Master of Science in Artificial Intelligence and Data Engineering
Data Mining and Machine Learning

Crypto Predictor

Project Documentation

Team Members:
Nello Enrico
Pierucci Matteo

Academic Year 2021-2022

Sommario

1 Abstract

Artificial Intelligence is an active research area, it provides solutions to many real-life problems and is an integral part of our daily lives.

Machine learning is a subset of Artificial Intelligence in which we implement human thinking using mathematical models. It has many applications in the areas of customer analysis, cost reduction, risk reduction, among others. The aim of machine learning is to provide some conclusion or prediction from the available dataset using a model.

Since the last decades, particular attention has been given to the topic of cryptocurrency.
Cryptocurrencies are exploited as a currency of exchange and nowadays, cryptocurrency trading is becoming more and more popular.
There are an increasing number of open-source trading strategies available and recently, the idea of using machine learning to develop a cryptocurrency trading strategy has also emerged. In addition, crypto trading is mostly unregulated and there are many opportunities to enter the market easily.

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.
Several studies have already been built to regulate the cryptocurrency exchange rate, visualize graphs, and analyze the trends with the aim of creating more and more profitable trading strategies.

We propose a classification model to predict whether the price is going to increase, decrease or remain stable in the next market moment. So, the main idea is to collect historical data about the price changes of cryptocurrencies for the current day and make a prediction on the trend for the day after.

2 Introduction

In finance, the price changes can be approached from several perspectives. One of them is represented by the 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 from them.

In this study, we propose a cryptocurrency trading strategy in Python based on some machine learning tools. We build a dataset from historical data of some cryptocurrencies, evaluate some financial indicators, and finally we define a classification problem to predict the change of a coin value 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, which is used as the core of the trading strategy.

Crypto Predictor is a web application which purpose is to help users in crypto currencies trading, providing a trend prediction for specified coins.
More specifically, it provides a prediction based on a Classifier trained upon financial Statistical indicator that are evaluated on a Time Series.

Users can select the preferred cryptocurrency from the index page, then the application will show a graph of the historical values of the coin together with the trend prediction for the following day.

2.1 Functional requirements and use cases

- A user can:
    - Select one cryptocurrency
    - Browse historical data
    - Change graphical parameters
    - Ask for trend prediction

2.2 Non-functional requirements

The following list outlines the non-functional requirements of the application.

- The application must embed one or more machine learning algorithms.
- The application must be user friendly, easy to use and provide a simple and intuitive user interface.

2.3 UML use case diagram

In our application there are two main actors:

- User
- Updater

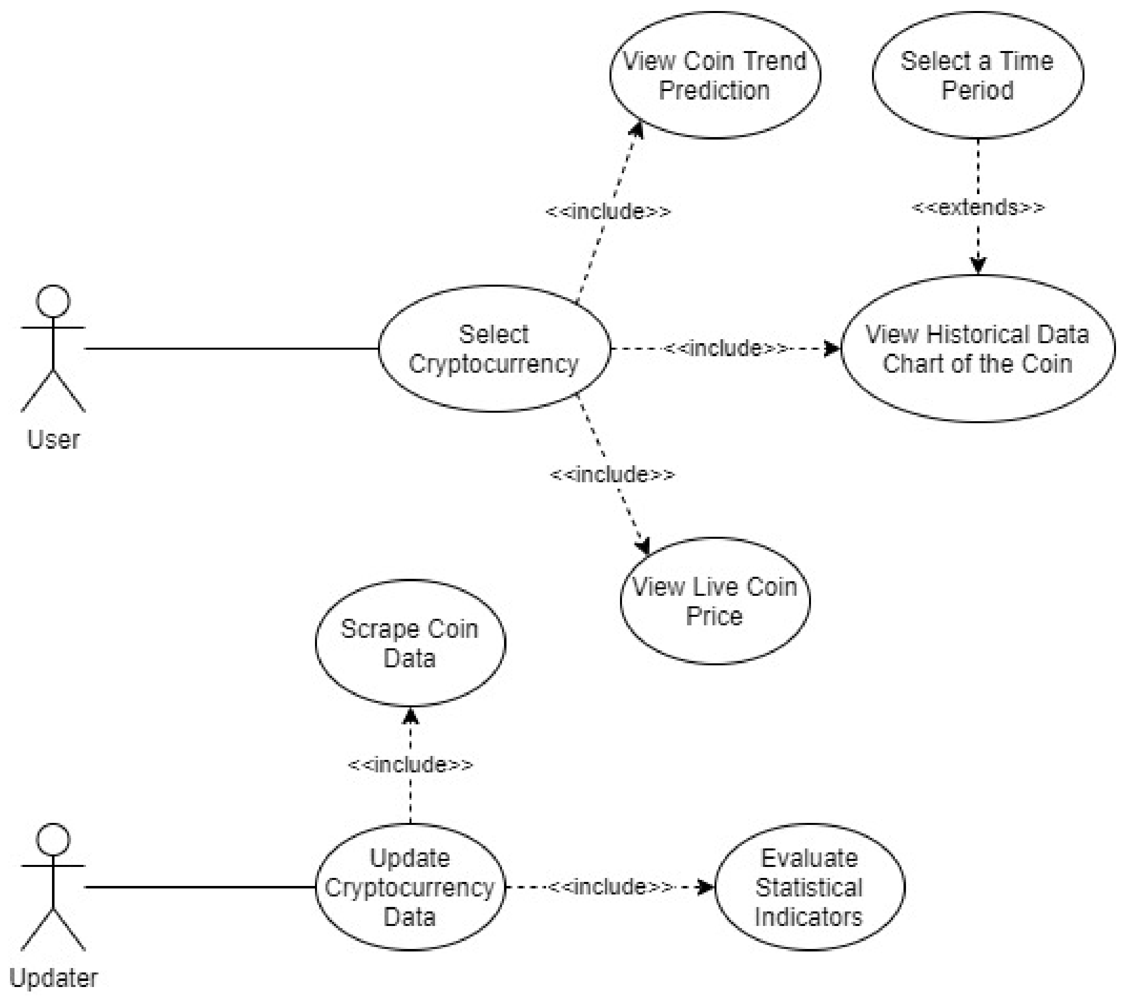


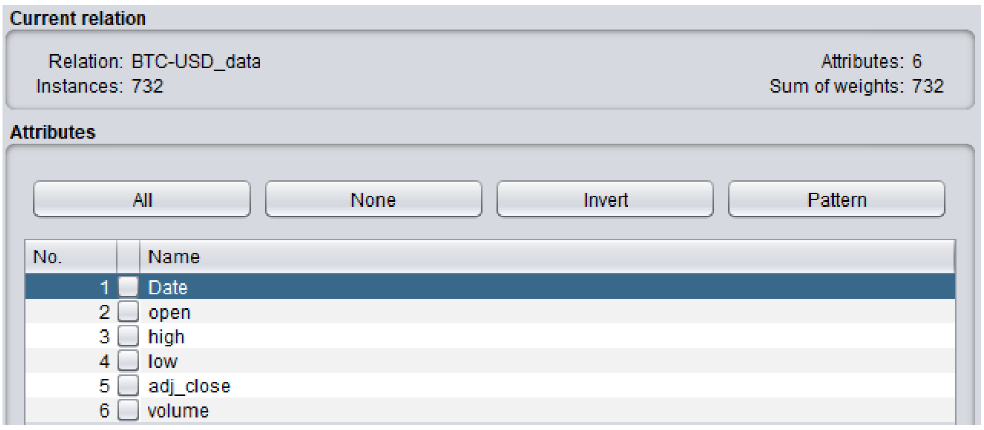
Figure 1 UML Use Case Diagram

3 Dataset Creation

Source: we scraped from Yahoo Finance API the crypto historical values of the last two years (between 2020 and 2022) for each of the three most popular currencies:

- Bitcoin
- Ethereum
- Binance

Description: This dataset contains information on the past financial values of the currencies, in particular data from the past two years from three different currencies for a total of 2190 (365 * 2 years * 3 currencies) different tuples.



Current relation

Relation: BTC-USD_data
Instances: 732
Attributes: 6
Sum of weights: 732

Attributes

No.	Name
1	Date
2	open
3	high
4	low
5	adj_close
6	volume

Fig. 1 This image refers only to a single crypto-currency dataset

3.1 Intuition

The main intuition behind our work is based on the *Moving Average Cross Strategy*, a common technical approach in finance, that exploits statistical indicators to predict the future value of an asset. Our goal is to train a model capable of predicting the future value of an asset, thinking like a trader that observes the MA Cross Strategy.

The *Moving Average Cross Strategy* exploits the Exponential Moving Average (EMA) that is a type of moving average that gives more weight to recent observations, which means it's able to react quickly to recent trends variations. The EMA formula is shown below:

$$y_t = \frac{x_t + (1-\alpha)\, x_{t-1} + (1-\alpha)^2\, x_{t-2} + \cdots + (1-\alpha)^t\, x_0}{1 + (1-\alpha) + (1-\alpha)^2 + \cdots + (1-\alpha)^t}$$

Formula 1: the Exponential Moving Average for a given instance

Specifically, it uses two different EMAs, one shorter and one longer, and estimates the trend for the next period from the reciprocal value of the two indicators:

1. an increasing trend signal is spotted when the short-term average crosses the long-term average and rises above it,

2. a decreasing trend signal is triggered by a short-term average crossing long-term average and falling below it,
3. a flat trend could be identified instead when the difference between the two averages is below a certain threshold; more specifically, following the financial Technical Approach, we have set the threshold to 1% of the asset value in that specific moment.

