
 &= -n \log((2\pi)^{\frac{d}{2}}) - \underbrace{\frac{1}{2} \log \det(\Sigma)}_{\text{const}} - \frac{1}{2} \sum_{i=1}^n (x_i - \mu)^T \Sigma^{-1} (x_i - \mu) \\
 \text{For } \mu: & -\frac{1}{2} \left( n \mu^T \Sigma^{-1} \mu - 2 \left( \sum_{i=1}^n x_i \right)^T \Sigma^{-1} \mu + \text{const} \right) \\
 \Rightarrow \mu &= (n \Sigma^{-1})^{-1} \left( \Sigma^{-1} \left( \sum_{i=1}^n x_i \right) \right) = \frac{1}{n} \sum_{i=1}^n x_i
 \end{aligned}$$

For " $\Sigma$ ", again we can take the gradient and calculate the points with gradient "0". It gets easier if we optimize " $Q = \Sigma^{-1}$ ".

Let " $Q = \Sigma^{-1}$ ", remember " $\det(A) = 1/\det(A^{-1}) \neq 0$ ". Let us replace " $\mu$ " with the optimal value obtained in the above expression and take the inverse of " $\det(Q)$ " as a negative sign outside the log function making the term now positive.

$$L(\theta) = \frac{n}{2} \cdot \log(\det(Q)) - \frac{1}{2} \cdot \sum_{i=1}^n ((x_i - \hat{\mu})^T \cdot Q \cdot (x_i - \hat{\mu}))$$

Taking the derivative of the likelihood function with respect to " $Q$ " (as " $\mu$ " is not a variable anymore  $\rightarrow \theta = Q$ ) and calculating the gradient:

$$\text{grad}_Q (L(Q))_{dxd} = \frac{n}{2} \cdot Q^{-1} - \frac{1}{2} \cdot \sum_{i=1}^n (x_i - \mu) \cdot (x_i - \mu)^T$$

**Note:** here there is an assumption with respect to the first term that comes from matrix calculus. We can check why this is reasonable assuming the case when " $Q$ " is a "1x1" matrix:

1. Bearing in mind derivative of a logarithmic function:

$$\frac{d}{dx} (\log(x)) = \frac{1}{x}$$

2. Assuming " $Q$ " is not a matrix but a number (1x1 dimension):

$$\det(Q) = Q$$

Then:

$$\text{grad}_Q (\log(\det(Q))) = \text{grad}_Q (\log(Q)) = \frac{d}{dQ} (\log(Q)) = \frac{1}{Q} = Q^{-1}$$

The **second part** of the gradient is the empirical covariance, which multiples a “dx1” vector with a “1xd” matrix resulting in a “dxd” matrix.

$$\text{const} \left[ -\frac{n}{2} \log \det(\Sigma) - \frac{1}{2} \sum_{i=1}^n (x_i - \mu)^T \Sigma^{-1} (x_i - \mu) \right]$$

$$\text{For } L(Q) = \frac{n}{2} \log \det(Q) - \frac{1}{2} \sum_{i=1}^n (x_i - \hat{\mu})^T Q (x_i - \hat{\mu})$$

$$\frac{\nabla_Q L(Q)}{d \times d} = \frac{n}{2} Q^{-1} - \frac{1}{2} \sum_{i=1}^n (x_i - \mu) (x_i - \mu)^T$$

$$\underline{Q = \Sigma^{-1}}$$

If  $Q$  is a number  
 $\nabla (\log \det Q) = \nabla (\log Q) = Q^{-1}$

Finally, making the “grad( $L(Q)$ ) = 0” we can isolate and obtain:

$$Q^{-1} = \frac{1}{n} \cdot \sum_{i=1}^n (x_i - \mu) \cdot (x_i - \mu)^T$$

This is exactly the empirical covariance and it is exactly the “ $\Sigma$ ” matrix:

$$\Sigma = Q^{-1} = \frac{1}{n} \cdot \sum_{i=1}^n (x_i - \mu) \cdot (x_i - \mu)^T$$

Multivariate Normal distribution for graphical models**Multivariate Normal Distributions**

- Multivariate distribution  $\mathcal{N}(\mu, \Sigma)$ :

$$p(\mathbf{x}) = \frac{1}{Z} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu)^\top \Sigma^{-1} (\mathbf{x} - \mu) \right), \quad Z = \sqrt{(2\pi)^d \det(\Sigma)}$$

- **Moment parameters:**

$$\text{Mean: } \mu = \mathbb{E}[\mathbf{X}], \quad \text{Variance: } \Sigma = \mathbb{E}[(\mathbf{X} - \mu)(\mathbf{X} - \mu)^\top].$$

- If we divide the vectors into two blocks:

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{bmatrix}, \quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}.$$

- Then the marginal distribution of  $\mathbf{x}_1$  is also Gaussian:

$$\mathbf{x}_1 \sim \mathcal{N}(\mu_1, \Sigma_{11}).$$

- Correlation:  $\Sigma_{12} = 0$  iff  $\mathbf{x}_1 \perp \mathbf{x}_2$ .

Let us recap some of the previously seen concepts.

We have a multinormal distribution if the probability density takes this form:

$$p(x) = \frac{1}{\sqrt{(2\pi)^d \cdot \det(\Sigma)}} \cdot e^{-\frac{1}{2} (x - \mu)^\top \Sigma^{-1} (x - \mu)}$$

Which is a quadratic form and where:

- “ $\mu$ ”: mean vector.

$$\mu = E_p[x] = \begin{Bmatrix} E_p[x_1] \\ \dots \\ E_p[x_d] \end{Bmatrix}$$

- “ $\Sigma$ ” is the covariance matrix.

$$\Sigma = [cov(x_i, x_j)]_{ij} = \begin{bmatrix} cov(x_1, x_1) & \dots & cov(x_1, x_d) \\ \dots & \dots & \dots \\ cov(x_d, x_1) & \dots & cov(x_d, x_d) \end{bmatrix}_{d \times d}$$

Where:

$$\text{cov}(x_i, x_j) = E[(x_i - \mu_i) \cdot (x_j - \mu_j)]$$

Variance:  $\Sigma = \mathbb{E}[(\underline{\mathbf{X}} - \underline{\mu})(\underline{\mathbf{X}} - \underline{\mu})^\top]$

$\Sigma = \begin{bmatrix} & \\ & \end{bmatrix}_{d \times d}$

no blocks:

We also mentioned that the partition of a vector into blocks turns into several distributions, which are Gaussian distributions too.

Finally, if covariance " $\Sigma_{12} = 0$ ", it implies that " $x_1$ " and " $x_2$ " are independent random variables with no correlation (i.e. " $x_i \perp x_j$ ", for " $i \neq j$ "). The overall covariance matrix " $\Sigma$ " would look like this:

$$\Sigma = \begin{bmatrix} \Sigma_{11} & 0 \\ 0 & \Sigma_{22} \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \Sigma_{11} & 0 \\ 0 & \Sigma_{22} \end{bmatrix}$$

Natural form**Natural Form**

□ Standard form:

$$p(\mathbf{x}) = \frac{1}{Z} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$

□ Natural (or Information) form:

$$p(\mathbf{x}) = \frac{1}{C} \exp\left(-\frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{b}^\top \mathbf{x}\right).$$

□ **Natural parameters:**

$$\mathbf{Q} = \boldsymbol{\Sigma}^{-1}, \quad \mathbf{b} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$$

$C = \exp(-\frac{1}{2}\mathbf{b}^\top \mathbf{Q}^{-1}\mathbf{b})/Z$  is the normalization constant.

In this lecture, we will see further properties and important results about multivariate normal distributions.

Another way to rewrite the same density function is something called the natural form: instead of writing the density function using “ $\mu$ ” and “ $\Sigma$ ” we can write it using a standard quadratic form which looks like this:

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d \cdot \det(\Sigma)}} \cdot e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})} \rightarrow p(\mathbf{x}) = \frac{1}{C} \cdot e^{-\frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{b}^\top \mathbf{x}}$$

Where:

$$\mathbf{Q} = \boldsymbol{\Sigma}^{-1} \text{ (inverse covariance matrix, precision matrix)}$$

$$\mathbf{b} = \boldsymbol{\Sigma}^{-1} \cdot \boldsymbol{\mu}$$

All the constant terms (where no “ $\mathbf{x}$ ” vectors appear) go into a “ $C$ ” formula, which is the normalization constant:

$$C = \frac{1}{\sqrt{(2\pi)^d \cdot \det(\Sigma)}} \cdot e^{-\frac{1}{2}\mathbf{b}^\top \mathbf{Q}^{-1}\mathbf{b}}$$

**Note:** many times, we rewrite the “ $p(\mathbf{x})$ ” formula with a proportional notation (“ $\propto$ ”) avoiding this way “ $C$ ” constant to appear in the formulas:

$$p(x) \propto e^{-\frac{1}{2}x^T Q x + b^T x}$$

□ Standard form:

$$p(x) = \frac{1}{Z} \exp \left( -\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

□ Natural (or Information) form:

$$= \frac{1}{2} x^T \underline{\Sigma^{-1} x} - \underline{\mu^T \Sigma^{-1} x} + \underline{\mu^T \Sigma^{-1} \mu}$$

$$p(x) = \frac{1}{C} \exp \left( -\frac{1}{2} x^T Q x + b^T x \right).$$

$$P(x) \propto \exp(-\frac{1}{2} x^T Q x + b^T x)$$

□ Natural parameters:

$$Q = \Sigma^{-1},$$

$$b = \Sigma^{-1} \mu$$

$C = \exp(-\frac{1}{2} b^T Q^{-1} b) / Z$  is the normalization constant.

Q: inverse covariance matrix  
precision matrix

Often "Q" and "b" are called the **natural parameters** in contrast with the **moment parameters** (covariance matrix " $\Sigma$ " and mean vector " $\mu$ ").

Conditional Distribution via Natural form**Conditional Distributions via Natural Form**

- The natural form makes it easy to derive conditional distributions.
- For  $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ ,

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \quad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}.$$

$$\mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} \\ \mathbf{Q}_{21} & \mathbf{Q}_{22} \end{bmatrix}.$$

- Then the conditional distribution is

$$\mathbf{x}_1 \mid \mathbf{x}_2 \sim \mathcal{N}(\mathbf{b}_1 - \mathbf{Q}_{12}\mathbf{x}_2, \mathbf{Q}_{11})$$

We can then convert back to moment parameters following matrix identities:

$$\boldsymbol{\Sigma}_{1|2} = \mathbf{Q}_{11}^{-1} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21},$$

$$\boldsymbol{\mu}_{1|2} = \mathbf{Q}_{11}^{-1}(\mathbf{b}_1 - \mathbf{Q}_{12}\mathbf{x}_2) = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2)$$

The natural form turns out to be useful in order to derive and handle the conditional distributions related to the Gaussian distribution.

Let us use a random variable “ $\mathbf{x}=\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_d\}$ ” that we partition into two blocks “ $\mathbf{x}=\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}=\{\mathbf{x}_1, \mathbf{x}_2\}$ ”:

$$\mathbf{x} = \begin{Bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{Bmatrix} \rightarrow \boldsymbol{\mu} = \begin{Bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{Bmatrix} \rightarrow \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}$$

Now, correspondingly we can partition the natural parameters:

$$\mathbf{x} = \begin{Bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{Bmatrix} \rightarrow \mathbf{b} = \begin{Bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{Bmatrix} \rightarrow \mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} \\ \mathbf{Q}_{21} & \mathbf{Q}_{22} \end{bmatrix}$$

**Note:** “ $\mathbf{Q}=\boldsymbol{\Sigma}^{-1}$ ”.

Remember the resulting marginal distribution of “ $\mathbf{x}_1$ ”, for example:

$$\mathbf{x}_1 \sim N(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11})$$

Previously with moment parameters the conditional distribution of “ $x_1$ ” given a fixed value “ $x_2 = a$ ” could take this form:

$$x_1 | (x_2 = a) \sim N(\mu_{1|2}, \Sigma_{1|2})$$

In this case, we can show that the conditional distribution of “ $x_1$ ” given a fixed value “ $x_2 = a$ ”:

$$x_1 | (x_2 = a) \sim N(b_1 - Q_{12} \cdot a, Q_{11})$$

**Note:** the covariance matrix of “ $x_1$ ” only depends in a submatrix of “Q”. The “b” vector of “ $x_1$ ” will change (its mean).

**Note 2:** the covariance matrix of “ $x_1$ ” when conditioning on “ $x_2$ ” (“ $\text{cov}(x_1 | x_2)$ ”) is actually the covariance of “ $x_1$ ” when conditioning **nothing** (“ $\text{cov}(x_1)$ ”).

**Note 3:** from matrix calculus, we can prove the following results when “ $x_1$ ” conditioning on “ $x_2=a$ ”:

$$\Sigma_{1|2} = Q_{11}^{-1} = \Sigma_{11} - \Sigma_{12} \cdot \Sigma_{22}^{-1} \cdot \Sigma_{21}$$

When “ $x_2=a$ ” also:

$$\mu_{1|2} = Q_{11}^{-1} \cdot (b_1 - Q_{12} \cdot a) = \mu_1 + \Sigma_{12} \cdot \Sigma_{22}^{-1} \cdot (a - \mu_2)$$

Summarizing, when using “ $\mu$ ” and “ $\Sigma$ ” the formulas are complicated but when using “ $b$ ” and “ $Q$ ” the formula is much easier.

### Proving the conditional distribution via natural form

Taking into account:

$$x = \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} \rightarrow b = \begin{Bmatrix} b_1 \\ b_2 \end{Bmatrix} \rightarrow Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}$$

Then, the density probability can be written as follows:

$$p(x) = p(x_1, x_2) \propto e^{-\frac{1}{2}x^T \cdot Q \cdot x + b^T \cdot x} \propto e^{-\frac{1}{2}(Q_{11} \cdot (x_1)^2 + 2 \cdot Q_{12} \cdot x_1 \cdot x_2 + Q_{22} \cdot (x_2)^2 + \dots + b_1 \cdot x_1 + b_2 \cdot x_2)}$$

If we want to derive the conditional probability “ $p(x_1 | x_2)$ ”, then “ $x_2=a$ ” will be a constant and we will just need to collect all the terms related to “ $x_1$ ” and drop the terms related to “ $x_2$ ” (like we can see above) and use again proportional notation for the conditional probability.

$$p(x_1 | x_2) \propto e^{-\frac{1}{2}Q_{11} \cdot (x_1)^2 - Q_{12} \cdot x_1 \cdot x_2 + b_1 \cdot x_1} = e^{-\frac{1}{2}Q_{11} \cdot (x_1)^2 + (b_1 - Q_{12} \cdot x_2) \cdot x_1}$$

This natural form is equivalent to a normal distribution like this:

$$p(x) \propto e^{-\frac{1}{2}x^T \cdot Q \cdot x + b^T \cdot x}; p(x_1 | x_2) \propto e^{-\frac{1}{2}Q_{11} \cdot (x_1)^2 + (b_1 - Q_{12} \cdot x_2) \cdot x_1}$$

Comparing both formulas, we see that:

$$b \text{ (left side)} = (b_1 - Q_{12} \cdot x_2) \text{ (right side)}$$

$$Q \text{ (left side)} = Q_{11} \text{ (right side)}$$

$$\mu_{1|2} = E[x_1 | x_2]$$

$$\Sigma_{1|2} = Cov(x_1 | x_2)$$

Proof:  $P(x_1 | x_2) \propto \exp(-\frac{1}{2}x^T Q x + b^T x)$   $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$$\propto \exp(-\frac{1}{2}(q_{11}x_1^2 + 2q_{12}x_1x_2 + q_{22}x_2^2) - b_1x_1 - b_2x_2)$$

$$P(x_1 | x_2) \propto \exp(-\frac{1}{2}(q_{11}x_1^2 + (b_1 - q_{12}x_2)x_1))$$

$$\sim \mathcal{N}(b_1 - q_{12}x_2, q_{11})$$

### Independence and conditional independence

Previously, we have shown that natural forms allows us to derive conditional distributions very conveniently.

Now we can find out why this is useful.

### Independence and Conditional Independence

□ Covariance matrix  $\Sigma = [\sigma_{ij}]_{ij}$  measures **(marginal) independence**:

$$\sigma_{ij} = 0 \quad x_i \perp x_j.$$

□ Precision matrix  $Q = [q_{ij}]_{ij}$  measures **conditional independence**:

$$q_{ij} = 0 \quad x_i \perp x_j \mid x_{-ij}.$$

The covariance matrix and the precision matrix are capturing different information about the dependency between the different values of the random variables: the **marginal independence**.

Looking at the covariance matrix “ $\Sigma = [\sigma_{ij}]_{ij} = \text{cov}(x_i, x_j)$ ”:

If  $\text{cov}(x_i, x_j) = 0$  then  $x_i \perp x_j$  ( $x_i$  and  $x_j$  independents – not correlated)

$$x_i \perp x_j \Leftrightarrow p(x_i, x_j) = p(x_i) \cdot p(x_j) \text{ (marginal independence)}$$

Turns out that the precision matrix captures a very different thing: the **conditional independence**.

Looking at the precision matrix “ $Q=[q_{ij}]_{ij}$ ”:

If  $q_{ij} = 0$  then  $x_i \perp x_j \mid x_{-ij}$  ( $x_i$  and  $x_j$  are independent conditioning other variables)

Where “ $x_{-ij}$ ” represents all other variables except “ $i$ ” and “ $j$ ”.

**Independence and Conditional Independence**

- Covariance matrix  $\Sigma = [\sigma_{ij}]_{ij}$  measures (marginal) independence:  
 $\sigma_{ij} = 0 \quad x_i \perp x_j$
- Precision matrix  $Q = [q_{ij}]_{ij}$  measures conditional independence:  
 $q_{ij} = 0 \quad x_i \perp x_j \mid x_{-ij}$

$x_i \perp x_j \Leftrightarrow P(x_i, x_j) = P(x_i)P(x_j)$

$\sigma_{ij} = \text{cov}(x_i, x_j) = 0$

$X_{7;j} = X_{[d]} / \{i, j\}$



This conditional independence implies:

$$x_i \perp x_j \mid x_{-ij} \Leftrightarrow p(x_i, x_j \mid x_{-ij}) = p(x_i \mid x_{-ij}) \cdot p(x_j \mid x_{-ij})$$

### Example

We have three random variables: “ $x=\{x_1, x_2, x_3\}$ ”

“ $x_1$ ”=if it rains today.

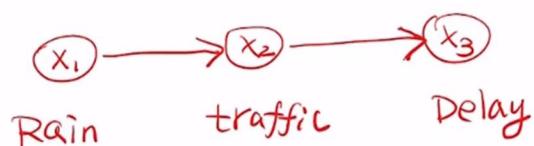
“ $x_2$ ”=traffic conditions.

“ $x_3$ ”=delay getting to work.

The dependency of these three variables follows this logic: if it rains, the traffic conditions will get worse. If there is heavy traffic, it will cause a delay.

$$x_i \perp x_j \Leftrightarrow P(x_i, x_j) = P(x_i)P(x_j) \quad \checkmark$$

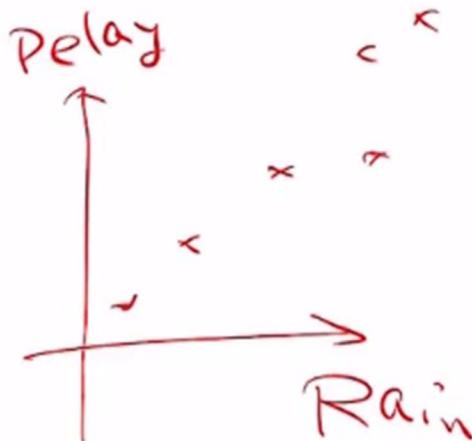
$$x_i \perp x_j \mid x_{-ij} \Leftrightarrow P(x_i, x_j \mid x_{-ij}) = P(x_i \mid x_{-ij})P(x_j \mid x_{-ij})$$



Some of the days the rain is heavy. There are days with no rain and we have light traffic and no delay.

