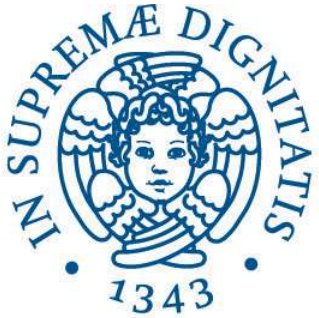


UNIVERSITÀ DI PISA

Master's degree in Artificial Intelligence and Data Engineering

Data Mining and Machine Learning

Stock Sentiment

Academic year 2021-2022

Edoardo Ruffoli, Tommaso Baldi

Github link: https://github.com/edoardoruffoli/StockSentiment

Table of contents

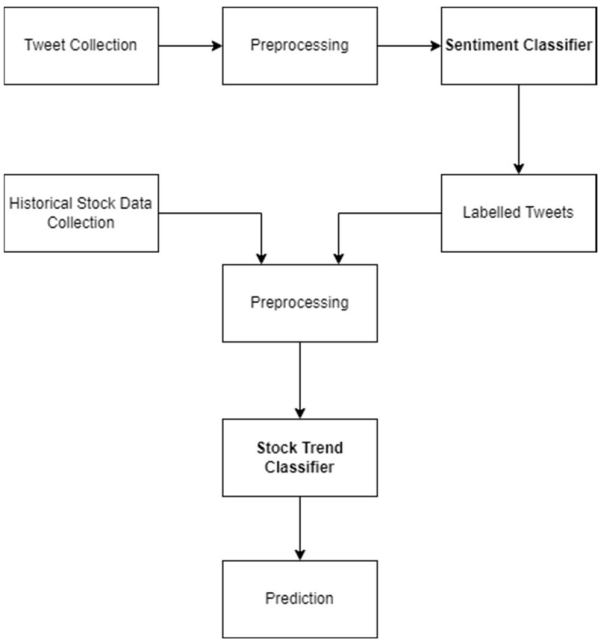
Introduction

StockSentiment is a web financial tool, which helps users with financial investments providing analysis on the opinions expressed by people on stock companies. More specifically, it provides a prediction on the next close price of the stock based on the previous days trend and on the polarity of the recent tweets related to the stock.
For the sake of simplicity, we included in the application only four American companies: Microsoft (Nasdaq: MSFT), Apple (Nasdaq: AAPL), Amazon (Nasdaq: AMZN) and Google (Nasdaq: GOOGL).

Regarding the machine learning part of the project, two classification models were realized:

- a **Tweet Sentiment Classifier**, used to classify the tweets related to the various stocks that then will be used to compute the daily sentiment polarity.
- a **Stock Trend Predictor**, used to give to the user the prediction of the next close price of the selected stock.

The following picture shows the workflow adopted by the application to perform a stock trend prediction.



Dataset

To train the tweet sentiment classifier, we used a labelled dataset of stock market related tweets that contains over 27k financial related tweets.
The dataset is available on Kaggle at the following link:
https://www.kaggle.com/utkarshxy/stock-markettweets-lexicon-data.

Always regarding the sentiment classifier, we also used a manually labelled tweet dataset to better evaluate the model. This dataset contains almost 800 tweets, that were scraped using the data collecting functions of the application.