Translating it graphically, we can identify the so-called Golden Cross (the start of a rising trend) and the Death Cross (the start of a decreasing trend) as follows:

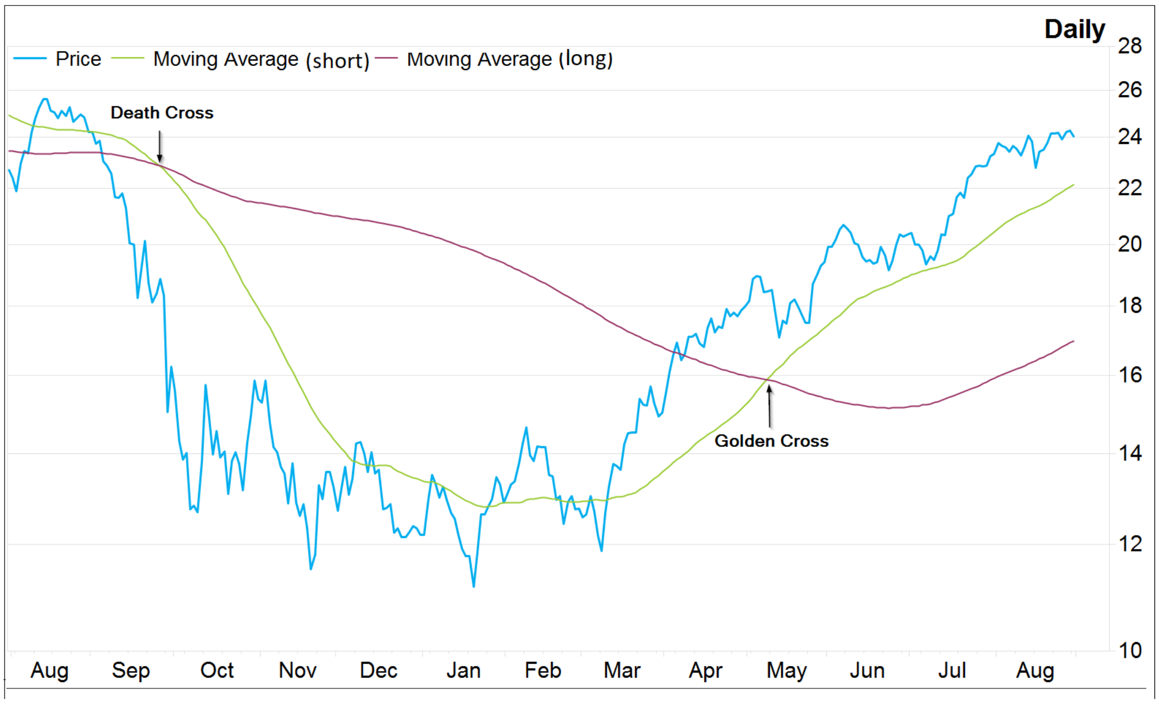


Figure 2 Example of the moving average cross strategy

Notice that EMA lengths, also called “look-back period”, can play a significant role on how effective the strategy is, and they should be chosen keeping in mind the chart time frame (one minute, daily, weekly, etc.) and the prediction time horizon.

In general, EMA couples with a short time span react much quicker to price changes than EMAs with a long look-back period, so as we collect daily data of the asset values and considering our model will make a short-term prediction, we choose a couple of EMAs with a short look-back period.

In particular, we selected three-day and six-day moving averages between the couples commonly used in short term financial trading, and we used them to estimate the trend of a specific day and to create the ground truth from which we trained our models.

In the following paragraphs, we explain our attempt to translate this *Moving Average Cross Strategy* for financial values into a prediction model based on classification.

3.2 Ground Truth Construction

We translated the above intuition starting from creating the ground truth from scratch, exploited to train the different models. During this phase our goal was generating a *Trend* attribute that describe the behavior of the asset in that specific day: up-trend, flat-trend, or down-trend.

First, we evaluated the short-term and long-term ema of the attribute Adj Close [Par. 4.1.1], that is the value of the asset at the closing time, with a granularity of one day. Then we evaluated the difference between short ema and long ema and we assign the classes:

- if the absolute value of the difference, in that specific day, is below the one percent of the Adj Close value in the same day, we associate the flat-trend class (0),
- if the absolute value of the difference is above the one percent of the Adj Close in that specific day and the value of the difference is positive, we associate the up-trend class (1),
- if the absolute value of the difference is above the one percent of the Adj Close in that specific day and the value of the difference is negative, we associate the down-trend class (-1).

As we can see in the plot below, the uptrend is identified by a green indicator, the down trend with a red indicator and the flat one with a yellow indicator.

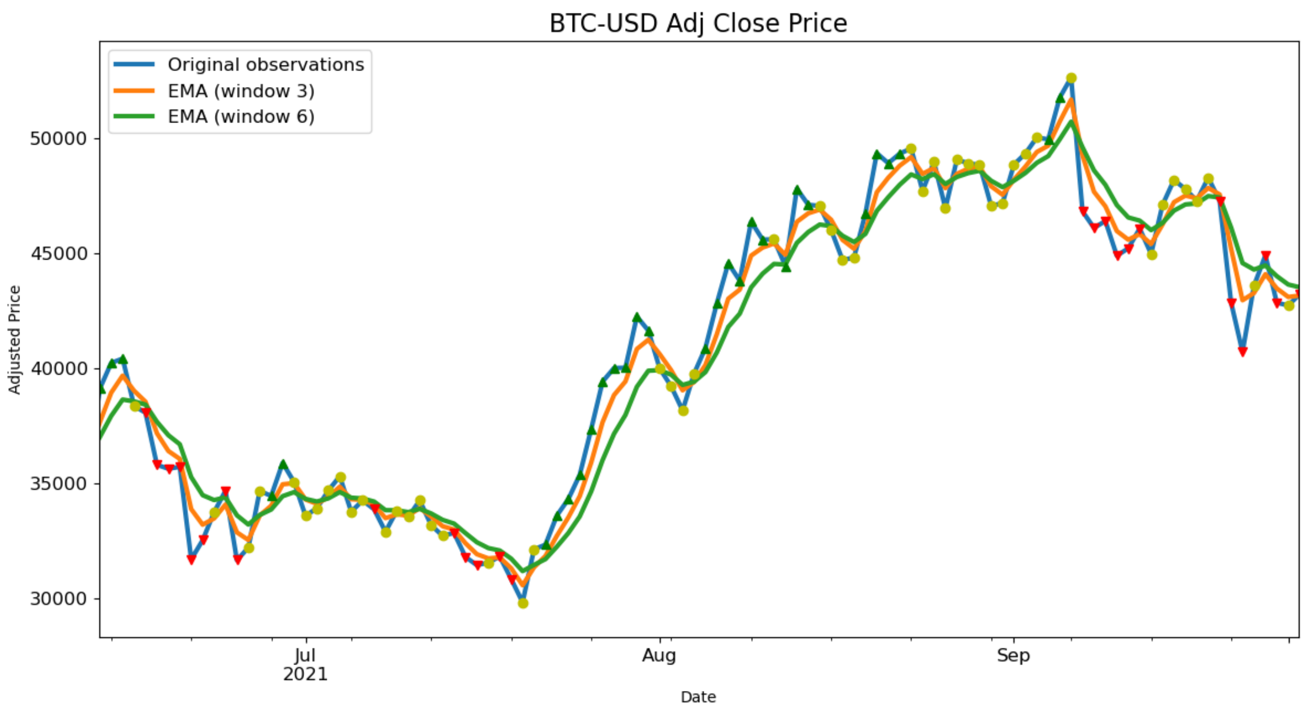
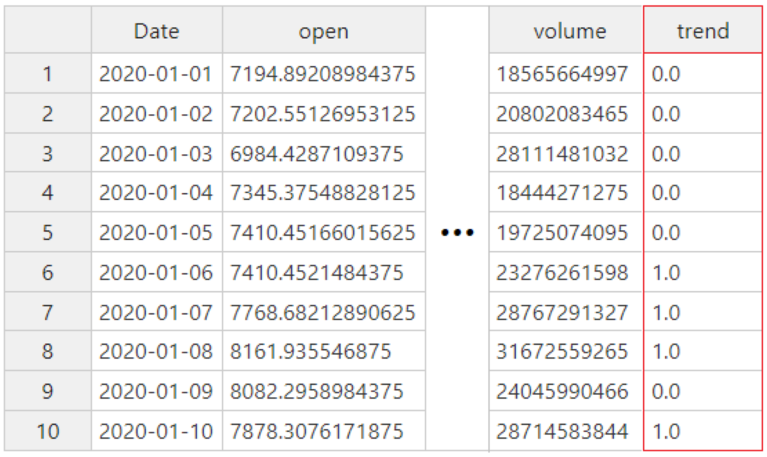


Figure 3 Dataset snapshot with label indicators

As we can notice, we were able to translate the theoretical idea of the *Moving Average Cross Strategy* quite effectively into the currency trend.

