UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

CRYPTOCURRENCIES PRICE PREDICTION BY USING MACHINE LEARNING AND DEEP NEURAL NETWORKS

Noémi Uzonyi

Tutor: José Javier López Monfort

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster en Ingeniería Telecomunicación

Curso 2020-21

Valencia, 29 de junio de 2021

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Table of Content

Abstract

Nowadays, bitcoin trading is becoming more and more popular. There are a big number of open-source trading strategies available. Recently, the idea of using machine learning to develop a bitcoin trading strategy has also emerged.

The study is inspired by the topic. By monitoring the bitcoin exchange rate, we created a machine learning classification model with the aim of determine the alteration of the next change based on some financial indicators. To achieve this, it is necessary to examinate the bitcoin price historically. Then, machine learning methods are used to predict whether the exchange rate will increase or decrease at the next trading time. Based on this knowledge, we propose our trading strategy.

In this dissertation, we involve some areas of supervised and unsupervised machine learning, focusing on the task of classification. The following methods are implemented with the aim of finding the most efficient model: Neural Network, Logistic Regression, K Nearest Neighbor, Supporting Vector Machine, Decision Tree, Random Forest, and Naive Bayes models. To compare the performance of classification methods, accuracy, precision, recall, F1-score, and runtime metrics are considered. We perform a comparative analysis of the different classification methods to find the most efficient one, which is used as the core of the trading strategy.

After defining the machine-learning-based trading model, we present several backtesting studies to monitor the behavior of the strategy. We perform backtesting on the exchange rates of several cryptocurrencies and traditional trading market products as well.

Finally, we recommend a trading bot, operating the defined classification-model-based strategy in real time.

Resumen

Hoy en día, el comercio de bitcoins es cada vez más popular. Existen varias estrategias de comercio en código abierto. Recientemente, también ha surgido la idea de utilizar el aprendizaje automático para desarrollar una estrategia de compraventa de bitcoins.

El estudio se inspira en el tema. Mediante el seguimiento del tipo de cambio del bitcoin, creamos un modelo de clasificación de aprendizaje automático con el objetivo de determinar la alteración del próximo cambio en función de algunos indicadores financieros. Para ello, es necesario examinar el precio del bitcoin históricamente. A continuación, se utilizan algunos métodos de aprendizaje automático para predecir si el tipo de cambio aumentará o disminuirá en el próximo periodo de cotización. Basándonos en este conocimiento, proponemos nuestra estrategia de inversión.

En esta disertación, involucramos algunas áreas del aprendizaje automático supervisado y no supervisado, centrándonos en la tarea de clasificación. Se implementan los siguientes

métodos con el objetivo de encontrar el modelo más eficiente: Redes neuronales, regresión logística, K vecindad más cercana, máquina de vectores de apoyo, árbol de decisión, bosque aleatorio y modelos Naive Bayes. Para comparar el rendimiento de los métodos de

clasificación, se consideran las métricas de exactitud, precisión, exhaustividad, puntuación F1 y tiempo de ejecución. Realizamos un análisis comparativo de los distintos métodos de clasificación para encontrar el más eficiente, el cual se utiliza como núcleo de la estrategia de comercio.

Tras definir el modelo de negociación basado en el aprendizaje automático, presentamos varios estudios de pruebas retrospectivas para supervisar el comportamiento de la estrategia. Realizamos pruebas retrospectivas con los tipos de cambio de varias criptodivisas y también con los productos tradicionales del mercado.

Por último, recomendamos un programa de inversión que opera la estrategia definida basada en el modelo de clasificación en tiempo real.

Resum

Hui en dia, el comerç de bitcoins és cada vegada més popular. Hi ha diverses estratègies de comerç en codi obert. Recentment, també ha sorgit la idea d'utilitzar l'aprenentatge automàtic per a desenvolupar una estratègia de compravenda de bitcoins.

L'estudi s'inspira en el tema. Per mitjà del seguiment del tipus de canvi del bitcoin, creem un model de classificació d'aprenentatge automàtic amb l'objectiu de determinar l'alteració del pròxim canvi en funció d'alguns indicadors financers. Per a això, és necessari examinar el preu del bitcoin històricament. A continuació, s'utilitzen alguns mètodes d'aprenentatge automàtic per a predir si el tipus de canvi augmentarà o disminuirà en el pròxim període de cotització. Basant-nos en aquest coneixement, proposem la nostra estratègia d'inversió.

En aquesta dissertació, involucrem algunes àrees de l'aprenentatge automàtic supervisat i no supervisat, centrant-nos en la tasca de classificació. S'implementen els següents mètodes amb l'objectiu de trobar el model més eficient: Xarxes neuronals, regressió logística, K veïnat més pròxima, màquina de vectors de suport, arbre de decisió, bosc aleatori i models Naive Bayes. Per a comparar el rendiment dels mètodes de classificació, es consideren les mètriques d'exactitud, precisió, record, puntuació F1 i temps d'execució. Realitzem una anàlisi comparativa dels distints mètodes de classificació per a trobar el més eficient, el qual s'utilitza com a nucli de l'estratègia de comerç.

Després de definir el model de negociació basat en l'aprenentatge automàtic, presentem diversos estudis de proves retrospectives per a supervisar el comportament de l'estratègia.Realitzem proves retrospectives amb els tipus de canvi de diverses criptodivisas i també amb els productes tradicionals del mercat.

Finalment, recomanem un programa d'inversió que opera l'estratègia definida basada en el model de classificació en temps real.

1. Introduction

Artificial Intelligence is an active research area that defines our recent life. The discipline provides solutions to many real-life problems and is an integral part of our daily lives. Personalized online advertisements, internet searches, smart cars are all systems based on artificial intelligence applications. Machine learning is a subset of Artificial Intelligence in which we implement human thinking using mathematical models, [14]. Machine learning has many applications in the areas of customer analysis, cost reduction, risk reduction, among others. Data science is an interdisciplinary field that uses scientific methods, algorithms, processes, and systems to explore knowledge from structured and unstructured data and then apply that information, [4]. Data science is related to data mining, machine learning, and big data.

The aim of machine learning is to provide some conclusion or prediction from the available dataset using a model. The essence of the method is that the model is continuously improves itself by learning new experiences. Machine learning deals with three types of tasks: supervised learning, unsupervised learning, and reinforcing learning. In supervised learning (SL), we construct a model that determines known output values based on the input objects, [10]. Classification and regression are typical cases of supervised learning. In the case of unsupervised learning (UL), the algorithm looks for an unknown pattern in the unlabeled data set, [5]. A typical task for this is clustering. Reinforcing learning (RL) deals with how smart agents should act in an environment to maximize cumulative reward, [7]. In this study, we deal with some areas of supervised and unsupervised learning.

Classification is a supervised machine teaching task, defined as follows. Given a training data set consisting of records, that is, a set of attribute values. A class attribute is given for each record. During the classification, we use the learning algorithm to find a model that can determine the target attribute i.e., the classifying variable, as a function of each attribute as accurately as possible, [11]. The test data set is used to evaluate the performance of the model. Numerous techniques can be used to perform the classification task. Based on a state-of-art literature review in the field of classification and crypto trading, the following methods will be considered in the study: Artificial Neural Network, Logistic Regression, k Nearest Neighbor, Support Vector Machine, Decision Tree, Random Forest, and Naive Bayes models. Various metrics can be used to compare the performance of the mentioned methods, of which accuracy, precision, recall, F1-score, and runtime will be included in this study.

Since the last decades, particular attention has been given to the topic of cryptocurrency. Cryptocurrency is a digital device that also functions as a currency of exchange. Cryptocurrencies represent a subset of digital currencies but can also be classified as a group of alternative currencies or virtual currencies. In terms of this currency, cryptography is used to secure transactions. A common feature of most cryptocurrencies is decentralization, i.e., operation without central supervision which allows it to be used as a means of payment across borders.

The popularity of cryptocurrency trading has increased significantly recently, and the cryptocurrency industry is rapidly moving forward, [6]. Cryptocurrency trading has high volatility, so with optimal trading actions high profitability could be realized. In contrast to traditional markets, cryptocurrencies operate continuously, without interruption. In addition,

bitcoin trading is mostly unregulated and there are many opportunities to enter the market easily. An entire industry has already been built to study the cryptocurrency exchange rate, visualize graphs, and analyze the trends with the aim of creating more and more profitable trading strategies. Price changes can be approached from several perspectives. One option, for example, is the fundamental analysis, which uses macroeconomic factors to try to predict whether the price may increase or decrease. They take into account, for example, what kind of news are coming from the industry, what developments, government regulations are expected, and so on. Another option is technical analysis, which uses market statistics. Exchange rate movements and other metrics of trading, such as volume, are monitored. This technique tries to identify trends and deduce what to expect. Moreover, the trading can be aided with automatization executed on an autonomous agent. Over the last decade more and more automated trading robots have emerged with the proposal of automatic cryptocurrency trading, [1].

In this study, we propose a cryptocurrency trading strategy in Python based on some machine learning tools. A dataset of historical cryptocurrency data will be built, and some financial indicators will be calculated. A classification problem will be defined on the obtained dataset to predict the change of exchange rate in the future. The classification will be carried out by several different methods, which performances will be compared by different metrics with the aim of finding the most suitable one. Then, a trading strategy will be implemented based on the classification model. The strategy will be simulated on several historical data with the aim of examining and evaluating the behavior of the created trading model in different situations. Finally, an automatic trading software will be proposed to apply the machine learning based strategy in real-life environment.

2. Related work

2.1 The data

Binance is the largest, cheapest, one of the safest and most popular stock exchanges in the world and offers a globally available, professional, easy-to-use and user-friendly trading interface for bitcoin and many other cryptocurrency trading. In the following sections we will describe our machine learning based crypto trading strategy designed for automatic trading on Binance.

The concept is to create a machine learning model which predicts the future behavior of bitcoin exchange rate. For this we propose a classification model to predict whether the price is going to increase or decrease in the next market moment. In this study we use six-hour time windows for trading. So, the main idea is to collect historical data about the price changes of bitcoin in every six hours. Several ways can be used to download this data. Here we present a function which sends a request to the Binance webpage specifying the start and end times as timestamps and defines the period which will be set to six hours.

```
def get_binance_data(symbol, interval, startTime, endTime):
    url='https://api.binance.com/api/v3/klines'
    req_params={'symbol': symbol, 'interval': interval, 'startTime':
        startTime, 'endTime': endTime}
    df=pd.DataFrame(json.loads(requests.get(url, params=req_params).text))
    df.columns=['datetime', 'open', 'high', 'low', 'close', 'volume']
    df=df.shift(1)
    df.dropna(inplace=True)
    return df
```

The function returns a pandas data frame containing records with bitcoin prices for every six hours between the start and the end date. The table below describes the information about the attributes in the dataset.

Attribute	Description
datetime	Timestamp of the closing price.
open	The price of bitcoin in USD six hours before the timestamp.
high	The highest price of bitcoin in USD during the six-hours period.
low	The lowest price of bitcoin in USD during the six-hours period.
close	The current price of bitcoin in USD at a time.

To obtain a more accurate prediction we extend our dataset with some financial indicators in another function. The core of the function is detailed below.

```
def calculate_indicators(data):
    ...
    return data,move
```

Since at a specific point of time the closing exchange rate returns the current price of bitcoin, in the trading and the classification we will focus on the close prices. First of all, we calculate the well-known moving average of the last five closes as follows.

```
data['ma5']=data['close'].rolling(window=5).mean()
```

Then, we calculate the ‘price change’ since the last close as a percentage of the previous close. The percentage of increase or decrease in price is also calculated based on the previous close as ‘upmove’ and ‘downmove’.

```
data['price_change']=data['close'].pct_change()
data['upmove']=data['price_change'].apply(lambda x: x if x>0 else 0)
data['downmove']=data['price_change'].apply(lambda x: abs(x) if x<0 else 0)
```

Furthermore, the up and down moves are averaged by using the Wilder's Smoothing Method. As the Wilder's Smoothing Method has no function implemented in Python, we can use the Exponential Moving Average with the appropriate alpha parameter. Later we would like to calculate the 10-period RSI as another financial indicator, so in the Wilder's Smoothing Method we use $\frac{1}{N} = \frac{1}{10}$ as parameter. However, in the Exponential Moving Method the parameter is in $\frac{2}{M+1}$ form. So, to work with the right parameter, we should call the Exponential Moving Method with $M = 19$ parameter. We apply the method on both up and down movements.

```
data['avg_up']=data['upmove'].ewm(span=19).mean()
data['avg_down']=data['downmove'].ewm(span=19).mean()
```

The Relative Strength (RS) and the Relative Strength Index (RSI) can be calculated by executing the following lines.

```
data['RS']=data['avg_up']/data['avg_down']
data['RSI']=data['RS'].apply(lambda x: 100-(100/(x+1)))
```

Besides the original open, close, low, and high prices, the calculated variables described above will be considered in the classification model. In the machine learning tasks we do not involve the timestamp. With the classification task we aim to predict whether the closing price is going to increase or decrease in the very next market moment, which is in exactly six hours in our case. So, we would like to train the models to predict if the price is going to go up or down. Thus, we order ‘up’ or ‘down’ labels to the records in the dataset as follows.

```
for i in range(len(data)-1):
    if data['close'][i] < data['close'][i+1]:
        move.append('up')
    else:
        move.append('down')
data['move']=label
```

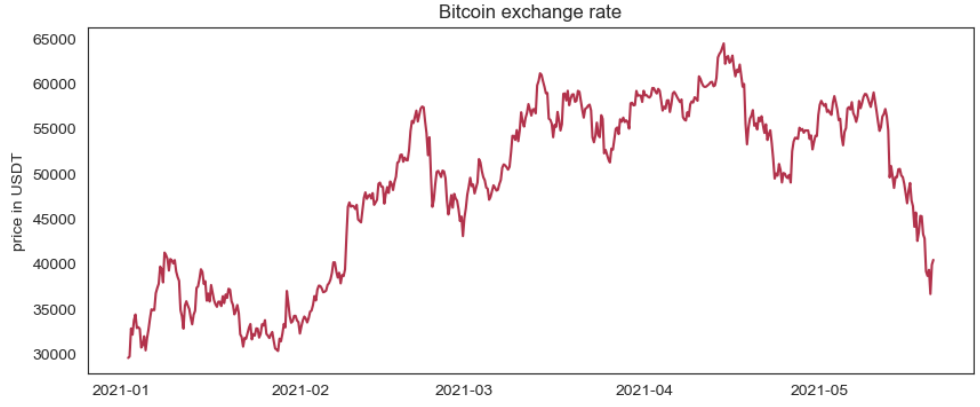

Figure 1. Bitcoin exchange rate between 01.01.2021 and 21.05.2021. in USDT.