Let us identify the covariance of " $x_1$ " and " $x_3$ ":

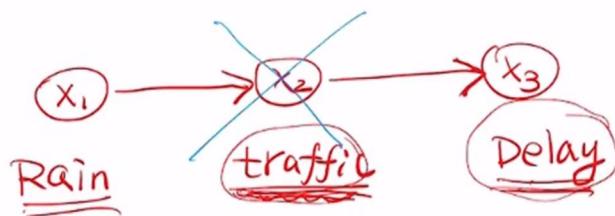
$$\sigma_{13} = \text{cov}(x_1, x_3) = \text{positive correlation} \neq 0$$



On the other hand, if we measure the conditional covariance given the traffic is fixed then the rain and delay may not be dependent.

$$q_{13} = \text{cov}(x_1, x_3 | x_2 = \text{heavy traffic}) \neq 0$$

$$q_{13} = \text{cov}(x_1, x_3 | x_2 = \text{light traffic}) = 0$$



$$\sigma_{13} = \text{cov}(x_1, x_3)$$

$$q_{13} = \text{cov}(x_1, x_3 | x_2) = 0$$

If the traffic is fixed, then the delay may not be actually relevant to rain because the traffic is the only factor that decides the delay.

This is because somehow, we are conditioning on " $x_2$ ", and we are blocking the dependency structure between " $x_1$ " and " $x_2$ ".

We need to understand the difference between marginal and conditional independence.

In order to summarize:

$x_1 \perp x_3 \mid x_2$  does not imply  $x_1 \perp x_3$

$x_1 \perp x_3$  does not imply  $x_1 \perp x_3 \mid x_2$

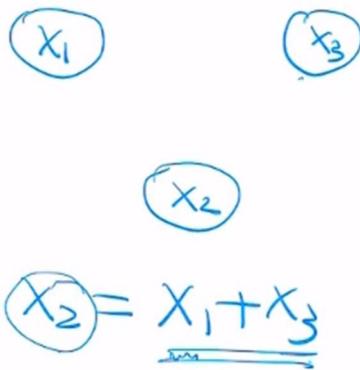
In other words, conditional independence does not imply marginal independence and viceversa.  
Let us see this with another example.

*Another example*

We have three random variables: "x={x<sub>1</sub>, x<sub>2</sub>, x<sub>3</sub>}"

Where:

$$x_2 = x_1 + x_3$$



"x<sub>1</sub>"=random number generated independently.

"x<sub>2</sub>"= random number generated independently.

"x<sub>3</sub>"=sum of random variables "x<sub>1</sub>" and "x<sub>2</sub>".

Then:

$$x_1 \perp x_2$$

However, with this setup if we know "x<sub>2</sub>" we can infer "x<sub>1</sub>" directly from "x<sub>3</sub>".

$$x_3 = x_2 + x_1$$

This actually has a negative correlation and gives us an example that marginal independence does not imply conditional independence.

Proof position matrix reflects conditional independence

We want to proof:

$$\text{If } q_{ij} = 0 \text{ then } x_i \perp x_j \mid x_{-ij}$$

Let us assume we have 3 random variables: "x={x<sub>1</sub>, x<sub>2</sub>, x<sub>3</sub>}" and they follow a multivariate normal distribution in the Natural form:

$$p(x_1, x_2, x_3) \sim N(\mu, Q)$$

We want to show that "x<sub>1</sub>" and "x<sub>2</sub>" are independent given "x<sub>3</sub>" (see previously formula for conditional distribution in the Natural form):

$$p(x_1, x_2 | x_3) \sim N(b_{1:2} - Q_{1:2,3} \cdot x_3, Q_{1:2,1:2})$$

Where we take corresponding rows/columns for variables "x<sub>1</sub>" and "x<sub>2</sub>" and a fixed value "x<sub>3</sub>".

$Q_{i:j,k:l}$  means submatrix with row i" and j", columns "k" and "l"

And where:

$$Q = \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{bmatrix}$$

Remember:

$$Q = \Sigma^{-1}$$

Then:

$$(cov(x_1, x_2 | x_3))^{-1} = Q_{1:2,1:2} = \tilde{Q}$$

If we apply this, and rewrite it in its natural form:

$$p(x_1, x_2 | x_3) \propto e^{-\frac{1}{2}(x_1 \cdot q_{11} \cdot x_1 + 2 \cdot x_1 \cdot q_{12} \cdot x_2 + q_{22} \cdot (x_2)^2 + \tilde{b}_1 \cdot x_1 + \tilde{b}_2 \cdot x_2)}$$

Bear in mind now " $\tilde{b}$ ":

$$\tilde{b} = b_{1:2} - Q_{1:2,3} \cdot x_3, \tilde{Q} = Q_{1:2,1:2}$$

Proof:  $p([x_1, x_2] | x_3) \sim \mathcal{N}(\tilde{b}, \tilde{Q})$

$$p([x_1, x_2] | x_3) \sim \mathcal{N}\left(\frac{\tilde{b}}{b}, \tilde{Q}\right)$$

$$\tilde{Q} = \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{bmatrix}$$

$$(cov([x_1, x_2] | x_3))^{-1} = Q_{1:2, 1:2}$$

$$p([x_1, x_2] | x_3) \propto \exp\left(-\frac{1}{2}(x_1 q_{11} x_1 + 2x_1 q_{12} x_2 + q_{22} (x_2)^2)\right)$$

In case "q<sub>12</sub>=0=q<sub>21</sub>" when "x<sub>1</sub>" and "x<sub>2</sub>" are independent given "x<sub>3</sub>":

$$p(x_1, x_2 | x_3) \propto e^{-\frac{1}{2}(x_1 \cdot q_{11} \cdot x_1 + q_{22} \cdot (x_2)^2 + \tilde{b}_1 \cdot x_1 + \tilde{b}_2 \cdot x_2)} = e^{-\frac{1}{2}(x_1 \cdot q_{11} \cdot x_1 + \tilde{b}_1 \cdot x_1)} \cdot e^{\frac{1}{2}(x_2 \cdot q_{22} \cdot x_2 + \tilde{b}_2 \cdot x_2)}$$

There is a term that only depends on "x<sub>1</sub>" and another that depends on "x<sub>2</sub>". Of course, they both will depend on "x<sub>3</sub>" which is a fixed value.

We can actually further show that the **first term** is exactly the conditional distribution and the **second term** is exactly the conditional distribution again:

$$p(x_1, x_2 | x_3) \propto p(x_1 | x_3) \cdot p(x_2 | x_3)$$

This is actually equivalent to the definition we did earlier when "x<sub>1</sub>" and "x<sub>2</sub>" are independent given "x<sub>3</sub>" (assuming "q<sub>12</sub>=0"):

$$x_i \perp x_j | x_{\neg ij} \leftrightarrow p(x_i, x_j | x_{\neg ij}) = p(x_i | x_{\neg ij}) \cdot p(x_j | x_{\neg ij})$$

$$Q = \begin{array}{|c c c|} \hline & q_{11} & q_{12} \\ \hline q_{21} & q_{21} & q_{22} \\ \hline q_{31} & q_{31} & q_{32} \\ \hline & q_{13} & q_{23} \\ \hline \end{array}$$

$$(Cov([x_1, x_2] | x_3))^{-1} = Q_{1:2, 1:2}$$

$$\underline{P([x_1, x_2] | x_3)} \propto \exp(-\frac{1}{2}(x_1 q_{11} x_1 + 2x_1 q_{12} x_2 + q_{22} x_2^2) + \tilde{b}_1 x_1 + \tilde{b}_2 x_2)$$

If  $q_{12}=0$

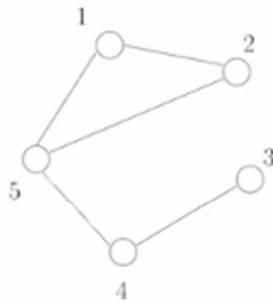
$$\propto \exp(-\frac{1}{2}x_1 q_{11} x_1 + \tilde{b}_1 x_1) \cdot \exp(-\frac{1}{2}x_2 q_{22} x_2 + \tilde{b}_2 x_2)$$

$$\propto \underline{P(x_1 | x_3)} \cdot \underline{P(x_2 | x_3)}$$

$$\Rightarrow x_1 \perp x_2 | x_3$$

Graphical models**Graphical Models**

- Gaussian graphical models, or Gaussian Markov random fields (MRF):
  - $\mathcal{N}(\mu, \Sigma)$  with sparse precision matrix  $Q = \Sigma^{-1}$ .
  - Use graph to represent the dependency structure of random variables.

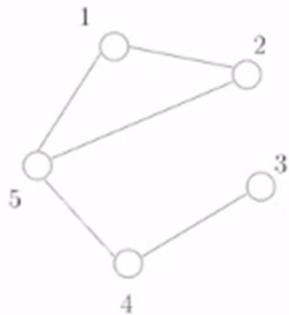


|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 1 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 0 | 1 | 0 | 1 |

- We can read Markov (conditional independent) properties from the graph.

We are ready talking about graphical models. In the case of Gaussian random fields (or Gaussian graphical models) this is simply a multivariate Gaussian distribution whose inverse covariance matrix is a sparse matrix.

What does sparse means: we are assuming we have a matrix which has a bunch of zero elements



|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 1 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 0 | 1 | 0 | 1 |

$$= Q.$$

If a precision/position matrix has zero elements then this matrix is called “Gaussian Markov”, “random field” or “Gaussian graphical model”.

This means we can represent the dependency structure, conditional dependency structure or the random variables using a graph.

It is using a graph to represent the dependency structure inside a random variable.

In this example, we have five random variables, and then we can represent each random variable using a node in a graph.

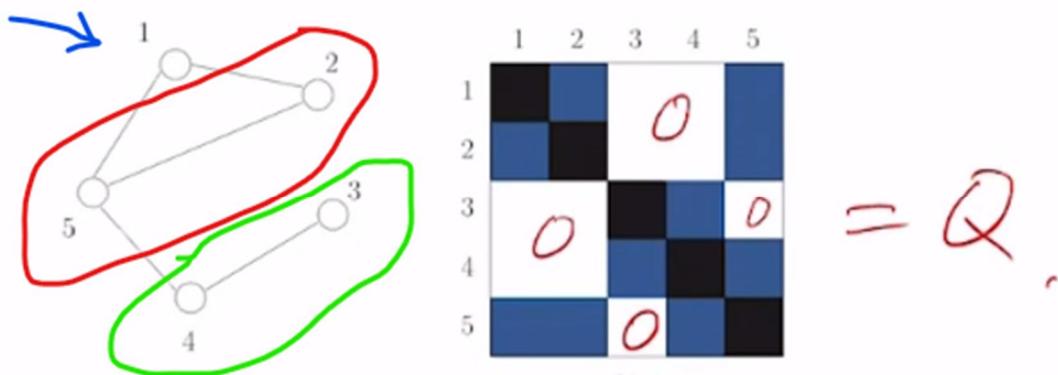
We can add an edge between two nodes if the precision matrix has a non-zero element between these two nodes (random variables).

Black and blue colors in the image represent positive or negative numbers and implies some sort of conditional dependency.

Basically, we can construct a graph by taking "Q" as a sort of the adjacent matrix of this graph. This graph (or Markov graph) includes some important information about conditional independence structures among random variables.

The connectivity between two nodes is related to the conditional dependency and we can imply some properties:

- If two nodes are not connected from the graph (i.e. " $x_2$ " and " $x_3$ "):  
 $i, j \text{ disconnected} \rightarrow x_i \perp x_j \mid x_{\neg ij}$
- Checking the neighborhood of " $x_1$ ": " $x_2$ " and " $x_5$ ", we can claim that when " $x_2$ " and " $x_5$ " are fixed to a value then " $x_1$ " is independent from " $x_3$ " and " $x_4$ ":  
 $x_1 \perp x_3, x_4 \mid x_2, x_5$

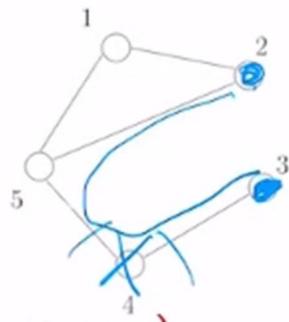


Basically, giving a neighborhood of the node the node is independent with all the other variables outside the neighborhood. The neighborhood of a node is called "Markov blanket".

- Taking a look at " $x_2$ " and " $x_3$ " we know they are independent. Moreover, we can claim something stronger reading the graph:

$$x_1 \perp x_3 \mid x_4$$

Blocking the connection " $x_4$ " (fixing a value) then " $x_2$ " and " $x_3$ " are no longer connected and we have removed all paths.



More, generically we can state that if we have two groups of labels “A” and “B” independent, conditional independence given “C” happens if “C” is a bottleneck between “A” and “B”:

$$x_A \perp x_B | x_C \text{ if } C \text{ is a bottleneck between } A \text{ and } B$$

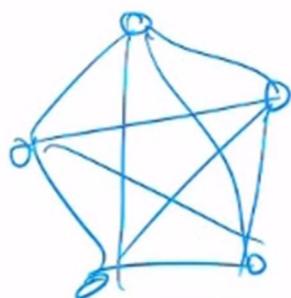
Markov graph reveals very important information about the dependency structure.

Markov graph is obtained using “ $Q = \Sigma^{-1}$ ”.

Using the covalence matrix we also obtain kind of a graph. However this graph does not reveal a lot of information as two nodes could be correlated with each other through other nodes:

For example “ $x_2$ ” and “ $x_3$ ” are conditionally independent because they have correlation going through “ $x_4$ ” and “ $x_5$ ”.

With the covalence graph we get a fully connected graph as everyone is correlated with each other:



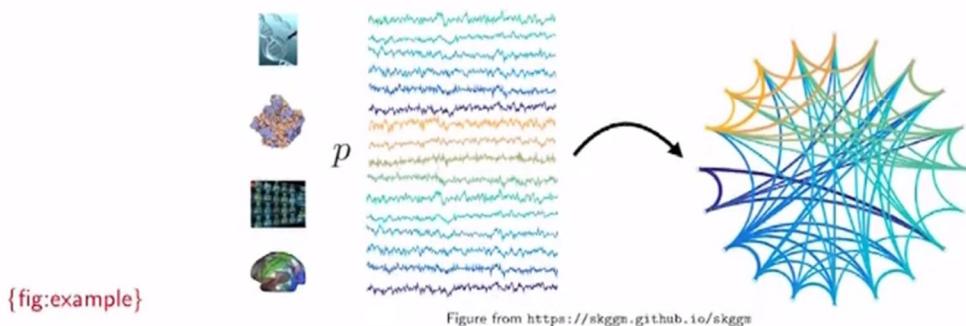
This happens because either there is a direct correlation or there is a correlation through other variables. But, it does not reveal which part of the dependency is direct or indirect.

Markov graph reveals a lot of useful information.

Learning Gaussian Markov Random Fields**Learning Gaussian Markov Random Fields**

- Given observation  $D = \{\mathbf{x}^k\}$  drawn from  $\mathcal{N}(\mu, \Sigma)$ .
- Assume the precision matrix  $\mathbf{Q} = \Sigma^{-1}$  is sparse.
- Goal: Estimate  $\mathbf{Q}$  and its graph structure.

- Broad applications: systems biology, neuroscience, psychometrics, and finance.



Applications of this graph turns out to be very useful and has many applications.

In practice, we observe a bunch of data points and different information. We can try to find the “Q” matrix as well as the graph structure. This graph structure allows us to reasoning about the dependency and the relations between different variables.

In biology, we want to measure the expression data of different genes.

In the example above, each row represents the expression of a particular gene. Then, we can view the joint distribution of all the genes as a multivariate normal distribution and we can try to estimate the struvture of the inverse covalence matrix.

The above allows us of reasoning about the relations between different genes.

In finance, we have price of different stocks and they are correlated in some way. Let us say Apple and Google the price could be correlated. We can view the stock prices as multivariate normal distributions and try to estimate the “Q” matrix.

**Graphical Lasso**

- Estimate sparse  $\mathbf{Q}$  from data  $D = \{\mathbf{x}^k\}$ .
- Idea: Use maximum likelihood estimation with some penalty to encourage sparsity.

We observe a bunch of data “ $D=\{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^k\}$ ”, the matrix of Data takes this shape:

$$D = \begin{bmatrix} x_1^1 & \dots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_1^d & \dots & x_n^d \end{bmatrix}_{dxn} \sim N(\mu, \Sigma)$$

We want to estimate “ $Q=\Sigma^{-1}$ ” and we hope “ $Q$ ” to be sparse (i.e. with a lot of elements being “0”, where two variables have direct correlation).

A simple way to solve this problem is **graphical lasso**: use MLE with some penalty to encourage sparsity.

**Assumption:** the multivariate distribution is:

$$D = \begin{bmatrix} x_1^1 & \dots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_1^d & \dots & x_n^d \end{bmatrix}_{dxn} \sim N(0, \Sigma)$$

This can be achieved centralizing/normalizing the data so it has “0” mean “ $\mu=0$ ”.

- Estimate sparse  $Q$  from data  $D = \{x^k\}$
- Idea: Use maximum likelihood estimation with some penalty to encourage sparsity.

$D =$   $x^k$   $d \times N$

$\sim N(\hat{\mu}, \hat{\Sigma})$

Estimate  $\underline{Q} = \hat{\Sigma}^{-1}$ . Sparse

**Applying MLE:** we have to apply it in a way we encourage sparsity, otherwise, it is the same approach we took earlier:

$$\max_Q \log p(D|Q) - \lambda \cdot \phi(Q)$$

The likelihood term is the standard likelihood function. For the sparsity it could mean the number of non-zero elements using again the indicator function “I”:

$$\phi(Q) \sum_{i \neq j} I(q_{ij} \neq 0) = \# \text{ of edges} = \|Q_0\|_1 \text{ (L}_1 \text{ norm of } Q)$$

If “ $q_{ij} \neq 0$ ” it will contribute with an indicator function and it will sum over all the nodes obtaining the number of edges.

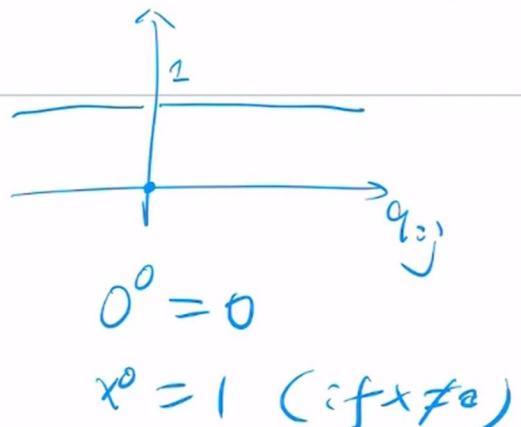
" $\phi$ " is the number of edges and we want it to be as less as possible.

In order to maximize likelihood we need to minimize edges. Unfortunately, this is difficult to optimize as it is not a continuous function and gradient descent cannot be used.

$$\max_Q \underbrace{\log P(D|Q)}_{\text{likelihood}} - \lambda \underbrace{\Phi(Q)}_{\text{edges}}$$

$$\begin{aligned} \Phi(Q) &= \sum_{i \neq j} \underbrace{\mathbb{I}(q_{ij} \neq 0)}_{\text{# of edges}} = \# \text{ of edges} \\ &= \sum_{i \neq j} q_{ij}^0 \end{aligned}$$

( $L_0$  Norm)



Fortunately, we can use a concept call the " $L_1$ " norm as approcimation of the " $L_0$ " norm:

$$L_1 = \|Q\|_1 = \phi(Q) = \sum_{i \neq j} |q_{ij}|$$

This is a continuous and convex function that can be optimized much more efficiently and return a similar result: " $L_1=0$ " if all elements outside the diagonal are "0", which is similar to the case " $L_0=0$ " and it is meaningful to replace..

