AI Foundations | AIFo

Zusammenfassung

1. HISTORY

AI is a bigger concept to create intelligent machines that can ***simulate human thinking***. We focus on algorithms and applications where a ***computer learns from data without being programmed explicitly***. This is called ***machine Learning (ML)***.

1.1. THE MECHANICAL TURK

A fraudulent chess-playing machine constructed in 1770 which appeared to be able to play a strong game of chess against a human opponent (There was a person hidden inside)

1.2. WHAT IS A COMPUTER?

Everything that has encoded information belongs to Informatics. However, not everything that belongs to Informatics is a computer. A computer needs to be able to compute something. It performs a mapping from input to output (Input – Processing – Output).

- ***Input:*** Any sort of data/information *sensory inputs, bits, mechanical configuration, …*
- ***Processing:*** Any sort of non-trivial "calculation" (mapping) or information processing
- ***Output:*** Any sort of response *data, actions, new stat of a system, …*

There are biological, biologically inspired, electronic, mechanical, … Computers.
Examples: *Microprocessors, "computers", CPU, Mechanical implementations of transformations, the brain, …*

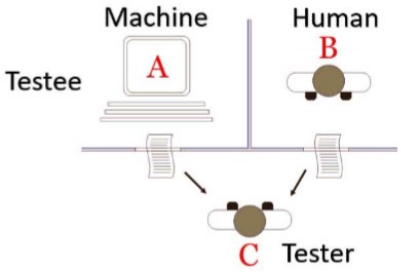
The oldest known analogue computer is the "Antikythera Mechanism" which was created around 100 BC. It was used to calculate astronomic positions and eclipses decades in advance.

1.3. THE MOST IMPORTANT DEVELOPMENTS IN INFORMATION SCIENCE IN THE LAST CENTURIES

- ***Shift from Analog to Digital*** Data Transmission and Storage *Signal Processing*
- ***Separation of Hardware and Software*** Information Processing *Computation*

Deep Implications: Programs are data, Algorithms can operate on programs.

1.4. TURING TEST (AKA IMITATION GAME), 1950



A way to test ***whether a machine has human intelligence or not.*** An Interrogator must find out via asked questions if a Machine or a Human answers him. The AI is intelligent if the human does not recognize it.

There are some limitations:

- The result ***depends on the tester***.
- A machine can ***imitate a human without having any understanding*** of what it is saying (ChatGPT).
- An intelligent system that ***solves a complex problem*** (too difficult for a human) would ***fail*** the test.

1.5. WHY DID THE DEEP-LEARNING REVOLUTION START ONLY RECENTLY?

- Someone had to try ***and*** succeed
- Computing ***Power*** was ***too low***
- The ***Amount of data*** was ***too small***
- Data was ***not shared***

2. NATURAL LANGUAGE PROCESSING (NLP), EMBEDDINGS

2.1. THE 4 INGREDIENTS OF MACHINE-LEARNING

- ***Data:*** Dataset including pre-processing
- ***Cost-Function (Loss):*** A formal expression for "good" and "bad" (MSE)
- ***Model:*** From linear model $\hat{y}_i = ax_i + b$ to a million-parameter Neural Network.
- ***Optimization Procedure:*** An algorithm that changes the parameters of the model such that the cost-function is minimized. (SGD)

2.2. HOW TO REPRESENT WORDS IN A COMPUTER

The meaning of words can be represented using ***vectors***.

2.2.1. One-hot representation

A one-hot vector is a vector with a single 1-value and all others set to 0. Count the number of different words. Then define one unique one-hot-vector per word.

It	rains	.	We	go	home	.	It	stops	raining	.
$\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$	$\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}$

2.2.1.1. Disadvantages

- ***High dimensional vector space:*** In an article with 100'000 Words, each word is represented by an 100'000-dimensional space.
- ***Sparse representation:*** The vectors are memory-inefficient, and algorithms cannot efficiently learn from such representations.
- ***No generalization:*** with one-hot vectors, all words are completely unrelated to each other. Therefore, we cannot generalize knowledge. One-hot representation does **not capture any aspect of the meaning** of a word.

2.2.2. Indexing

List of all the words (optionally alphabetically sorted), the index represents each word.

cat	0
mat	1
on	2
sat	3
the	4
.	5

"The cat sat on the mat." → $[4, \quad 0, \quad 3, \quad 2, \quad 4, \quad 1, \quad 5]$

This is the ***"dense equivalent"*** of one-hot encoding. Indexes are ***not more useful*** than one-hot vectors. But indexing is (often) used as a preprocessing step. These indices are then fed into a network which learns more useful representations.

2.2.3. Distributed representation

A word can be ***"defined" by context***. Words with ***similar semantics share some context*** (e.g., rat and cat are both animals). Distributed representations can be ***learned***. *Example below: Similar traits are coloured similarly / at the same positions. We can subtract the properties of "Man" from "King" and add "Woman" to get the word "Queen".*

Word to Vector

A ***mathematical function maps*** a ***word*** or the corresponding index (input) to a ***high dimensional vector*** (output). Machine-learning algorithms can ***learn*** this function. In neural networks this function is implemented in a ***embedding layer***.

Advantages of vectors

Ideally, a "good" embedding ***maps similar/related words to similar regions of the vector-space***. That is, nearby words have a semantic similarity. Once we have "good" vectors, we can do math. ***The dot-product*** (Skalarprodukt) is a measure of ***similarity***. Vectors also can be added/subtracted.

For specific tasks, it is sufficient to have a notion of similarity between words. The question "how to understand words" becomes ***"how to calculate similarity between words"***.

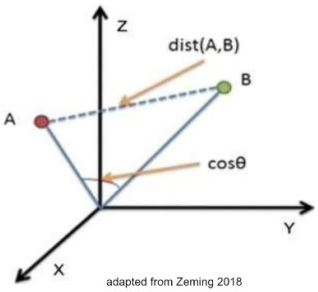
Basic properties of the dot product:

- The dot product between two vectors is ***maximal*** when they are both the same (going in the ***same direction***). If the two vectors both have ***norm / length 1***, then the ***maximal*** value is ***1***.
- The dot product between two vectors is ***zero*** iff (if and only if) they share no components (are ***perpendicular*** / 90° difference / orthogonality)
- The dot product between two vectors is ***minimal*** (negative) when they are ***anti-parallel*** (pointing in opposite directions). If the two vectors both have ***norm 1***, then this minimal value is ***-1***.

Cosine-similarity or cosine-distance

A and B are vectors. Say A represents "cat", B represents "dog". The cosine-distance is a way to calculate how similar two words (vectors) are.

$$cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} = \frac{dot\ product\ of\ the\ vectors}{Length\ of\ the\ vectors\ multiplicated}$$



Calculation of cosine similarity

The **dot product** can be used to multiply two vectors of equal size. The result is a real number.

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} * \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 * b_1 + a_2 * b_2 + a_3 * b_3$$

The **length / magnitude / norm** of a vector is calculated by squaring all the values of the vector, adding them together and then take the square root of the sum. The result is a real number.

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \sqrt{(a_1)^2 + (a_2)^2 + (a_3)^2}$$

Cosine-similarity:

$$similarity(A, B) = \frac{a_1 * b_1 + a_2 * b_2 + a_3 * b_3}{\sqrt{(a_1)^2 + (a_2)^2 + (a_3)^2} * \sqrt{(b_1)^2 + (b_2)^2 + (b_3)^2}}$$

The resulting similarity ranges from -1 meaning ***exactly opposite***, to 1 meaning ***exactly the same*** vector/word, with 0 indicating ***orthogonality (perpendicular)***.

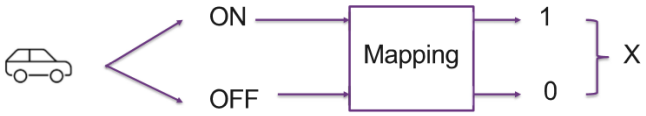
There are two approaches to training a general language model: ***Download and use a predefined language model*** or the ***usage and optimization of an Embedding Layer***. A known architecture for training embeddings is known under the term ***word2vec***.

2.3. LARGE LANGUAGE MODELS (LLMS)

Large Language Models dominate the current success stories. LLMs are ***large Artificial Neural Networks*** that are used for translation, chat, Q&A, programming etc.
Current LLMs are ***Transformer based***. Transformers are computational units with a particular structure and a trainable "Attention mechanism". *Not further covered in AiFo*
Generative Language models produce sequences by ***calculating a probability distribution over the next word given the past text***. They sample one word (or token) and repeat the process.