After completing our dataset, we check some basic statistical properties of the obtained data. First of all, we visualize the close prices in time which is presented on Figure 1.

Secondly, we examine the ratio of price increases to price decreases. The historical data was downloaded from the 1st of January, to the 21st of May, 2021. During this period, we can say that the number of price increases and decreases was broadly balanced. It can be seen in the Figure 2. During the implementation of the study, we checked this distribution several times and always found a similar result.

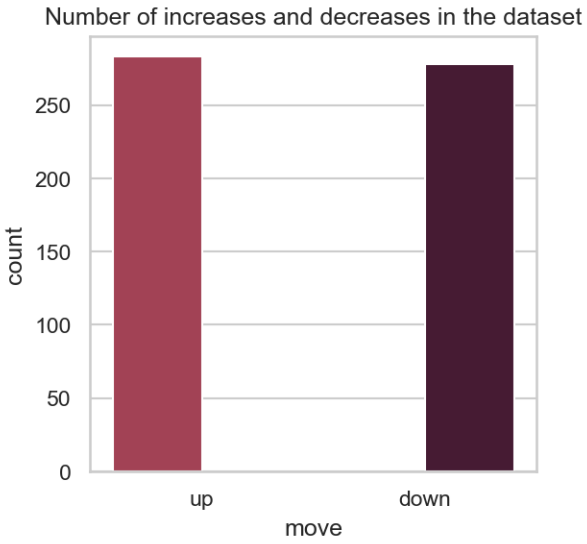

Figure 2. Number of increases and decreases is bitcoin price between 01. 01. 2021 and 21. 05. 2021.

Then, we examine the covariance matrix of the variables. We consider the Pearson correlation of all pairs of the variables to check if there is a linear relationship between any of the attributes.

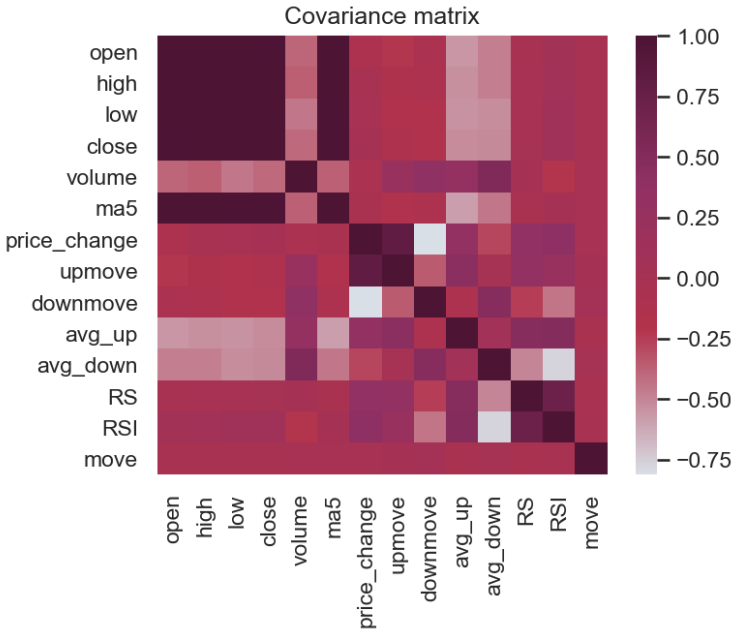

Figure 3. Covariance matrix of the dataset

As the Figure 3. shows above, we verify a strong linear relationship between the original open, close, high and close variables. Furthermore, there variables are highly correlated with the 5-period moving average as well. Obviously, the up move and down move variables are also highly correlated to the price change, since they were derived from it. The most remarkable result that we can identify is that move attribute, so our target label is uncorrelated with all the variables. This means that any single attribute can be used to determine whether the price will move up or down in the next market moment. Thus, in the classification task, we will try to predict this price change with some combination of the attributes.

In the next step we would like to visualize the dataset. To do this, we can use Principal Component Analysis (PCA) to transform the fourteen-dimensional dataset into two dimensions. The PCA can be considered as a data reduction method and as a special case of factor analysis. It implements a reduction in the dimensions while retaining the variance present as much as possible. Therefore, with this method, we can transform our data into a form that separates the different data points as much as possible for data visualization. Figure 4. shows the visualization of the records of the dataset represented with coloring according to the class. As we can see, the two classes are not linearly separable.

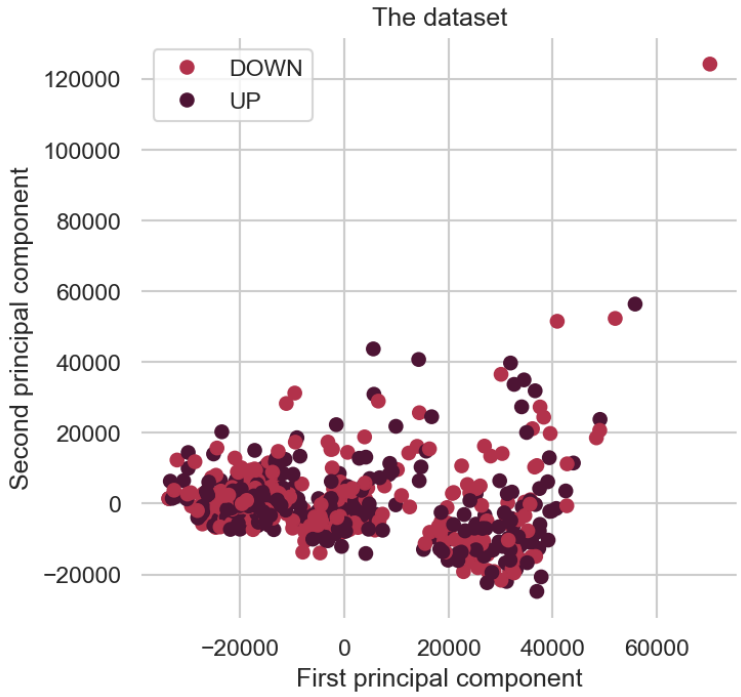

Figure 4. Visualization of the dataset

2.2 Classification models

The aim of the study is to predict the label of the records according to whether the exchange rate will rise or fall in the next market moment compared to the current price. To accomplish this, we use a subset of the assembly attributes in the data set. Our goal is to create a classification model that can predict the direction of the next exchange rate change with some accuracy using some simple financial indicators presented above. To find the best classification

model for this issue, our idea is to implement a comparative analysis of different classification methods based on several metrics to find the most suitable one.

Before the classification, it is necessary to preprocess the data. Missing values are already handled when the dataframe is created. Since we work with numeric variables in classification tasks, we need to modify the values of the target variable. It can be easily implemented by using the Label Encoder method which converts categorical variables to numeric. The essence of the simple method is to assign a number to each possible categorical value. If the next exchange rate increases, the value of the 'move' attribute will be changed to 1, otherwise it will be 0. This label encoding can cause a new problem because it uses a series of numbers, thus introducing comparability and sorting between values. The danger of using the algorithm is that it assigns some hierarchy to the data - that results in an ordinal variable, even though the original variable was nominal. However, as the use of the label encoder is still practical in our case.

```
from sklearn.preprocessing import LabelEncoder
LE=LabelEncoder()
data['move']=LE.fit_transform(data['move'])
```

A prerequisite for using many machine learning methods is to work with standardized data. In order for a given property not to dominate the machine learning task, each variable is scaled to a zero-mean, one-standard deviation with a normal distribution. This can be done in Python using the StandardScaler tool in the scikit-learn library preprocessing package. In the code presented below we scale all the data in the dataset with the exception of the target variable.

```
data_without_move=data.drop('move',axis=1)
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(data_without_move)
scaled_data=scaler.transform(data_without_move)
```

When implementing the classification, it is recommended to use feature selection to select the most appropriate set of x attributes, [3]. Using this method results in a dimension reduction to increase performance. The univariate feature selection selects the best properties based on statistical tests. During selection, we use the recommended default f_classif() function for the classification task, which calculates the ANOVA F-value of the specified sample. Out of the 13 attributes in the database, disregarding the target variable, 10 attributes are considered after the selection.

After preprocessing the data, we define the classification task. Classification is the task of learning an objective function f that assigns one of the predefined y class labels to a set of attributes x, [11]. In training process, our input data are (x, y) recodes in which we know the discrete class variable y for the values of the attribute set x. After teaching the model, in the case of predictive modeling, we can use the model to predict the class labels of new x records.

After identifying the variables x and y, we define the classification task. To train and test the model, the data set is divided into two sets. 70% of the records are assigned to the training data set and 30% of the records to the testing set. Since in our case we are dealing with a time series, the training and testing set can be assigned as follows.

```
x_train = x[:int(len(x) * (70 / 100))]
y_train = y[:int(len(y) * (70 / 100))]
x_test = x[int(len(x) * (70 / 100)):]
y_test = y[int(len(y) * (70 / 100)):]
```

The classification task will implemented with different machine learning models in order to find the most accurate model. In each case, the models are fitted to the training data set and their performance is tested on the testing data set. The results of the classifications are displayed by a confusion matrix. The matrix displays the class label predicted by the classification model on the x-axis and the expected class label on the y-axis. In the main diagonal, we get the number of correctly classified records when testing the model, the other fields show the number of incorrect classifications.

In our experiment, we ran each classification algorithm a hundred times. The results of the classifications are recorded in each case. The confusion matrices below show the average number of classifications experienced during the series of experiments.

2.2.1 Artificial Neural Netrowk (ANN)

The Artificial Neural Network (ANN) is a classification model that mimics the biological nervous system, [9]. It consists of interconnected processing elements, neurons that cooperate to produce the output function. Each neuron generates its output value using an aggregation node and an activation function for the input values as follows.

$$f(x) = (\sum_{i=1}^{m} w_i * x_i) + b$$

In the equation, x_i denotes the input values, b is the bias parameter, and w_i indicates the weights, which is the learner parameter of the model. The adaptive framework changes its structure during the learning process by error backpropagation - by changing the weighting of the input nodes - to minimize the error experienced at the output. The ANN can be a simple perceptron model or a Multi Layer Perceptron (MLP). MLP can be composed of several intermediate, hidden layers that connect the input and output layers and can be used to model complex relationships between input and output variables.

The advantages of the method are rapid prediction, the ability to handle noisy data, and the ability to easily identify complex relationships, [9]. However, the input data must always be converted to numeric. When using this method, there is a risk of over-fitting and it should be taken into account that processing time may increase in the case of large neural networks.

The implementation of the default artificial neural network in Python can be seen below. The default model contains 100 hidden layers and uses relu as activation function. In this study we implanted several different configurations and increased the maximum number of iterations as well. However, the improvement in accuracy that we could achieve was not remarkable.

```
from sklearn.neural_network import MLPClassifier
mlp_model=MLPClassifier()
mlp_model.fit(X_train, y_train)
mlp_prediction=mlp_model.predict(X_test)
```

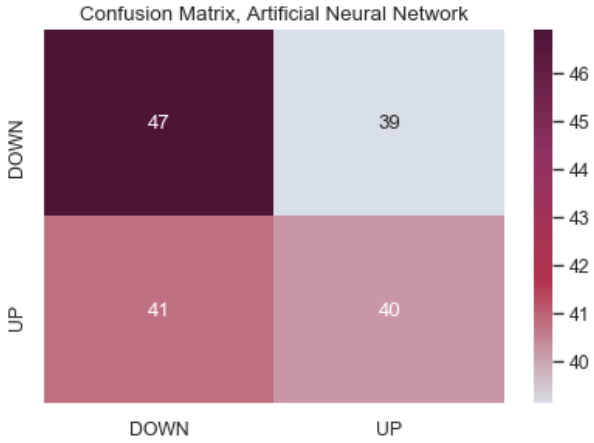

Figure 5. Confusion matrix of the Artificial Neural Network (ANN) model

The matrix shows that the neural network correctly classified an average of 47 price decreases and 40 price increases during the hundred experiments. In 39 cases the model predicted that the price will increase, while the price went down. In 41 cases the model predicted that the price will go down, though it went up.

