# Machine Learning

The field of Machine Learning contains concepts how computers can obtain information without explicitly programming this kind of information retrieval. These concepts of “Learning” can be divided in three main categories: Supervised learning, Unsupervised learning and Reinforcement Learning.

Supervised learning always deals with a user that ‘feeds’ input to the program as well as desired output. The program should recognize patterns that lead from the given input parameters to the desired output.

Unsupervised learning instead, is an approach where the program does have an input and needs to find a structure in those data. The finding of some pattern can be the goal of the program itself.

Using reinforcement learning, every output of the program is being valued by the user again. Output that has been found correctly will be strengthened, whereas incorrect values will act repulsive on the algorithm. After multiple iterations of this process, the program can find the best answer (but not always the correct answer) using the attracting and repulsive values.

Our goal is to use one machine learning technique in order to improve the outcome of our application. One major objective that can be addressed with Machine Learning is the relation between accounting record positions and how they are assigned to all the accounts that are important for this position.  
We identify two major problems regarding this classification:

1. What does a position represent?
2. Which accounts should be assigned to this position?

As we are processing an invoice, we will retrieve a position as a String. An accountant would be able to identify the position (which means a semantic identification of the object) and assign it to the accounts that are important in this matter. But, as there is no concrete rule which position belongs to which accounts, every company can apply this position to different accounts.

For instance, the maintenance of a car in the car pool of a company could be booked as car costs, or (if the company defines it more specifically) as maintenance, car parts and worker time.

Hence we need an algorithm that is capable of the following:

1. Assign involved accounts depending on the user (-> allow different account structure)
2. Learn relationships between a string and a set of accounts

While the algorithm should be able to deal with those problems, we will have another problem to deal with: OCR errors (e.g. “CAB” instead of “CAR”)[[1]](#footnote-1) and similar words (e.g. plural words such as “apples” instead of “apple”).

Keeping those constraints in mind, we can start thinking about a Machine Learning technique that satisfies our goal or at least helps us to reach it.

%To narrow our search, we also have to think about automation. As this application %should be able to reduce the time an accountant needs to process an invoice, we %want to make this process as automatically as possible. Using a supervised machine %learning method would lead to an application, that requires to validate each invoice %every time. Hence supervised machine learning algorithms will not be considered here.

## An abstract approach on accounting records

Before we can compare different Machine Learning algorithms we have to think about the model that is used. On one hand, there is the position value which can be seen as a single String. On the other hand, we have 1195 Accounts (as proposed in the SK03 account system) that can be involved in this accounting process. Between those accounts, we also have to divide between accounts for debit and credit.

Evaluating each account per position would need two iterations: First if the account is relevant for this string and second if it is involved as a credit or debit account.

We come up with a different approach. As we see the position abstract as a string, we see the combination of accounts as a combined structure. This way we do not only reduce the iterations to one, but also enable a 1:1 relation.

For instance, given two accounts a1 and a2 we would have two possible structures: s1: {a1|a2} and s2: {a2|a1}[[2]](#footnote-2). One time a1 is related to the credit side, one time to the debit side and vice-versa for a2. Given a position p1 there are now two possible structures we can assign p1 to.

The downside of this mapping from 1:N to 1:1 relations is the increasing number of possible solutions. But the advantage of it is the flexibility to assign a position to a known structure again after the user has defined how they want to it to be accounted.

## Possible machine learning algorithms

We now know about how our model looks like. We also need an accountant initially to define how the position should be accounted. After these information have been given, the algorithm is able to assign a similar position to the same structure of debit and credit accounts. This means that we will look for an algorithm in the field of supervised learning.

There are several well-known algorithms in this field. For instance, Artificial Neural Networks (ANN), the K-Nearest-Neighbour Algorithm (KNN) or Decision Trees. To select the appropriate algorithm for our problem, we will be using a randomly generated trainingset consisting of a combination between 30 positions and 9 different structures in 1920 cases. We will evaluate the performance and accuracy of some of these algorithms to find the one that suits the most by using a testset of 2000 positions to be tested. To do so, we will be using an open-source software (RapidMiner Studio) that enables us to switch between those algorithms and evaluate the accuracy before implementing them.

