Individual Project Statistical & Machine learning

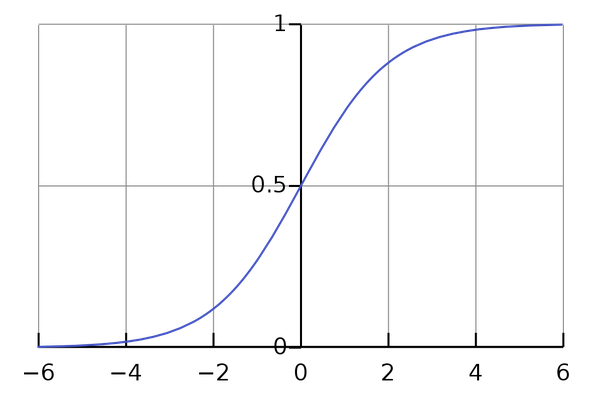
Edward Vrijghem

**Part 1: Description of the Algorithms**

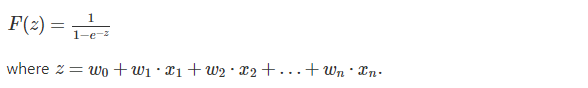
1. **Logistic Regression**

Logistic regression is a statistical technique which is used to predict the probability of a binary outcome. It predicts chances of something happening (1), or not happening (0), a very common example is prediction of customers churning or not. In the individual Kaggle we predicted whether a client had subscribed for a term deposit (1) or not (0), this based on a selection of variables.

How does it work: The probabilities which are the outcome of logistic regression can be estimated using a sigmoid function, which is also called a logistic function (see example below). As you can see, the sigmoid function always takes on a value between 0 and 1, it’s used to map predictions to probabilities.



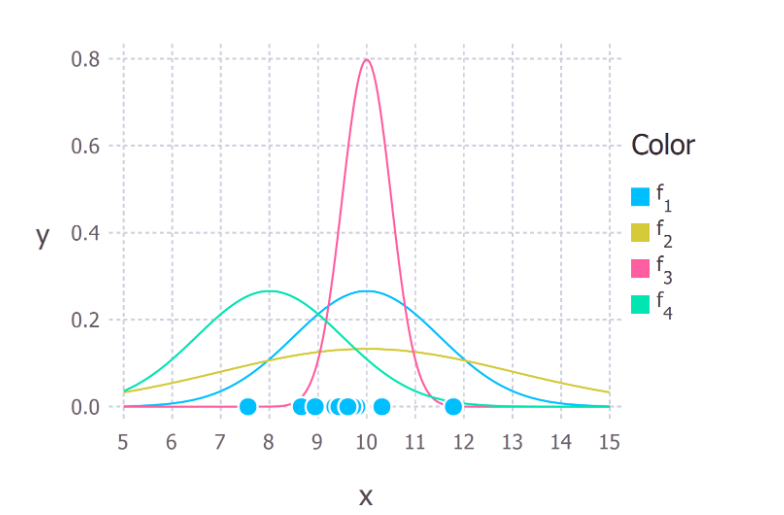
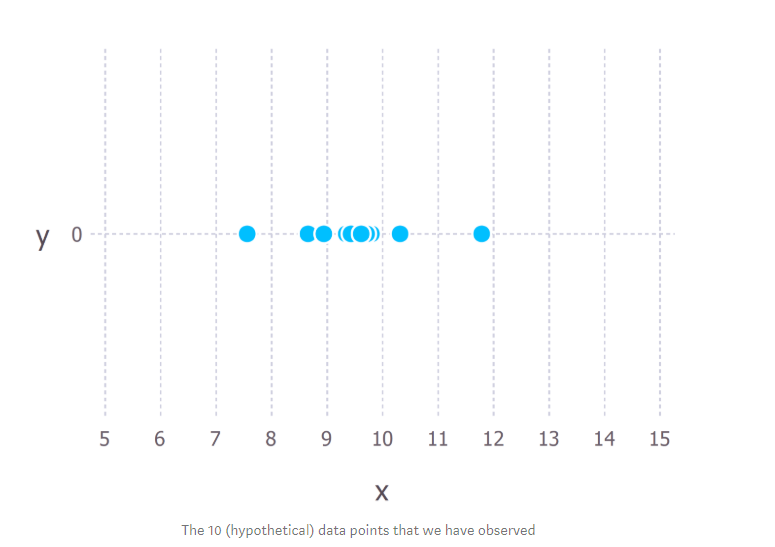
The mathematical expression is given by:



Now say hypothetically we have the following function f(x) = 0.30\*X1 – 0.09\*X2+0.001\*X3, with f(x) being the probability of dying to corona. The X values are:

X1= got diagnosed dummy, X2 = consecutive kilometers one can run, X3 = age.

