UNIVERSITÉ DU LUXEMBOURG

Stock Trading

Mathématiques Expérimentales
2020

Béatrice Bach
Anne Fisch

Contents

1 Linear Regression **5**

1.1 Gradient Descent 5

1.2 Ordinary least square 7

1.3 Advantages and Disadvantages 10

1.4 Some Extras 11

2 Random Forest **12**

2.1 Decision Tree - Example 12

2.2 Decision Tree 14

2.3 From Decision Tree to Random Forest 16

2.4 Random Forest 18

2.5 Advantages and Disadvantages 19

2.6 Some Extras 19

3 Examples **20**

3.1 Netflix 20

3.2 Bitcoin 22

4 References **24**

Abstract

The goal of this project is to get to know the basics of stock trading and to get in touch with different methods that try to predict the stock market. As a final product, we create programs written in Python that will complete our theoretical researches on a practical term.

Note that all of the graphs and schemes are made by ourselves unless something else is specifically indicated.

1 Linear Regression

The basic idea of simple linear regression is to analyse two separate variables in order to define a linear relationship between them. In fact we are searching for a regression line which minimizes the distance from itself to each point of the graph and thus provides a way to forecast trends. One of the variables (the one on the x-axis) is used to predict the data and is called the independent variable. The other one (on the y-axis) is called the dependent variable and is the one which is being predicted. Mathematically speaking, simple linear regression consist in finding a straight line which models the points of a given data the best. Such a regression line satisfies the following equation:

$$y = ax + b + \epsilon, \tag{1}$$

where y are the real values/observations, a is the slope of the regression line, b is the y-intercept and ϵ is the error term which we are trying to minimize. The sign of a depends on the type of relationship between x and y. The sign of a is positive if the y-values increase with the x-values, which is represented by a regression line that slopes upwards and is called a positive relationship. Whereas the sign of a is negative if the y-values decrease while the x-values increase, which is represented by a regression line that slopes downwards. There also may be no relationship if a= 0. In this case we would have a flat regression line.
Since we want to find the best matching regression line, we need to find the best values for a and b. To do so, there exist different regression models. Two of them are now explained in more detail.

1.1 Gradient Descent

In order to find the best values of a and b, we start by choosing a and b randomly and choose a good learning rate α. The method consists in minimizing the difference between the predicted values and the observed ones by updating the values of a and b. For this, we define a function, known as "‘cost function"’ or "‘mean squared error function"’, by squaring the difference between the predicted and the real values, summing them all together and dividing them though the number of summed values (The number of summed values n depends on the type of gradient descent we use):

$$J(a,b) = \frac{1}{n}\sum_{i=1}^{n}(ax_i + b - y_i)^2$$

In order to minimize this cost function, we calculate the gradients of it. So let's calculate the partial derivatives of J with respect to a and b:

$$\frac{\partial J}{\partial a} = \frac{1}{n}\sum_{i=1}^{n}2(ax_i + b - y_i)x_i = \frac{2}{n}\sum_{i=1}^{n}(ax_i + b - y_i)x_i$$

$$\frac{\partial J}{\partial b} = \frac{1}{n}\sum_{i=1}^{n} 2(ax_i + b - y_i) = \frac{2}{n}\sum_{i=1}^{n}(ax_i + b - y_i)$$

Now we have to update a and b by the gradient descent update rule, using the learning rate α:

$$a_{new} = a - \alpha\frac{\partial J}{\partial a}$$

$$b_{new} = b - \alpha\frac{\partial J}{\partial b}$$

Finally we need to rename $a = a_{new}$ and $b = b_{new}$. This procedure is being repeated until the minima is found or no further improvement is possible. (The learning rate α is a parameter which determines the size of improvement step to take on each iteration. So it is important to choose the learning rate wisely. If α is small, the minimum which is found, is more precise but it takes longer to get it. If the α is large, the minimum is found sooner but there is a risk that the minima is being overshot)

There are three types of Gradient Descent algorithms:

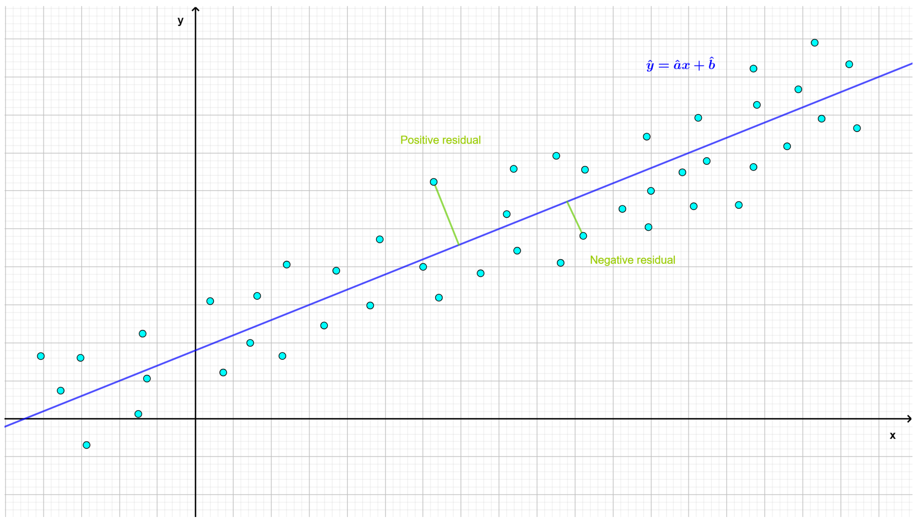
1. Batch gradient descent:
   By using batch gradient descent (BGD), the gradients of all the training examples are calculated and then the average of those gradients is taken to update our parameters a and b (n is in this case the number of training examples in the dataset). So for each iteration we only have one step of gradient descent. If the dataset is big, it may not fit in the memory. Thus this method is computationally very intense and may be very slow. That is why this type is used with smaller data sets.

