01-03-2022 - 26-04-2022, EVERY TUESDAY FROM 18-20h

MACHINE LEARNING SPRING SCHOOL

THE BASICS OF ML 4 SOCIAL GOOD

FROM CORRELAID #LC-MUENCHEN

This module is comprised of two parts

– The first week can be used to study this presentation and try to tackle the problem of classifying trees on your own.

– Hence this is an invitation to explore some of the possibilities for yourself or with the others from the course.

– After the first week, you will get a notebook from me which you can compare with your own approaches taken in the first week.

– In our session we will discuss some of the findings you made.

I want to encourage you to explore the data and make findings by yourselves and in a group or team like real Data Scientists at a company or research institution would do.

Please keep in mind that participants have different skill levels and everyone should learn as much as they can from each other.

Ask yourselves the following questions

– What is the difference between a Data Scientist, Data Engineer, Machine Learning Specialist and how do you think do they approach a problem?
– What does LEARNING in Machine Learning even mean?
– If you were an entrepreneur, would you see Machine Learning as a tool to build a product or as a product in itself?
– Would you trust a car‘s autopilot so much that you would dispose of the car‘s pedals and steering wheel?
– What do you think are common misconceptions when talking about Machine Learning and specifically so called Artifical Intelligence?

Data Engineer

Profile

- Assessing the data infrastructure.
- Assessing already existing pipelines.
- Automating the process to collect, centralize and pre engineer the data such that Data Scientists can work with it.
- Guaranteeing consistent quality of the data, by implementing validations, tests etc.

Proficiencies

- Knowledgeable in everything data base management systems.
- Myriad of possible pipeline solutions, even automated data base management (creation, updating, dropping) systems are more and more introduced.
- General understanding of "useful" data and their (relational) structure in an IT ecosystem.

My experience has shown that companies introducing Machine Learning models have the data, but in most cases do not know what to do with it. Hence, there is no useful infrastructure present to develop useful Machine Learning on.

Data Scientist

Profile

- Assessing the (possibly existing) inherent structure of the data or parts of it.
- By experience or intuition in the applied to field or past work knows what to look for to get useful results.
- Is able to implement and use Machine Learning Models, furthermore does know their limitations.

Proficiencies

- The more diverse the background knowledge of a team of Data Scientists the better (a social scientist differently analyses problems compared to a physicist).
- Knowledge of applied Machine Learning Models in the form of programming and mathematical theory.
- Has the ability to feature engineer useful information into the existing data.

Usually Data Scientists do some of the work of a Data Engineer and Machine Learning Specialist too, especially useful feature engineering is something all profiles share.