The w values: These are also called the beta values. In my example, these were determined at random. We see that being diagnosed with the virus has a bigger positive influence on dying of the virus than being one year older. We see that betas can also be negative, in this example being able to run more kilometers, will decrease the probability of dying to corona. In the Kaggle example these parameters were determined by a Maximum likelihood estimation. By selecting the parameters through MLE you guarantee the likelihood that an observation produced by the model has actually happened. A quick and easy example of MLE (by towards data science) is given here:



Say we are given 10 data points and we want to select a function which is most likely to predict these points. Here we see that the data is centered on 9.5 and I assume it can be described by a normal distribution. Now on the right we see 4 different normally distributed models, the MLE selects the model which is the most likely to have predicted the different observations. On the left you can see the probability (y) the model has to correctly predict the observations. We see that on average model f1 will have the highest chance, compared to the other models, to correctly predict the observations. This will be the model used. If we now extrapolate this to an actual logistic regression, the 10 observations will be the outcomes of the target variable provided by the training data and the functions will be different combinations of beta values.

Futher building on our example function (f(x) = 0.30\*X1 – 0.009\*X2+0.001\*X3):

Say we put a threshold of 0.5, which means every probability lower than 0.5 will be classified as 0 (expected to survive), higher means prediction of 1 (expected death).

Say we have two patients coming in:

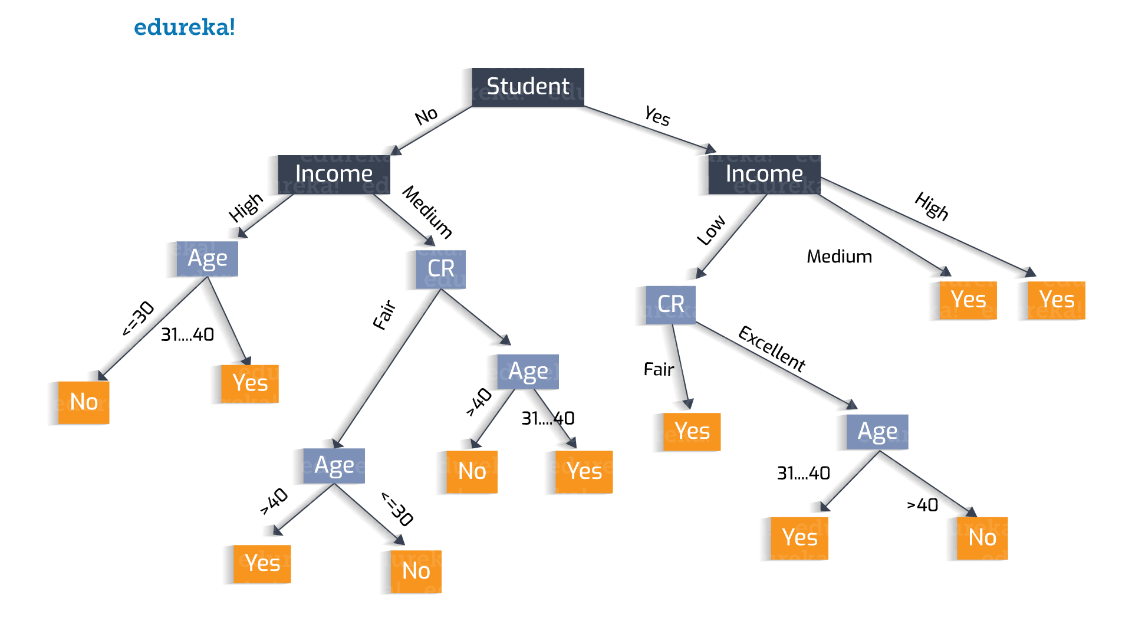
1. One is 30 years old; he got diagnosed with corona; is in very good shape (can consecutively run 25 km). The probability of him dying to the virus is 0.105. The model predicts 0, which seems reasonably, the patient is young and in very good condition.
2. The other patient that came in is 95 years old; he was diagnosed corona-free; his age does not allow him to run. The probability of him dying to corona is 0.095; Based on the threshold value, the model also predicts a 0, which is also very reasonable since he doesn’t have the disease.