2. Stochastic gradient descent:
   By using stochastic gradient descent (SGD), one single data point at a time is considered for updating the parameters (n=1). So first we take one training example, calculate its gradient and update the parameters before going on to the next training example and repeating the steps. However, before computing those gradients, we need to shuffle the dataset. This, and the fact that we consider one data point for each iteration, implies that the cost function will be fluctuating over the training examples, which helps to reach a global minima rather than getting stuck at a local minima. However, since the cost function will fluctuate over the training examples, it will not necessarily decrease at the beginning. So this method is used for bigger datasets. Moreover it converges faster to the minimum than BGD when the dataset is large because there is much fewer data to manipulate at a single time. Furthermore since the cost function is fluctuating all the time, it will mostly not reach the minima but it will end up dancing around it.

3. Mini-batch gradient descent:
   This type is a combination of the batch gradient descent and the stochastic gradient descent. It divides the data set into so called mini-batches which are subsets (also called batches) of the dataset consisting of a fixed number of training examples which are randomly chosen. The number of training examples in one mini-batch can be chosen arbitrary and depends on the size of the different datasets. However in average the mini-batch size is chosen to be a power of 2 which lies between 64 and 256. For each mini-batch, we calculate the average of all the gradients of this mini-batch and use this mean gradient to update the parameters a and b. So by taking those mini-batches, our parameters are on the one side updated more frequently than by using BGD, which implies that it converges faster to the minima. And on the other side the parameters are updated less frequently than by using SGD which can lead to more stable convergence. So that we get the advantages of the both preceding methods.

1.2 Ordinary least square

Another model for finding the best matching linear line is called the Ordinary Least Square method. This method tries to minimize the total squared error terms. Those error terms (also called residuals) are the difference between the observed values of y and the predicted values of y based on the regression line. Since those residuals can be positive or negative, it is easier to take the square of them. The line which is represented in the following graphic is the ordinary least square line and denoted by $\hat{y} = \hat{a}x + \hat{b}$.



Let y_i be the real observations and $\hat{y}_i$ the predicted values for $i \in \{0, ..., n\}$. So we have that

$$y_i = \hat{y}_i + \epsilon_i$$

so that

$$\epsilon_i = y_i - \hat{y}_i$$

Then the sum of squared residuals is:

$$\sum_{i=0}^{n} \epsilon_i^2 = \sum_{i=0}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=0}^{n} (y_i - (\hat{a}x_i + \hat{b}))^2$$

Now this quantity is sought to minimize. To do so, we first have to take the partial derivatives of it with respect to $\hat{a}$ and $\hat{b}$:

$$\begin{aligned}
\frac{\partial}{\partial \hat{b}} \sum_{i=0}^{n} (y_i - (\hat{a}x_i + \hat{b}))^2 &= \sum_{i=0}^{n} \frac{\partial}{\partial \hat{b}} (y_i - (\hat{a}x_i + \hat{b}))^2 \\
&= \sum_{i=0}^{n} 2(y_i - (\hat{a}x_i + \hat{b}))(-1) \\
&= -2 \sum_{i=0}^{n} (y_i - (\hat{a}x_i + \hat{b}))
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial \hat{a}} \sum_{i=0}^{n} (y_i - (\hat{a}x_i + \hat{b}))^2 &= \sum_{i=0}^{n} \frac{\partial}{\partial \hat{a}} (y_i - (\hat{a}x_i + \hat{b}))^2 \\
&= \sum_{i=0}^{n} 2(y_i - (\hat{a}x_i + \hat{b}))(-x_i) \\
&= -2 \sum_{i=0}^{n} x_i(y_i - (\hat{a}x_i + \hat{b}))
\end{aligned}$$

Setting the partial derivatives equal to 0 and solve the system for $\hat{a}$ and $\hat{b}$

$$\begin{cases} \sum_{i=0}^{n} (y_i - (\hat{a}x_i + \hat{b})) = 0 & (1) \\ \sum_{i=0}^{n} x_i(y_i - (\hat{a}x_i + \hat{b})) = 0 & (2) \end{cases}$$

From (1) we get:

$$\begin{aligned}
& \sum_{i=0}^{n} (y_i - (\hat{a}x_i + \hat{b})) = && 0 \\
\Rightarrow \quad & \sum_{i=0}^{n} y_i - \hat{a} \sum_{i=0}^{n} x_i - \sum_{i=0}^{n} \hat{b} = && 0 \\
\Rightarrow \quad & \sum_{i=0}^{n} y_i - \hat{a} \sum_{i=0}^{n} x_i = && n\hat{b} \\
\Rightarrow \quad & \frac{1}{n} \sum_{i=0}^{n} y_i - \frac{1}{n}\hat{a} \sum_{i=0}^{n} x_i = && \hat{b}
\end{aligned}$$

By denoting $\bar{x} = \frac{1}{n}\sum_{i=0}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=0}^{n} y_i$, we finally get:

$$\hat{b} = \bar{y} - \hat{a}\bar{x} \quad (3)$$

Substituting (3) in (2), we get:

$$\begin{aligned}
& \sum_{i=0}^{n} x_i(y_i - (\hat{a}x_i + \bar{y} - \hat{a}\bar{x})) = && 0 \\
\Rightarrow\quad & \sum_{i=0}^{n} x_i(y_i - \bar{y} - \hat{a}(x_i - \bar{x})) = && 0 \\
\Rightarrow\quad & \sum_{i=0}^{n} x_i(y_i - \bar{y}) - \sum_{i=0}^{n} \hat{a}x_i(x_i - \bar{x}) = && 0 \\
\Rightarrow\quad & \sum_{i=0}^{n} x_i(y_i - \bar{y}) = && \hat{a}\sum_{i=0}^{n} x_i(x_i - \bar{x})
\end{aligned}$$

So that we get:

$$\begin{aligned}
\hat{a} &= \frac{\sum_{i=0}^{n} x_i(y_i - \bar{y})}{\sum_{i=0}^{n} x_i(x_i - \bar{x})} \\
&= \frac{\sum_{i=0}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=0}^{n}(x_i - \bar{x})^2}
\end{aligned}$$