Machine Learning Specialist

Profile

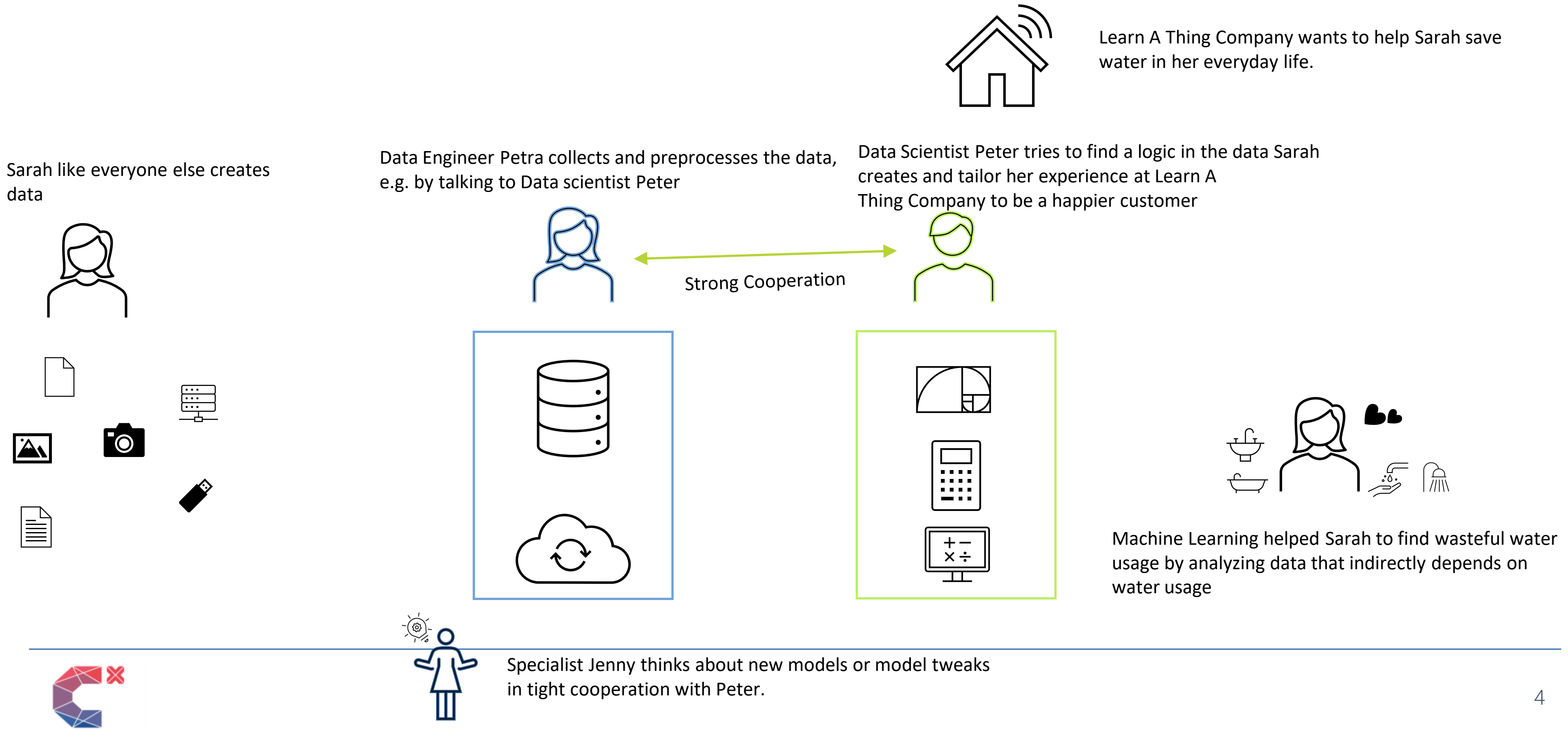
- Deep and intricate understanding of the theory behind models.
- Mathematical understanding of modelling and the learning process of the models.
- Can ask and (try to answer) theoretical questions how to improve models under certain assumptions.

Proficiencies

- Theoretical tools to approach Machine Learning questions and problems.
- Proficiency in statistics and optimization.
- Ability to implement proof of concept versions of theory, e.g. implementing a Support Vector Machine w/o using for example scikit-learn (NO easy task, doing it correctly).

Although Data Scientists need some of a Specialists's tools, "you do not have to be a mathematician to apply a linear regression or Support Vector Machine and understand what it does".

The Process



Learn A Thing Company wants to help Sarah save water in her everyday life.

Sarah like everyone else creates data

Data Engineer Petra collects and preprocesses the data, e.g. by talking to Data scientist Peter

Data Scientist Peter tries to find a logic in the data Sarah creates and tailor her experience at Learn A Thing Company to be a happier customer

Strong Cooperation

Machine Learning helped Sarah to find wasteful water usage by analyzing data that indirectly depends on water usage

Specialist Jenny thinks about new models or model tweaks in tight cooperation with Peter.

General Introduction to Modelling

Or, what are we even doing?

What are you going to learn?

- To help you to learn and understand on your own, I want to give you a more general view on some of the more common models used in Data Science.
- We will take the role of someone who is a mix of a Data Scientist and a Machine Learning Specialist.
- The theory presented here should help you with working on the actual problem you already know from the last sessions.

Supervised and Unsupervised Learning

Supervised Learning

We are given pairs of Variables (X, Y)

consisting of a *Feature or Input Vector X* and a *Label or Output Vector Y*. In most cases it holds:

$$X \in \mathbb{R}^d.$$

Then we can make two distinctions:

1. If it holds that: $Y \in \mathbb{R}$ we talk about *regression*.
2. If it holds that: $Y \in \{1, \ldots, K\}$ we talk about *classification*.