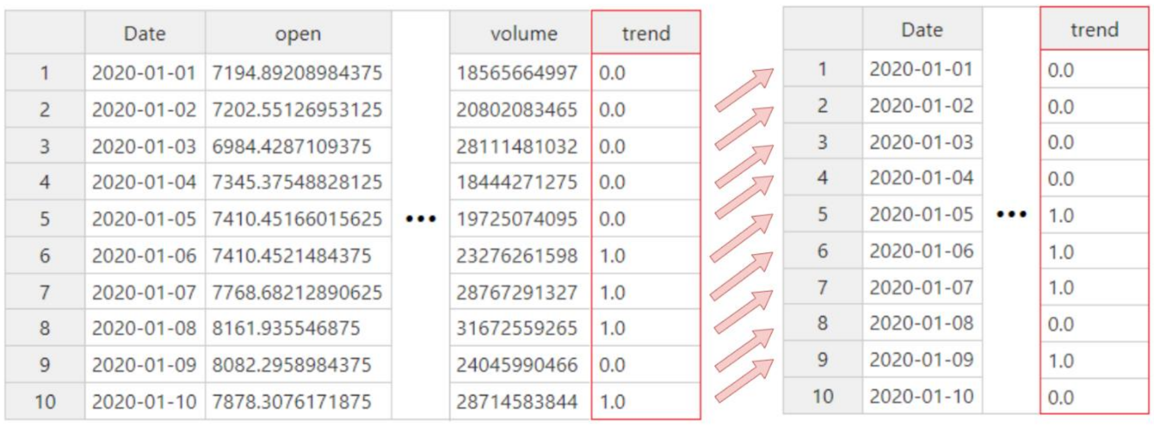
At this point, this is what our dataset was like with the addition of the class label Trend:



	Date	open		volume	trend
1	2020-01-01	7194.89208984375		18565664997	0.0
2	2020-01-02	7202.55126953125		20802083465	0.0
3	2020-01-03	6984.4287109375		28111481032	0.0
4	2020-01-04	7345.37548828125		18444271275	0.0
5	2020-01-05	7410.45166015625	•••	19725074095	0.0
6	2020-01-06	7410.4521484375		23276261598	1.0
7	2020-01-07	7768.68212890625		28767291327	1.0
8	2020-01-08	8161.935546875		31672559265	1.0
9	2020-01-09	8082.2958984375		24045990466	0.0
10	2020-01-10	7878.3076171875		28714583844	1.0

Figure 4 Dataset with trend labels

After generation of the labels, we need to introduce the concept of prediction to make the model forecast future value of the asset. So, we shift the Trend feature on the dataset to associate the features (High, Low, Open, Close, Adj Close, Short ema, ...) of one day to the trend of the day after.
In this way the model will be trained to classify the trend of the future day from the features provided in a specific moment.

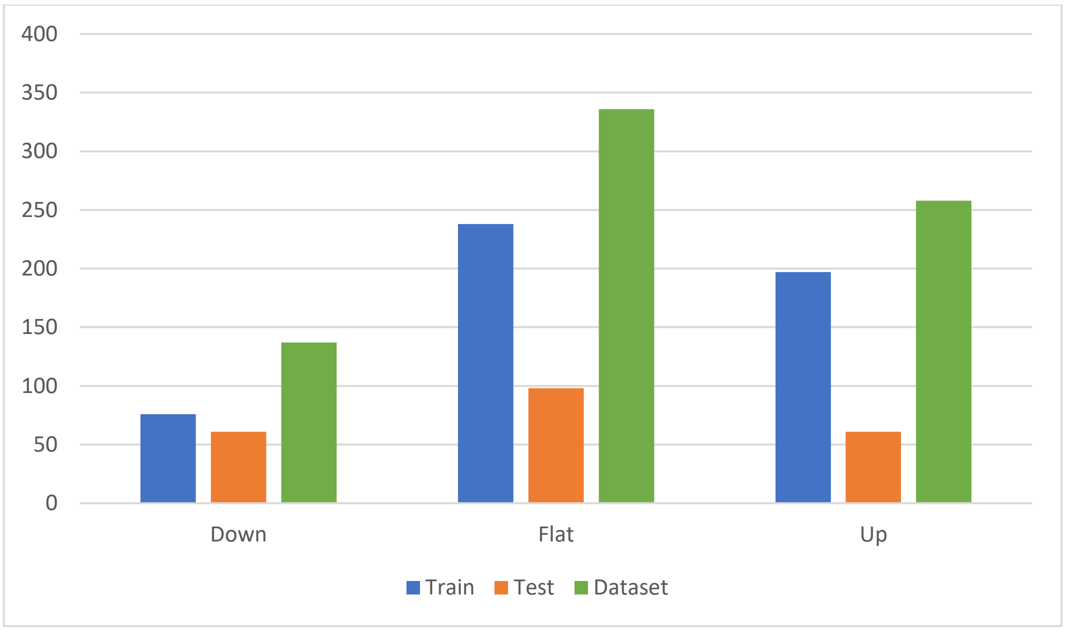


	Date	open		volume	trend
1	2020-01-01	7194.89208984375		18565664997	0.0
2	2020-01-02	7202.55126953125		20802083465	0.0
3	2020-01-03	6984.4287109375		28111481032	0.0
4	2020-01-04	7345.37548828125		18444271275	0.0
5	2020-01-05	7410.45166015625	•••	19725074095	0.0
6	2020-01-06	7410.4521484375		23276261598	1.0
7	2020-01-07	7768.68212890625		28767291327	1.0
8	2020-01-08	8161.935546875		31672559265	1.0
9	2020-01-09	8082.2958984375		24045990466	0.0
10	2020-01-10	7878.3076171875		28714583844	1.0

	Date		trend
1	2020-01-01		0.0
2	2020-01-02		0.0
3	2020-01-03		0.0
4	2020-01-04		0.0
5	2020-01-05	•••	1.0
6	2020-01-06		1.0
7	2020-01-07		1.0
8	2020-01-08		0.0
9	2020-01-09		1.0
10	2020-01-10		0.0

Figure 5 Dataset after the shift on trend labels

In the following histogram we show the distribution of the classes in the newly obtained dataset:



From the picture we can see that the dataset is slightly imbalanced in favor of the flat-trend instances (as we would normally expect).
As the unbalancing is not so strong, we decided not to apply any rebalancing stages.

4 Machine Learning

4.1 Pre-processing

4.1.1 Feature Generation

Attribute construction (or feature generation) is the procedure in which new attributes are constructed and added from the given set of attributes to help the mining process.
We decided to exploit it as it helps improving accuracy and understanding the structure in high dimensional data.

The aim is to create new attributes (features) that can capture the important information in a data set more effectively than the original ones. By combining attributes, attribute construction can discover missing information about the relationships between data attributes that can be useful for knowledge discovery.

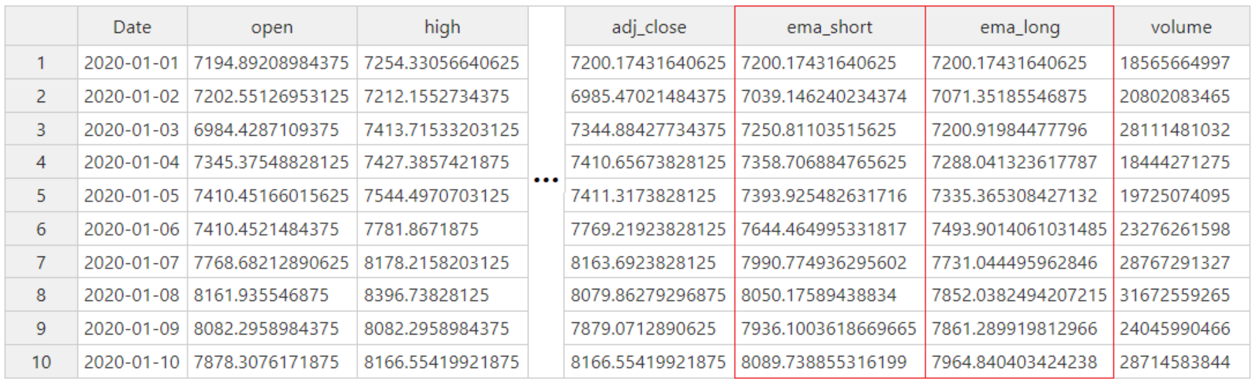
So, as described in the previous paragraph, starting from the raw financial attributes we had to compute the value for the two EMAs, respectively short-window ema with a three-day width and long-window ema with a six-day width. Those two newly created values were used to create the ground truth of our dataset (to set the value of the class attribute *Trend*).

To generate the values for the attribute *short_ ema* and *long_ ema* for each instance of the dataset, we computed the formula described above [*EMA Formula*] exploiting the *ewm()* function inside Pandas package:

```
# evaluate the exponential moving average of the short and long windows
    ema_short = dataframe['Adj Close'].ewm(span=SHORT_WINDOW).mean()
    ema_long = dataframe['Adj Close'].ewm(span=LONG_WINDOW).mean()
```

As we can spot from the code snippet above, we decided to follow once again the best practices as defined in the *Moving Average Cross Strategy* that uses the financial value of the Adjusted Close for computing the EMA value.

Once we added this couple of attributes, this is how our data frame looks like:



	Date	open	high		adj_close	ema_short	ema_long	volume
1	2020-01-01	7194.89208984375	7254.33056640625		7200.17431640625	7200.17431640625	7200.17431640625	18565664997
2	2020-01-02	7202.55126953125	7212.1552734375		6985.47021484375	7039.146240234374	7071.35185546875	20802083465
3	2020-01-03	6984.4287109375	7413.71533203125		7344.88427734375	7250.81103515625	7200.91984477796	28111481032
4	2020-01-04	7345.37548828125	7427.3857421875		7410.65673828125	7358.706884765625	7288.041323617787	18444271275
5	2020-01-05	7410.45166015625	7544.4970703125	...	7411.3173828125	7393.925482631716	7335.365308427132	19725074095
6	2020-01-06	7410.4521484375	7781.8671875		7769.21923828125	7644.464995331817	7493.9014061031485	23276261598
7	2020-01-07	7768.68212890625	8178.2158203125		8163.6923828125	7990.774936295602	7731.044495962846	28767291327
8	2020-01-08	8161.935546875	8396.73828125		8079.86279296875	8050.17589438834	7852.0382494207215	31672559265
9	2020-01-09	8082.2958984375	8082.2958984375		7879.0712890625	7936.1003618669665	7861.289919812966	24045990466
10	2020-01-10	7878.3076171875	8166.55419921875		8166.55419921875	8089.738855316199	7964.840403424238	28714583844

Figure 6 Dataset after ema addition

4.1.2 Feature Selection

Ideally, we would like to implement these two intuitions in our attribute selection:

1) we want to select features that are strongly correlated with the classes, so correlation between features and output,
2) we want to avoid choosing attributes that are correlated each other, because they don't add any kind of information.