The advantages of logistic regression is that it is a very intuitive algorithm, it is easy to understand what is going on and easy to implement, easy to train and interpret. It is also less prone to over fitting compared to other more sophisticated algorithms.

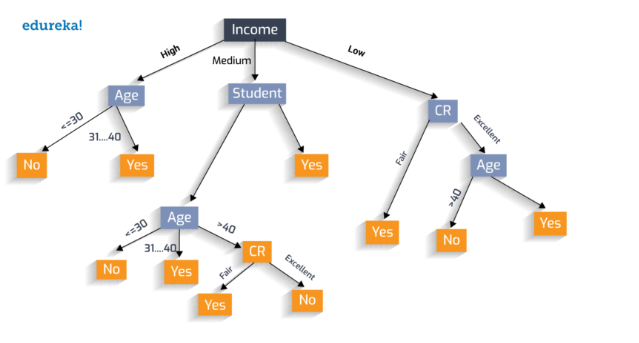
A disadvantage could be that it assumes linearity between dependent and independent variables. In the real world this not often the case. Another disadvantage would be that is can only be used to predict discrete variables.

1. **Decision trees**

The decision three algorithm is probably the easiest to understand, it is a flowchart. You start with the full set of observations, and split them based on decisions. The following examples are decision trees that determine if a computer is bought or not. The first step when creating a decision tree is defining what will be the starting variable, in our case student. This is determined by entropy and information gain. Entropy being the homogeneity of the tree, if the data is completely homogenous it is 0. Otherwise if the data is completely divided entropy is 1. Information gain is the decrease or increase in entropy after splitting a node. Say you have population of male and female, splitting based on gender will greatly increase the information gain since it the two newly created groups will be highly heterogeneous (entropy close to 1). There is no in between value for gender.



Depending on the data this could also be a feasible option for a decision tree:



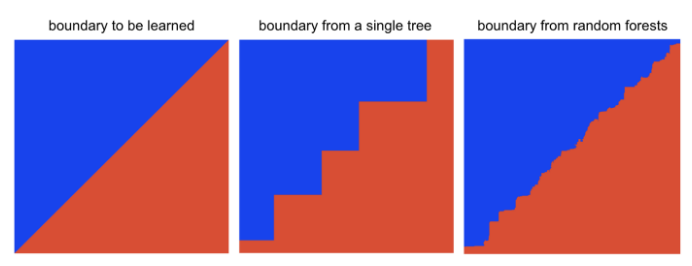
Even though in this example the classification is binary, a decision tree can perfectly be used for multiclass problems. As you see the number of nodes, (9 and 6 in the above trees) can be increased or decreased to almost perfectly classify all the observations. In theory if you have enough variables and heterogeneous observations, you could reach almost a 100% accuracy on the training set just by adding nodes and layers. This means they are easy to overfit. Other disadvantages are that they are unstable; a small change in the data could lead to a very different tree. They are also considered inaccurate, which means other algorithms are likely to outperform the decision tree algorithm if given the same dataset and task. This is why these days the random forest algorithm is a more feasible choice in for predictive modelling (see next).   
The advantages of decision trees are somewhat similar to logistic regression, it is very simple to understand the concept and easy to combine with other decision techniques. Another advantage is that even with a small dataset it can produce valuable insights.

1. **Random Forest**

A random forest algorithm is basically a group of multiple single trees based on a random sample of the used training data. It feeds the data into every single tree and takes the majority vote as output. Using the majority vote the tendency of a single decision tree to overfit is corrected. Different trees are trained on different parts of the dataset which will reduce the variance and thus reduce the chances of the model overfitting. Next to the number of nodes and layers of every individual tree the number of trees used in the model now also plays a role. In general the number of trees won’t play too big of a role since random forests use bagging and the random subspace method.