In practice, the *dimension of the Input d* is (very) large. Although this can lead to the so-called *Curse of Dimensionality* – meaning massive optimization problems with increasing dimension and hence problems finding solutions to a given problem – there are models accounting for this.

Unsupervised Learning

We are only given the Input Vector X without an observed label Y.

Here, the task is to find a structure in X which is not generated by random “noise”.

Theoretically this means to do inference on the probability distribution of X.

Examples for unsupervised problems are:

1. Reduction of the complexity of data.
2. Reduction of the problem’s dimension.
3. “Clustering” data, although not necessarily classifying them.

From Data To Algorithm 1/2

We want to find a (deterministic) function

$$f^* : \mathcal{X} \to \mathcal{Y}, X \mapsto f^*(X) = Y,$$

where $X \in \mathcal{X} \subset \mathbb{R}^d, Y \in \mathcal{Y} \subset \mathbb{R}$ for regression and $\mathcal{Y} \subset \{1, \ldots, K\}$ for classification.[1]

In reality, we are given n pairs of data $(X_i, Y_i), i = 1, \ldots n.$ Before we connect these observations with above function we have to consider the following technicality.

Throughout the remainder we assume that the pairs (X_i, Y_i) **are identical and stochastically independent**.

1. Independence means that for two arbitrary observations $(X_i, Y_i), (X_j, Y_j)$ they do not influence each other. (For example, time series from economic variables are highly dependent without further transformations.)
2. Identical means that they all follow the same probability distribution. Meaning that they all follow the same structure. There exist models for the cases where one assumes or does not assume such a distribution.

1 This is just some fancy mathematical notation for two things. There exist "spaces" for a function that maps data given as X onto a value given by Y. The cursive X and Y are just some abstraction, as your values do not always have to be in the "real" physical space. Thus you can interpret these just as you would in school. There is a function mapping your data onto a number or a class, where "f star" is the real function which we want to estimate with our Machine Learning models.

From Data To Algorithm 2/2

The problem we face is that we do not know the true function f^*.

Although, we can limit the “space” of possible functions, which we call a *model assumption,* e.g. by assuming that the dependance between X and Y is linear.

Given this model assumption and the data given in the form of the n pairs (X_i, Y_i), we can define an approximating function $\hat{f}_n$ which depends on our data and maps an Input Vector X onto an Output Y:

$$\hat{f}_n\,(X_1, Y_1, \ldots X_n, Y_n | X) = Y.$$

Machine Learning is concerned with finding useful model assumptions (recall that you could predict anything from everything or nothing at all) and useful optimization methods to approximate with $\hat{f}_n$ the true dependence f^*. Thus, in the following, we will introduce different approximations for different model assumptions which we can call algorithms or machine learning models.

Cross Validation – CV (1/2)

A problem we face in Machine Learning algorithms is directly depending on the training or fitting of a model, given your data. Think of the following scenario.

You fit your cool model with the data you collected. But if you apply it to new data its results are obviously useless.

This happens (a lot) if you do not take necessary precautions. Think of it this way:

the model is optimized with respect to predicting your training data. This data may follow rules which not necessarily are the real structure of the problem, it is a sample. We have to account for this in the training process, i.e. think about the case that new data could look quite different from the data collected in the past. As we can not take real “future” data as we are no wizards, we solve this problem with the so-called cross validation.

For this we do a train and test split of the data. Usually this is done by taking a random collection of your data you already have, train the model on this data and let it predict the other data. Whereas the first pool of data is the training set, the other is the test set. Formally you would do the following (there are two main ways of doing CV):

Cross Validation – CV (1/2)

Leave one out:

In this case you just take one observation of your data, train your algorithm on the rest. Then you would predict the one missing value which was taken out and measure the error your model did when predicting it from the "new" data point. This means that for n observations you would have to run your algorithm n times.

"Normal" or M-Fold Cross Validation:

In this case you split your complete data set into M parts. Then you would iterate over each part in the following way. You take the first of M parts and train your model on the other M – 1 parts. Then you predict the data on the left-out part and measure the error. This process is repeated for each of the M parts.[1] Hence, you try to predict each block of data after training the model on the rest of the collection. Then, after you measured the errors each of the M iterations made you try to find the best suiting model.