The problem gets adjusted:

$$\max_Q \log p(D|Q) - \lambda \cdot \|Q\|_1 \quad (\text{Graphical Lasso})$$

**Note:** from Natural form the probability can be expressed as this:

$$p(x) \propto e^{-\frac{1}{2}x^T Q x + b^T x} = e^f$$

**Note 2:** log cancels exp.

$$\log p(x) = \log(e^f) = f$$

Then, finally applying this formula to all points in data "D" we get the "p(x)" as a summation:

$$\begin{aligned} \max_Q \log p(D|Q) - \lambda \cdot \|Q\|_1 &= \sum_{k=1}^n \log p(x^k|Q) - \lambda \cdot \|Q\|_1 \\ &= \sum_{k=1}^n \left( -\frac{1}{2} \cdot (x^k)^T \cdot Q \cdot (x^k) + \frac{1}{2} \cdot \log \det(Q) \right) - \lambda \cdot \|Q\|_1 \end{aligned}$$

We can use gradient descent, coordinate descent, or use another efficient algorithm.

There are many different implementations for Graphical Lasso which are interesting.

*Convex function*       $x^* = 1 \quad (\text{if } x \neq 0)$

$$\begin{aligned} \max_Q \quad & \underbrace{\log P(D|Q)}_{\downarrow} - \underbrace{\lambda \|Q\|_1}_{\text{Graphical Lasso}}. \\ & \sum_{k=1}^N \log P(x^k|Q) \\ & = \sum_{k=1}^N \left( -\frac{1}{2}(x^k)^T Q (x^k) + \frac{1}{2} \log \det(Q) \right) \end{aligned}$$



If we optimize this problem we will get an estimation of "Q" that is sparse. This will result in a nice graphical structure that captures the dependency structure and dependency relation between variables.

## Kernel method

### Introduction

A kernel method is a very important class of algorithms for constructing very flexible, nonlinear functional approximations in machine learning.

### Supervised learning

## Supervised Learning Framework

- Given training data  $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$ .
- Want to find  $f(x)$ , such that  $y_i \approx f(x_i)$ .
- Empirical risk minimization:
  - Decide a function class  $\mathcal{F}$ .
  - Define an empirical loss function  $L(f; \mathcal{D})$  for  $f \in \mathcal{F}$
  - Solve optimization:

$$\hat{f} = \arg \min_{f \in \mathcal{F}} L(f; \mathcal{D}).$$

First, let us review the frameworks for supervised learning, which also works for unsupervised learning.

In supervised learning, we have:

- We have a collection of pairs: " $x_i = \{x_1, x_2, \dots, x_n\}$ ,  $y_i = \{y_1, y_2, \dots, y_n\}$ ". Where " $x_i$ " is an input variable and " $y_i$ " is a value to predict. This is the input Data "D".
- Goal: estimate some function " $f(x)$ " which takes " $x$ " and outputs a prediction. We hope that the prediction matches the label " $y$ ".



The classical way to solve this is by empirical risk minimization:

- Start defining a function class " $F$ " that specifies all possible functions we want to consider for our problem (i.e. in linear regression we assume our " $F$ " is a set of linear functions).

$$F \triangleq \left\{ f_{\theta}(x) = \sum_{k=1}^d \theta_k \cdot x_k + \theta_0 \mid \forall \theta \in R \right\}$$

Where " $\theta$ " is the vector with coefficients for the linear function and " $\theta_0$ " is the bias term.

- Goal: find among a set of linear functions the best linear function that fits with the data.
- Once defined the function class " $F$ ", we need to define the loss function that measures the consistency between the data "D" and the function " $f_{\theta}$ ".

$$L(f_{\theta}, D) = \frac{1}{n} \cdot \sum_{i=1}^n (y^i - f_{\theta} \cdot (x^i))^2$$

- Then, the problem reduces to finding the best possible function that minimizes this loss. In other words: minimize the loss function among all possible " $\theta$ " values.

$$\min_{\theta} L(f_{\theta}, D) = \frac{1}{n} \cdot \sum_{i=1}^n (y^i - f_{\theta} \cdot (x^i))^2$$

- Use numerical procedure or algorithm to solve this optimization problem: in linear regression there is a nice formula, more complicated cases need gradient descent.

Reviewed this, we can conclude that supervised learning needs three key elements:

1. Function class flexible enough to capture complexity of the data set. Not too large or we might overfit it.
2. Loss function that quantifies the goodness of each function inside the function class.
3. Solve the optimization problem using an algorithm or procedure.

$$\mathcal{F} \triangleq \left\{ f_{\theta}(x) = \sum_{e=1}^d \theta_e x_e + \theta_0 \mid \forall \theta_e \in \mathbb{R} \right\}$$

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$$

$$\min_{\theta} L(f_{\theta}, D) = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - f_{\theta}(x^{(i)}))^2$$

In this lecture the problem to consider is regarding the function class. In linear regression we are considering a set of linear functions. This is obviously more restricted. Instead, we want to consider more general, more flexible, nonlinear function classes. Kernel method is one of them, actually one of the most important class of nonlinear class approximation. Another class of nonlinear function approximation is something called neural network which is the building block of the deep learning methodology very popular today.

#### Basic ideas to build flexible function classes

Remember linear regression is sum over a weighted combination of our features: "x={x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>d</sub>}":

$$\text{Linear: } f_{\theta}(x) = \sum_{j=1}^d \theta_j \cdot x_j$$

Let us extend this function to make it nonlinear with a naïve approach:

$$\text{Nonlinear: } f_{\theta}(x) = \sum_{j=1}^{d'} \theta_j \cdot \phi_j$$

Where phi " $\phi=\{\phi_1, \phi_2, \dots, \phi_d\}$ " is a set of nonlinear functions that takes input "x" and extracts something that is some sort of feature. Then, our function is defined as a linear combination of these features.

$$\text{Linear: } f_{\theta}(x) = \sum_{e=1}^d \theta_e x_e$$

$$\text{Nonlinear: } f_{\theta}(x) = \sum_{e=1}^{d'} \theta_e \underline{\phi}_e(x)$$

Basic functions “ $\phi$ ”:

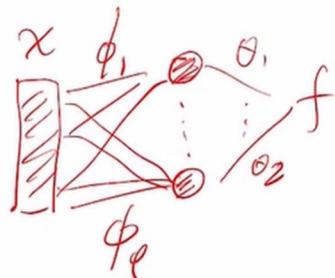
$$\phi_l(x) = x^l \rightarrow 1, x^1, x^2, \dots$$

These functions perform a nonlinear transform to extract features. This is a fundamental and basic that essentially allows us to build all kind of nonlinear approximations.

Nonlinear:  $f_{\theta}(x) = \sum_{l=1}^{d'} \theta_l \underline{\phi_l(x)}$

$\phi_l$ : Basis function

$\phi_l(x) = x^l \Rightarrow 1, x, x^2, \dots$



We hope we can construct the feature basis in a more adaptive way. For example, we have the dataset “D”, and we have an algorithm that automatically constructs a bunch of basis functions and then define a linear regression on the basis function defined. This gives us a very flexible nonlinear approximation. With many basis functions we can make approximations even more flexible.

We want to use “adaptive basis functions”, so basic functions are built automatically from the training data. The Kernel method is one of these approaches together with neural networks.

#### Kernel method

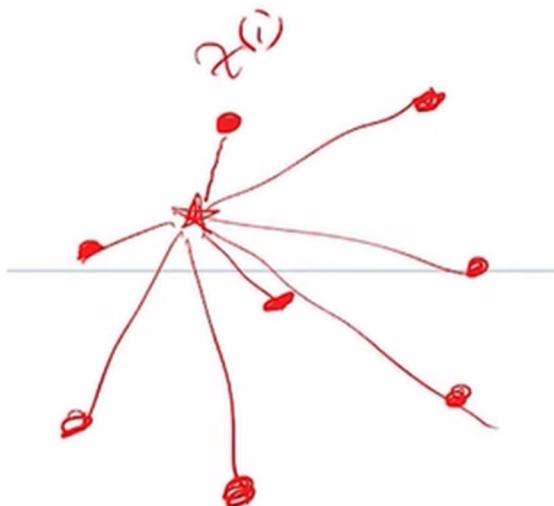
Let us say we have a set of data points:

$$D = \{x^i, y^i\}$$

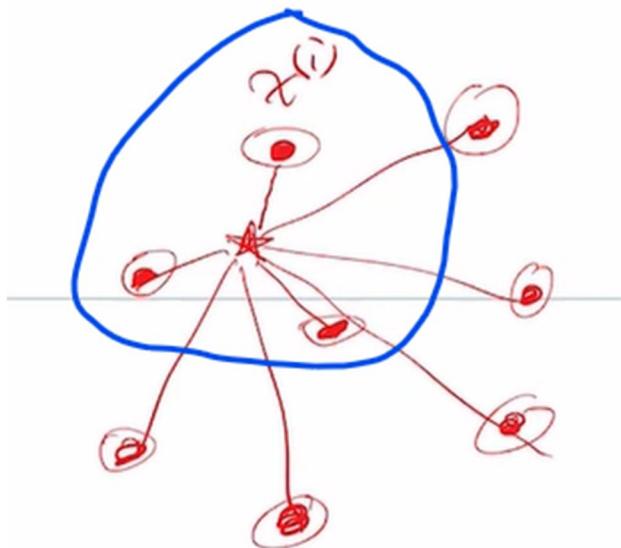
These data points are dots in the space.

The idea of Kernel methods is that instead of using the original coordinates “ $x_i$ ” to represent each data point I can represent them using its similarity with the other points.

The most basic way to represent a point is using its coordinate, which tells us the location of this point and carries all the information we need.



With sufficiently training data we can identify similar points that are close to each other. This would allow us to have more dimensions to represent the data points so we can get more flexible representation and approximations.



To achieve this, we can define a similarity function that takes two points and outputs a similarity score.

$$K(x, x'): X \times X \rightarrow R$$

*Example of similarity function: Gaussian functions*

$$K(x, x') = e^{-\frac{1}{2 \cdot h^2} \|x - x'\|^2}$$

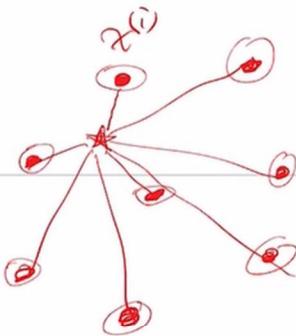
We have two points "x" and "x'", calculate square distance and then scale it by some parameter "h". Then, we take exponential to make sure this is scaled to be positive.

This is the Gaussian Radial Basis kernel.

$$\mathcal{D} = \{x_i, y_i\}$$

**Similarity function**

$$K(x, x') : X \times X \mapsto \mathbb{R}$$



Example:

$$K(x, x') = \exp\left(-\frac{1}{2h^2} \|x - x'\|^2\right)$$

Gaussian Radial Basis function (RBF)

Kernel.

A similarity function, a Kernel: a function that takes two variables and is symmetric:

$$K(x, x') = K(x', x)$$

*Example of similarity function: laplacian kernel*

$$K(x, x') = e^{-\frac{1}{h} \|x - x'\|}$$

This form does not have the quadratic form. It is the Laplacian kernel.

*Choosing hyperparameter "h" value*

In all these Kernels there is a parameter "h" in place. "h" is a hyperparameter that defines the scale of the similarity. "h" is also called the bandwidth. Deciding this hyperparameter is not a difficult job: typically we get the range of data points and "h" is normally the **empirical variance of the data points**.

Kernel and phi

Remember phi " $\phi = \{\phi_1, \phi_2, \dots, \phi_d\}$ " is a set of nonlinear functions that takes input "x" and extracts something that is some sort of feature.

Back to the idea of constructing features using similarity we can do the following: let us define a feature map using the similarity.

$$\phi(x) = \begin{Bmatrix} \phi_1(x) \\ \vdots \\ \phi_n(x) \end{Bmatrix} = \begin{Bmatrix} K(x, x_1) \\ \vdots \\ K(x, x_n) \end{Bmatrix}$$

The idea is to calculate the similarity from all the data points inside the training data. Therefore, if the data point “ $x$ ” is one of the points in the training data “ $D$ ” then the similarity with itself will take the corresponding maximum value:

$$\phi(x_1) = \begin{Bmatrix} \phi_1(x) \\ \vdots \\ \phi_n(x) \end{Bmatrix} = \begin{Bmatrix} K(x_1, x_1) \\ \vdots \\ K(x_1, x_n) \end{Bmatrix} = \begin{Bmatrix} 1 \\ \vdots \\ 0 \end{Bmatrix}$$

We can use this representation to construct the linear regression. This allows us to define a fairly flexible class of functions:

$$f_{\theta}(x) = \theta^T \cdot \phi(x) = \sum_{i=1}^n \theta_i \cdot \phi_i(x) = \sum_{i=1}^n \theta_i \cdot K(x, x_i)$$

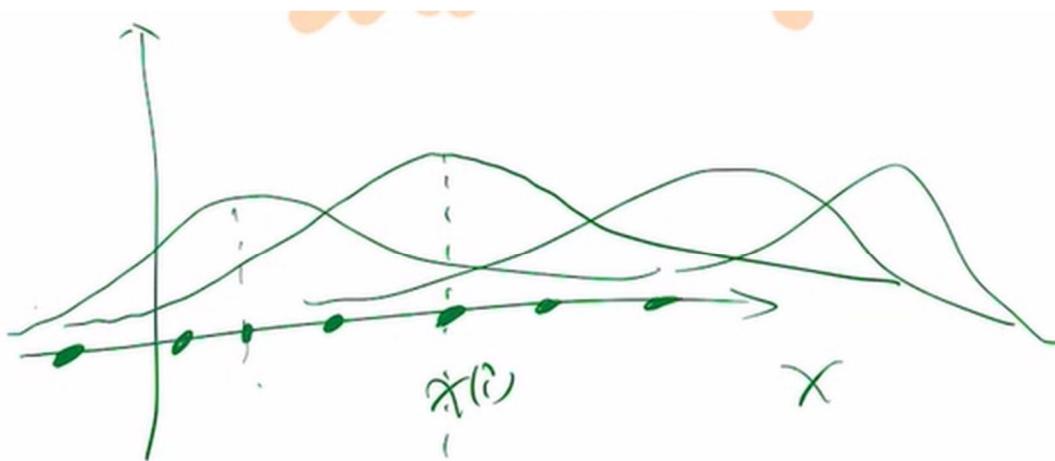
What this does is taking the sum over all the data points, measures the similarity from “ $x$ ” to each data point and then each data point is weighted by some parameter “ $\theta_i$ ”.

“ $\theta_i$ ” measures the importance of each of the data points. By minimizing the loss function we will be able to estimate “ $\theta_i$ ” from the data.

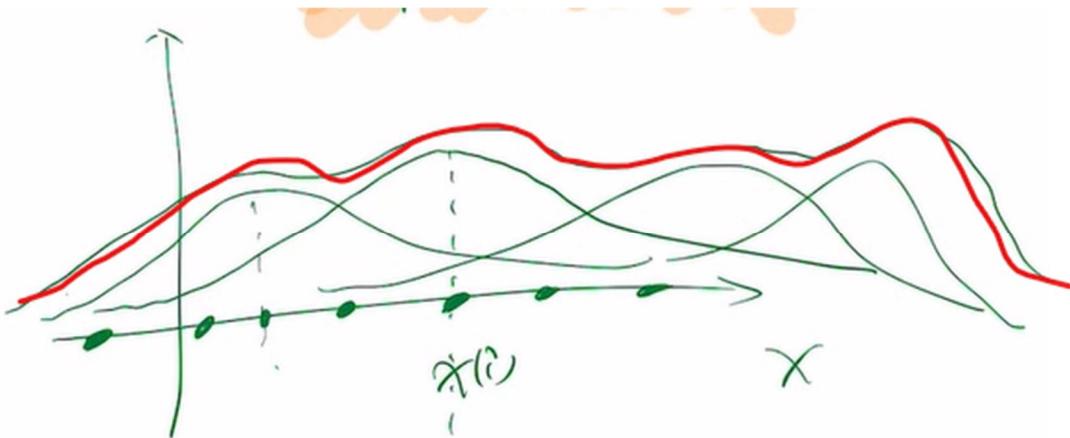
#### *Example in 1D*

Using Gaussian Kernel. We have a bunch of data.

Each data point has a Gaussian bell curve using it as Kernel.



So eventually, we are getting a fairly flexible nonlinear function:



Summing all together we get a nonlinear function.

#### Properties of Kernel functions as basis functions

- It always measure the similarity with the data itself. So it allows us to represent the features in our data in a adaptive way.
- If we have more data, we will have more features to represent and make the function more flexible.
- The number of parameters in this function class is exactly the size of the training data: the larger the data set, the larger the parameters and flexibility of this methodology.

We want to keep our function class to be flexible in order to capture all the complex patterns in the data.

The function class “F” to be adaptive with data “D”:

$$\dim(F) = n$$

This class of function methods where dimension of function class grows with the data set is called “**nonparametric methods**”

The Kernel method is a special case of nonparametric methods.

#### Estimating theta “θ”

Eventually, we want to find a optimal “θ” that minimizes the loss function:

$$\min_{\theta} L(\theta) = \sum_{i=1}^n \left( y^i - \sum_{j=1}^n \theta_j \cdot K(x^i, x^j) \right)^2$$

For each point “ $x_i$ ” we measure the square difference between “ $y_i$ ” and the kernel function evaluated at “ $x_i$ ”.

We can write this loss function in its matrix form:

$$Y = \begin{Bmatrix} y^1 \\ \vdots \\ y^n \end{Bmatrix}_{nx1} ; K = \begin{bmatrix} K(x_1, x_1) & \dots & K(x_1, x_n) \\ \vdots & \ddots & \vdots \\ K(x_n, x_1) & \dots & K(x_n, x_n) \end{bmatrix}_{nxn} ; \theta = \begin{Bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{Bmatrix}_{nx1}$$

Which ends up with the L2-norm appearing:

$$\min_{\theta} L(\theta) = \|Y - K \cdot \theta\|_2^2$$

This is a standard linear regression problem. In order to solve it we take the derivative with respect to the parameters “ $\theta$ ”:

$$\frac{dL(\theta)}{d\theta} = 2 \cdot K^T \cdot (K \cdot \theta - Y) = 0 \rightarrow K^T \cdot K \cdot \theta = K^T \cdot Y$$

Assuming “K” is invertible then we can solve this:

$$\theta = K^{-1} \cdot Y$$



$$\min_{\theta} L(\theta) = \sum_{j=1}^n (y^{(j)} - \sum_{i=1}^n \theta_i k(x^{(j)}, x^{(i)}))^2$$

$$= \|Y - K\theta\|_2^2$$

$$Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}, \quad K = \underbrace{\begin{bmatrix} k(x_1, x_1) & \dots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \dots & k(x_n, x_n) \end{bmatrix}}_{n \times n}$$

$$\Rightarrow \nabla L(\theta) = 2K^T(K\theta - Y)$$

$$\Rightarrow K^T K \theta = K^T Y$$

$$\Rightarrow \boxed{\theta = K^{-1} Y} \quad (\text{Assume } K \text{ is invertable})$$

This is a very special case of linear regression as we have as many “ $\theta$ ” (parameters) as number data points. We actually can perfectly fit each of the data point perfectly. Unfortunately, this can overfit the actual curve as we have many parameters. We can try a regularization to enforce that this solution does not overfit.

In order to enforce the regularization instead of doing exact linear regression we can actually solve a regularized version of the Loss function as follows:

$$\min_{\theta} L(\theta) = \left\| Y - K \cdot \theta \right\|_2^2 + \alpha \cdot \phi(\theta)$$

In many cases, the regularization term can be the 2-norm:

$$\phi(\theta) = \left\| \theta \right\|_2^2 = \theta^T \cdot \theta$$

This enforces that as “ $\alpha$ ” weight grows we prefer solutions that reduce the overfitting (2-norm of the parameters “ $\theta$ ”).