2.2.2 Logistic Regression

Logistic regression is an effective statistical method for analyzing data sets that have at least one independent variable to determine the outcome.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Using the logistics function, we can measure the relationship between one or more independent variables and the categorically dependent variable by estimating probabilities. The result shows the probability that the record belongs to a given class. Using this method, the output can be interpreted as a probability. Its advantage is that it is resistant to noise. The disadvantage is that it requires more data for stable operation, and overfitting may occur during use, [9]. The implementation of logistic regression in Python is as follows.

```
from sklearn.linear_model import LogisticRegression
logreg_model=LogisticRegression()
logreg_model.fit(X_train,y_train)
logreg_prediction=logreg_model.predict(X_test)
```

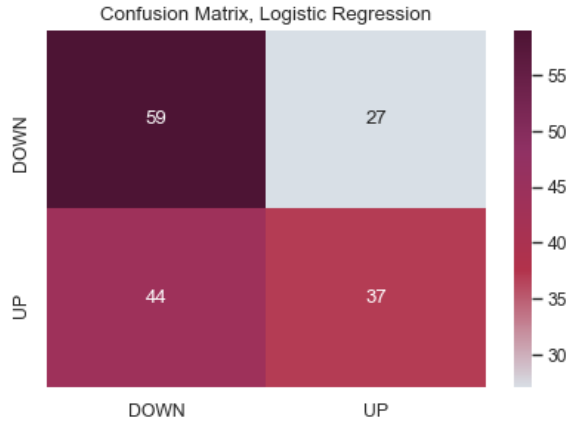

Figure 6. Confusion matrix of the Logistic Regression classifier

In Figure 6. we can see that the behavior of the Logistic Regression classifier is different from the MLP. It identifies more price decreases correctly. But we can also see that it classifies more records to the down class incorrectly as well.

2.2.3 K Nearest Neighbor (KNN)

The KNN algorithm represents each training record as a data point in the *n*-dimensional space, where *n* is the number of attributes of the record, [8]. During classification, the input of the model always consists of *k* nearest neighbors of the record to be classified. The proximity metrics, which are widely used to find the nearest neighbors, are Euclidean distance or Manhattan distance. In the KNN classification, the output will be the most common class among the the *k* nearest neighbors. The advantage of the model are the easy feasibility and fast training, while the disadvantage is slow testing due to slow distance processing. Tracking all items and neighbors also requires a lot of storage space. Finally, the method is noise sensitive, [9].

```
from sklearn.neighbors import KNeighborsClassifier
KNN_model=KNeighborsClassifier(n_neighbors=k)
KNN_model.fit(X_train, y_train)
KNN_prediction=KNN_model.predict(X_test)
```

When implementing the model, it is necessary to specify the parameter *k*. The optimal *k* value is determined using the Elbow Method. The method examines the classification error for several *k* parameters. The test result gives the optimal parameter *k* for which the experienced error is minimal.

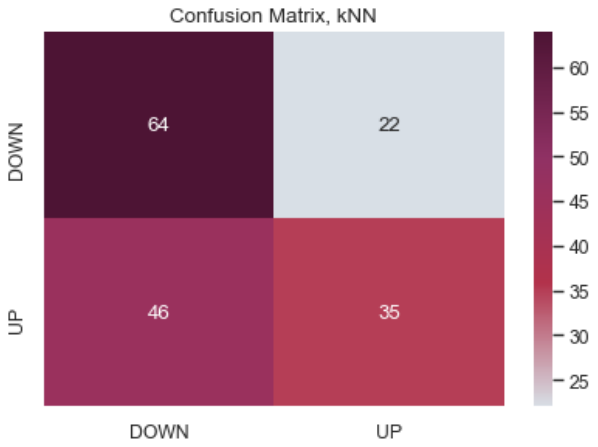

Figure 7. Confusion matrix of KNN classifier

Regarding the confusion matrix, we experience similar behavior as in the previous classes. However, the number of correctly classified price reductions is higher than for previous classifiers.

2.2.4 Support Vector Machine (SVM)

The basic idea of the Support Vector Machine (SVM) is to find the hyperplane separating the two classes with the widest margins, [13]. The points closest to the optimal hyperplane are called support vectors. SVM uses these support vectors to find the optimal hyperplane that provides the classification. To classify nonlinear data, the model converts the

original data to a higher dimension using nonlinear kernel functions such as the polynomial kernel and the radial basis function kernel (RBF). The advantage of SVM is that it can classify complex problems using different kernels. Teaching the model can be time consuming.

```
from sklearn import svm
svm_model=svm.SVC(kernel='rbf', gamma=gamma, C=Cgamma)
svm_model.fit(X_train, y_train)
svm_prediction=svm_model.predict(X_test)
```

During the simulation we experienced extremely long runtime while using the model with polynomial kernel. Thus, in the study we involved models with linear and RBF kernel. To optimize the model, we tested several *gamma* and *C* parameters. Notice the different behavior caused by the different kernels in Figure 8.

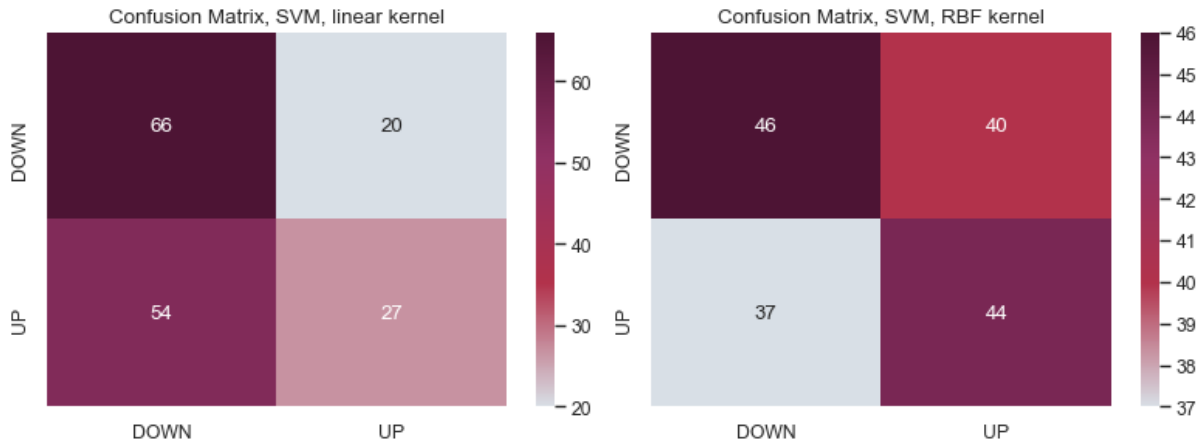

Figure 8. Confusion matrix of SVM classifiers with linear (on the left) and RBF kernel (on the right)

2.2.5 Decision Tree and Random Forest

The basic idea of the decision tree is to divide the data into recursive subsets in order to achieve that each subset contains nearly homogeneous records. The model forms a tree structure, each vertex of which represents a test for an attribute. The criterion for selecting the split is to separate the data set as much as possible. The edges leading out of the vertices represent the possible outcomes of the decisions. Elements of the decision tree letter determine classification, [11]. The advantage of this method is that it is easy to interpret. It handles both numeric and nominal attributes. However, the method works well with just a few significant variables. Its performance may be weaker if there are a number of complex interactions in the database, [9].

```
from sklearn.tree import DecisionTreeClassifier
decisiontree_model=DecisionTreeClassifier()
decisiontree_model.fit(X_train,y_train)
decisiontree_prediction=decisiontree_model.predict(X_test)
```

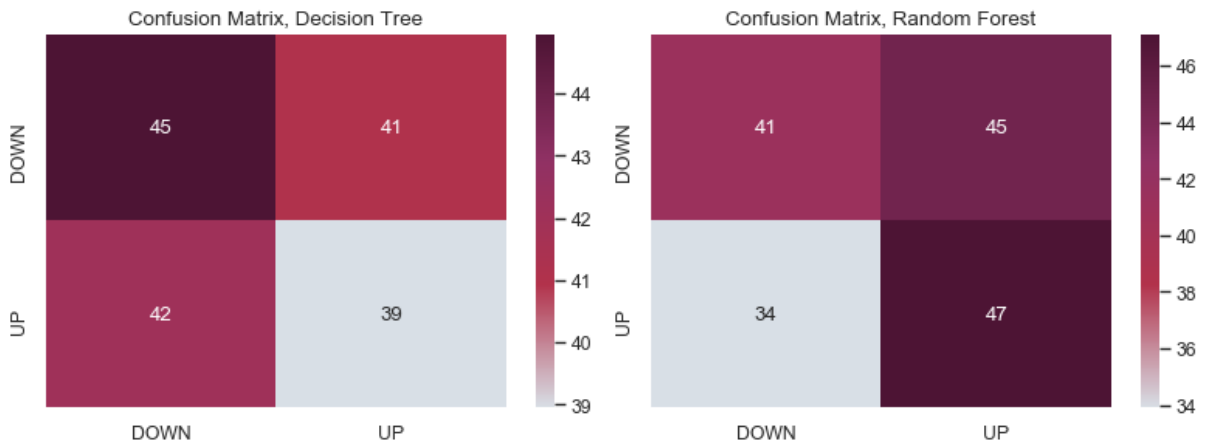

Figure 9. Confusion matrix of Decision Tree (on the left) and Random Forest (on the right)

The random forest classifier performs the classification based on predictions given by several different decision trees. So, the random forest (RF) model consists of a combination of decision trees. It improves the classification performance of a single tree model by aggregating the outputs of multiple decision trees and applies randomization in the selection of data nodes when constructing decision trees. In this study we implemented the random forest with 100 estimators. The disadvantage is that the model results in longer runtime and higher storage requirements compared to the decision tree.

```
from sklearn.ensemble import RandomForestClassifier
randomforest_model=RandomForestClassifier(n_estimators=100)
randomforest_model.fit(X_train, y_train)
randomforest_prediction = randomforest_model.predict(X_test)
```

In Figure 9. we can notice the differences between the behavior of a single decision tree and the random forest model.

2.2.6 Naive Bayes

The Naive Bayes classifier assumes attribute values that are independent of each other and there are no dependency relationships between the columns in the database, [9]. Suppose there are *m* classes denoted by C_1, C_2 ... C_m. For an X record, the model predicts that X belongs to a given C_i class as follows.

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

Calculating with conditional probabilities, find the class for which this probability is highest. The advantage of this method is that it is fast and easy to calculate. The disadvantage is that the assumption of the Naive Bayes method is often not met in the data set, and in the absence of independence of the variables, the classifier does not give an accurate result.

```
from sklearn.naive_bayes import GaussianNB
naive_model=GaussianNB()
naive_model.fit(X_train,y_train)
naive_prediction=naive_model.predict(X_test)
```

As we can notice in Figure 10., the performance of the method seems to be poorer. This can be explained by the fact that in the covariance matrix we detected a strong relationship between several variables, so the condition of the classification model is not fulfilled.

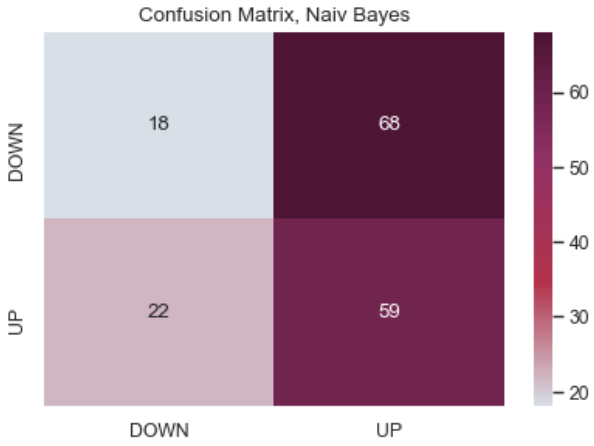

Figure 10. Confusion matrix of Naive Bayes classifier

2.3 Evaluation of the classification result

In order to select the most appropriate classifier for the implementation of the trading strategy, a comparative analysis of the described classification methods is performed. As we have seen when performing classifications and examining confusion matrices, classifying a data set is a challenge for different machine learning models. In the following, we examine the performance of the described classification procedures with different metrics to select the best model or models to be used.

2.3.1 Accuracy

After examining the confusion matrixes, we calculate the classification accuracy of each model. The accuracy of the classification model expresses the percentage of test records that the model classifies correctly during testing.

$$Accurecy = \frac{Number\ of\ correct\ classifications}{Total\ number\ of\ classifications}$$

In the 100 experiments performed in the study, the accuracy is recorded in each case, and then their arithmetic mean is taken for each classifier. The arithmetic mean is sensitive to outliers, so to avoid information bias, we examined the values of the accuracies experienced during the study and their standard deviations. We find that the standard deviation of the accuracies of each classification method is small in all cases. Figure 11 shows the average accuracy of the different classification models, where the size and color of the markers also represent the accuracy of the models.