3. STATISTICS

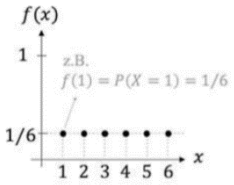
3.1. RANDOM VARIABLES



A random variable X is a variable that ***takes a numerical value*** x which ***depends on a random experiment***. This is a way to represent outcomes of a random experiment as numbers.

- ***Discrete:*** X takes any of a finite set of values, e.g. $\{-8, 1.5, 2.693, 5\}$
- ***Continuous:*** X takes any value of an uncountable range, e.g. the real numbers in the interval $(2; 7)$.

3.1.1. Probability Mass Function (Wahrscheinlichkeitsfunktion)



Function $f(x)$ that provides the probability for each value x of a discrete random variable X. The tables below are PMFs, and the graph at the right-hand side.

3.1.2. Example: Rolling a single dice

The discrete random variable X is the number observed when rolling a fair dice. The possible values and the probabilities they take are:

Value x ***of the random variable*** X	***1***	***2***	***3***	***4***	***5***	***6***
$\Pr(X = x)$ *can also be written as* $\mathrm{P}(\mathrm{x})$, $\mathrm{p}(\mathrm{x})$ *or* $\mathrm{P}_{\mathrm{x}}(\mathrm{x})$ *(Probability that the random variable* X *takes the value* x*)*	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

3.1.3. Example 2: Sum of two dice

The discrete random variable X is the sum of eyes of two dice. *($6^2 = 36$ Options, only 2 dice combinations can result in a 3 (1/2 and 2/1), while for a 9, there are 4 combinations (6/3, 5/4, 4/5 and 3/6)*

Value x	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
$\Pr(X = x)$	$\frac{1}{36}$	$\frac{2}{36}$	$\frac{3}{36}$	$\frac{4}{36}$	$\frac{5}{36}$	$\frac{6}{36}$	$\frac{5}{36}$	$\frac{4}{36}$	$\frac{3}{36}$	$\frac{2}{36}$	$\frac{1}{36}$

Probability that the sum is between 4 and 7 (sum of probabilities):

$$\Pr(4 \leq sum \leq 7) = \Pr(X = 4) + \Pr(X = 5) + \Pr(X = 6) + \Pr(X = 7) = \frac{3}{36} + \frac{4}{36} + \frac{5}{36} + \frac{6}{36} = \frac{18}{36} = 50\%$$

In Example 2, we have two random variables. X is the number of eyes on dice #1 and Y is the number of eyes of dice #2.

3.2. JOINT PROBABILITIES

The joint properties of two random variables are defined by the ***Joint Probabilities Mass Function***.

3.2.1. Joint Probabilities with independent random variables

For independent random variables, the joint probability is the ***product of the individual probabilities***. This is also true with more than two independent random variables.

Example: The ***first*** dice does ***not affect*** the probability of ***the second*** dice, so the probability for die #1 showing 5 and die #2 showing 4 is still $\frac{1}{36}$:

$$\Pr(X,Y) = \Pr(X) * \Pr(Y) \rightarrow \Pr(X=5, Y=4) = \Pr(X=5) * \Pr(Y=4) = \frac{1}{6} * \frac{1}{6} = \frac{1}{36}$$

3.2.2. Joint Probabilities with dependent random variables

If the ***events are not independent***, the variables are dependent or correlated. Example:

- X: The event to observe clouds
  ($0 =$ no clouds, $1 =$ small clouds, $2 =$ big clouds)
- Y: The event to observe that it rains
  ($0 =$ no rain, $1 =$ light rain, $2 =$ moderate rain, $3 =$ heavy rain)

	$X=0$	$\boldsymbol{X=1}$	$X=2$
$Y=0$	0.35	0.21	0.03
$Y=1$	0.10	0.07	0.04
$\boldsymbol{Y=2}$	0.00	**0.05**	0.05
$Y=3$	0.00	0.02	0.08

Given there are ***small clouds***, what is the probability for ***moderate rain***? This value cannot be read directly from the table, because all the probabilities in the full table together are 1, in this case however we only look at Y given that $X = 1$.

3.2.3. Marginal Probability

The probability of an event occurring, ***irrespective of the outcome of another*** random variable. For example, the probability of $Y = 2$ for all outcomes of X. If the two variables are visible in a table, then the marginal probability of one variable Y would be the ***sum of probabilities*** for the other variable X on the margin of the table. This is often used to "normalize" the values across a "row" or "column".

In other words, the probability of Y regardless of X is the sum of all probabilities of X where Y appears.

Written as $\Pr(X) = \sum_Y \Pr(X,Y)$ *or* $\Pr(Y) = \sum_X \Pr(X,Y)$

Example: The probability of rain, regardless of cloud size

$$\Pr(Y=2) = \Pr(X=0, Y=2) + \Pr(X=1, Y=2) + \Pr(X=2, Y=2) = 0.00 + 0.05 + 0.05 = 0.10$$

3.2.4. Conditional Probability (X when given Y)

The probability that something will happen in relation to knowledge we already have about another correlating event. Joint probability $\Pr(X,Y)$ and conditional probability $\Pr(X|Y)$ are related in the following way:

$$\Pr(what\ we\ want\ to\ know \mid what\ we\ know) = \frac{\Pr(what\ we\ want\ to\ know)}{\Pr(what\ we\ know)}$$

$$\Leftrightarrow \Pr(X,Y) = \Pr(X|Y) * P(Y) \Leftrightarrow \Pr(Y|X) = \frac{P(X,Y)}{P(X)}$$

Example: Probability of moderate rain *(what we want to know)* given small clouds *(what we know)*

$$\Pr(\boldsymbol{Y=2} \mid \boldsymbol{X=1}) = \frac{\Pr(\boldsymbol{X=1}, \boldsymbol{Y=2})}{P(\boldsymbol{X=1})} = \frac{\mathbf{0.05}}{\mathbf{0.21+0.07+0.05+0.02}} = \frac{\mathbf{0.05}}{\mathbf{0.35}} \approx 0.14$$

3.3. TWO-STEP EXPERIMENTS

Example: Consider a box with three different coins, a red, a blue and a green one. The red coin is fair. The others have different probabilities for head/tail.

- Red coin R: $\mathrm{P}(head) = 0.5, \mathrm{P}(tail) = 0.5$
- Blue coin B: $\mathrm{P}(head) = 0.7, \mathrm{P}(tail) = 0.3$

- Green coin G: $\mathrm{P}(head) = 0.1, \mathrm{P}(tail) = 0.9$

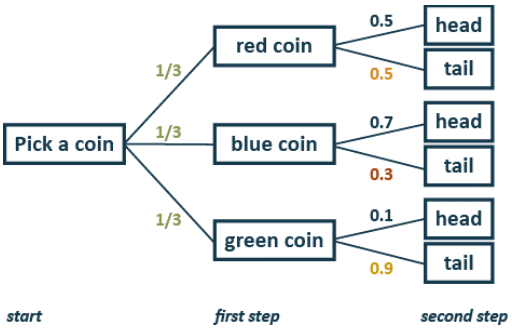
We now do a ***two-step experiment***:

- ***Step 1:*** Pick a random coin from the box.
- ***Step 2:*** toss the coin and observe the outcome.

What is the probability to observe "tail"?

3.3.1. Tree Diagram

Tree diagrams are a probabilistic model that explains how data is generated. They are also a structured visualization of the experiment.



To calculate the probabilities, you just need to multiply along the path.