We then decided to perform the supervised heuristic approach based on the Mutual Information, as quite effective for classification problems. This selection is done with the following formula:

$$G = I(C, f_i) - \frac{1}{S} \sum_{f_s \in S} NI(f_i, f_s)$$

Where the two intuitions written above are respectively translated into the first part and second part of the formula. Here is the relative code snippet:

```python
def ranking_attributes_contribution(dataset):
    X = dataset[predictors]  # independent columns
    y = dataset[['trend']]  # target column

    best_features = SelectKBest(score_func=mutual_info, k='all')
    fit = best_features.fit(X, y)
```

The k-best selection provided the following contribution rank for each attribute:

Feature	Score
low	0.150541
adj_close	0.144122
high	0.138869
open	0.123823
ema_short	0.118101
ema_long	0.093307
volume	0.056170

As we deal with a small number of features, we ended up keeping all the attributes because with a subset of them we obtained lower performances.

4.1.3 Data Normalization

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not look like standard normally distributed data.

For instance, many elements used in the objective function of a learning algorithm, assume that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

Standardizing is the process of converting a raw score into a standard score, commonly called *z-score*. Computing a z-score requires knowing the mean and standard deviation of the complete population to which a data point belongs.

We then decided to standardize features by removing the mean and scaling to unit variance using z-score normalization. The standard score of a raw score x is converted into a standard score by:

$$z = \frac{x - \mu}{\sigma}$$

where μ is the mean of the training samples, and σ is the standard deviation of the training samples. The absolute value of z represents the distance between that raw score x and the population mean in units of the standard deviation. z is negative when the raw score is below the mean, positive when above.
We exploited the StandardScaler class defined inside the S*ci-kit learn* package:

```python
def create_preprocessor(predictors):
    numeric_transformer = Pipeline(steps=[
        ('scaler', StandardScaler()),
        ('pca', PCA())
    ])
```

StandardScaler helped us standardize the dataset's features onto unit scale (mean = 0 and variance = 1) which is a requirement for the optimal performance of many machine learning algorithms. In particular PCA is affected by the scaling process, so we need to scale the features before applying the principal component analysis.

This method of normalization is useful when the actual minimum and maximum of an attribute are unknown (as in our application we cope with financial values that variate continuously), or when there is the need to cope with outliers that would otherwise dominate (like in other normalization techniques, e.g., min-max).

In the z score we don't fix the new range, we only use the *mean* value and the *std deviation.*

In particular, the denominator is the std deviation, so it reflects how much the data are spread, and it has the effect of making the data vary in the same range.

4.1.4 Feature trasformation

We tried to obtain a reduced representation of the dataset which could be possibly smaller in volume but could yet produce the same (or almost) analytical results.
For this purpose, we decided to apply the Principal Component Analysis (PCA), as being very effective with numerical data.

Applying PCA maintaining all the principal components will attempt to un-correlate data: in cases where features are highly correlated before its application, this method could lead to increased performances for some algorithms, and it let us verify the Naïve Bayes hypothesis (independency between attributes).

4.1.4.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a widely popular technique used in the field of statistical analysis. Considering an initial dataset of N data points described through P variables, its objective is to reduce the number of dimensions needed to represent each data point, by looking for the K ($1 \leq K \leq P$) principal components.

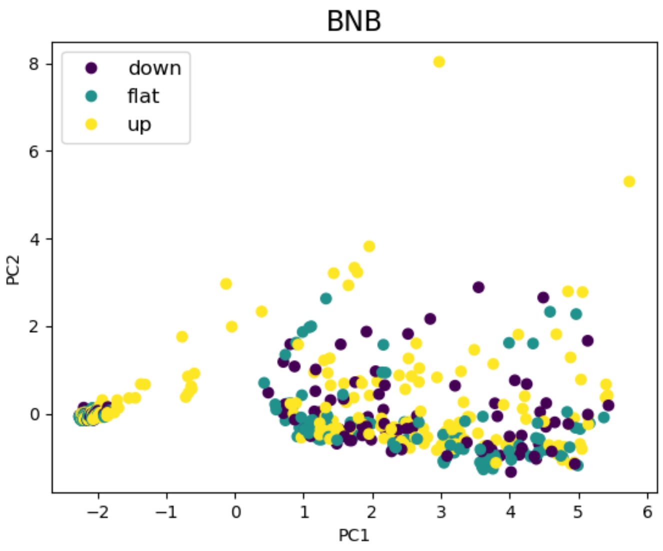
These principal components are calculated iteratively, by looking each time for the axis that explains as much inertia as possible, while being orthogonal to the previous principal components. In the end, we can directly describe the data points through the K principal components.

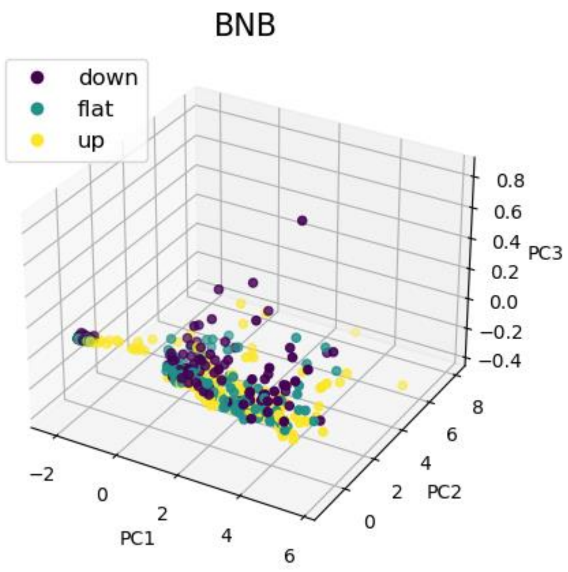
Even though they are harder to interpret, they enable us to keep as much information as possible from the original dataset.

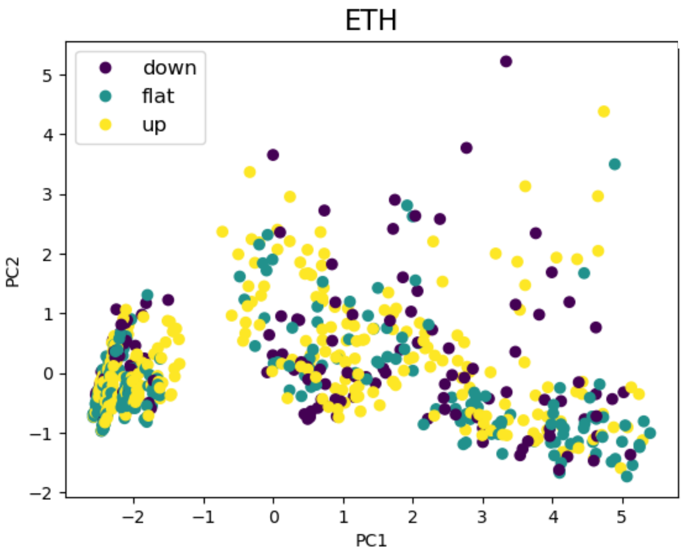
We exploited the PCA class defined inside the sci-kit learn package:

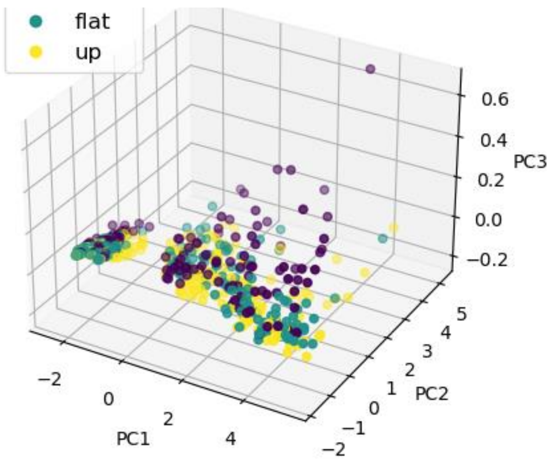
```
def create_preprocessor(predictors):
    numeric_transformer = Pipeline(steps=[
        ('scaler', StandardScaler()),
        ("pca", PCA())
    ])
```

Another common application of PCA is for data visualization; in our case, for instance, we exploited it to reduce our 7-dimensional dataset into 2 or 3 dimensions so that we could plot and understand the data better. We obtained the following 2D and 3D scatter plot:









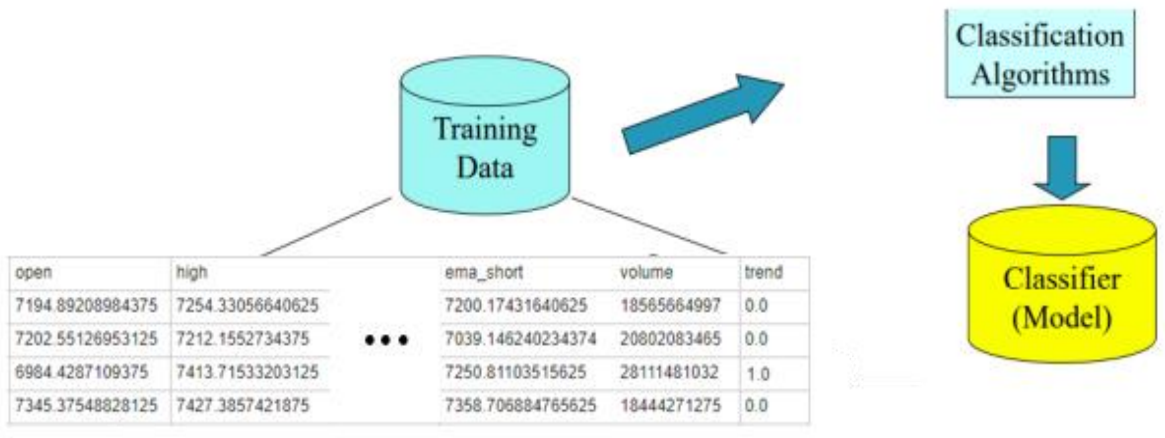
This PCA plots makes it possible to visualize strong patterns, such as groups of similar observations, in our dataset. The key difference between 2D PCA and 3D PCA is the number of principal components being selected for plotting.

