Stock Market Predictions with Artificial Intelligence

Jennifer Sissi Lange
Department of Information Systems
Hanyang University
Seoul, Republic of Korea
9207120238
Jenni-sissi.lange@debeef.de

Jade Lu
Department of Computer Science
Hanyang University
Seoul, Republic of Korea
9206220239
Jlu23@student.gsu.edu

Kerk Zong Zhe
Department of Electronic Engineering
Hanyang University
Seoul, Republic of Korea
9206820235
Kerk0013@e.ntu.edu.sg

Gin Lo Hoi Ching
Department of English Language and Literature
Hanyang University
Seoul, Republic of Korea
921082035
S216213@hsu.edu.hk

***Abstract*—This paper describes an approach on how to use artificial intelligence to predict a possible rise or fall of the stock market value in the near future.**

Keywords—Artificial Intelligence, Decision Tree, Random Forest, Receiver Operating Characteristic, Dataset Modification

I. Introduction

The stock market is a dynamic system that presents the economic well-being of companies. Investors and market participants tend to make decisions based on the traditional econometric models and technical analysis indicators. Due to the complexity of the past performance of the stock, the accuracy of the traditional method is not highly reliable. This uncertainty and the unpredictability of the stock market means investors might face serious financial risks in optimizing their investment strategy. 61% of Americans own stock because they are more open to buying stock (Caporal, 2023). There is a need for a sophisticated and data-driven solution that can provide more reliable and actionable predictions.

In this project, an AI model is proposed to predict the stock market trend and to capture the intricate patterns and subtle relationships within financial data. An ensemble learning model utilizing Random Forest and Decision Tree algorithms to analyze historical stock market data and price movements will be developed. The model will provide labels to indicate the trend of the stocks. The user could make better decisions based on our model.

By providing more accurate and reliable predictions, this model can help investors make informed investment decisions, optimize portfolio management strategies, and mitigate financial risks. Financial institutions can benefit from improved risk assessment and asset allocation, leading to enhanced profitability and stability. Moreover, a more precise understanding of market trends can contribute to overall market efficiency and investor confidence.

II. Dataset

This chapter describes the datasets that were used for the realization of this project.

A. TESLA Stock Dataset (TSLA)

The TESLA dataset includes approx. 10 years of the TSLA stock data from June 29, 2010 - February 3, 2020. The original dataset was slightly modified removing Adj Close for consistency reasons. Hence, the features that are included in the modified dataset are as follows: Date, Open, High, Low, Close, Volume, and Rate of Change. The prices are based on USD (kaggle, Tesla stock data from 2010 to 2020, 2020).

B. SAMSUNG Stock Dataset (SSNLF)

The SAMSUNG dataset includes decades of SSNLF stock data from January 4, 2000 - May 23, 2022. The prices are also based on USD. It's also modified as described in the TESLA Stock Dataset description (kaggle, Samsung Stocks, 2022).

C. TWITTER Stock Dataset (TWTR)

The TWITTER dataset was chosen to be used because of its interesting history. Due to Elon Musk's purchase of Twitter back in 2022, it became a private company, was delisted from the New York exchange, and is no longer in the stock market. This dataset contains Twitter's history on the stock market from November 7, 2013 - October 27, 2022. The prices are also based on USD. It's also modified as described in the TESLA Stock Dataset description (kaggle, Twitter Stock Market Dataset, 2022).

III. Methodology

The project focuses on the prediction of stock prices with the help of Decision Tree and Random Forest. The selection of these artificial intelligence methods is based on the analysis of the dataset and the conclusion which model is suitable in this case. This is discussed in chapter IV in more detail. This chapter refers to the general methodology of implementation.

A. Original Dataset

These are the features of the original dataset:

Date - Includes the year, month, and day in that order (YYYY-MM-DD)

Open - This is the opening price of the day.

High - The highest price of the day.

Low - The lowest price of the day.

Close - This is the closing price of the day.

Volume – The total number of shares traded in the day.

Adj Close - This means the adjusted closing price of the day. It's adjusted to better reflect the stock's value after anything such as corporate actions would affect the stock price after the market closes. However, this was taken out for consistency reasons.

B. Modified Dataset and Features

The dataset is extended with modified data to make the dataset bigger and create more data for the AI to train on. With the additional created data, possible patterns and relationship between different information will be recognized by the AI. In the following it is described how the data is going to be modified and extended.

Slope/rate of change - It is calculated using the formula below. The purpose is to train the AI to consider the influence that stock prices on the days prior have on that of the present day. This helps the AI recognize certain patterns happening in the given datasets. For instance, when calculating the rate of change for the opening price, y_2 would be the opening price of the current day, y_1 would be the previous day's opening price, and x_2 - x_1 would be the number of days. The slope is calculated as follows:

$$slope = \frac{y_2 - y_1}{x_2 - x_1}$$

This calculation would be repeated based on the fixed number of days. The following figure illustrates the calculation of the slope.