Validation data sets are a third step which you would use to do cross validation if hyper-parameters[2] are involved. If you find the time read up on it! wikipedia

1 Again you would separate the data into the M parts by randomly choosing a datapoint and assigning it to one of the M subsets of data. This process is sometimes an art in itself. For example you could force that each of the M parts is the same size, or not. Or a more complex way to split the data is by trying to find observations that are mostly following the same distribution (which implies a lower variance within the data). This is called stratified Cross Validation. In the end, if its mathematically sound there are not boundaries on this part. And a lot of good results come from experience.
2 Introduced later.

Regression

By now you may have already guessed that the tree dataset leads to a classification problem. Nevertheles, regression is as import as classification and should not come short.

We will introduce three different regression models, but have to start with a short introduction to the overall problem.

Assume we have a true dependence of Y on X given by $Y = f^*(X)$. Where X is a d dimensional vector and Y is a real number. In its most simple form we assume a linear dependence, thus:

$$Y = f^*(X) = \sum_{j=1}^{d} \beta_j X_j + \epsilon$$

Here, $\beta_j \in \mathbb{R}$ are the real coefficients giving the weight of a feature with regards to the Output Y. Furthermore, the $\epsilon \in \mathbb{R}$ is a noise or error term. These can be most generally understood as errors introduced by measuring or collecting our data. Now, we can introduce n such observations to get:

$$Y_i = \sum_{j=1}^{d} \beta_{ij} X_{ij} + \epsilon_i$$

Regression

This again can be written in matrix notation as: $Y = \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} X_{11} & X_{12} & \dots & X_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & \dots & X_{nd} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_d \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_d \end{pmatrix} = X\beta + \epsilon.$

In most classes and textbooks the most understood case where the error term ϵ is a Gaussian with mean 0 and bounded variance is presented.

In statistics or machine learning you want to find the weights β and there is rich theory how to approximate f^* by using $\hat{f}_n(X) = X\beta + \epsilon.$ Here approximation is literally visible as we have to account for the error term.

Many of you already know that there is a direct estimator for the weights under the assumption that the error is a Gaussian, given as:

$$\hat{\beta} = \left(X^t X\right)^{-1} X^t Y.$$

Here X^t is the transpose of a matrix and X^{-1} its inverse. So why do I introduce you to this?

Regression is prone to the curse of dimensionality. Recall that this means that the dimension d of our inputpace is very big. Furthermore, if we have more parameters than observations in our model $(n << d)$ the general question would be how useful our estimations are.

Curse of Dimensionality

Unfortunately this is not the whole picture of this curse. In addition (and some of you may already know) matrix inversion is computationally very expensive. Although modern machines are more sophisticated, there are lower bounds on the runtime algorithms for matrix inversion can achieve.

Plainly speaking: even for a simple model as linear regression we face the problem that we may not be able to solve our problem with the “standard” approaches you may have learned at school or university.

Before we introduce the following two models which were developed in statistics but found their way into the machine learning canon, we have to think of one more technicality.

Matrix inversion is unfeasible if the matrix’ dimensions are very big and most of the values are not zero (thus we have measurements of X which are not zero). Two questions arise:

1. If most of the values in this very big matrix are zero, can we achieve some runtime optimization?
2. Even if most of the values are not zero, are really all features $X_{j1}, \dots, X_{jd}, j = 1, \dots, n$ of our observations relevant?

These two questions lead in the past to the so-called Ridge and Lasso regression models.

One note: this is just one shortcoming of linear regression applied in reality, hence there are many more methods developed for different kinds of problems that arise. For example, a search for generalized least squares would be a good starting point for further information when the error term behaves in a more complex way.

Short Recap

Recall the empirical error from Berhard‘s talk, given by[1]:

$$\mathcal{R}\left(\hat{f}_n\right) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}_n\left(x_i\right)\right)^2 .$$

This can be rewritten as:

$$\mathcal{R}\left(\hat{f}_n\right) = \frac{1}{n}\sum_{i=1}^{n}\left(\beta_i x_i + \epsilon_i - \hat{\beta}_i x_i\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\left(\beta_i - \hat{\beta}_i\right)x_i + \epsilon_i\right)^2 = \mathcal{R}\left(\hat{\beta}_n\right).$$

Thus, we combined the error measure of a least squares with a model assumption of a linear regression. Our problem is finding a solution to:

$$\min_{\hat{\beta}_n \in \mathbb{R}^n} \mathcal{R}\left(\hat{\beta}_n\right).$$

The mathematical optimal[2] solution to this so called Ordinary Least Squares (OLS) problem is: $\hat{\beta}_n = \left(X^t X\right)^{-1} X^t Y.$

1 Using this empirical error function one talks about least squares problems.

2 Mathematical optimal in itself can get rather complicated so we omit discussion of this. Just think of the solution as the „best“ of a set of different solutions to the problem.

