AI Foundations

1 General

Artificial Intelligence is a broad concept with different interpretations, module is focused on algorithms and applications where the computer learns from data, which is called **Statistical Machine Learning (ML)**

Artificial General Intelligence (AGI): hypothetic computer program that can perform intellectual tasks as well as or better than humans, it is the ultimate goal of research

Computer: Input → Processing → Output, processing is non-trivial, different possible In- and Outputs

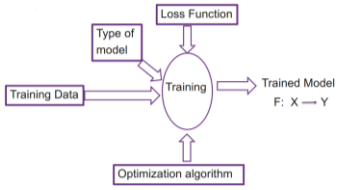
Turing Test: Human asks a question and receives an answer from a human and a computer. If he can't find out, which one was from the computer, the machine is intelligent.

Today's AI: **Artificial neural networks** (data structure to express complex non-linear functions) with artificial neurons and deep learning, data-driven optimization

4 ingredients of ML: Data, Cost-Function (Loss), Model, optimization procedure, **optional ingredients for success**: performance optimization, visualization and evaluation of the learning process, cross-validation & regularization

Large Language Models (LLMs): large transformer-based artificial neural networks which are used for translation, chat, Q&A, programming, …

Supervised Machine Learning: (**Regression**: Linear regression, decision trees, boosting regressor, bagging regressor, **Classification**: Logistic regression, K nearest neighbors, decision trees)

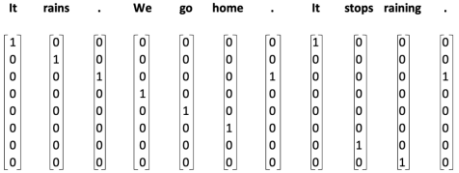


Machine Learning is not just about models, it's also about: Data Engineering, Feature Engineering, Optimization methods, Validation methods, Parameter tuning, Model fitting & evaluation

2 Natural Language Processing (NLP)

automated processing of human language (written and/or spoken), aims to understand and generate human language

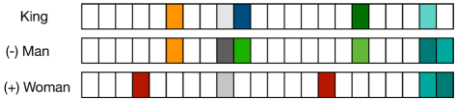
1. One-Hot representation vectors: Count the number of words and define one unique vector per word with one 1



Disadvantages: very **high dimensional** when a lot of words are used, **sparse representation** (many zeroes, memory-inefficient), **no generalization** (all words completely unrelated, because the vectors share no entries)

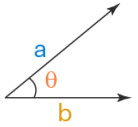
2. Indexing: make a list of words (optionally alphabetically sorted), assign index to each word, represents words as an array of indexes

3. Distributed Representation: a word can be defined by context, words with similar meanings occur in similar context, similar words share similar representations, needs a lot of data, time and CPU/GPU, predefined language model can be downloaded, known architecture: word2vec



Word to Vector (word-embedding): mathematical function maps word to vector, this function is implemented in a so-called **Embedding Layer** in neural networks, nearby words have semantic similarity, we can make calculations if we have "good" vectors

dot-product is a measure of similarity, computer does not understand the meaning of a word, but it can make calculations.



a . b = |a||b| cos θ

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

The bigger $cos(\theta)$, the bigger is the **cosine similarity**. **Max=1**= exact same direction, **0**= orthogonal (90°), **Min=-1**=opposite direction

3 Dialogflow

2 architectures: business logic in own application, use Dialogflow to receive matched entities and react to them in own application or use other services **OR** implement business logic in the cloud, use Dialogflow fulfillments to call other services

Intent: detect what the client wants, many different phrases can express the same intent (e.g. order food, want to eat)

Follow-up intent: intent that only makes sense after another (e.g. order drink in addition to food)

Entities: to fulfill an intent, application needs to know some parameters (e.g. what food?), entities can be required or not, they can have a custom name and there are system entities, for example to match numbers, dates, geographical data, color, etc

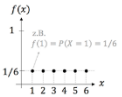
4 Random Variables and Random Experiments

Random Variables come in two flavours: **discrete**: X takes any of a finite set of values, e.g. {-8, 1.5, 2.693, 10}, **continuous**: X takes any value of an uncountable range, e.g. the real numbers in the interval (2, 7)

Random vars represent **outcomes of random experiments** as numbers and are denoted with an **uppercase letter** (often X), the **actual outcome** is denoted with a **lowercase letter** (often x)

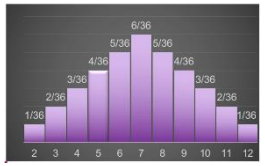
Probability Mass Function (PMF) of a dice:

Value x of the random Variable X	1	2	3	4	5	6
Pr(X=x)	1/6	1/6	1/6	1/6	1/6	1/6



Sum of two dice:

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12



4.1 Joint Probability

Independent random variables, if you throw a dice twice, the second number does not depend on the first number.

$\Pr(X, Y) = \Pr(X) \cdot \Pr(Y)$

Example: first dice is a 5, second dice is a 4

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

4.2 Conditional Probability

Dependent random variables, the probability of rain depends on the clouds observed, below is a joint probability table

X: Event to observe clouds (0=no clouds, 1= small clouds, 2= big clouds)

Y: Event that it rains(0=no rain, 1=light rain, 2=moderate rain, 3=heavy rain)

P(X,Y)	***X=0***	***X=1***	***X=2***	
Y=0	0.35	0.21	0.03	Pr(Y=0)= 0.59
Y=1	0.10	0.07	0.04	Pr(Y=1)= 0.21
Y=2	0.00	0.05	0.05	Pr(Y=2)= 0.10
Y=3	0.00	0.02	0.08	Pr(Y=3)= 0.10
	Pr(X=0)= 0.45	Pr(X=1)= 0.35	Pr(X=2)= 0.20	1

Probabilities of Y given X: $\Pr(Y|X) = \frac{\Pr(X,Y)}{\Pr(X)}$

Example: Probability that you observe moderate rain when there are small clouds

$\Pr(Y = 2|X = 1) = \frac{\Pr(X = 1, Y = 2)}{\Pr(X = 1)} = \frac{0.05}{0.35} = 0.14 \ldots$

4.3 Marginal Probability

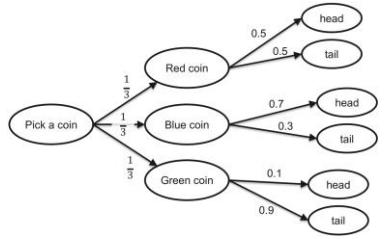
The marginal probability can be read in the last row and column at the right above.

$$Pr(X) = \sum_Y Pr(X, Y) \text{ or } Pr(Y) = \sum_X Pr(X, Y)$$

4.4 Two-Step Experiment

Example: We have a box with a red, blue & green coin.
Red: $Pr(S = head) = 0.5, Pr(S = tail) = 0.5$
Blue: $Pr(S = head) = 0.7, Pr(S = tail) = 0.3$
Green: $Pr(S = head) = 0.1, Pr(S = tail) = 0.9$

Step 1: Randomly pick a coin from the box
Step 2: Flip the coin and observe outcome, head or tail



$\mathbf{Pr}(\boldsymbol{X}) = \Pr(red) = \Pr(blue) = \Pr(green) = \frac{1}{3}$

$\mathbf{Pr}(\mathbf{Y}|\mathbf{X}) = \Pr(head|blue) = 0.7$

$\mathbf{Pr}(\boldsymbol{X}, \boldsymbol{Y}) = \mathbf{Pr}(\boldsymbol{Y}|\boldsymbol{X}) \cdot \mathbf{Pr}(\boldsymbol{X}) = \Pr(blue, head) = \Pr(head|blue) \cdot \Pr(blue) = 0.7 \cdot \frac{1}{3} = 0.2\overline{3}$

Joint Probability Table

	Coin=red	Coin=blue	Coin=green	*Marginal Pr(Side)*
Side=head	Pr(red, head) ≈ 0.1666	0.2333	0.0333	*0.433*
Side=tail	Pr(red, tail) ≈ 0.1666	0.1	0.3	*0.566*
Marginal Pr(Coin)	*0.333*	*0.333*	*0.333*	1

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \qquad \text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Normalizer}}$$