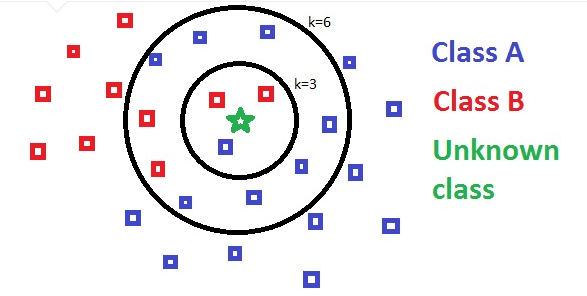
Bagging is picking a subsample of your training set to build a tree on, using the random subspace method means that it only will use a subset of the features (attribute bagging). Using these methods the number of trees is does not play a huge role. It is of course possible to have too little trees (features and/ or observations left unused).

The main advantages of the random forest approach is that it can both handle classification and regression problems. Because of the bagging method it can handle data sets with a very high number of variables. Also it is possible to train models with a relatively small number of samples and have pretty good results. The problem here is that the saturation point is quickly reached, the point where adding more samples stops increasing accuracy. Even though you can use random forests for regression problems it tends to perform poorly since it doesn’t give the precise continuious predictions. Also, it is considered to be quite black box since you have little control over the model. At most you can set different parameters and random seeds. As you can see in the following example, the result of using multiple trees is that you get a much smoother decision boundary. This means that it, instead of making a decision based on wrong or right, is more prone to nuance.

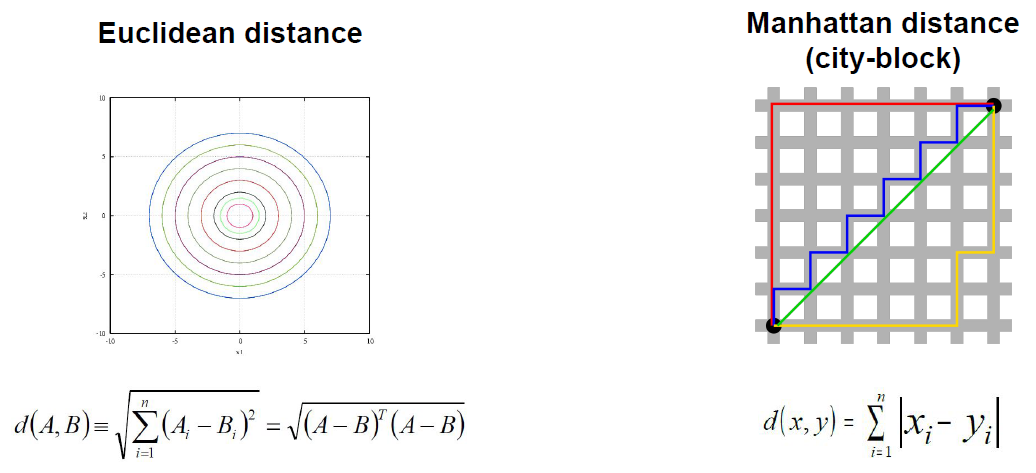


1. **K-Nearest Neighbours**

The general idea behind the KNN algorithm is that an observation would take the value that the majority of its K-neighbors have. It can be used for both classification and regression problems. Now how does it work: Imagine a wide multidimensional space filled with observations, positioned according to the different values their dimensions take. Similar observations, have similar variable/dimension values and are thus assumed to be positioned very close to each other. This is one of the main foundations of the KNN algorithm. Now you may ask why the K? K means the number of neighbors the algorithm takes into account. The observation itself is labeled as the majority of its neighbors. In the following example the green star will be labeled as red square for K =3 (2 of the 3 closest neighbors are RS). If we up K to 6 the algorithm will classify the green star as a blue square.



Next to the number of neighbors K, the distance metric used in the KNN algorithm is of major importance. As for distance, the two most commonly used metrics are the Manhattan or city-block distance, which is the sum of the absolute differences between units for each variable, and the Euclidean Distance, which is the ordinary straight-line distance.

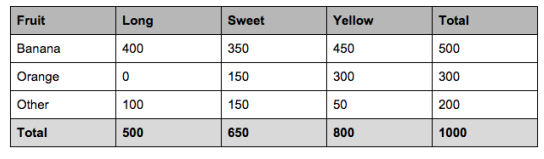


The strengths of the KNN algorithm are that it is a so-called lazy learner, it does not have a training phase, this allows for much quicker predictions than other algorithms, which allow the user a first quick glance. Another strength is that the logic behind it is simple, intuitive, and easy to use (only two parameters). It is also easy to update, new data will just be added to the so-called “multidimensional space” and will not affect the accuracy of the algorithm.