- $\mathrm{P}(red, tail) = \frac{1}{3} * \mathbf{0.5} = \frac{1}{6} = 0.1\bar{6}$
- $\mathrm{P}(blue, tail) = \frac{1}{3} * \mathbf{0.3} = \frac{3}{30} = 0.1$
- $\mathrm{P}(green, tail) = \frac{1}{3} * \mathbf{0.9} = \frac{9}{30} = 0.3$

The probability of observing tail is therefore: $\frac{1}{6} + \frac{3}{30} + \frac{9}{30} = \frac{17}{30} = 0.5\bar{6}$

To generalize: The first step can be written as $\mathrm{P}(X)$ and the second as $\mathrm{P}(Y|X)$. So, the full calculation can be written as $\mathrm{P}(X, Y) = \mathrm{P}(Y|X) * \mathrm{P}(X)$

The same can be shown in a table of joint probabilities.

	red coin	***blue coin***	***green coin***	*marginal $P(side)$*
head	$\mathrm{Pr}(red, head) = 0.1\bar{6}$	$0.2\bar{3}$	$0.0\bar{3}$	$0.4\bar{3}$
tail	$\mathrm{Pr}(red, tail) = 0.1\bar{6}$	0.1	0.3	$0.5\bar{6}$
marginal $P(coin)$	$0.\bar{3}$	$0.\bar{3}$	$0.\bar{3}$	1

3.4. BAYES RULE

Bayes theorem lets us ***"invert" the direction of the tree model***. From observed outcomes, we can make probabilistic statements about the ***"hidden causes"*** of the observation. The main goal is to ***update the probability of an event based on prior knowledge***. *H = Hypothesis, E = Evidence*

$$\mathrm{Pr}(H|E) = \frac{\mathrm{Pr}(E|H) * \mathrm{Pr}(H)}{\mathrm{Pr}(E)} \quad \Leftrightarrow \quad Posterior = \frac{Likelihood * Prior}{Normalizer\ (all\ of\ the\ likelihoods = marginal\ probability)}$$

Example 1: If we observe tail, which coin was drawn in step 1?
We can formulate this question in terms of probabilities.
Prior: $1/3$, ***Likelihood:*** $\mathbf{0.5}$ *for red,* $\mathbf{0.3}$ *for blue,* $\mathbf{0.9}$ *for green*

- $\mathrm{P}(H = red \mid E = tail) = \frac{P(E=tail|H=red) * P(H=red)}{P(E=tail)} = \frac{\mathbf{0.5} * 1/3}{1/6 + 3/30 + 9/30} = \frac{1/6}{17/30} = \frac{30}{17*6} \approx 0.294$
- $\mathrm{P}(H = blue \mid E = tail) = \frac{P(E=tail|H=blue) * P(H=blue)}{P(E=tail)} = \frac{\mathbf{0.3} * 1/3}{1/6 + 3/30 + 9/30} = \frac{1/10}{17/30} = \frac{30}{17*10} \approx 0.176$
- $\mathrm{P}(H = green \mid E = tail) = \frac{P(E=tail|H=green) * P(H=green)}{P(E=tail)} = \frac{\mathbf{0.9} * 1/3}{1/6 + 3/30 + 9/30} = \frac{3/10}{17/30} = \frac{90}{17*10} \approx 0.529$

The result is a ***posterior distribution***. It is the result of updating the prior distribution with the evidence / data / observation.

Example 2: If we flip the same coin 3 times and observe ***tail, head, head***. Which coin was drawn?

For this example, we need to ***repeatedly apply Bayes rule***. We use the first observation to calculate the first Posterior Distribution. The Posterior then becomes the new Prior in the second application. To calculate the second posterior Distribution (getting tail & head), we also need to adjust the normalizer with the first posterior.

The ***first application*** is the same as in Example 1.

Second application

Prior: 0.294 *for red,* 0.176 *for blue,* 0.529 *for green,* ***Likelihood:*** 0.5 *for red,* 0.7 *for blue,* 0.1 *for green*

- $P(H = red \mid E = head) = \frac{P(E=head|H=red) * 0.294}{P(E=head)} = \frac{0.5 * 0.294}{0.294*0.5 + 0.176*0.7 + 0.529*0.1} = \frac{0.147}{0.323} \approx 0.455$
- $P(H = blue \mid E = head) = \frac{P(E=head|H=blue) * 0.176}{P(E=head)} = \frac{0.7 * 0.176}{0.294*0.5 + 0.176*0.7 + 0.529*0.1} = \frac{0.123}{0.323} \approx 0.381$
- $P(H = green \mid E = head) = \frac{P(E=head|H=green) * 0.529}{P(E=head)} = \frac{0.1 * 0.529}{0.294*0.5 + 0.176*0.7 + 0.529*0.1} = \frac{0.0529}{0.323} \approx 0.163$

Third application

Prior: 0.455 *for red,* 0.381 *for blue,* 0.163 *for green,* ***Likelihood:*** 0.5 *for red,* 0.7 *for blue,* 0.1 *for green*

- $P(H = red \mid E = head) = \frac{P(E=head|H=red) * 0.455}{P(E=head)} = \frac{0.5 * 0.455}{0.455*0.5 + 0.381*0.7 + 0.163*0.1} = \frac{0.228}{0.510} \approx 0.444$
- $P(H = blue \mid E = head) = \frac{P(E=head|H=blue) * 0.381}{P(E=head)} = \frac{0.7 * 0.381}{0.455*0.5 + 0.381*0.7 + 0.163*0.1} = \frac{0.268}{0.510} \approx 0.523$
- $P(H = green \mid E = head) = \frac{P(E=head|H=green) * 0.163}{P(E=head)} = \frac{0.1 * 0.163}{0.455*0.5 + 0.381*0.7 + 0.163*0.1} = \frac{0.016}{0.510} \approx 0.032$

This means, it is most likely that the blue coin was drawn (52%)

4. LINEAR REGRESSION

Linear Models are the ***simplest model*** to explain a ***relationship*** between "Input" and "Output". Standard method to find an optimal linear model.
Interpretation: Understand if some input has an effect on the output.
Example: Is there a relationship between smoking cigarettes and the risk of lung cancer?
Prediction: Given a new x, use the model to predict / estimate the y.
Example: x is smoking rate, y is death rate

Linear regression belongs to ***supervised learning***: The algorithm learns a linear relationship between x and y, both are given.

4.1. MODEL

A model is a ***mathematical function that "explains the data".***
$y_i \approx f(x_i), \quad y_i = f(x_i) + \varepsilon_i$

ε_i ***is "unexplained noise"***. It is assumed that $\varepsilon_i$follows a normal distribution (Bell Curve/Glockenkurve). The function f can be simple or very complicated. The goal of ML is to ***find the model which explains the data*** (as good as possible). Also, instead of approximating y_i, we calculate an estimate $\hat{y}_i$ of the usually unknown y_i.

In linear regression, we ***only consider a linear relationship*** between the input and output. There are therefore only ***two free parameters,*** a ***and*** b. The goal is to identify a and b for which the linear model ***"best explains the data"***. a ***is usually called slope,*** b ***the intercept.***

$\hat{y}_i = ax_i + b$

How to find out if a model is good or bad?

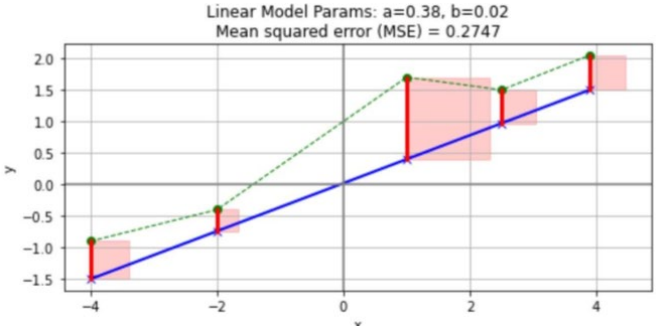
4.2. MEAN SQUARED ERROR (MSE)

This is the loss we want to minimize.

$\hat{y}_i = ax_i + b$

$e_i = y_i - \hat{y}_i$ (e_i = the residual *Difference between our estimate and the actual value*)

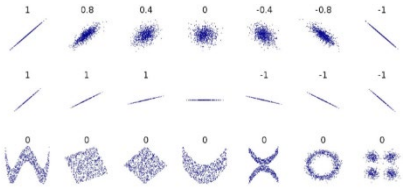
$$\boldsymbol{E = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - (ax_i + b)\right)^2}$$



E (Error) is the Sum of the areas of the residual Squares (red) divided by two times the number of squares.