Finally the linear least square regression line is determined by $\hat{y} = \hat{a}x + \hat{b}$ with

$$\hat{a} = \frac{\sum_{i=0}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=0}^{n}(x_i - \bar{x})^2} \quad \text{and} \quad \hat{b} = \bar{y} - \hat{a}\bar{x}$$

Generalisation to the multidimensional case:

This regression line can also be generalized to the multidimensional case. Now we are no longer speaking of simple linear regression but of multiple linear regression, where we have more than one independent variable. Let's assume that we have k independent variables and n dependent variables/observations. The OLS equation will be of the form $\hat{Y} = \hat{b}X$, where $b \in \mathbb{R}^{k+1}$ and $X \in \mathbb{R}^{n\times(k+1)}$ so that every single observation y_i will follow:

$$y_i = \epsilon_i + \hat{b}_0 + \sum_{j=1}^{k} \hat{b}_j x_{ij} \quad \text{for} \quad i \in \{1, ...n\}$$

where $\epsilon \in \mathbb{R}^n$. (In the 2-dimensional case we had $\hat{b}_0 = \hat{b}$ and $\hat{b}_1 = \hat{a}$, so we need to determine the values of the vector $\hat{b}$)

The whole sample can be expressed in matrix notations as follows:

$$Y = X\hat{b} + \epsilon$$

where $Y \in \mathbb{R}^n$ represents the observations on the dependent variable, $\hat{b} \in \mathbb{R}^{k+1}$ represents the unknown parameters, $X \in \mathbb{R}^{n\times(k+1)}$ is a matrix where the first columns will only contain ones because the model will mostly contain a constant term, and $\epsilon \in \mathbb{R}^n$ represents the errors, so that we have:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & ... & x_{1k} \\ 1 & x_{21} & ... & x_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & ... & x_{nk} \end{pmatrix} \begin{pmatrix} \hat{b_0} \\ \vdots \\ \hat{b_k} \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix} \tag{2}$$

Now, to find the parameters of the vector $\hat{b}$, we need to follow exactly the same procedure as we did in the two-dimensional case. The vector of residuals is in this case given by:

$$\epsilon = Y - X\hat{b}$$

So the sum of squared residuals is the following scalar product

$$\begin{aligned} \epsilon^T\epsilon &= (Y - X\hat{b})^T(Y - X\hat{b}) \\ &= Y^TY - Y^TX\hat{b} - \hat{b}^TX^TY + \hat{b}^TX^TX\hat{b} \\ &= Y^TY - 2\hat{b}^TX^TY + \hat{b}^TX^TX\hat{b} \end{aligned}$$

where we used the fact that $Y^TX\hat{b}$ is a scalar and since the transpose of a scalar is also a scalar we get that $Y^TX\hat{b} = (Y^TX\hat{b})^T = \hat{b}^TX^TY$.
Now, setting the partial derivative with respect to $\hat{b}$ to zero, we get:

$$\begin{aligned} \frac{\partial \epsilon^T\epsilon}{\partial \hat{b}} &= 0 \\ -2X^TY + 2X^TX\hat{b} &= 0 \\ X^TX\hat{b} &= X^TY \end{aligned}$$

Assuming the inverse of X^TX exists, we can multiply by this inverse on both side so that we get

$$\hat{b} = (X^TX)^{-1}X^TY.$$

1.3 Advantages and Disadvantages

Linear regression method is easy to interpret and to implement. It gives us a general direction of the stock prices. If we assume that the trend of some data will continue in the same direction, at least for a while, then we can even extend the regression line and obtain a forecast. However, this method assumes that the relationship between the input and the output is linear. If we have for example a V-shaped or a parabola-shaped data series and try to associate a single regression line to it, we will not get a good approximation of the stock prices. In those cases we would need to calculate two regression lines, to get a better approximation of the evaluation of the data.

1.4 Some Extras

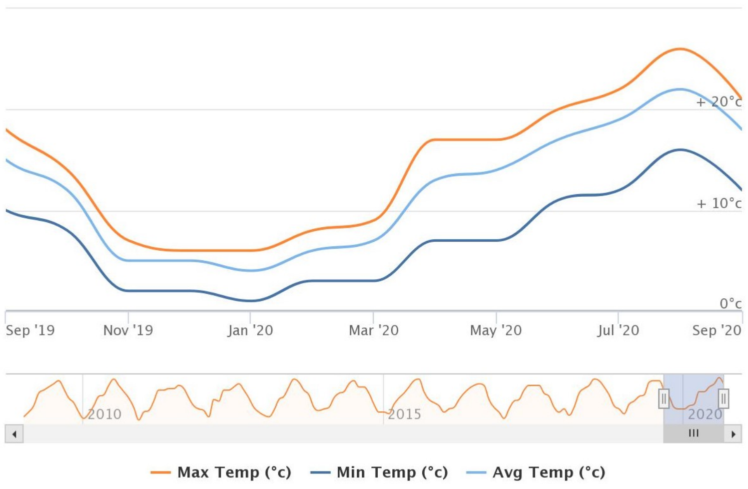
The Ordinary Least Square method and the Gradient Descent method seems to be very similar. By using the OLS method, we find the parameters a and b by using fix formulas, whereas GD depends on many factors so that we construct the parameters a and b which are close to the real ones but not exactly the same. This implies that we get more precise values by using the OLS method than by using the GD method in the two-dimensional case. However, if we move on to the multidimensional case, we see that by using the OLS method, we assume that the matrix $X^T X$ is invertible, if that is not the case, we can not use this method. In this case we need to use the GD method in order to find the best matching linear regression line. If this matrix is invertible, it may be very computationally intense to calculate the inverse of this matrix since X has $k+1$ columns and n lines. So after computing the matrix $X^T X$, we need to invert this $(k+1) \times (k+1)$ matrix, which is again very computationally intense since k is often an integer bigger than 1000.
As a result, we see that even if the OLS method is more precise and faster in the two-dimensional case, the GD method may be the best method for the multidimensional case as it is faster and computationally less intense.