Another natural regularization can be approached using this function:

$$\phi(\theta) = \left\| \theta \right\|_K^2 = \theta^T \cdot K \cdot \theta$$

This regularization allows us to capture the similarity between the data points.

$$\Rightarrow \min_{\theta} \left\| Y - K \theta \right\|_2^2 + \alpha \underline{\phi(\theta)}$$

$$(L_2): \underline{\left\| \theta \right\|_2^2}$$

$$= \theta^T \theta$$

$$\left\| \theta \right\|_K^2 = \theta^T K \theta$$

### *Kernel regression for regularization*

Why the kernel regularization instead of the 2-norm regularization? Let us try proving that the kernel regularization captures the similarity between the points.

Let us say we have a very small data set with only three data points:

$$D = (x^1 \quad x^2 \quad x^3)$$

We will assume:

$$x_2 = x_3$$

In other words, we have duplicate points. This happens often all the time in practice. Very commonly, we get points that are very similar in our datasets.

Our function or kernel representation to estimate will look like:

$$f(x) = \theta_1 \cdot K(x, x^1) + \theta_2 \cdot K(x, x^2) + \theta_3 \cdot K(x, x^3) = (\theta_1 + \theta_2) \cdot K(x, x^1) + \theta_3 \cdot K(x, x^3)$$

Because “ $x_1$ ” and “ $x_2$ ” are the same point we have combined the kernel values of “ $K(x, x_1)$ ” and “ $K(x, x_2)$ ” to make it “ $K(x, x_1)$ ”. We can see that this function only depends on “ $\theta_1$ ” and “ $\theta_2$ ” through the sum of “ $\theta_1$ ” plus “ $\theta_2$ ” because the two points are the same.

If the points are similar to each other the difference between “ $\theta_1$ ” and “ $\theta_2$ ” is not important but the sum of them.

In this case, it makes sense to regularize:

$$\phi(\theta) = \|\theta\|_2^2 = \theta_1^2 + \theta_2^2 + \theta_3^2$$

This expression does not make a lot of sense because “ $x_1$ ” and “ $x_2$ ” are the same point.

Instead, it is more reasonable:

$$\phi(\theta) = \|\theta\|_2^2 = (\theta_1 + \theta_2)^2 + \theta_3^2$$

In fact, it turns out the kernel form is actually doing something similar to this.

$$\|\theta\|_K^2 \approx (\theta_1 + \theta_2)^2 + \theta_3^2$$

In order to prove this, let us assume the kernel is a Gaussian RBF kernel which takes this form:

$$K(x, x') = e^{-\frac{1}{2 \cdot h^2} \|x - x'\|^2}$$

If “ $x=x'$ ” then “ $K(x, x')$ ” = 1.

Whereas if “ $x \neq x'$ ” then “ $K(x, x')$ ” = 0 (we get far from the average in the Gaussian distribution).

Plotting the Kernel matrix of this example:

$$K = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where we can notice that “ $x_1$ ” and “ $x_2$ ” are the same points and “ $x_1$ ” and “ $x_2$ ” are very different to “ $x_3$ ”.

$$\begin{aligned} f(x) &= \underline{\theta_1} K(x, x^{(1)}) + \underline{\theta_2} K(x, x^{(2)}) + \theta_3 K(x, x^{(3)}) \\ &= (\underline{\theta_1 + \theta_2}) K(x, x^{(1)}) + \theta_3 K(x, x^{(3)}) \end{aligned}$$

$$\Rightarrow \|\theta\|_2^2 = \underline{\theta_1^2 + \theta_2^2 + \theta_3^2}$$

$$\text{Better: } (\underline{\theta_1 + \theta_2})^2 + \theta_3^2 = \|\theta\|_K^2$$

$$K(x, x') = \exp(-\frac{1}{2h^2}\|(x - x')\|^2)$$

$$K = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The Kernel regularization takes this shape:

$$\phi(\theta) = \|\theta\|_K^2 = \theta^T \cdot K \cdot \theta = \{\theta_1 \quad \theta_2 \quad \theta_3\} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = (\theta_1 + \theta_2)^2 + \theta_3^2$$

**Conclusion:** it is very reasonable to use the Kernel regularization for Kernel regression.

$$\begin{aligned} \theta^T K \theta &= [\theta_1 \ \theta_2 \ \theta_3] \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \\ &= (\theta_1 + \theta_2)^2 + \theta_3^2 \end{aligned}$$

*Recap the problem*

We want to solve the next problem:

$$\min_{\theta} L(\theta) = \left\| Y - K \cdot \theta \right\|_2^2 + \alpha \cdot \theta^T \cdot K \cdot \theta$$

The solution for this problem taking again derivatives:

$$\begin{aligned} \frac{dL(\theta)}{d\theta} &= 2 \cdot K^T \cdot (K \cdot \theta - Y) + 2 \cdot \alpha \cdot \theta \cdot K = 0 \rightarrow K^T \cdot K \cdot \theta - K \cdot Y + \alpha \cdot K \cdot \theta = 0 \\ &\rightarrow K^T \cdot \theta + \alpha \cdot \theta = Y \rightarrow \theta = (K + \alpha \cdot I)^{-1} \cdot Y \\ \theta &= (K + \lambda \cdot I)^{-1} \cdot Y \end{aligned}$$

**Note:** In this case, I believe the teacher was wrong. I do believe “ $\alpha=\lambda$ ”,

Remember the solution expression with no regularization:

$$\theta = K^{-1} \cdot Y$$

$$\begin{aligned} \min_{\theta} \quad & \|Y - K\theta\|_2^2 + \alpha \theta^T K \theta, \\ \Rightarrow \quad & \hat{\theta} = (K + \lambda I)^{-1} Y. \end{aligned}$$

Kernel method is very easy. The key idea is really just using the similarity as the feature. We assume that our function class is a linear combination of this kernel score functions and then we define a loss function. This expression leads us to the need of regularization in order to improve overfit.

In this case, we solved the linear regression case but we can use Kernel also for classification problems.

#### Kernel for classification problems

In classification problems, we have pairs of data where “y” can only take binary values:

$$D = \{x^i, y^i\} \text{ where } y^i \in \{0, 1\}$$

The Loss function can be a surrogate function like the hinge loss or logistic loss which we basically write as sigmoid (“ $\sigma$ ”):

$$L(\theta) = \sum_{i=1}^n \sigma(y^i \cdot f_\theta(x^i))$$

If we want to do Kernel we simply plug in the Kernel as follows:

$$L(\theta) = \sum_{i=1}^n \sigma\left(y^i \cdot \sum_{j=1}^n \theta_j \cdot K(x^j, x^i)\right)$$

Finally, we solve this numerically using gradient descent or any other numerical algorithm.

We can again use a Kernel regularization to control the complexity and minimize overfit.

The handwritten notes show the following:

- A green circle around the word "Classification".
- A set of training data points:  $\{x^{(i)}, y^{(i)}\}$
- A condition:  $y^{(i)} \in \{\pm 1\}$
- The loss function  $L(\theta) = \sum_{i=1}^n \sigma(y^{(i)} f_{\theta}(x^{(i)}))$
- This is expanded to:  $= \sum_{i=1}^n \sigma(y^{(i)} \sum_{j=1}^n \theta_j k(x^{(i)}, x^{(j)}))$
- With a regularization term:  $+ \lambda \theta^T K \theta$

We should be able to implement a Kernel function or Kernel method for the methods or problems we have.

#### Kernel as equivalent infinite amount of features

The kernel regression is actually equivalent to solving an infinite dimensional optimization problem.

Let us define a set of basis functions in an infinite sequence:

$$\varphi_l(x) : l = 0, 1, 2, \dots, \infty$$

Example with polynomial basis:

$$\varphi_l(x) = x^l$$

The idea behind is to have infinite number of basis functions so we can represent arbitrary functions.

$$f(x) = \sum_{l=0}^{\infty} w_l \cdot \varphi_l(x)$$

Doing a weighted combination of these infinite basis functions, where " $w_l$ " is the weight on the basis function. It turns out that using Taylor expansion any function can be expanded into the sum of infinite sequence of polynomial functions.

This means that setting up a function class in this way to have infinite number of basis functions we can guarantee, theoretically, to closely approximate or represent differentiable and continuous functions which are very flexible and handy.

$$\{\varphi_\ell(x) : \ell = 0, 1, 2, \dots, \infty\}$$

Example :  $\varphi_\ell(x) = x^\ell$

$$f(x) = \sum_{\ell=0}^{\infty} w_\ell \varphi_\ell(x)$$

The idea is to use infinite basis function to represent arbitrary and non-linear functions.

We can define a Loss function:

$$L(w) = \hat{E}_D \left[ \left( y - \sum_{l=0}^{\infty} w_l \cdot \varphi_l(x) \right)^2 \right]$$

This is an infinite dimensional linear regression that will overfit as we have finite number of data but infinite number of parameters.

Using again the weighted 2-regularization with a penalty value that penalizes the magnitude or dimension of "w":

$$L(w) = \hat{E}_D \left[ \left( y - \sum_{l=0}^{\infty} w_l \cdot \varphi_l(x) \right)^2 \right] + \alpha \cdot \sum_{l=0}^{\infty} \frac{w_l^2}{\lambda_l}$$

Where " $E_D$ " is the empirical average or expectation.

We scale each term with " $\lambda$ " which is a positive coefficient for "w". Its value controls how much we want to penalize over "w". And:

$$\lambda_l > 0, \lambda_1 > \lambda_2 > \dots > 0$$

Bear in mind that if " $\lambda$ " is small then the penalty is huge as we are strongly penalizing the value of "w":

$$\lambda_l \text{ small: } w_l \rightarrow 0$$

Therefore, because we have infinite number of data or infinite number of features we need a very strong penalty over the high-order basis functions so we try to find low-order magnitudes.

The high-order terms are penalized exponentially heavily.

$$L(\underline{\omega}) = \hat{E}_D \left[ (y - \sum_{k=0}^{\infty} \omega_k \varphi_k(x))^2 \right] + \alpha \cdot \sum_{k=0}^{\infty} \frac{\omega_k^2}{\lambda_k}$$

$\lambda_k > 0$ , Regularization Coeff.  
for  $\omega_k$ .

$$\lambda_1 \geq \lambda_2 \geq \dots \geq 0$$

$\lambda$  small  $\Rightarrow \omega_k \rightarrow 0$

In practice, solving this problem shows some remarkable results:

- The infinite dimensional optimization problem where we want to find infinite number of “ $w$ ” is actually equivalent to learning a Kernel function that we just showed earlier.
- Difference is that the Kernel function is finite dimensional whereas this problem is infinite.
- Reason: this strong regularization allows us to turn off the high-order terms.
- Another reason: this Loss function only depends on finite number of data. The effective number of parameters that we need is at most equal to the size of the data.

Let us show the demonstration.

*Demonstration finite kernel is equivalent to infinite basis functions*

Let us find minimum “ $\hat{w}$ ” that optimizes and minimizes the expression for the Loss function using the infinite regularization formula we have seen earlier:

$$\hat{w} = \min_w \left( \hat{E}_D \left[ \left( y - \sum_{l=0}^{\infty} w_l \cdot \varphi_l(x) \right)^2 \right] + \alpha \cdot \sum_{l=0}^{\infty} \frac{w_l^2}{\lambda_l} \right)$$

Similarly, we can define the estimated function “ $f(x)$ ” as:

$$\hat{f}(x) = \sum_{l=0}^{\infty} \hat{w}_l \cdot \varphi_l(x)$$

In addition, we can show that "f(x)" always have a kernel representation:

$$\hat{f}(x) = \sum_{i=1}^n \theta_i \cdot K(x, x_i)$$

We can show that the optimal "f(x)" is a linear combination of a kernel function for some " $\theta_i$ " we want to find and the Kernel is defined the following way:

$$K(x, x') = \sum_{l=0}^{\infty} \lambda_l \cdot \varphi_l(x) \cdot \varphi_l(x')$$

We can use the basis function " $\phi(x)$ " and the regularization magnitude " $\lambda$ " in order to define a Kernel. The kernel for "x" and "x'" is the sum over all the possible "l" from "0" to infinite.

Then, each of the basis function is weighted by " $\lambda$ " which is the importance of that basis function.

$$\begin{aligned} \hat{f}(x) &= \sum_{l=0}^{\infty} \tilde{\omega}_l \varphi_l(x) \\ \text{Then: } \hat{f}(x) &= \sum_{i=1}^n \theta_i \underline{K(x, x_i)} \\ \text{for } \theta_i, \text{ and } K(x, x') &= \sum_{l=0}^{\infty} \lambda_l \underline{\varphi_l(x)} \underline{\varphi_l(x')} \end{aligned}$$

The basis function defines some kernel and this infinite dimensional optimization is equivalent to finding a basis representation. In other words, the optimal solution always has a finite dimensional optimization so instead of searching for the infinite dimensional parameters we can just search for the optimal Kernel representation.

We can also prove that the regularization step is equivalent to the Kernel regularization:

$$\alpha \cdot \sum_{l=0}^{\infty} \frac{w_l^2}{\lambda_l} = \alpha \cdot \theta^T \cdot K \cdot \theta$$

And.

$$\sum_{l=0}^{\infty} w_l^2 / \lambda_l = \theta^T K \theta$$

There are lots of advanced methods in machine learning taking advantage of this results.

The idea to proof this statement is quite simple.

If we optimize the original function:

$$\hat{w} = \min_w \left( \hat{E}_D \left[ \left( y - \sum_{l=0}^{\infty} w_l \cdot \varphi_l(x) \right)^2 \right] + \alpha \sum_{l=0}^{\infty} \frac{w_l^2}{\lambda_l} \right)$$

We need to find the zero-gradient equation to find the optimal values.

$$\min_w \left( \hat{E}_D \left[ \left( y - \sum_{l=0}^{\infty} w_l \cdot \varphi_l(x) \right)^2 \right] + \alpha \cdot \sum_{l=0}^{\infty} \frac{w_l^2}{\lambda_l} \right) = \min_w L(w)$$

The gradient of this expression needs to be "0" in order to find the minimum:

$$\frac{dL(w)}{dw} = 0$$

Prior to solving this let us define:

$$\delta(x) = y - \sum_{l=0}^{\infty} w_l \cdot \varphi_l(x) \quad (\text{residual difference})$$

Taking the gradient for point "l":

$$\left( \frac{dL(w)}{dw} \right)_l = 2 \cdot \hat{E}_D [-\delta(x) \cdot \varphi_l(x)] + 2 \cdot \alpha \cdot \frac{w_l}{\lambda_l}$$

Where " $E_D$ " can be the empirical average, if we isolate " $w_l$ " to obtain the optimum value:

$$\hat{w}_l = \lambda_l \cdot \hat{E}_D [\delta(x) \cdot \varphi_l(x)] \rightarrow \hat{w}_l = \frac{\lambda_l}{n} \cdot \sum_{i=1}^n (\delta(x^i) \cdot \varphi_l(x^i))$$

This gives us a representation for the optimal "l-th" element of "w".

Plugging this solution in for the optimal "f(x)":

$$\begin{aligned}
 \hat{f}(x) &= \sum_{l=0}^{\infty} \hat{w}_l \cdot \varphi_l(x) = \sum_{l=0}^{\infty} \left( \frac{\lambda_l}{n} \cdot \sum_{i=1}^n (\delta(x^i) \cdot \varphi_l(x^i)) \right) \cdot \varphi_l(x) \\
 &= \sum_{l=0}^{\infty} \sum_{i=1}^n \left( \frac{\lambda_l}{n} \cdot \sum_{i=1}^n (\delta(x^i) \cdot \varphi_l(x^i)) \right) \cdot \varphi_l(x) \\
 &= \sum_{i=1}^n \sum_{l=0}^{\infty} \frac{\lambda_l}{n} \cdot \delta(x^i) \cdot \varphi_l(x^i) \cdot \varphi_l(x) = \sum_{i=1}^n \frac{\delta(x^i)}{n} \cdot \sum_{l=0}^{\infty} \lambda_l \cdot \varphi_l(x^i) \cdot \varphi_l(x) \\
 &= \sum_{i=1}^n \frac{\delta(x^i)}{n} \cdot K(x^i, x)
 \end{aligned}$$

Therefore, the optimal "f(x)" is a linear combination of Kernel functions defined exactly as the infinite sum of the " $\lambda$ " and the basis functions.

We are almost there:

$$\hat{f}(x) = \sum_{i=1}^n \frac{\delta(x^i)}{n} \cdot K(x^i, x) = \sum_{i=1}^n \theta_i \cdot K(x^i, x)$$

$$\Rightarrow \text{Define } \underline{\delta(x)} = \underline{y} - \sum_{\ell=0}^{\infty} \underline{\omega_\ell \varphi_\ell(x)}$$

$$\nabla_{\omega_\ell} L(\hat{\omega}) = 2 \hat{E}_D [-\delta(x) \varphi_\ell(x)] + \frac{2 \hat{\omega}_\ell}{\lambda_\ell} = 0$$

$$\Rightarrow \underline{\hat{\omega}_\ell} = \lambda_\ell \hat{E}_D [\delta(x) \varphi_\ell(x)]$$

$$= \frac{\lambda_\ell}{n} \sum_{i=1}^n [\delta(x^i) \varphi_\ell(x^i)]$$

$$\boxed{f(x) = \sum_{\ell=0}^{\infty} \underline{\hat{\omega}_\ell \varphi_\ell(x)}}$$

$$\Rightarrow \underline{\hat{\omega}_c} = \lambda_c \hat{E}_{\mathcal{D}} [\delta(x) \underline{Y_c(x)}]$$

$$= \frac{\lambda_c}{n} \sum_{i=1}^n [\delta(x^i) \underline{Y_c(x^i)}]$$

$$\boxed{f(x) = \sum_{c=0}^{\infty} \underline{\hat{\omega}_c} \underline{Y_c(x)}}$$

$$= \sum_{c=0}^{\infty} \frac{\lambda_c}{n} \sum_{i=1}^n \delta(x^i) \underline{Y_c(x^i)} \underline{Y_c(x)}$$

$$= \sum_{c=0}^{\infty} \sum_{i=1}^n \frac{\lambda_c}{n} \delta(x^i) \underline{Y_c(x^i)} \underline{Y_c(x)}$$

$$= \sum_{i=1}^n \frac{\delta(x^i)}{n} k(x^i, x)$$

$$= \boxed{\sum_{i=1}^n \theta_i k(x^i, x)}$$

$$\theta_i \triangleq \frac{\delta(x^i)}{n}$$

Conclusions

There are two equivalent views of Kernel methods:

1. “ $f(x)$ ” is a linear combination of the kernel functions evaluated at each data point so kernel serves a similarity measure for finite representations.
2. “ $f(x)$ ” is equivalent to a potentially infinite number of basis functions.

Two Views :

$$\textcircled{1} \quad \hat{f}(x) = \sum_{i=1}^n \hat{\theta}_i k(x^i, x)$$

$$\textcircled{2} \quad \hat{f}(x) = \sum_{\ell=0}^{\infty} \hat{\omega}_{\ell} \phi_{\ell}(x)$$

These two views are connected by this third idea:

$$\Leftrightarrow k(x, x') = \sum_{\ell=0}^{\infty} \lambda_{\ell} \phi_{\ell}(x) \phi_{\ell}(x')$$

Where “ $\lambda$ ” is a regularization coefficient for the basis functions. And “ $\lambda > 0$ ”.

This representation of kernel function is fairly general and turns out that any basis function can somehow be represented using basis functions.

We can show the following statements:

If  $K(x, x')$  is positive definite  $\leftrightarrow K = [k(x^i, x^j)]_{j=1}^n \geq 0$  for any  $n, x^i$

**Note:** matrix kernel “ $K$ ” is positive-definite for vector “ $x = \{x_1^i, x_2^i, \dots, x_n^i\}$ ” if:

$$x^{iT} \cdot K \cdot x^i \geq 0$$

Then exists a basis function “ $\phi$ ” and a coefficient “ $\lambda$ ”. Being a positive-definite will correspond to having a positive “ $\lambda$ ”.

The typical kernels we use are all positive definites.