4.2.1. Pearson Correlation Coefficient (r)

Most common way of measuring a linear correlation. It is a number between -1 and 1 that measures the strength and direction of the relationship between two variables.



r	*Correlation type*	*Interpretation*
$0 < r < 1$	**Positive** Correlation	Both variables change in the **same direction**. *Positive Steigung*
0	**No** correlation	There is **no linear relationship** between the variables.
$-1 < r < 0$	**Negative** Correlation	The 2. variable changes in the **opposite direction.** *Negative Steigung*

4.3. MULTIPLE LINEAR REGRESSION

Add ***more "explaining factors"*** to the model, since most of the time, multiple factors contribute to the result to different degrees: $y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$

Same concept, but different notation / indexing: $Y_i = \alpha + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_n x_i^{(n)}$

Idea: a single "dependent" variable y is ***explained by multiple independent variables*** x. To be able to change the importance of each variable, we also add a ***weight*** w_n/β_n. *Example:* y_i *is an observed/measured quantity. Example: blood pressure.* $x_1 \ldots x_n$ *are "factors" like age, weight, sex, … .* $w_1 \ldots w_n$ *are weights. How much does each factor* x *explain the outcome* y*?*

A variant of multiple linear regression is polynomial linear regression, where each variable is an exponent

$y = w_1x_1 + w_2x_2^2 + w_3^3 + \cdots + w_px_p^d + w_0$

4.3.1. Matrix Notation

Dataset: n points (x, y) where x is a vector with p features (=dimensions)

The model: $\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}$ can be written much more compactly if we use matrix notation:

$y = X\beta + \beta_0$, where X = Datapoints, β = Weights, y = Estimates

$$X = \begin{bmatrix} X_{11} & X_{12} & \dots & X_{1p} \\ X_{21} & X_{22} & \dots & X_{2p} \\ \dots & \dots & \ddots & \vdots \\ X_{n1} & X_{n2} & \dots & X_{np} \end{bmatrix}, \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

5. GRADIENT DESCENT

Gradient Descent is a fundamental ***optimization algorithm***. When an AI is "training" or "learning", this means that an algorithm is performing some sort of optimization, like ***minimizing the loss function***. It only works if we can express the loss function as a differentiable function, this is not always the case.

The ***gradient of a function*** is the ***collection of all its partial derivatives*** organized in a vector.

$$\nabla f(x_0, x_1, x_2, \dots, x_n) =$$

$$\begin{bmatrix} \frac{\partial f}{\partial x_0} = \frac{1}{N}\sum_{j=1}^{N}(y_j - h(w, x_j))(-x_j), w_1 = w_1 - \alpha . dw_1 & \text{derivative of f with respect to } x_0 \\ \frac{\partial f}{\partial x_1}(x_0, x_1, x_2, \dots, x_n) & \text{derivative of f with respect to } x_1 \\ \frac{\partial f}{\partial x_2}(x_0, x_1, x_2, \dots, x_n) & \text{derivative of f with respect to } x_2 \\ \dots & \dots \\ \frac{\partial f}{\partial x_n}(x_0, x_1, x_2, \dots, x_n) & \text{derivative of f with respect to } x_n \end{bmatrix}$$

It will ***always point in the direction where there is the greatest increase*** in the function, in our case the loss. Since we want to minimize our loss, we need to invert our gradient (descenting the gradient). Gradient Descent is an iterative operation, so we run GD again until the result converges (no/very small change). We can also set a convergence threshold, if the last move is smaller than this, we also end the GD.

The gradient descent follows these steps:

1. Pick ***a random point*** w in the function, this is the ***starting point***. This point is represented by the row vector $w = [x_0 \quad x_1 \quad x_2 \quad \dots \quad x_n]$
2. While the gradient hasn't converged (iterative part of the algorithm)
   a. ***Compute the negative gradient at*** w *to all other data points and pick the one with the greatest descent.*
   b. ***Move the location by the result of 2a***.
3. ***Repeat*** until you have found the minimum or reached the ***convergence threshold***. If, compared to the previous iteration, the new gradient of point w has not changed more than the convergence threshold, the algorithm has converged.

$$\underbrace{w^{(t+1)}}_{\text{position of next iteration}} = \underbrace{w^{(t)}}_{\text{position of previous step}} \underbrace{- \alpha \nabla f(w^{(t)})}_{\text{step}}$$

learning rate

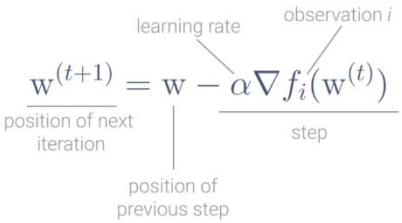
Learning rate α

The learning rate alpha is the ***size of the step*** Gradient Descent takes all the way until it reaches a minimum, and it directly impacts the performance of the algorithm. When it's ***too big***, you're taking big steps, so you may step over the minimum and ***never reach it***. When the learning rate is ***too small***, the algorithm might ***take a long time*** to find the minimum.

Limitations of Gradient Descent

- Calculating derivatives for the entire dataset is ***time consuming***.
- ***Memory*** required is proportional to the size of the dataset.

5.1. STOCHASTIC GRADIENT DESCENT (SGD)



It is a probabilistic approximation of Gradient descent. It is an approximation because, at ***each step***, the algorithm ***calculates the gradient for one data point picked at random***, instead of calculating the gradient for the entire dataset. This represents a significant performance improvement. But because the gradient is not computed for the entire dataset, and only for one random point on each iteration, the updates have a ***higher variance***. This makes the ***cost function fluctuate more*** on each iteration, making it harder for the algorithm to converge.

5.1.1. Batch-Gradient-Descent

Often, ***batch-gradient-descent*** is used which ***uses random subsets*** (or batches) ***instead of one random point***. This is more efficient. Typical batch sizes: 32, 64, …, 1024 samples.

5.1.2. Annealed Stochastic Gradient Descent

The ***learning rate α gets adapted***. Starts the algorithm with a ***large*** learning rate and then ***reduces*** it over time for example by multiplying it at each iteration by a *decay_factor* like 0.99. Typically, there's a lower bound where the learning rate doesn't decrease any further.

Formeln noch einfügen

Look at exercise solutions and recap week 8

Hier fehlt noch was lalala

6. REGULARIZATION

Too complex models generalize badly. Too simple models may miss information and perform sub-optimally. Those two observations are related by the bias-variance trade-off (aka bias-variance dilemma). With regularization, we can constrain the learning process.

6.1. MODEL TESTING

The model must perform well on new, unseen inputs which means it must generalize well to new data.

In-sample Error (aka Training Error): It is possible to find a model which perfectly fits the data. When a model has an MSE of 0, the data was probably overfitted. Overfitted models perform great on the training data, but badly on new data. So ideally, the training error should be minimal, but not zero.
Out-of-Sample Error (aka Test Error): If we use our model on new data sample (x_{unseen}, y_{unseen}) to test how well it predicts for new data, we can calculate the error of the prediction using $\hat{y}_{unseen} = h(w, x_{unseen})$ and y_{unseen}. This is the out-of-sample error.

The goal is to ***learn a model*** from data that ***generalizes well*** to new data. A "good" model has a low generalization error. So ***both errors*** should be ***as low as possible***.

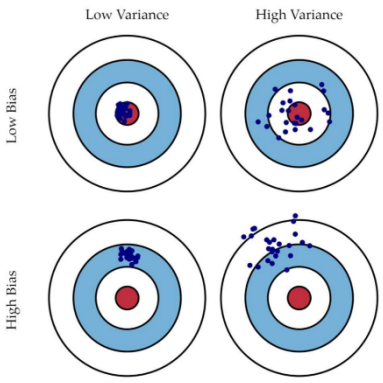
6.1.1. Splitting Technique

We can't calculate the generalization error, because we do not have "new data" to test our model. We can only estimate it by ***splitting the data*** we are given into two sets: The training and test set. A common split ratio is 80% / 20%, so we have most of the data in the training set, while still having enough data to test with. The data in the test-set does not get used during fitting.

- ***Training Phase:*** Fit the model to the training set. This minimizes the in-sample error.

- ***Evaluation:*** Evaluate the model using the test-set. This gives us an estimate of the generalization error.

6.1.2. Bias-Variance Trade-Off



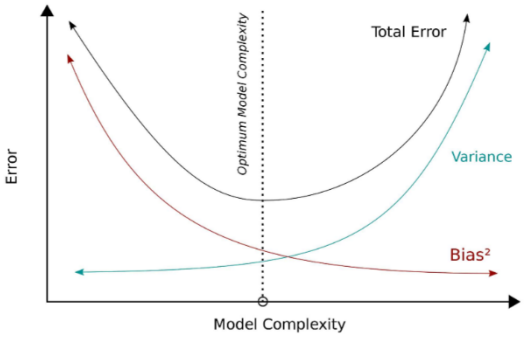
By analyzing the prediction error mathematically, one can decompose it into two terms: bias *(average difference between predicted and actual values)* and variance *(difference between different runs of a model)*. The expression "high bias" is used in the sense of "a too simple model for the given data"

- High Variance: Not Precise *Estimates are spread out*
- Low Variance: Precise *Estimates are clustered together*
- High Bias: Not Accurate, high training error *Model missed relevant relations between features and outputs*
- Low Bias: Accurate *Estimates are close to the correct result*

Too Simple Model: High Bias, Low Variance (underfitting)

A very simple model *(i.e. simple linear regression)* cannot do better than finding a line, no matter how much data we give it to learn. Such a model imposes a ***high bias***. It ***fails to learn the underlying structure*** of the data. By imposing a high bias, we assume a "simple world" and reduce the space of what can be learned.