The weaknesses is that a big and small k’s lead to under and overfitting, and the setting of the k’s is not an exact science. There is also a considerable chance that there will be close data points that could outvote the correct data points in some region of the multidimensional space. Other than this it is sensitive to outliers. It also tends to be computationally expensive (when facing high dimensionality >10), and suffer from the effects of the ‘curse of dimensionality’ this is why for most KNN algorithms the dimension is reduced.

1. Naive Bayes classifier for discrete predictors

Naïve bayes is a classifier based on the Bayes Theorem. I will illustrate how it works with the following textbook example; Say we have data on fruits and we want to know based on the three attributes of every fruit what type of fruit it is most likely to be.

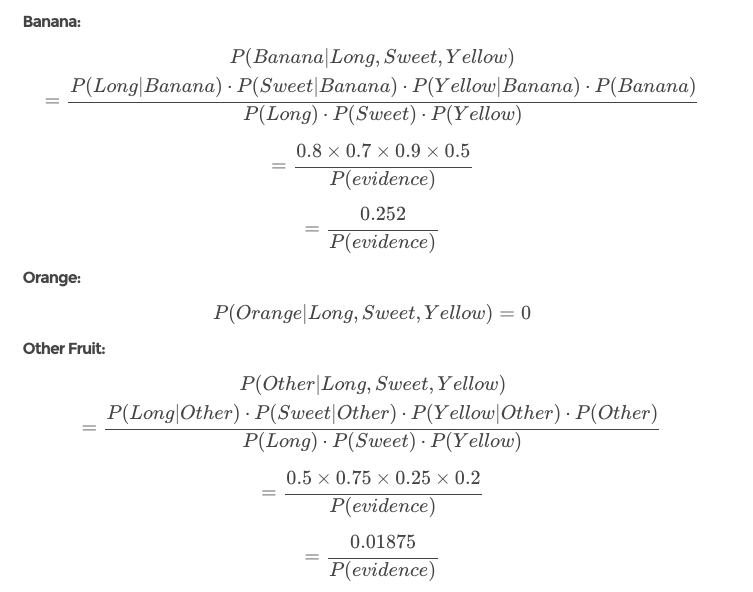


From just the number of fruits we see that the probability of a fruit being a banana is 50%, 30% are oranges and there is a 20% probability of the fruit being something else.

The second thing we see that from the 500 bananas, 400 of them are long (80%), 350 of them are sweet (70%) and 450 are yellow (90%). This prior information enables us to predict the odds of a new fruit being labelled ‘banana’ based on its length, taste and color. The same will be done for the other fruits.

The highest probability will be the label the new fruit gets. Keep in mind that this is a very limited representation with only three fruits and three attributes.

Say the new fruit is long, sweet and yellow; the model will predict three probabilities:



As you can see the probability of our fruit, (which had all the normal attributes a banana would have) is classified as a banana, the highest probability (0.01875 < 0.252).

The Naïve part of this algorithm refers to the fact that it makes the assumption that features of a measurement are independent of each other. This is naïve because in real life this is almost never true. Without this assumption the algorithm would only be able to classify exact duplicates, making it useless.

The great thing about Naïve Bayes is that again it is easy to understand and works very intuitive. You calculate the probabilities per class and pick the highest one to be the classification of choice, this makes it very fast. This makes it a desirable algorithm when speed is preferred over accuracy. It also performs well with high-dimensional data.

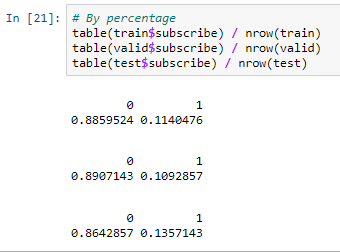
The major drawback is the naïve assumption; this is almost never deemed to be true. This makes the algorithm less accurate compared to other, more complicated, algorithms.