In PCA, principal components are constructed to capture the most variation in the dataset: PC1 describes the most variation, PC2 describes the second most variation, and so forth.

4.2 Classification

The problem our application aims to resolve is a multiclass classification one; indeed, as previously anticipated, each tuple relative to a trend day can belong to one of the three classes, respectively down-trend (class label -1), flat-trend (class label 0) or up-trend (class label 1).

In order to classify a new tuple whose trend is not known, we had to build and train a classification model for each cryptocurrency (three different classifiers, after choosing the best classification algorithm).



open	high		ema_short	volume	trend
7194.89208984375	7254.33056640625		7200.17431640625	18565664997	0.0
7202.55126953125	7212.1552734375	•••	7039.146240234374	20802083465	0.0
6984.4287109375	7413.71533203125		7250.81103515625	28111481032	1.0
7345.37548828125	7427.3857421875		7358.706884765625	18444271275	0.0

Figure 7: Model construction for each cryptocurrency

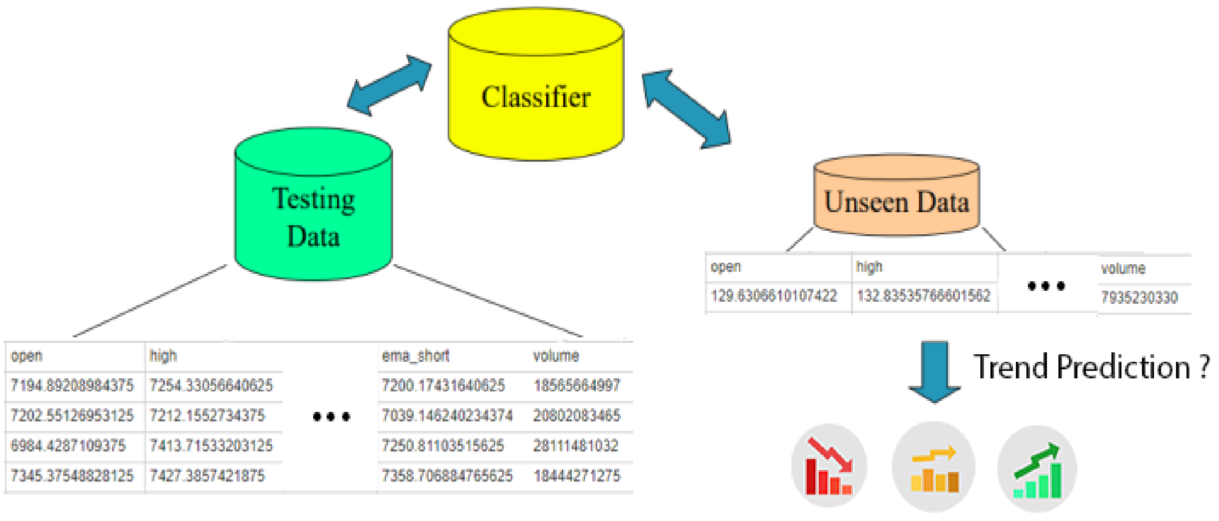
So, in the first phase of our analysis we tested various classifiers in combination with different attribute selection algorithms.
The classification task was implemented with different machine learning models, in order to find the most accurate one. In each case, the models are fitted to the training data set and their performance is tested on the testing data set.
Among the various classifiers tested we have chosen the following ones:

1. K-Nearest Neighbors Classifier,
2. Logistic Regression Classifier,
3. Gaussian Naïve Bayes

We also evaluated ensemble methods such as:

1. AdaBoost
2. Random Forest.

Our goal was to obtain the highest accuracy and F-Measure; in particular, we targeted at least an accuracy of 70%. To build the best model possible, we decided to test different classifiers and evaluate their results on our dataset.



open	high	•••	ema_short	volume
7194.89208984375	7254.33056640625		7200.17431640625	18565664997
7202.55126953125	7212.1552734375		7039.146240234374	20802083465
6984.4287109375	7413.71533203125		7250.81103515625	28111481032
7345.37548828125	7427.3857421875		7358.706884765625	18444271275

open	high	•••	volume
129.6306610107422	132.83535766601562		7935230330

Figure 8: Model usage for each cryptocurrency

5 Models Evaluation and Selection

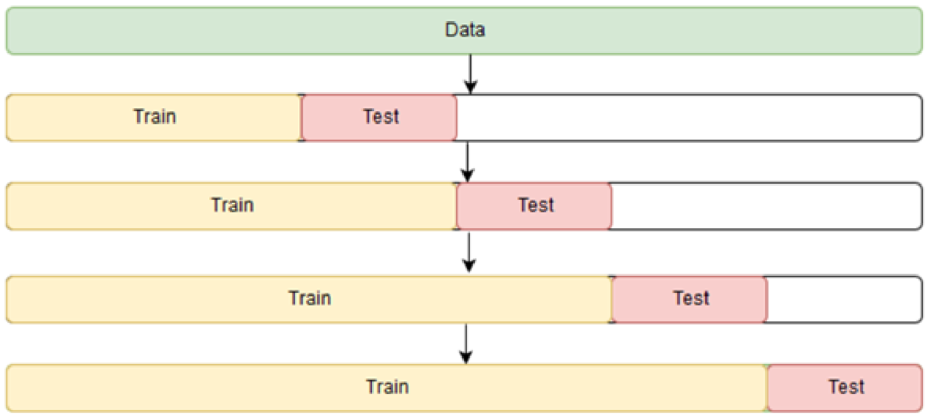
5.1 Time Series Split Cross-Validation

Training and evaluating machine learning models usually require a training set and a test set. In most cases, train and test splitting is done by randomly taking 20 / 30 % of the data as test data, (unseen by the model) and using the rest for training.

In order to provide statistically robust model, we perform a Cross Validation process and then we chose the model that performs better in different iterations.

We cannot use the traditional CV process in Time Series, in fact we cannot choose random samples and assign them to either the test set or the train set, because it makes no sense to use the values from the future to forecast values in the past. In simple word, we want to avoid future-looking when we train our model. There is a temporal dependency between observations, and we must preserve that relation during testing.

For this reason, we used *time-based cross validation*, a method taken from the time-series field, which forms a type of "sliding window" training approach.
We start with a small subset of the dataset for training purpose and then we forecast later data point checking the accuracy for this prediction. The same forecasted data points are then included as part of the next training dataset and subsequent data points are forecasted.
To make things intuitive, here is an image of our solution:



The idea for time series splits is that, at the first iteration, the model is trained on the trend values from January to June and validates on July's and August's data, and for the next iteration, model is trained on data from January to August, and validate on September's and October's data, and so on to the end of the training set. In this way dependency between the records is respected.

In our case, we decide to implement the following folds, starting from 75% for the training set and 25% for the test set in the first fold, then we increment the training set with the test set of the previous iteration.

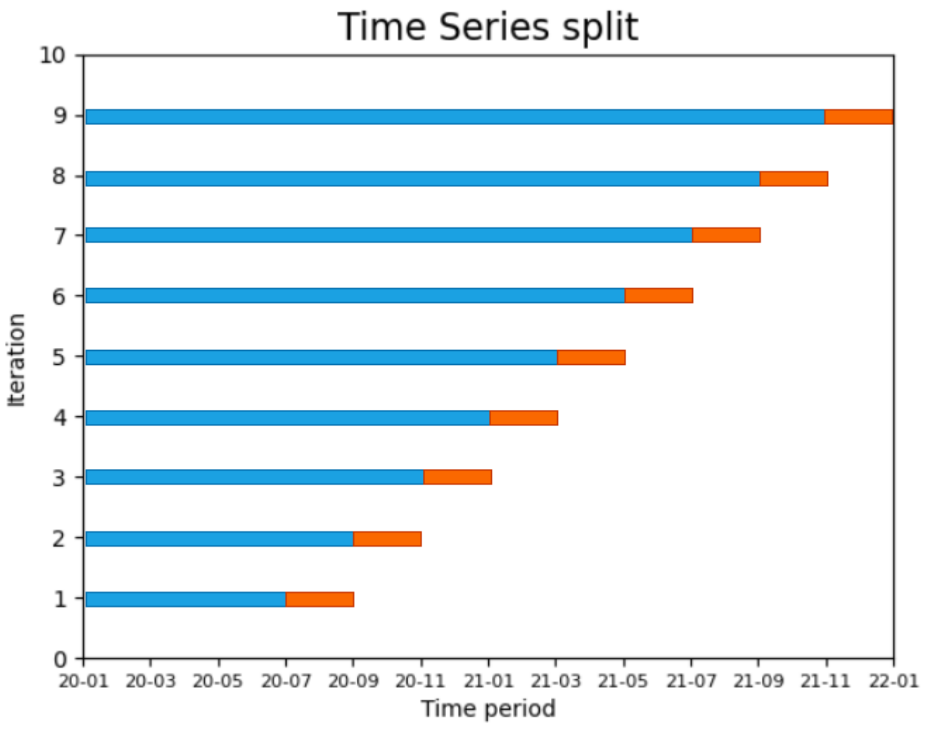


Figure 9 Incremental Time Series split

We can intuitively interpret the horizontal axis as a time progression line since we have not shuffled the dataset and maintained the chronological order of the records.