The flip-side of a too simple model is that it is relatively stable. With another test-sample, the model would find a very similar line. So for a ***change in data***, we would fit ***almost the same model***. This is the meaning of ***low variance***.



Too Complex Model: Low Bias, High Variance (overfitting)

Low bias: A more complex model is ***less restrictive***. It can better "explain" the data. A ***high variance*** means that for a ***different set*** of data points, the ***model*** could be ***very different***.

The trade-off

Higher bias implies lower variance, lower bias implies higher variance. In practice, we do not directly care about the bias, we just want a low variance (reliable predictions). But we can only build a model as complex as the data permits. We therefore have to find an optimal balance between bias and variance.

6.2. REGULARIZATION

With regularization, we reduce the number of polynomial degree to ***avoid overfitting*** or increase the degree to ***avoid underfitting***. It does this by decreasing variance at the cost of increasing bias. This in turn decreases the training accuracy, but increases generalizability. Regularization ***adds a Constraint*** to the model, rather, its Optimizer, to achieve this.

- ***Measure of performance***: regression error (MSE) *how well the model predicts data*
- ***Measure of complexity***: regularization term *control the complexity of the model*

We want to Minimize the regression error + regularization term. It is common to have two separate functions: An optimization function for the optimizer (Gradient Descent) and a performance evaluation function to evaluate the error.

6.2.1. How to express Model Complexity

The complexity of a model can be expressed by multiple parameters: degree of polynomial, number of features and size of coefficients. There are two different ways to express the complexity:

- L2-Norm (Euclidean Norm, Sum of weights): $\sum_{j=1}^{p} w_j^2$
- L1-Norm (Manhattan distance / Taxicab norm, Sum of absolute weights, «Häuschen zählen»): $\sum_{j=1}^{p} |w_j|$

We add one of these constraints to the optimizer to find the best weights for our model

6.2.2. Ridge

Ridge uses the L2-Norm (Euclidean Norm). Minimize:

$$MSE_{ridge}(\boldsymbol{X}, h(w,x)) = \frac{1}{2N}\sum_{j=1}^{N}\left(y_i - \boldsymbol{h}(w, x_j)\right)^2 + \boldsymbol{\lambda} * \sum_{j=1}^{p} w_j^2 = \text{MSE} + \text{Hyperparameter} * \text{L2-Norm}$$

Example Calculation of L2-Norm: Point 1 is at $x = 3, y = 3$ and Point 2 is at $x = 2, y = 2$. So, Lambda gets multiplied by $\sqrt{2}$.

$$d_E(x_1, x_2) = \sqrt{\sum_{i=1,p}(x_{1,i} - x_{2,i})^2} \Rightarrow d_E(1,2) = \sqrt{(3-2)^2 + (3-2)^2} = \sqrt{2}$$

L2 regularization is not robust to outliers. The squared terms will blow up the differences in the error of the outliers. The regularization would then attempt to fix this by penalizing the weights.

6.2.3. Lasso

Lasso uses the L1-Norm (Manhattan distance). Minimize:

$$MSE_{lasso}(\boldsymbol{X}, h(w,x)) = \frac{1}{2N}\sum_{j=1}^{N}\left(y_j - \boldsymbol{h}(w, x_j)\right)^2 + \boldsymbol{\lambda} * \sum_{j=1}^{p} |w_j| = \text{MSE} + \text{Hyperparameter} * \text{L1-Norm}$$

Lasso can force the weights to 0, unlike Ridge. It enables us to perform feature selection, making certain weights 0. $w_i = 0$ means that x_i is not relevant.

When we have highly correlated features *i.e. number of rooms and house area size*, the L1 norm would select only 1 of the features from the group of correlated features in an arbitrary nature, which is something that we might not want.

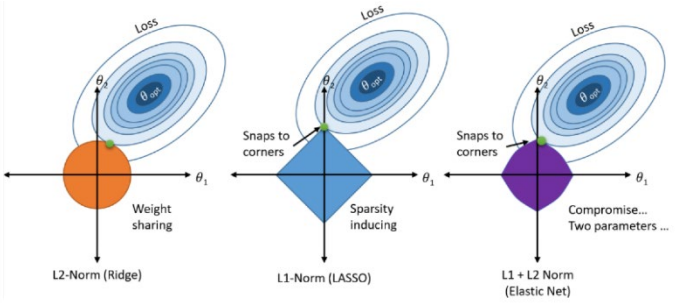
Example Calculation of L1-Norm: Point 1 is at $x = 3, y = 3$ and Point 2 is at $x = 2, y = 2$.

$$d_M(x_1, x_2) = \sum_{i=1}^{p}|x_{1,i} - x_{2,i}| \Rightarrow d_M(1,2) = (3-2) + (3-2) = 2$$

To test if you have too many features, you can use lasso regression to see if it eliminates any features.

Lambda (λ)

λ is a hyperparameter, it does not belong to the optimization process as such. It is varied to find the best fit. ***When it is zero, the*** $\text{MSE}_{\text{ridge/lasso}}$ ***is just the normal MSE***. As λ gets larger, we are enforcing the weights to be smaller by constraining the squared sum of weights more and more. ***Increasing*** λ ***makes the model simpler, increases bias and reduces variance.***



7. CROSS VALIDATION

Hyperparameter: Specifies details of the learning process such as parameters of the optimizer *(learning rate, type of gradient descent, regularization parameters (L1, L2, Lambda) etc.)*

3-way holdout: Split data in ***training-data***, ***validation-data*** and ***test-data***.

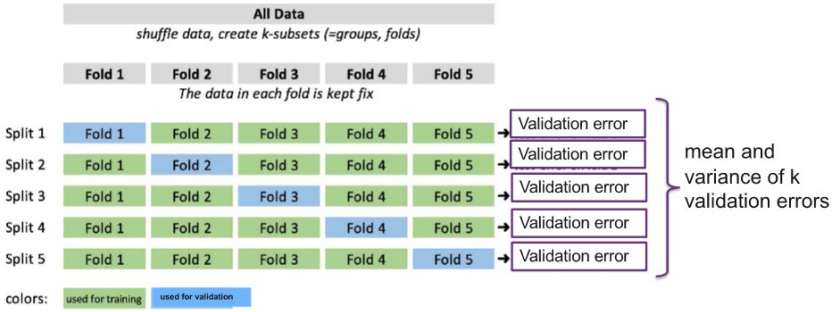
- Train the model with the ***training-data***, multiple models with different hyperparameters
- Validate the trained models with the ***validation-data***. Take the model with the best score.
- Test the best model with the ***test-data***.
- Optional: Train the best model again with all the data.

Problems

- Training error is ***too optimistic*** about generalization.
- Test error is ***unbiased*** but can be ***too pessimistic***. The generalization error is calculated only on certain 20% set.
- Test and training data may ***not be representative*** of the general/overall dataset.

7.1. K-FOLD CROSS VALIDATION

Cross validation is a technique to address these problems. It is an extension of the holdout method. With the k-fold cross-validation, ***the data is split into k folds***. Then the ***train/validation process is repeated k-times***. Each fold participates in k-1 training phases and is used once for validation.



We can use cross-validation to ***obtain a better estimate of the generalization error***. This is also known as model evaluation. ***After*** k-fold cv, we can ***train the model on the complete data*** using the ***fixed hyperparameters*** and deploy that model. If $k = 2$, the Model is split into 50% training data and 50% test data. If $k = n$, only one value is used for testing on every split (LOOCV – Leave one out cross validation). Typical values for k are $5,10$ or N. It is better to ***apply the preprocessing pipe-line*** (e.g. standardization) to ***each split***, ***not only once in the beginning*** for the whole dataset. Otherwise, the results may be distorted.

8. FEATURE SCALING

Feature scaling is a method to ***normalize the range of independent variables*** of data. If for example, you have multiple independent variables like age, salary and height with ranges 18-100 years, 25'000 – 75'000 and 1-2 meters, feature scaling transforms them all to be in the ***same range***. If the range differences are too big, small changes in the weights of large features have a huge impact on the MSE, while weights of small features need huge changes to affect the MSE.