① If  $k(x, x')$  is positive definite.

$$\Leftrightarrow K = [k(x^{(i)}, x^{(j)})]_{i,j=1}^n \geq 0$$

for any  $n$ ,  $\{x^{(i)}\}$

②  $\Leftrightarrow \exists \psi_\epsilon, \lambda_\epsilon \geq 0$

In the case of the Gaussian distribution we can prove it:

Gaussian RBF Kernel:

$$\begin{aligned} k(x, x') &= \exp(-\gamma \|x - x'\|^2) \\ &= \exp(-\gamma \|x\|^2 + 2\gamma x^T x' - \gamma \|x'\|^2) \\ &= \underbrace{\exp(-\gamma \|x\|^2)}_{=} \underbrace{\exp(-\gamma \|x'\|^2)}_{=} \underbrace{\exp(2\gamma x^T x')}_{=} \end{aligned}$$

Using Taylor expansion we can represent the exponential function for the last term (as the previous two only depend on either "x" or "x'", the coupling of "x" and "x'" only happens in the last term):

$$\begin{aligned} \exp(t) &= 1 + t + \frac{t^2}{2} + \dots \uparrow \\ &= \sum_{\ell=0}^{\infty} \frac{t^\ell}{\ell!} \\ \exp(2\gamma x^T x') &= \left[ \sum_{\ell=0}^{\infty} \frac{(2\gamma x^T x')^\ell}{\ell!} \right] \end{aligned}$$

Assume that "x" is a real number (no longer a vector for this example):

$$\Rightarrow \text{Assume } x \in \mathbb{R}.$$

$$K(x, x') = \sum_{l=0}^{\infty} \lambda_l \varphi_l(x) \varphi_l(x')$$

$$\varphi_l(x) = \underbrace{\exp(-\gamma x^2)}_{\lambda_l} x^l$$

$$\lambda_l = \frac{(2\gamma)^l}{l!}$$

If we define our basis function in this way we can write the Gaussian RBF kernel as a infinite dimensional representation where " $\lambda$ " is greater or equal to zero and we can show that Gaussian RBF kernel is positive definite.

## Neural networks

### Introduction

A neural network is a extremely powerful non-linear approximation widely used and successful in machine learning.

## Approaches to Nonlinear Regression:

- Fixed basis function:

$$f(x; w) = \sum_i w_i \phi_i(x),$$

e.g., polynomial regression:  $f(x; w) = w_0 + w_1 x + w_2 x^2 + \dots$ .

- Adaptive basis functions:

- Kernel method:

$$f(x; w) = \sum_{i \in \text{data}} w_i k(x, x_i).$$

- Neural networks:

$$f(x; w, v) = \sum_i w_i \phi(x; v_i).$$

We want to extend linear regression, which is using linear functions, to approximate complex data sets with the extension to nonlinear approximations.

We can use basis functions where “x” is a set of features and we map the feature into some sort of abstract latent variable using nonlinear functions “ $\phi$ ”. Then, we can perform a linear regression over these basis functions

We have talked about kernel methods to represent adaptive basis functions based on our dataset so we can capture the pattern from the data more efficiently.

In the kernel method the basis function to be the kernel function “ $k(x, x^i)$ ” to represent the similarity from a particular “x” to a particular point “ $x^i$ ”. In the case of kernel this kernel function evaluated at each of the data points.

In the case of neural networks, we have another simple way to design adaptive basis functions. The idea is to assume each of the basis function has a parametric form. This parametric form has some parameters and then we just jointly train the parameter of the basis function (“ $v_i$ ”) together with the weighted parameters we have “ $w_i$ ”:

$$\phi_i(x) = \phi(x, v_i)$$

“ $v_i$ ” is a parameter we introduced for the basis function.

Summarizing:

“ $w_i$ ”: original linear coefficients.

“ $v_i$ ”: coefficient of the basis function.

Now, we want the optimization to estimate “ $w_i$ ” and “ $v_i$ ” jointly.

The problem now turns out to be the minimization of the mean square error:

$$\min_{w,v} (\hat{E}_D[(y - f(x|w,v))^2])$$

Where the previous problem with Kernel functions was in this form:

$$\min_{\theta} \left( \sum_{i=1}^n \left( y^i - \sum_{j=1}^n \theta_j \cdot K(x^i, x^j) \right)^2 \right) \propto \min_{\theta} \left( E_D \left[ \left( y^i - \sum_{j=1}^n \theta_j \cdot K(x^i, x^j) \right)^2 \right] \right)$$

Where " $E_D$ " is the empirical average or expectation.

**Note:** the expressions are proportional as we are missing constant term " $1/n$ " in the righthand expression.

**Note 2:** " $1/n$ " is not meaningful in the optimization problem, as the minimum point will not depend on " $1/n$ " when deriving and setting the gradient to zero.

The general idea behind neural network is nothing complicated. It just takes to parametrize the basis functions using some simple parametric functions and, then, jointly learn.

In order to implement a neural network we need to parametrize the basic functions and we need to make de decision which shape will these basis functions take.

The way we parametrize is particularly simple in the case of neural networks:

$$\varphi_i(x) = \sigma \left( \sum_{l=1}^d v_l \cdot x_l + v_0 \right)$$

Where:

$$x = \begin{Bmatrix} x_1 \\ \vdots \\ x_d \end{Bmatrix}_{dx1}, \text{each point has d features}$$

The basis functions are applied as a linear function over the features like we did for linear regression.

$$v = \begin{Bmatrix} v_1 \\ \vdots \\ v_d \end{Bmatrix}_{dx1}, \text{each point has d features}$$

Then, the " $\sigma$ " function takes the sum over each coordinate (or feature) in order to get a weighted sum of features that returns a single number. After we get this number, we apply an arbitrary, simple nonlinear function " $\sigma$ ".

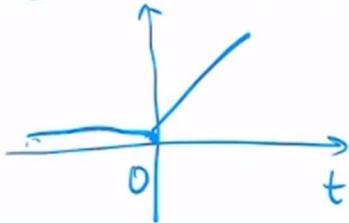
This " $\sigma$ " function is called the **activation function**.

### Commonly used activation functions

#### *RELU – Rectified Linear Unit*

$$\sigma(t) = \max(0, t)$$

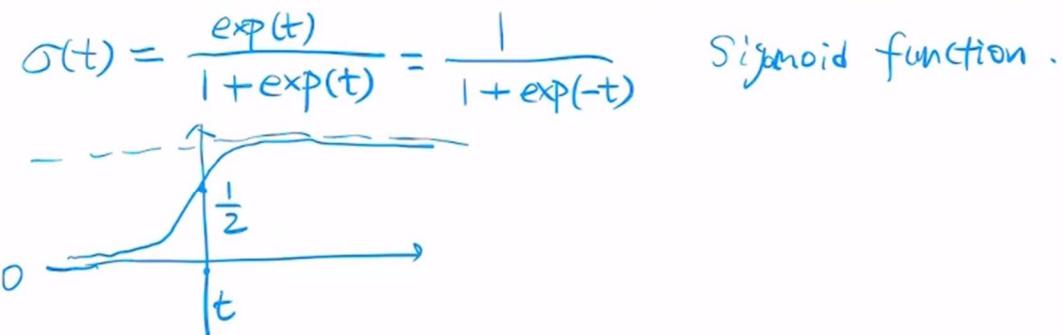
$\sigma(t) = \max(0, t)$  : Rectified Linear Unit (ReLU)



This function specifies that if "t" smaller than zero then set it to zero, otherwise, set it to "t".

Sigmoid

$$\sigma(t) = \frac{e^t}{1 + e^t} = \frac{1}{1 + e^{-t}}$$



The difference between RELU and sigmoid is that RELU will continue to grow whereas sigmoid is bounded.

Thresholding function

$$\sigma(t) = 1 \text{ if } x > 0$$

$$(t) = 0 \text{ if } x \leq 0$$

$$\sigma(t) = \mathbb{I}(x > 0) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$



In the case of neural network each of these basic functions is called a neuron.

Building the neural network

In the case of neural network each of these basic functions is called a neuron.

$$\varphi_i(x) = \varphi(x, v_i): \text{Neuron}$$

$\sigma$ : activation function

$v_i$ : weights of the neural network

Here we have, each of the “i” elements above represents a Neuron. Each neuron has a different parameter “ $v_i$ ”. Summing them together gives us the overall neural network.

The neural network consists of the linear combination of “m” Neurons:

$$f(x|w, v) = \sum_{i=1}^m w_i \cdot \sigma(v_{il} \cdot x_l + v_{i0})$$

Where each “ $v_i$ ” is a vector “1xd”. The overall parameter “v” of all neurons is actually a matrix “d xm”:

$$v_i = \{v_{i0} \quad \dots \quad v_{id}\}_{1 \times d}$$

$$v = \begin{bmatrix} v_{10} & \dots & v_{m0} \\ \vdots & \ddots & \vdots \\ v_{1d} & \dots & v_{md} \end{bmatrix}$$

Where “m” is actually a parameter that we need to decide as a user. “m” controls the complexity of the model. The more neurons the more complex but more flexible function.

We also have a set of linear parameters “ $w_i$ ”.

Finally, we train the neural network and we will write down the loss function plugging the definition. This gives us a function for both “w” and “v” that we can numerically optimize via gradient descent for example.

The optimization is a complicated part with not a closed form but we still hope to solve it using gradient descent. In fact, in practice when you have large data set is actually sufficient to get a reasonably good estimation of the parameters.

#### Optimizing the neural network: non-convexity

The optimization of the neural network is always non-convex. We can only hope to find a local minimum when we want to solve the problem.

**Fact:** learning Neural network is a non-convex optimization.

Let us assume we have two neurons with one single parameter per neuron (no weight “w” parameters).

$$f(x|w, v) = f(x | v) = \sigma(x - v_1) + \sigma(x - v_2)$$

Even for this simple neural network solving the optimization is actually non-convex.

$$L(v) = E_D \left[ (y - \sigma(x - v_1) + \sigma(x - v_2))^2 \right]$$

Assume at some point we have an optimal solution:

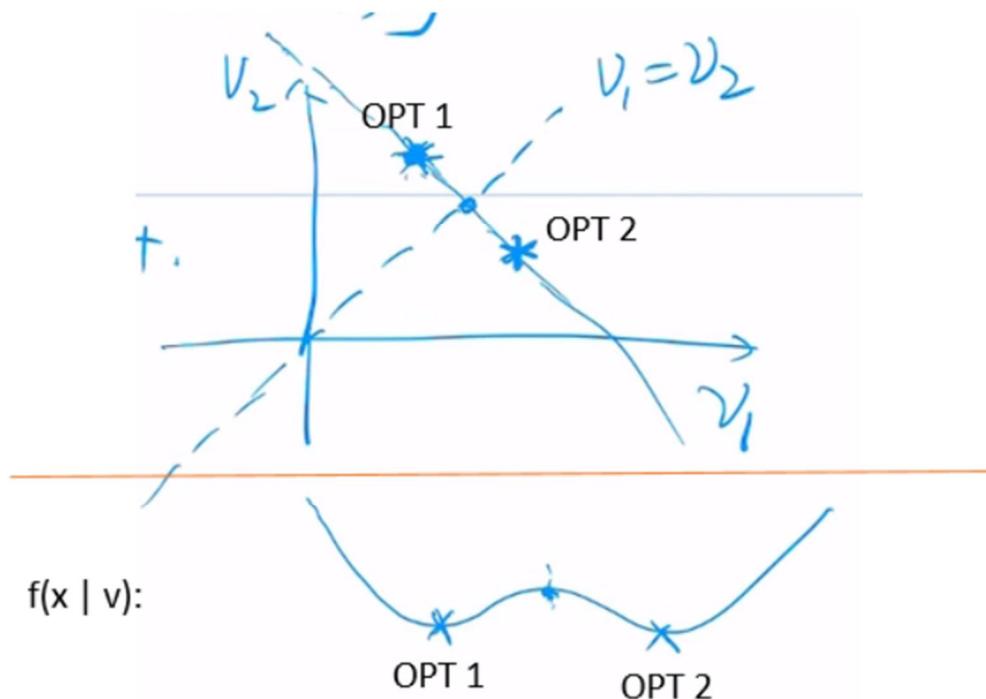
$$[v_1^*, v_2^*] \text{ is optimal}$$

Because the neurons are symmetric we can permute them arbitrarily:

$[v_2^*, v_1^*]$  is also optimal

If a point “ $p(v_1, v_2)$ ” is optimal then “ $p(v_2, v_1)$ ” is also optimal regardless the configuration of the neurons.

Connecting the two optimal points we get a one dimensional function. So if the loss function is actually convex then it has to be convex along this line.

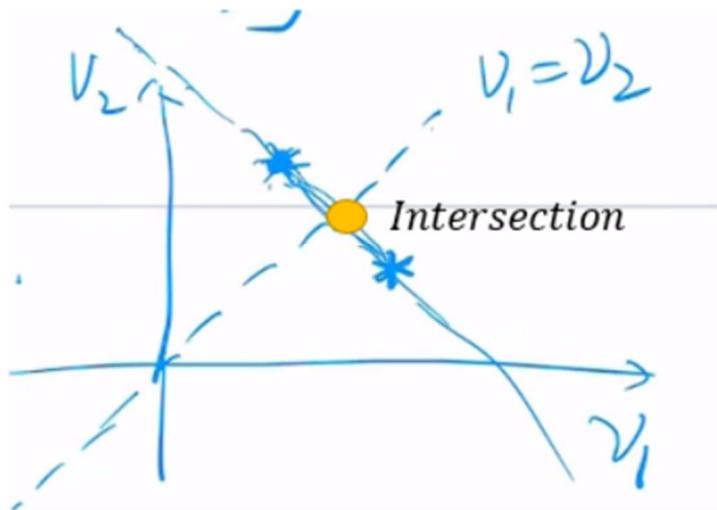


In this case, the Loss function is not convex. If the Loss function is convex then any convex combination of these two symmetric points has to be also optimal.

If the loss function “ $L(v)$ ” is convex then  $[v_1^*, v_2^*]$  are optimal as well as:

$$v^* = \frac{v_1^* + v_2^*}{2}$$

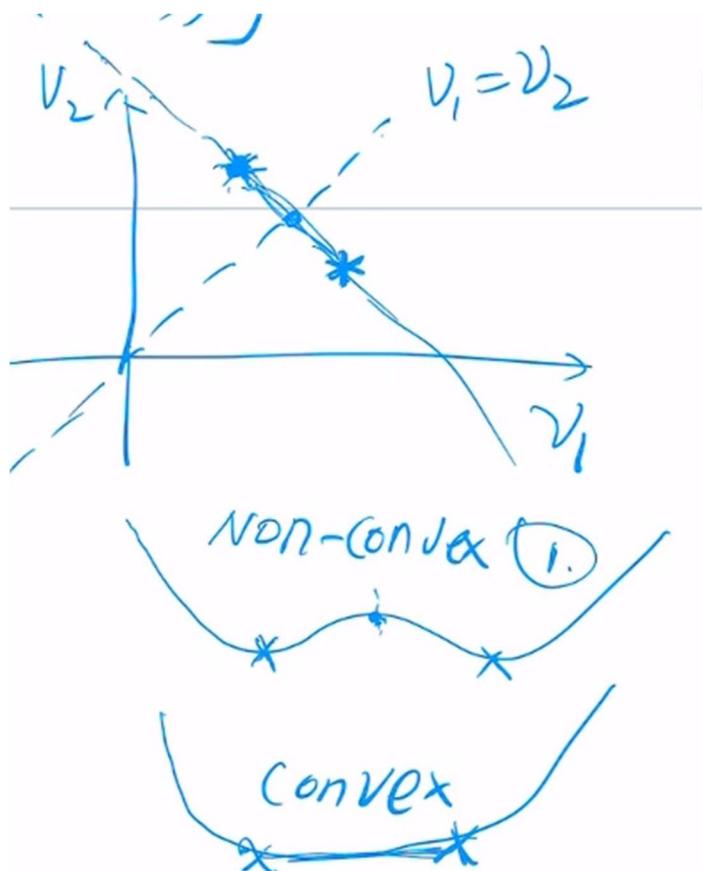
Which is the intersection point:



This is actually unlikely, as these symmetric points for these two neurons there is the same value. The idea of using multiple neurons is to capture different patterns to get an increasingly flexible. If we restrict these 2 neurons to have the same values (to be the same neuron) we are restricting the function class.

In general, it is unlikely that when we restrict to 2 neurons to be the same this is actually an optimal point as the loss function " $L(w)$ " is unlikely to be convex.

In reality, the slicing of these two points tends to be the first case below: non-convex.



In practice, we can still solve the optimization problem using gradient descent and that allows us to find a local optimum that is sufficient for practice.

#### Solving the optimization problem

In gradient descent, the problem to solve is as follows:

$$\min_{w,v} L(w, v)$$

In order to solve this problem we update the parameters following the negative gradient direction.

These days we have automatic differentiation tools that we can use to solve the gradient descent.

We need to keep in mind that when implementing gradient descent the initialization is very critical. **People normally tend to initialize parameters “v” to sort of a small random value** rather than a “v” obtained deterministically or a large random variable as the gradient descent is easy to get stuck.

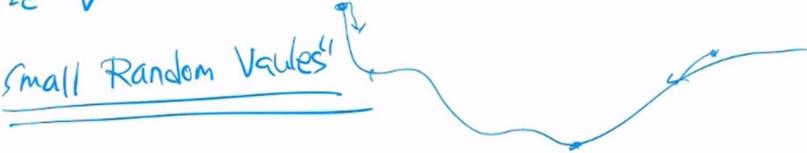
## Gradient descent

$$\min_{W, V} L(W, V)$$

$$\begin{cases} W \leftarrow W - \epsilon \nabla_W L(W, V) \\ V \leftarrow V - \epsilon \nabla_V L(W, V) \end{cases}$$

Initialize  $V$

to be "small Random Values"



This initialization to small random values is key for Sigmoid functions. Initializing to a large value the function is already saturated, and this makes it very difficult to apply gradient descent as the gradient is already almost zero and the function to optimize is almost flat.

On the other hand, initializing the weights to be close to zero then in the beginning we get very large slope and this allows us to move faster using gradient descent.

Furthermore, at the initialization we need to make sure that the parameter of different neurons is initialized differently. The reason for this is simple, if we have two neurons and we initialize them to the same value, they will keep being the same all the time. That is because if we keep "v" the same value then the gradient for both neurons is also the same. In this case, they are just symmetric and the same unless we are lucky to get some random noise different added for each neuron.

Overall, training a neural network is kind of empirical work where we need to be careful about initialization and the step-size as it can influence the results.

Training many neural network despite the non-convexity we can state that the optimal point derived from gradient descent is actually not far away from the global optimum.

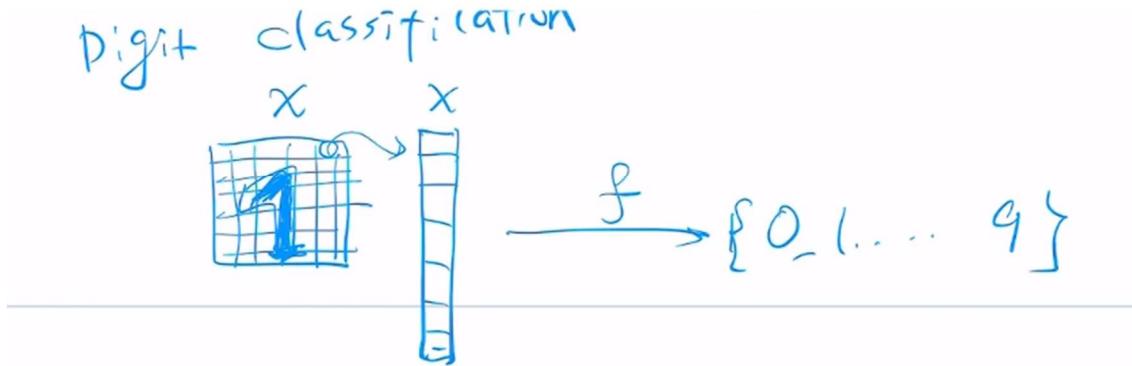
### Justification of Neural networks

#### Example digit classification

Let us assume we have handwritten images of digits and we want to classify that given a collection of pixels (an image) we predict if it is number 0,1,2,3,...,9. Each pixel has a value that indicates the color or the brightness of the image. The image will be actually a vector of pixels.