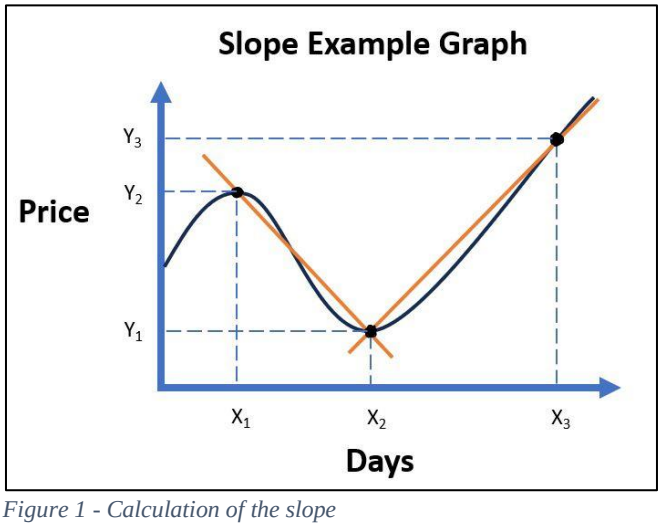

Figure 1 - Calculation of the slope

Average slope - Using the outcome of the calculation of the slopes, the average of those single slopes can be calculated, allowing one to determine whether the price will rise or fall. If the number is positive, the value increases. If the number is negative, the value falls.

Number of rises and falls - In this new feature, the individual calculated slopes are taken into account again. The number of slopes with a positive or negative sign are counted. This means that the number of times the share price rises or falls in a certain time frame is counted. In this way, it is intended that the artificial intelligence in this work may be able to recognize patterns of behavior on the stock market in order to predict the further course. In the following illustration, the rising and falling of the price is shown with arrows.

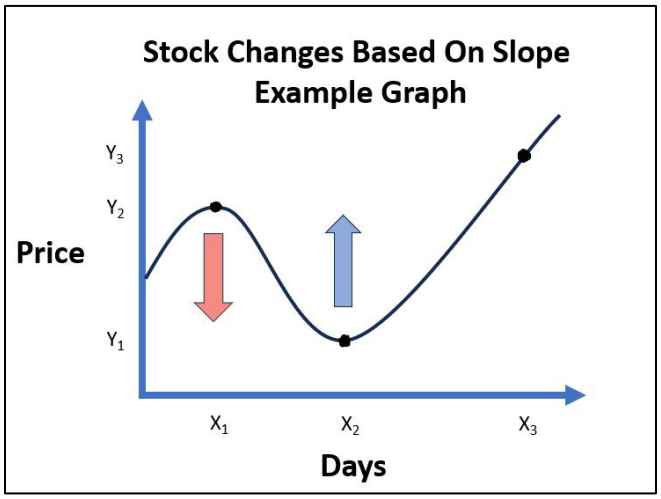

Figure 2 - Stock price changes based on slope

Difference between High and Low - Here the difference between the High and Low prices are calculated. This is done for the current day and all past days within a certain time frame. The difference shows how much the price fluctuates within a day. This is intended to illustrate further behavior in the stock market in order to identify certain patterns.

Average difference between High and Low - The average difference between High and Low is calculated. This summarizes whether the difference in the days is generally high.

Difference between Open and Close – Here the difference between the Open and Close prices are calculated. This gives us more information on the stability of the price. This is intended to provide further insight that may be useful in exploring patterns for artificial intelligence.

Average difference between Open and Close - Similar to the High and Low values, the average difference between Open and Close is also calculated here over a specific time window.

C. *Creating a label*

The labels 1 and 0 are used to predict a rising or falling price, respectively. The labels are determined on the basis of the average rate of the low value. The low value, which represents the lowest price on a day, is used to forecast based on a worst-case scenario. To check the quality of the labels, the stock price performance is visualized with 3 data points each for a fall and rise in the stock price. The visualization is shown in the figure below. The green dots represent the rise and the red dots the fall of the share price.

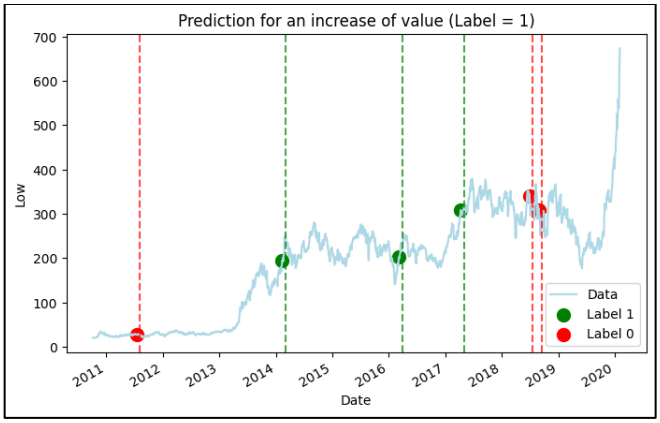

Figure 3 – Visualization of the labels

D. *Distribution of the dataset*

In terms of training and testing, the modified dataset will be split up 70% of the data will be used for training and 30% used for testing. The amount of training data is distributed as such because there needs to be as much data as possible to train the AI. On the other hand, there must be enough testing data available to ensure an accurate testing result.

E. *Implementation of the code for modifying and labelling the dataset*

Several functions were declared for the realization of the project and subsequently applied. Their use is described below. The exact code can be found in the Jupyter notebook file.

def combine_data(trade_datas):

This function is used to merge different datasets. A list of DataFrames is passed, which are combined into one at the end. Since datasets from different companies are used in this project, it is necessary to merge them in a standardized way.

def modify_data(trade_data,t_previous_days, t_label_days, comprimize_data)

This function is used to modify a dataset. The purpose of modifying the dataset is to extract more information from the given dataset and use it as an additional feature. The new features are used, for example, to recognize temporal patterns. The label is also created in this function. To execute the functions, the dataset is required as well as the number of days immediately before and after the respective data points that are to be analyzed. comprimize_data is a boolean which is used to determine whether all modified features should be used or a compromised version.

def plot_label_visualization(data):

This function displays the result of the automatically created labels. The progression of a price is displayed graphically. For each label, 3 data points are selected at random and displayed in the graph in the form of dots in different colors. A vertical line with the respective color is then drawn for each data point, making it clear which point in time is being considered for a forecast.