Regularization penalizes larger coefficients more than the smaller ones.
Standardization puts all the features on equal footing.

Raw Data	Normalized Data
2	-1.403
4	-0.468
4	-0.468
4	-0.468
5	0.000
5	0.000
7	0.935
9	1.871

8.1. SKLEARN STANDARDSCALER

Rescales a dataset to have a mean of 0 and a standard deviation of 1.

$$x_{std} = \frac{x - X_{mean}}{s},\ s = \sqrt{\frac{1}{N+1}\sum_{i=+}^{N}(x_i - X_{mean})^2}$$

Example: Data Points x_i: $2,4,4,4,5,5,7,9$. $X_{mean} = 40/8 = 5$. Sample variance = 4.57, Sample Std. Deviations $= 2.138$. Standardization of $2 = \frac{2-5}{2.138} = -1.4$

9. CLASSIFICATION AND LOGISTIC REGRESSION

- ***Binary Classification:*** Only two classes. Example: Epileptic seizure or healthy state?
- ***Multi-Class Classification:*** More than two classes. Example: Match is won, it's a tie, Match is lost

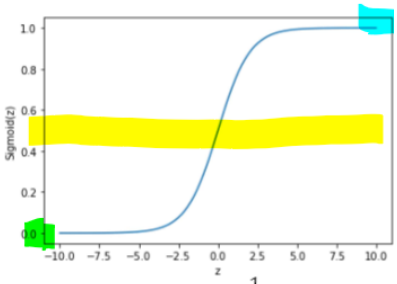
9.1. LOGISTIC REGRESSION

Used for binary classification (Yes/No, Spam/no spam). Linear regression is not usable for binary classification, because it is linear.

Why not linear regression for binary classification? Because even with a threshold, the function does not work well with only two outputs. The MSE does not work. So we need a probabilistic function like the sigmoid.

9.1.1. Sigmoid function

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



$sigmoid(0) = \frac{1}{1+e^{-0}} = 0.5$

$sigmoid(\infty) = \frac{1}{1+e^{-\infty}} = 1$

$sigmoid(-\infty) = \frac{1}{1+e^{\infty}} = 0$

$z = h(w, x) = \boldsymbol{w_1}x_1 + \boldsymbol{w_2}x_2 + \boldsymbol{w_3}x_3 + \cdots$

This is where our features (data x) enters the calculation. The ***w***'s are unknown. That's what needs to ***be learned***.

Why sigmoid? Because of the odds ratio: $odds(p) = \frac{p}{1-p} = \frac{\Pr(y=1)}{\Pr(y=o)}$

We can write the estimated probability as

$\Pr(y = 1|x; W) = \Pr(y = 1|x) = p(x) = \frac{1}{1+e^{-(W^T x)}} = \frac{1}{1+e^{-(w_1x_1+w_2x_2+w_3x_3+w_4)}}$

9.1.2. Maximum Likelihood

Given all the data points (X, Y), we want to maximize the probability that all the predictions are correct. The objective of training is to set the coefficients W so that p is close to 1 when $y = 1$ and close to 0 when $y = 0$. This can be calculated using gradient descent.

$Minimize\ cost(W) = -\frac{1}{N}\sum_{i=1}^{N}(y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i))$

10. CLASSIFIER EVALUATION

How to calculate accuracy and error from the confusion matrix?

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP),	False negative (FN),
	Negative (N)	False positive (FP),	True negative (TN),

- ***Accuracy:*** How often is the classifier correct: $\frac{TP+TN}{n}$
- ***Error:*** How often is the classifier wrong: $\frac{FP+FN}{n}$

It ***depends on the data*** if a False Positive or a False Negative is worse. *(sickness: false negative, Spam: false positive)*

10.1. RECALL

Useful, when false negatives are worse. Among the positive ground truth samples, how many did we correctly classify? *If you have no false negatives because you have no negatives, you can fool recall.*

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

10.2. PRECISION

Useful, when false positives are worse.

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

10.3. F-SCORE

Combining precision and recall. In real life, false negatives and false positives are bad. So we need the harmonic mean of Precision P and Recall R. If ***both*** P and R are ***high***, the F_1 score is ***high***. If ***one of them*** is ***low***, the F_1 score is ***also low***.

$F_1 = \frac{2PR}{P+R} = \frac{2}{\frac{1}{P}+\frac{1}{R}}$

There is also a combined metric $F_\beta = \frac{(1+\beta^2)*P*R}{\beta^2*(P+R)}$. β acts as a dial to decide the emphasis between precision and recall. $\beta = 1, F_\beta = F_1$ *(Both equally important)*, $\beta = 0, F_\beta = P$ *(Recall not important)*, $\beta = infinity, F_\beta = R$ *(Precision not important)*.

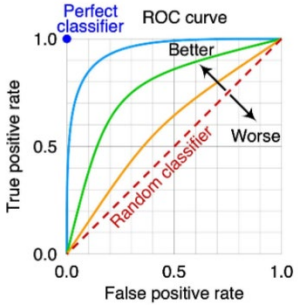
10.4. THRESHOLD

There is ***no universal solution***, different goals require different thresholds.

1. Train your machine learning model
2. Use the trained model to make predictions on your test set, so that each example has a classification probability between 0 and 1.
3. Using a variety of threshold values, ***convert the predicted probabilities to predicted classes***. Calculate True positive rate and False positive rate. ***Different thresholds result in different TPR and FPR***.
4. ***Plot a curve*** of TPR vs FPR for the different thresholds.

10.4.1. Receiver Operating Characteristics (ROC)

A ROC space is defined by FPR and TPR as x and y axes, respectively, which depicts ***relative trade-offs between true positive and false positive***.

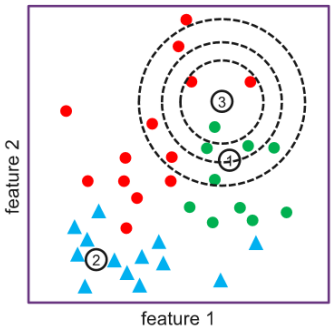


Area under the curve (AUC) shows ***how well*** the TPR and FPR is looking in the aggregate. The ***greater*** the ***area*** under the curve, the ***greater*** the ***quality*** of the model.

11. K-NEAREST-NEIGHBOURS KNN

If a simple line perfectly separates the classes, then the classes are said to be linearly separable. However, what to do when the classes are not linearly separable? Logistic regression is possible, but very inconvenient. That's where KNN comes into play. Basic idea: "A datapoint is known by the company it keeps".

Given a test data point, KNN computes k nearest neighbours of it and returns the most frequent class of the k neighbours.



Example in image:

	$k = 3$	$k = 5$	$k = 10$
Sample 1	*green*	*green*	*green*
Sample 2	*blue*	*blue*	*blue*
Sample 3	*red*	*green*	*red* **or** *green*

11.1. KNN DETAILS

- Load the training and test data
- ***Chose value of*** k (the number of nearest neighbours to consider for classification. $k = 1$ *results in overfitting,* $k = n$ *results in underfitting*
- ***Chose a distance metric*** *(Euclidean, Manhattan, cosine, Minkowsky, …)*
- For each test data points x_{test}:
  - For all training data x_{train}, ***calculate the distance*** $d(x_{test}, x_{train})$ with your distance metric
  - ***Sort*** training data in the ascending order of the distance

- ***Choose the first*** k data points from the sorted training data
- ***Choos the most frequently occurring class*** from the k data points as the classification result.

Advantages of KNN: Easy and simple machine learning model. Few hyperparameters to tune.
Disadvantages: k should be wisely selected, Large computation cost during runtime if dataset is large. Not efficient for high dimensional datasets, proper scaling should be provided.