Summarizing: given an input we want to decide output: a class of the image.

When using neural network it is almost like using a sequence of templates where each of the neurons is a template.



Each of the weights ("vi") is also a vector. The value of these weights is going to be estimated by minimizing the loss. However, let us assume that we can sort of set the weights to some realistic digits.

In this example, we make the weights of one neuron so it is similar to digit 2. Then, some other neuron may have a shape that is very similar to digit 1. Finally, let us assume we have 10 different neurons and each neuron has a weight to represent that particular digit.

What we are doing is taking the input "x" and then we take the inner product with each of the 10 templates.

$$\sum_{i=1}^m w_i \cdot \sigma(v_i^T \cdot x - v_0)$$

With the thresholding activation function (example):

$$\sum_{i=1}^m w_i \cdot I(v_i^T \cdot x \geq v_0)$$

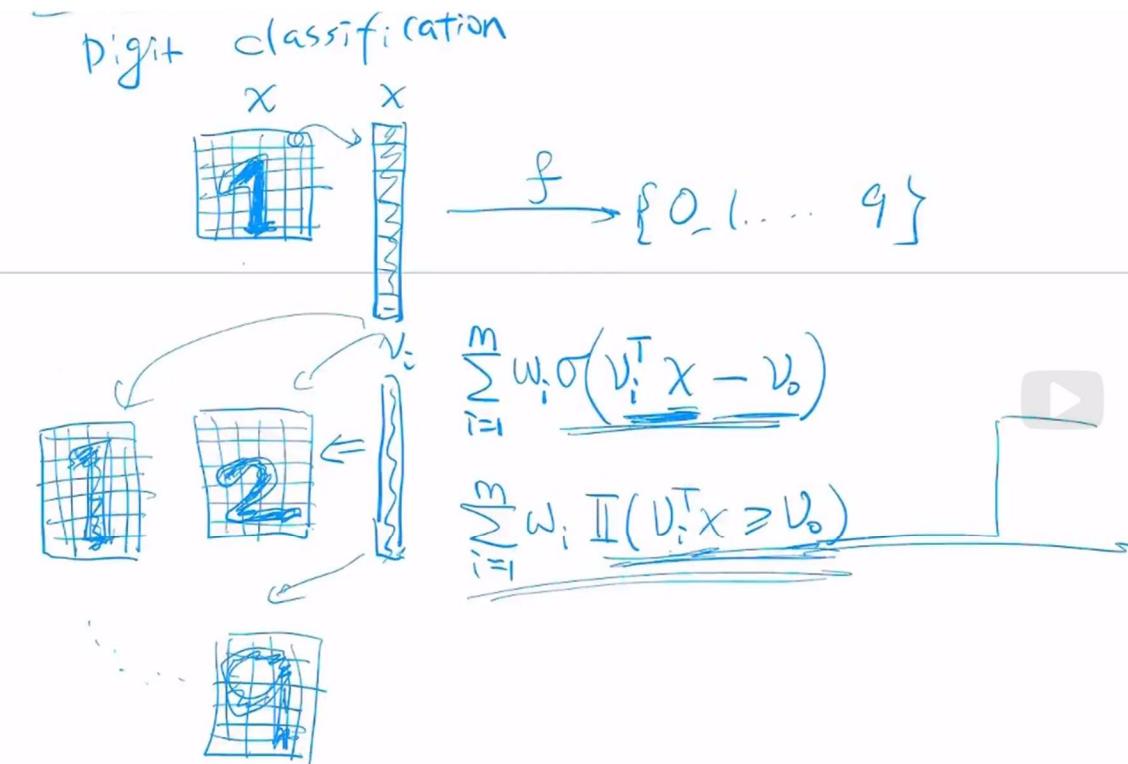
What is happening is the following:

1. Take the input.
2. Take inner product with each of the templates.
3. Apply the minus bias term that basically shifts the activation a little bit.
4. Apply the activation functions: assume thresholding activation function: if the inner product is larger than " $v_0$ " then the neuron fires "1", otherwise "0".
5. These activations are summed together applying some weight to give the final output.

Let us assume that our input is "1":

We compare this input with the 10 different templates: 0,1,2,...,9. In this case we find that the first Neuron has largest similarity and the Neuron fires for template "1" and does not fire for the others.

This idea of comparing different templates and then picking the one that has this largest similarity allows the neural network detect different patterns.



Turns out that this definition of a neural network is fairly flexible: if we have an infinite number of neurons we can actually approximate arbitrary non-linear functions. And that brings new possibilities.

**Claim:**

- If we have infinite number of Neurons: “m” is large. Then, the neural network “ $f(x|w,v)$ ” can approximate any continuous function.

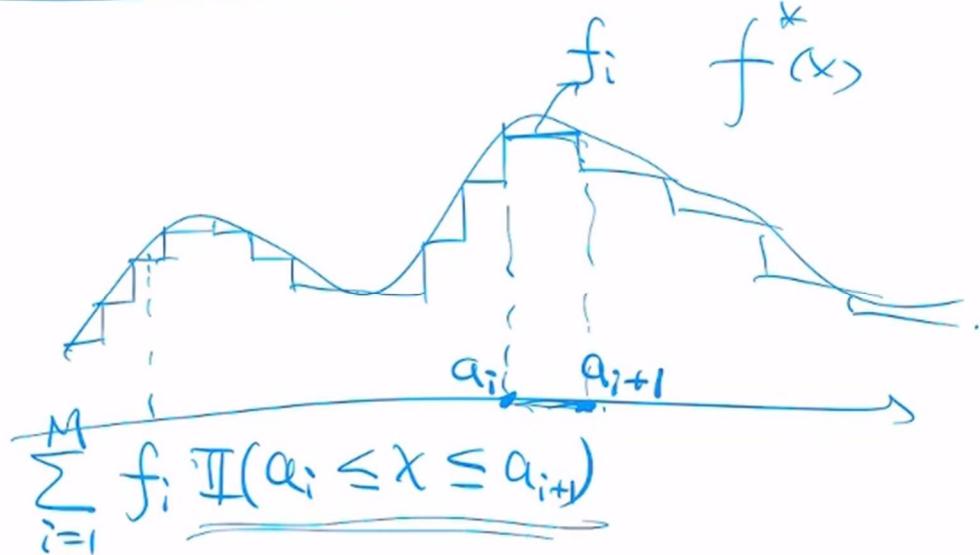
Let us say, in 1-dimensional case we have some arbitrary function and we have a true function “ $f^*(x)$ ” that we want to approximate. No matter how complex the function is, once we allow to have a large enough amount of Neurons, we can approximate any arbitrary complex function.

The reason is simple, we can always approximate this function using a step function that uses a bunch of step functions. This step function can be written as a linear combination of the thresholding functions.

Let us say we have each of the data points for each of the thresholding functions:

$$f(x) \sim \sum_{i=1}^m f_i \cdot I(a_i \leq x \leq a_{i+1})$$

So if “ $x$ ” is between some value “ $a_i$ ” and “ $a_{i+1}$ ” we know the approximately value it takes in this interval.



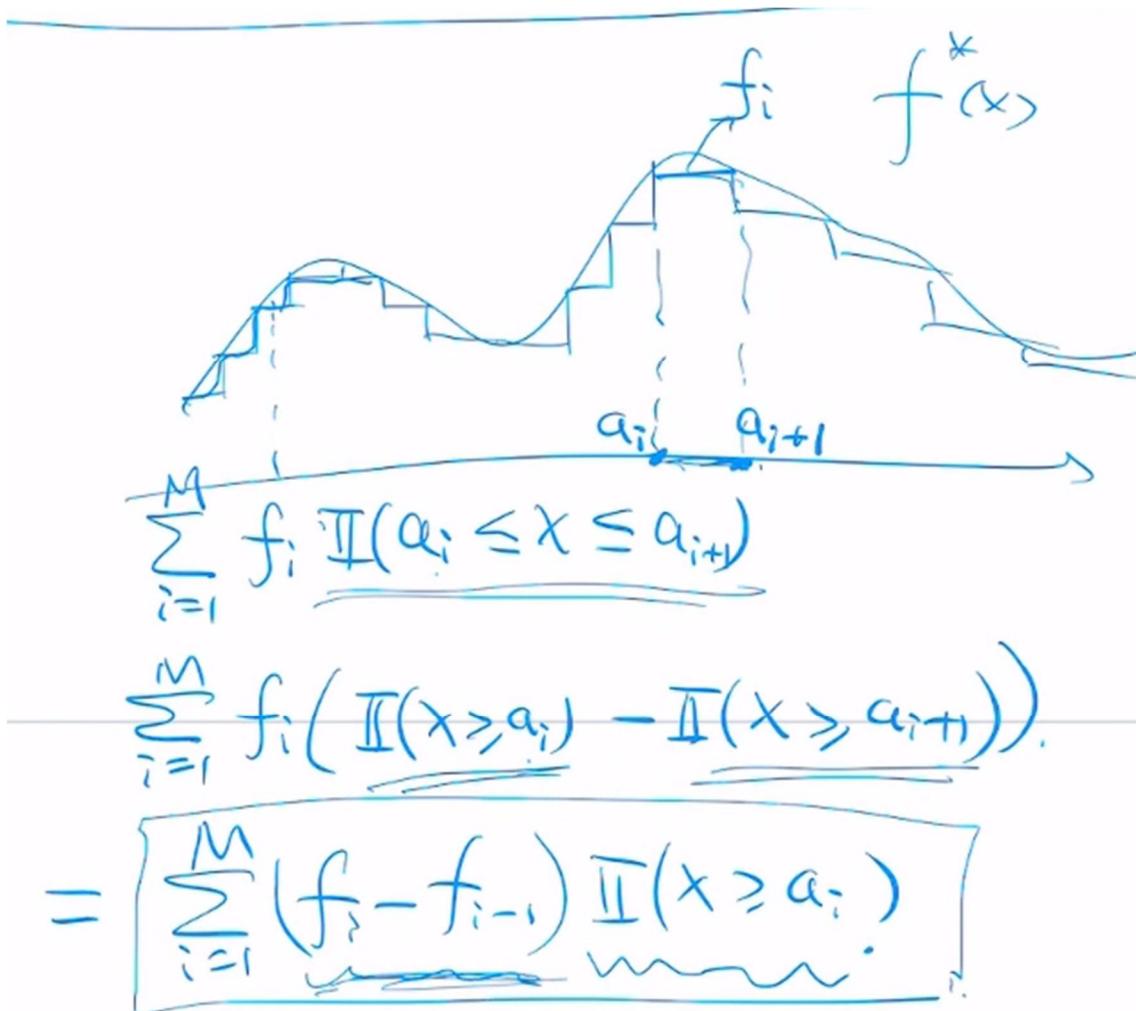
If we take more and more points, we can approximate the function.

This function can also be written as a combination of binary activation functions:

$$\begin{aligned} f(x) &\sim \sum_{i=1}^m f_i \cdot I(a_i \leq x \leq a_{i+1}) = \sum_{i=1}^m f_i \cdot (I(x \geq a_i) - I(x \geq a_{i+1})) \\ &= \sum_{i=1}^m (f_i - f_{i-1}) \cdot (I(x \geq a_i) - 0) \end{aligned}$$

The last expression is the actual neural network: for any arbitrary function we can always handcraft a function which is actually a neural network.

This means, there must exist some parameter that can approximate a continuous function arbitrarily well.



Same arguments work for different activation functions, like the sigmoid functions: smooth step functions or RELU (they just improve the values in each interval-neuron).

#### Representing neural network as a graph

Many people tend to represent neural network using graphs. It turns out that graphical representation of neural networks is actually to the graphical model of Gaussian-markov model.

The idea is that we can represent each of the variables using a node and represent the dependency between different variables using connections.

Let us say we start from the simplest case: linear regression.

$$y = \sum_{l=1}^d v_l \cdot x_l$$

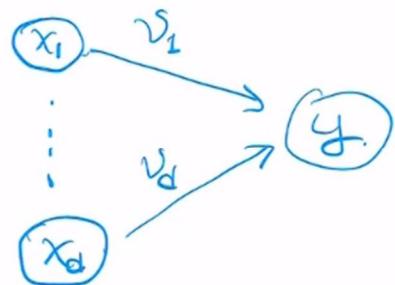
Using a graph where each of the nodes represent each of the dimensions of the feature. Then there is "y" which is a function that takes inputs from all dimensions to return a value.

Nodes: dimensions of features.

The arrows represent each of the terms in the linear combination.

Arrows/edges: parameters.

$$\underline{y} = \sum_{l=1}^d v_l \underline{x}_l$$



For neural network this is similar, the main difference is that we have more nodes:

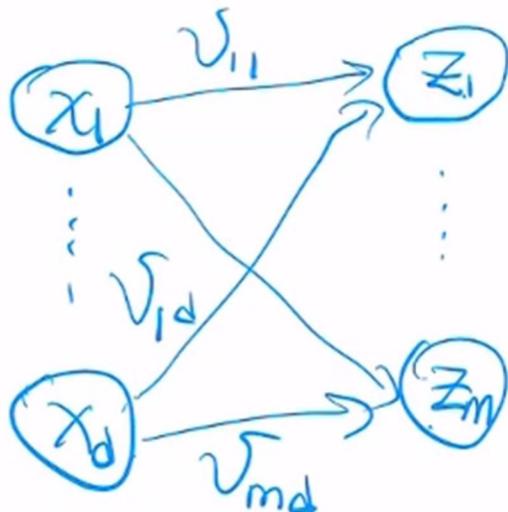
$$y = \sum_{i=1}^m w_i \cdot \sigma \left( \sum_{l=1}^d v_l \cdot x_l + v_0 \right) = \sum_{i=1}^m w_i \cdot z_i =$$

**Note:** "m" represent the nodes whereas "d" represent the inputs.

In this case " $\{x_1, x_2, \dots, x_d\}$ " represent the input.

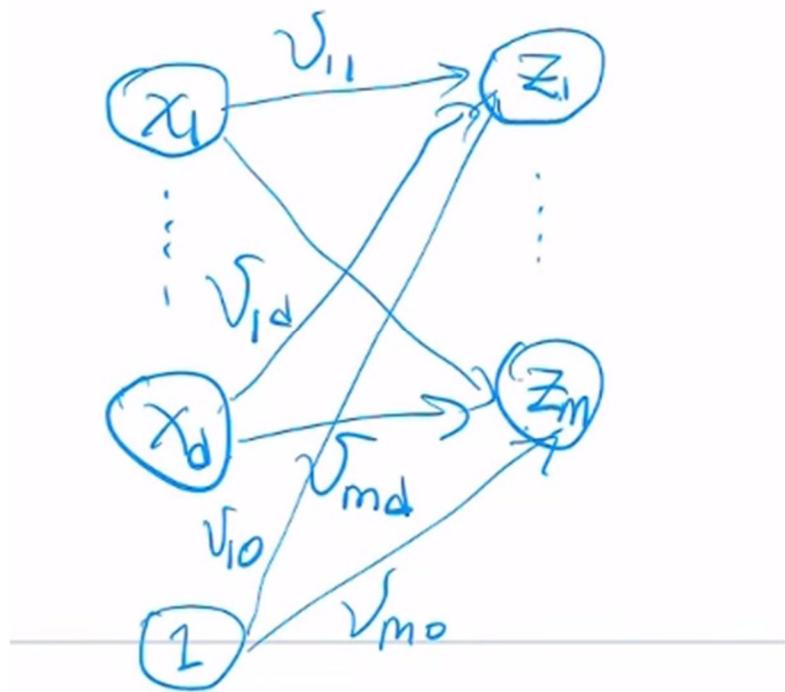
" $\{z_1, z_2, \dots, z_m\}$ " represent each of the neurons (nodes).

The connections are represented via the "v" matrix where element " $v_{11}$ " represents the connection from " $x_1$ " to " $z_1$ ".

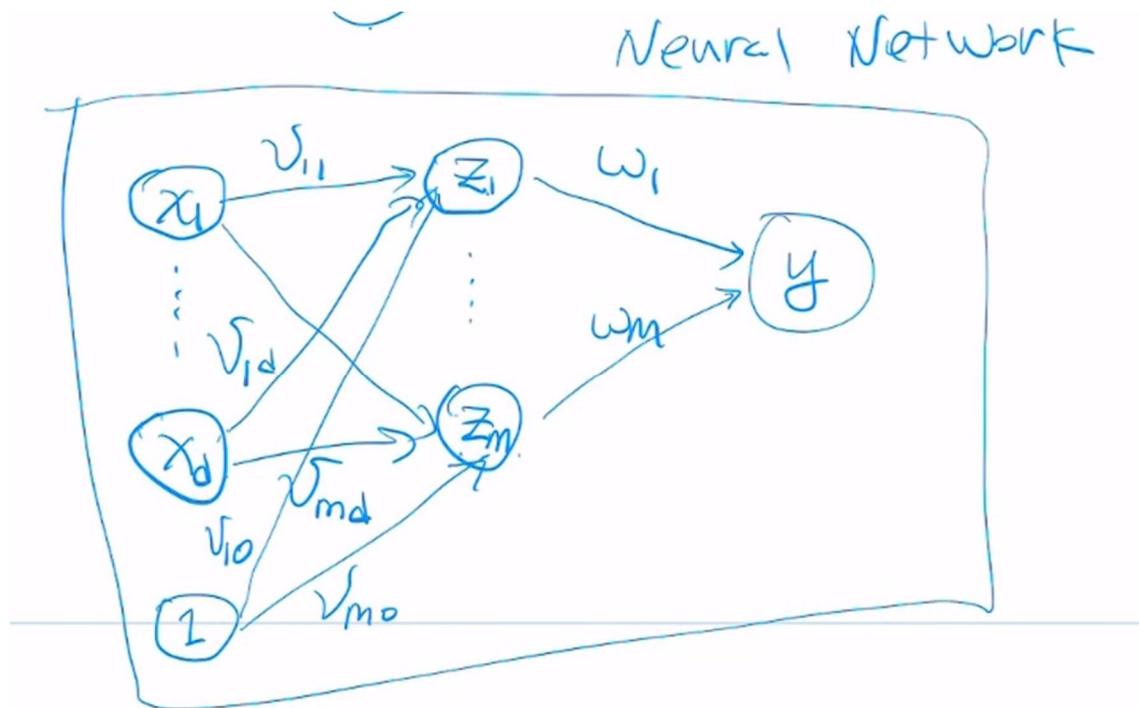


In this case, we have a fully connected bipartite graph where each of the neurons is connected.

Obviously, we should also represent the bias term: " $\{v_{10}, v_{20}, \dots, v_{m0}\}$ " although normally we ignore the bias term when plotting the neural network.

