Contents

Chapter 1

Decision Trees

Let's start with training set:

Definition 1.0.1: Training set

is defined training set a *set of examples*, where: $\langle x^{(i)}, y^{(i)} \rangle$ where:

- i is the instance of the example
- $x^{(i)} \in X$ is the set of *input*
- $y^{(i)} \in Y$ is the set of *output*

the problem of machine learning is to find a function $h : X \to Y$ that approximates the real function $f : X \to Y$. We have two types of problems:

- **Classification**: Y is a discrete set of values (e.g. $\{0, 1\}$)
- **Regression**: Y is a continuous set of values (e.g. $\mathbb{R}$)

1.1 Hypothesis space

In machine learning the *hypothesis space* H is defined as the set of all possible functions that can be used to approximate the real function $f : X \to Y$. Formally:

Definition 1.1.1: Hypothesis space

A hypothesis space H is defined as the set: $H = \{h | h : X \to Y\}$ where:

- h is a function (hypothesis) that maps input X to output Y
- X the input space (features, domain of data).
- Y the output space (labels, range of data).
- $|H|$ is the size of the hypothesis space (number of possible hypotheses)

this let us to define the model:

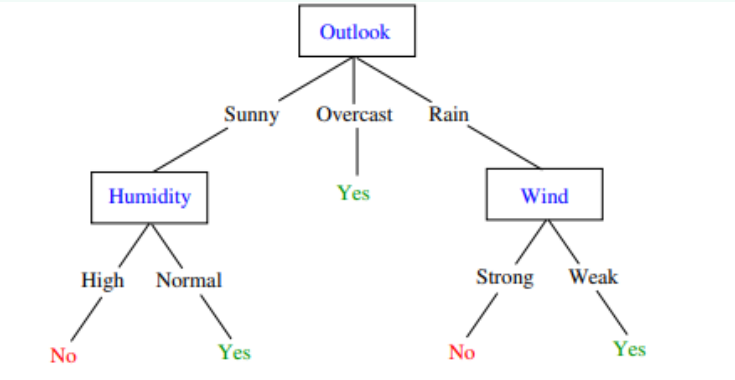
Definition 1.1.2: Model

A model is a way to compute a function $h \in H$ from the training set.

Example 1.1.1 (Decision tree)

A good day to play tennis? Our function F is:

$$\mathbb{F} : Outlook \times Humidity \times Wind \times Temp \rightarrow PlayTennis?$$

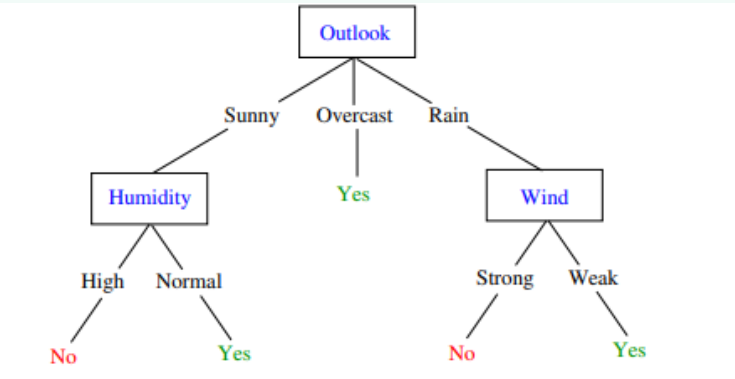


where:

- $Outlook \in \{Sunny, Overcast, Rain\}$
- $Humidity \in \{High, Normal\}$
- $Wind \in \{Weak, Strong\}$
- $PlayTennis? \in \{Yes, No\}$

Every node tests an attribute. Each branch corresponds to one of the possible values for that attribute. Each leaf node assigns a classification (Yes or No), in other words predicts the answer Y.
The prblem configurationg is the following:

- X is the set of all possible $x \in X$ that corresponds to a vector of attributes $(Outlook, Humidity, Wind, Temp)$
- Target function $f : X \rightarrow Y$ is the function that maps the attributes to the target variable $PlayTennis?$ (booleans)
- Hypothesis space $H = \{h | h : X \rightarrow Y\}$ is the set of all possible decision trees that can be constructed using the attributes in X to predict the target variable Y



1.1.1 Top-down inductive construction

Let $X = X_1 \times X_2 \cdots \times X_n$ where $X_i = \{\text{Truee}, \text{False}\}$
Can we represent, for instance, $Y = X_2 \wedge X_5$? or $Y = X_2 \wedge X_5 \vee (\neg X_3) \wedge X_4 \wedge X_1$?
and:

- do we have a decision tree for each h in the space hypothesis?
- if the tree exists, is it unique?
- if it is not unique, do we have a preference?

Theorem 1.1.1 Basta - Bonzo

Main loop:

- **Pick the "best" attribute** X_i: At the current node, choose which featue/attribute will bwst split the training data.

  Best means: the attribute that gives the most information gain
- **Create a child node for each possible value of** X_i: for instance if attribute is "weather" with values "sunny", "rainy", "overcast", create three child nodes.
- **check if all examples in the chils node are pure**: if all examples belong to the same class (e.g., all "yes" or all "no"), make that node a leaf node with that class label. If not repeat the process recursively for each child node.

1.2 Entropy

Definition 1.2.1: Entropy

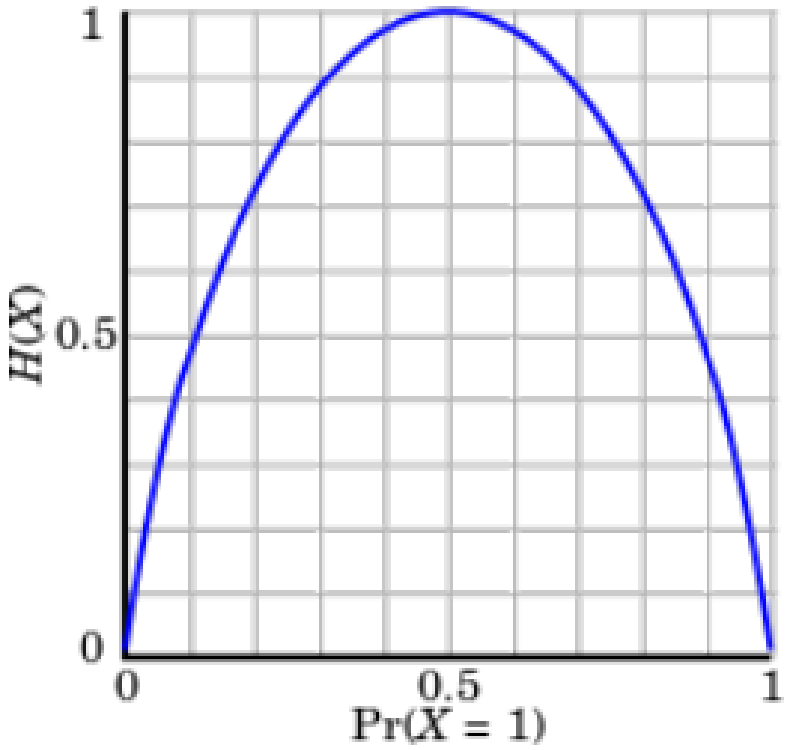
The entropy $H(S)$ of a set of examples S is defined as:

$$H(S) = -\sum_{i=1}^{n} P(X = i) \log_2 P(X = i)$$

where:

- $P(X = i)$ is the proportion of examples in S that belong to class i
- n is the number of classes (the number of possibile values of X)

In other words, Entropy meausers the *degree of uncertainty* of the information. It is maximal when X is uniformly distributed (all classes have the same probability) and minimal (zero) when all examples belong to the same class (pure set)



1.2.1 Information Theory (Shannon 1948)

The entropy is the average amount of information produced by a stochastic source of data. The *information* is associated to the *probability* of each datum (the surprise element):

- An event with probability 1 (certain event) provides no information (no surprise): $I(1) = 0$.
- An event with probability 0 (impossible event) provides infinite information (really surprising): $I(0) = \infty$.
- Given two independent events A and B, the information provided by both events is the sum of the information provided by each event:
$$I(A \cap B) = I(A) + I(B)$$

So is natoral defining

$$I(p) = -\log_2(p)$$

1.2.2 Code Theory (Shannon-Fano 1949, Huffman 1952)

The entropy is also related to the avarage number of bits required to transmit outcomes produces by a stochastic source process x.
Let suppose to have n events with same probability $p_i = \frac{1}{n}$. Home many bits do we need to encode these events? The answer is $\log_2(n)$ bits. For instance, if we have 4 events, we need 2 bits to encode them:.
In this case:

$$H(X) = -\sum_{i=1}^{n} P(X = i) \log_2 P(X = i) = -\sum_{i=1}^{n} \frac{1}{n} \log_2 \frac{1}{n} = \log_2(n)$$

1.3 Information Gain

In a decision tree, the goal is to maximize the information gain during the execution of the algorithm. In other words, the final split should result in the minimum possible impurity. Here are the main formulas:

Theorem 1.3.1 Entropy of X

$$H(X) = -\sum_{i=1}^{n} P(X = i) \log_2 P(X = i)$$

Theorem 1.3.2 Conditional Entropy of X given a specific $Y = v$

$$H(X \mid Y = v) = -\sum_{i=1}^{n} P(X = i \mid Y = v) \log_2 P(X = i \mid Y = v)$$

This measures the entropy of X restricted to the subgroup where $Y = v$.

Theorem 1.3.3 Conditional Entropy of X given Y

$$H(X \mid Y) = \sum_{v=1}^{m} P(Y = v)\, H(X \mid Y = v)$$

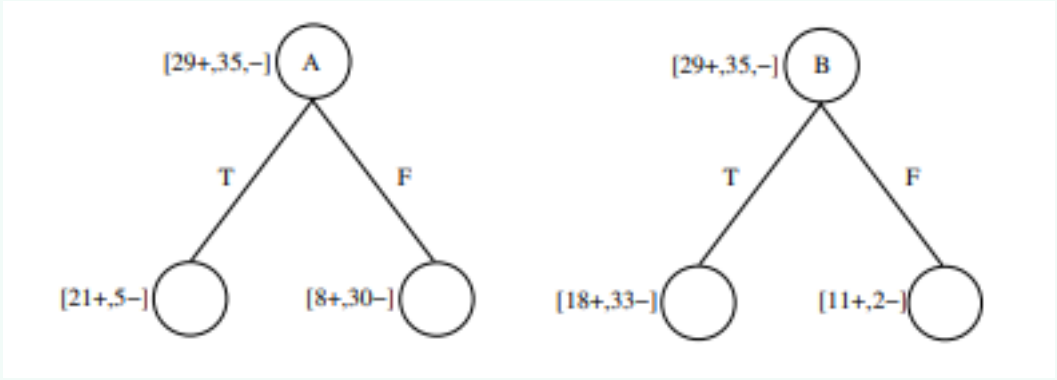
This is the generalization of 1.3, used to evaluate the utility of an attribute. It measures the average impurity that remains in X after splitting the data using all possible values of Y.