F. *Function of a Decision Tree*

A Decision Tree classifier is a machine learning algorithm that can be used for classification purposes. The model resembles a tree that recursively splits the dataset into subsets decided by the values of parameters, that are regularly adjusted when training the model to achieve a higher level of accuracy. The following image illustrates the function of a Decision Tree:

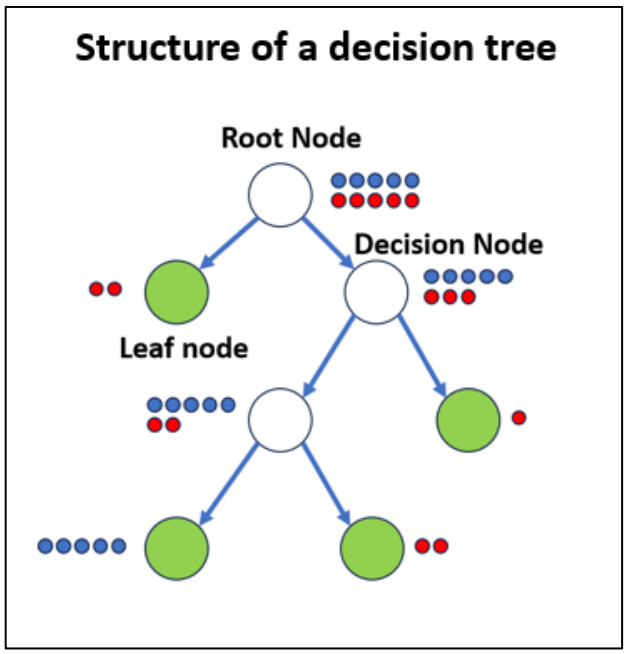

Figure 4 - The structure of a Decision Tree referring to (Dash, 2022)

A Decision Tree consists of the following basic components:

Root node – starting (top) node with all data points

Decision node(s) – nodes with condition(s) to split data

Leaf nodes – nodes with ideally only data points of a single class

Starting from the root node, the Decision Tree classifier decides on the most optimal way to split the dataset into subsets. There are conditions in the decision nodes for splitting the data. Based on the conditions, the data is

distributed to two or more branches. For example, a threshold can be used as a possible condition with which the features in the dataset are compared. The value of the threshold is random at the beginning and is adjusted during training. Optimal conditions are achieved during training with the highest possible information gain.

Information gain is defined as the effectiveness of a splitting condition in reducing entropy or Gini impurity within the data subset. The information gain can be calculated by the changes in entropy or the Gini index when training the model.

Entropy is the measure of impurity of data points. The aim here is to achieve a low value. The lower the entropy value, the more homogeneous the data is after splitting. The entropy value is calculated as follows:

$$Entropy = -\sum_{i=1}^{n} p_i * log\ (p_i)$$

p_i refers to the percentage of class i in the data subset at the node. Using the entropy determined, the information gain can be calculated as follows:

$$Information\ Gain = Entropy(parent) - \sum w_i * Entropy(child_i)$$

w_i refers to the weight of class i expressed as a fraction of the data subset at the node.

Compared to entropy, the Gini index concentrates on the probability that a randomly selected instance will be misclassified. The aim here is to achieve the lowest possible value. The lower the Gini index, the lower the probability that something will be classified incorrectly. The Gini index is calculated as follows:

$$Gini = 1 - \sum_{i=1}^{j} {p_i}^2$$

The variable j is the total number of classes in the target variable and p_i refers to the percentage of class i in the data subset at the node. Similar to the use of entropy, the information gain can be calculated using the Gini index and the following formula:

$$Information\ Gain = Gini(parent) - \sum w_i * Gini(child_i)$$

Once the best possible condition based on the highest information gain has been determined, the data is split based on this and the process is repeated for each new node. The process continues until the stop conditions are met. The stop conditions are described in the following:

- All data points in a node belong to the same class
- The maximum depth of the tree has been reached
- The minimum number of data points in a node has been reached.

As soon as the stop conditions are met, a leaf node is created instead of a decision node. The leaf node determines the class affiliation of the respective root in the Decision Tree.

After the Decision Tree has been created, a data point is classified based on the conditions of the decision nodes. Depending on the condition, a specific branch in the Decision Tree is followed. The leaf node that is reached at the end determines the predicted class. (Dash, 2022) The following image shows how a class of a data point is determined.