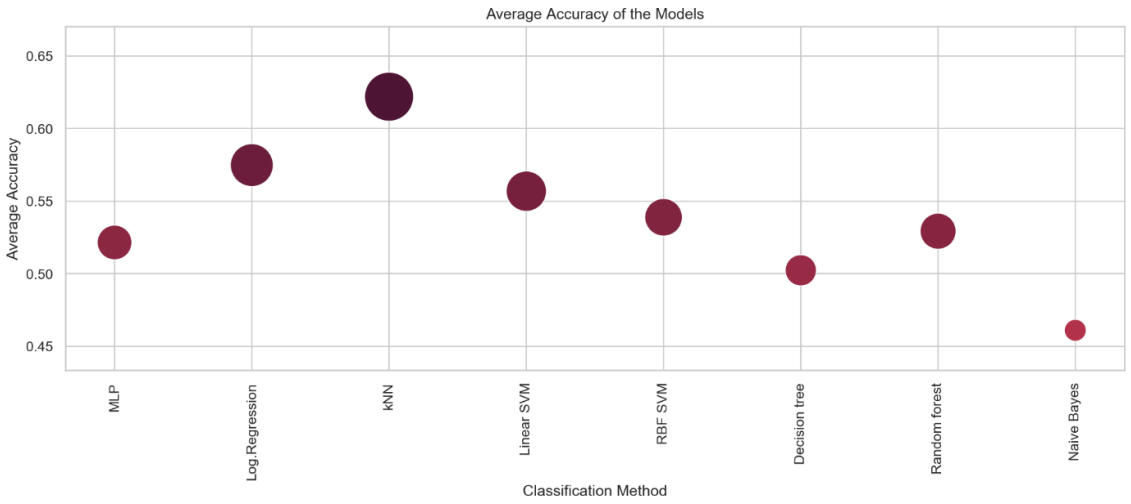

Figure 11. Average accuracy of the classification models during the study

The average accuracy of the models during the study was 53.5%. The lowest mean accuracy is found for the Naive Bayes classifier, at 46.1%. The highest accuracy was produced by the KNN model, its value was 62.3%. In earlier stages of the study, we found mean accuracies a few percent higher. The highest accuracy was shown in all cases by the KNN classifier. Before the significant exchange rate crash on May 10, shown in Figure 1, the accuracy of the KNN classifier approached 65%.

The accuracy of the models is all above 50% except for the Naive Bayes method. This means that by using classification models, we have a slightly better chance of correctly predicting the direction of exchange rate change than if we randomly guessed.

2.3.2 Precision

When examining precision, we analyze the accuracy with which each classification model predicted records for different classes.

$$\text{Precision} = \frac{\text{Number of records correctly classified in the given class}}{\text{Total number of records classified in the given class}}$$

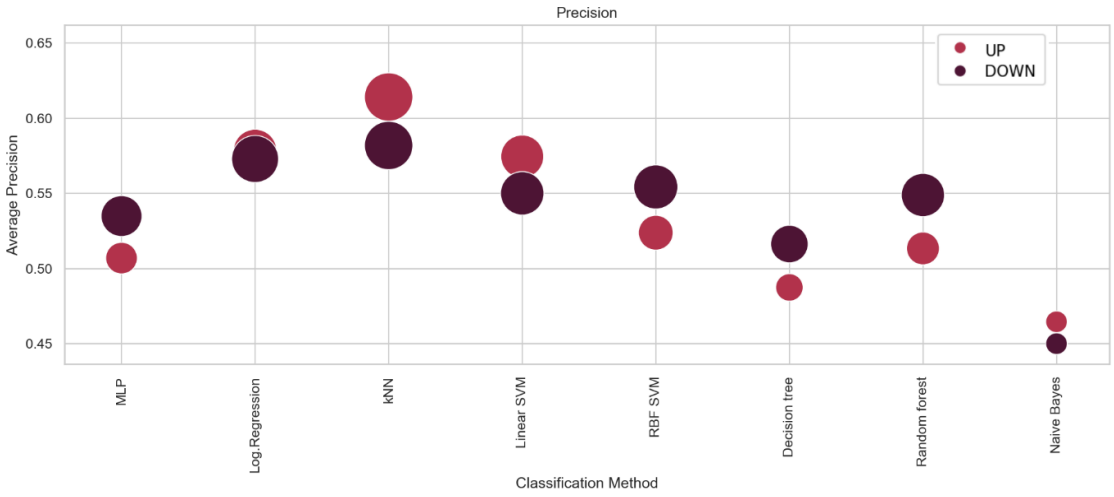

Figure 12. Precision of the 'up' and 'down' classes in the different classification models.

The precisions experienced in the study are recorded for all classes. Their arithmetic mean is shown in Figure 12. The precision of an increase in the exchange rate is shown in light and a decrease is shown in dark. Of the classifiers, KNN, Logistic Regression, and Linear SVM perform best in both classes. These classifiers are able to classify the 'up' class with slightly greater precision than the 'down' class. For other classifiers, this is mostly true in the other way.

2.3.4 Recall

Recall measures how many of the records in a given class in the database were correctly classified by the model.

$$Recall = \frac{\text{Number of records correctly classified in a given class}}{\text{Total number of records in a given class}}$$

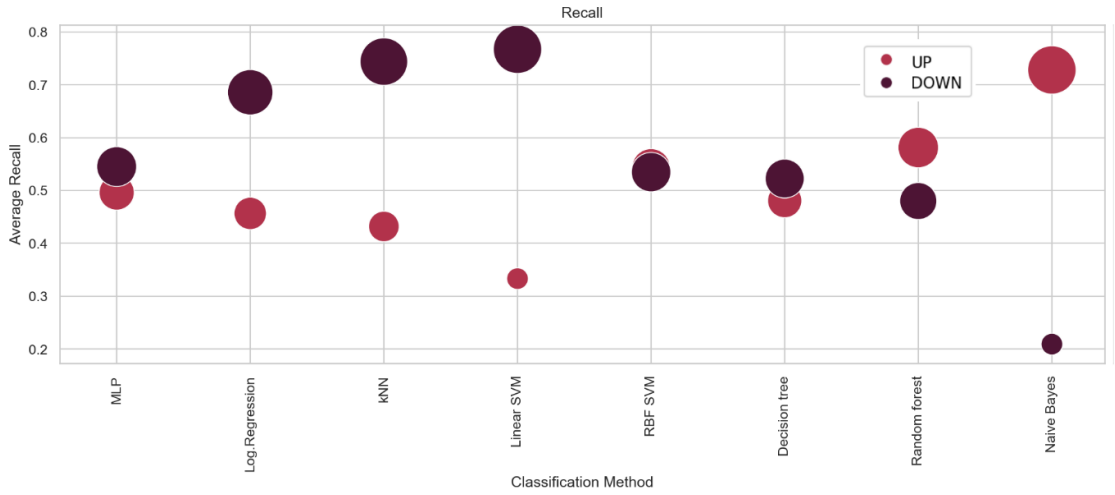

Figure 13. Recall of the classes in the classification models.

Figure 13. shows that linear SVM, KNN, and Logistic Regression models are able to identify individuals in the 'down' class above 70%. However, only a small percentage of exchange rate increases were identified by the models. An exception to this is the Naive Bayes classifier, but its confusion matrix also explains the phenomenon: the classifier has classified most of the records, rightly and wrongly, into the 'up' class.

2.3.5 F1-score

The F1 score is the harmonic mean of precision and recall, which is used to characterize classification algorithms with a single metric.

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

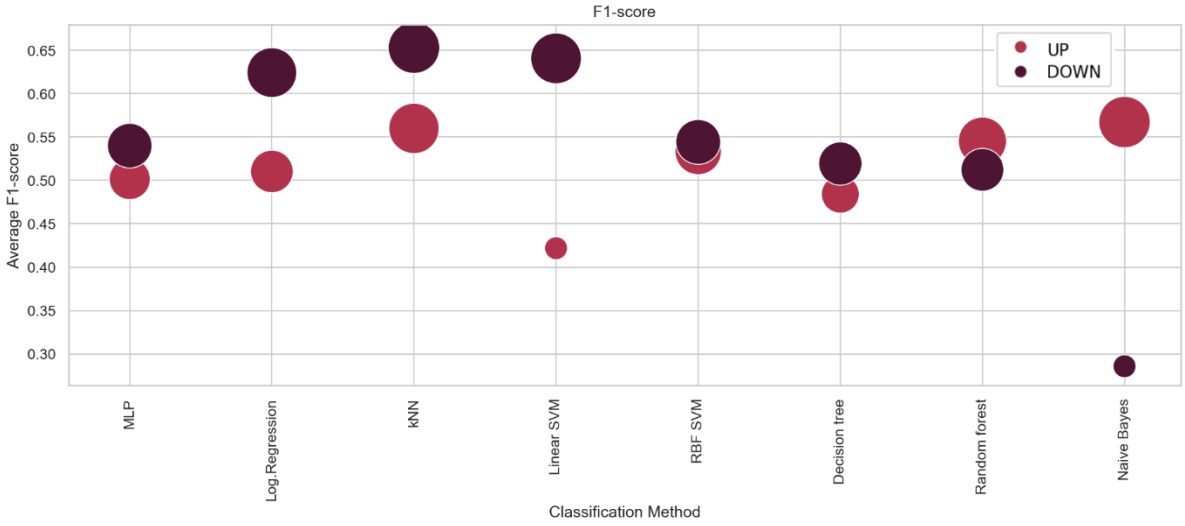

Figure 14. F1-score of the classes in the classification models.

The F1-score shows that the decline in the exchange rate is best classified by the KNN, followed by the linear SVM and logistic regression. Price increases are ranked by Naive Bayes, random forest, and KNN with the highest F1 scores. Considering the confusion matrices, we can conclude that Naive Bayes and the random forest classified most of the records into this category, so if we based our trading strategy on these classifiers, we would engage in very risky trading. The operation of the KNN classifier proves to be somewhat more balanced.

2.3.6 Runtime

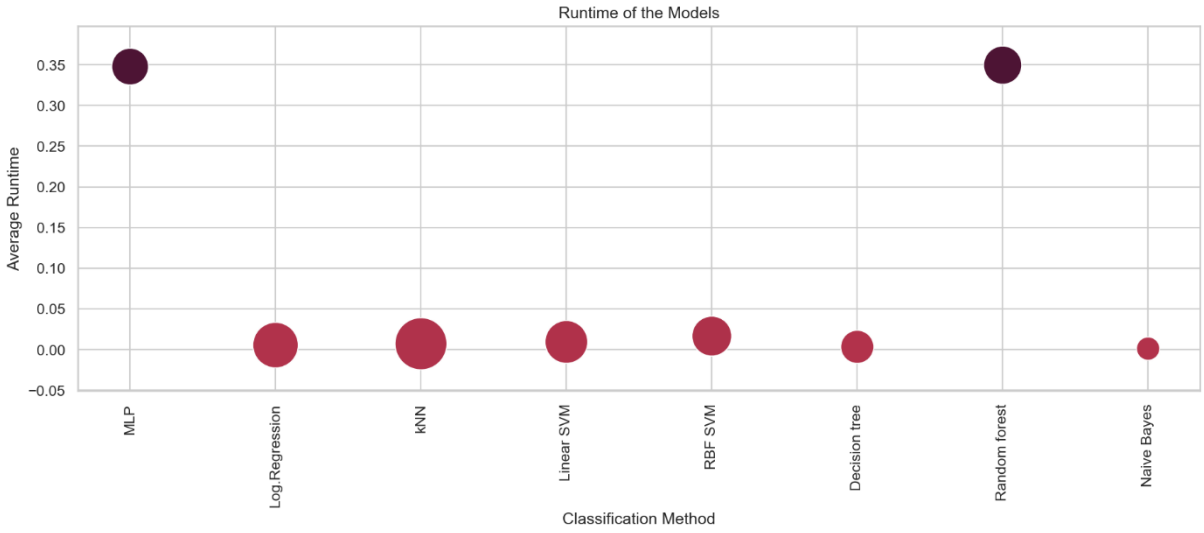

Figure 15. Runtime of the classification models

Classification models are at the core of our trading strategy, since they will decide on the trading. Because the exchange rate is constantly changing, the run time of classification, prediction, and decision can be critical to the successful execution of a trade. During the study, the running time from the definition of each model to the classification of the test records was recorded in each experiment, the average of which is shown in Figure 15. We find that MLP and random forest classifiers require significantly longer run times than the other methods. In

Figure 15, the average accuracy of each method is shown by the size of the points, helping the decision to find the most accurate and fastest possible model to implement our strategy.

Since in the examined metrics the KNN classifier showed outstanding performance in classifying both exchange rate decrease and increase, as well as its running time is promisingly short, in this study we define our trading strategy based on KNN classifier.

2.3 Trading strategy and backtesting

In this section we describe our trading strategy based on the previously described machine learning method. We also present backtesting of the created model in different market situations to examine the performance of it.

2.3.1 Trading strategy

The core aim is implement cryptocurrency trading strategy, based on the decision of a machine learning model, which makes a trading decision in every 6 hours.

To accomplish this task, the first step is to prepare the machine learning model. According to the previously presented comparative study of the different classification models applied on the data of bitcoin exchange rate, we use KNN classification method.