4.5 Bayes Rule

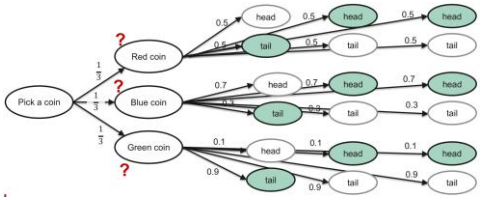
Example: We observe tail and want to calculate the probabilities of which coin was used.

$$\Pr(red|tail) = \frac{\Pr(tail|red)\Pr(red)}{\Pr(tail)} = \frac{0.5 \cdot 0.\overline{3}}{0.5\overline{6}} \approx 0.294$$

$$\Pr(blue|tail) = \frac{\Pr(tail|blue)\Pr(blue)}{\Pr(tail)} = \frac{0.3 \cdot 0.\overline{3}}{0.5\overline{6}} \approx 0.176$$

$$\Pr(green|tail) = \frac{\Pr(tail|green)\Pr(green)}{\Pr(tail)} = \frac{0.9 \cdot 0.\overline{3}}{0.5\overline{6}} \approx 0.529$$

The three results above are called the **posterior distribution**. It is the result of updating the **prior distribution** with the evidence. It is possible that we want to calculate the posterior distribution of multiple sequential outcomes. Then the posterior distribution from before becomes the new prior distribution.



new $\Pr(head) = 0.294 \cdot 0.5 + 0.176 \cdot 0.7 + 0.529 \cdot 0.1 = 0.323$

$$\Pr(red|head) = \frac{\Pr(head|red)\Pr(red)}{\Pr(head)} = \frac{0.5 \cdot 0.294}{0.323} \approx 0.454$$

$$\Pr(blue|head) = \frac{\Pr(head|blue)\Pr(blue)}{\Pr(head)} = \frac{0.7 \cdot 0.176}{0.323} \approx 0.381$$

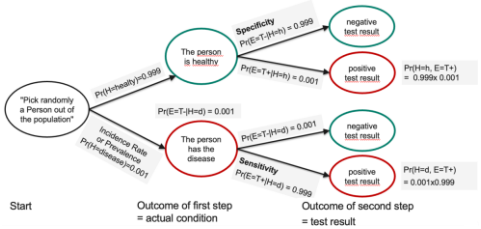
$$\Pr(green|head) = \frac{\Pr(head|gren)\Pr(green)}{\Pr(head)} = \frac{0.1 \cdot 0.529}{0.323} \approx 0.163$$

The same 4 calculations would have to be done for the third head outcome.

Another example:

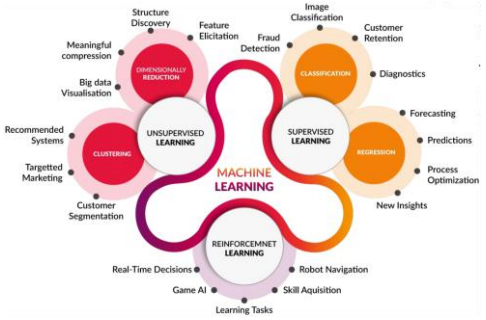
Given: AIDS Test has sensitivity=0.999, specificity=0.999, incidence rate=0.001

Asked: Person is tested positively, what is the probability that the person has AIDS?



$$\Pr(H = disease|E = pos) = \frac{\Pr(E = pos|H = disease) \times \Pr(H = disease)}{\Pr(E = pos)}$$

$$= \frac{0.999 \times 0.001}{0.999 \times 0.001 + 0.001 \times 0.999} = \frac{1}{2}$$



5 Linear/Polynomial Regression

Linear Models are the simplest models to explain a relationship between input and output. Linear regression is a standard method to find an optimal linear model.

In ML, we use the term **model** for any mathematical function that "explains the data". ε_i is the unexplained noise

$$y_i \approx f(x_i)$$
$$y_i = f(x_i) + \varepsilon_i$$

Instead of approximating y_i, we calculate an estimate $\hat{y}_i$ of the usually unknown y_i.

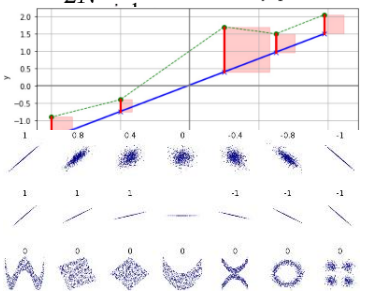
$$\hat{y}_i = f(x_i)$$

A **linear model** looks like this, where a is the **slope** and b the **intercept**.

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

In ML, the **loss** is what we want to minimize. An example of a loss function is the **mean squared error (MSE)**. The difference e_i is called the **residual**.

$$\hat{y}_i = a \cdot x_i + b$$
$$e_i = y_i - \hat{y}_i$$
$$E = \frac{1}{2N}\sum_{i}^{N} e_i^2 \quad = \frac{1}{2N}\sum_{i=1}^{N}(y_i - (a \cdot x_i + b))^2$$



Pearson correlation coefficient

Data can still be highly structured, despite coefficient=0

5.1 Multiple/Polynomial Linear Regression

$$y = w_1x_1 + w_2x_2 + \ \ldots + w_nx_n + b$$

$\boldsymbol{y}$: Dependent variable (DV), $\boldsymbol{x_i}$: independent variables (IVs), explaining factors, $\boldsymbol{w_i}$: weights for the factors

Example: y=blood pressure, x_1=age, x_2=weight, x_3=sex, etc.

Matrix Notation:

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \ \ldots + \beta_p x_{ip}$$

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

Cubic Model:

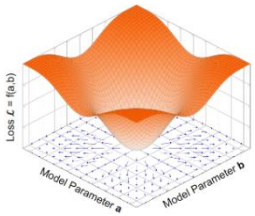
$$\hat{y} = w_1x + w_2x^2 + w_3x^3 + \ \ldots + b$$

6 Stochastic Gradient Descent (SGD)

When we say AI is learning, that means an algorithm is performing some sort of optimization. Optimization is the **problem of finding a set of inputs to an objective function** that results in a maximum or minimum function evaluation. In our applications the objective is to minimize the loss function.

6.1 Gradient Descent

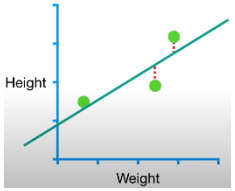
This is a fundamental optimization algorithm.



The gradient is a **vector** in parameter space. It is the **direction of fastest increase of the Loss**. That is, if we change the parameters in the direction of the gradient, the loss increases. If we "move" in the opposite direction, the loss decreases.

At each iteration, the model parameters are updated such that the Loss (MSE) is reduced. The procedure continues until the result converges (no / very small changes).

Gradient Descent to optimize Linear Regression:



The three points (Weight/Height) have the coordinates:

(0.5/1.4)
(2.3/1.9)
(2.9/3.2)

1. Pick a random value for slope and intercept, it's an initial guess, that gives Gradient Descent something to improve upon $slope = 1,\ intercept = 0$
2. We define a learning rate $lr = 0.01$
3. Calculate MSE: $\frac{1}{2N}\sum_{i=1}^{N}(y_i - (slope \cdot x_i + intercept))^2 = \frac{1}{2N}\left(\left(1.4 - (1 \cdot 0.5 + 0)\right)^2 + \left(1.9 - (1 \cdot 2.3 + 0)\right)^2 + \left(3.2 - (1 \cdot 2.9 + 0)\right)^2\right) = \frac{1}{6}(0.81 + 0.16 + 0.09) = 0.17\overline{6}$
4. Take the derivative of the MSE for each parameter in it (slope & intercept). It's also called take the Gradient of the Loss function.

$$\frac{d}{d\ slope} = \frac{1}{N}\sum_{i=1}^{N} -x_i(y_i - (slope \cdot x_i + intercept))$$
$$= \frac{1}{3}(-0.5(1.4 - (1 \cdot 0.5 + 0)) - 2.3(1.9 - (1 \cdot 2.3 + 0)) - 2.9(3.2 - (1 \cdot 2.9 + 0)))$$
$$= -0.1\overline{3}$$

$$\frac{d}{d\ intercept} = \frac{1}{N}\sum_{i=1}^{N} -(y_i - (slope \cdot x_i + intercept))$$
$$= \frac{1}{3}(-(1.4 - (1 \cdot 0.5 + 0)) - (1.9 - (1 \cdot 2.3 + 0)) - (3.2 - (1 \cdot 2.9 + 0))) = -0.2\overline{6}$$