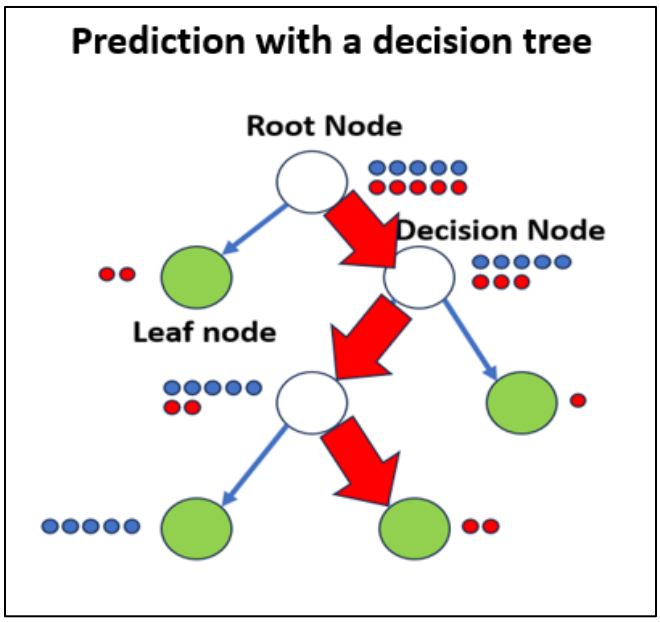

Figure 5 - The process of prediction with a Decision Tree referring to (Dash, 2022)

G. Function of a Random Forest

The Random Forest algorithm is an enhanced version of the Decision Tree classification model, making use of a collection of multiple Decision Trees. Decision Trees are highly sensitive to the training dataset and any modification to the training data can result in an entirely different Decision Tree. Random Forest can reduce the sensitivity towards the training data through generalization.

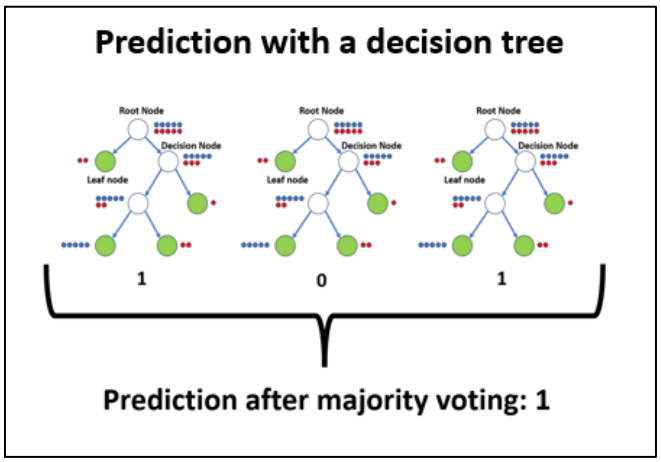

Figure 6 - The function of a Random Forest referred to (Donges, 2023)

A Random Forest consists of several Decision Trees that are created independently of each other. Bootstrapping is used to train the trees. Bootstrapping is a process in which random data points are taken from the dataset for training for each of the trees, so that each of the trees is trained with different datasets, increasing the versatility of the forest. It can happen that the same data point occurs in several training datasets. It is also possible that the random selection means that certain data points do not appear in any tree in the training data. Bootstrapping prevents the use of the same dataset in the training process, reducing the sensitivities of trees towards the original dataset. Limiting the random feature selection reduces the correlation between trees, as trees are trained with a variety of decision nodes, hence increasing the variance in prediction outcomes.

To merge the predictions of all trees and create a single prediction of the Random Forest, an aggregation is performed. To make a prediction using Random Forest, the data point is passed through each tree to get prediction outputs from every tree. The final decision is then made through majority voting of all predicted outcomes. The structure of a Random Forest is shown in figure 6 (Donges, 2023).

H. Implementation of the code to create the model

In this project, the Decision Tree and Random Forest are both created with the library sklearn and programmed from scratch. As the models are each trained and compared with different parameters, functions are defined for the executions. The following functions have been created to summarize all the individual steps involved in creating the respective models. The general procedure of the functions is very similar. To execute the function, a previously modified dataset is provided. Firstly, the dataset is split into test and training data. Then the model is created and trained with the training data. At the end, the trained model is tested with the test data and a value for accuracy is created. At the end, the model, the predictions made during testing, the labels and the accuracy are returned. The declared functions of the respective models are presented in the following.

def run_decision_tree_classifier_by_sklearn(modified_trade_data)

This function is about the creation of a Decision Tree classifier with the help of the module of the sklearn library. The following function by sklearn is used: *DecisionTreeClassifier(max_depth=10,min_samples_split=5)*

def run_random_forest_classifier_by_sklearn(modified_trade_data):

This function is about the creation of a Random Forest classifier with the help of the module of the sklearn library. The following function by sklearn is used: *RandomForestClassifier(n_estimators=10, random_state=42,max_depth=10, min_samples_split=5)*

def run_decision_tree_from_scratch(modified_trade_data)

This function involves the creation of a Decision Tree classifier whose algorithm was implemented within the scope of this project. The exact algorithm will be explained in more detail later.

def run_random_forest_from_scratch(modified_trade_data)

This function involves the creation of a Random Forest classifier whose algorithm was implemented within the scope of this project. The exact algorithm will be explained in more detail later.