The KNN model will be trained with some historical data of the bitcoin price. Then, in every six hours, the current price of the bitcoin is downloaded as the close price of the last 6-hour period data. Based on this 6-hour period open, close, high, low and volume values, the other attributes, such as moving average, are calculated as presented in the Section 2.1. After scaling the values and feature selection, the obtained record is given to the classification model as an input. The model returns the output indicating if the bitcoin price six hours later is predicted to be higher or lower than the current bitcoin price. If the exchange rate is predicted to increase, we buy bitcoin right now. Otherwise, if the exchange rate is predicted to decrease, we do short selling, so we sell bitcoin right now and buy it six hours later. Afterwards, six hours later we evaluate the trade according to the real price change. We assign the label ‘up’ or ‘down’ to the record and load this information into the training dataset of the model. This way the KNN model is continuously learning by itself with the aim of improving the classification performance.

```
data_train,move_train=get_binance_data(symbol, interval,
        train_start_time,train_end_time)
data_train,move_train=calculate_indicators(data_train)
scaled_data_train=scale(data_train)
X=feature_selection(scaled_data_train)
k=get_k(X,move_train)
X_train=X
y_train=move_train
model=KNeighborsClassifier(n_neighbors=k)
model.fit(X_train,y_train)
```

The code above shows the initialization of the trade. We download the historical training data, calculate the indicators and the 'move' target variable. We perform scaling and feature selection.

Then, define the classification input data set and the target variable. We determine the optimal parameter k, thus creating the model and fitting it to the entire historical data set.

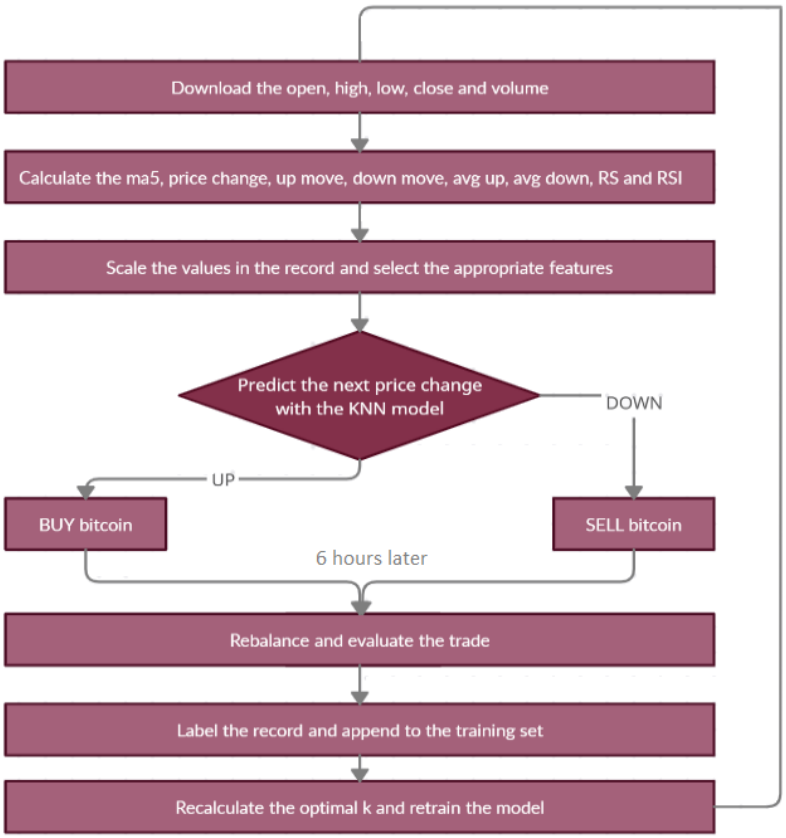

Figure 16. Trading strategy.

```
data_simulation=get_binance_bars(symbol, interval,
        test_start_time,test_end_time)
data_simulation,move_simulation=calculate_indicators(data_simulation)
```

The backtest data is also downloaded. To simplify the test, we calculate the indicators immediately. We could do this later when processing the data line by line, just before the trading decision, as will happen in a live environment. We do not attach much importance to this during testing, it does not affect the result. But it should be noted that in a live environment, calculating the indicators at runtime will mean a few milliseconds extra between downloading data and trading.

```
start=100000
btc=0
success=0
cash=start
```

Before the trading cycle, we initialize the amount of money for trading, which is recorded in the ‘start’ variable. In this case, this is the $ 100,000 provided by the Binance demo account. Initially, we set the number of bitcoins to zero. Also, for statistical reporting, we will record the number of the successful trades in a variable. Finally, we set the ‘cash’ variable to all of our money is currently available in dollars. After these initialization steps, the simulation can begin.

The backtest is run on the rows of the test dataset. Depending on where the test stops and if the class label of the very last record is known, we run the loop to the end of the test dataset or until the last record with known target attribute. We retrieve the current exchange rate of the bitcoin, which is shown by the current 'close' value of the test set. Scale the values in the current test data set. Then, we collect the values of the attributes selected by the feature selection into the variable 'sample'. The classification model will receive this sample as input.

```
for i in range(len(data_simulation)-1):
    btc_price=data_simulation['close'][i]
    scaled_sample=scaler.transform(np.asarray(data_simulation.iloc[i])
        .reshape(1, -1))[0]
    sample=[]
    for j in range(13):
        if feature.get_support()[j]==True:
            sample.append(scaled_sample[j])
        ...
```

After preparing the input, we call the model. We make the trading decision based on the prediction of the KNN model. If the model returns 1 as an output, it indicates that the exchange rate will rise in the next market moment, i.e. in 6 hours in this example. If the model predicts 0 as the output, the exchange rate will decrease after 6 hours.

```
        ...
        prediction=model.predict(np.asarray(sample).reshape(1,-1))[0]
        if prediction==1:
            btc = cash * 0.5 / btc_price
            cash = 0.5 * cash
        else:
            btc = -cash * 0.5 / btc_price
            cash = 1.5 * cash
        ...
```

Therefore, in the case of the model predicts 1, it is now worth buying bitcoin, as we hope for an increase in the exchange rate. So, the bitcoin we just buy right now will worth more in six hours, consequently, we can increase our wealth. As a result, the strategy at this point is to buy bitcoin for half the cash we have.

If the output predicted by the model is 0, we expect a decrease in the exchange rate. So, if we sell a certain amount of bitcoin at the current rate, we can buy that particular amount cheaper in six hours. This is recorded as a negative number of ‘btc’. In this case as well, we invest half of our money in the trade. Our wealth is growing at this moment. In six hours, however, we are obliged to purchase the amount of bitcoin sold, to offset the negative number of bitcoins.

```
        ...
        btc_newprice=data_simulation['close'][i+1]
        cash=cash+btc*btc_newprice
        btc=0
        if (btc_newprice>btc_price and prediction==1) or
                  (btc_newprice<btc_price and prediction==0):
            success+=1
        ...
```

After completing the trade, we can move to the next market moment in the simulation. So, after six hours, we retrieve the current bitcoin rate contained in the 'close' attribute of the next row in the backtest dataset. We rebalance our properties. If our tactic were to buy, we will

now sell that quantity at the current price. If we have made a short selling, we will buy the quantity sold at the current exchange rate. Furthermore, we evaluate the trade. If the bitcoin price has increased and the model predicted 1, or if the bitcoin price has decreased and the model predicted it well, we increase the number of successful trades. We could have checked the same using 'move_simulation' variable, which already includes this evaluation, but this above presented way is a bit more lifelike solution in the simulation.

```
...
X_train=np.append(X_train, np.asarray(sample).reshape(1, -1), axis=0)
y_train=np.append(y_train, data_simulation.iloc[i]['move'])
k=get_k(X_train, y_train)
model=KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
```

The last step is to feed the experience gained back into the model. We expand the list of training inputs and training outputs with the current record and the direction of the actual price change. We re-optimize the parameter k and then build and fit the model to the entire training data set. Then, we proceed to run the next cycle in the loop and continue the simulation with the next line.

2.3.2 Backtesting in ordinary situation

The backtesting of the model was performed with different timings. This example examines the behavior of the model between the 1st of May and the 10th of May 2021, taught from the 1st of January 2021 to 30th of April 2021.

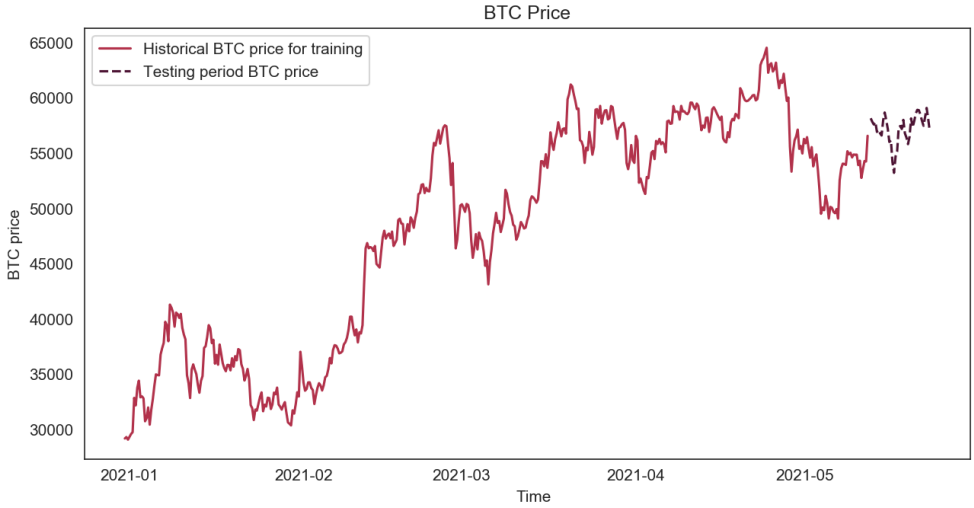

Figure 17. Bitcoin prices for backtesting between 1 May and 10 May, trained from 1 January to 30 April 2021.

The trading steps performed by the strategy is logged. The model operates the following trades. We process 40 trades over the 10-day period. First, we observe a period of loss, which is roughly balanced by the alternation of correct and erroneous predictions. Correct and incorrect trades alternate roughly periodically during the test. Then comes a profitable period and profits start to grow. It can be seen in the Figure 19.

Figure 18. shows the trading log. The trade serial number is followed by the trading decision to buy or sell. The current price of bitcoin is then given in bracket. This is followed by the amount of money invested. We evaluate if the decision of the model was correct, or it made

a mistake. This is followed by the current price of our investment after 6 hours, and in brackets the current exchange rate after 6 hours. Finally, our account balance is presented.

```
 1. SELL: ( 57694.27 ) --  50000.00 -->...mistake...--> -50381.47  -- ( 58134.44 ) -----  99618.53
 2. SELL: ( 58134.44 ) --  49809.27 -->...correct...--> -49537.20  -- ( 57816.90 ) -----  99890.60
 3. SELL: ( 57816.90 ) --  49945.30 -->...correct...--> -49732.01  -- ( 57569.99 ) ----- 100103.89
 4. SELL: ( 57569.99 ) --  50051.95 -->...mistake...--> -50252.24  -- ( 57800.37 ) -----  99903.60
 5. BUY:  ( 57800.37 ) --  49951.80 -->...mistake...-->  49175.48  -- ( 56902.07 ) -----  99127.28
 6. BUY:  ( 56902.07 ) --  49563.64 -->...correct...-->  49727.33  -- ( 57090.00 ) -----  99290.97
 7. SELL: ( 57090.00 ) --  49645.48 -->...correct...--> -49375.90  -- ( 56779.99 ) -----  99560.55
 8. SELL: ( 56779.99 ) --  49780.28 -->...correct...--> -49603.37  -- ( 56578.21 ) -----  99737.46
 9. SELL: ( 56578.21 ) --  49868.73 -->...mistake...--> -51068.02  -- ( 57938.86 ) -----  98538.17
10. SELL: ( 57938.86 ) --  49269.08 -->...mistake...--> -49880.30  -- ( 58657.63 ) -----  97926.95
11. SELL: ( 58657.63 ) --  48963.47 -->...correct...--> -48379.97  -- ( 57958.60 ) -----  98510.45
12. SELL: ( 57958.60 ) --  49255.23 -->...correct...--> -48584.53  -- ( 57169.39 ) -----  99181.15
13. SELL: ( 57169.39 ) --  49590.58 -->...correct...--> -48566.62  -- ( 55988.95 ) ----- 100205.10
14. BUY:  ( 55988.95 ) --  50102.55 -->...correct...-->  50240.78  -- ( 56143.42 ) ----- 100343.33
15. BUY:  ( 56143.42 ) --  50171.67 -->...mistake...-->  48405.31  -- ( 54166.82 ) -----  98576.98
16. BUY:  ( 54166.82 ) --  49288.49 -->...mistake...-->  48408.75  -- ( 53200.01 ) -----  97697.24
17. BUY:  ( 53200.01 ) --  48848.62 -->...correct...-->  50219.58  -- ( 54693.10 ) -----  99068.20
18. SELL: ( 54693.10 ) --  49534.10 -->...mistake...--> -49910.73  -- ( 55108.95 ) -----  98691.58
19. BUY:  ( 55108.95 ) --  49345.79 -->...correct...-->  51290.99  -- ( 57281.33 ) ----- 100636.78
20. SELL: ( 57281.33 ) --  50318.39 -->...mistake...--> -50454.35  -- ( 57436.11 ) ----- 100500.81
21. SELL: ( 57436.11 ) --  50250.41 -->...correct...--> -50014.99  -- ( 57167.03 ) ----- 100736.23
22. BUY:  ( 57167.03 ) --  50368.11 -->...correct...-->  51091.45  -- ( 57988.00 ) ----- 101459.56
23. SELL: ( 57988.00 ) --  50729.78 -->...correct...--> -49670.36  -- ( 56777.00 ) ----- 102518.98
24. BUY:  ( 56777.00 ) --  51259.49 -->...mistake...-->  50913.42  -- ( 56393.68 ) ----- 102172.91
25. BUY:  ( 56393.68 ) --  51086.46 -->...mistake...-->  50544.44  -- ( 55795.36 ) ----- 101630.90
26. SELL: ( 55795.36 ) --  50815.45 -->...mistake...--> -51425.09  -- ( 56464.74 ) ----- 101021.26
27. BUY:  ( 56464.74 ) --  50510.63 -->...correct...-->  51991.34  -- ( 58119.99 ) ----- 102501.97
28. SELL: ( 58119.99 ) --  51250.99 -->...correct...--> -50540.91  -- ( 57314.75 ) ----- 103212.04
29. BUY:  ( 57314.75 ) --  51606.02 -->...correct...-->  52113.14  -- ( 57877.97 ) ----- 103719.16
30. SELL: ( 57877.97 ) --  51859.58 -->...mistake...--> -52467.19  -- ( 58556.09 ) ----- 103111.56
31. SELL: ( 58556.09 ) --  51555.78 -->...mistake...--> -51857.68  -- ( 58898.98 ) ----- 102809.66
32. SELL: ( 58898.98 ) --  51404.83 -->...correct...--> -51372.60  -- ( 58862.05 ) ----- 102841.89
33. SELL: ( 58862.05 ) --  51420.95 -->...correct...--> -51057.94  -- ( 58446.51 ) ----- 103204.90
34. SELL: ( 58446.51 ) --  51602.45 -->...correct...--> -51126.55  -- ( 57907.49 ) ----- 103680.80
35. SELL: ( 57907.49 ) --  51840.40 -->...correct...--> -51438.66  -- ( 57458.73 ) ----- 104082.54
36. SELL: ( 57458.73 ) --  52041.27 -->...mistake...--> -52749.64  -- ( 58240.84 ) ----- 103374.17
37. BUY:  ( 58240.84 ) --  51687.09 -->...correct...-->  52421.21  -- ( 59068.05 ) ----- 104108.30
38. SELL: ( 59068.05 ) --  52054.15 -->...correct...--> -51175.17  -- ( 58070.63 ) ----- 104987.28
39. SELL: ( 58070.63 ) --  52493.64 -->...correct...--> -51512.54  -- ( 56985.30 ) ----- 105968.38
40. SELL: ( 56985.30 ) --  52984.19 -->...correct...--> -51897.12  -- ( 55816.14 ) ----- 107055.45
```