The following table shows the algorithms and the accuracy as well as the time needed to evaluate the result:

|  |  |  |
| --- | --- | --- |
| Algorithm | **Accuracy** | **Duration** |
| K-Nearest-Neighbour | 75,35% | <1s |
| Artifical Neural Network | 73,61% | 30s |
| Decision Trees | 72,92% | <1s |
| Naïve Bayes | 72,92% | <1s |
| Random Forest | 55,03% | <1s |

Except the Random Forest algorithm, there is not much of a difference between the algorithms. While Decision Trees and Naïve Bayes result in the same accuracy, ANN are slightly more accurate. The disadvantage of this algorithm is its duration that increases exponentially by the amount of data. The data that we used resulted in a duration of around 30 seconds execution time. The K-Nearest Neighbour algorithm instead, while still as fast as the other algorithms, also results in a higher accuracy.

We will now introduce the remaining three algorithms and take a closer look on the results regarding the data set that we used. After that, we will decide which algorithm we want to use in our application.

## The K-Nearest-Neighbour algorithm

The KNN algorithm tries to classify an object by using *k* neighbours of the object. Each of the *k* neighbours already have a class ci which enables the calculation of a likelihood value for the unclassified object.

If we would choose k = 1 then the object would be given the same class then the closest neighbour. But this can lead to wrong assumptions since only neighbour has been taken into account. Hence we would need to select a value for k > 1. The given accuracy from Table (TODO: REF) has been achieved with k = 5.

When further investigating in the results provided by the algorithm we found something we did not expect. In our trainingsdata we defined a position p1 and two different structures s1 and s2. n times p1 has been classified to s1, and n times to s2. This means, p1 is equally distributed between those two classes. But the KNN algorithm resulted in a classification of s1 = 60% and s2 = 40%. This can be explained by the chosen value for k. As we defined k = 5, there were 5 neighbours, 2/5 that belonged to s2 and 3/5 that belonged to s1. And indeed, as we changed k to an even value (k = 6) we had the expected classification of 50% for s1 and 50% for s2.

This should be taken into consideration when using this algorithm. As we do not know how much different classes exist and the amount of classes can increase by the user assigning positions to new structures, we could have wrong classification values.

## Decision Trees

Decision Trees are a simple and still effective way of classify an object to different classes. Usually the object that should be classified contains several attributes and each of them will be taken into consideration iteratively.

We want to explain the behaviour of decision trees on the following example:

A telecommunication company had an increasing amount of customer loss recently. To find out the reasons behind that and to find out which actions to take to get new customers again, they build up a tree with the data they have of their customers:

%TODO IMPLEMENT IMAGE

The companies offers three packages to their customers: Data, Voice and Data & Voice. The data package has a high amount of male customers that left, whereas the most of the female customers stayed with the companies offer. The Voice package shows a difference between male customers that have been charged over 80 monetary units and the customers with a maximum of 80 monetary units.  
Hence it is very likely that reducing the monthly rate for this package will result in a higher amount of customers staying with the offer of the company.

Using this decision tree, it is possible to split between objects even further, using different attributes. In our case, we only have the position as an attribute. If we would apply a decision tree on this problem, it would result in a tree with a depth of 1. This way the actual idea behind the decision tree is not used. The results would still be valid though.

## Naïve Bayes

Naïve Bayes is a simple probabilistic classifier that calculates the probability that an object belongs to a class by taking each attribute of the object and comparing it with the probability of this attribute in the given class.

What this means for our case is the following: As we do not have any additional information on the position, the only attribute there is, is the string itself. This attribute is compared with the already classified positions. The result of this calculation will basically assign the position p to the class that has the most positions that are similar to p.

In addition to that, a way to improve the comparison is to use a numerical value that represents the similarity between the position p and another position that p is compared with. This can be done using the Levenshtein distance. The distance value represents the amount of changes needed to transform one position to the other. Using a relative value, we can make this result relative by the size of the position string:

%%%%For instance, let us assume we have two different structure s1 and s2 , a position we want to check pc and five other positions p1…p5 that have been processed before. The positions p1 and p2 are relted to structure s1, p3 … p5 to structure s2.

When calculating the distance, we would find the following values between pc and pn:

P1: 0,33, P2: 0,65, P3: 0,73, P4: 0,23, P5: 0,45

1. We used uppercase letters here to make the possibility of OCR errors between those two words more easily understandable. [↑](#footnote-ref-1)
2. Each structure will be written the following way: The curly braces mark beginning and end of the structure, the pipe divides between credit and debit accounts. [↑](#footnote-ref-2)