Theorem 1.3.4 Information Gain between X and Y

Here we are! $I(X, Y) = H(X) - H(X\|Y) = H(Y) - H(Y\|X)$

Example 1.3.1 (Information gain)

Let us measure the entropy reduction of the target variable Y due to some attribute X, that is the information gain $I(Y, X)$ between Y and X



$$H(Y) = -\frac{29}{64}\log_2\left(\tfrac{29}{64}\right) - \frac{35}{64}\log_2\left(\tfrac{35}{64}\right) = 0.994$$

$$H(Y \mid A = T) = -\frac{21}{26}\log_2\left(\tfrac{21}{26}\right) - \frac{5}{26}\log_2\left(\tfrac{5}{26}\right) = 0.706$$

$$H(Y \mid A = F) = -\frac{8}{38}\log_2\left(\tfrac{8}{38}\right) - \frac{30}{38}\log_2\left(\tfrac{30}{38}\right) = 0.742$$

$$H(Y \mid A) = 0.706 \cdot \tfrac{26}{64} \; + \; 0.742 \cdot \tfrac{38}{64} = 0.726$$

$$I(Y, A) = H(Y) - H(Y \mid A) = 0.994 - 0.726 = 0.288$$

$$H(Y \mid B) = 0.872$$

$$I(Y, B) = 0.122$$

Chapter 2

Overfitting

Let us consider the error of the hypothesis h

- on the training set, $error_{train}(h)$
- on the full data set $\mathcal{D}$, $error_{\mathcal{D}}(h)$

Definition 2.0.1: Overfitting

It's said taht h *overfits* the training set if there exixsts another hypotheses h' such that:

$$\text{error}_{\text{train}(h)} < \text{error}_{\text{train}(h')}$$

but

$$\text{error}_{\mathcal{D}(h)} > \text{error}_{\mathcal{D}(h')}$$

These models (h and h') represent two different situations. The first corresponds to a model that fits the training dataset very closely, including its uncertainty and noise. The second is simpler: it captures only the general trend of the training data and avoids fitting the noise. As a consequence, the error with respect to the true data distribution $\mathcal{D}$ is larger for the first model than for the second. The second one is better! Let's generalised.
But *We do not know* $\mathcal{D}$

2.1 Avoiding the over fitting

2.1.1 Detecting the Overfitting: validation set

For Detecting the Overfitting it's usefull divinding the dats aviables in two disjoint sets:

- **Training set**: set of datas that the model *use for learning*. The dtree is built by this datas
- **validation set**: This set is not shown during the training- It's used as "test" for evaluating the accuracy of the model

2.1.2 Early stopping

This is a proactive strategy. Instead of let the tree grows untill his major complexity, it's sopped first the possibility of Overfitting. The growing of a branch is stopped if these two conditions is verified:

- **The improvement is too small**: if a possible divsion of datas produces a gain of information below a certain threshold, it means that it's not usefull to continue
- **There are not enaugh datas**: if a node contains a number of examples too musch low, any decision taken would be statistically unreliable and probably based on noise. The tree stops to avoid creating rules based on coincidences.

2.1.3 Post - Pruning

This strategy is **reactive**. The decision tree is let grow completely on the training set, which may lead to overfitting, and then the useless or harmful branches are pruned.

Definition 2.1.1: Reduce-Error Post-Pruning

The *reduce-error post-pruning* technique works as follows:

- build the tree completely
- evaluate each branch using a validation set
- prune the branch whose removal improves accuracy the most
- repeat until no further pruning improves the accuracy

Chapter 3

Probabilistic approach

3.1 core idea

we have two main points of views:

- **traditional view**: we wanna to approximate a function $f : X \to X$
- **Probabilist view**: we wanna compute probablilities: $p : P(Y \mid X)$

3.1.1 Probs basics

Random variables

A random variables X represents an oyt come about which we're ncertain

Example 3.1.1 (Random variables)

- X =`true` if a randomly drawn stdent is male
- X = first name of the student
- X =`true` if a randomly drawn stdent have the same birthday

Formal def:

Definition 3.1.1: Probs variables

the set Ω of the possible outcomes is called the sample space. It is said random variable a measurable function over Ω:

- Discrete: $\Omega \to \{m, f\}$
- Continuos: $\Omega \to \mathbb{R}$

Definition 3.1.2: Probs def

it is defined $P(X)$ is the fraction of times X is true in repeated runs of the same experiment.

Note:

The definition requires that all samples

Pay attention:

Wrong Concept 3.1: bad examples

Sample space, let Ω be a space made the possibile sum:

$$\Omega = \{2, 3, 4, \ldots, 12\}$$

Problem: not all sums are equally likely! It should be:

$$P(sum = 2) = 1/11$$
$$P(sum = 7) = 1/11$$

but in reality:

- Sum = 2: can only happen one way: $(1, 1)$
- Sum = 7: can happen six ways:$(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)$

so

$$P(sum = 2) \neq P(sum = 7)$$

A correct approach is

Claim 3.1.1 correct approach

Be $\Omega = (1, 1), (1, 2), (1, 3), ..., (6, 5), (6, 6)$, where $|\Omega| = 36$outcomes
each pair has equally probability $= \frac{1}{36}$
Now here is a correctly computing:

$$P(sum = 2) = \frac{|(1,1)|}{36} = \frac{1}{36}$$
$$P(sum = 7) = \frac{|(1,6),(2,5),(3,4),(4,3),(5,2),(6,1)|}{36} = \frac{6}{36}$$

The Axioms of Probability Theory

These are the fundamental rules that make probability a "reasonable theory of uncertainty":

Axioms of probability theory

(1) Non-negativity: $0 \leqslant P(A) \leqslant 1$ for all events A. (3.1)

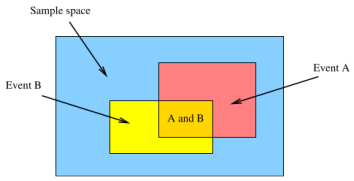
(2) Normalization: $P(\Omega) = 1$. (3.2)

(3) Countable additivity: If $A_1, A_2, \ldots$ are disjoint, then $P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i)$. (3.3)

Then:

Corollary 3.1.1 consequences of the axioms

- Monotonicity: If $A \subseteq B$, then $P(A) \leq P(B)$
- Union rule (for two events): $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- $P(True) = 1$
- $P(False) = 0$



Derivied theorems

Corollary 3.1.2 Complement Rule

$$P(\neg A) = 1 - P(A)$$

Dm:

$$P(A \cup \neg A) = P(A) + P(\neg A) - P(A \cap \neg A)$$

But:

$$P(A \cup \neg A) = P(\text{True}) = 1 \quad \text{and} \quad P(A \cap \neg A) = P(\text{False}) = 0$$

Therefore:

$$1 = P(A) + P(\neg A) - 0 \quad \Longrightarrow \quad P(\neg A) = 1 - P(A)$$

Corollary 3.1.3 Partition Rule

$$P(A) = P(A \cap B) + P(A \cap \neg B)$$

Proof:

$$\begin{aligned} A &= A \cap (B \cup \neg B) && \text{[since } B \cup \neg B \text{ is always True]} \\ &= (A \cap B) \cup (A \cap \neg B) && \text{[distributive law]} \end{aligned}$$

Hence,

$$\begin{aligned} P(A) &= P((A \cap B) \cup (A \cap \neg B)) \\ &= P(A \cap B) + P(A \cap \neg B) - P((A \cap B) \cap (A \cap \neg B)) \\ &= P(A \cap B) + P(A \cap \neg B) - P(\text{False}) \\ &= P(A \cap B) + P(A \cap \neg B) \end{aligned}$$

Multivalued Discrete Random Variables

Definition 3.1.3: k-value Discrete Random Variables

A random variable A is *k-valued discrete* if it takes exactly one value from

$$\{v_1, v_2, \ldots, v_k\}.$$

Proposition 3.1.1 Key proprieties

1. **Mutual exclusivity:** For $i \neq j$,
$$P(A = v_i \cap A = v_j) = 0$$
2. **Exhaustiveness:**
$$P(A = v_1 \cup A = v_2 \cup \cdots \cup A = v_k) = 1$$

Conditional Probability

Definition 3.1.4: Conditional probs

The Conditional probs of the event A *given* the event B is defined as the quantity

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Corollary 3.1.4 Cahin roule

$$P(A \cap B) = P(B)P(A \mid B) = P(A)P(B \mid A)$$

Independent Events

Definition 3.1.5: Independent Events

Events A and B are independent when:

$$P(A \mid B) = P(B)$$

(Meaning: B provides no information about A.)

Corollary 3.1.5 consequences

- $P(A \cap b) = P(A)P(B)$ (from chail roule)
- $P(B|A) = P(B)$ (symmetry)

Bayes' Rule: The Heart of Probabilistic ML (ok chat... really?)

Theorem 3.1.1 Bayes's roule

Now we have Bayes roule

$$P(A \mid B) = \frac{P(A)P(B|A)}{P(B)}$$

Proof: It's true by the chain roule that: $P(A \cap B) = P(B)P(A \mid B)$. It's true also the reverse case $P(A \cap B) = P(B)P(A \mid B)$.
Since both expressions equal $P(A \mid B)$, they must equal each other:

$$P(A)P(B \mid A) = P(B)P(A \mid B)$$

that it's equal to

$$P(A \mid B) = \frac{[P(A)P(B \mid A)]}{P(B)}$$

Example 3.1.2 (The trousers problem)

Setup:

- 60% of students are boys, 40% are girls
- girls wear in the same number skirt and trousers
- boys only wear trousers

If we see a student wearing trousers, what is the probability that is a girl?

Solution: The probs a priori that a strudent is a girl is

$$P(G) = \frac{2}{5}$$

the probability that a student wears trousers is

$$P(T) = \frac{1}{5} + \frac{3}{5} = \frac{4}{5}$$

the probability that a student wear trousers, given that the student is a girl, is

$$P(T \mid G) = 1/2$$

So

$$P(G \mid T) = \frac{p(G)p(T \mid G)}{P(T)} = \frac{2/5 \cdot 1/2}{4/5} = 1/4$$

Machine Learning Form

Machine Learning Form For discrete Y with values $\{y_1, y_2, \ldots, y_m\}$ and X with values $\{x_1, x_2, \ldots, x_n\}$:

$$P(Y = y_i \mid X = x_j) = \frac{P(Y = y_i) \cdot P(X = x_j \mid Y = y_i)}{P(X = x_j)}$$

Expanding the denominator:

$$\begin{aligned} P(X = x_j) &= \sum_i P(X = x_j, Y = y_i) \quad \text{[sum over all } Y \text{ values]} \\ &= \sum_i P(Y = y_i) \cdot P(X = x_j \mid Y = y_i) \quad \text{[chain rule]} \end{aligned}$$

Complete Bayes' Rule:

$$P(Y = y_i \mid X = x_j) = \frac{P(Y = y_i) \cdot P(X = x_j \mid Y = y_i)}{\sum_i P(Y = y_i) \cdot P(X = x_j \mid Y = y_i)}$$

Terminology:

$$\underbrace{P(Y \mid X)}_{\text{posterior}} = \frac{\overbrace{P(X \mid Y)}^{\text{likelihood}} \cdot \overbrace{P(Y)}^{\text{prior}}}{\underbrace{P(X)}_{\text{marginal}}}$$

- **Posterior** $P(Y \mid X)$: What we want – probability of Y given observed X
- **Likelihood** $P(X \mid Y)$: How likely is X if Y is true?
- **Prior** $P(Y)$: What we believed before seeing X
- **Marginal** $P(X)$: Overall probability of observing X (normalization constant)

Alternative form:

$$\text{Posterior} = \frac{\text{Likelihood} \cdot \text{Prior}}{\text{Marginal Likelihood}}$$

where:

$$\text{Marginal} = \sum_Y P(X \mid Y) \cdot P(Y)$$

The term "marginal" means we've **marginalized** (integrated/summed) over Y.