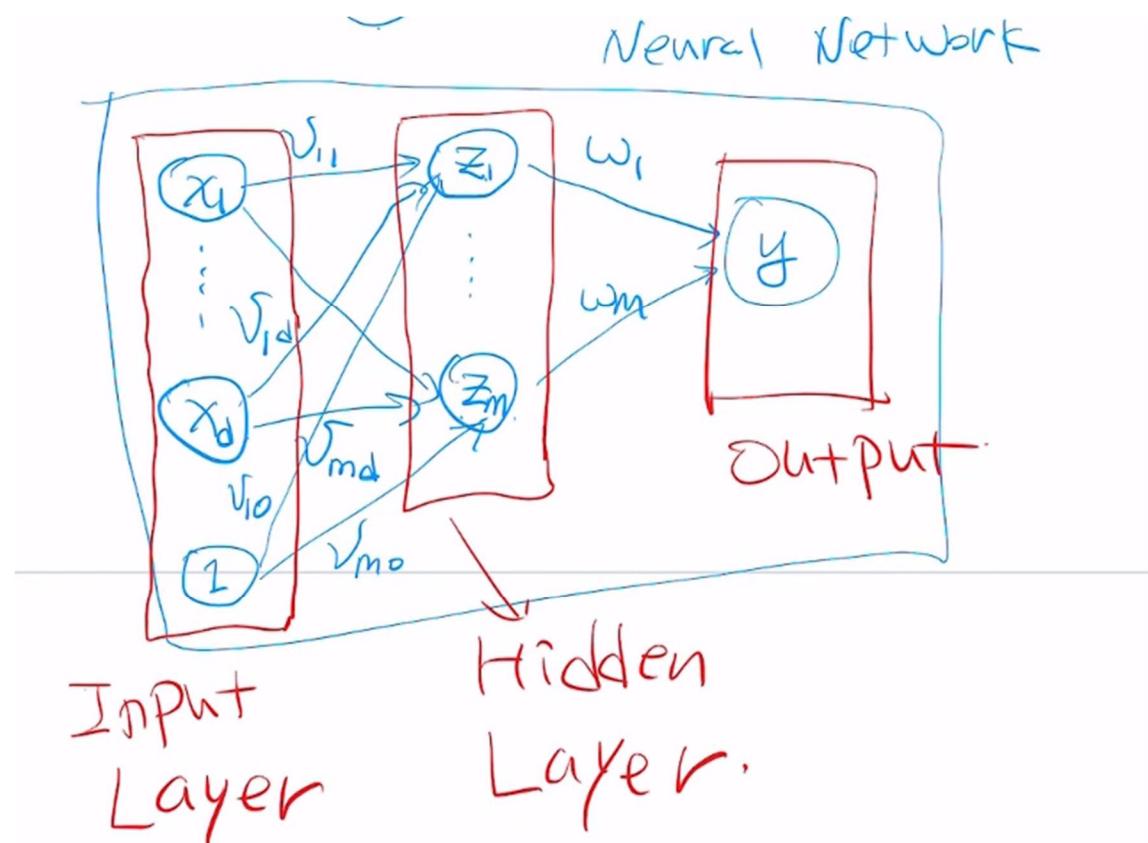


And then we have “ $\{z_1, z_2, \dots, z_m\}$ ” connected to the output “ $y$ ” which is called a neural network:



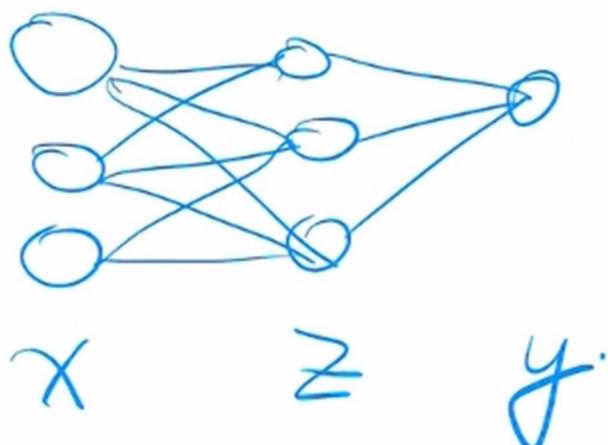
In this neural network we have different layers:

- Input layer: “ $\{x_1, x_2, \dots, x_d\}$ ”
- Hidden layer: “ $\{z_1, z_2, \dots, z_m\}$ ”
- Output: “ $y$ ”



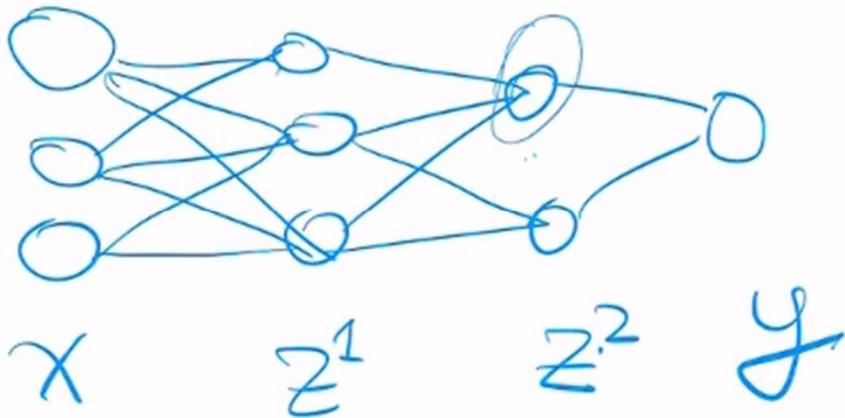
### Deep neural networks

Given the previous examples, we can extend this graph to use larger and deeper layers. The case above was a case for one hidden layer. When we insert multiple hidden layers that gives us a **deep neural networks**.



This is one of the most powerful tools in machine learning and artificial intelligence these days. The idea is to have multiple layers that we can sort of plug in the neural network.

Deep neural networks can extend the simple graph above with more hidden layers. So instead of outputting "y" after "z" as feature we have another layer of neural network.



Previously, we mapped the input “x” to some latent variables “z”. Now “z<sup>1</sup>” is mapped to another set of latent variables “z<sup>2” prior to “y”.</sup>

“z<sup>1” becomes a new set of features that is treated as “x”.</sup>

If we are not satisfied with this we can carry on adding new layers to the neural network. Different layers can have different number of neurons so the network can grow very complex.

We can end up having millions of hundreds of layers in the state of art computer vision or natural language processing in order to capture complex patterns in the data set.

#### *Mathematical background*

The idea behind is that:

$$z^1 = \phi(x | v^1) = \begin{Bmatrix} \sigma((v_1^1)^T \cdot x + v_0^1) \\ \vdots \\ \sigma((v_{m1}^1)^T \cdot x + v_0^1) \end{Bmatrix}$$

Where “ $\sigma((v_1^1)^T \cdot x + v_0^1)$ ” means that we take the first neuron of the first layer and then multiply it with the input and apply the sigmoid activation function. This gives us the first hidden node in the first layer.

Overall, we have “m1” neurons in the first layer. The second layer has “m2” neurons and the third layer has “m3” neurons.

From the input to the first layer, we just take a bunch of linear functions and map it using the activation functions: this is the first layer.

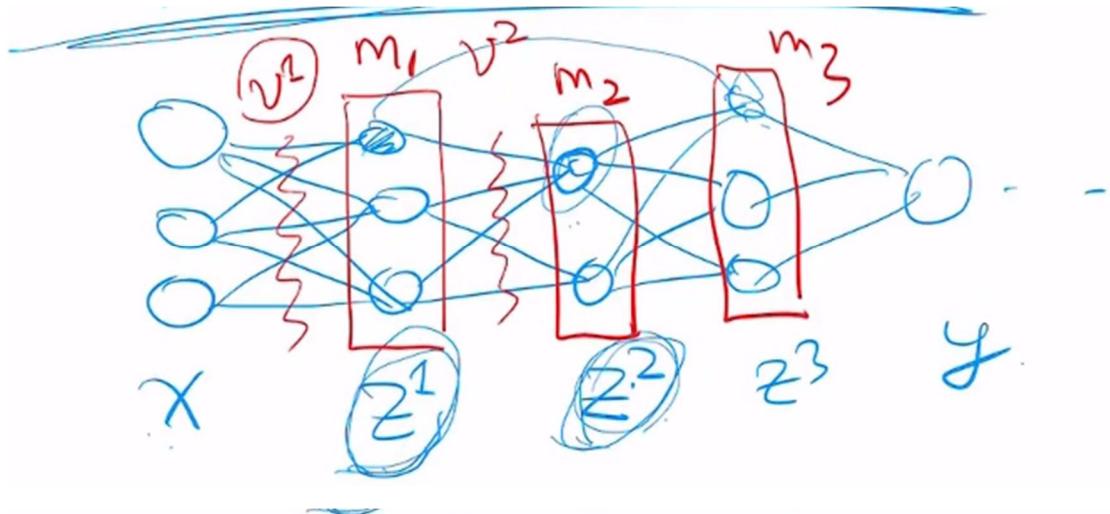
Then, the other layers are defined in the same way. For example the second layer:

$$z^2 = \phi(z^1 | v^2) = \begin{Bmatrix} \sigma((v_1^2)^T \cdot z^1 + v_0^2) \\ \vdots \\ \sigma((v_{m2}^2)^T \cdot z^1 + v_0^2) \end{Bmatrix}$$

This gives us the second hidden note.

If we repeat this for several times we will get the output:

$$y = \sum_{i=1}^{m_3} w_i \cdot z_i^3$$



$$z^1 = \Phi(x, v) = \begin{bmatrix} \sigma((v_1^1)^T x) \\ \vdots \\ \sigma((v_{m_1}^1)^T x) \end{bmatrix}$$

$$z^2 = \Phi(z^1, v^2) = \begin{bmatrix} \sigma((v_1^2)^T z^1) \\ \vdots \end{bmatrix}$$

$$y = \sum_{i=1}^{m_3} w_i z_i^3$$

Defining a deep neural network becomes complicated because we have to write many formulas. Fortunately, modern deep learning platforms allow us to specify the structure of neural networks using some simple language or configuration so we do not need to write down all these equations.

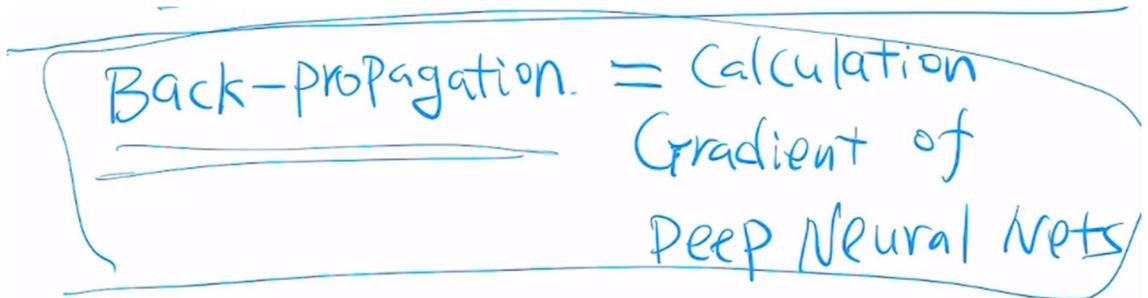
*Training and solving – back propagation*

Again the way to train deep neural networks is again using gradient descent the same as before except that we need to take gradients of this huge neural network.

We are going to use automatic differentiation tools in order to solve these problems.

One concept worth mentioning is that when we calculate the gradient of a deep neural network it is almost back-propagating the error from the output to the input, this is what people call **back-propagation**.

When talking about back-propagation it refers to calculating gradient of deep neural networks.



The reason why it is called back propagation is a simple reason.

Let us say we have a very simple neural network where each layer has only one neuron. In this network we have parameters " $\{v^1, v^2, \dots, v^m\}$ " for each layer. Obviously " $v^m = w$ ".

Then the Loss function:

$$L(v) = E_D[(y - f(x | v^1, v^2, \dots, v^m))^2] = E_D[(y - f(x | v))^2]$$

Let us take the gradient for " $v^1$ " applying chain rule to this function:

$$\text{grad}_{v_1} L(v) = 2 \cdot E_D \left[ f(x|v) \cdot \frac{df(x|v)}{dv^1} \right]$$

Whereas by the chain rule:

$$\frac{df(x|v)}{dv^1} \approx \frac{dy}{dv^1} = \frac{dy}{dz^m} \cdot \frac{dz^m}{dz^{m-1}} \cdot \dots \cdot \frac{dz^3}{dz^2} \cdot \frac{dz^2}{dz^1} \cdot \frac{dz^1}{dv^1}$$

This derivative depends always on its immediate parents, which goes through " $z^m$ " to " $z^{m-1}$ " all the way up to " $z^1$ " and then " $v^1$ ". Therefore, we need to calculate all the terms: this corresponds to **back-propagating**.

$$L(V) = \mathbb{E}_D[(y - f(x, \underbrace{V^1 \dots V^m}_{V}))^2]$$

$$\nabla_{V^2} L(V) = 2 \mathbb{E}_D[(f(x, V) - y) \frac{\partial f(x, V)}{\partial V^2}]$$

$$\frac{\partial f(x, V)}{\partial V^1} = \frac{\partial y}{\partial V^1} = \left( \frac{\partial y}{\partial z^m} \right) \left( \frac{\partial z^m}{\partial z^{m-1}} \right) \dots \left( \frac{\partial z^3}{\partial z^2} \right) \left( \frac{\partial z^2}{\partial z^1} \right) \left( \frac{\partial z^1}{\partial V^1} \right)$$

I strongly encourage you to look at some of the code and study some of the basic frameworks for deep networks such as TensorFlow or PyTorch.

## Statistical distributions

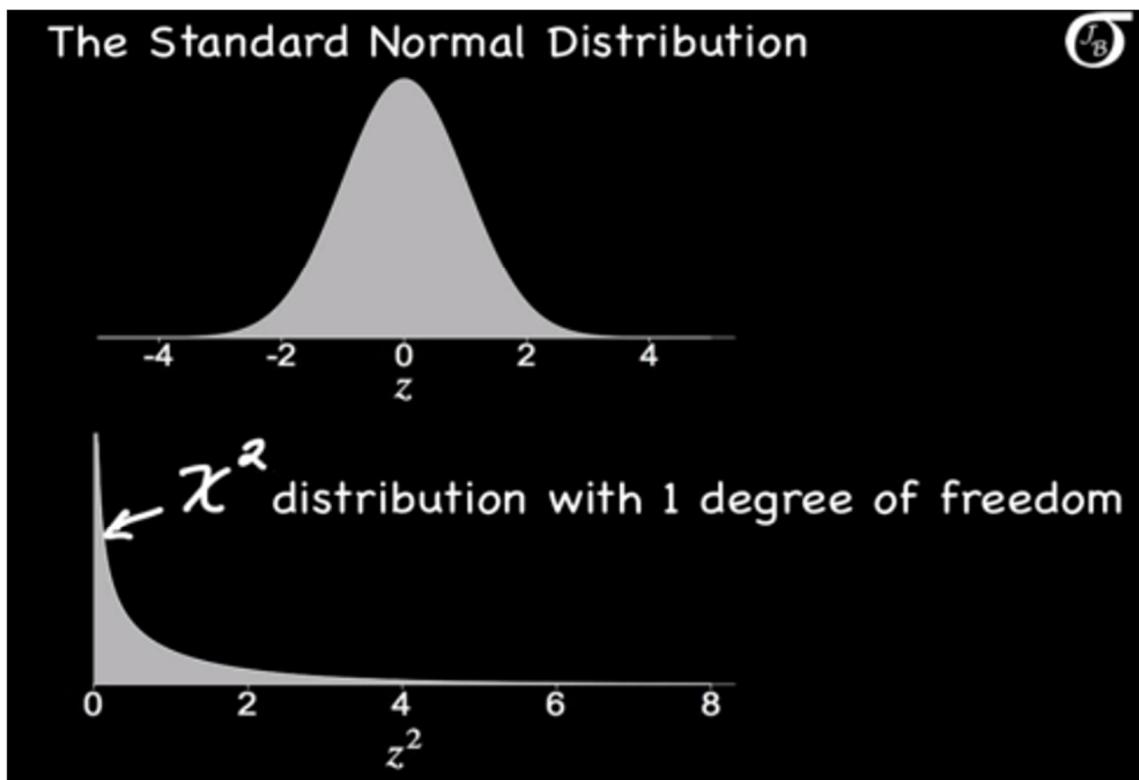
### $\chi^2$ Distribution

It is a continuous probability distribution that is widely used in statistical inference.

It is related to the standard normal distribution:

If a random variable "Z" has the standard normal distribution, then "Z<sup>2</sup>" has the  $\chi^2$  distribution with one degree of freedom.

**Degrees of freedom:** random variables or features to analyze.



Furthermore:

If  $Z_1, Z_2, \dots, Z_k$  are **standard normal random variables**

$\rightarrow Z_1^2 + Z_2^2 + \dots + Z_k^2$  follows a  $\chi^2$  distribution with  $k$  degrees of freedom.

In order to normalize a standard normal distribution to have “0” mean “ $\mu$ ” and “1” standard deviation “ $\sigma$ ”:  $N(\mu, \sigma^2) \rightarrow N(0,1)$  we do the following:

$$Z = \frac{X - \mu}{\sigma}$$

**Note:** This formula applies when we know the entire population we want to study and therefore we can calculate the true mean “ $\mu$ ” and true standard deviation “ $\sigma$ ”. In case we are studying a set of observations of a random variable, and the true value of “ $\sigma$ ” is unknown we need to consider the “t-distribution” that takes into account the uncertainty of using “ $s$ ” instead of “ $\sigma$ ” when the true underlying distribution is Gaussian (although with unknown “ $\sigma$ ”).

**Note 2 usages:**

- This distribution assumes “ $k$ ” variables are random and follow a standard normal distribution independently from each other. If the variables depend on each other then they do not follow a  $\chi^2$  distribution.
- It can also be used to study if multiple excluding features of the population to study (i.e. being deaf and poor) follow an expected distribution/frequency or not, in this case the  $\chi^2$  formula can be obtained bearing in mind “ $k$ ” is the sum of all combinations for different values that the features can take:

$$\chi^2 = \sum_{i=1}^k \frac{(\text{#observed} - \text{#expected})^2}{\text{#expected}} = \sum_{i=1}^k \frac{(n_i - e_i)^2}{e_i} \sim \left( \frac{x - \mu}{\sigma} \right)^2 ; \text{ where } \mu = e_i, \sigma = \sqrt{e_i}$$

Where:

**#observed:** amount of observed members of the populations with the “k” combination of features.

**#expected:** amount of expected members of the population to have the “k” combination of features.

#### EJEMPLO:

Para estudiar la dependencia entre la práctica de algún deporte y la depresión, se seleccionó una muestra aleatoria simple de 100 jóvenes, con los siguientes resultados:

|               | Sin<br>depresión | Con<br>depresión |
|---------------|------------------|------------------|
| Deportista    | 38               | 9                |
| No deportista | 31               | 22               |

Determinar si existe independencia entre la actividad del sujeto y su estado de ánimo. Nivel de significación (5%)

#### SOLUCIÓN:

Debemos primero calcular las frecuencias esperadas bajo el supuesto de independencia. La tabla de frecuencias esperadas sería:

|               | Sin<br>depresión | Con<br>depresión |     |
|---------------|------------------|------------------|-----|
| Deportista    | 32.43            | 14.57            | 47  |
| No deportista | 36.57            | 16.43            | 53  |
|               | 69               | 31               | 100 |

Calculamos ahora el estadístico del contraste:

$$\chi^2 = \frac{(38 - 32.43)^2}{32.43} + \frac{(9 - 14.57)^2}{14.57} + \frac{(31 - 36.57)^2}{36.57} + \frac{(22 - 16.43)^2}{16.43} = 5.82$$

The probability density function of the  $\chi^2$  distribution is as follows:

$$p(x \geq 0) = \frac{x^{\frac{k}{2}-1} \cdot e^{-\frac{x}{2}}}{\frac{k}{2} \cdot \Gamma\left(\frac{k}{2}\right)}$$

Where “ $\Gamma$ ” is the gamma function:

$$\Gamma(z) = \int_0^\infty x^{z-1} \cdot e^{-x} \cdot dx = (z-1)!$$

Example for “z=2”:

$$\begin{aligned} \Gamma(3) &= \int_0^\infty x^{3-1} \cdot e^{-x} \cdot dx = [-x^2 \cdot e^{-x}]_0^\infty + \int_0^\infty 2 \cdot x \cdot e^{-x} \\ &= [-x^2 \cdot e^{-x} - 2 \cdot x \cdot e^{-x}]_0^\infty \\ &+ \int_0^\infty 2 \cdot e^{-x} = [-x^2 \cdot e^{-x} - 2 \cdot x \cdot e^{-x} - 2 \cdot e^{-x}]_0^\infty \\ &= (0 + 0 + 0) - (0 + 0 - 2) = 2 = (3-1)! \end{aligned}$$

Bear in mind we only consider positive values as the variables are squared and there are only positive values possible.

The degree of freedom is also a positive value normally.