11.2. DISTANCE METRIC

Given $x_1 = (x_{1,1}, x_{2,1}, \dots, x_{p,1})$ and $x_2 = (x_{1,2}, x_{2,2}, \dots, x_{p,2})$. ***Example:*** $x_1 = (1,1)$, $x_2 = (2,2)$, $p = 2$

Cosine distance

$cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{dot\ product\ of\ the\ vectors}{Length\ of\ the\ vectors\ multiplicated}$, $\frac{1*2+1*2}{\sqrt{1^2+1^2}*\sqrt{2^2+2^2}} = \frac{4}{\sqrt{2}*\sqrt{8}} = \frac{4}{\sqrt{16}} = \mathbf{1}$

Manhattan distance

$d_{MH}(x_1, x_2) = \sum_{j=1}^{p} |x_{i,1} - x_{i,2}|$, $(|1-2|) + (|1-2|) = 1 + 1 = \mathbf{2}$

Euclidean distance

$d_E(x_1, x_2) = \sqrt{\sum_{j=1}^{i=p} (x_{i,1} - x_{i,2})^2}$, $\sqrt{(1-2)^2 + (1-2)^2} = \sqrt{\mathbf{2}}$

Minkowsky distance:

$d_{MK}(x_1, x_2) = \left(\sum_{i=1}^{p} \left(|x_{i,1} - x_{i,2}|^p\right)\right)^{\frac{1}{p}}$, $(|1-2|^2 + |1-2|^2)^{\frac{1}{2}} = 2^{\frac{1}{2}} = \sqrt{\mathbf{2}}$

12. NAÏVE BAYES CLASSIFIER

Naïve Bayes is a generative method for classification *(it generates something)* based on Bayes' Theorem **(See page xy)**. It assumes that ***all the features that predict the target value are independent***. It describes the ***probability of an event based on a prior knowledge*** of conditions.

Naïve Bayes is good when the dataset is small and there is no training phase. It is used extensively when data contains categorical features but not much used in numerical features.

$$\Pr(y|X) = \frac{\Pr(X|y) * \Pr(y)}{\Pr(X)} = \frac{P(x_1|y)*P(x_2|y)*\dots*P(x_3|y)*P(y)}{P(x_1)*P(x_2)*\dots*P(x_n)}$$

Assume we have a bunch of emails we want to classify as spam or not spam. How to calculate $\Pr(spam|\text{"Hurry"})$?

nr.	*email header*	*spam*
1	***Hurry*** Sale Tomorrow	1
2	Rain tomorrow	0
3	Sale price tomorrow	1
4	Tomorrow workshop rain	0
5	***Hurry*** sale	**?**

We can simply take each word as a separate feature.
All the words: ***Hurry***, ***Sale***, ***Tomorrow***, ***Rain***, ***Price***, ***Workshop***

So "Hurry Sale Tomorrow" can be encoded as:
$x_{hurry} = 1, x_{sale} = 1, x_{tomorrow} = 1, x_{rain} = 0, x_{price} = 0, x_{workshop} = 0$

Now when an email contains "hurry", would it be classified as spam or ham? We can find out by calculating the probability that the email is spam given that it contains "hurry".

$$\Pr(spam|\, x_{hurry} = 1) = \frac{\Pr(x_{hurry}=1\,|y=1)*\Pr\,(y=1)}{\Pr\,(x_{hurry}=1)}$$ *(y = 1 means spam)*

- $\Pr(y = 1) = \Pr(spam) = \frac{\#entries\ in\ the\ data\ set\ that\ are\ spam}{\#size\ of\ data\ set} = \frac{2}{4} = \frac{1}{2}$

- $\Pr(x_{hurry} = 1) = \Pr(\text{"hurry"}) = \frac{\#entries\ that\ contain\ "hurry"}{\#size\ of\ data\ set} = \frac{1}{4}$

- $\Pr(x_{hurry} = 1 \mid y = 1) = \Pr(\text{"hurry"} \mid spam) = \frac{\#occurences\ that\ are\ spam\ and\ contain\ "hurry"}{\#occurences\ that\ are\ spam} = \frac{1}{2}$

Final calculation: $\Pr(spam \mid x_{hurry} = 1) = \frac{\frac{1}{2} * \frac{1}{2}}{\frac{1}{4}} = \frac{\frac{1}{4}}{\frac{1}{4}} = 1 \Rightarrow$ email 5, "hurry sale" will be classified as ***spam***.

13. UNSUPERVISED LEARNING - CLUSTERING

When we are given ***data without labels***, can we still learn something from the data? ***Yes***. Often, the data has some structure. The goal of ***unsupervised learning*** is to ***self-discover patterns*** from the data.

A simple example of a ***structure in the data*** are ***clusters***. i.e., the data points which have some shared properties will fall into one cluster.

13.1. CLUSTERING

The goal of clustering is to group n data points into k_c number of clusters. How do we do that?

13.1.1. Naïve K-means

1. Let us assume we ***know the number of clusters*** k_c.
2. ***Initialize*** the value of k ***cluster centres*** (aka means, centroids) $C_1, C_2, \dots, C_{k_c}$.
3. Find the ***squared Euclidean distance*** between the ***centres*** and ***all the data points***. ***Assign*** each data point ***to the cluster*** of the nearest center.
4. Each cluster now potentially has a ***new centre*** (mean). ***Update the centre*** for each cluster. The new center is the average of all the data points in the cluster.
5. If some ***stopping criterion*** met, done *(like centres do not change anymore, the distance of datapoints to the centre is bigger than a set threshold or a fixed number of iterations has been reached)*. Else, ***go to step 3***.

13.1.2. Cluster quality

The number of clusters is a ***hyperparameter***. You need at least two clusters, but less than the amount of data points. How can one evaluate the ***cluster quality***?

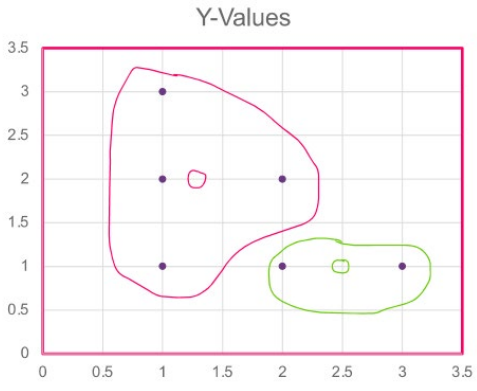
Goal of good clustering: Make clusters so that for each cluster the distance of each cluster member from its centre is mimimized. There are two approaches to find the optimum.

Inertia or within-cluster sum-of-squares (WCSS)

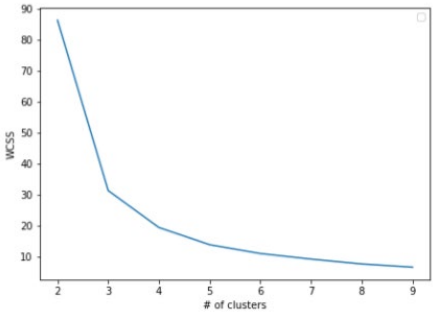
Sum of squared distances of samples to their closest cluster centre *(How far away the points within a cluster are)*. A ***small inertia is desired***.

Example:

	Squared Euclidean distance from red centre	***Squared Euclidean distance from green centre***
	1.25, 2	**2.5, 1**
1,3	$(1.25-1)^2 + (3-2)^2 = 1.0625$	-
1,2	0.0625	-
1,1	1.0625	-
2,1	-	0.25
2,2	0.5625	-
3,1	-	0.25
WCSS	**2.75**	**0.5**



Do this ***calculation*** as you ***increase*** the amount of clusters. Then draw a ***plot*** k_c vs inertia.



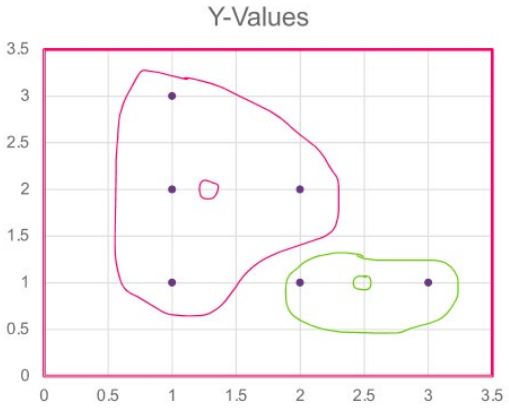
The optimal amount of clusters is found at the "***elbow***" of the graph.

Silhouette Score

The silhouette score considers both the ***cohesion*** a (how similar data points are within the same cluster) and ***separation*** b (how different data points are in different clusters). It provides a value between -1 ***and*** 1 for each data point, with ***higher values*** indicating ***better-defined clusters***. The overall Silhouette Score for a clustering solution is the ***average*** of these individual scores.