3.2 The Joint Distribution

Definition 3.2.1: Joint Distribution

Let $X_1, X_2, \ldots, X_n$ be discrete random variables. The *joint probability distribution* (or *joint distribution*) of these variables is the function:

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$$

which assigns to every possible combination of values $(x_1, x_2, \ldots, x_n)$ the probability that the random variables simultaneously take those values.
Formally, for discrete variables, the joint distribution satisfies:

- $0 \leqslant P(x_1, x_2, \ldots, x_n) \leqslant 1$ for all $(x_1, \ldots, x_n)$
- $\sum_{x_1} \sum_{x_2} \cdots \sum_{x_n} P(x_1, x_2, \ldots, x_n) = 1$

Let's see an example

Example 3.2.1 (Joint distribution)

- build a table with all possible combinations of values of random variables (features)
- compute the probability for any different combination of values

This table is the "Joint distribution"!
Having that we may compute the probability of any event expressible as a logical combination of the features, with this formula

$$P(E) = \sum_{row \in E} (row)$$

in words for calculating an event we must add each row that is contained by the event. Let's provide an example (of an example)
Let us compute the probability $P(M, poor)$

gender	w. hours	wealth	prob.
F	≤ 40	poor	0.25
F	≤ 40	rich	0.03
F	> 40	poor	0.04
F	> 40	rich	0.01
M	≤ 40	poor	0.33
M	≤ 40	rich	0.10
M	> 40	poor	0.13
M	> 40	rich	0.11

we have: $P(M, poor) = 0.33 + 0.13 = 0.46$

3.2.1 Inference with the Joint distribution

Here are with the inference:

Definition 3.2.2: Contintional probability

Let E_1 and E_2 be two events defined as logical conditions over subsets of the random variables (e.g., $E_1 : X_i = a, X_j = b$; $E_2 : Y = y$)
Then, *conditional probability* of E_1 given E_2 is:

$$P(E_1 \mid E_2) = \frac{P(E_1 \wedge E_2)}{P(E_2)} = \frac{\sum_{row \in (E_1 \wedge E_2)} P(row)}{\sum_{row \in (E_2)} P(row)}$$

for instance:

Example 3.2.2 (Conditional probability)

Let's compute $P(M|poor) = \frac{P(M \wedge poor)}{P(poor)}$. We know that $P(M, poor) = 0.46$. Let us compute $P(poor)$:

gender	w. hours.	wealth	prob.
F	≤ 40	poor	0.25
F	≤ 40	rich	0.03
F	> 40	poor	0.04
F	> 40	rich	0.01
M	≤ 40	poor	0.33
M	≤ 40	rich	0.10
M	> 40	poor	0.13
M	> 40	rich	0.11

Easy! $P(poor) = .75 \wedge P(M|poor) = 0.46/0.75 = 0.61$

3.2.2 Complexity issues

Let us build the joint table relative to

$$P(Y = wealth|X_1 = gender, X_2 = orelav.)$$

X_1=gender	X_2=ore lav.	$P(rich\|X_1, X_2)$	$P(poor\|X_1, X_2)$
F	≤ 40	.09	.91
F	> 40	.21	.79
M	≤ 40	.23	.77
M	> 40	.38	.62

To fill the table we need to compute $4 = 2^2$ parameters
If we have n random variable $X = X_1 \times X_2, \ldots, X_n$ where each X_i is boolean, we need to compute 2^n parameters.
These parameters are *probabilities*: to get reasonable value we would need a huge amount of data.
In particular the The Joint Distribution Requires *Exponential parameters*

Example 3.2.3 (features and params)

- With just 10 binary features, you need $2^{11} - 1 = 2047$ parameters
- With 20 features: over 1 million parameters
- With 100 features: 2^{101} a number larger than the estimated atoms in the observable universe.

This is computationally and statistically infeasible.

USing Bayes

for reducing complexity, we can rewrite the formula with the Bayes' rule:

$$P(Y = y_i \mid X = x_j) = \frac{P(Y = y_i) \cdot P(X = x_j \mid Y = y_i)}{\sum_i P(Y = y_i) \cdot P(X = x_j \mid Y = y_i)}$$

generalising:

$$P(Y \mid X_1, X_2, \ldots, X_n) = \frac{P(Y) \cdot P(X_1, X_2, \ldots, X_n \mid Y)}{P(X_1, X_2, \ldots, X_n)}$$

But... there is a problem, it's required to know

$$P(X_1, X_2, \ldots, X_n \mid Y)$$

that is the joint distribution of the features given Y, that requires, another time, 2^n params

3.2.3 Naive Bayes

For atteniung the complexity, it's possible assume an indipendencies conditional hypotesis, called "Naïve Bayes":

$$P(X_1, X_2, \ldots, X_n|Y) = \prod_i P(X_i|Y)$$

So given Y, X_i and X_j are independent from each other. In other therms:

$$P(X_i|X_j, Y) = P(X_i|Y)$$

Note:

This means: onece we know Y, the feature $X_i \forall i$ are independents between each others

Example 3.2.4 (example 1)

A box contains two coins: a regular coin and a fake two-headed coin ($P(H) = 1$).Choose a coin at random, toss it twice and consider the following events:

- A = First coin toss is H
- B = Second coin toss is H
- C = First coin is regular

Example 3.2.5 (example 2)

For individuals, height and vocabulary are not independent, but they are if age is given.

Giga formula with naive bayes

Theorem 3.2.1 Bayes rule

$$P(Y = y_i \mid X_1, \ldots, X_n) = \frac{P(Y = y_i) \cdot P(X_1, \ldots, X_n \mid Y = y_i)}{P(X_1, \ldots, X_n)}$$

Proof: Left to mesco as exercice

Theorem 3.2.2 Naïve Bayes

$$P(Y = y_i \mid X_1, \ldots, X_n) = \frac{P(Y = y_i) \cdot \prod_j P(X_j \mid Y = y_i)}{P(X_1, \ldots, X_n)}$$

Proof: Left to Bonzo as exercice

Theorem 3.2.3 Classification of a new sample $x^{\text{new}} = \langle x_1, \ldots, x_n \rangle$

Given a new instance represented by the feature vector $x^{\text{new}} = (x_1, x_2, \ldots, x_n)$, the predicted class is obtained as:

$$Y^{\text{new}} = \arg\max_{y_i} P(Y = y_i) \cdot \prod_j P(X_j = x_j \mid Y = y_i)$$

Proof: Seen as, using Bayes' formula, $\forall i$ the denominator used to calculate $P(Y = y_i|X_1, ..., X_n)$ remains the same,if we're only interested in maximizing the probability it's possible to only consider the numerator.
Given $P(X_1, ..., X_n) = C$,

$$P(Y = y_i) \cdot \prod_j P(X_j|Y = y_i) = C \cdot \frac{P(Y = y_i) \cdot \prod_j P(X_j|Y = y_i)}{C}$$
$$= C \cdot P(Y = y_i|X_1, ..., X_n)$$

Because C is a positive constant for each y_i, $argmax_{y_i} P(Y = y_i|X_1, ..., X_n) = argmax_{y_i} C \cdot P(Y = y_i|X_1, ..., X_n)$.

Note:

Theorem 3.2.3 expresses the decision rule of the Naïve Bayes classifier. Given a new vector of features $x^{\text{new}} = (x_1, x_2, \ldots, x_n)$, we estimate the most probable class y_i by maximizing the posterior probability $P(Y = y_i \mid X_1 = x_1, \ldots, X_n = x_n)$, which—under the conditional independence assumption—reduces to the product of the prior $P(Y = y_i)$ and the individual likelihoods $P(X_j = x_j \mid Y = y_i)$.

3.3 Learning algorithm

Given discrete Random Variables X_i, Y, there are two phases

- **Training**: in this phases the maching learn from the data of training set, estimating two types of probs:
  - **Prior** (prob of the classes). For any possible value y_k of Y , estimate

    $$\pi_k = P(Y = y_k)$$

    example: if 9 out of 14 matches are "Play = Yes", then $\pi_{yes} = \frac{9}{14}$, $\pi_{no} = \frac{5}{14}$
  - **Likelihoods**:(conditional probabilities of features). for any possible value x_{ij} of X_i estimate:

    $$\theta_{ijk} = P(X_i = x_{ij} \mid Y = y_k)$$

    It's the probability that a certain feature X_i assumes the value X_{ij}, given y_k.
    example: $P(Outlook = Sunny \mid Play = Yes) = \frac{2}{9}$
- **Classification of** $a^{new} = \langle a_1, \ldots a_n \rangle$ (a vector with n observed values (one for each feature)). We want to establish which class it belong to

  decision-making formula:

  $$Y^{\text{new}} = \arg\max_{y_k} P(Y = y_k) \cdot \prod_i P(X_i = a_i \mid Y = y_k)$$
  $$= \arg\max_k \pi_k \prod_j \theta_{ijk}$$

  where:

- $P(Y = y_k)$: prior
- $P(X_i = a_i \mid Y = y_k)$: likelihood for each features
- the prod $\prod_i$ is given by Naive assumption

Example 3.3.1 (a good day to play tennis?)

we wanna build a model that, given certain weather conditions, predict whether it is a good day to play tennis or not
Our class variable is:

$$Y = Play \in \{Yes, No\}$$

and the features observed are:

$$X_1 = Outlook \quad X_2 = Temp \quad X_3 = Humidity \quad X_4 = Wind$$

Here we have the dataset:

Table 3.1: Dataset for the *Play Tennis* classification problem

Outlook	Temp	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

TODO: TABELLA FATTA FARE DA UN LLM NON È VENUTA BENISSIMO

Calculating the prior

From the dataset we can compute the prior probabilities of the class variable Y:

$$P(Y = \text{Yes}) = \frac{9}{14}, \qquad P(Y = \text{No}) = \frac{5}{14}.$$

These represent the empirical frequencies of the two possible outcomes of Y.

Calculating the likelihoods

For each feature X_i and each class $Y = y_k$, we estimate the conditional probabilities

$$P(X_i = x_{ij} \mid Y = y_k),$$

that is, the probability of observing a certain feature value x_{ij} given that the class is y_k.
For example:

$$P(\text{Outlook} = \text{Sunny} \mid Y = \text{Yes}) = \frac{2}{9}, \qquad P(\text{Outlook} = \text{Sunny} \mid Y = \text{No}) = \frac{3}{5}.$$

These values are computed as the relative frequencies in the dataset.