The implementation from scratch is described below, starting with the Decision Tree.

class Tree_Node():

The class Tree_Node represents a Node in a Decision Tree. It contains information about the splitting of the tree including the feature index, a threshold, and a subtree. Additionally, it contains the information gain resulting from the splitting of a tree. Furthermore, it contains the label if it is a leaf node.

class DecisionTree():

The DecisionTree class represents a Decision Tree classifier which is implemented from scratch in the purpose of this project. It contains the parameters like min_datas_branching and max_depth which describe how often a tree will be split and the maximum depth of the split. Furthermore, it contains the roots of the previous branches.

def create_tree(self, dataset, curr_depth=0)

The function create_tree builds a Decision Tree recursively based on the given dataset. It uses values like min_datas_branching and max_depth as conditional values to decide whether conditions are met to calculate the best split and build another subtree recursively. If the conditions aren't met, a leaf node will be created.

def get_best_branching(self, dataset, number_of_datas, number_of_features)

This method searches for the best split in a Decision Tree based on the dataset. It iterates through the features and their possible thresholds to calculate the best branching. It calculates the potential threshold for every feature, divides the dataset and calculates the information gain using whether the Gini index or the entropy. It updates the dictionary of the best branching based on the highest possible improvement in the information gain value.

def branch_tree(self, dataset, feature_index, threshold)

The function branch_tree divides a dataset based on a feature and a threshold. It creates two arrays in which the data smaller or bigger than the thresholds are separated.

def information_gain(self, parent, l_child, r_child, mode="entropy")

The information gain function is used to calculate the information gain after a splitting of a branch. For the calculation it uses either the Gini index or the entropy, determined by the 'mode' parameter.

def entropy(self, y):

With the entropy function, disorders and randomness within a set of labels can be calculated. By examining the different types of labels and quantifying how unpredictable they are, it calculates the probability.

def gini_index(self, y):

With the function Gini index, similar to the entropy function, the impurity or disorder is going to be measured. By iterating through the labels, it calculates the Gini index based on the probability of the occurrence of each of the labels. After applying a specific mathematical formula, it returns the Gini index.

def calculate_leaf_value(self, label):

The function calculate_leaf_value detects the label of a leaf note. It predicts the value by doing a majority voting of the labels that the leaf contains. The most frequent label will be returned as its label.

def fit(self, data, label):

With the fit function, the Decision Tree is going to be trained. For this, the dataset and the labels are merged before using the function create_tree to establish the nodes.

def predict(self, data):

The predict function is used to create predictions based on the given dataset. It iterates through each data point and creates a prediction with the function make_prediction function.

def make_prediction(self, x, tree):

The make_prediction function is used within the predict function. It uses the Decision Tree structure and evaluates a single data point against the nodes of the tree. It compares the feature values with the nodes and finds the correct root through the tree. Finally, it finds the predicted label when reaching the leaf note and returns it.

The implementation of the Random Forest in scratch is now presented below. It should be mentioned that the Decision Tree programmed in scratch presented above is used to create the forest.

class RandomForest():

The class RandomForest is used to build a Random Forest model. It is implemented from scratch and creates Decision Trees that are, as described above, implemented inin scratch as well. The RandomForest class uses parameters like n_trees to describe the number of trees within the decision forest. Furthermore, it uses the parameters like max_depth and min_datas_branching which are necessary to create a Decision Tree. Furthermore, it contains a parameter which describes the number of features in the dataset and another which contains all trees in an array.

def fit(self, X, y):

The Random Forest gets trained by the function fit. It creates multiple Decision Trees for the forest. When creating the Decision Tree, it uses the functions that are explained in the Decision Tree class section. After the creation of the individual Decision Trees the function samples are going to be used to create a diverse subset for the input data of each tree. This way, the Random Forest model has a higher variability among a tree which leads to more robustness and a higher predictive performance.

def samples(self, X, y):

After each Decision Tree is created, the function samples create different training datasets for each tree. These datasets are created randomly out of the given dataset. This way, unique training datasets are created which leads to individual training of the trees.

def identify_most_common(self, y):

This function detects the most frequently occurring label within a set of labels. These labels are created by all of the Decision Trees in the Random Forest. The most common label will be returned.

def predict(self, X):

With the predict function, the created Random Forest model will be used to predict the label of a given test dataset. The data will run through all created Decision Trees in the forest. Each Decision Tree will predict the label of the data. Using the predictions of all trees, the most common predicted label is going to be taken as the final predicted label.

IV. Evaluation and analysis

A. Concept for analysing the model

Various tests are carried out to analyze the quality of the predictions and the behavior of the Decision Tree and the Random Forest for stock performance and to determine whether this method is suitable. The tests are carried out in different variants based on the following aspects.

a) Decision Tree vs. Random Forest

On the one hand, the behavior of the predictions is compared based on the use of a Decision Tree and Random Forest.

b) Forecast for a specific time in the future

The model is tested for a prediction of the stock price performance in 10 and 20 days. The past 10 days are taken into account for the creation of the modified features.

c) Different versions of the dataset

Each model is tested on different versions of the dataset. Three different types of datasets are used in this project. Firstly, training takes place with the dataset that only contains the original features. Then the dataset is used with all the modified features created. The number of features can be large depending on the parameters from b). For this reason, a compressed version of the dataset is then used for training, which, regarding the created slopes, only considers the average value. This compromises the number of features from 221 to 21.

d) Demonstration of the models created by scratch

The Decision Tree and Random Forest created in scratch in this project is run for demonstration with the compromised dataset for the prediction of the stock performance in 10 days.

Based on the different variations, a total of 16 different models are created and then compared. However, these parameters are fixed in every model:

- Min samples split =5
- Max depth of a tree =10
- Number of trees in the forest = 10

B. *Analyzis of the dataset*

Before the dataset is used to train an artificial intelligence, it is first analyzed. The analysis is used to determine whether the dataset is suitable for classification and whether certain features have dependencies with regard to classification. A table with a large number of two-dimensional coordinate systems is created to analyze the dataset. In each coordinate system, two features are compared with each other by assigning them an axis and drawing data points based on them, which are colored differently according to their labels. If the differently colored data points are visibly separated in the coordinate system, this indicates that the respective features have a high significance in terms of classification. In addition, a graph is created for each feature, which shows the distribution of the data points according to their labels. A graph in the corresponding color is generated for each label. The more the distributions differ, the more meaningful the features are for classification.