**Part 2: Description and insights in the R code**

1. **Data summary and variable creation**

After the installation of the packages used during our benchmark we start by summarizing the data. We start by loading in the data and checking the correctness of the variables, this means checking if the format is correct, if the variable represents what we expect it to represent and fix any other errors. In the given dataset the campaign included the last contact, we removed this by reducing the campaign column with 1.

After checking the data, a validation set and test set are created, in order to test the performance of our algorithms in a later stage. This is done by using the caret package (short for classification and regression training), it includes a simple function for data partition called ‘createDataPartition’ which simply splits the data while somewhat maintaining the distribution of the target variable. We first split the training data set into a train and test set, the training set containing 60 percent of the observations the test set 40. Then we split a second time, now splitting the test set into a test and validation set (both 20%). Since we used the caret package we somewhat maintain the general division of our target variable. This is nicely illustrated here:



After the division of the data into a train, test and validation set, the PIMP-algorithm (Permutation Variable Importance Measure) is used to check the importance of the different variables in the data. With the information generated by this, it was easier to see which variables are more important compared to others and should be used for feature engineering. This is done by using all the variables except client id and the DV in the x value, and the DV in the y value to be used in a random forest algorithm. The PIMP then evaluates the importance of every variable based on the outcome of the random forest. As you can see the most important variable is ‘month’, which will lay at the foundation of 4 of the newly created variables.

For feature engineering 6 new variables were created, 4 containing the season. Based on the month column. One indicating if the age of the client was above the average age (true/false) and one containing the numeric value “999”.

1. **Data processing**

The first step of data processing is the transformation of variables, categorical variables get remapped, continuous variables get discretized. Logically the first step is to split the categorical, Boolean and continuous variables into three different lists.

2.1 Binning

The first step is creating a list containing all the column names except for the DV and the clientid. After this the class of every column is checked using a loop, every column type is checked and the column is added to the correct list.

The first option used is decision tree based remapping, for the categorical variables a remapping is performed, this is basically grouping the different categories of a variable into groups. As you see in the example the categories of job get grouped into 3 groups. After the groups are defined we use the function woe.binning.deploy to apply the binning to the variable. Doing this for every categorical variable would be a tedious process so this is fed into a loop. In our example the “add.woe.or.dum.var” = woe, adds an additional variable containing the woe scores.

A similar process happens for the numerical variables, the groups of choices are now replaced by numeric intervals.

The second option used to bin numeric variables is equal frequency discretization, the third option is equal width discretization both of which make use of the dataPreparation package. For equal frequency discretization, the type is set to ‘equal\_freq’, for width its set to ‘equal\_width’. The difference, unsurprisingly, is that the former focusses on dividing the data in groups with the same frequencies while the latter focusses on dividing the data into groups with equal widths. The process is similar to decision trees, it first defines the groups as a discrete variables and then applies those to the different datasets. Both of these processes are also fed into a loop where the binning happens for all the numeric columns, for training, validation, test and holdout data.

* 1. Changing value representations

First we update our list of categorical, Boolean and numerical variables using the previously described code. Then the categorical variables get converted to dummies.

After this they get represented using a incidence of the target variable, this is the correlation between the category and the target variable, once again this process gets fed into a loop and is applied to all the categorical variables. Another way they are represented is using a weight-of-evidence conversion, the weight of evidence tells the predictive power of an independent variable in relation to the dependent variable.

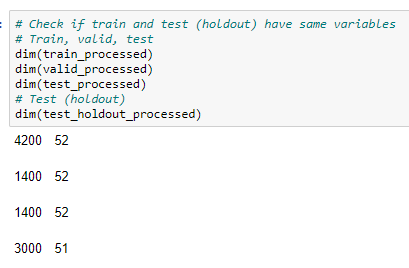
Age is also used to create two new variables, which are just transformations of age. One being the standardized age and one being the log transformation of age.

1. **Variable selection**

To select the variables we again first update our list of categorical, Boolean and numerical variables using the previously described code. Then the categorical variables get converted to dummies. After this infinite values are replaced by NA. Missing values are replaced by the mean and the categorical variables (which were processed earlier) are dropped, Boolean values are converted to integers and constant variables are dropped.