Classification of a new instance

Suppose we want to classify the new day

$$x^{\text{new}} = (\text{Outlook} = \text{Sunny},\ \text{Temp} = \text{Cool},\ \text{Humidity} = \text{High},\ \text{Wind} = \text{Strong}).$$

We apply the Naïve Bayes decision rule:

$$Y^{\text{new}} = \arg\max_{y_i} P(Y = y_i) \cdot \prod_j P(X_j = x_j \mid Y = y_i).$$

For Y = Yes:

$$P(\text{Yes}) \cdot P(\text{Sunny}|\text{Yes}) \cdot P(\text{Cool}|\text{Yes}) \cdot P(\text{High}|\text{Yes}) \cdot P(\text{Strong}|\text{Yes}) = \frac{9}{14} \cdot \frac{2}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} \approx 0.0053$$

For Y = No:

$$P(\text{No}) \cdot P(\text{Sunny}|\text{No}) \cdot P(\text{Cool}|\text{No}) \cdot P(\text{High}|\text{No}) \cdot P(\text{Strong}|\text{No}) = \frac{5}{14} \cdot \frac{3}{5} \cdot \frac{1}{5} \cdot \frac{4}{5} \cdot \frac{3}{5} \approx 0.0205$$

Decision:
Since

$$P(Y = \text{No} \mid x^{\text{new}}) > P(Y = \text{Yes} \mid x^{\text{new}}),$$

the predicted class is

$$Y^{\text{new}} = \text{No}.$$

Therefore, according to the Naïve Bayes model, it is **not a good day to play tennis**.

3.4 Generative techniques

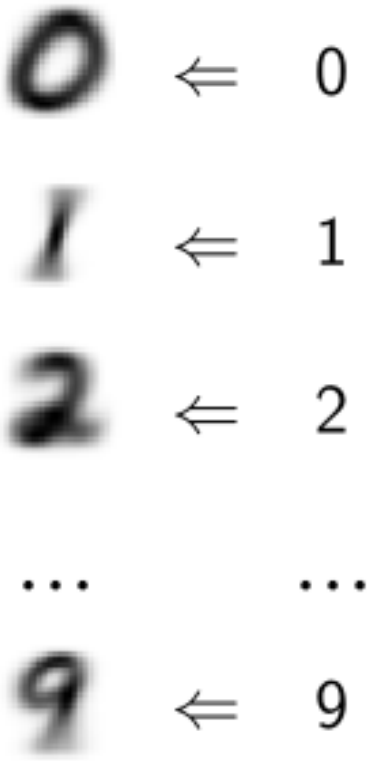
When we want to classify data (determine which category Y something belongs to given its features X), there is a fundamentally philosophical approaches, the **Generative Approach** that "learn how each class generates data", here is a sketch:

- Ask: "How does each class produce its characteristic data?"
- Model: $P(X|Y)$ (probability of features given category)
- Then use Bayes' Rule to reverse it and get $P(Y|X)$

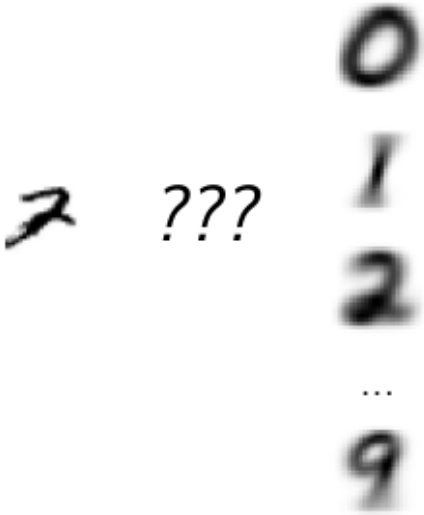
The term "Generative" come from the fact we're modeling the data generation process. We're essentially saying: "if I knew the category Y, I could generate/simulate typical data X from that category"

3.4.1 Big example: the visual intuition

We want to calssify images into categories $\{0, 1, \ldots, 9\}$ The generative approach says: "for each digit, learn waht imgs form that category typically look like. Then given a new img, see which category would most naturally produce such an img"



Okay, now we want to classify a new img:



Which of these distributions would most likely have generated this image?
Mathematically, for each category k:

$$\begin{aligned}\text{Score for category k} &= P(Y = k) \cdot P(X = \text{image} | Y = k) \\ &= \text{Prior} \cdot \text{Likelihood}\end{aligned}$$

You pick the category with the highest score. You're asking which generative mode (which category's distribution) best explains the observed data

Note:

The "score" is just the numerator of byes rule:

$$P(Y = k|X = \text{image}) = [P(Y = k) \cdot P(X = \text{image}|Y = k)]/P(X = \text{image})$$

Joint Distribution vs. Naïve Bayes

Ideally we'd want to model the Complete joint distribution:

$$P(X_1, X_2, \ldots, X_n|Y = y_i)$$

For MNIST, this would be the distribution of all 784 pixels (28×28 image) for images of each digit. This distribution would capture all the correlations between pixels - how pixel 1 relates to pixel 2, how groups of pixels form edges and curves, etc. Howver, modeling the full joint distribution for 784 dimensions is impossibly complex. We'd need:

- 2^{784} parameters just for binary pixels (more than atoms in the universe!)
- An astronomical amount of training data

So we play the card "Naïve Bayes" with 2500 atk and 1000 def:

$$P(X_1, \ldots, X_n \mid y = y_i) \approx \prod_j P(X_j \mid Y = y_i)$$

The assumption (effect) is: Given the category Y, all pixels are *independent*. This means we model each pixel separately:

$$P(X_1|Y = 0), P(X_2|Y = 0), ..., P(X_{784}|Y = 0)$$
$$P(X_1|Y = 1), P(X_2|Y = 1), ..., P(X_{784}|Y = 1)$$

This is computationally feasible, but we've lost all information about how pixels relate to each other!

3.4.2 Caution 1: The Zero Probability Problem

From a previous example if we have $P(Play = No|Outlook = Overcast)$ the result is $\theta_Overcast, No = 0$, This happened because in our training data, we never observed a "No" (don't play tennis) when the outlook was overcast

Note:

Remember the classification formula:

$$\text{Score} = \pi_k \cdot \prod_i \theta_{ijk}$$

If any single $\theta_{ijk} = 0$, the entire product becomes zero. So A single feature value you've never seen in training can completely eliminate a category from consideration, even if all other features strongly support it!

3.4.3 Caution 2: The Independence Assumption

Naïve Bayes assumes events are independent from each other (given Y). What if this is not the case?

The XOR Problem: A Fatal Limitation

Consider random binary images where pixels are either 0 or 1. We classify based on two pixels: p_1 and p_2.
Classification rule:

- **Category A:** if $p_1 = p_2$ (both same) — Images: $\{(0, 0), (1, 1)\}$
- **Category B:** if $p_1 \neq p_2$ (different) — Images: $\{(0, 1), (1, 0)\}$

This is an **XOR (exclusive OR)** relationship — a simple logical rule.
What Naïve Bayes learns:
For Category A (training: $(0,0)$ and $(1,1)$):

$$P(p_1 = 1 \mid A) = \frac{1}{2} \quad \text{[one out of two has } p_1 = 1\text{]}$$
$$P(p_2 = 1 \mid A) = \frac{1}{2} \quad \text{[one out of two has } p_2 = 1\text{]}$$

For Category B (training: $(0,1)$ and $(1,0)$):

$$P(p_1 = 1 \mid B) = \frac{1}{2} \quad \text{[one out of two has } p_1 = 1\text{]}$$
$$P(p_2 = 1 \mid B) = \frac{1}{2} \quad \text{[one out of two has } p_2 = 1\text{]}$$

Result: All probabilities are identical! For any test image (a, b):

$$\text{Score}_A = P(A) \cdot P(p_1 = a \mid A) \cdot P(p_2 = b \mid A) = 0.5 \cdot 0.5 \cdot 0.5 = 0.125$$
$$\text{Score}_B = P(B) \cdot P(p_1 = a \mid B) \cdot P(p_2 = b \mid B) = 0.5 \cdot 0.5 \cdot 0.5 = 0.125$$

Naïve Bayes cannot distinguish the categories! It achieves only 50% accuracy (random guessing) despite the trivially simple classification rule.
Why? The features p_1 and p_2 are **not independent** given the category — they're perfectly correlated:

- In Category A: if $p_1 = 0$ then $p_2 = 0$ (with certainty)
- In Category B: if $p_1 = 0$ then $p_2 = 1$ (with certainty)
- $P(p_2 = 1 \mid p_1 = 1, A) = 1 \neq P(p_2 = 1 \mid A) = 0.5$ — violates independence!

General lesson: Naïve Bayes cannot learn relationships between features. It only learns how common each individual feature value is within each class, not how features combine, interact, or correlate.

3.5 About Maximum Likelihood Estimation (MLE)

3.5.1 Problem definition

In words, the Maximum Likelihood Estimation (MLE) is the parameter value that maximizes the probability of observing the given data. For instance, if we have a model where all possible outcomes are 0 or 1, the parameter to estimate is computed using the *Bernoulli distribution*[1] $P(w) = \theta^w(1-\theta)^{(1-w)}$ where $w \in \{0,1\}$ represents all possible outcomes. However, normally we have a dataset (in this case a sequence of observations $D = \{w_1, w_2, \ldots, w_n\}$) and we don't know θ. The goal is to find the value of θ that makes our observed sequence most probable.
For independent observations, the likelihood is:

$$L(\theta|D) = P(D|\theta) = \prod_{i=1}^{n} P(w_i|\theta) = \prod_{i=1}^{n} \theta^{w_i}(1-\theta)^{1-w_i}$$

Let $\alpha_0 = \sum_{i=1}^{n} w_i$ be the number of times we observed 1 (successes), and $\alpha_1 = n - \alpha_0$ be the number of times we observed 0 (failures). Then:

$$L(\theta|D) = \theta^{\alpha_0}(1-\theta)^{\alpha_1}$$

The MLE is obtained by maximizing this likelihood (or equivalently its logarithm) with respect to θ.

[1] If you don't know this, please read the *Basta - Giolapalma notes for probability*

Definition 3.5.1: MLE

Given:

- A parametric probability model with parameter(s) θ
- Observed data $D = \{x_1, \ldots, x_n\}$
- A likelihood function $L(\theta|D) = P(D|\theta)$ (probability of data given parameters)

The *Maximum Likelihood Estimator* is defined as:

$$\hat{\theta}_{MLE} = \arg\max_\theta L(\theta \mid D) = \arg\max_\theta P(D|\theta)$$

In words: $\hat{\theta}_{MLE}$ is the parameter value that maximizes the probability of observing the given data.