2 Random Forest

Random forest is a popular model because of its flexibility and simplicity. It promises quality results, meaning that the predictions of a random forest are often close to the reality. It is thus a powerful machine learning algorithm which can be used for both regression and classification tasks. A random forest is composed of multiple decision trees. To better understand how the concept works, we will start by explaining decision trees since they are the fundamental building block of random forests.

2.1 Decision Tree - Example

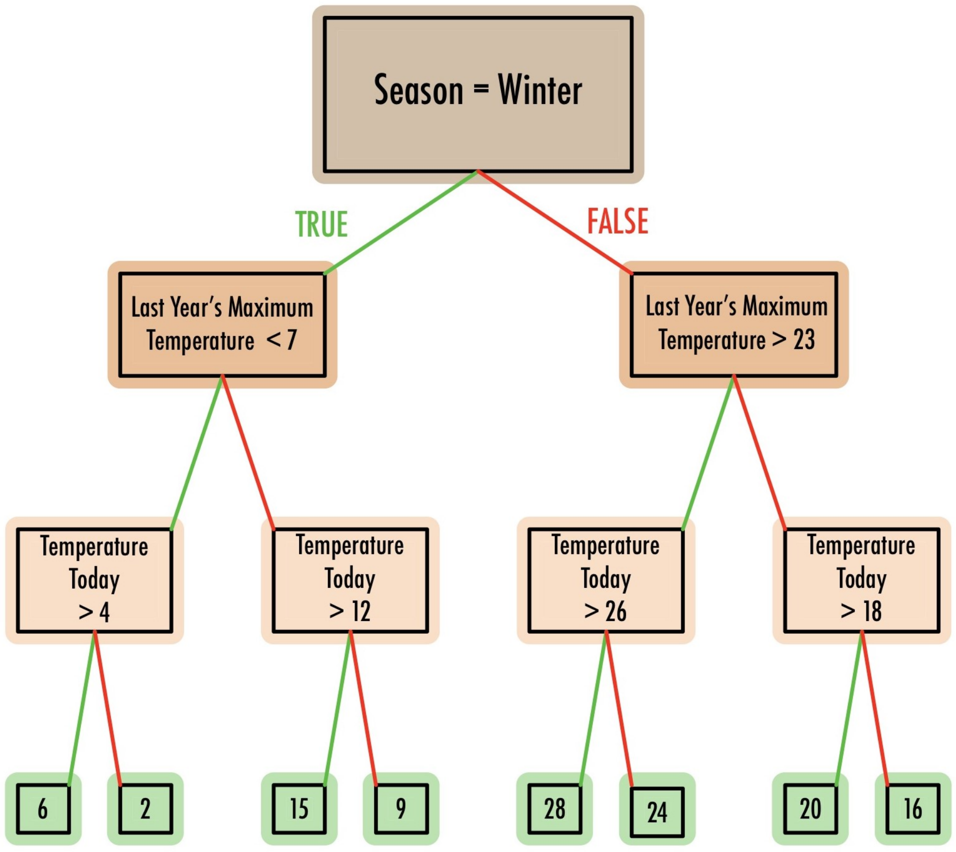
Decision trees use different pieces of information in order to predict a possible outcome. To better understand the functioning of such a decision tree we will start by giving an example which is close to our everyday life and easy to follow.
We will try to predict tomorrow's maximum temperature in Belval.



Data extracted from this website

We start by setting up an initial range. We will take the range from 0° to 30° Celsius since all our maximum temperature values lie between those two values. To predict tomorrow's maximum temperature in Belval it would be useful to find out more information. A good question to start with would be regarding the season. We could ask if it's winter since this allows us to reduce our range to $0° - 10°$ Celsius. We then could ask last year's maximum temperature of tomorrow's date. The information we gain from the answer allows us to again reduce our range, but it may not be enough to already make a prediction. A last question we could therefore ask is what today's maximum temperature is. In this way, we'll know if it's warmer or colder

than last year and we could do a prediction based on this. We could of course ask more and more questions, but let's settle with these three. The questions are indeed high value questions since they greatly reduce the scope of our estimate.
We can retain that in order to arrive at an estimate, we need to use a series of questions which each narrow our possible values until we feel confident enough to make a prediction.
The following picture shows a temperature prediction decision tree. The last row represents the predictions this tree will make, so as an example if the season is winter and last year's maximum temperature for tomorrow's date was 5° Celsius and today's temperature is 3° Celsius, then this tree will predict that tomorrow's temperature will be 2° Celsius.