Once all these steps are done the fisher score is calculated and based on this score 50 variables are selected. The Fisher score is a way of measuring the amount of information that an observable random variable carries about an unknown parameter of a distribution that models the variable. It’s one of the most used metrics to select features. It describes the probability that we observe a given outcome of *X*, *given* a known value of *θ*. If *f* is sharply peaked with respect to changes in *θ*, it is easy to indicate the “correct” value of *θ* from the data. Variables with a very high fisher score will have a high chance of correctly predicting the DV.

After the variables to be used in the prediction models are selected, the number of observations are checked as well as the number of variables.



As you see, the number of observations is nicely distributed 60/20/20 and the number of variables are the same. Except for the holdout set, this one is of course missing the DV.

1. **Prediction methods and parameters**

**4.1 General**

We start by with the creation of a pipeline, by selecting cross validation, the desired model and the data and the parameters and their tuning. The second part checks if there needs to be parameter tuning or not. If there are no parameters to tune the code will just perform a cross validation and compute the AUC.

**4.2 No hyper parameter tuning**

Since logistic regression and Naïve Bayes does not require hyper parameter tuning, we just perform the model with 10 k cross validation.

**4.3. Hyper parameter tuning**

General:

We used three algorithms which required some level of hyperparameter tuning. The way this works is using the command getParamSet("classif.ctree") (example for decision tree), you get the possible parameters which you then can select out off. The way the MLR package works is that it selects a random value somewhere inside your specified range for your parameters and evaluates the AUC using the K-fold validation (k = 10) in my models. After computing the AUC over K-folds the model will generate an average AUC. The iterations are always set to 10 which means this process happens 10 times, after which an average is displayed.

Hyperparameters:

KNN required K, the number of neighbors to take into account. I calculated for the values between 50 and 1000 neighbors. A K higher than 3500 would lead definitely lead to underfitting, lower than a 100 might lead to overfitting.

For the decision tree I used the minbucket and minsplit parameters. The minbucket is “the minimum number of observations in any terminal node”. A bigger minbucket will mean the decision tree will have less branches and will thus be less prone to overfitting the training data. For the minbuckets end nodes, I picked values between 50 and 500, which seemed reasonable. (50 variables and only two classes). The minsplit is the minimum number of observations that must be in a node in order for a split to be attempted. Here I accepted a minimum value between 50 and 100, less than 50 would not have made much sense.

For the random forest, I picked a model with minimum 50 and maximum 500 trees, a nodesize between 10 and 50 and a mtry, which is the number of variables available for splitting at each tree node, between 10 and 50. These parameters make it that the random forest computation is very intensive.

**4.4. Comparison**

|  |  |  |
| --- | --- | --- |
|  | **Validation** | **Test** |
| **Logistic Regression** | **0.8066** | **0.8130** |
| **Decision Tree** | **0.7628** | **0.7662** |
| **K Nearest Neighbors** | **0.7309** | **0.6958** |
| **Naïve Bayes** | **0.7806** | **0.7929** |
| **Random forest** | **Around 78%** | **Around 78%** |

In the end, we see that logistic regression outperformed the other algorithms; the reason for there not being exact numbers for random forest is that I had some computer issues and the model did not run after 2 hours (that is why this assignment was handed in late.)

**Sources :**

Distance images of KNN were taken from a powerpoint used in the predictive modelling course.

Code for the predictive modelling was partly sourced from ieseg online.

<https://st3.ning.com/topology/rest/1.0/file/get/2808297821?profile=original>

<https://dzone.com/articles/how-to-create-a-perfect-decision-tree>

<https://www.quora.com/What-are-the-advantages-and-disadvantages-for-a-random-forest-algorithm>

<https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm>

<https://towardsdatascience.com/naive-bayes-classifier-explained-50f9723571ed>

<https://en.wikipedia.org/wiki/Decision_tree>

<https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/>

<https://www.datasciencecentral.com/profiles/blogs/naive-bayes-for-dummies-a-simple-explanation>

<https://towardsdatascience.com/why-random-forests-outperform-decision-trees-1b0f175a0b5>

<https://towardsdatascience.com/a-gentle-introduction-to-maximum-likelihood-estimation-9fbff27ea12f>

<https://towardsdatascience.com/whats-so-naive-about-naive-bayes-58166a6a9eba>

<https://stats.stackexchange.com/questions/376190/how-would-you-use-decision-trees-to-learn-to-predict-a-multiclass-problem-involv>

as well as stackoverflow.com to help me with code related doubts.