Figure 18. Trading log from the 01 May to 10 May 2021.

During the tested period, our model correctly predicted 25 out of 40 cases. This represents an accuracy of 62.5%. We increased our starting capital from $ 100,000 to $ 107055.45, so the model generated a profit of $ 7055.45 over the 10-day period. This represents a profit of 7.06%.

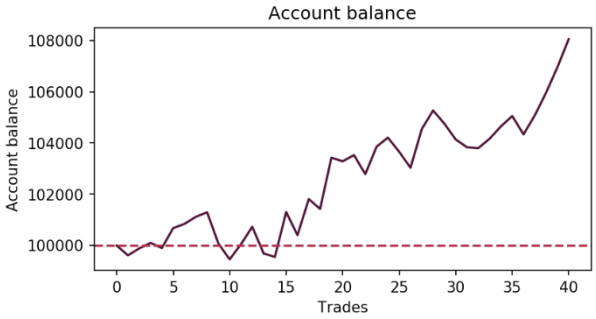

Figure 19. Account balance from 01 May to 10 May 2021.

2.3.3 Backtesting on the unexpected period of exchange rate fall

During the period of the study, an unexpectedly large exchange rate fall occurred. A massive wave of sales swept across the cryptocurrency market, with bitcoin falling 10 to 40 percent in a matter of hours and the overall market value of the cryptocurrency market falling by hundreds of billions of dollars. One of the main triggers for the sharp fall was that three Chinese financial regulators again warned financial service providers under their control not to provide cryptocurrency services. All in all, it seems that in one of the world's largest economies, China, which is also dominant in terms of global cryptographic mining and retail cryptographic investment, there are no signs of legalization, but of further tightening and prohibition, which may have scared investors, hence the strong stock market reaction.

So, in this subsection, we will test our trading strategy in a really unexpected situation. We teach the model between January 1st and May 9th. The simulation will start at dawn on 10 May 2021 and run until the moment of writing the dissertation, 24 May 2021, morning.

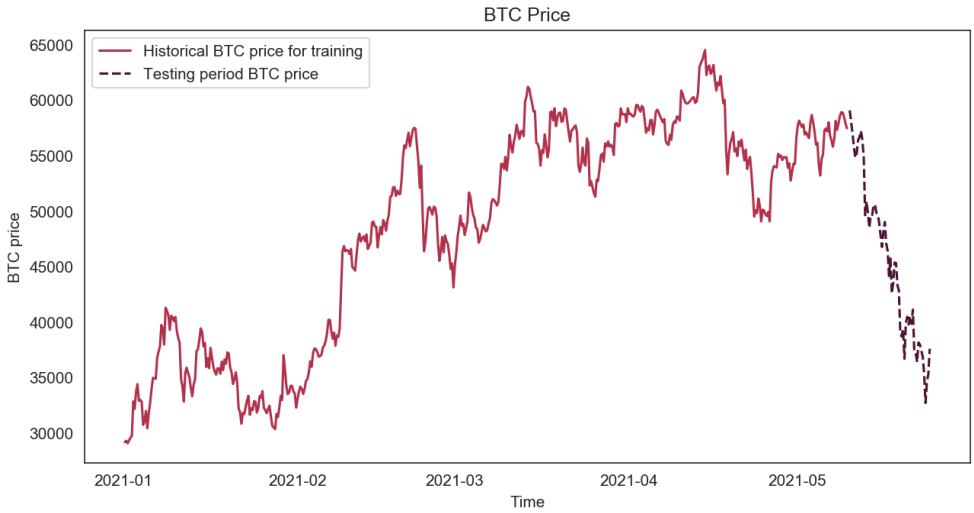

Figure 20. Bitcoin prices for backtesting between 10 May and 24 May, trained from 1 January to 9 May 2021.

During this period, we have about 60 trades. The trading diary shows that the prediction accuracy of the model fell to 48.3%, for 28 times out of 58 the prediction of the direction of exchange rate change was correct. However, responding to the unexpected market situation is promising. Even in such an extreme case, the loss generated by the strategy does not fall below 10%. The output shows that the model learns from the mistakes and reduces the loss in response. We close the two-week extreme period with a loss of 1.5%.

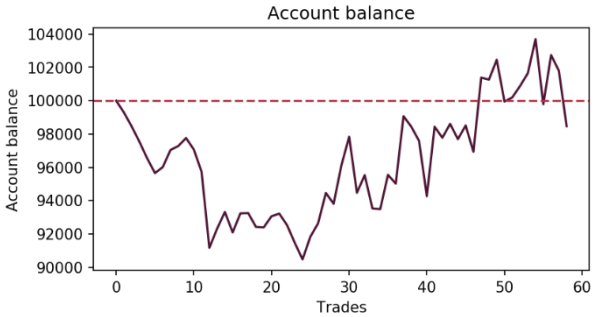

Figure 21. Account balance during the unexpected market crash in 2021 May.

2.3.4 Backtesting in changeable situation

In this subsection, we test the performance of the trading strategy over the entire May period in 2021. The model is taught from the 1st of January to the 30th of April 2021. The simulation is started at dawn on the 1st of May and run until the time of writing the dissertation, the morning of May 24, 2021. This period starts with a normal price fluctuation, which is well modeled by the learning period. This is followed by a two-week market crash. In these two periods, we have already checked the behavior of our strategy separately by simulation. We now examine how the strategy behaves in a changing environment with a normal and extraordinary market situation. This study may even provide a basis for considering long-term trading, as it tests the model in both normal and extreme situations.

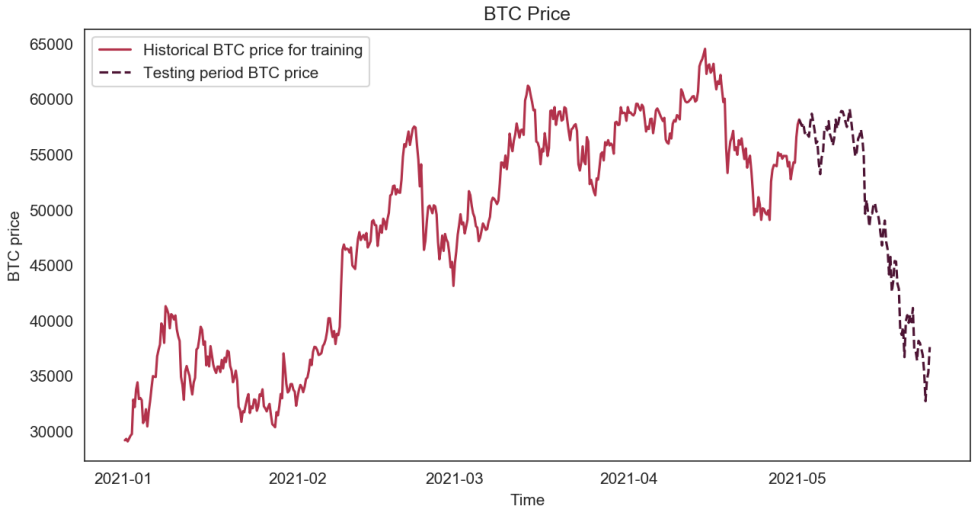

Figure 22. Bitcoin prices for backtesting between 1 May and 24 May, trained from 1 January to 30 April 2021.

The test period includes 92 trading points, of which the model provides prediction of the direc tion of price change in 54 cases. This represents an accuracy of 58.7%. During the May period , the initial capital of $100,000 was increased to $106,498.7. The profit is therefore $6,498.7, which represents a profit of about 6.5%.

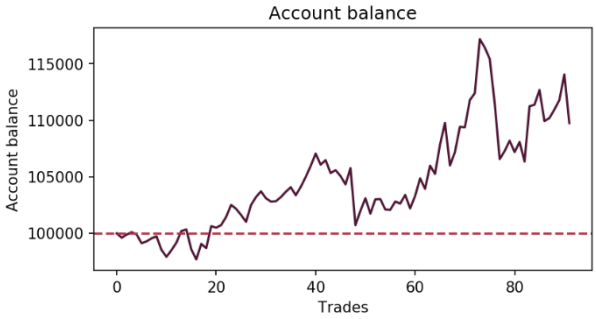

Figure 23. Account balance in 2021 May.

The strategy starts with a loss first, then the model corrects itself and starts making a profit. Figure 23. shows well the result of the unexpected collapse of the market in the decline in profits. However, the model responds to change and becomes profitable again. For exchange rate fluctuations, we sometimes lose out on profits, but the chart suggests that the model is able to correct its mistakes relatively quickly. Examining this requires longer-term study.

2.4 Backtesting the strategy on different assets

Testing of the trading strategy is also performed in other currencies. Some of both cryptocurrencies and traditional exchange products are included in the study. During testing, the initial models are taught with historical real market data from January 1 to April 30, 2021, and then test trading begins on May 1 and closes on May 24, dawn.

In case of cryptocurrencies, the data is downloaded from the Binance. In case of other assets, the data is downloaded by the Yahoo Finance, yfinance Python package.

2.4.1 Backtesting on Ether

Ether is the cryptocurrency built on top of the open source Ethereum blockchain, which runs smart contracts, [2]. While bitcoin is meant to be a unit of currency, ether acts as a fuel that allows smart contracts. Unlike bitcoin, Ether's supply is not capped.

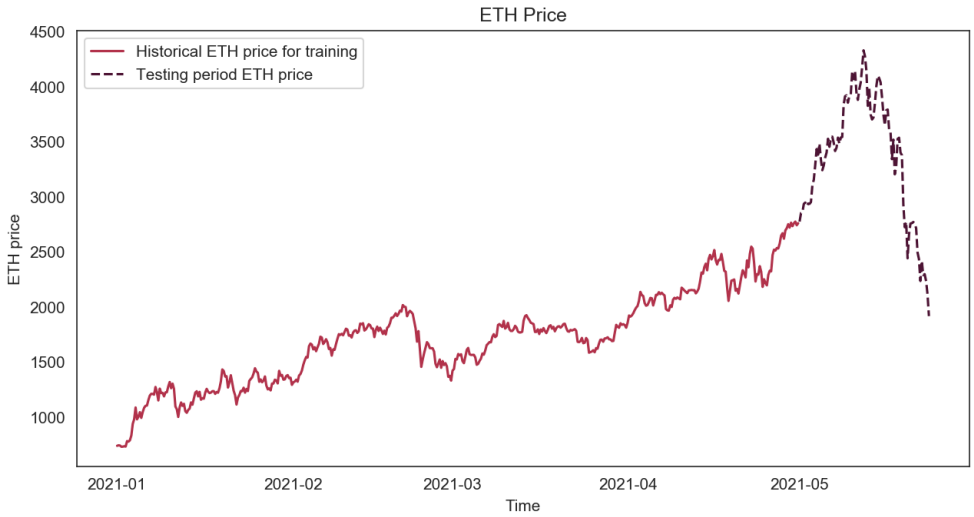

Figure 24 Ether prices for backtesting between 1 May and 24 May, trained from 1 January to 30 April 2021.