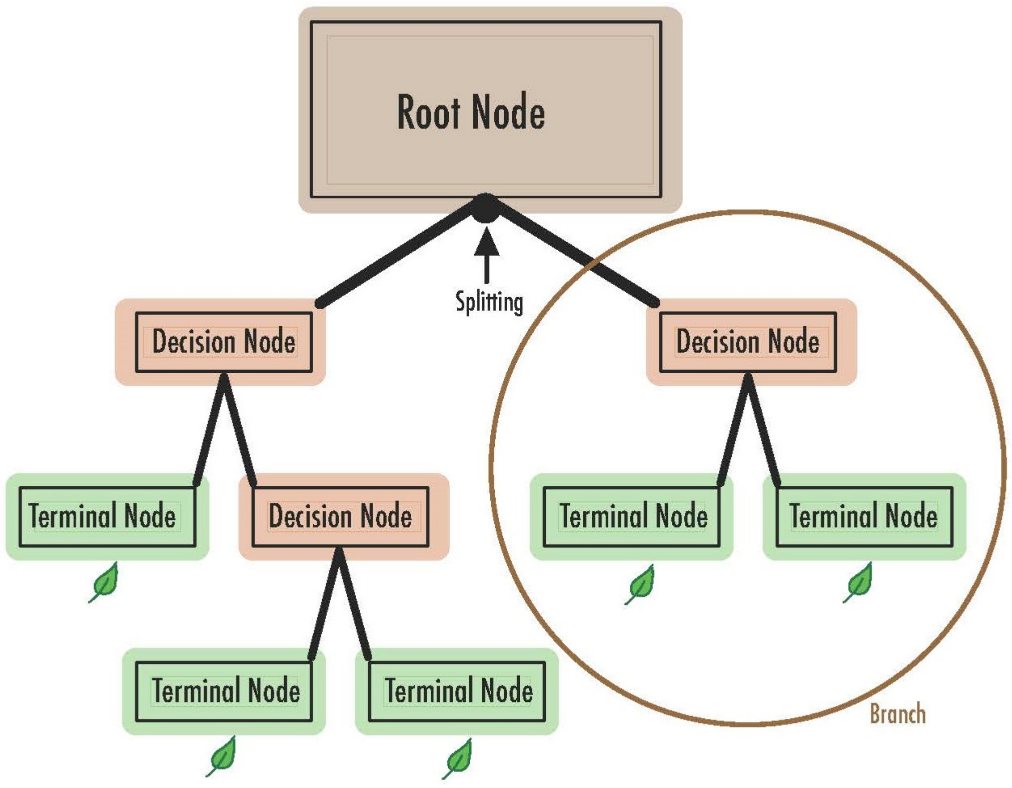


This task is called a **regression task**. It is a way of predicting a value, in this case tomorrow's maximum temperature. The other class of problems is called **classification** where targets are a discrete class label. In this case we could for example want to predict if it's going to be sunny or cloudy.

2.2 Decision Tree

We can observe that each individual tree is composed by branches, nodes and leaves. We will begin with some basic definitions to simplify the lecture:

- The **root node** represents the entire sample. This sample gets divided into two or more homogeneous sets.
- Dividing a node into two or more sub-nodes is called **splitting**.
- A **decision node** splits a sub-node into further sub-nodes.
- When a node does not split we call it a **leaf** or **terminal node**. Each leaf node in the tree specifies a value to be returned by the function.
- The opposite process of splitting is called **pruning**. This means that we remove sub-nodes of a decision node.
- A **branch** is a sub section of an entire tree.
- A **parent node** is a node which is divided into sub-nodes whereas sub-nodes are the **child nodes** of the parent nodes.

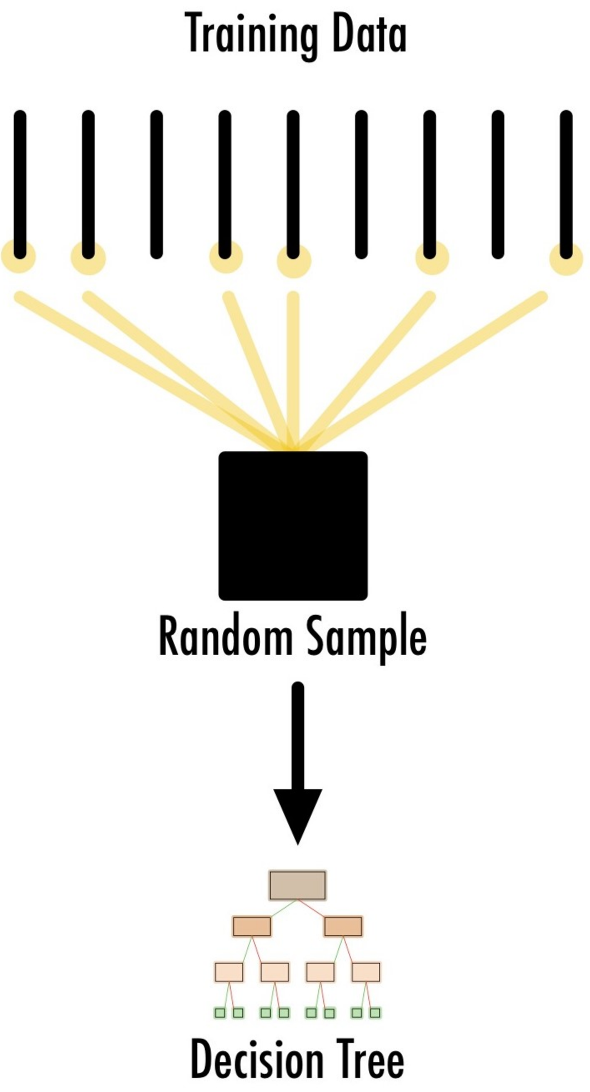


The above example shows us the structure of a decision tree. We quickly realize that the questions asked are under a True or False form. For each True or False answer there are separate branches. For every value we should reach a prediction, under the condition that the object to which we apply the decision tree contains all the necessary data to answer the questions. Note that two splits at each node are fully sufficient since the tree can have an infinite depth. Each question narrows our possible values until the model gets strong and confident enough to make single predictions. The model determines the order as well as the content of the questions.
During training, the model is fitted with any historical data that is relevant to the problem domain and the value we want the model to predict. The model learns any relationships between the data and the target variable. After this training phase, the decision tree produces a similar tree calculating the best questions in the best order in pursuance of the most accurate estimation possible. The prediction will therefore be an estimate based on the train data that it has been trained on.
One important asset of a decision tree is how it learns everything about the problem from the data we provide. The model must be taught about all the relationships there exist. The random forest is a supervised machine learning model and tries to map data to outputs in the training phase of model building.
Note that when we ask the decision tree to make a prediction, it needs to get the same data as used during training. This means that we need to acquire the data in an accessible format containing all the necessary information. It then gives us an estimate based on the structure it has learned.
We can say that the decision tree learns through experience, of course without any previous knowledge. Therefore, after enough training with quality data, the decision tree will know how to map a set of features to targets and make an adequate prediction. We can resume the concept of a decision tree as a flowchart of questions leading to a prediction.
Now raises the question on how such a simple yet successful form of machine learning "learns" and finally builds such a decision tree. The decision tree represents a function which takes an input and returns an output ("the decision"). The attributes are considered as a part of the input. As we explained above, we will have a training and a testing set. Each object in this training set is of the form (x, y) where x is a vector of values for the input attributes and y is the value we want to predict. The decision tree algorithm wants to build a small and efficient tree. Therefore, it adopts a greedy divide-and-conquer method, meaning that it always tests the most important attribute first. The most important attribute is the one that makes the most difference to the classification of an example of the training set. By this divide-and-conquer method, the alogrithm divides the problem into smaller subproblems, which then can be solved recursively. This procedure also underlines the importance of the training set. It is crucial for constructing