3.5.2 Results for Bernoulli

Theorem 3.5.1 MLE for Bernoulli

Given a set of n i.i.d. (independent and identically distributed) observations $D = \{w_1, \ldots, w_n\}$ from a Bernoulli distribution with parameter θ. The Maximum Likelihood Estimate (MLE) for θ is the sample frequency of successes.
If α_0 is the number of successes (observations equal to 1), then the estimate is given by:

$$\hat{\theta}_{MLE} = \frac{\alpha_0}{n}$$

Derivation for Bernoulli: For independent Bernoulli trials with outcomes $w_i \in \{0, 1\}$, the likelihood is:

$$L(\theta|D) = \prod_{i=1}^{n} P(w_i|\theta) = \prod_{i=1}^{n} \theta^{w_i}(1-\theta)^{1-w_i}$$

Let $\alpha_0 = \sum_{i=1}^{n} w_i$ (number of 1's) and $\alpha_1 = n - \alpha_0$ (number of 0's). Then:

$$L(\theta|D) = \theta^{\alpha_0}(1-\theta)^{\alpha_1}$$

Taking the logarithm (which preserves the maximum since log is monotonically increasing):

$$\ell(\theta) = \log L(\theta|D) = \alpha_0 \log(\theta) + \alpha_1 \log(1-\theta)$$

To find the maximum, take the derivative and set to zero:

$$\frac{d\ell}{d\theta} = \frac{\alpha_0}{\theta} - \frac{\alpha_1}{1-\theta} = 0$$

Solving:

$$\begin{aligned} \frac{\alpha_0}{\theta} &= \frac{\alpha_1}{1-\theta} \\ \alpha_0(1-\theta) &= \alpha_1\theta \\ \alpha_0 &= \theta(\alpha_0 + \alpha_1) \\ \hat{\theta}_{MLE} &= \frac{\alpha_0}{\alpha_0 + \alpha_1} = \frac{\alpha_0}{n} \end{aligned}$$

To verify this is a maximum, check the second derivative:

$$\frac{d^2\ell}{d\theta^2} = -\frac{\alpha_0}{\theta^2} - \frac{\alpha_1}{(1-\theta)^2} < 0$$

Since the second derivative is negative for $\theta \in (0, 1)$, this confirms a maximum

3.5.3 Multivalued case

Now I presented the formula just for two possible cases using the Bernoulli distribution, and for multivalued cases? WE HAVE THE *multinomial distribution*

Note:

Multinomial distribution is $P(X^n = \alpha_i \mid \theta) = c_{\alpha_i} \prod_i \theta_i^{\alpha_i}$. where α_i is the number of i in the sequence and c_{a_i} is a combinatorial constant not depending on θ

Theorem 3.5.2 MLE for Discrete Distributions

The Maximum Likelihood Estimate (MLE) for the probability θ_i of each outcome is its observed sample frequency.
If α_i is the number of times the i-th outcome has been observed in the n trials, then the estimate for its probability is:

$$\hat{\theta}_i = \frac{\alpha_i}{n}$$

Proof: Basta exercice

Corollary 3.5.1 MLE for Naïve Bayes Parameters

As a direct consequence of the main theorem, the Maximum Likelihood Estimates for the parameters of a Naïve Bayes classifier are also given by their sample frequencies:

1. (π_k) The MLE for the prior probability of a class y_k is its relative frequency in the dataset.

$$\hat{\pi}_k = P(Y = y_k) = \frac{\#\mathcal{D}\{Y = y_k\}}{|\mathcal{D}|}$$

2. (θ_{ijk}) The MLE for the conditional probability of a feature X_i taking the value x_{ij} given a class y_k, is the relative frequency of that feature value within the subset of data belonging to class y_k.

$$\hat{\theta}_{ijk} = P(X_i = x_{ij} | Y = y_k) = \frac{\#\mathcal{D}\{X_i = x_{ij} \wedge Y = y_k\}}{\#\mathcal{D}\{Y = y_k\}}$$

3.6 Document classification with Bag of Words (BoW) approach

The Bag-of-Words (*BoW*) model is a technique for document classification, which involves assigning a document to a predefined category (like Sport, politics, Tech, ect...). The core idea is to treat a doc not like a sorted sequence of phrases but like a simple Bag where words have no order or grammar, in this technique the primary focus is on capturing the occurrence frequency of each word within the document (this is a very surface level method that doesn't consider things like sarcasm and negation).
So we assume that:

- words are the elementary value of events (X_i is the i-th word in the document): $\theta_{i,j,k} = P(X_i = x_j | Y = y_k)$, where x_j is the j-th distinct word contained somewhere in the document and y_k is the k-th category[2].
- events are independent (given the category [3]): $\forall i, j, m, n, k. i \neq m.\ P(X_i = x_j | X_m = x_n, Y = y_k) = \theta_{i,j,k}$
- distribution is independent from the position: $\forall i, m, j, k.\ \theta_{ijk} = \theta_{mjk}$

[2] Words and categories aren't ordered in any particular way.

[3] Obviously, knowing a word in the document will give information regarding the type of the document (this is the notion that the model relies on), thus changing the distribution for all the other positions. For this reason we say that events are independent *only* if the category is considered known.

3.6.1 Training and classification

Event X_i = i-th word in the document: a discrete random variable assuming as many values as words in the language

$$\theta_{i,j,k} = P(X_i = x_j \mid Y = y_k)$$

In words: "the probability that in a document of the category y_k the word x_j appears at position i", but for the BoW technique we assume that we have a distribution independent from the position and that all event are independents so

$$\theta_{i,j,k} = \theta_{m,j,k} = \theta_{j,k}$$

For this reason, we can just consider the stochastic variable X = "word randomly extracted from document", with $\theta_{j,k} = P(X = x_j | Y = y_k)$. Then with discrete random variables X, Y let's define the training and classification:

- **Training**:
  - for any possible value y_k of Y, estimate the prior

    $$\pi_k = P(Y = y_k)$$

  - for each distinct word in the document x_j, estimate the likelyhood:

    $$\theta_{jk} = P(X = x_j \mid Y = y_k)$$

    (the condition prob that a certain word x_j appears in a doc, knowing that that doc belongs to category y_k)
- **Classification of** $a^{\texttt{new}} = \langle a_1, \ldots, a_n \rangle$ (new sequence of words):

$$\begin{aligned} Y^{\texttt{new}} &= \arg\max_{y_k} P(Y = y_k) \cdot \prod_{i=1}^{n} P(X = a_i | Y = y_k) \\ &= \arg\max_{k} \pi_k \cdot \prod_{i=1}^{n} \theta_{j(i),k} \end{aligned}$$

  where $j(i)$ is such that $x_{j(i)} = a_i$.

The probabilities needed for the classifier are estimated from the dataset using Maximum Likelihood Estimates (MLE's):

- The MLE for the prior probability of a class, π_k, is the fraction of the documents in the training set that belong to category y_k
- The MLE for the conditional probability of a word, $\theta_{\texttt{word},k}$, is the frequency of that word within all documents belonging to category y_k

The classification formula involves multiplying many small probabilities, which can lead to numerical instability (underflow). To solve this, we can maximize the logarithm of the likelihood instead, since the logarithm is a monotonically increasing function and will not change the location of the maximum.
The classification rule becomes:

$$Y^{\texttt{new}} = \arg\max_{y_k} \left(\log(\pi_k) + \sum_i \log(\theta_{j(i)k}) \right)$$

This can be further simplified by grouping identical words (values of i with the same $j(i)$). If n_j is the number of occurrences of the unique word x_j in the new document, the sum becomes a weighted sum:

$$\sum_i \log(\theta_{j(i)k}) = \sum_j n_j \cdot \log(\theta_{jk})$$

This final expression can be elegantly interpreted as a dot product in a high-dimensional vector space where each word of the vocabulary is a dimension

3.6.2 Dot product, correlation, Cosine similarity

the core idea is manipulating texts in numerical vectors. For doing this, you can create a vectorial space at $n-$dimension where each dimension correspond to a word to one unique word of vocabulary. HEre is the training and classification:

- **Training** For any category k build a "spectral" vector

$$s_k = \langle \log_{(\theta_{jk})} \rangle$$

  θ_{jk} = frequency of the word j in documents of the category k

  For each category (ex. "Sport") we'll create a spectral vector where each componet of this vector is the log of the probs ($\log(\theta_{jk})$) of a specific category. This vector represent the "profile" or carateristic direction of that category in the space of words

- **classification** Given a new doc, compute a vector

$$d = \langle n_j \rangle_{j \in \texttt{words}}$$

  and classify the doc as

$$\arg\max_k d \cdot s_k = \sum_j d_j \cdot s_{jk}$$

  so we're searching for the value of k that leads to the highest *correlation* between the two vectors.

geometrically and analytically

Definition 3.6.1: Dot product - Geometric def

We define dot product

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| \cdot |\mathbf{b}| \cos(\theta)$$

Definition 3.6.2: Dot product - Analytic definition

given $\mathbf{a} = (a_1, a_2, \ldots, a_n), \mathbf{b} = (b_1, b_2, \ldots, b_n)$

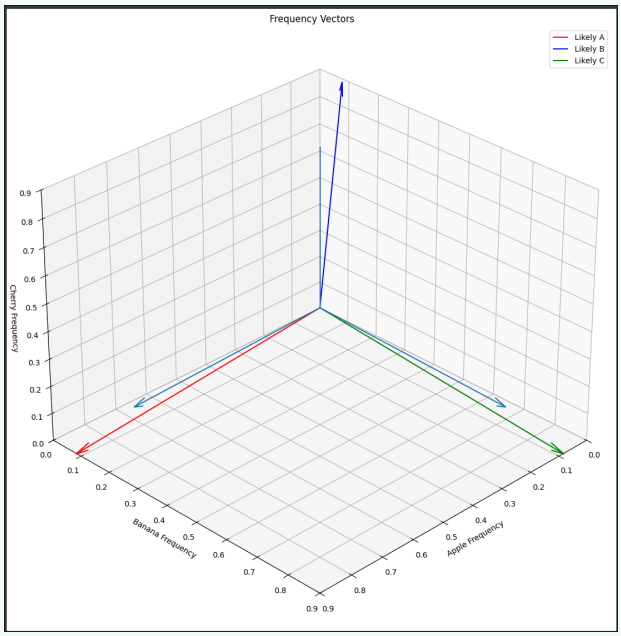
$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

The dot product can be influenced by the length of docs (a longer doc can have high frequencies). The cosine similarity resolve this problem normalizing the dot procd respect to the lenght of two vectors

$$S_c(\mathbf{a}, \mathbf{b}) = \cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|}$$

the results is a value $\in [-1, 1]$ that measures the "similarity of direction" ignoring the length. A value of 1 means that the vectors points in the same direction

Example 3.6.1 (Vector correlation maximization)



This is an example in 3 dimensions showing how the BoW approach can be seen as minimizing the angle between vectors (or maximizing their correlation)

Theorem 3.6.1 Equivalence of Dot Product Definitions

The geometric definition of the dot product, $\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos(\theta)$, is equivalent to the analytic definition, $\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i$.