5.2 Comparison Metrics

Inside the following tables we reported the comparison metrics for each class and for every classifier. These values are evaluated considering the mean of the different metrics in each iteration of the CV process. The first one refers to the Bitcoin currency results:

Classifier Name	Class	Precision	Recall	F1-score	Accuracy
RandomForestClassifier	-1.0	0.58	0.48	0.49	0.709
	0.0	0.63	0.66	0.63	
	1.0	0.82	0.78	0.77	
AdaBoostClassifier	-1.0	0.42	0.35	0.36	0.652
	0.0	0.58	0.66	0.60	
	1.0	0.71	0.75	0.71	
KNeighborsClassifier	-1.0	0.42	0.30	0.23	0.470
	0.0	0.36	0.33	0.32	
	1.0	0.42	0.72	0.51	
LogisticRegression	-1.0	0.53	0.13	0.19	0.611
	0.0	0.45	0.68	0.53	
	1.0	0.72	0.72	0.63	
GaussianNB	-1.0	0.39	0.60	0.39	0.598
	0.0	0.45	0.42	0.43	
	1.0	0.72	0.65	0.65	

The second one refers to the Ethereum currency results:

Classifier Name	Class	Precision	Recall	F1-score	Accuracy
RandomForestClassifier	-1.0	0.53	0.64	0.56	0.669
	0.0	0.56	0.50	0.51	
	1.0	0.77	0.82	0.77	
AdaBoostClassifier	-1.0	0.50	0.45	0.42	0.589
	0.0	0.44	0.63	0.49	
	1.0	0.88	0.55	0.64	
KNeighborsClassifier	-1.0	0.29	0.09	0.12	0.446
	0.0	0.23	0.16	0.13	
	1.0	0.41	0.84	0.53	
LogisticRegression	-1.0	0.85	0.40	0.49	0.631
	0.0	0.41	0.37	0.34	
	1.0	0.66	0.92	0.72	
GaussianNB	-1.0	0.30	0.86	0.43	0.467
	0.0	0.23	0.17	0.19	
	1.0	0.73	0.48	0.53	

The third one refers to the Binance currency results:

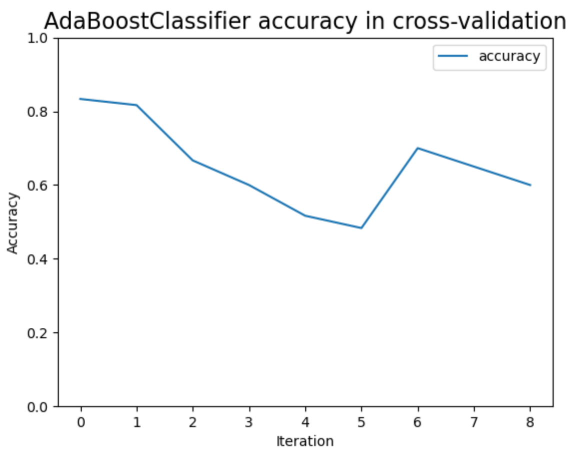
Classifier Name	Class	Precision	Recall	F1-score	Accuracy
RandomForestClassifier	-1.0	0.45	0.56	0.45	0.580
	0.0	0.53	0.55	0.52	
	1.0	0.74	0.65	0.67	
AdaBoostClassifier	-1.0	0.48	0.58	0.50	0.583
	0.0	0.44	0.53	0.48	
	1.0	0.79	0.55	0.61	
KNeighborsClassifier	-1.0	0.43	0.29	0.25	0.439
	0.0	0.28	0.35	0.30	
	1.0	0.46	0.64	0.46	
LogisticRegression	-1.0	0.50	0.35	0.38	0.583
	0.0	0.50	0.48	0.45	
	1.0	0.68	0.81	0.67	
GaussianNB	-1.0	0.28	0.57	0.35	0.470
	0.0	0.28	0.17	0.17	
	1.0	0.60	0.70	0.57	

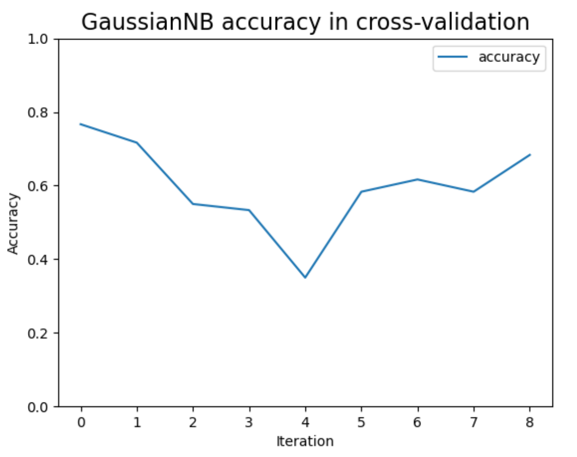
5.3 Model Selection

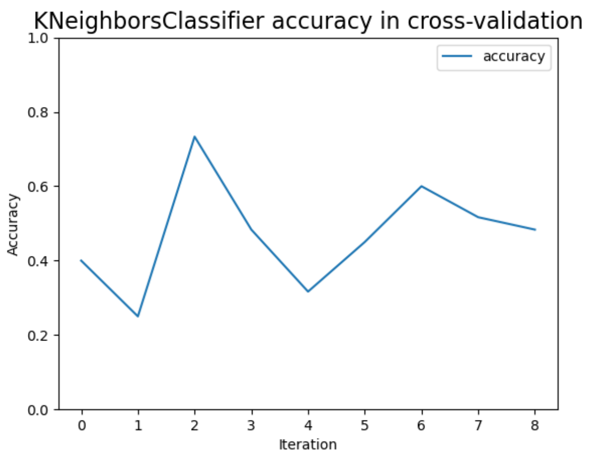
We made some qualitative choices for selecting the best classifier. Specifically, in addition to the metrics results obtained from the *Time Series Cross Validation* (average values upon all folds), we generated the plots of the Accuracy and F1-score during the various iterations.

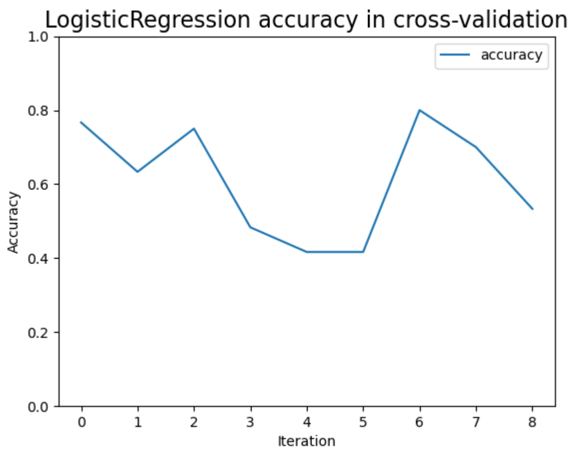
The following graphs refers to the performance of every classifier, in every incremental fold of the CV (e.g., first plot represent the performances on six-month training set and two-month the test set, the second result eight-month training set and two-month test set and so on).

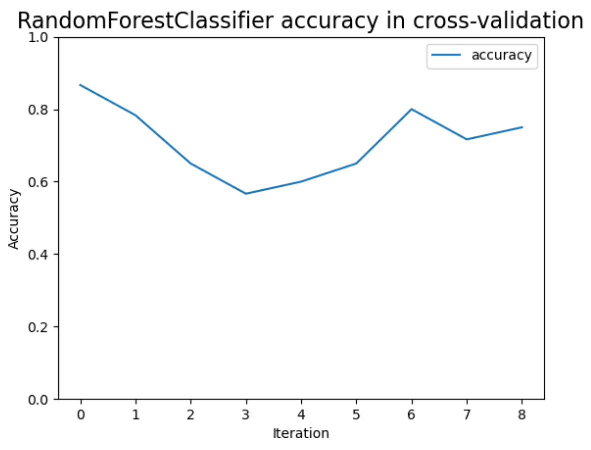
Here we reported the accuracy result throughout the cross-validation iterations (just for Bitcoin, for sake of simplicity):