5. Calculate the new slope and intercept and repeat step 4 with the new values

$$new\ slope = slope - \frac{d}{d\ slope} \cdot stepsize$$
$$= 1 - (-0.1\overline{3} \cdot 0.01) = 1.001\overline{3}$$

$$new\ intercept = intercept - \frac{d}{d\ intercept} \cdot stepsize$$
$$= 0 - (-0.2\overline{6} \cdot 0.01) = 0.002\overline{6}$$

6. Repeat steps 4 and 5, until all of the step's sizes are very close to 0 (threshold for stop can be defined) or we reach de defined maximum number of steps

6.2 Stochastic Gradient Descent

When you have millions of data points or thousands of parameters in the loss function, the Gradient Descent takes ages. **O(parameters * datapoints * steps)** Stochastic Gradient Descent uses a **randomly selected subset of the data at every step**, rather than the full dataset. This reduces time spent calculating the derivatives of the Loss function. Gradient-based methods only work if we can express a Loss-function as a **differentiable function**. This is not always the case.

In the example in Gradient Descent, SGD would only **calculate the two derivatives for one point per step**. It is **especially useful when there are clusters of points** in the data. Like GD, SGD **is sensitive to the learning rate**, the general strategy is to **start with a relatively large learning rate and make it smaller with each step**, many implementations will take care of this for you by default, it's called **(simulated) annealing**. There are many different options (called **schedules**) how to reduce **alpha** over time. (e.g. exponential decay). Typically, the learning decays to some **lower bound** (e.g. 0.001) and is then kept fix.

Mini-Batch GD: It is rarely used and inefficient to use only one data point per step, instead we use **a few random data points** to calculate the two derivatives. This takes the best of both worlds, GD and SGD: it can result in **more stable estimates in fewer steps** than SGD, but is **much faster** than GD. The smaller the batch size is, the "noisier" the gradient approximation is. Typical batch sizes are 32/64/.../1024.

When new data is added, we can take only one more step from where we left off, using the new sample, instead of starting from scratch.

7 Tools

Data: The internet is a huge source of data, but most data is unstructured. There are curated for example on kaggle.com

Model: usually we do not start from scratch, we can use known architectures or pretrained modes, for example from hugging-face.co, pretrained models contain information about their training data, therefore we can refine huge pretrained models with only a few additional data points, for example stable diffusion for image generation

Hardware resources (CPU/GPU/RAM/Storage): cloud infrastructure is well suited, free instances with limited time can be used on colab.research.google.com

8 Risks

Bias and fairness issues in AI algorithms: algorithms are created by people, which are never fully objective, never just accept decisions from algorithmus, they must be explainable, auditable and transparent

Privacy and data security concerns: personal data collected by AI systems can be used by businesses for marketing, AI apps like self-driving cars can track your location and habits, AI can predict which information you want to see, creating a "filter bubble", risk of data breaches because of the amount of data that AI collects and processes

Ethical implications of autonomous systems: respect human rights, impact on individual and societal well-being should be a central criterion in the development of AS, individual's ability to maintain appropriate control over their personal data must be respected, safety must be prioritized, accessibility

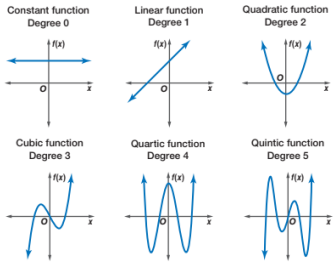
Job displacement and economic impacts: AI has the potential to displace 30% of the jobs, but it also creates new ones, AI could deliver additional global economic activity of around $13 trillion by 2030 or about 16% higher cumulative GDP

AI-enabled misinformation and deepfakes: AI systems are playing an overreaching role in the disinformation phenomena, it is easier now to create realistic fake content like deepfakes, states will abuse it

9 Regularization

The central challenge of ML: The model must perform well on new, unseen inputs.

Polynomial models of degree 0-5:

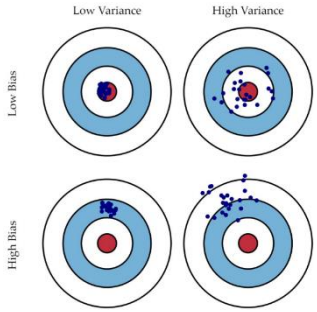


Out-of-sample/Generalization/Test Error: We receive a new data sample (x_{unseen}, y_{unseen}) and calculate $\hat{y}_{unseen} = h(w, x_{unseen})$, this error is the difference between $\hat{y}_{unseen}$ and y_{unseen}.

Our goal is to learn a model from data that generalizes well to new data. A "good" model has a low generalization error. It is possible, that a more complex model has a lower in-sample error but a higher out-of-sample error.

We can't calculate the generalization error, because we don't have new data, only the data we were given. But we can estimate it using the simple technique **split**. A common split-ratio is 80/20 (80% of the data for training and the other 20% for testing). First, we fit the model to the training set to minimize the in-sample error, then we evaluate the model against the test set to estimate the out-of-sample error.

9.1 Bias-Variance Trade-off

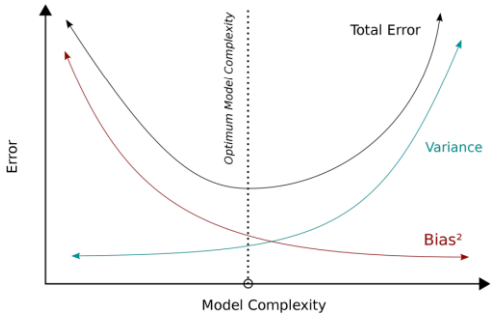


bias=accuracy

variance=precision

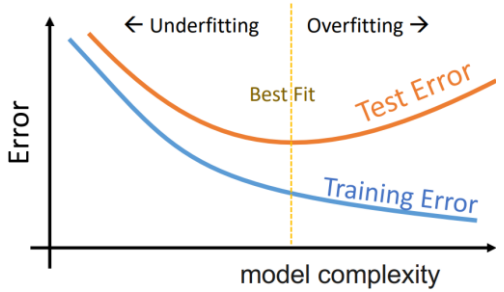
underfitted model: **high bias** (failed to learn underlying structure in the data), **low variance** (stable model, for a change in data, we would fit (almost) the same model)

overfitted model: **low bias** (complex model can explain the data well), **high variance** (optimizer has learned noise the data, MSE (almost) 0, high generalization error)



Trade-off: higher bias → lower variance, lower bias → higher variance

When building a supervised machine-learning algorithm, the goal is to achieve low bias and low variance for the most accurate predictions. We would just build a model «as complex as the data permits».



9.2 Regularization

Regularization is a technique to control the model complexity, it helps us to find the "Best Fit" spot regarding bias, variance, training error and test error.

Simple way: reduce polynomial degree to avoid overfitting, increase degree to avoid underfitting

Regularization adds a constraint to the model, rather, its optimizer, to achieve this. We add a penalty term to the loss function, aka, cost function. Optimizer fits the data (minimize MSE) and also minimizes the constraint. Penalty is typically a constraint over the weights (slope for degree 1) of the model.

We need 2 loss functions:

1. Cost function for training/optimization
2. Loss Function for error calculation in prediction $\hat{y}$

It is common to have two separate functions as the function for 2 may not be differentiable. But we will use MSE for both.

9.2.1 Ridge Regression (L2)

We can add a constraint to optimization to control model complexity, so we need a way to measure the model complexity.

This is the L2-Norm (Euclidean Norm): $\sum_{j=1}^{p} w_j^2$

We add it to MSE (this is number 2 from above):

$$MSE_{ridge}(\mathbf{X}, h(\mathrm{w}, \mathrm{x})) = E = \frac{1}{2N}\sum_{j=1}^{N}\left(\boldsymbol{y_j} - h(\mathrm{w}, \mathrm{x}_j)\right)^2 + \lambda \sum_{j=1}^{p} w_j^2$$

$\mathrm{MSE}_{\mathrm{ridge}}$ is now minimized during training, so we have 2 measures: performance/regression error: MSE, complexity/regularization term: Ridge

λ is a hyperparameter, that does not belong to the optimization process as such. It is varied to find the best fit. 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. It can have any value from 0 to positive infinity.