Observing the exchange rate, we find that the exchange rate of ETH, like bitcoin, shows extreme behavior during the test period. First, there was an extreme rise in prices, followed by an extreme fall in the exchange rate.

From the 92 trading steps of the test period, the model performs correct trading in 53 cases. This means that the KNN model correctly predicts the direction of exchange rate movements with 57.6% accuracy. Following the rise in profits, due to the unexpected fall in prices, the strategy eventually closes with a loss of -1.7%. However, Figure 25. suggests that after the loss-making period, the model probably learns from its mistakes, and we may expect a profitable period again.

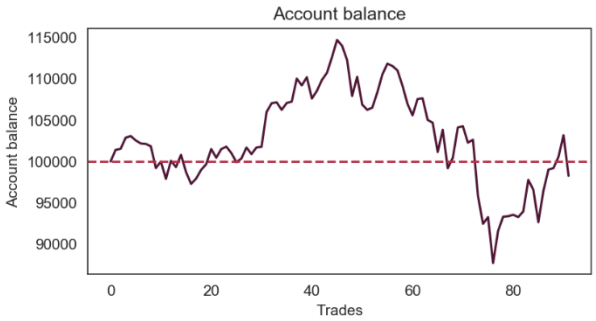

Figure 25. Account balance in 2021 May

2.4.2 Backtesting on Filecoin

Filecoin is an open source, public cryptocurrency and digital payment system designed as a blockchain-based cooperative digital storage and data retrieval method. It is made by Protocol Labs and builds on top of the InterPlanetary File System, allowing users to rent unused hard drives, [12].

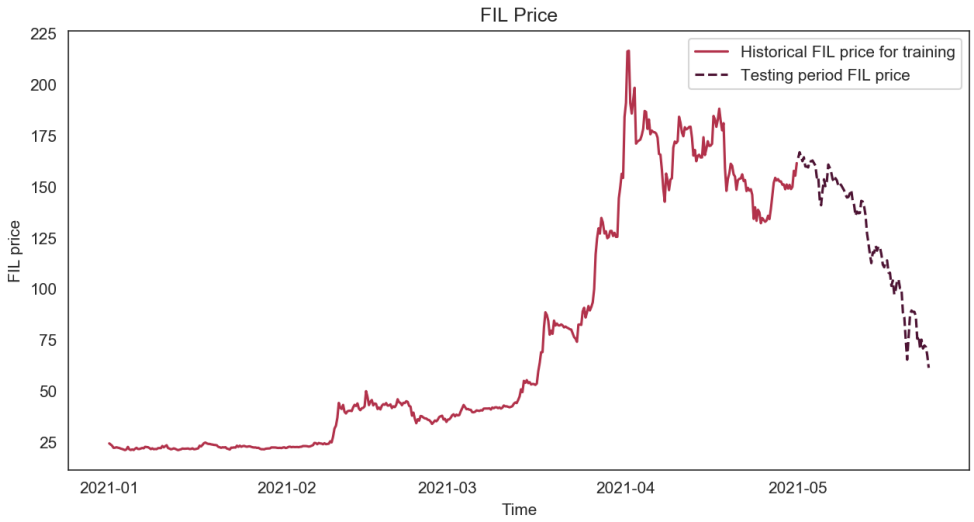

Figure 26. Filecoin prices for backtesting between 1 May and 24 May, trained from 1 January to 30 April 2021.

Figure 26 shows that after a relatively stagnant period, the FIL exchange rate started to rise strongly in March 2021. The training period of the model ends with a price fluctuation and a decline. The test period, like the cryptocurrencies studied earlier, includes a declining phase.

The model correctly predicts 49 times out of 92 over the test period. This represents an accuracy of 53.3%. At the same time, due to the favorable development of exchange rates, the strategy closes with a profit of 28.9%. This experience warns us not to base our profit expectations solely on the predictive accuracy of the classification model at the core of the strategy.

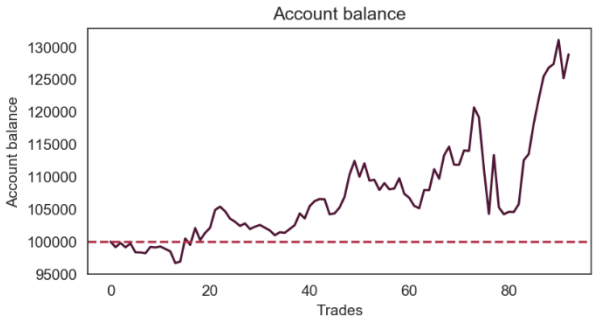

Figure 27. Account balance in 2021 May

2.4.3 Backtesting on Tesla stock price

As mentioned earlier, we also perform a testing of our strategy on some traditional stock market products. The model changes in that this time, for the sake of simplicity, we are working with a trading period of 1 hour instead of 6 hours. This change is simply explained by the need for the amount of data - in the traditional trading market, trading is not continuous, so by monitoring the data of market hours only in every 6 hours, there would be very few training and even less testing data in the given period.

Our first traditional market exchange product is the stock price of Tesla. The model will be taught from 1 January to 30 April 2021. The strategy will be tested from May 1 to 24. So, we take the data every trading hour.

Figure 28. Tesla stock prices for backtesting between 1 May and 24 May, trained from 1 January to 30 April 2021

The period contains 107 trading moves, of which the model predicts 52 times correctly, thus achieving only 48.6% accuracy. However, after a weak start, we can observe the relative stability of profitability. The exchange rate in the study period is slightly lower than the exchange rate in the teaching period, as shown in Figure 28. The model closes the test period with 3.9% of profit.

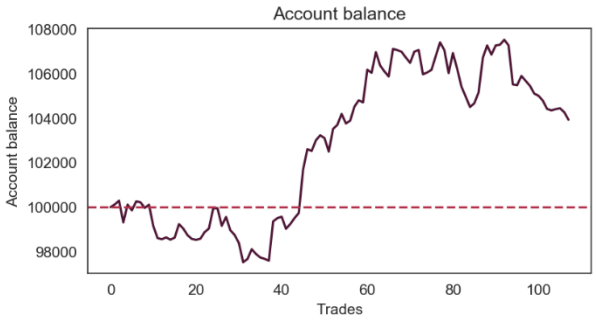

Figure 29. Account balance in May 2021.

2.4.5 Backtestion on Pelton Inc. stock price

Our next experiment in run on a randomly selected product, the stock price of the Peloton Incorporation. As it is shown in Figure 30, we experience a gradual exchange rate depreciation during the training period. This continues for a while during the test period, followed by an increase.

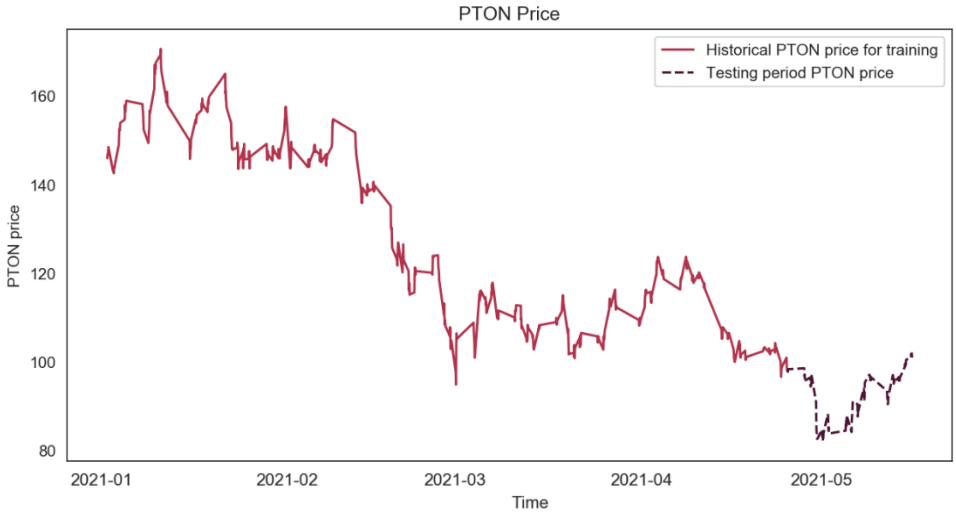

Figure 30. Pelton Inc. stock prices for backtesting between 1 May and 24 May, trained from 1 January to 30 April 2021

The prediction capability of the model achieves an accuracy of 60.2%, making 66 of 107 trading decisions correctly. The volatility of profit is observed in Figure 31. We close the period with a 7.4% profit.

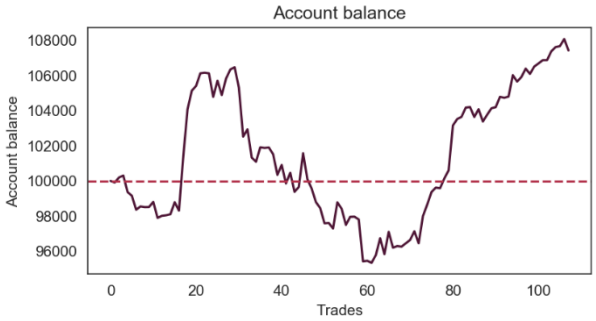

Figure 31. Account balance in May 2021.

2.4.6 Backtesting on Silvergate Capital Corporation stock price

Finally, we examined the operation of the strategy on the Silvergate Capital Corporation exchange rate. The behavior of the exchange rate in 2021 is shown in Figure 32.

Figure 32. Silvergate Captial Corporation stock prices for backtesting between 1 May and 24 May, trained from 1 January to 30 April 2021

The model is able to predict the change in the SI exchange rate with an accuracy of only 47.7%, so it makes the right decision in only 51 cases out of 107 trades. The strategy is seems to be profitable at the beginning of test, but the balance will soon turn negative. In Figure 33, we see that the model is trying to correct its errors, but the loss is significant for the time being. The model finishes the test period with a loss of 13%.

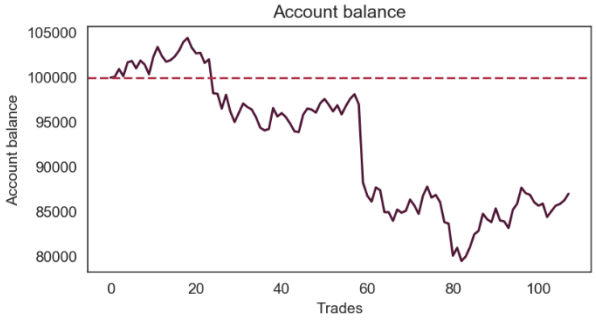

Figure 33. Account balance in May 2021

Figure 34. Account balance in May

This experience warns us against using the strategy without being careful. We find that with different exchange rate behaviors, the strategy can incur losses due to poor classification and incorrect trading decisions. Thus, further research and further development of the model and the inclusion of additional indicators in machine learning are necessary.

This experience also raised the question of long-term testing of the model. The last backtesting study of the dissertation will be conducted with a model trained between January 1, 2020 and April 30, 2020. The test period runs from May 1, 2020 to May 24, 2021.

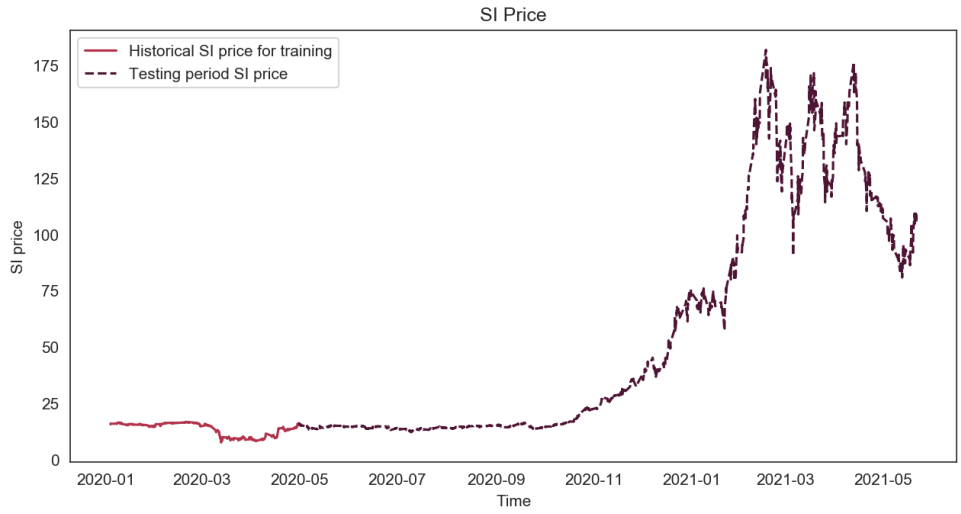

Figure 35 Silvergate Capital Corporation stock prices for backtesting between 1 May 2020 and 24 May 2021, trained from 1 January to 30 April 2020.