Proof: The proof begins with the *Carnot's theorem*, which generalizes the Pythagorean theorem. Consider the triangle formed by the vectors $\vec{a}$, $\vec{b}$, and their difference, $\vec{a} - \vec{b}$.
The Law of Cosines states:

$$|\vec{a} - \vec{b}|^2 = |\vec{a}|^2 + |\vec{b}|^2 - 2|\vec{a}||\vec{b}| \cos(\theta)$$

Here, the term $|\vec{a}||\vec{b}| \cos(\theta)$ corresponds to the geometric definition of the dot product, $\vec{a} \cdot \vec{b}$. By rearranging the formula algebraically, we can express the dot product in terms of vector magnitudes:

$$\vec{a} \cdot \vec{b} = \frac{|\vec{a}|^2 + |\vec{b}|^2 - |\vec{a} - \vec{b}|^2}{2}$$

Next, we substitute the analytic definition of the squared magnitude of a vector. For simplicity, let's consider the two-dimensional case where $\vec{a} = [a_1, a_2]$ and $\vec{b} = [b_1, b_2]$. We have:

$$|\vec{a}|^2 = a_1^2 + a_2^2,$$
$$|\vec{b}|^2 = b_1^2 + b_2^2,$$
$$|\vec{a} - \vec{b}|^2 = (a_1 - b_1)^2 + (a_2 - b_2)^2.$$

Substituting these into the dot product expression yields:

$$\vec{a} \cdot \vec{b} = \frac{(a_1^2 + a_2^2) + (b_1^2 + b_2^2) - \left((a_1 - b_1)^2 + (a_2 - b_2)^2\right)}{2}.$$

Expanding the squared terms in the numerator:

$$\vec{a} \cdot \vec{b} = \frac{a_1^2 + a_2^2 + b_1^2 + b_2^2 - (a_1^2 - 2a_1b_1 + b_1^2 + a_2^2 - 2a_2b_2 + b_2^2)}{2}.$$

Simplifying by canceling out the squared terms:

$$\vec{a} \cdot \vec{b} = \frac{2a_1b_1 + 2a_2b_2}{2}.$$

This simplifies to the final result:

$$\vec{a} \cdot \vec{b} = a_1b_1 + a_2b_2.$$

This is the analytic definition of the dot product for the two-dimensional case. The proof can be generalized to n dimensions, demonstrating the equivalence of the geometric and analytic definitions of the dot product.

3.7 The linear nature of Naive Bayes (boolean case)

If both X_i and Y are boolean variables, it's possible to transform the Naive Bayes formula using certain properties of boolean functions to obtain a *linear* expression.

Theorem 3.7.1 Linarity of classifier Naive bayes for boolean feature

Given X_i, Y booleans, the classification of a new $\vec{x} = \langle x_1, \ldots, x_n \rangle$ happens comparing the probs that x belong to class 1 and 0:

$$\frac{P(Y = 1 \mid X_1 \ldots X_n = \vec{x})}{P(Y = 0 \mid X_1 \ldots X_n = \vec{x})} = \frac{P(Y = 1) \prod_i P(X_i = x_i \mid Y = 1)}{P(Y = 10 \prod_i P(X_i = x_i \mid Y = 0} \geqslant 1$$

if the results is $\geqslant 1$ we choose the class one.
Passing to logarithms we have:

$$\log \frac{P(Y = 1)}{P(Y = 0)} + \sum_i \frac{X_i = x_i \mid Y = 1}{X_i = x_i \mid Y = 0} \geqslant 0$$

if the results is $\geqslant 0$ we choose the class one.
Now define the conditional probs loke $\theta_{ik} = P(X_i = 1 \mid Y = k)$. Of consequence, $P(X_i = 0 \mid Y = k) = 1 - \theta_{ik}$. The generic therm of the sum, $\log \frac{P(X_i = x_i|Y=1)}{P(X_i = x_i|Y=0)}$, using the fact that for a boolean function $f(x) = xf(1) + (1-x)f(0)$ we have:

$$\sum_i \left(x_i \cdot \log \frac{P(X_i = 1|Y = 1)}{P(X_i = 1|Y = 0)} + (1 - x_i) \cdot \log \frac{P(X_i = 0|Y = 1)}{P(X_i = 0|Y = 0)} \right)$$

with θ, the expression becomes:

$$\log \frac{P(Y = 1)}{P(Y = 0)} + \sum_i \left(x_i \cdot \log \frac{\theta_{i1}}{\theta_{i0}} + (1 - x_i) \cdot \log \frac{1 - \theta_{i1}}{1 - \theta_{i0}} \right) \geq 0$$

that is a linear expression in the set of features x_i in the form

$$w_0 + \sum_i w_i x_i \geqslant 0$$

Classification algorithms based on a linear combination of the features values. Every feature is evaluated independently from the others and contributes to the result in a linear way, with a suitable weight (that is a parameter of the model, to be estimated). For instance, if the features are pixels of some image, we may use linear methods only up some normalization (in position and dimension) of the object to be recognized

3.8 Gaussian Naive Bayes

The standard Naïve Bayes algorithm is designed for discrete features. However, in many real-world scenarios, features can be continuous, such as the height of an individual, the temperature, or the color intensity of a pixel. For continuous variables, the probability of observing a specific value is infinitesimally small, making the pointwise probability $P(X_i|Y)$ effectively null.

To address this challenge, Gaussian Naïve Bayes extends the classifier by making a crucial assumption: that the data for each continuous feature, conditioned on a specific class, follows a **Gaussian (or Normal) distribution** [4]. Instead of estimating a probability for each value, we estimate the parameters of this distribution (the mean and variance) from the training data.

Definition 3.8.1: Gaussian Distribution

The *Gaussian Distribution* is a continuous probability distribution characterized by its mean μ and variance σ^2. Its probability density function (PDF) is given by:

$$p(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where:

- The mean $E[X] = \mu$ defines the center of the distribution.
- The variance $\text{Var}[X] = \sigma^2$ defines the spread of the distribution.

3.8.1 Accuracy, Precision and Recall

Definition 3.8.2: Accuracy

$$\texttt{Accuracy} = \frac{TP + TN}{ALL}$$

number of correctly classified samples

Definition 3.8.3: Precision

$$\texttt{Precision} = \frac{TP}{TP + FP}$$

percentage of true prositives over predicted ones

Definition 3.8.4: Recall

$$\texttt{Recall} = \frac{TP}{TP + FN}$$

percentage of true positives over all positives

Definition 3.8.5

$$F1 = 2\frac{Precision \cdot Recall}{Precision + Recall}$$

harmonic mean of precision and recall

3.8.2 Descriptive parameters of the model

We assume (inductive bias) that for every value y_k of Y the random variable $P(X_i|Y = y_k)$ has a Gaussian distribution

$$\mathcal{N}(x \mid \mu_{ik}, \sigma_{ik}) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} e^{-\frac{(x-u_{ik})^2}{2\sigma_{ik}^2}}$$

[4] The reason we make this assumption is because, given a mean and a variance, the Gaussian is the distribution with the highest entropy. This means that it makes the weakest assumptions, distributing the probability in the most even way possible for all possible results.

- **Learning**: estimate the values of the parameters μ_{ik}, σ_{ik} and $\pi_k = P(Y = y_k)$
- **Classification** of $x^{new} = \langle a_1, \ldots, a_n \rangle$

$$Y^{\texttt{new}} = \arg\max_{y_k} P(Y = y_k) \cdot \prod_i P(X_i = a_i \mid Y = y_k)$$
$$= \arg\max_k \pi_k \cdot \prod_i \mathcal{N}(a_i \mid \mu_{ik}, \sigma_{ik})$$

3.8.3 MLE for Gaussian Parameters

Given a set of training data, the Maximum Likelihood Estimates (MLE) for the parameters of the Gaussian distribution for each feature X_i and class y_k are the sample mean and sample variance, calculated from the subset of data belonging to class y_k.

- **Mean Estimate (μ_{ik}):** The mean value of X_i for samples with label $Y = y_k$. Formally:

$$\mu_{ik} = \frac{\sum_j X_i^j \delta(Y^j = y_k)}{\sum_j \delta(Y^j = y_k)}$$

- **Variance Estimate (σ_{ik}^2):** The variance of X_i for samples with label $Y = y_k$. Formally:

$$\sigma_{ik}^2 = \frac{\sum_j (X_i^j - \mu_{ik})^2 \delta(Y^j = y_k)}{\sum_j \delta(Y^j = y_k)}$$

where j ranges over all samples in the training set, and $\delta(Y^j = y_k)$ is the indicator function:

$$\delta(Y^j = y_k) = \begin{cases} 1 & \text{if } Y^j = y_k \\ 0 & \text{otherwise} \end{cases}$$

3.9 Logistic Regression

Logistic regression is a discriminative machine learning method used primarily for classification problems

3.9.1 Core idea

- Naive Bayes allows us to compute $P(Y|X)$ after having learned $P(Y)$ and $P(X|Y)$
- Why not directly learn $P(Y|X)$?

Theorem 3.9.1 The shape of $P(Y \mid X)$

hypotesis:

- Y boolean random variable
- X_i continuous random variable
- X_i independent from each other Y
- $P(X_i \mid Y = k)$ have Gaussian distributions $N(\mu_{i_k}, \sigma_i)$(Warning not $\sigma_i k$!)
- Y has a Bernoulli distribution (π)

then

$$P(Y = 1 \mid X = \langle x_1 \ldots x_n \rangle) = \frac{1}{1 + \exp(w_0 + \sum_i w_i x_i)} \tag{3.4}$$

Proof: By hypothesis, the conditional probability of a feature X_i is given by a Gaussian PDF:

$$P(X_i|Y=k) = \mathcal{N}(x_i; \mu_{ik}, \sigma_i) = \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(x_i-\mu_{ik})^2}{2\sigma_i^2}}$$

We want to find the shape of $P(Y=1|\vec{x})$. Using the definition of conditional probability and the law of total probability, we have:

$$\begin{aligned} P(Y=1|\vec{x}) &= \frac{P(Y=1)P(\vec{x}|Y=1)}{P(\vec{x})} \\ &= \frac{P(Y=1)P(\vec{x}|Y=1)}{P(Y=1)P(\vec{x}|Y=1) + P(Y=0)P(\vec{x}|Y=0)} \end{aligned}$$

Dividing the numerator and denominator by $P(Y=1)P(\vec{x}|Y=1)$ gives:

$$\begin{aligned} P(Y=1|\vec{x}) &= \frac{1}{1 + \frac{P(Y=0)P(\vec{x}|Y=0)}{P(Y=1)P(\vec{x}|Y=1)}} \\ &= \frac{1}{1 + \exp\left(\ln\left(\frac{P(Y=0)P(\vec{x}|Y=0)}{P(Y=1)P(\vec{x}|Y=1)}\right)\right)} \end{aligned}$$

Using the Naïve Bayes assumption of conditional independence $P(\vec{x}|Y) = \prod_i P(x_i|Y)$ and substituting $P(Y=1) = \pi$, we get:

$$P(Y=1|\vec{x}) = \frac{1}{1 + \exp\left(\ln\frac{1-\pi}{\pi} + \sum_i \ln\frac{P(x_i|Y=0)}{P(x_i|Y=1)}\right)}$$

Substituting the Gaussian PDF and simplifying the term inside the summation leads to a linear function of x_i:

$$P(Y=1|\vec{x}) = \frac{1}{1 + \exp\left(\ln\frac{1-\pi}{\pi} + \sum_i \left(\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}x_i + \frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2}\right)\right)}$$

This expression has the form $\frac{1}{1+\exp(z)}$, where z is a linear combination of the features x_i. By defining w_0 and w_i as the appropriate combinations of the Gaussian parameters (μ, σ, π), we obtain the logistic form:

$$P(Y=1|\vec{x}) = \frac{1}{1 + \exp(w_0 + \sum_i w_i x_i)}$$

3.9.2 Training for logistic regression

So, logistic regression assumes

$$P(Y=1 \mid X = \langle x_1 \dots x_n \rangle) = \frac{1}{1 + \exp\left(w_0 + \sum_i w_i x_i\right)}$$

and directly tries to estimate the parameters w_i (the change in sign is ininfluent).