9.2.2 Lasso Regression (L1)

We can add a constraint to optimization to control model complexity, so we need a way to measure the model complexity.

This is the L1-Norm (Manhattan Distance/Taxicab norm): $\sum_{j=1}^{p}|w_j|$

We add it to MSE (this is number 2 from above):

$$MSE_{lasso}(\mathbf{X}, h(\mathrm{w}, \mathrm{x})) = E = \frac{1}{2N}\sum_{j=1}^{N}\left(\boldsymbol{y_j} - h(\mathrm{w}, \mathrm{x}_j)\right)^2 + \lambda \sum_{j=1}^{p}|w_j|$$

$\mathrm{MSE}_{\mathrm{lasso}}$ is now minimized during training, so we have 2 measures: performance/regression error: MSE, complexity/regularization term: Lasso

Lasso is likely to force the weights to 0 as compared to Ridge. Lasso enables us perform feature selection -- making certains weights 0.

9.2.3 Ridge vs Lasso

Same: they make our predictions less sensitive to the training date and create less generalization error, they can be applied in the same context

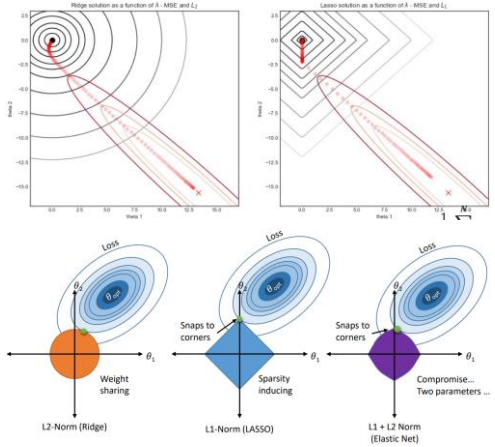
Ridge can shrink the slope asymptotically close to 0, Lasso to 0.

Example: $y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$ & λ increases

Ridge: w_1 & w_2 shrink a bit, w_3 & w_4 shrink a lot

Lasso: w_1 & w_2 shrink a bit, w_3 & w_4 shrink to 0 and go away

So Lasso is better for models which contain a lot of useless variables, Ridge does better when most of the variables are useful.



10 Feature Scaling

For example, we have a dataset with the number of bedrooms x_{brooms} which ranges from 1 to 8 and the sqft area x_{area} which ranges from 370 to 4980.

A multiple linear regression model to predict the price would look like this:

$$\hat{y_j} = h(\mathrm{w}, \mathrm{x}_j) = w_{brooms}x_{brooms,j} + w_{area}x_{area,j}$$

As we can see, x_{brooms} has almost no impact on the MSE (price), because of the small numbers in comparison to x_{area}. Therefore w_{brooms} must be much higher than w_{area}. The problem is that regularization penalizes larger weights (smaller scales) more than smaller weights (larger scales), so we must put all the features on equal footing.

When using Gradient descent, the derivatives of the parameters will be much bigger for the area than for the bedrooms. Therefore, the steps will also be much bigger for the area than for the bedrooms.

The goal is that both should have equal impact on MSE. The Sklearn Standardscaler will rescale a dataset to a mean of 0 and standard deviation of 1.

Raw	2	4	4	4	5	5	7	9
Normalized	-1.403	-0.468	-0.468	-0.468	0	0	0.935	1.871

$$X_{mean} = \frac{2+4+4+4+5+5+7+9}{8} = 5$$

Sample Std. Deviation = s = $\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - X_{mean})^2}$

$s = 2.138$

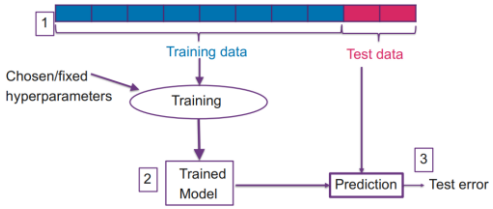
Normalized value = $x_{std} = \frac{x - X_{mean}}{s}$

Example for 2: $\frac{2-5}{2.138} = -1.403$

After doing this, x_{brooms} ranges from about -2.6 to 8.5 and x_{area} ranges from about -2 to 3.6.

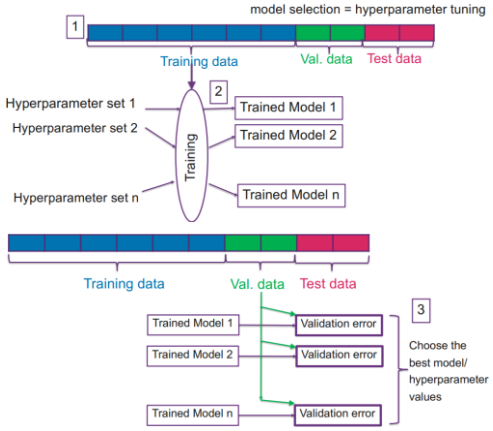
11 Cross-Validation

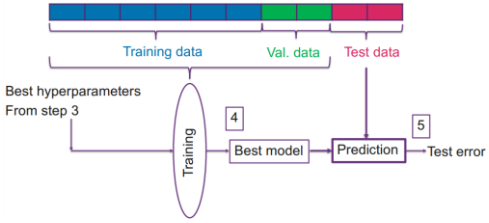
2-way holdout:



Optionally, the trained model with fixed hyperparameters from above is trained again on the whole data. That will then be the **final model**.

3-way holdout:

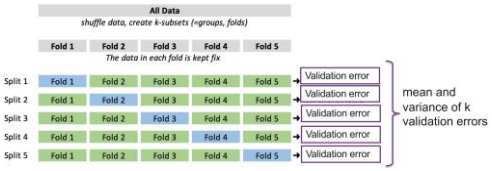




Optionally, the trained model with fixed hyperparameters from above is trained again on the whole data. That will then be the **final model**.

Problems with holdout: training error may be too optimistic about generalization, test error may be too pessimistic about generalization, test and training data may not be representative over all dataset

The solution: **cross-validation**, it is an extension of the holdout method, it's a technique to compare different parameter values (model evaluation), used to obtain a better estimate of the generalization error, **we only study k-fold cross-validation**, there are others

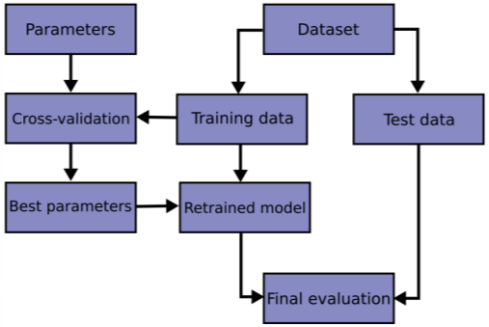


used for training, used for validation

The data is divided into k parts, every part is used once for validation and $k-1$ times for training, typical values for k are 5, 10 or N, do not preprocess the whole dataset, apply the preprocessing pipeline (e.g. standardization) to each split. When comparing different models with CV, the best model is the one with the highest average performance over all folds.

Use case 1: estimation of generalization error
Use case 2: model selection using K-Fold CV (different hyperparameters like λ for ridge regression)

Leave one out cross validation (LOOCV): k=N splits the data into as many parts as there are data points



12 Logistic Regression

Binary classification: 2 possible classes, e.g. win/lose
Multi-class classification: >2 possible classes, e.g. win/tie/lose

Example: We have a dataset about MSE admission decisions. The parameters are: x_1=years of working experience, x_2=BS grades, x_3=secondary school grades, y=1/0 (accepted/rejected)

Secondary School Grade	BS Grade	Years of Work experience	Acceptance
4.5	5.5	4	0
5	6	10	1
. . .			
3	4	6	1

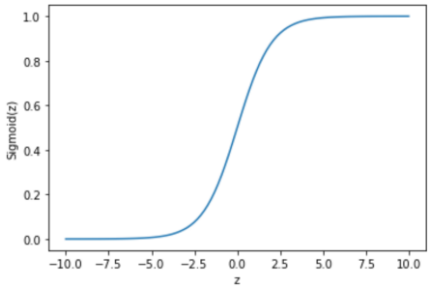
Our model should predict the probability of being accepted on a scale of 0 to 1. We define a threshold (e.g. 0.5) from where applicants are accepted. Linear regression is not suited because: output values are not discrete (0/1), but continuous ($-\infty$ *to* ∞), we model the response y by minimizing the MSE and threshold it to get the probability, but that has nothing to do with the classification probabilities.