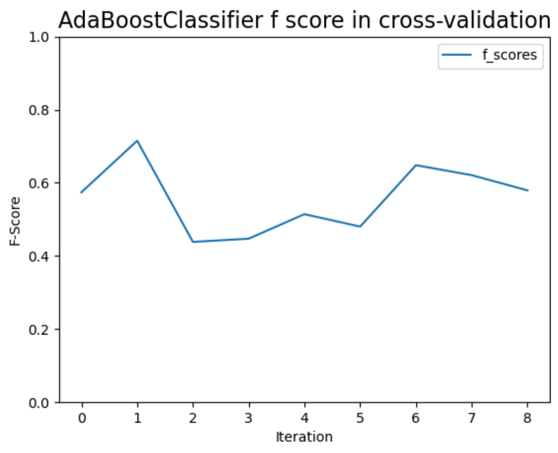


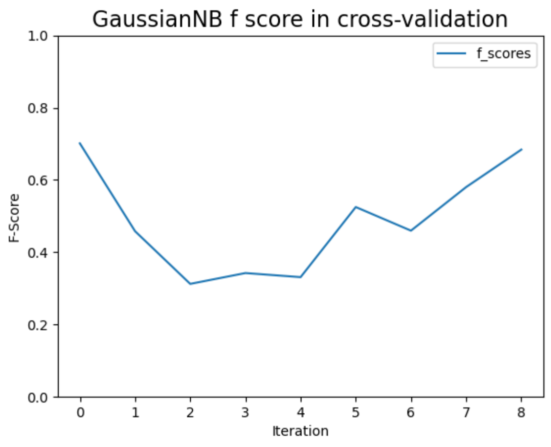


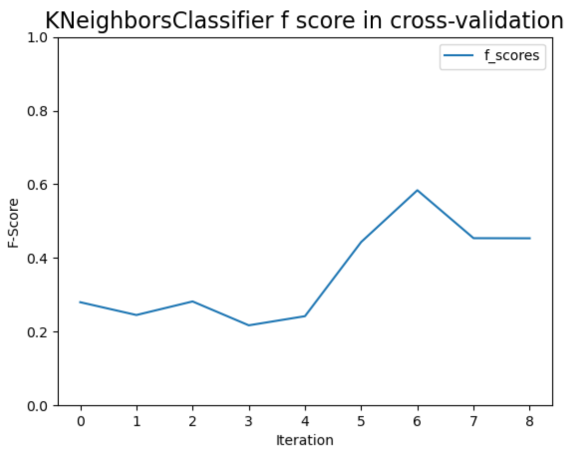


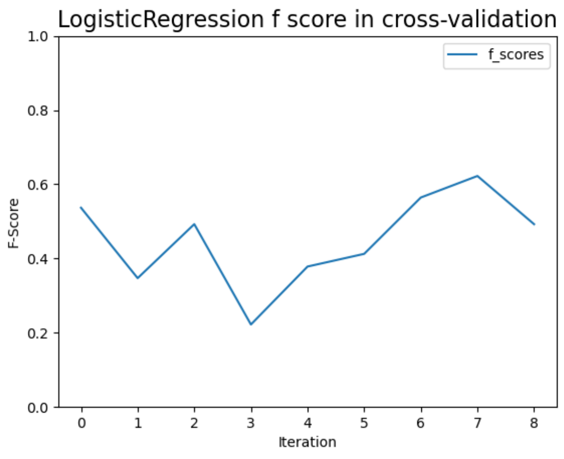


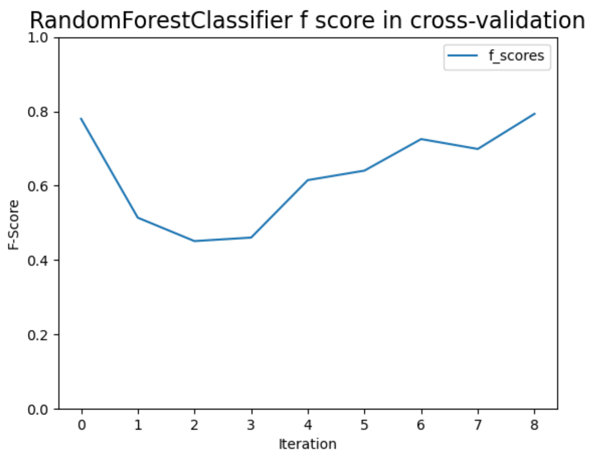
And the average F1-score of the three classes for each classifier, in every cross-validation iteration (just for Bitcoin, for sake of simplicity):











As we can notice from the various trends above, there are some down-spikes that represent the drifts corresponding to performance losses, caused by a smaller support of one of the three classes inside a specific fold; this class support distribution reflects sudden changes in currency trend (e.g., during a raising trend period there will be fewer down-trend class instances and vice versa).

So, we chose the classifier that has the least number of down-spikes (the most stable trend both in accuracy and F1-score), meaning that it corresponds to the most robust classifier.

Analyzing the graphs and the average results from the previous paragraph [Par. 5.2], we decided to pick RandomForest as the best overall classifier. In fact, it seemed to be the most resilient and the most robust against the trend drifts; in addiction to that, its accuracy and f1-score results seems to raise at each incremental temporal-fold during the cross-validation.

Very similar graphs were obtained for the other two currencies (ETH, BNB), so RandomForest was the one selected also for them as its performance was the most robust upon trend variances.

Once we selected the final classifier, we trained it using the whole dataset for each cryptocurrency, as a result we obtained three trained models. Below we reported the corresponding metrics:

Currency	Selected Model	Class	Precision	Recall	F1-score	Support	Accuracy
BTC	Random Forest	-1.0	0.73	0.54	0.62	61	0.722
		0.0	0.67	0.77	0.71	98	
		1.0	0.81	0.84	0.82	61	

Currency	Selected Model	Class	Precision	Recall	F1-score	Support	Accuracy
ETH	Random Forest	-1.0	0.83	0.74	0.78	65	0.618
		0.0	0.52	0.15	0.23	73	
		1.0	0.55	0.94	0.69	82	

Currency	Selected Model	Class	Precision	Recall	F1-score	Support	Accuracy
BNB	Random Forest	-1.0	0.74	0.58	0.65	59	0.641
		0.0	0.59	0.43	0.50	84	
		1.0	0.63	0.92	0.75	77	

These models have been exported and saved to be then loaded from the front-end, which provides a trend prediction to the user of the web app.

6 Design Choices and Implementation

6.1 Application Architecture

Crypto Predictor is an application based on the *client-server* paradigm. The general architecture is shown below:

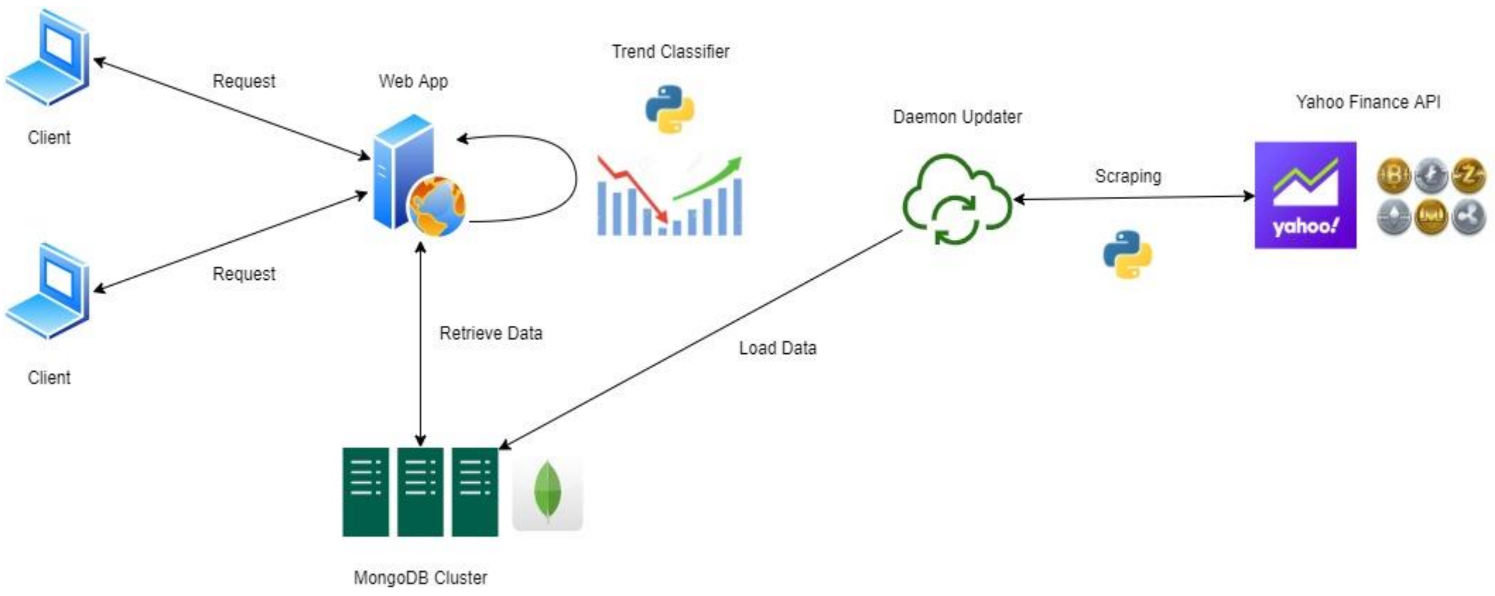


Figure 10 Application architecture example

6.2 MongoDB Design

This document database contains three collections, that store information about the selected crypto currencies.

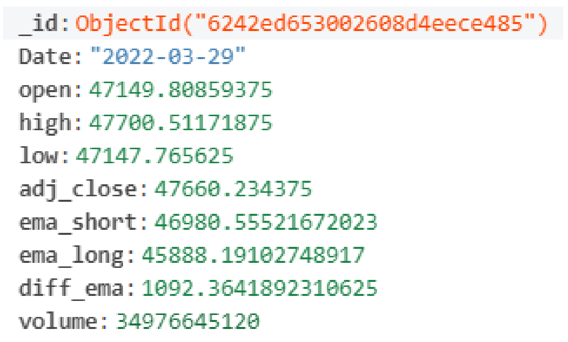
6.2.1 Bitcoin Collection

A document inside *Bitcoin* collection contains all information about the historical trend of that currency. Each document stores several fields of stock values for a certain day in the past (provided by Yahoo Finance API), and some other that contains the results of the application-domain computation (EMAs).