the tree, since a tree consists of just tests on attributes in the splitting nodes, values of the attributes on the branches and output values on the leaf nodes. The learning algorithm in fact looks at the data from the training set and the decision tree will be consistent with all the examples from this exact training set. This also implies that the decision tree is bound to make some mistakes for cases it has not encountered in the training set. This greedy search we have just explained is used in decision tree learning to minimize the depth of the final tree.

2.3 From Decision Tree to Random Forest

The predictions we did above are probably wrong. They have variance because they will be widely spread around the right value. To create an accurate prediction we would need to use hundreds or thousands of different decision trees. This is the fundamental idea of random forests. They combine many decision trees into a single model and combined together the predictions will be closer to the true value. One decision tree contains a limited scope of information whereas the combination of many decision trees provides much greater information. A random forest is therefore better than a single decision tree. It is the diversity of different information that makes the random forest more robust.
In fact, each decision tree in the forest considers a random subset of features when forming questions. It then only has access to a random set of the training data points. This implies that the errors of the models are not correlated with each other because they are based on different random subsets. By combining, we increase the diversity in the forest and hence make the overall prediction more robust. We therefore have to keep in mind that the number of decision trees is important and has an impact on the accuracy of our predicted result.
This leads us to the following structure of a random forest:



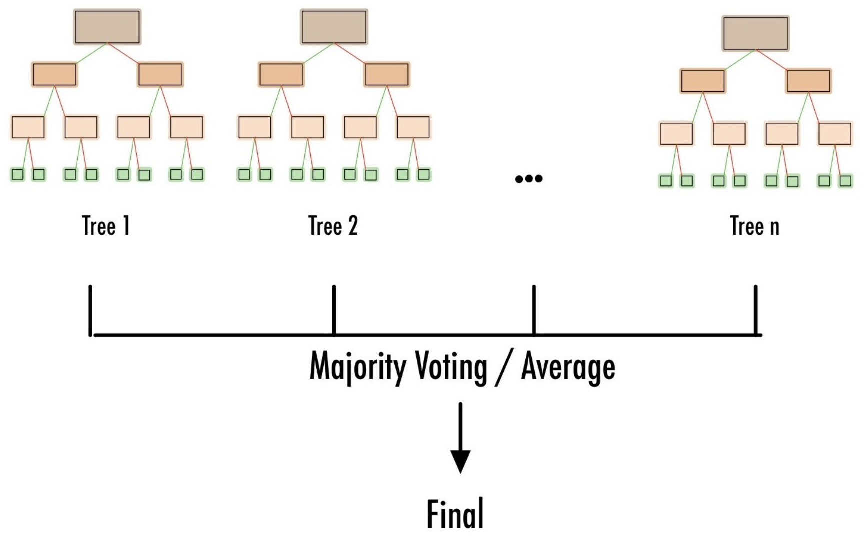
Our first step to create a random forest consists in selecting random samples from the training data set.

↓

The algorithm will then construct a decision tree for every sample and get the prediction result from every decision tree. Note that each tree sees only a subset of all the features.

↓

The prediction result will be the average of all the individual decision trees estimates if it is a regression problem. If we work with classification, we perform a majority voting for the predicted class and our final prediction result will be obtained by the most voted prediction result.



2.4 Random Forest

The first thing we need to build a random forest is data. We need to make sure that there is no missing or incorrect data, because this can impact the analysis. Once we have obtained the raw data it needs some minor modifications in order to put it into machine-understandable terms. We then need to separate the data into the features and targets. The **target** is the value we want to predict. One last step concerning the data preparation is splitting the data into **training and testing set**. Note that the sampling of training observations is random. During training, the model learns how and what to predict by "seeing" the answers. It tries to learn the relationship between all the features and the target value. Afterward, when we want to evaluate the model and ask it to make predictions on the testing set. The model then only has access to the features and not to the answers. We then can compare the prediction with the actual answer and judge the accuracy of the model.
After preparing the data, we finally start to create and train the model. We therefore use the **Scikit-learn** from which we import the random forest classification (resp. regression) model and fit the model on the training data. This simple tool allows the model to learn the relationships between the features and the targets. Recall that each individual tree brings their own information sources to the problem as they only consider a random subset of the training data.

After the training phase, we start to look at the accuracy of the model by comparing its predictions on the test features to the known answers. Recall that the random forest builds multiple decision trees and merges their predictions together in order to get a more accurate and stable prediction

than relying on individual decision trees.
To improve we can try different hyperparameters. This means that we adjust the settings to improve performance. The settings are known as hyperparameters, not to be confound with the model parameters learned during training. The hyperparameters must be set by the data scientist before training. These settings can include the number of decision trees in the forest and the number of features considered by each tree when splitting a node.

2.5 Advantages and Disadvantages

Random forest requires only a very few pieces of feature engineering and can be used for both regression and classification tasks.
Of course a random forest is more complex than a decision tree algorithm but it is also more robust because of its increased diversity. Recall that each tree in a random forest learns from a random sample of the training observations. Each tree might have high variance with respect to a particular set of the training data but overall, the entire forest will have a significantly lower variance.
Furthermore, the random forest is less susceptible to overfitting. Overfitting means that the model does not generalize well from the training data to the unseen data. So even if a decision tree has a very high accuracy on the training set, this does not necessarily imply that it also has high accuracy on the testing set. This phenomenon is known as overfitting and is often a problem of decision trees but because of the increased diversity of a random forest, it presents hardly ever a problem to the random forest.