We are interested in probabilistic output:

$$P(y = 1 \mid x_1 = 4.5, x_2 = 5, x_3 = 5.5)$$

We want to optimize using the probabilities and not the response. That can be done with the **sigmoid function**:

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



$$\mathbf{z = h(w, x)} = w_1x_1 + w_2x_2 + w_3x_3 + \cdots$$

We can plug in our features (x_i) and the weights (w_i) are unknown, they need to be learned.

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

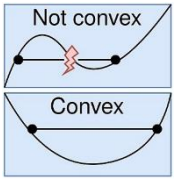
$$\Pr(y = 1 \mid \mathbf{x}) = 1 - \Pr(y = 0 \mid \mathbf{x})$$

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

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

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

How to find the weights?



MSE no suited, because it is **not convex** for this non-linear model with many local minima, in which GD can get stuck

we can use GD to find the optimum weights $W^T$with the convex **Maximum Likelihood** cost function

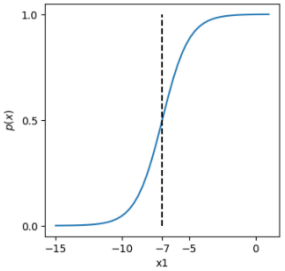
Maximum Likelihood: maximize likelihood of correct prediction, p is close to 1 when $y = 1$ and p is close to 0 when $y = 0$, the goal of this algorithm is to find the best fitting squiggle (sigmoid for the points)

Function to be minimized:

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

Decision Boundary for this sigmoid function:

$$\Pr(x) = \frac{1}{1+e^{-z}}$$
$$\rightarrow z = (x_1 + 7)$$
$$0.5 = \frac{1}{1+e^{-z}} \rightarrow z = 0$$
$$0 = x_1 + 7 \rightarrow x_1 = -7$$



13 Evaluation of Classification

How to evaluate the classifier models, accuracy is not always a preferred performance measure for classifier

Predicted class = $\begin{cases} 0 \text{ if } p < threshold \\ 1 \text{ if } p \geq threshold \end{cases}$

Sec. Grade	BS Grade	Years of Work exp.	Acceptance (Ground Truth)	Predicted probability (p)	Model Class Prediction	
4.5	5.5	4	0	0.8	1	False positive
5	6	10	1	0.2	0	False negative
4	5	7	0	0.4	0	True Negative
3	4	6	1	0.6	1	True positive

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),

13.1 Accuracy

$$Accuracy = \frac{TP + TN}{n}$$

most common metric to evaluate classifiers, does not fully describe the performance of the model

Negative example: 10 out of 1000 patients are sick, the test says 1000 are not sick, $\frac{0+990}{1000} = 99.9\%$ accuracy

13.2 Error

$$Error = \frac{FP + FN}{n}$$

13.3 Sensitivity/Recall/True Positive Rate (TPR)

false negatives worse than false positives at e.g. corona tests, FN=person spreads it further, FP=healthy person quarantined

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

13.4 Miss Rate/False Negative Rate (FNR)

$$FNR = 1 - TPR$$

13.5 Specificity

the percentage of people who test negative for a disease among a group of people who do not have the disease

$$Specificity = \frac{TN}{TN + FP}$$

13.6 Precision

false positives are worse than false negatives at e.g. email spam classification, FN=spam in inbox, FP=important mail in spam

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

13.7 False Positive Rate (FPR)

$$FPR = \frac{FP}{FP + TN}$$

13.8 F1-Score

Combination of precision and recall, both are high → F1-Score high, one is low → F1-Score is low

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Accurracy, error, recall and precision are useful, but can be fooled. Decide which errors are more expensive than others. It is decided by the problem at hand, which one is important.

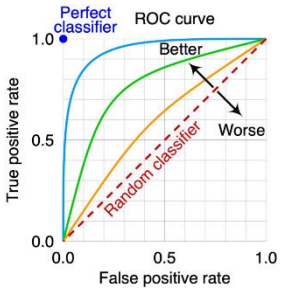
$$F_\beta = \frac{(1 + \beta^2) \cdot Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}$$

$$\beta = 0 \rightarrow F_\beta = P$$

$$\beta = 1 \rightarrow F_\beta = F_1$$

$$\beta = \infty \rightarrow F_\beta = R$$

13.9 Receiver Operating Characteristics (ROC)

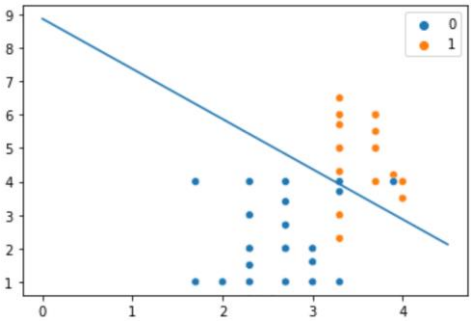


The area under the curve (**AUC**) should be as big as possible, perfect would be area=1

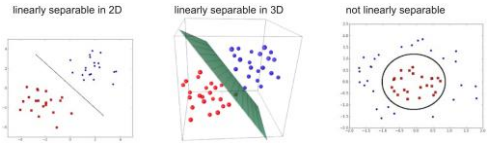
If you have a value for FPR and TPR, you have one point of the curve. You need to evaluate a classifier with different thresholds in the interval [0,1] to get many points which then create the curve.

14 KNN Classification

14.1 Decision Boundary



Y-Axis=years of work experience, **X-Axis**=BSc grade, **blue line**=linear decision boundary for logistic regression



When the classes are not linearly separable and we have additional information about the structure of the data, we can apply a non-linear transformation.

Logistic Regression can be extended to multiple classes:

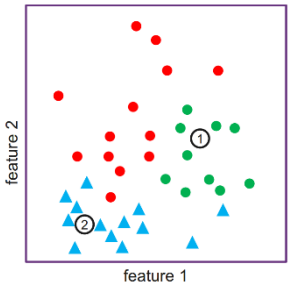
- ***One-vs-rest***: Single classifier trained for each class *c* with the samples of class *c* as positive samples and all other samples as negatives. All classifiers are applied to an unseen sample and the highest *p* is the class.
- ***One-vs-one***: Train a classifier to distinguish between **each** pair of classes. All classifiers are applied to an unseen sample and the results combined produce final classification.

Logistic regression: parametric model (W), that needs training to find optimum W, computer the probabilities, not the class, a suitable threshold needs to be determined (ROC/AUC, FPR vs. TPR) to decide the class, multiple classes can be supported recursively, linear decision boundaries between classes, non-linear boundaries can be supported, finding the best fit via regularization

Dream Method: No training, predicts classes, supports multiple classes, non-linear decision boundaries

Solution: KNN

14.2 K Nearest Neighbours



"A datapoint is known by the company it keeps."