Figure 7 shows the analysis of the dataset without modified features. A larger figure can be found in the appendix.

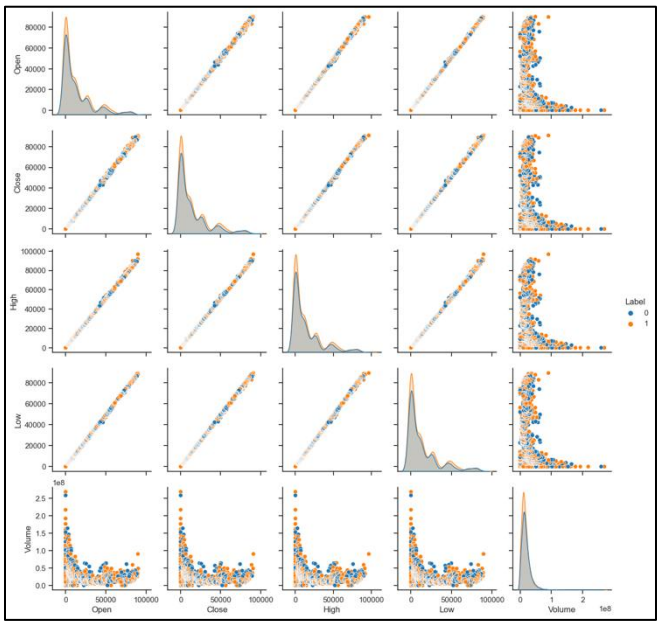

Figure 7 - The analysis of the dataset without modified features

Figure 7 shows that none of the features is highly significant for a clear classification. On the one hand, the data points cannot be grouped by their labels. On the other hand, the created graphs overlap and do not show any differentiation.

Figure 8 shows the analysis of the dataset that was expanded with the modified features. This is an excerpt. The entire figure can be found in the appendix. In this case, it is noticeable that some of the data points are better grouped in the coordinate systems. This indicates a higher significance of the features in the classification of data points. In addition, a difference can be seen between some of the graphs created.

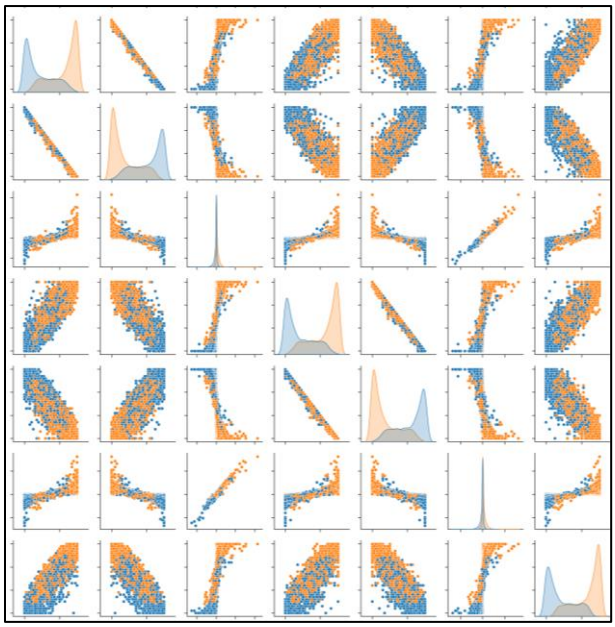

Figure 8 - The analysis of the dataset that was expanded with the modified features

Based on the analyzed datasets, it is now possible to forecast the behavior of the model's prediction quality based on the dataset used in the training.

C. AI Methodology Selection

Based on the analysis of data, the selection of Decision Tree and Random Forest is justified in this chapter. Due to the grouped distribution of data points in their labels, most data points are spatially separable. Under these circumstances, Random Forests are well suited. As this model is based on the Decision Tree concept, this method is also considered in this thesis.

D. Receiver operating characteristic

The measure of a classification model's performance at all classification thresholds can be visualized on a graph called the ROC Curve (receiver operating characteristic curve).

The ROC Curve plots two parameters, True Positive rate (TPR) against False Positive rate (FPR), for all classification threshold values between 0 and 1. To understand TPR and FPR, the following key definitions from the Confusion Matrix are used:

True Positive (TP): positive cases successfully classified into the positive group.

False Positive (FP): negative cases incorrectly classified into the positive group.

True Negative (TN): negative cases successfully classified into the negative group.

False Negative (FN): positive cases incorrectly classified into the negative group.

The TPR is the measure of percentage of positive cases correctly distinguished out of all positive cases, given by the following formula:

$$TPR = \frac{TP}{TP + FN}$$

FPR, on the other hand, is the measure of percentage of negative cases incorrectly distinguished out of all negative cases, given by the following formula:

$$FPR = \frac{FP}{FP + TN}$$

With the ROC Curve, reading the AUC (2D area under the ROC Curve) provides us insight on the ability of the binary classifier to separate between classes.