2.6 Some Extras

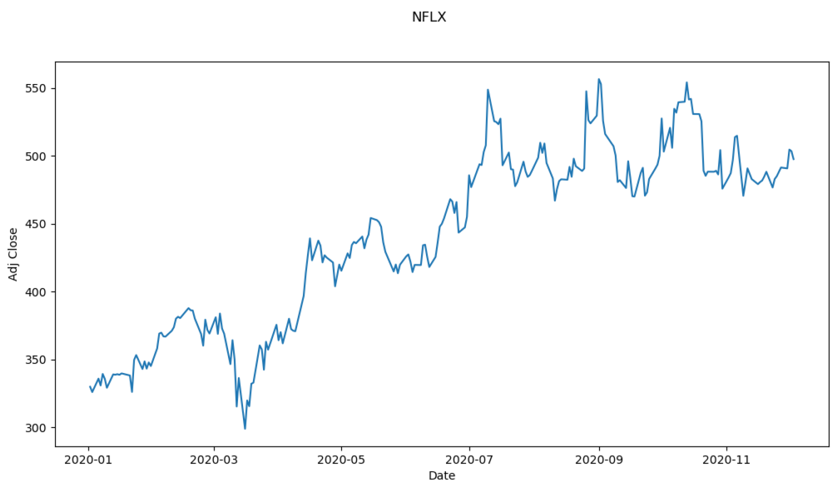
Bagging is the idea of combining the predictions of different models which only are somewhat predictive. They are not correlated with each other and thus have different insights into the relationship of the data. The idea behind is that every tree has errors but the errors are random. The error of these trees which all have been trained on a different random subset will average out to zero and this is why bagging is so important.
Another recurring term when reading about random forest is bootstapping. Each tree in a random forest learns from a random sample of the training observations. The samples are drawn with replacement, which is known as bootstrapping. This means that some samples will be used multiple times in a single tree.

3 Examples

In this section we will apply the methods explained above to a company in order to predict the stock prices or similar. To do so, we first choose a company, then extract the historical data of the stock prices of this company from yahoo finance and use this historical data to make a forecast of the evolution of the stock prices. This is done by using different programs. The details of those programs can be found by opening the associated html file of the program.

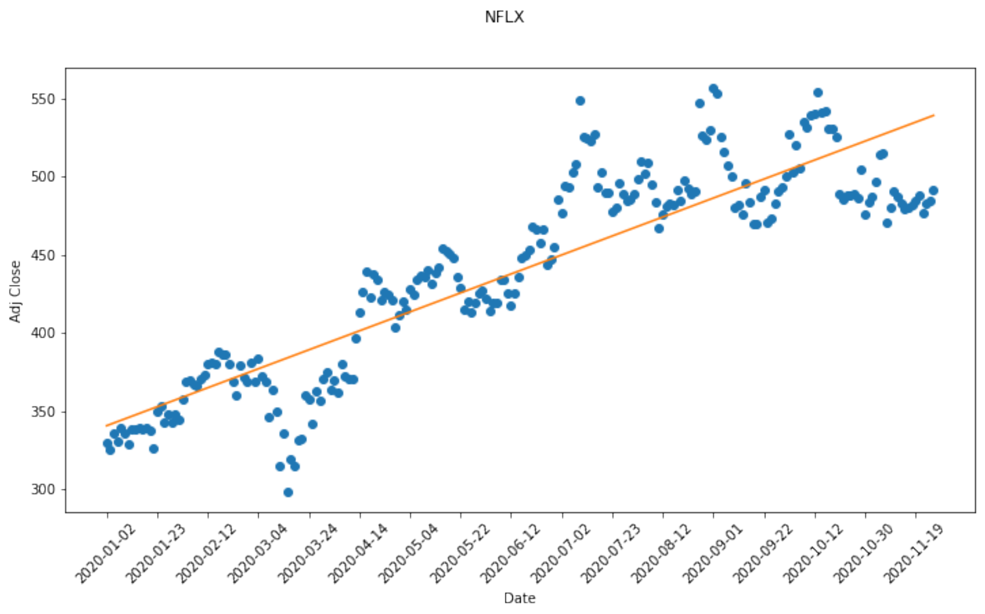
3.1 Netflix

As a first company we chose the well-known streaming service Netflix. Here is the output of our first program which plots the historical data of the stock prices of the year 2020.



In this graph, we can see that the stock prices of Netflix increased rapidly from March until July which might be in relation with the Covid-19 pandemic. However the stock prices were not only increasing during these four months, but we have a general upward trend of the stock prices over the whole year. This implies that we can apply the linear regression models to our dataset to get a general direction or even a forecast for the stock prices of Netflix. Since we can find the best matching linear regression line using two different methods, we wrote two programs, one using the Gradient Descent method and the other one using the Ordinary Least Square method. The objective of both programs is to find the best matching parameters a and b and as an output they return the equation of the regression line and plot the data with the appropriate linear regression line.
Here we can see the output of the program using OLS-method which plots the graph of the stock prices over the whole year 2020 combined with the

best matching linear regression line:

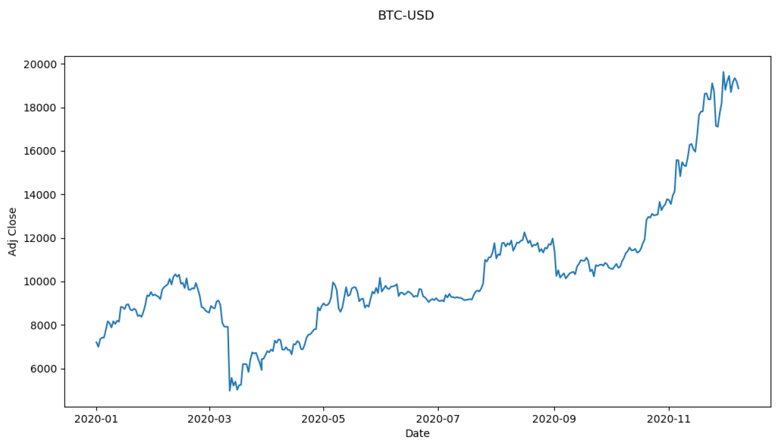


The program using the GD method returns almost the same linear regression line. However it takes longer to compute it and it is less precise since this method depends on more factors. Not only the choice of the learning rate and of the initial parameters a and b is important but also the number of iterations and the number of gradient descent steps is important. The latter depends on which method of gradient descent is used.