14.2.1 Preparations

1. Load the training as well as test data.
2. Choose the value of k
3. Choose a distance metric

For choosing k and distance metric we can use: test-train split, cross validation, accuracy, precision, recall

14.2.2 For each test data points x_{test}

1. For all training data x_{train}, calculate the $d(x_{test}, x_{train})$
2. Sort training data in the ascending order of distance
3. Choose the first k data points from the sorted list
4. Return the most frequently occurring class among the k data points as the classification result

14.2.3 Distance metrics

Given $\mathbf{x}_1 = (x_{1,1}, x_{2,1}, \ldots, x_{p,1})$ and $\mathbf{x}_2 = (x_{1,2}, x_{2,2}, \ldots, x_{p,2})$

Cosine distance, $cost\ \theta = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{||\mathbf{x}_1|| \ ||\mathbf{x}_2||}$

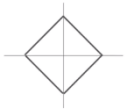
Manhattan Distance , $d_{MH}(\mathrm{x}_1, \mathrm{x}_2) = \sum_{i=1}^{p} |x_{i,1} - x_{i,2}|$

Euclidean Distance, $d_E(\mathrm{x}_1, \mathrm{x}_2) = \sqrt{\sum_{i=1}^{i=p} (x_{i,1} - x_{i,2})^2}$

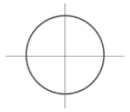
Minkowski Distance

$$d_{MK}(\mathrm{x}_1, \mathrm{x}_2) = \left(\sum_{i=1}^{p} (|x_{i,1} - x_{i,2}|^p)\right)^{1/p}$$

Manhattan:



Euclidean:



14.2.4 Properties

Hyperparameters: k and distance metric, both have a big impact on the decision boundaries and with that a big impact on the precision, recall and accuracy, with $k = N$, every datapoint is classified the same, bigger k → lower variance, higher bias

Advantages: easy and simple ML 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 for fair treatment among features

15 Naive Bayes

applied for classification, simple to implement, works well for smaller datasets, no training phase, used extensively when data contains categorical features but not much used in numerical features

We have the following dataset:

No	Email Text	Spam
1	Hurry Sale Tomorrow	1
2	Rain tomorrow	0
3	Sale price tomorrow	1
4	tomorrow workshop rain	0

We want to know the probability, that an email containing "tomorrow" is spam, more formal: $\Pr(spam|\text{"tomorrow"})$

Bayes Rule:

$$\Pr(spam|\text{"tomorrow"}) = Pr(spam) \cdot \frac{\Pr(\text{"tomorrow"}|spam)}{\Pr(\text{"tomorrow"})}$$

Let's calculate all the factors of bayes rule:

$$Pr(spam) = \frac{\#\ dataset\ entries\ that\ are\ spam}{\#\ total\ entries} = \frac{2}{4} = \frac{1}{2}$$

$$\Pr(\text{"tomorrow"}) = \frac{\#\ entries\ containing\ \text{"tomorrow"}}{\#\ total\ entries} = \frac{4}{4} = 1$$

$$\Pr(\text{"tomorrow"}|spam) = \frac{\#\ spam\ entries\ containing\ \text{"tomorrow"}}{\#\ spam\ entries} = \frac{2}{2} = 1$$

$$\Pr(spam|\text{"tomorrow"}) = \frac{1}{2} \cdot \frac{1}{1} = 0.5$$

16 K-means Clustering

We are given data (features, x), without labels (y). But we can still learn something from the data because it has some structure. The goal of unsupervised learning is to self-discover patterns from the data. Data without any structure is the exception, but the structure can be hidden by noise. The human brain is extremely efficient at noting patterns in data.

Algorithms are also extremely efficient at dealing with large and high-dimensional datasets, where humans fail.

Applications that use clustering: social network analysis, astronomical data, find similar articles, market segmentation, recommendation systems

16.1 Naïve K-means

1. Let us assume that we know the number of clusters k_c
2. Initialize the value of k cluster centres/means/centroids $(c_1, c_2, \dots, c_{k_c})$
3. **Assignment**:
   a. Find the squared Euclidean distance between the centres and all the data points.
   $$\forall_{k=1}^{k_c} \forall_{i=1}^{N} \; \mathrm{d}_{\mathbf{i},k} = (\mathbf{x}_i - \mathbf{C}_k)^2$$
   Example: $d\big((1,2),(3,4)\big) = (3-1)^2 + (4-2)^2 = 8$
   b. Assign each data point to the cluster of the nearest centre
   $$d_{i,k}^{min} = Minimum\big(\mathrm{d}_{\mathrm{i},1}, \mathrm{d}_{\mathrm{i},2}, \dots., \mathrm{d}_{\mathrm{i},k_c}\big)$$
   $$x_i \in Cluster\ k$$
4. **Update**: Each cluster now potentially has a new centre. We update the centre for each cluster, the new centres $(c_1', c_2', \dots, c_{k_c}')$ = average of all data points in the cluster
   $$\forall_{k=1}^{k_c} \boldsymbol{C}_k = \frac{1}{size(\boldsymbol{C}_k)} \sum_{x_i \in cluster\ k} \mathbf{x}_i$$
   Example: center of $(1,2)$ and $(3,4)$: $\left(\frac{1+3}{2}, \frac{2+4}{2}\right) = (2,3)$
5. If some stopping criterion met, done. Else, go to step 3. That means after every new data point, steps 3 and 4 are repeated.

16.2 Properties

Initialization: performance depends on the random initializations of the seeds for the centers, some seeds can result in poor convergence rate or suboptimal clustering, if initial centers are very close together, we need a lot of iterations, therefore run it multiple times with different random initializations and check if the clusters are stable

Possible stopping criterions: centers don't change, datapoints assigned to specific cluster remain the same, set threshold for distance of datapoints from their centers, fixed number of iterations is reached (attention: insufficient iterations lead to poor cluster quality, choose wisely)

Standardization: Features with large values may dominate the distance value, features with small values will have no impact on the clustering (look at the Euclidean distance formula), therefore the features should be standardized beforce executing clustering

Evaluating unsupervised learning models: we have no ground truth labels, therefore we cannot use the evaluation metrics from the supervised learning models like accuracy, precision, recall, etc.

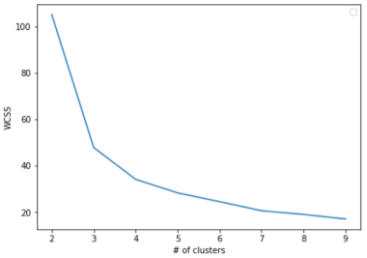
The goal of good clustering is, that for each cluster the distance of each cluster-member from its center is minimized.

Minimize $\sum_{k=1}^{k_c} \sum_{\mathbf{x_i} \in \boldsymbol{Member}(\boldsymbol{C_k})} \boldsymbol{d}(\boldsymbol{C_k}, \mathbf{x})$

Inertia/Within-cluster sum-of-squares (WCSS): sum of squared distances of samples to their closest cluster center

When we increase the number of clusters, when the number of clusters is equal to the number of data pointer → WCSS=0

We can see an elbow at 3-4 clusters, 4 clusters would be a good choice here



Silhouette score: How far away the data points in one cluster are, from the data points in another cluster. Formula for points:

$$\frac{b-a}{\max(a,b)}$$

a = the average distance between each point within a cluster

b = the average distance between a cluster and its nearest neighbour cluster

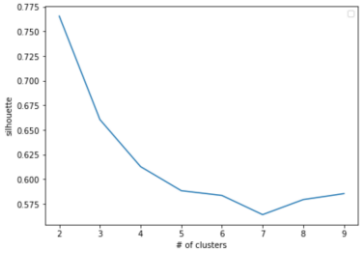
The range is [-1;1], the higher the better.

Example: Cluster1 has points (2,5), (3,4) & (4,6), Cluster2 has points (6,10), (7,8) & (8.9), we want silhouette score of (2,5)

a:
$$d\big((2,5),(3,4)\big) = \sqrt{(3-2)^2 + (4-5)^2} = \sqrt{2}$$
$$d\big((2,5),(4,6)\big) = \sqrt{(4-2)^2 + (6-5)^2} = \sqrt{5}$$
$$a = \frac{\sqrt{2}+\sqrt{5}}{2} = 1.825$$

b:
$$d\big((2,5),(6,10)\big) = \sqrt{(6-2)^2 + (10-5)^2} = \sqrt{41}$$
$$d\big((2,5),(7,8)\big) = \sqrt{(7-2)^2 + (8-5)^2} = \sqrt{34}$$
$$d\big((2,5),(8,9)\big) = \sqrt{(8-2)^2 + (9-5)^2} = \sqrt{52}$$
$$b = \frac{\sqrt{41}+\sqrt{34}+\sqrt{52}}{3} = 6.482$$

$$\frac{6.482 - 1.825}{6.482} = 0.784$$



a good choice would be 4 or 5 clusters

17 Ensemble Methods

Wisdom of Crowd: When you have very difficult question to answer and you don't know the answer, it is a good idea to ask many random people and then aggregate their answers.