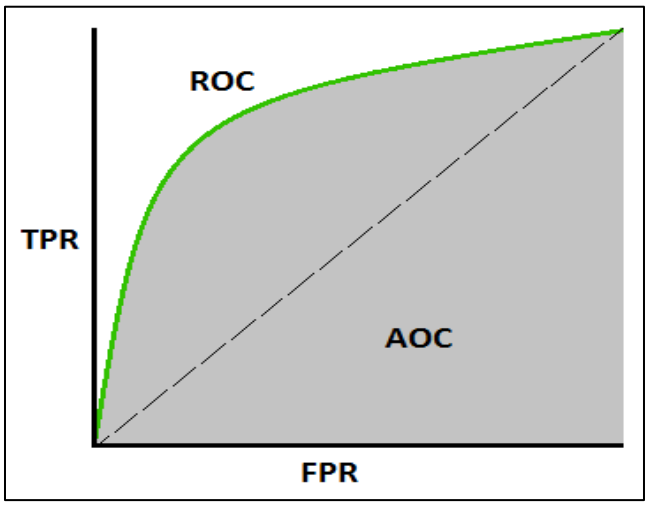

Figure 9 - The ROC Curve (Narkhede, 2018)

The AUC is a metric ranging from 0 to 1. A value of 0 suggests that all positive and negative instances are completely misclassified, while a value of 1 signifies perfect classification, where all instances are correctly placed in their respective classes. In the diagram above, the AUC is exactly 0.5, implying that the chance of a correct classification is only 50%, equivalent to that of a random coin flip. The ROC trending upwards and covering a larger AUC indicates better model performance, demonstrating an enhanced ability to accurately distinguish between positive and negative classes. On the other hand, a downward trend in the curve represents a worse performance of a model comparatively (Bhandari, 2023) (developers.google.com, kein Datum) (Narkhede, 2018).

V. Results

Type of dataset	Type of model	Prediction in t days	
		t = 10	t = 20
only with orginal features	*Decsion Tree*	0.5569	0.5891
	Random Forest	0.5870	0.6073
with all 221 modified features	*Decsion Tree*	0.9349	0.9082
	Random Forest	0.9554	0.9194
compramized with 21	*Decsion Tree*	0.8315	0.9088
	Random Forest	0.8602	0.9194
	Decsion Tree (scratch)	0.8397	-
	Random Forest (srcatch)	0.8589	-

Table 1 - The accuracies of the models

In Table 1, the accuracies of the various models are recorded. From this, different behaviors of the model can be observed. In general, it should be noted that the Random Forest shows a slightly better result in each of the cases. This confirms the theory that a Random Forest has improved robustness, accuracy and resistance to overfitting compared to a Decision Tree.

Based on the analysis of the dataset in chapter IV, it has been confirmed that training an artificial intelligence with only the original features leads to a poor result. Among all

models, both the Decision Tree and the Random Forest show by far the worst results. Figure 10 shows the ROC curve of the Decision Tree and Random Forest that were trained with the original features. The curves are very close to the 45-degree line, which is similar to the probability of a coin flip.

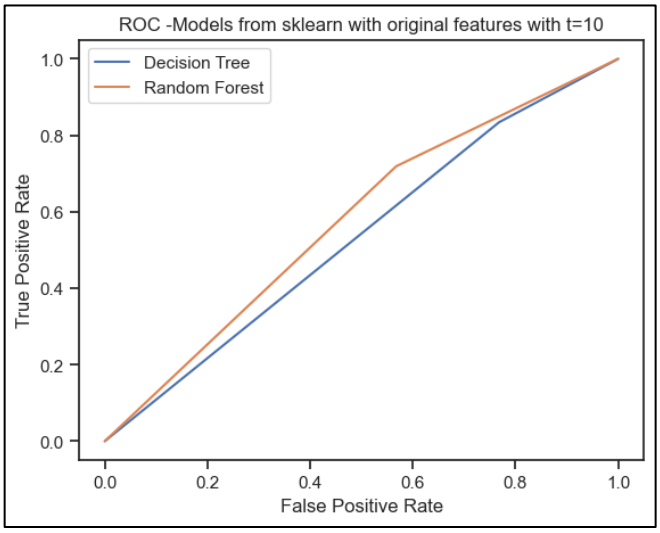

Figure 10 - ROC curve: trained with the original features

Comparing the results of the models with t = 10, which were trained with all or compromised features, it can be seen that it can be seen that the accuracy decreased with the reduction in the number of features. This can be explained by the reduction in available information, which is accepted in return for the increase in speed of the model. The following figures show the difference between the ROC curves in this regard.