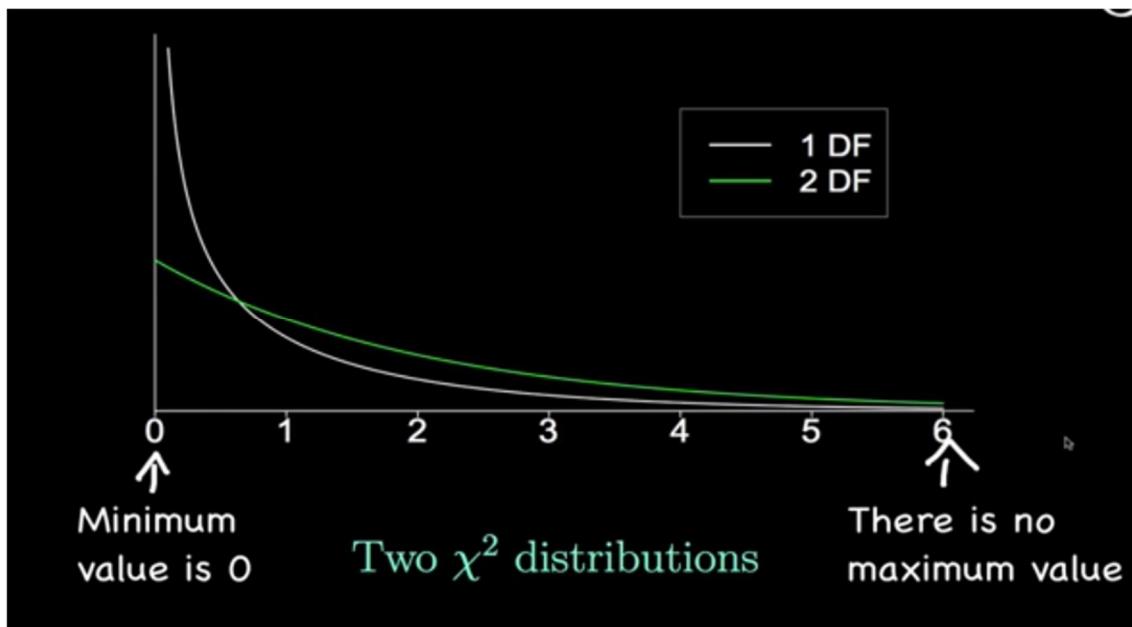
Average and standard deviation of the  $\chi^2$  distribution are as follows:

$$\mu = k$$

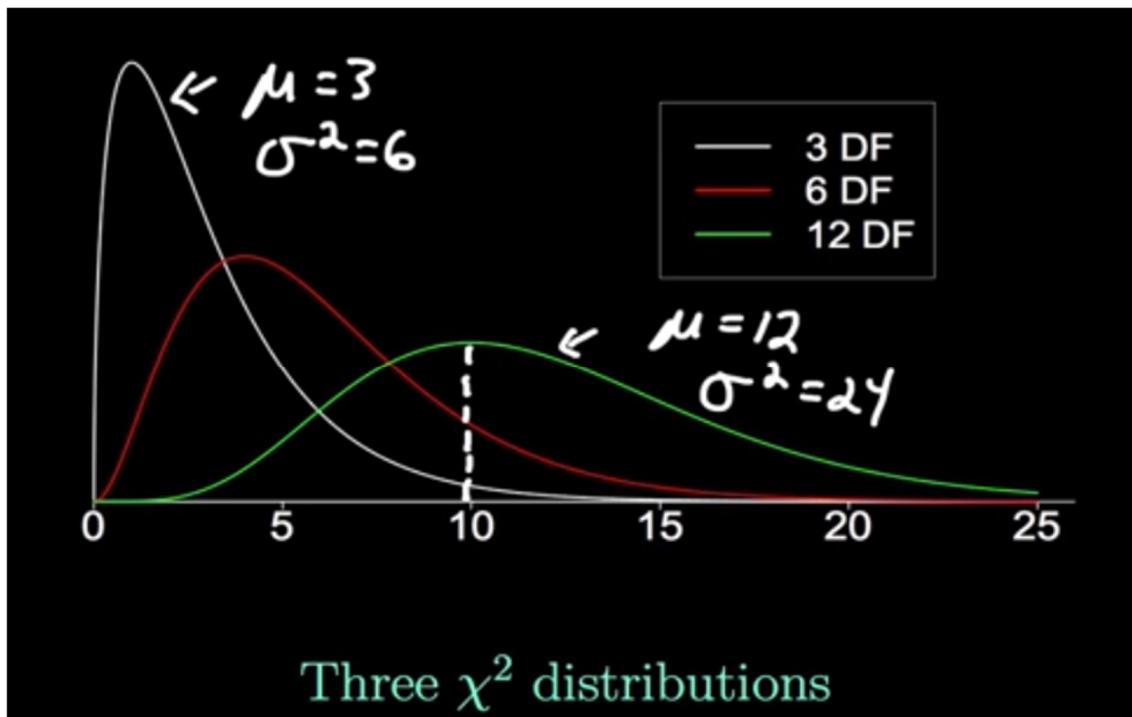
$$\sigma^2 = 2 \cdot k$$

Let us plot the probability function “p(x)”:

- When “k<3”: greatest is when x=0.



- When “k≥3” the probability function stretches, gets more symmetric and tends to look like a normal distribution.



For practical usages, we need to find areas under the curve with percentiles of this distribution. It involves integrating the function “ $p(x)$ ” which is a tedious task as the integration is numerical. We can do this integration using a table or software.

#### Usages

A chi-square test is a statistical test used to compare observed results with expected results. The purpose of this test is to determine if a difference between observed data and expected data is due to chance, or if it is due to a relationship between the variables you are studying.

|                            | <b>Chi-Square Goodness of Fit Test</b>                                       | <b>Chi-Square Test of Independence</b>                                                           |
|----------------------------|------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| <b>Number of variables</b> | One                                                                          | Two                                                                                              |
| <b>Purpose of test</b>     | Decide if one variable is likely to come from a given distribution or not    | Decide if two variables might be related or not                                                  |
| <b>Example</b>             | Decide if bags of candy have the same number of pieces of each flavor or not | Decide if movie goers' decision to buy snacks is related to the type of movie they plan to watch |

|                                              |                                                                                                        |                                                                                                                                                                                                                                                     |
|----------------------------------------------|--------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Hypotheses in example</b>                 | $H_0$ : proportion of flavors of candy are the same<br>$H_a$ : proportions of flavors are not the same | $H_0$ : proportion of people who buy snacks is independent of the movie type<br>$H_a$ : proportion of people who buy snacks is different for different types of movies                                                                              |
| <b>Theoretical distribution used in test</b> | Chi-Square                                                                                             | Chi-Square                                                                                                                                                                                                                                          |
| <b>Degrees of freedom</b>                    | Number of categories minus 1<br>In our example, number of flavors of candy minus 1                     | Number of categories for first variable minus 1, multiplied by number of categories for second variable minus 1<br>In our example, number of movie categories minus 1, multiplied by 1 (because snack purchase is a Yes/No variable and $2-1 = 1$ ) |

### T-student distribution

Note: In order to normalize a standard normal distribution to have "0" mean " $\mu$ " and "1" standard deviation " $\sigma$ ":  $N(\mu, \sigma^2) \rightarrow N(0,1)$  we do the following:

$$Z = \frac{X - \mu}{\sigma}$$

Suppose we draw "n" samples or observations of a variable "X" that follows a normal distribution. Each observation is different, of course. If we calculate the empirical average " $\bar{X}_n$ " of these "n" observations this value will be similar but not exactly the true " $\mu$ ". Actually, if we draw again "m" samples of observations the empirical average " $\bar{X}_m$ " again the value will be different.

We can then say, that the empirical average " $\bar{X}$ " follows a standard normal distribution like this:

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$$

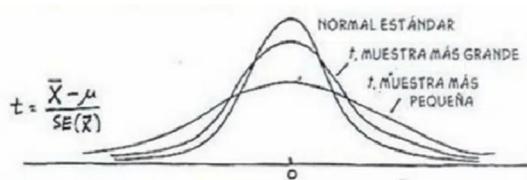
Where " $\bar{X}$ " is the empirical average, " $\mu$ " the true average of the entire population, " $\sigma$ " the true standard deviation and "n" is the number of observations made.

$$\frac{\sigma}{\sqrt{n}} = \text{Standard Error}(SE) \approx \frac{s}{\sqrt{n}}$$

When we try to build confidence intervals, in practice, many times we don't know the actual " $\sigma$ ", and we can just estimate it with the empirical deviation "s" (formula to obtain it based on the "n" observations" we did). The new expression using "s" is known as "t" and it follows a "t-distribution" instead of the standard normal distribution:

$$t = \frac{\bar{X} - \mu}{s/\sqrt{n}}$$

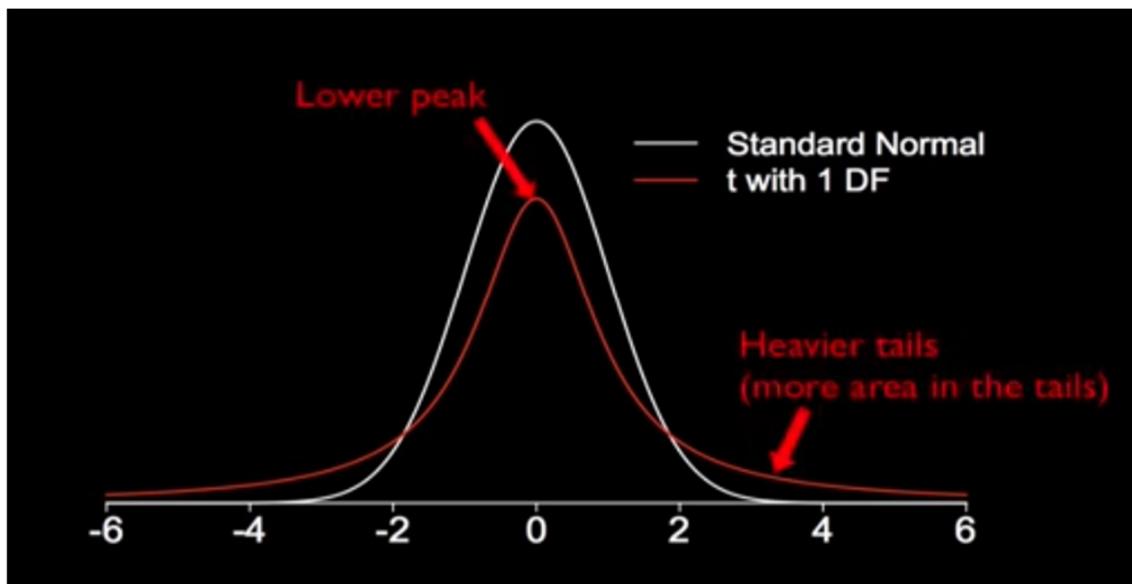
"s" will be different from sample to sample and will no longer be " $\sigma$ ". So, we call it "t" and no longer "Z".



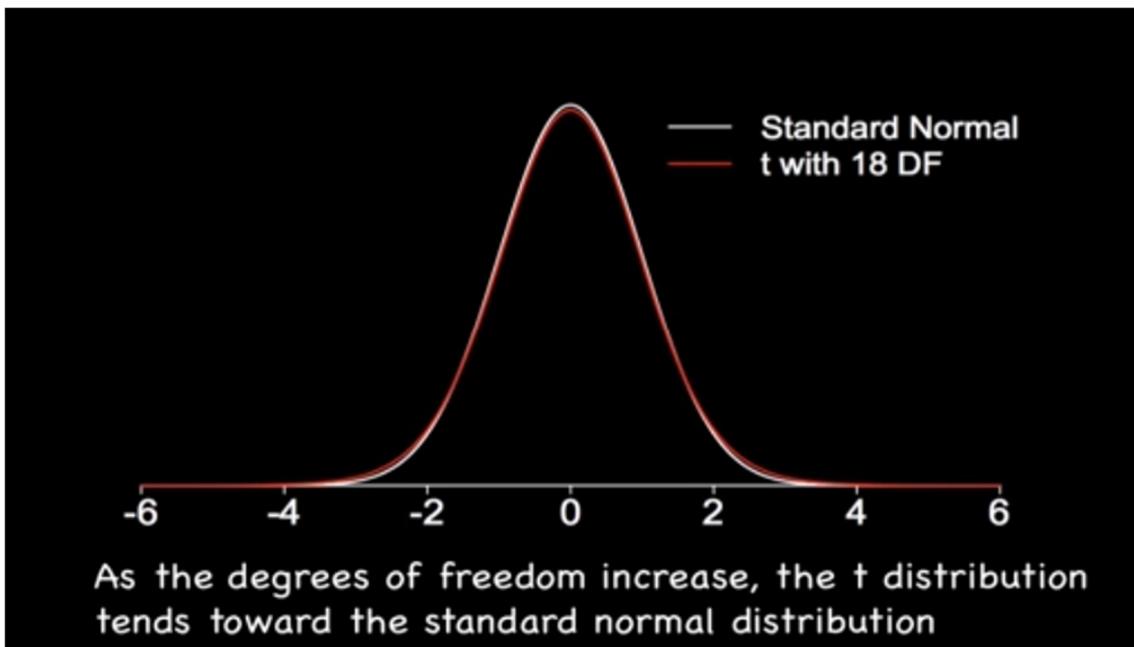
When observing a variable "X" that follows a normal distribution, then the value "t" follows a "t-distribution" with "n-1" degrees of freedom. As we cannot use the actual " $\sigma$ " of our population.

The degrees of freedom of "t" are related for the sample variance " $s^2/n$ ".

Because the "t" variable looks like "Z" except for the usage of "s" instead of " $\sigma$ ", we are replacing the usage of a constant value with another observed value: the t-distribution will look like the normal distribution but greater variance as there is more uncertainty regarding using "s" which is varies and adds randomness.



As the degrees of freedom increase the T-student tends to be the normal distribution as we remove the uncertainty and our variance "s" gets closer to " $\sigma$ ".



At 20 DF still has slightly higher tails and lower peak.

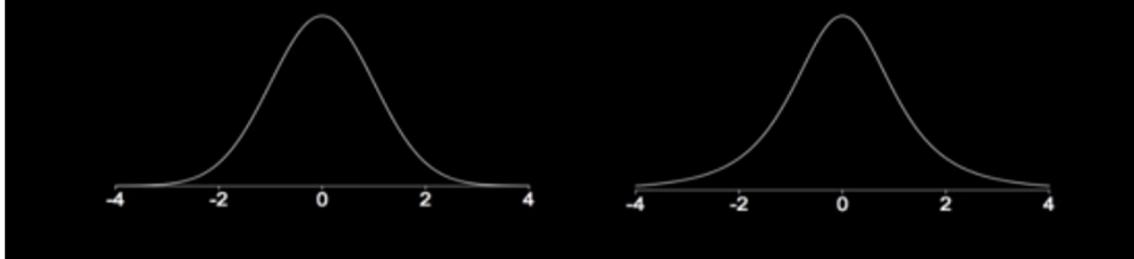
### Constructing a 95% confidence interval (for the population mean)

If  $\sigma$  is known:

$$\bar{X} \pm 1.96 \times \frac{\sigma}{\sqrt{n}}$$

If  $\sigma$  is not known:

$$\bar{X} \pm ? \times \frac{s}{\sqrt{n}}$$



Not knowing the standard deviation of our population makes it complicated to construct the 95% confidence interval.

Because the “t” distribution has a bigger area to the tails the confidence interval will return a bigger value compared with the equivalent normal distribution that is safer to use. How large is the “t” value compared with the normal distribution value depends on the number of samples: the more samples or observations the most similar the values will be.

Constructing a 95% confidence interval  
(for the population mean)

|                                                                                           |                                                                                       |
|-------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| <p>If <math>\sigma</math> is known:</p> $\bar{X} \pm 1.96 \times \frac{\sigma}{\sqrt{n}}$ | <p>If <math>\sigma</math> is not known:</p> $\bar{X} \pm ? \times \frac{s}{\sqrt{n}}$ |
|-------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|

The standard normal distribution      A t distribution

$Z_{0.025} = 1.96$

The t value that has an  $t_{0.025} > 1.96$

Well that depends on the degrees of freedom,

**Note:** 1,96 means that a value outside the 95% interval confidence is at least 1,96 times the standard deviation.

Degree of freedom =  $n-1$  observations.

$t_{.025}$  values (for 95% confidence intervals)

| $n$ | $df$     | $t_{.025}$   |
|-----|----------|--------------|
| 6   | 5        | 2.571        |
| 11  | 10       | 2.228        |
| 31  | 30       | 2.042        |
| 51  | 50       | 2.009        |
| 101 | 100      | 1.984        |
|     | $\infty$ | <u>1.960</u> |

$Z_{.025} \rightarrow$

## F-distribution

It is related to the  $\chi^2$  distribution. Suppose:

$U_1$  has a  $\chi^2$  distribution with  $v_1$  degrees of freedom.

$U_2$  has a  $\chi^2$  distribution with  $v_2$  degrees of freedom.

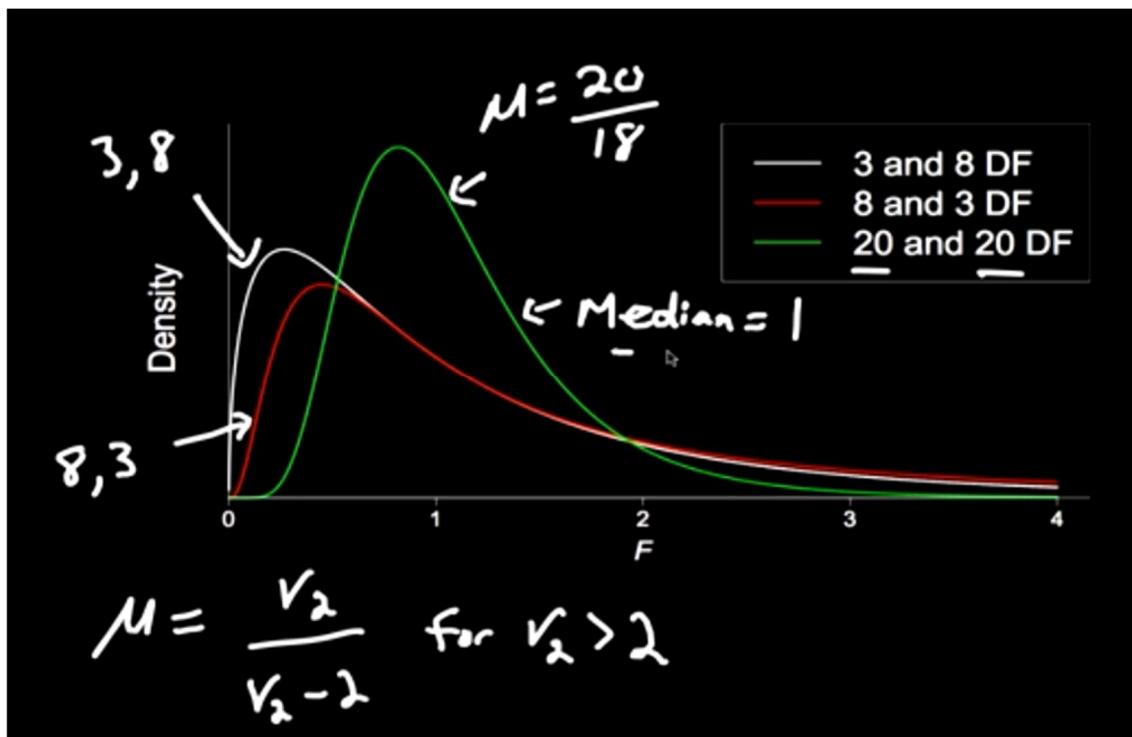
Then:

$\frac{U_1/v_1}{U_2/v_2}$  has an F distribution with  $v_1, v_2$  degrees of freedom

The F-distribution appears when dealing with ratios of variances in different inference scenarios.

In this case it can be proved:

$$\mu = \frac{v_2}{v_2 - 2} \text{ for } v_2 > 2$$



It is mostly used for statistical inference procedures to define areas under curves and percentiles. This F-distribution can be used together with tables.