Ridge and Lasso (1/2)

As mentioned earlier, OLS is prone to the curse of dimensionality. There are two ways to account for the problem of „too much information“. They are called Ridge and Lasso estimators. In both cases the model assumption does not change (hence we still assume a linear model). The change is given by the loss or error functions. For the Ridge estimator it is given as:

$$\min_{\hat{\beta}_n \in \mathbb{R}^n} \mathcal{R}\left(\hat{\beta}_n\right) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{\beta}_{in} x_i\right)^2 + \lambda \sum_{i=1}^{d} \hat{\beta}_{in}^2.$$

The minimizing estimator (thus the solution) for the weight vector $\hat{\beta}_n$ is given as:

$$\hat{\beta}_n = \left(X^t X + \lambda I_n\right)^{-1} X^t Y.$$

Ridge and Lasso (2/2)

For the Lasso estimator the empirical loss loses the squares in the second part of the sum and reduces to:

$$\min_{\hat{\beta}_n \in \mathbb{R}^n} \mathcal{R}\left(\hat{\beta}_n\right) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{\beta}_{in} x_i\right)^2 + \lambda \sum_{i=1}^{d} |\hat{\beta}_{in}|.$$

Although there exists a closed form solution (hence a plain formula as for the other cases) we will not go into detail on it.

This is a good point to introduce you to another concept of ML, namely so called hyper paramters.

Hyper Parameters

You may have already noticed the parameter λ. Notice, that lambda is not part of the model assumption but the empirical loss or error. This implies that we can neither estimate nor learn the parameter given data. We could call this parameter exogenous or in Machine Learing speak a hyper parameter. [1]

There are ways to find optimal[2] hyper parameters, which leads to yet another important concept, called cross validation and validation data sets. [3]

For Ridge and Lasso, λ is called a penalty term. Without going into detail:

the greater the value of λ the more weight parameters in the linear model are set to zero. This implies that the matrices in the linear models are „sparse“ hence there are fewer weight parameters for which to solve. This in turn reduces runtime of the algorithms immensly. [4]

1 Hyper-parameters need not necessarily be part of the loss function. They can be exogenous parameters in the model assumption too.

2 Optimal in this case can mean different things (mathematically and practically). Usually, it is in the scope of the Data Scientist to define what conditions a hyper-parameter has to achieve.

3 More on this later.

4 Introducing the penalty term has not to be done just out of runtime concerns. It can be introduced to control for the number of relevant parameters in a computational feasible model too.

Classification – Introduction (1/3)

Recall that in contrast to regression, classifiaction is tasked with the problem to make inference between pairs of data $(X_i, Y_i)\, i = 1, \ldots, n,$ where $Y_i \in \{1, \ldots, K\}$. Unfortunately , this complicates the mathematical theory of modelling and finding solutions to the problem. In general, discrete (when values can not be plotted as a line, as known from functions in school) problems are harder to handle than continuous (values can be plotted as a line, as known from functions in school) problems.

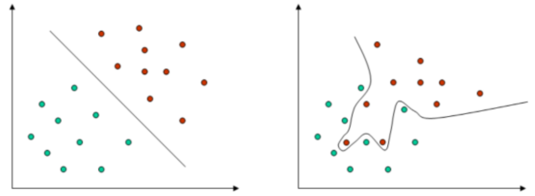
This means that we will have to take a „higher-level“ approach to classification in the remainder.

Classification – Introduction (2/3)

There are several technical approaches one can take to introduce classification problems. One that fits our purpose are decision regions. You can understand a decision region as:

A way to separate points in a space (not necessarily in three dimensions) such that you can find planes separating different data points from each other. Each „sub-space“ is then identified with one of the classes $Y_i \in \{1, \ldots, K\}$.