$(\boldsymbol{b}-\boldsymbol{a})/\mathbf{max}\,(\boldsymbol{a},\boldsymbol{b})$

Example: calculation of a ***value***



Distance from other points in the cluster

	1,3	1,2	1,1	2,2	a
1,3	–	1	2	$\sqrt{2}$	1.47
1,2	1	–	1	1	1
1,1	2	1	–	$\sqrt{2}$	1.47
2,2	$\sqrt{2}$	1	$\sqrt{2}$	–	1.27

Distance from other points in the cluster

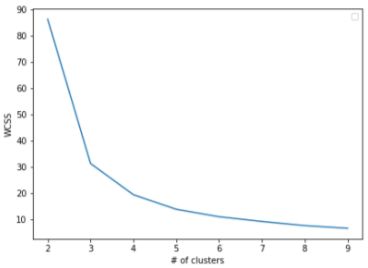
	2,1	3,1	a
2,1	–	1	1
3,1	1	–	1

Calculation of b ***value***

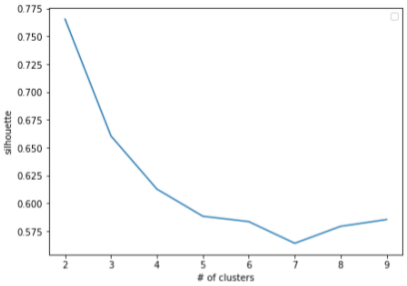
	Distance from points in the RED cluster				
	1,3	1,2	1,1	2,2	b
2,1	$\sqrt{5}$	$\sqrt{2}$	1	1	1.41
3,1	$\sqrt{8}$	$\sqrt{5}$	2	$\sqrt{2}$	7.42

So the Silhouette Score of $(2,1) = (\mathrm{b}-\mathrm{a})/\max(\mathrm{a},\mathrm{b}) = (1.41-1)/1.41 = 0.29$.

We can now decide on an amount of clusters in which the WCSS and the Silhouette Score look good, in this example around 3-5.



WCSS, Elbow Method



Silhouette Score

13.1.3. Performance

The ***performance depends*** on the random ***initialization*** of the seeds for the centroids. Some seeds result in a ***poor convergence rate***, some can converge to ***suboptimal clustering***. If the initial centers are ***very close*** together, it would take ***a lot of iterations*** for the algorithm to converge. The best way is to ***initialize randomly*** and ***run multiple times***. If the clusters are ***stable***, the clustering is ***optimal***.

Features with large values may dominate the distance value. Features with small values will have no impact in the clustering. That's why you should ***always employ feature scaling*** (normalize values).

13.1.4. Example calculating cluster centre

$$\frac{Sum\ of\ x\ coefficients}{amount\ of\ data\ points\ in\ cluster\ (without\ \ centre)}, \frac{sum\ of\ y\ coefficients}{amount\ of\ data\ points\ in\ cluster\ (wihout\ centre)}$$

- ***Center of red cluster:*** $C_{R_x} = \frac{1+1+0.4}{3} = 0.8,\ C_{R_y} = \frac{3+2+2}{3} = 2.33$
- ***Center of green cluster:*** $C_{G_x} = \frac{2+2+2.5}{3} = 2.17,\ C_{G_y} = \frac{2+1+1}{3} = 1.33$

Will a new point $[0.5,1]$ be assigned to the red or the green cluster?
Squared Euclidean distance from red cluster:
$(0.5 - 0.8)^2 + (1 - 2.33)^2 = 1.85$

Squared Euclidean distance from green cluster:
$\mathbf{(0.5 - 2.17)^2 + (1 - 1.33)^2 = 2.99}$

Now that a new point is added to the red cluster, the center needs to be recalculated.

14. ENSEMBLE

The ***combining of multiple weak models*** and the aggregation of their results is called ensemble learning. Aggregating results of many weak predictors for a better prediction. Techniques: Voting, Bagging, Boosting.

Ensemble works best, when:
- The weak models are ***better than random***.
- The models are ***independent from one another*** and make uncorrelated errors.
- There is a ***sufficient number*** of weak learners.
- The models are ***not trained on the same data***.

Different learners use different Algorithms *(KNN, Logistic Regression)*, Different Hyperparameters and different training data.

14.1. HARD VOTING

There are 5 classifiers to check if an email is spam or ham. For a particular data, the prediction of the classifiers are [spam, spam, ham, ham, spam]. The final prediction of the ensemble is spam, ***because 3 of the 5 models voted for spam***.

14.1.1. Hard voting with weights

There are 3 classifiers to predict class spam (1) and ham (0). The predictions from these classifiers have ***weights defined as [0.1,0.3, 0.6]***. For one email, the ***predictions are [spam, spam, ham]***. For spam, we calculate the sum of weights from all classes: $\boldsymbol{sum_{spam}} = w_1 * (prediction_1 == spam) + w_2 * (prediction_2 == spam) + w_3 * (prediction_3 == spam) = 0.1 * 1 + 0.3 * 1 + 0.6 * 0 = \mathbf{0.4}$. For ham, we calculate the same but with ***ham***: $0.1 * 0 + 0.3 * 0 + 0.6 * 1 = \mathbf{0.6}$. The final prediction of the ensemble is ham, because the ***weighted sum of ham was bigger than the sum of spam***.

14.2. SOFT VOTING

Predict the class with the ***highest class probability***, averaged over all classifiers. Only possible if predictions are probabilities. ***Example:*** There are 3 classifiers "C". For a prediction, the classifiers return the following probabilities for each class. $C_1 = [0.85,0.05,0.1], C_2 = [0.15, 0.15, 0.7], C_3 = [0.1, 0.08, 0.82]$. The average for each class ("K") is the following: $K_1 = (0.85 + 0.15 + 0.1)/3 = 0.37, K_2 = 0.09, K_3 = \mathbf{0.54}$. Class 3 has the highest class probability and wins.

14.2.1. Soft voting with weights

There are 3 classifiers and a 3-class classification problem where we assign equal weights to all classifiers. The weighted average probabilities for a sample would then be calculated as follows:

Classifier	*Class 1*	*Class 2*	*Class 3*
Classifier 1	$w_1 * 0.2$	$\boldsymbol{w_1 * 0.5}$	$w_1 * 0.3$
Classifier 2	$w_2 * 0.6$	$\boldsymbol{w_2 * 0.3}$	$w_2 * 0.1$
Classifier 3	$w_3 * 0.3$	$\boldsymbol{w_3 * 0.4}$	$w_3 * 0.3$

In this example, the predicted class label is 2 since it has the highest average probability.

14.3. BAGGING

Bagging methods form a class of algorithms which build ***several instances of a black-box estimator*** on ***random subsets*** of the original training set and then ***aggregate their individual predictions*** to form a final prediction. There are two ways of bagging: ***Sampling with replacement*** *(Putting the cookie back in the bowl after taking it out)* is called ***Bagging*** *(Bootstrap aggregating)*, ***sampling without replacement*** *(eating the cookie after taking it out of the bowl)* is called ***pasting***.

Only bagging allows ***data points to be used several times*** for the same predictor. The individual models have a relatively ***low bias and high variance***. Bagging *(reuse of data)* ***reduces the variance***. This provides a way to reduce overfitting. Bagging works best with strong and complex models.

Random Subspaces: Samples are drawn as random subsets of the features.
Random Patches: Samples are drawn as random subsets of both samples and features.

14.3.1. Out of Bag (oob) Evaluation

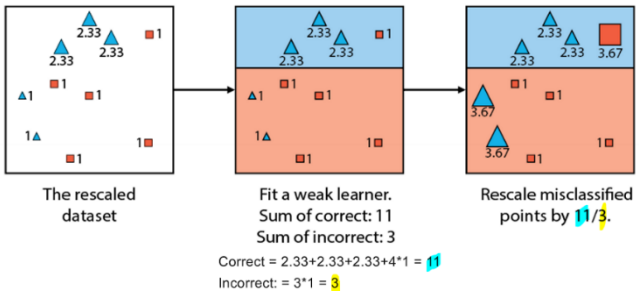
If the data points are random, it is possible that some data points never get chosen. Those points are called oob-points and can be used as test-data.

14.4. BOOSTING

Boosting is an ensemble method to train predictors ***sequentially***. Each ***predictor tries to correct its predecessor***. It tries to ***reduce the bias*** of the combined estimator – the training error reduces.

14.4.1. AdaBoost (Adaptive Boosting)

AdaBoost assigns ***equal weights*** to each training sample. Then it trains a ***model to fit the given data***. After that, it ***increases the weight of the misclassified samples*** so they will make up a larger part of the next classifier training set, so the next classifier will perform better on them.



14.5. NO FREE LUNCH THEOREM

"No single machine learning algorithm is universally the best-performing algorithm for all problems". All models are only as good as the assumptions that they were created with and the data that was used to train them. To find a good model for a problem, you may have to try different models and compare them using a robust cross-validation strategy.