This can be applied to ML too, we can aggregate the results of several weak models, instead of finding the best model.

Ensemble: a group of predictors

Case	Result of Each Model			Result of the Ensemble	Probability
	m1	m2	m3		
1	Correct	Correct	Correct	**Correct**	0.7*0.7*0.7 = 0.343
2	Correct	Correct	Incorrect	**Correct**	0.7*0.7*0.3 = 0.147
3	Correct	Incorrect	Correct	**Correct**	0.7*0.3*0.7 = 0.147
4	Incorrect	Correct	Correct	**Correct**	0.3*0.7*0.7 = 0.147
5	Incorrect	Incorrect	Correct	**Incorrect**	0.3*0.3*0.7 = 0.063
6	Incorrect	Correct	Incorrect	**Incorrect**	0.3*0.7*0.3 = 0.063
7	Correct	Incorrect	Incorrect	**Incorrect**	0.7*0.3*0.3 = 0.063
8	Incorrect	Incorrect	Incorrect	**Incorrect**	0.3*0.3*0.3 = 0.027

Probability, that **each model** makes a **correct** prediction: 0.7
Probability, that **ensemble** makes a **correct** prediction: $0.343 + 0.147 + 0.147 + 0.147 = 0.784$ → ensemble is better

Probability, that **each model** makes a **wrong** prediction: 0.3
Probability, that **ensemble** makes a **wrong** prediction: $0.063 + 0.063 + 0.063 + 0.027 = 0.216$ → ensemble is better

But ensemble is not always better, there are cases where it performs worse than one model. Ensemble can be a strong learner when:

- Weak learners/models/predictors are **independent** from one another and make **uncorrelated errors**
- There is a **sufficient number of weak learners**
- The models make **different types of errors**
- The models are **not trained on the same data** (they will make the same type of error otherwise)
- The models are **better than random models**

Diverse models use:

- **Different algorithms** (e.g. KNN & Logistic regression)
- **Different hyperparameters** (KNN: various k, Regression: various regularization parameters)
- **Different training data** (split and/or preprocess data, cross validation, features engineering)

Training is faster, because the different learners are trained faster and can the training of them can be done in parallel

17.1 Voting

17.1.1 Hard voting

class that gets the most votes from the different learners

Hard voting with weights: we have 3 learners with weights $[0.1, 0.3, 0.6]$ and their classification is $[spam,\ spam, ham]$

$$sum_{spam} = w_1 \cdot (prediction_1 == spam) + w_2 \cdot (prediction_2 == spam) + w_3 \cdot (prediction_3 == spam) = 0.1 \cdot 1 + 0.3 \cdot 1 + 0.6 \cdot 0 = 0.4$$

$$sum_{ham} = w_1 \cdot (prediction_1 == ham) + w_2 \cdot (prediction_2 == ham) + w_3 \cdot (prediction_3 == ham) = 0.1 \cdot 0 + 0.3 \cdot 0 + 0.6 \cdot 1 = 0.6$$

The result is $\boldsymbol{ham}$, because the sum is bigger than 0.5. Note that spam has 2 votes and ham only 1 vote with more weight.

17.1.2 Soft voting

Predict the class with the highest class probability, averaged over all classifiers. Only possible if predictions are probabilities aka the classifiers are well calibrated.

Soft voting with weights: we have 3 classifiers with weights $[0.1, 0.3, 0.6]$ and 3 classes

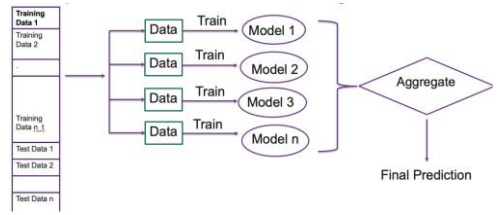
Classifier1: $[\Pr(class_1) = 0.85,\ [\Pr(class_2) = 0.05,\ [\Pr(class_3) = 0.15\ ]$
Classifier2: $[\Pr(class_1) = 0.15,\ [\Pr(class_2) = 0.2,\ [\Pr(class_3) = 0.7\ ]$
Classifier3: $[\Pr(class_1) = 0.65,\ [\Pr(class_2) = 0.03,\ [\Pr(class_3) = 0.9\ ]$

Class1: $0.1 \cdot 0.85 + 0.3 \cdot 0.15 + 0.6 \cdot 0.65 = 0.325$
Class2: $0.1 \cdot 0.05 + 0.3 \cdot 0.2 + 0.6 \cdot 0.03 = 0.083$
Class3: $0.1 \cdot 0.15 + 0.3 \cdot 0.7 + 0.6 \cdot 0.9 = 0.5166$

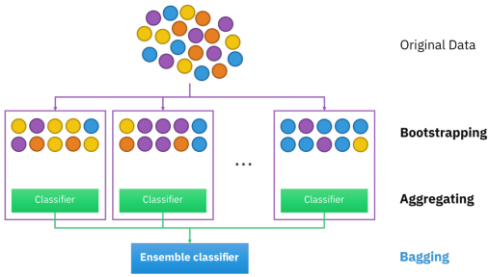
Winner is $\boldsymbol{Class3}$, because it has the highest probability

17.2 Bootstrap Aggregating (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. Reduces variance, can increase bias slightly



sampling with replacement: a data point can be selected more than once for different subsets of data (Bagging)



the individual (simple!) models have a relatively low bias and high variance, bagging (reuse of data) reduces variance, bagging provides a way to reduce overfitting, bagging methods work best with complex models

sampling without replacement: a data point can be selected only once by a subset of data (Pasting)

Out of bag (oob) evaluation: because we are training the different models on subsets of data, they can be evaluated on the data, which is not in the subset. There is no need for a separate validation or cross-validation set.

No free lunch theorem: Why does it make sense to invest (train) multiple simple ("stupid") models, instead of putting all our effort into optimizing a single model? "no single machine learning algorithm is universally the best-performing algorithm for all problems"

Random Subspaces: random subsets of the features

Random Patches: Random subsets of both samples & features

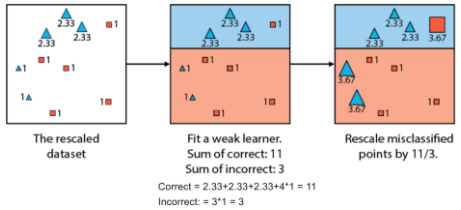
Different features: bagging can be done not only on different samples but also on different features, each predictor can be trained on random subset of features

17.3 Boosting

train predictors **sequentially**, each predictor tries to correct its predecessor, each predictor improves on the weakness of its predecessor, reduces bias, can increase variance slightly

AdaBoost (Adaptive Boosting):

1. AdaBoost assigns equal weights to each training sample (**instance weights**)
2. AdaBoost trains a model to fit the given data
3. AdaBoost increases the weight on the misclassified samples (**instance weights**), the misclassified samples will make up a larger part of the next classifiers training set and hopefully the next classifier trained will perform better on them



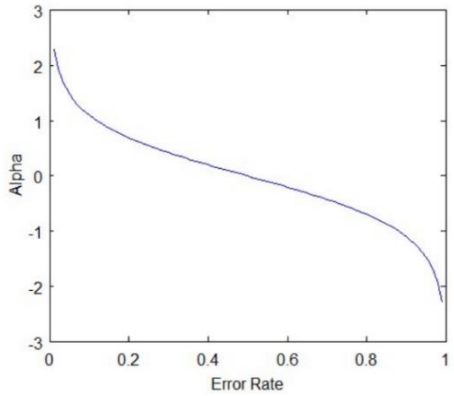
4. Stop if the desired number of estimators are trained
5. AdaBoost then trains a next classifier (step 2)
6. For the Ensemble, each classifier's weight is calculated based on its accuracy (**predictor weight or α**), more accurate classifiers are given more weight

$$Predictor\ weight = \ln\left(\frac{sum\ scores\ of\ correctly\ classified\ points}{sum\ of\ scores\ of\ incorrect\ points}\right) = \ln\left(\frac{accuracy}{1-accuracy}\right)$$