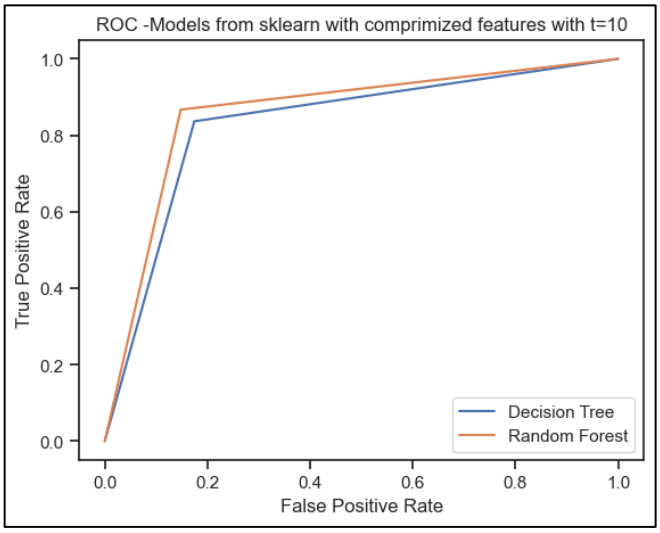

Figure 11 - ROC curve: trained with compromised features

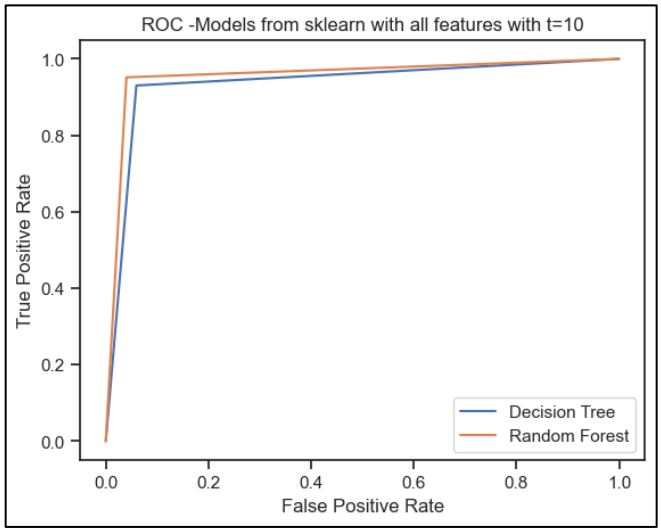

Figure 12 - ROC curve: trained with all features

For most models, a prediction of the stock price at a later date is of higher quality. However, it is noticeable that the predictions lose quality when training with all features and t = 20. In comparison, the quality of the predictions increases significantly when training with compromised features and t = 20.

VI. Related Works

In a recent study, employed the Random Forests method to forecast cleaning energy stock prices. The study's results demonstrated a remarkable level of accuracy, with a prediction rate of over 80% for a 10-day forecast and an even more impressive 90% for a 20-day forecast. Modifying the dataset can have an impact on the accuracy of the model. Multiple widely recognized technical indicators are used as features, such as moving average cross-over divergence (MACD), relative strength indicator (RSI), stochastic oscillator (slow, fast), price rate of change (ROC), and advance-decline line (ADX) (Sadorsky, 2021). These features are similar to those created as part of this project work. However, they differ in terms of the calculation and the type of display. For example, the change in the file price is shown in both datasets. However, in this project this is done as a gradient, whereas in the study the percentage change is taken into account.

In another project, the future performance of the S&P 500 stock was predicted. Here, the prediction focused only on the values that were already available in the dataset and used these as the only features. The features are similar features that reflect, for example, open, close, high, low, etc. Here, an accuracy of 57% was achieved. The large difference in the accuracy values is possibly due to the feature extraction, which was not carried out in this case (VikParuchuri, 2023).

VII. Conclusion

In this project, artificial intelligence is used to predict whether the price of a share will rise or fall in the near future. Based on the analyses and tests carried out in the previous chapters, it becomes clear that the realization and quality of the models subsequently created depend on various points.

On the one hand, the quality of the model's prediction depends on the quality of the dataset. The analyses have made it clear that training with less meaningful features leads directly to a poorer quality of the prediction.

On the other hand, the tests in this thesis made it clear that the creation and training of the Decision Tree and Random Forest depend on the prediction target. In the test, predictions of the stock price for the next 10 or 20 days were considered. The significant increase in accuracy was noticeable for a model that was trained with compromised features as soon as the stock price is to be predicted for a longer period in the future. In comparison, in a training with all extracted features, the accuracy for the prediction of the price in the near future has decreased.

This behavior leads to the conclusion that there is no perfect concept for creating and training the model that produces the same result regardless of the circumstances. In relation to this project, the concept depends on what exactly is to be predicted. It can be noted that some features that are missing in the compromised version of the dataset contain more meaningful information for a shorter-term prediction, which might be confusing for a longer-term prediction. It can thus be concluded that a higher number of features does not directly lead to better accuracy.

VIII. References

Bhandari, A. (2023, August 31). *Analytics Vidhya*. Retrieved November 14, 2023, from https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/

Caporal, J. (2023, November 01). *The Motley Fool*. Retrieved October 30, 2023, from https://www.fool.com/research/how-many-americans-own-stock/

Dash, S. (2022, November 02). *Medium*. (T. D. Science, Editor) Retrieved November 14, 2023, from https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c

developers.google.com. (n.d.). *developers.google.com*. Retrieved November 2023, 14, from https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=de

Donges, N. (2023, September 15). *built in*. Retrieved December 01, 2023, from https://builtin.com/data-science/random-forest-algorithm

kaggle. (2020). *kaggle*. Retrieved October 15, 2023, from https://www.kaggle.com/datasets/timoboz/tesla-stock-data-from-2010-to-2020

kaggle. (2022). *kaggle*. Retrieved November 14, 2023, from https://www.kaggle.com/datasets/ranugadisansagamage/samsung-stocks

kaggle. (2022). *kaggle*. Retrieved October 25, 2023, from https://www.kaggle.com/datasets/amirmotefaker/twitter-stock-market-dataset

Narkhede, S. (2018, June 17). *Medium*. (T. D. Science, Editor) Retrieved November 14, 2023, from https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Sadorsky, P. (2021). *MDPI*. Retrieved November 14, 2023, from https://www.mdpi.com/1911-8074/14/2/48

VikParuchuri. (2023, December 01). *GitHub*. Retrieved from https://github.com/dataquestio/project-walkthroughs/tree/master/sp_500