In two dimensions such a „plane“ would be a line which separates „clouds of points“.



Classification – Introduction (3/3)

Above picture shows a crucial difference one can make. Whereas in the first picture the two decision regions (class 1 red, class 2 green) are separated by a straight line, i.e. a linear function, the decision regions in the second picture are separated by an (arbitrary) looking function in two dimensions.

We have to keep this difference in mind when working on classification problems. Nevertheless, there are the so called Support Vector Machines which techincally are a linear model but are highly flexible when working with non „linear separable“ data. Thus, we want to focus on a few standard and linear Machine Learning algorithms for calssification.

Linear and Quadratic Discriminant Analysis (1/2)

Two of the most basic models for linear classification are Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA).

We have to introduce the concept of conditional probabilities to understand classification.

Assume the question: How probable is it, that my data belongs to a certain class $k = 1, \ldots, K$ given the data I collected? Mathematically this can be written as (where $\mathbb{P}$ assigns probabilities): $\mathbb{P}\left(Y_i = k | X_i = x\right).$

Now, of course we want to find the most probable class $k = 1, \ldots, K$ our data belongs to. This implies the problem we have to solve:

$$max_{k \in \{1, \ldots, K\}} \mathbb{P}\left(Y_i = k | X_i = x\right).$$

Linear and Quadratic Discriminant Analysis (2/2)

There are two cases. The first being that we assume a probability distribution which our aforementioned $\mathbb{P}$ follows. And the second being that we can or will not assume a distribution for $\mathbb{P}$.

LDA and QDA belong to the former. In both cases one assumes that the data is distributed according to a Gaussian (Normal distribution). This implies that they have a (multidimensional) means and (multidimensional) covarianve matrices.

In the case of LDA we assume that classes have different means but identical covariance matrices between classes. In the case of QDA we assume that classes have different means and different covariance matrices. LDA has linear separated decision regions and QDA has quadratically separated decision regions.[1]

1 What we mean by that is the following. Decision regions in itself can be functions of the space in which our data is found. So a linear plane separating two classes would be a linear function $y = mx + b$. In the case of QDA the decision region is a function in which the highest exponent is x^2. In fact, recalling from school, in the case of QDA the function separating the decision regions is a polynomial with second degree. Furthermore, as the model assumption of a Gaussian is rather simple, there are in fact closed forms for both "separating functions". One can rewrite the mathematical technicalities of classification in terms of discriminant functions. With them we can rewrite our problem of maximizing a probability into maximizing a function. Although this topic would be too extensive for this course, I would encourage you to take a closer look at this approach to decision regions, in contrast to maximizing probabilities by the often used maximum likelihood estimation.

Logistic Regression (1/4) – Abstract Introduction

Although the name suggests otherwise, logistic regression is an approach to solve classification problems. In contrast to LDA and QDA logistic regression does not necessitate a probability distribution assumption for $\mathbb{P}$ which was a Gaussian in LDA and QDA. The logic behind Logistic Regression is to find decision functions for each of the possible classes. Then, after the training procudere, a new datapoint X is plugged into each decision function and is assigned to the class for which the decision function has the greatest value.[1] Thus, while LDA and QDA find classes by litterally finding regions in space to separate the data into classes, Logistic Regression uses functions to rank the probability for a new data point belonging to a certain class.

1 You can understand a decision function as a mapping where a new data point gets assigned a probability for each class. Thus there are as many decision functions as classes.

Logistic Regression (2/4)

There are versions of LR which use linear decision functions and non-linear decision functions. Technically they are not that much of a difference so we will focus on the linear version. In the linear case we assume that it holds[1]:

For each $k = 1, \ldots, K$ there exists a weight vector $\beta^{*(k)}$ such that[2]:

$$\log\left(\frac{\mathbb{P}\left(Y = k | X = x\right)}{\mathbb{P}\left(Y = K | X = x\right)}\right) = x^t \beta^{*(k)}.$$

Under this assumption we can calculate probabilities for the classes directly:

$$\mathbb{P}\left(Y = k | X = x\right) = \frac{\exp\left(x^t \beta^{*(k)}\right)}{1 + \sum_{j=1}^{K-1} \exp\left(x^t \beta^{*(k)}\right)}, k = 1, \ldots, K-1,$$

$$\mathbb{P}\left(Y = k | X = x\right) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp\left(x^t \beta^{*(k)}\right)}.$$

1 This in fact is not a given fact. Technically we would start with decision regions in the case of not assuming a distribution for $\mathbb{P}$ and would simplify the problem by assuming that decision regions can be described by linear discriminant functions. Unfortunately, this would get rather complicated so we want to take a few steps ahead and introduce the model assumption for linear logistic regression.