Example from step 3:

$$accuracy = \frac{11}{14} = 0.7$$

$$predictor\ weight = \ln\left(\frac{11}{3}\right) = 1.3$$



Error rate=0.5 → α=0 because it's random guessing
Error rate approaching 0 → exponentially more weight
Error rate approaching 1 → exponentially negative weight, do the opposite than this classifier says

7. AdaBoost makes predicition based on **Hard voting with weights**

Other Boosting methods: Gradient Boosting, Extreme Gradient Boosting, CatBoost, LightGBM:

17.4 Coefficient of Determination (R^2)

- The sum of squares of residuals, also called the residual sum of squares:

$$SS_{res} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

- The total sum of squares (proportional to the variance of the data):

$$SS_{tot} = \sum_i (y_i - \bar{y})^2$$

The most general definition of the coefficient of determination is

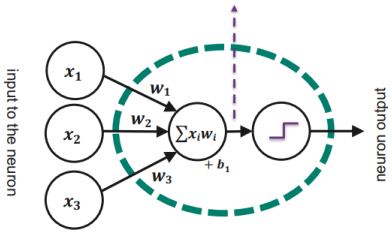
$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

18 Artificial Neural Networks

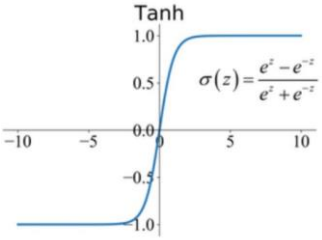
18.1 Artificial neuron/Threshold Logic Unit (TLU):

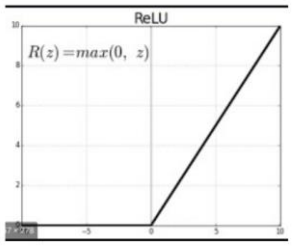
The TLUs have been trained to find the right values for the weights and bias, a technique for that is **backpropagation**.

$$w_1x_1 + w_2x_2 + w_3x_3 + b_1$$



The neuron calculates the sum of the weighted input (dot product $\vec{x} \cdot \vec{w}$), adds a bias b, and passes it through a nonlinear activation function (input x → output y). Examples of activation functions are the **sigmoid function**, **Tanh** or **Rectified Linear Unit (ReLU)**.





Step functions are also specific types of activation functions. They are very useful for binary classification problems. But they are not differentiable, therefore they cannot be used with backpropagation. Examples of step functions:

$$heaviside(z) = \begin{cases} 0\ if\ z < 0 \\ 1\ if\ z \geq 0 \end{cases}$$

$$sign(z) = \begin{cases} -1\ if\ z < 0 \\ 0\ if\ z = 0 \\ +1\ if\ z > 0 \end{cases}$$

18.2 Perceptron

Perceptron is one of the simplest Artificial neural network architectures. It was introduced by Frank Rosenblatt in 1957s. The nodes are TLUs, and they use the Heaviside step function.

Single-Layer Perceptron: There is only TLU. This type is limited to a linear decision boundary and cannot learn complex patterns. It can easily classify instances simultaneously into multiple classes It can implement the logical gates AND, OR & NOT.

Multilayer Perceptron: Multilayer perceptron's possess enhanced processing capabilities as they consist of two or more layers (1x input, >0x hidden, 1x output), adept at handling more complex patterns and relationships within the data. It is also called **Artificial Neural Network (ANN)**.

Deep neural networks (DNNs): ANNs with multiple hidden layers. They are trained like other supervised learning techniques, on a dataset with known input and output. We start with random weights and an optimizer like SGD reduces a loss function like MSE using the ANN output and the known output that should appear. Alternatives to MSE and maximum likelihood:

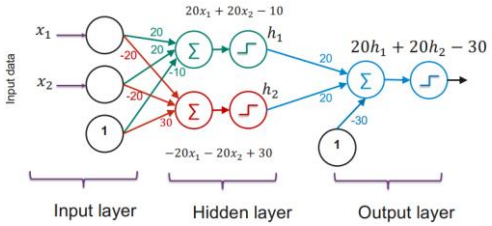
$$\text{Binary Cross Entropy Loss} = \frac{-1}{N}\sum_{i=1}^{N}(\boldsymbol{y_i} * \log(p_i))$$

$$\text{Categorical Cross entropy Loss} = \frac{-1}{N}\sum_{i}^{N}\sum_{j=1}^{m}(\boldsymbol{y_{ij}} * \log(p_{ij}))$$

Backpropagation: An efficient algorithm for training a DNN. It is basically GD in reverse-mode.
Forward Pass: give the DNN a sample, let it go all the way to the end and measure error using a loss function
Backward Pass: go through each layer in reverse order to measure error contributions from each connection, finally tweak the connection weights to reduce the error (GD)

Hyperparameters of backpropagation: GD learning rate, number of steps for training, batch vs mini-batch vs SGD, number of layers, number of neurons in layer, activation functions used, regularization parameters (L1/L2/Lambda)

An example of a more complex pattern like XOR (3 neurons, 2 layers, 11 trainable parameters):

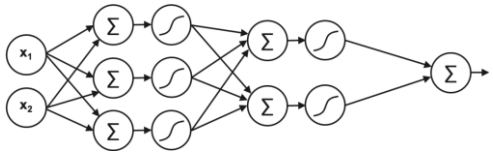


$x_1 = 0, x_2 = 0$
- $h_1 = heaviside(-10) \Rightarrow 0,$
- $h_2 = heaviside(30) \Rightarrow 1,$
- $y_{pred} = heaviside(-10) \Rightarrow 0$

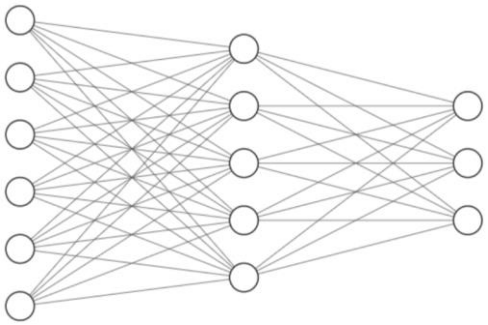
$x_1 = 1, x_2 = 0,$
- $h_1 = heaviside(10) \Rightarrow 1,$
- $h_2 = heaviside(10) \Rightarrow 1,$
- $y_{pred} = heaviside(10) \Rightarrow 1$

- $x_1 = 0, x_2 = 1,$
  - $h_1 = heaviside(10) \Rightarrow 1,$
  - $h_2 = heaviside(10) \Rightarrow 1,$
  - $y_{pred} = heaviside(10) \Rightarrow 1$
- $x_1 = 1, x_2 = 1$
  - $h_1 = heaviside(30) \Rightarrow 1,$
  - $h_2 = heaviside(-10) \Rightarrow 0,$
  - $y_{pred} = heaviside(-10) \Rightarrow 0$

Draw ANN: two dimensional input, first hidden layer with 3 neurons, second hidden layer with 2 neurons, output layer with 1 neuron and without activation function



Calculate number of parameters: 6 unit input layer, 5 unit hidden layer, 3 unit output layer



hidden layer: weight for each unit of input layer and a bias per unit: $(6 \cdot 5) + 5 = 35$

output layer: weight for each unit of hidden layer and a bias per unit: $(5 \cdot 3) + 3 = 18$

$Total: 35 + 18 = 53$

18.3 Softmax

Takes in a vector of raw outputs of the neural network and returns a vector of probability scores for a sample belonging to class k.

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{m} e^{z_j}}$$

Example: classes cat, dog, bird, fish, NN-output: $[1,6,2,3]$

$$p_{dog} = \frac{e^6}{e^1 + e^6 + e^2 + e^3} = 0.93$$

18.4 Universal Approximation Theorem

Neural networks can represent a wide variety of interesting functions when given appropriate weights. On the other hand, they typically do not provide a construction for the weights, but merely state that such a construction is possible.

Most universal approximation theorems can be parsed into two classes:

- The first quantifies the approximation capabilities of neural networks with an arbitrary number of artificial neurons ("arbitrary width" case)
- the second focuses on the case with an arbitrary number of hidden layers, each containing a limited number of artificial neurons ("arbitrary depth" case)

Multilayer feed-forward networks with as few as one hidden layer are universal approximators. The multilayer feed-forward architecture gives neural networks the potential of being universal approximators.