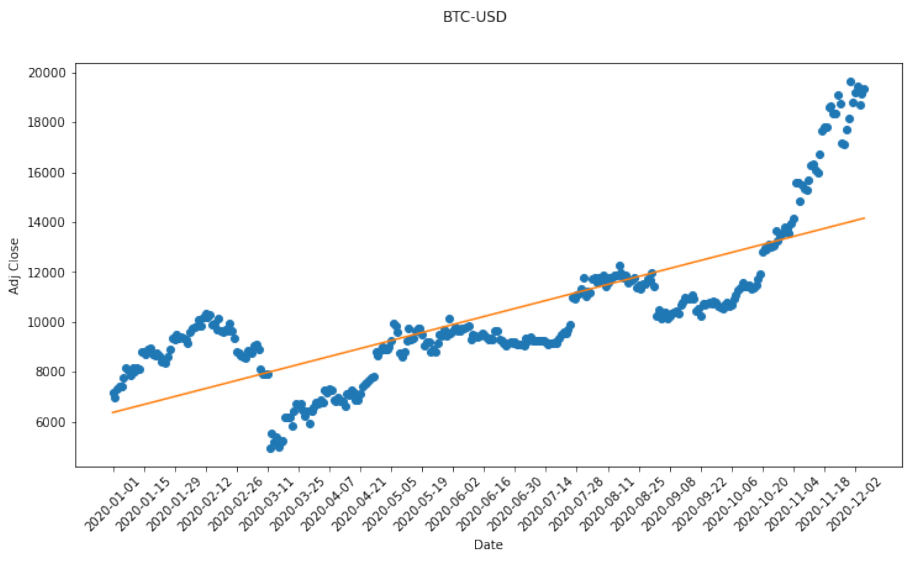
We also wrote two random forest models, one solving a regression task and one solving a classification task. The Random Forest Classifier tries to predict whether the stock is closing up or down and the Random Forest Regressor tries to predict the closing price. Both random forests seem to be pretty accurate. They both are easy to write if one is used to the SciKit Learn metrics. The time consuming part is the data preprocessing, where one adds relevant information via formulas to the default data.

3.2 Bitcoin

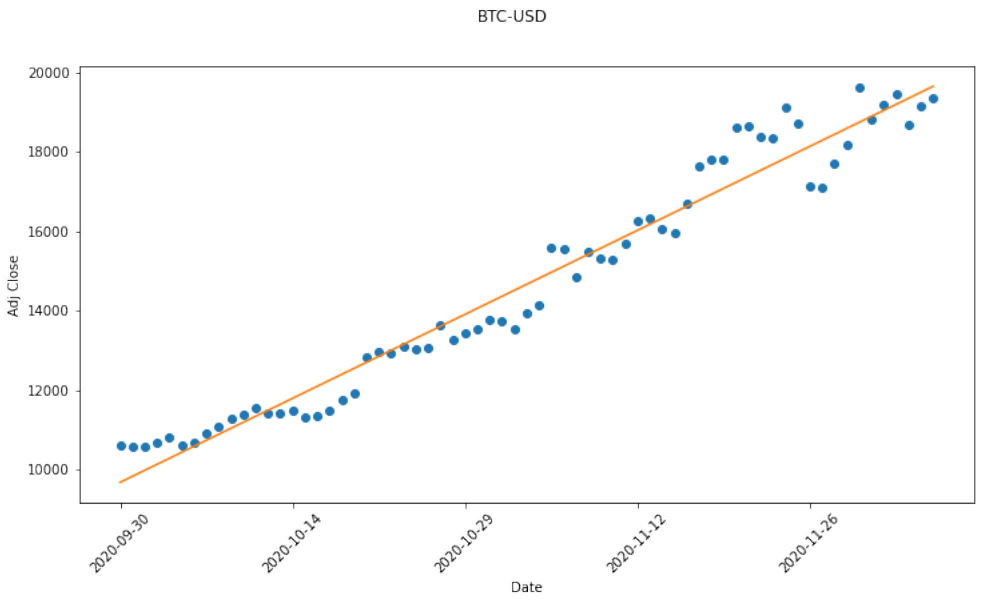
As a second company we chose the well-known digital and global money system currency Bitcoin. Again we use our fist program to plot the historical data of the stock prices of the year 2020.



In this graph we can see that the stock prices of Bitcoin are rapidly increasing at the end of the year, whereas there were only little variations during the rest of the year. To use the linear regression model, we first use the OLS-program to plot the stock prices of Bitcoin over the whole year 2020, associated with the OLS regression line:



In this graph, we can see that we will not get a good approximation of the stock prices using this regression line because the difference between the variations is to big. So by dividing the dataset into subsets such that we get more than one regression line, we can approximate the stock prices much better. By plotting the dataset from October first until the end of the year we get for example the following graph



Here we see, that the regression line is a good approximation of the stock prices and if we assume that the trend will continue in this direction, we can even get a good prediction of the stock prices in the near future.

However, both of the Random Forest models maintain their accuracy for this second example.

4 References

https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325

https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a

https://medium.com/@kumaranupam2020/difference-between-batch-gradient-descent-bgd-minibatch-gradient-

https://gdcoder.com/random-forest-regressor-explained-in-depth/

https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

https://www.worldweatheronline.com/lang/fr/belval-weather-averages/luxembourg/
lu.aspx

https://towardsdatascience.com/random-forest-in-python-24d0893d51c0