2 Why we use the logarithm of these two specific conditional probabilities is beyond the scope of this presentation. The following gives an intuition: fracture is in the value range between 0 and 1. The nominator giving the probability of class k given the data x. And the denominator giving the probability of class K given the data X. The denominator is used to "normalize" the probabilities. To make "things work" such that we can have a simple vector multiplication on the right hand side we have to transform the interval 0, 1 into the real numbers (making it continuous), thus the logarithm.

Logistic Regression (3/4)

You might have already noticed that we do not know the true weights $\beta^{*(k)}$ for each class. Hence, we have to estimate them given our data. At this point Machine Learning is introduced. Estimation of these parameters is not possible in a closed form, implying that we can not calculate the probabilities in a closed form to get a ranking of possible classes for a new data point X .

Nevertheless, using the logic from OLS, Ridge and Lasso we can define an $\beta^{*(k)}, k = 1, \dots, K.$ error or loss function[1] for estimators $\hat{\beta}^{(k)}$ for the true weights

Then the weights are optimized[2] such that this error is minimized and we get our decision functions in the form of[3]:

$$\delta_k(x) = \begin{cases} x^t \hat{\beta}^{(k)}, & k = 1, \dots, K-1 \\ 0, & k = K. \end{cases}$$

1 Unfortunately it is not feasible to introduce this loss at this point as it would be relatively technical. **But if you would reduce the number of classes to two, the general loss function reduces to the cross entropy loss you already saw in Bernhard's talk.**
2 This is the training process. Given observed data an algorithm (in this case based on maximum likelihood estimation tries to find the optimal weights by minimzing the loss function.
3 This comes from the fact that aforementioned formulas to calculate the probabilities are functions of $\delta_k(x)$ given by $\mathbb{P}(Y = k | X = x) = f\left(\delta_k(x) = x^t \beta^{*(k)}\right)$.

Logistic Regression (4/4)

The last step would be to plug in the new data point X into all decision functions

$$\delta_k(X) = \begin{cases} X^t \hat{\beta}^{(k)}, & k = 1, \ldots, K-1 \\ 0, & k = K \end{cases}$$

and chose the class for X given by the rule:

$$Y = \max_{k \in \{1, \ldots, K\}} \delta_k(X).$$

As you can see there is a lot of (ommitted) machinery behind one of the simpler classification algorithms. I encourge you to read on about LR, especially how the estimators for the weights $\hat{\beta}^{(k)}$ are calculated.[1]

1 Please do not get demotivated by this technical presentation. Everything so far is already implemented in numerous open source programming libraries. Although it is necessary to understand the basic concepts of the theory, in practice it is more about intuition combined with experience and some mathematical, theoretical knowledge. Compare to the introductory slides how companies do data science.

k-Nearest-Neighbors (kNN)

There is another easy to understand but often useful classification algorithm. k-Nearest-Neighbors[1] does not assume any probability distribution as it does some simple geometry.

The idea is quite simple:

for every new data point X calculate its distance[2] from every data point given by the training data already collected. Then assign the class to X which has the most counts across the k neighbors with the least distance to X. This is called a majority vote. For example, if k = 3 you would calculate the distance from X

to all points of the training set, take the 3 closest to X and count the occurences of possible classes. Hence if two neighbors are of class 1 and one neighbor is of class 2 you would assign class 1 to X .[3]

1 At first it might be confusing that the k is not the index of the classes as in $k = 1, \dots, K$ but the (hyper) parameter how many neighbours should be taken into account for the so-called majority vote.
2 Although mathematics has more than enough possible distances, usually the Euclidian distance is used. Hence the "normal" and intuitive distance between two points Wikipedia. **One more addition: usually it is rather helpful to normalize the data, which you have already done in the previous session.**
3 How would you decide if the three nearest neighbors are from different classes? Thus all three classes are equally probable classes for the new datapoint X ? Think of hyper-parameter tuning.

k-Nearest-Neighbors (kNN) - Implementation

Of course it would be interesting to programm every presented model and their optimization algorithms (which we ommitted, frankly). The problem with this is the sheer amount of work you would have to do to implement something like a feasible usable Logistic Regression. Thus, in reality everyone uses open source[1] libraries to solve their ML or Data Science problems.