Field	Description
_id	Corresponds to the ID generated by MongoDB
Date	Contains the date of the historical stock value
open	Contains the Open Price of the cryptocurrency, the price at which stock opened
high	Contains the Highest price of the stock on that day
low	Contains the Lowest price of the stock on that day

adj_close	Contains the Close Price of the cryptocurrency, the price at which stock closed
ema_short	Contains the result of the computation of the exponential moving average with window size = 3
ema_long	Contains the result of the computation of the exponential moving average with window size = 6
diff_ema	Contains the difference between the two EMA values
trend	Contains one of the three possible value [-1,0,1] representing respectively a down-trend day, flat day or up-trend day

Below we can see the structure of an example document inside the currency collections:



```
_id: ObjectId("6242ed653002608d4eece485")
Date: "2022-03-29"
open: 47149.80859375
high: 47700.51171875
low: 47147.765625
adj_close: 47660.234375
ema_short: 46980.55521672023
ema_long: 45888.19102748917
diff_ema: 1092.3641892310625
volume: 34976645120
```

Figure 11 Example document of the coin collection

6.3 Dataset Periodic Updates

Our datasets, one for each coin, are distributed in different collections on a MongoDB cluster. We need to keep data updated to make predictions for the future days, retrieving information from Yahoo Finance API.

We define a class named Updater responsible for the updating of all the collections, it starts with the frontend and performs the following actions for each collection:

1. Retrieves the date of the most recent record in that collection,
2. Makes a request to Yahoo Finance API to retrieve data of the missing days,
3. Evaluates the statistical indicators from the data of the missing dates to complete each record,
4. Inserts the final records to the specific collection, making the historical data up to date.

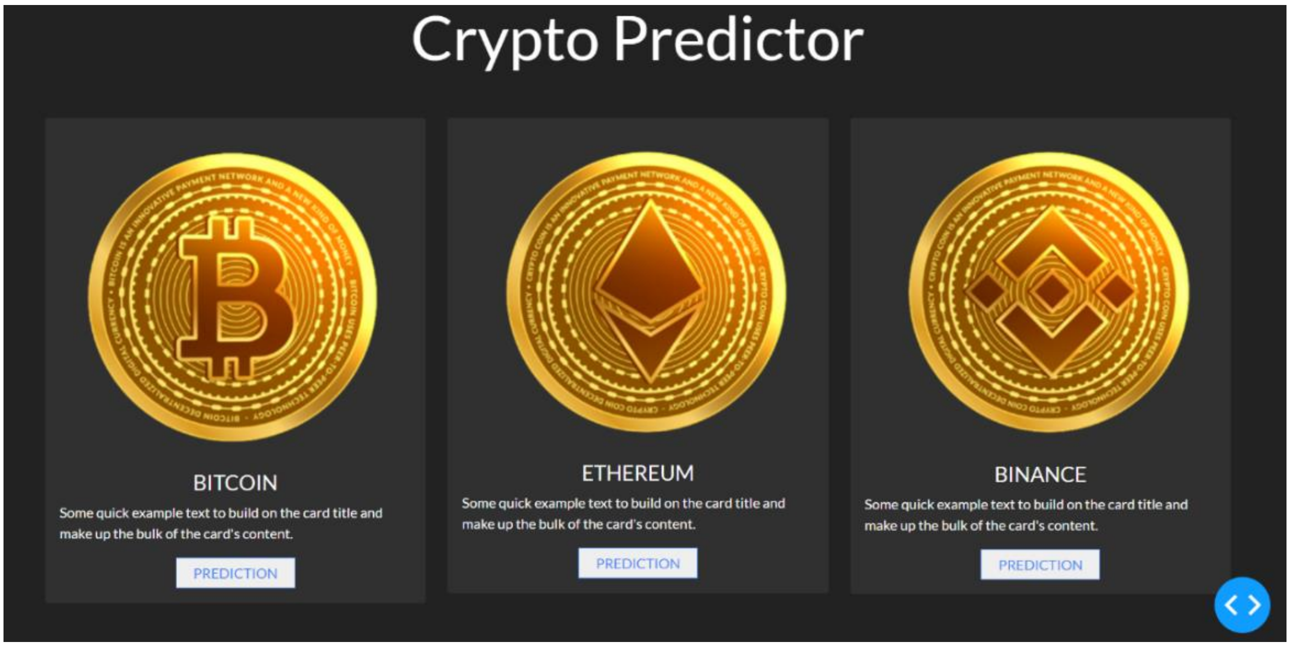
6.4 Frontend

6.4.1 Dash

In order to develop a user-friendly application and meet our non-functional requirements, we exploited the *Python Flask* web framework to create the front-end.

In addition to that, we made an interactive interface to better browse through the functionalities using *Dash,* which is an open-source Python framework used for building analytical web applications. This powerful library allows us to simplify the development of that data-driven applications.

Crypto Predictor's home page along with some other visual of the application while running, are reported below. Users can select the preferred cryptocurrency from the index page:



Some quick example text to build on the card title and make up the bulk of the card's content.

Some quick example text to build on the card title and make up the bulk of the card's content.

Some quick example text to build on the card title and make up the bulk of the card's content.

Figure 12 Main menu in which user can choose the preferred coin

Then the application will show a graph of the historical values of the coin together with the possibility to trigger the trend prediction for the following days by clicking the *Prediction* button:

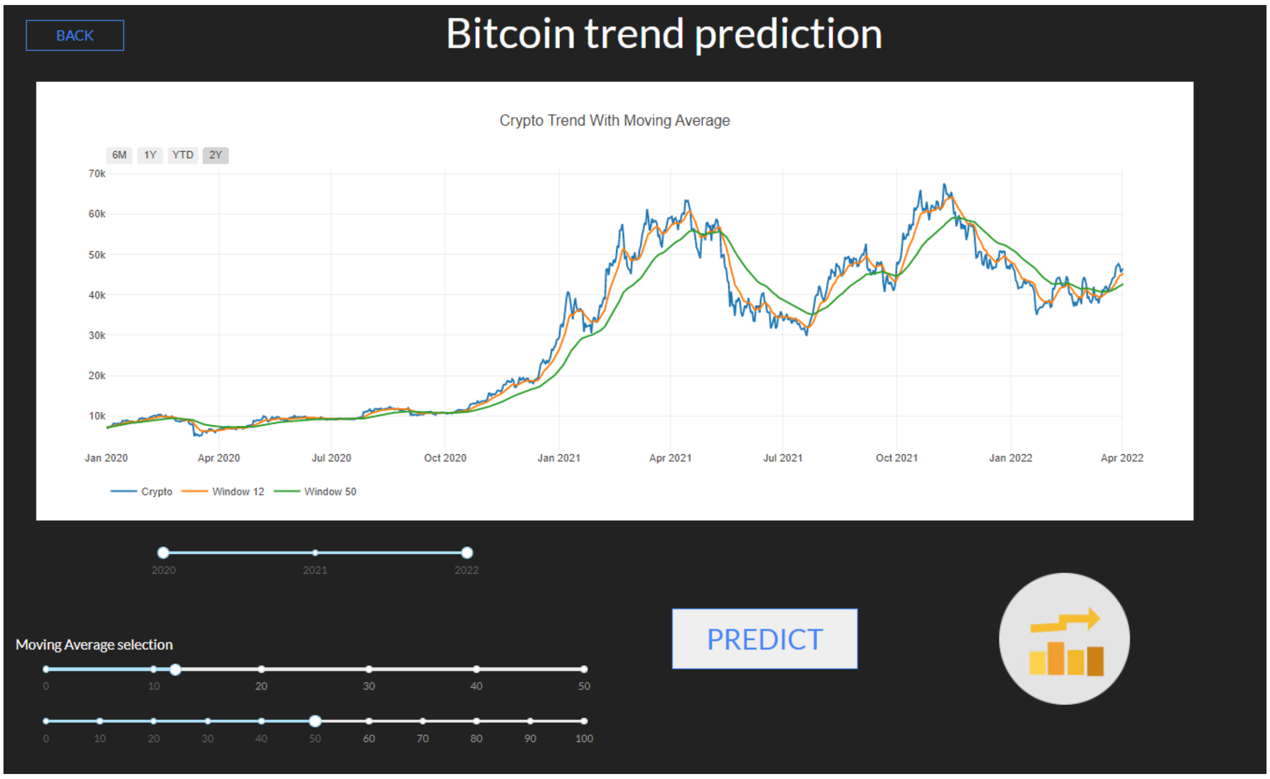


Figure 13 Snapshot of the coin trend page

7 Conclusions

In our study, we develop and implement a cryptocurrency trading strategy based on artificial intelligence. In the first part, we perform a comparative study of different classification methods. Examining several metrics, we found that the prediction of the trend of the three different currencies is classified by the Random Forest classifier with the best overall performance. Based on this, we use the Random Forest classifier to implement the strategy.

The results collected from the metrics of the final classifier for each currency are not outstanding, but still acceptable, as our goal was just to provide a support to crypto traders. Anyway, an error rate like ours of about 25-30% can be considered consistent with real life, as it reflects that it is not trivial to become rich with trading supported by machine learning.

The reasons for such poor performances could be caused by the choice of Classification to tackle the prediction problem; in fact, as we know it may not be considered the best Machine Learning approach for making predictions.
Another cause could be addressed to the ground truth construction we performed for our dataset, as it might not be the most valuable and reliable one. A more accurate and robust analysis of the labeling phase could lead to better results and usefulness.

In the future, it may be worthwhile to include additional indicators in the model, like EMAs with a different look-back period, and to involve polarity of social media posts in the dataset.

Another additional feature could be built around the change of the width for the EMAs; for instance, increasing the couple of window width from 2-6 days to 30-60 or higher to get a more long-term prediction.