Definition 3.9.1: Logistic function

It's a very important funcition and it's called logistic function:

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

With this def we know that $P(Y=1 \mid x, w) = \sigma(w_0 + \sum_i x_i w_i)$

Training via Maximum Likelihood Estimation

Given independent training samples $\langle x^\ell, y^\ell \rangle$, we want to find the parameters w that **maximize** the probability of observing our training data.
The probability of all samples is:

$$\prod_\ell P(y^\ell | x^\ell, w) = \prod_\ell P(y^\ell = 1 | x^\ell, w)^{y^\ell} \cdot P(y^\ell = 0 | x^\ell, w)^{(1-y^\ell)}$$

Note:

This formula works because when $y^\ell = 1$, only the first term contributes (the second becomes 1), and vice versa when $y^\ell = 0$.
Since $P(Y = 0|x, w) = 1 - P(Y = 1|x, w)$, we can write:

$$\prod_\ell P(y^\ell | x^\ell, w) = \prod_\ell \sigma(w_0 + \sum_i w_i x_i^\ell)^{y^\ell} \cdot (1 - \sigma(w_0 + \sum_i w_i x_i^\ell))^{(1-y^\ell)}$$

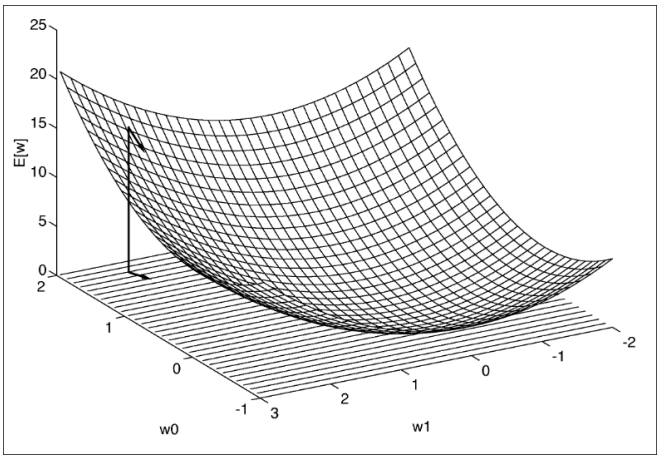
Log-Likelihood

Instead of maximizing the product directly, we **maximize the logarithm** of this probability (which is equivalent since log is monotonic):

$$\ell(w) = \sum_\ell \log P(y^\ell | x^\ell, w) = \sum_\ell \left(y^\ell \cdot \log P(Y = 1 | x^\ell, w) + (1 - y^\ell) \cdot \log P(Y = 0 | x^\ell, w) \right)$$

3.10 The Gradient Technique

The training set is fixed, so the loss function depends on the parameters of the model. Unfortunately, there is no analytic solution for the previous optimization problem. So, we use iterative optimization methods, like the **gradient technique**:



simple model with 2 parameters

Definition 3.10.1: Gradient

The *gradient* of the log-likelihood function $\ell(w)$ (which the slide denotes as $E[w]$) is a vector composed of the partial derivatives of $\ell(w)$ with respect to each parameter w_i. It points in the direction of the steepest ascent on the likelihood surface.

$$\nabla_w[\ell] = \left[\frac{\partial \ell}{\partial w_0}, \frac{\partial \ell}{\partial w_1}, \ldots, \frac{\partial \ell}{\partial w_n} \right]$$

This is a *general* technique (applicable to any differentiable function) to find a direction that lowers the loss function (the direction of greatest descent) and to subsequently perform a "step" (determined by the learning rate) in that direction.

Note:

This is a *local* technique: it's not given that the minimum we find is global, because it could just be a local one. For logistic regression, we can be certain it's global, but for more complicated models this isn't always true. For the latter, it's possible to balance out these fake minimals by having lots of parameters, because this way it's possible to simply ignore the outliers and trust the value found by the most parameters.

The correct direction is calculated using - you guessed it - the gradient (although tecnicaly in the opposite direction). Repeating these stepps:

Theorem 3.10.1 Training Rule (Gradient Ascent)

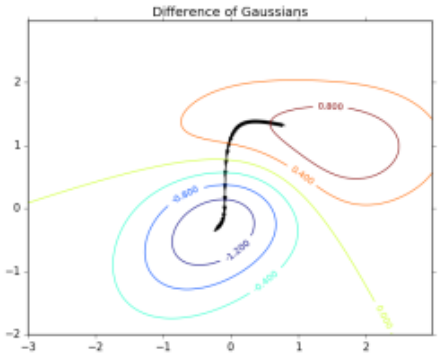
To find the parameters w that maximize $\ell(w)$, we use *Gradient Ascent*. We start with an initial guess for w and iteratively update the parameters by taking small steps in the direction of the gradient. The update rule for a single weight w_i is:

$$\Delta w_i = \eta \cdot \frac{\partial \ell}{\partial w_i}$$

$$w_i = w_i + \Delta w_i$$

where η (denoted μ in the slide) is the *learning rate*, a small constant that controls the size of each step. This process is repeated until the parameters converge.

The concept is conceptually simple. We can visualize the model's error function, $E(w)$, as a topographical map. On this map, the altitude represents the error (the vertical axis), and the goal is to find the lowest point (the minimum error).
To achieve this, at iteration 0, we start from a random point. We then calculate the gradient ($\nabla E(w)$) at that position. The gradient, by definition, points in the direction of steepest ascent (the "maximum climb"). Therefore, to descend ('go down') and minimize the error, we must move in the opposite direction of the gradient



In ideal case, the error surface is convex, this is fundamentally 'cause it means that there is just a single global minimum. Fortunately we have THAT:

Theorem 3.10.2 Convexity of the Error Surface for Logistic Regression

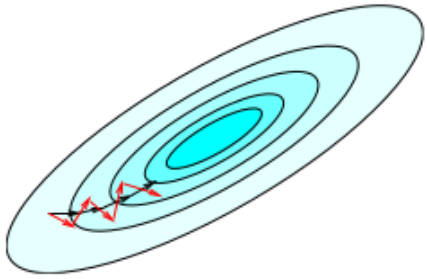
Let $E(w) = -\ell(w)$ be the error function for a logistic regression model, defined as the *negative log-likelihood* of the training data.
This error function $E(w)$ is a *convex function* with respect to the parameter vector w.

Note:

This is a critical property of the logistic regression model. Because the error surface is convex, it possesses a single global minimum and no local minima. This guarantees that iterative optimization methods, such as Gradient Descent, will converge to the unique optimal solution, regardless of the initial parameter values.

How many points to use for measuring error and gradient? there are two ways:

- **Full Batch**: all the features in training set are used for calculating the gradient
- Online: one sample at a time. Gradient zig-zags around the direction of the steepest descent.
- *mini-batch*: random subset of training samples, a good compromise



SGD (Stochastic Gradient Descent): the direction calculated by the Mini-Batch isn't necessarily the best, but the hope is that the sum of the errors for all the considered parameters rounds out.

3.10.1 Calculating the Gradient for logistic regression

First a small review of the other formulas:

Definition 3.10.2: probability that sample ℓ is in category $Y = 1$

$$P(Y = 1|x^\ell, w) = \sigma(w_0 + \sum_i x_i x_i^\ell) = \alpha_l$$

then:

Definition 3.10.3: Likelihood Log-likelihood$l(w)$

$$\ell(w) = \sum_\ell \log P(Y = y^\ell|x^\ell, w) = \sum_\ell \left(y^\ell \log(\alpha^\ell) + (1 - y^\ell)\log(1 - \alpha^\ell)\right) \tag{3.5}$$

Theorem 3.10.3 Gradient of the Log-Likelihood for Logistic Regression

The partial derivative of the log-likelihood function $l(w)$ with respect to a single weight w_i is given by the sum over all samples:

$$\frac{\partial l(w)}{\partial w_i} = \sum_\ell x_i^\ell \cdot (y^\ell - \alpha^\ell) \tag{3.6}$$

Proof: We want to compute the partial derivative $\frac{\partial l(w)}{\partial w_i}$ of the log-likelihood function $l(w)$. First, let's define the components, where $\alpha^\ell = \sigma(z)$ and $z = w_0 + \sum_i w_i x_i^\ell$.
Step 1: Derivative of $\log(\alpha^\ell)$ *w.r.t.* z

$$\frac{\partial \log(\alpha^\ell)}{\partial z} = \frac{1}{\alpha^\ell}\frac{\partial \alpha^\ell}{\partial z}$$

Using the property $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$, which is equivalent to $\alpha^\ell(1 - \alpha^\ell)$, we get:

$$\frac{\partial \log(\alpha^\ell)}{\partial z} = \frac{1}{\alpha^\ell}(\alpha^\ell(1 - \alpha^\ell)) = 1 - \alpha^\ell$$

(Note: The slide shows an intermediate step $\frac{\exp(-z)}{1+\exp(-z)}$ which simplifies to $1-\alpha^\ell$).
Step 2: Derivative of $\log(1-\alpha^\ell)$ *w.r.t.* z First, we simplify $\log(1-\alpha^\ell)$:

$$\log(1-\alpha^\ell) = \log\left(1 - \frac{1}{1+\exp(-z)}\right) = \log\left(\frac{\exp(-z)}{1+\exp(-z)}\right)$$

$$= \log(\exp(-z)) - \log(1+\exp(-z)) = -z + \log(\alpha^\ell)$$

Now, we differentiate this expression with respect to z, using the result from step 1:

$$\frac{\partial \log(1-\alpha^\ell)}{\partial z} = \frac{\partial}{\partial z}(-z + \log(\alpha^\ell)) = -1 + (1-\alpha^\ell) = -\alpha^\ell$$

Step 3: Final Gradient Calculation (using Chain Rule) We recall the log-likelihood function:

$$l(w) = \sum_\ell \left(y^\ell \log(\alpha^\ell) + (1-y^\ell)\log(1-\alpha^\ell)\right)$$

We apply the chain rule: $\frac{\partial l(w)}{\partial w_i} = \frac{\partial l(w)}{\partial z}\frac{\partial z}{\partial w_i}$.
First, $\frac{\partial z}{\partial w_i} = x_i^\ell$.
Second, we find $\frac{\partial l(w)}{\partial z}$ by differentiating the sum:

$$\frac{\partial l(w)}{\partial z} = \sum_\ell \left(y^\ell \frac{\partial \log(\alpha^\ell)}{\partial z} + (1-y^\ell)\frac{\partial \log(1-\alpha^\ell)}{\partial z}\right)$$

Substituting the results from steps 1 and 2:

$$\frac{\partial l(w)}{\partial z} = \sum_\ell \left(y^\ell(1-\alpha^\ell) + (1-y^\ell)(-\alpha^\ell)\right)$$

$$= \sum_\ell (y^\ell - y^\ell\alpha^\ell - \alpha^\ell + y^\ell\alpha^\ell) = \sum_\ell (y^\ell - \alpha^\ell)$$