But we would like you to try it:

kNN is a really simple algorithm which you could and should try to implement for the forest data set.[2]

1 And to be honest, how could one person compare to a whole armada of people working on the de facto ML libraries for python like sklearn, pytorch or tensorflow (which is a Google project)?
2 This would mean: you take your dataset you already prepared in the last session, and try to implement the majority vote logic for the test-subset of the original dataset. And because you already know the true class of the test data you can even calculate the accuracy of the algorithm (for example by counting the misclassified test datapoints divided by the number of all datapoints in the test set).

Support Vector Machines – SVM (1/2)

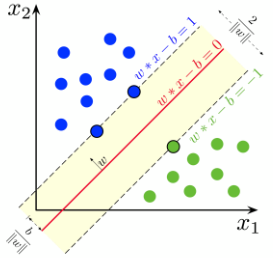
Support Vector Machines are the most advanced[1] models we want to take a look at. They are highly flexible and have (under certain circumstances) good runtimes.

Firstly, they are highly flexible because they allow not only for linear decision boundaries but more or less arbitrary ones.[2] This implies that the decision regions can be rather arbitrary which again implies highly flexible classification possibilities.

1 The mathematical formulation would be too extensive without any knowledge of convex optimization, thus is omitted.
2 This is a rather loose formulation. But we want to keep it at that. This is due to the fact that SVMs are able to work with so called kernel functions which try to map a non-linear decision region onto a linear decision region. This in turn implies that the solutions are easier to find. Unfortunately, this linearization implies that the number of possible parameters increases. In some cases they even double. Thus, SVMs can be prone to the curse of dimensionality. As always in ML there are tradeoffs which the user has to know and learn from experience.

Support Vector Machines – SVM (2/2)

Secondly, they are highly flexible because they allow for errors in their training process.[1] This is due to the fact that their decision regions are not only one line but three, in fact. This „street“[2] is always visualized in presentations of SVM results as three parallel lines. The best main-line (red) is the one maximzing the distance to all data points simultainously.



At last, they can be reformulated as a convexe optimization problem. The theory behind these problems is a highly advanced mathematical field. Thus there exist optimal and fast solutions in many cases.

1 This is due to the fact that they allow for a so called slack variable which allows some margin of error in the optimization process. Slackvariables (Schlupfvariablen in German) are a way to allow variance in an equation.
2 Misclassifications within this "street" are weighted less in the calculation of the error term (but they are not excluded). Here the slack variables are the margin of error which is allowed.

Classification if K is greater 2

Usually, if the Machine Learning algorithm does classification by building decision regions it is only usable if there are only two classes. (In contrast, kNN which you should try to implement yourself does not have this problem.)

There are several ways to accomodate this problem.

<u>One vs. Rest:</u>

In this case you would split your data by class and train your algorithm on this set only.[1] Then in the prediction process you would use the model for each class and use the classification with the highest probability (in analogy to decision functions and their rankings in Logistic Regression).

1 Thus leaving you with K models and K train, test processes.

Classification if K is greater 2

One vs. one:

In this version you do not classify between one class and the rest of the classes but between each and every class. This means that a classifier for a class k would consist of k-1 models. Where you classify between class k and all classes that are not the class k. This would leave us with $\frac{1}{2}k(k-1)$ models and train, test processes. Then again, you would choose the class with the highest probability over all predictions given a new data point. The aforementioned SVMs are usually classifying by using this strategy.

Again, above two approaches are implemented in every standard Machine Learning library. You do not have to do this yourselves. Nevertheless you should understand the approaches on a high level.[2]

1 Thus leaving you with K models and K train, test processes.
2 Try to make yourself comfortable with one of the most used python libraries sklean or scikit-learn for everything presented in this presentation.