In the long run, we can see that the price of Silvergate Capital Corporation has been staying below $25 for some time, then reaching $175 with significant and sustained price increases after November 2020. This is followed by larger fluctuations. The test period thus begins with a relatively long stable exchange rate, followed by large price increases and larger price movements, as Figure 35 shows.

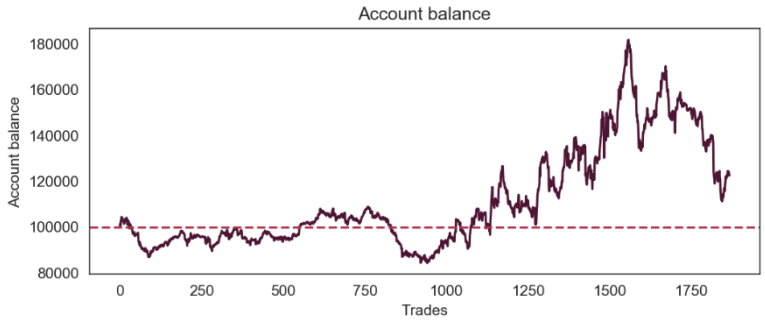

Figure 36 Account balance from 1 May 2020 to 24 May 2021.

Figure 36 shows that the strategy is profitable in the long run. First, we experience a weak loss followed by a weak rise. In our experience, the model responds well to large price increases with a slightly delay and makes large profits by taking advantage of it. However, it reacts to exchange rate falls with a loss of profit. The predictive power of the model is 52%, out of 1865 trades it decides correctly in 969 cases. The strategy closes the roughly 1-year test period with a 22.5% profit. The highest instantaneous profit experienced during the test period was close to 80%.

2.5 Trading bot

The trading strategy presented and tested above is also tested in a real-time environment. To do this, we use the free demo account provided by Binance. To implement automated trading, we use an application programming interface (API). The strategy will therefore be run

by an automated trading robot. This robot is written in Python and consists of the following elements.

The first step is to register a demo account on the Binance Testnet website. We can then generate the API key and secret key. To connect to the demo account, we need to specify these keys in the program code. Then we can establish the connection to the account by initializing the client.

```
client = Client(api_key, api_secret)
```

The classification model is trained with data from 1 January 2021 until the writing of this section of the dissertation, 28 May 2021. Data preparation and KNN model initialization are performed as previously described.

Building on the characteristics of the Binance test environment, the strategy implemented in the robot is modified as follows. Because the Binance API does not directly support negative balances that would result from short selling at the predicted declining exchange rate, we are adapting this branch of strategy to the capabilities of the API and demo environment.

At the start of the demo account, we will receive a certain amount of starting capital on both our USDT account and our BTC account. In order to roughly imitate our short selling strategy, we trade half the amount of bitcoin on our BTC account. So, if the model predicts a declining exchange rate change, we sell half of the bitcoin on our BTC balance. At the end of the period, we buy the same amount of bitcoin at the next exchange rate. In this way, the balance of our BTC account remains constant during the entire run of the robot after the completion of each trading step, and we realize the result of the step on the USDT account.

In the other branch of the strategy, i.e., if the strategy predicts an increasing price change, we buy bitcoin for half the amount on our USDT account as described earlier and then sell the purchased quantity at the end of the trading period. This step does not change the balance of the BTC account either, so it remains constant throughout the trading. The result of the operation is again realized on the balance of the USDT account.

Before starting the robot, we declare some variables to statistically examine the performance of the strategy. We record the starting USDT and BTC balances. In addition, we create a variable to record the number of successful trades and the total number of trades.

```
start_usdt=float(client.get_asset_balance(asset='USDT')['free'])
start_btc=float(client.get_asset_balance(asset='BTC')['free'])
successful_trades=0
total_trades=0
```

After initialization, we start the trading robot in a ‘while True’ loop, which will run until the kernel in interrupted. Each time the cycle is run, the robot takes the following steps to implement the strategy.

We record the USDT balance before trading. By calling the newdata () function, we update the dataset with the current open, close, high, low and volume. The function also calculates the missing financial indicators. The classification_input () function then prepares the current row of the dataset according to the input pattern of the classification model. Thus,

it scales the current row of the dataset and performs the feature selection. The preprocessed data is received by the classifier model as input. The classification method predicts the movement of the next exchange rate. If it predicts an increase in the price, it returns 1, otherwise it returns 0.

```
balance_beforetrading=float(client.get_asset_balance(asset='USDT')['free'])
data=newdata(data)
sample=classification_input(data)
if  model.predict(np.asarray(sample).reshape(1,-1))[0]== 1:
        quantity=round(((
                float(client.get_asset_balance(asset='USDT')['free'])) / 2 )
                /float(client.get_symbol_ticker(symbol=symbol)['price']), 5)
        order1 = client.create_order
                (symbol=symbol,side='BUY',type='MARKET',quantity=quantity)
        trade='buy'
    else:
        quantity=round(float(client.get_asset_balance(asset='BTC')['free'])
                / 2, 5)
        order1 = client.create_order
                (symbol=symbol,side='SELL',type='MARKET',quantity=quantity)
        trade='sell'
```

The prediction is followed by the execution of the trading step. First, we calculate the amount of bitcoin you want to buy or sell. This is done as follows for buying. We retrieve the balance of the current USDT account, half of which we will invest. We need to divide that amount by the current exchange rate of the bitcoin to get the number of pieces we want to sell. The quantity must be rounded to 5 decimal places for the API to handle the request. If we want to sell bitcoin, we retrieve the available BTC account balance according to the modified strategy and invest half of it. In this case, too, it is necessary to round the quantity to 5 decimal places.

We create the client order with the calculated quantity as follows. We specify the symbol to be traded, in our case it is BTCUSDT. We also specify the side we want to take in the trade, indicating selling or buying. Then, we specify the type of order, in our case it is a simple market transaction. Finally, we add the desired quantity. In the ‘trade’ variable we record the executed trading step.

After executing the trading, the robot is waiting until the next market moment. In our case, the time of the next trade is timed after 6 hours. It can be achieved with a sleep time of 21,600 seconds. However, it can be more professional if we set this period by using the server time. Furthermore, by calculating the runtime, we will continue running the code a few seconds earlier than 6 hours.

```
price_stop = float(client.get_symbol_ticker(symbol=symbol)['price'])
if order1['status'] == 'FILLED':
    if trade == 'buy':
        order2 = client.create_order(symbol=symbol, side='SELL',
                type='MARKET', quantity=quantity)
    else:
        order2 = client.create_order(symbol=symbol, side='BUY',
                type='MARKET', quantity=quantity)
else:
    print('Order was not completed')
```

Before rebalancing, we check if the order has run successfully. This is indicated by the status of the order. If the status is 'filled', the transaction has been executed. In this case, we

rebalance the rounding. If we have bought bitcoin, we will now sell the same quantity. If we have sold bitcoin, we will buy the same quantity. The result is realized on the USDT account. The BTC account balance is constant for the purposes of the model. We may redefine the quantity, if necessary, considering rounding for 5 decimals.

We also check that we have performed the correct trading step.

```
if (trade == 'buy' and (price_start < price_stop)) or (trade == 'sell' and
        (price_start > price_stop)):
    evaluation = 'correct'
    success += 1
else:
    evaluation = 'mistake'
```

As a last step, the automated trading robot develops the machine learning model with the experience gained. The model is then redefined by defining the new optimal parameter k and retraining the model on the extended data set.

```
X_train=np.append(X_train, np.asarray(sample).reshape(1, -1), axis=0)
y_train=np.append(y_train, data.iloc[len(data) - 1]['move'])

k=get_k(X_train, y_train)
model=KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
```

The robot prints some information on the console about the executed trades. It specifies what type of trade is it doing and how many bitcoin in bought or sold. Then it shows the result of the trade, realized on the USDT balance. The BTC balance is shown as well, just for control. Then, some statistics are presented about how the model works, such as the number of correct predictions and the percentage of accuracy. Finally, we check the change in the USDT account balance compared to the initial state, as well as the profit achieved so far.

Here is an example of how a trading robot works.

```
TRADE INFO 1
...buying 0.00567 ... to finish: ... sell now 0.00567
Account balance: 417.18930634 --> 417.39558094 correct
Account balance BTC: 1.03337671 --> 1.03337671
MODEL:  1 correct predictions out of 1
Correct prediction rate: 100.0 %
Account since the beginning: 417.18930   --->   417.39558
Profit:  0.20627  ....   0.04944%


TRADE INFO 2
... selling 0.51668 ... to finish: ...  buy now 0.51668
Account balance: 417.39558094 --> 417.40856524 mistake
Account balance BTC: 1.03337671 --> 1.03337671
MODEL:  1 correct predictions out of 2
Correct prediction rate: 50.0 %
Account since the beginning: 417.18930   --->   417.40856
Profit:  0.21925  ....   0.052556%
```

Figure 37. An example of how the trading robot works.

3. Summary

In the study, we developed and implemented a cryptocurrency trading strategy based on artificial intelligence. The basis of the strategy is a classification problem. In the first half of the dissertation, to implement this classification, we conducted a comparative study of different

classification methods. Examining several metrics, we found that the data set consisting of the historical price of bitcoin and some financial indicators is classified by the KNN classifier with the best performance. Based on this, we use the KNN classifier to implement the strategy.

The behavior of the strategy was examined in several backtesting processes. We have observed that the model adapts even in case of extreme price changes. However, for different currencies, the model had different behaviors. In some cases, we realized short-term losses. In the long run, the strategy seems profitable. Though, it requires more examination.

In the final phase of the project, we implemented a trading robot in Python operating in the test environment offered by Binance. The robot executes the trading strategy we have developed. We are planning to study and improve its operation in the long term.

In the future, it may be worthwhile to include additional indicators in the model. An idea that might come up is perhaps to monitor and involve social media in the classification in some form. It can be considered to use other types of transactions as well. We can also thing about to change the fixed trading period time according to some forecast. Finally, improving the robot requires more studies.

References

[1] Abouloula, K.; Krit, S-d. (2018) Using a Robot Trader for Automatic Trading. ICEMIS '18: Proceedings of the Fourth International Conference on Engineering & MIS. 3: 1-9.

[2] Hertig, A. (2017) What is Ethereum? https://www.coindesk.com/learn/ethereum-101/what-is-ethereum [downloaded: 23. 05. 2021.]

[3] Chen, R.C.; Dewi, C.; Huang, S.W.; Caraka, R.E. (2020) Selecting critical features for data classification based on machine learning methods. Jurnal of Big Data, 2020; 7: 52

[4] Dhar, V. (2013). Data Science and Prediction. Communications of the ACM. 56(12): 64-73.

[5] Duda, R.O.; Hart, P.E.; Stork, D.G. (2001). Unsupervised Learning and Clustering. Pattern classification, Wiley

[6] Farell, R. (2015) An Analysis of Cryptocurrency Industry. Wharton Research Scholars. 130.

[7] Hu, J.; Niu, H.; Carrasco, J.; Lennox, B.; Arvin, F. (2020). Voronoi-Vased Multi-Robot Autonomous Exploration in Unknown Environmens via Deep Reinforcement Learning. IEEE Transactions on Vehicular Technology 2020; 69(12): 14413-14423

[8] Kushwana, K.; Mishra, P. (2016) A Survey on Data Mining using Machine Learning Techniques. International Journal of Advanced Research in Computer and Communication Engineering 2016, 5(9): 177-180

[9] Rani, T.U.; Priyanka, C.H.S.; Monica, B.S.S. (2019) A dynamic data classification techniques and tools for big data. Journal of Physics: Conference Series, 2019, 1228: 1-12.

[10] Russel, S.J.; Norvig, P. (2010) Artificial Intelligence: A modern Approach. Third Edition, Prentice Hall

[11] Tan, P.N.; Steinbach, M.; Kumar, V. (2011) Introduction to Data Mining

[12] Wagner, A. (2020) What Is Decentralized Storage?: A Deep Dive by Filecoin https://coinmarketcap.com/alexandria/article/what-is-decentralized-storage-a-deep-dive-by-filecoin [downloaded: 23. 05. 2021.]

[13] Wang, L. (2005) Support Vector Machine: Theory & Applications, Springer, Berlin

[14] Zhang, X.D. (2020). Machine Learning. A Matrix Algebra Approach to Artificial Intelligence, Singapore, 223-440.

Acknowledgment

First of all, I would like to thank my supervisor, professor José Javier López Monfort, for recommending the topic. I am grateful that you introduced me to the world of cryptocurrency and helped my work with interesting materials. Thank you for whenever I had a problem, you were always interested and helpful. Implementing the project was a challenge for me, but I enjoyed every minute of it, and I am sure that I will continue to work on improving the strategy and the trading robot in the future.

Thank you for the support and understanding of my family. Thank you for accepting that I did not have time to have facetime with them, even though they were worried about me and we missed each other in the distance of 2,600km. I would like to say thank you for my partner, for supporting and reassuring me whenever my program stopped with an exception caused by another exception. Thank you for listening to me constantly philosophizing on the project.

At last, but not least, I would like to say thank you for the Erasmus program, which provided me the opportunity to study at the Universitat Politècnica de València. The love of work and helpfulness I experienced here is a huge motivation for me to start my career.