Finally, combining the parts with the chain rule:

$$\frac{\partial l(w)}{\partial w_i} = \frac{\partial l(w)}{\partial z}\frac{\partial z}{\partial w_i} = \left(\sum_\ell (y^\ell - \alpha^\ell)\right) x_i^\ell$$

(Note: The slide applies the chain rule slightly differently by summing at the end, leading to the same result presented):

$$\frac{\partial l(w)}{\partial w_i} = \sum_\ell \left(\frac{\partial l^\ell(w)}{\partial z}\frac{\partial z}{\partial w_i}\right) = \sum_\ell \left((y^\ell(1-\alpha^\ell) - (1-y^\ell)\alpha^\ell)x_i^\ell\right)$$

$$= \sum_\ell ((y^\ell - y^\ell\alpha^\ell - \alpha^\ell + y^\ell\alpha^\ell)x_i^\ell) = \sum_\ell (y^\ell - \alpha^\ell)x_i^\ell$$

Thus, we arrive at the final formula:

$$\frac{\partial l(w)}{\partial w_i} = \sum_\ell x_i^\ell \cdot (y^\ell - \alpha^\ell)$$

Per la regressione logistica E' possibile calcolare il gradiente (proporzionale alla distanza dal valore corretto e all'intensita' del parametro relativo)2

Note:

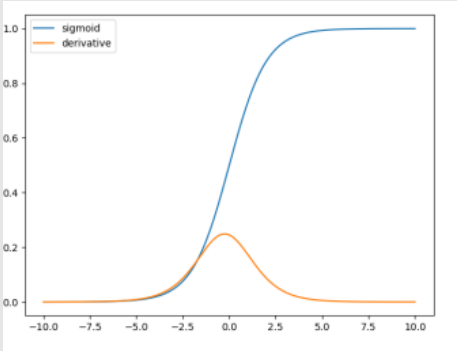
previsuly we proved that:

$$\frac{\partial \log \alpha^{\ell}}{\partial z} = \frac{\partial \sigma(z)}{\partial z} = \frac{1}{\sigma(z)} \frac{\partial \sigma(z)}{\partial z} = 1 - \sigma(z)$$

hence:

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$$

the slope (derivative) of the sigmoid at a point z can be calculated simply using the value of the sigmoid itself at that point. But let's see the graph



we can see the the derivate of the logistic function is very flat! When the input z is big (negatively of positively) the line of sigmoid is flat this lead the derivative to be almost zero.
If the derivative is zero, the weight updates become zero ($w_i = w_i + 0$), and the model stops learning. This phenomenon, in which the gradients become very small, is known as the *Vanishing Gradient*

The learning process

The learning process for logistic regression is iterative. Since we cannot find an analytical solution that maximizes the log-likelihood $l(w)$, we must find the optimal parameters w using an iterative method like Gradient Ascent. The process involves starting with an initial guess for the weights (e.g., all zeros) and repeatedly applying an *update operation*. Each iteration, or step, moves the weights in the direction of the gradient, thereby increasing the log-likelihood.

Definition 3.10.4: Update Rule (Gradient Ascent)

The update rule for a single weight w_i at each iteration is given by:

$$w_i \leftarrow w_i + \mu \sum_{\ell} x_i^{\ell} \cdot (y^{\ell} - P(Y = y^{\ell} | x_i; w_i))$$

Where:

- w_i is the weight for the i-th feature.
- μ is the *learning rate*, a small constant that controls the size of each step.
- $\sum_{\ell}(\dots)$ is the gradient component $\frac{\partial l(w)}{\partial w_i}$, summed over all training samples ℓ.
- $(y^{\ell} - P(\dots))$ is the *error* for sample ℓ: the difference between the true label and the model's prediction.

This operation is iterated until the model converges, which can be defined as the point where the accuracy on a held-out testing dataset is satisfactory, or when the magnitude of the update (the increment) falls below a small threshold ϵ.

Regularization

To improve the model's ability to generalize to new data and prevent overfitting, a *regularizer* (also known as a prior) is frequently added to the update rule. This acts as a penalty on large weights.

Definition 3.10.5: Update Rule with Regularization

The modified update rule includes a decay term:

$$w_i \leftarrow w_i - \mu\lambda|w_i| + \mu \sum_{\ell} x_i^{\ell} \cdot (P(Y = y^{\ell}|x_i; w_i) - y^{\ell})$$

Note:

Note: The slide shows a sign flip in the gradient term, which implies a switch from maximizing log-likelihood (Gradient Ascent) to minimizing a negative log-likelihood (Gradient Descent). The logic remains the same: the regularizer always pushes the weight towards zero.

The new term $-\mu\lambda|w_i|$ is a penalty proportional to the magnitude of the weight w_i, controlled by a new hyperparameter λ.
This regularization has two main benefits:

- It helps to keep the parameter values w_i small, or close to 0.
- By penalizing complex models (those with large weights), it tends to reduce overfitting and improve the model's performance on unseen data.

Chapter 4

Neural Networks

Only known way to implement deep learning.
An artificial neuron has a set of inputs that are summed with a bias, and the output is calculated using an ctivation function. invented in the '50 with the name 'perceptron', the activation function was discrete (0 or 1). This method is obsolete because it doesn't permit training. The function musn't be linear, because we want to sequencially compose neurons and the composition of linear functions is just another linear function and deosn't add any complexity (could be calculated with just one neuron).
There are different activation functions (threshol, logistic, hyperbolic, rectified linear (newest used in AlexNet, big jump)).
The name neuron comes from the biological counterpart (obviously) which works in a similar way:

- Dendritic tree: connections with other neurons (synapses)
- In the body the inputs are summed and passed through the axon hilock, which performs a sort of thresholding before being passed to other neurons

Comparing artificial neural networks to our brains, the number of neurons can be similar ($2 \cdot 10^{10}$for the biggest models), but the sheer size doesn't necessarily correlate to a more intelligent system (other animals have bigger brains), we actually don't really know what else there is. The switching time for real neurons is actually slower than artificial ones, seen as it's a chemical reaction and not electrical. Each neuron is connected to meny other neurons in the brain (10^{4-5}), reaching reaction times < 100, so the brain isn't very deep (number of intermediate nodes) and it's very parallelised.

4.1 Topologies

Definition 4.1.1: Feed-forward

Acyclic networks with unidirectional data flux.

Note:

Our brains are cyclic networks.

4.1.1 Layers

The network is built by sets of structured neurons that are combined to build the whole.
For each dense layer, each neuron has an input, weights, a constant bias from which an output is calculated. This operation can be parallelised using the whole layer with a matrix of weights and a vector of biases, using the same inputs over all the neurons. The input is a vector in multiple dimensions (tensor), and after algebraic manupulations a new tensor is returned.

4.2 Features and deep features

TODO: manca una lezione intera

4.3 Expressivness

What functions can we calculate? Does a set of weights exist for a neural network to calculate a function? Does it have to be deep?

4.3.1 Single Layer

Can we implement a NAND function with a perceptron? If so, we can calculate all logical expressions, or simulate all circuits.

Note:

NAND is a complete set of logical operators, but they need to be composed to do so. Meaning a single layer being able to calculate this function does not imply that the perceptron is complete, as we'll later see.

Can we find two weights w_1, w_2 and a bias b such that

$$nand(x_1, x_2) = \begin{cases} 1 & \sum_i w_i x_i + b \geqslant 0 \\ 0 & \end{cases}$$

We can do this graphically looking at a plane. We want to be able to draw a hyperplane (a line in R^2) that divides the plane in two different areas, one for values that should equal 0 ($(1,1)$) or 1 ($(0,0),(0,1),(1,0)$).
TODO: drawing
A possible equation for such line is

$$-2x_1 - 2x_2 + 3 \geqslant 0$$

So the output is

$$output = \begin{cases} 1 & -2x_1 - 2x_2 + 3 \geqslant 0 \\ 0 & \end{cases}$$

What we can't calculate is the **XOR** or its complementary (equality). This is clear when looking at the plane
TODO: drawing
This is a limitation in linear methods (like logistic regression), because we need to compare both input features.
In conclusion, single layer perceptrons are not logically complete.

4.3.2 Multi-layer perceptron

Can we compute XOR by stacking perceptrons? Yessir, look at the image TODO: ahhhh
How deep does a logic network need to be to calculate a certain expression? We can look at the depth as the number of nested logical connectors, or the depth of the tree generated by the expression. In conjunctive or disjunctive normal form, all trees have depth of 3, and each logic formula can be transformed in one of these forms. To be honest, we can remove the negation layer and use only 2 layers with cooler gates.
So the single layer is cringe, but two layers can express any continuous function.

Note:

It's thanks to the activator function that the composition of layers adds expressivity, as composition of linear functions is always linear.

4.3.3 Deep layers

So why go deep? Two layers are already complete!

- Convolution layers (TODO: non ho capito)
- Transformers arent actually that deep, as transformer blocks are essentially made up of 3-4 layers, creating a total depth of 30-40 layers, which isn't actually that deep.

Note:

In some cases, transforming a logical expression into normal form can create an exponentially longer expression, making it a lot less efficient.

In short, deep layers can be more efficient than shallow ones in some cases

4.4 Training

4.4.1 Current loss

Suppose we have a neural network with some configurations of trhe parameters θ.
We can pass a **batch** of data in input to be processed in parallel and get an output that can be compared to an expected output. This is a **forward pass**.
Using gradient descent, we can minimize the loss function by changing certain parameters. This is called **backward pass** because we need to calculate the parameter's gradinets from back to front.
The cost of both passes is similar. During the forward pass, we need to memorise each tensor output in order to calculate gradients later on. So during training there's a higher memory use.

4.4.2 Backpropagation

To calculate the gradients we need, we use the **chain rule**:

$$h'(x) = f'(g(x)) * g'(x)$$

The derivative of a composition of a sequnce of function is the product of the derivatives of the individual functions. This function is iterable, so for each layer we multiply its derivative on their forward input (this is why we need to memorize it).
Binary thresholding has null derivative everywhere (its a step function), so it would ruing the chain.
Given linear layers combined by activator functions, at layer ℓ we have

$$a^l = \sigma(b^l + w^l \cdot x^l)$$

- a^l is the **activation vector**
- z^l is the **weighted input**
- $x^{l+1} = a^l, x^1 = x$

Partial derivative

$$\delta^l = \frac{\delta E}{\delta z^l}$$

We have to traverse the dense part. The parameters we're interested in are the parameters of the linear transformation. We transpose the matrix to calculate the layer at level $l + 1$.
We can calculate all partial derivatives with respect to z, but we don't want this, we want b. But in reality we just need to multiply by the derivative with respect to b of $z = ...$, which is one. So the two values are the same.
We can do the same for w, which has derivative x^l, which is the last layer's output ??.
We don't really need to know all the formulas, just the main concept.

Vanishing gradient problem

When going backwards, its possible for the gradient to become 0. In this case we don't have any movement. This is a problem for certain activator function. The sigmoid is the continuous version of a binary threshold with a very flat derivative. After many levels of backpropagation being multiplied by numbers less than 0.25, we can quickly go to 0 ???. The Relu function doesn't have this problem (derivative is 0 or 1)