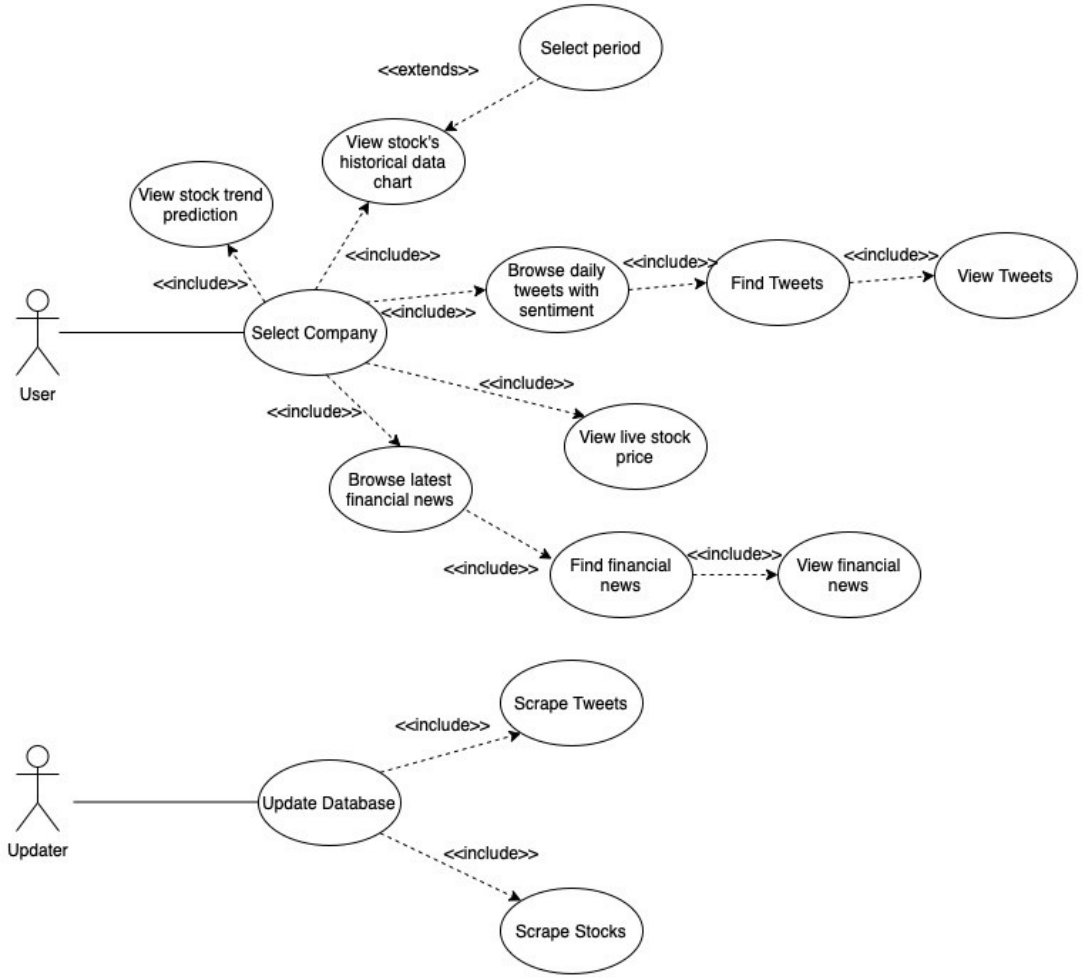
As regards to the stock trend prediction model, we generate the training dataset with the following procedure:

- We scraped stocks historical data between 2021-01-01 and 2022-01-14 for each of the selected stocks.
- We computed the correspondent average polarity of tweets related to each stock, for each day of which we have downloaded the data. The polarity of the tweets is computed with the Tweet Sentiment Classifier.
- We merged the data obtained in the previous steps joining on the 'Date' field.

The target class is set in the following way: if the previous day stock close price is higher than the one of the current day, we set the target class to 0 (Down), otherwise the target class is set to 1 (Up).
The final training dataset for the stock trend predictor contains 1056 rows, 264 per stock.

Analysis

The figure shows a basic Use Case Diagram.



Functional Requirements:

User:

- User can **select** a stock.
- User can **view** the historical close data chart of the selected stock.
- User can **view** the historical daily sentiment of the selected stock.
- User can **view** the current live price of the selected stock.
- User can **browse** daily tweets, with the sentiment, related to the selected stock.
- User can **browse** the latest financial articles related to the selected stock.
- User can **view** a prediction of the stock next close price.

Updater:

- Updater can **add** new tweets to the database.
- Updater can **add** new stock data to the database.

Non-Functional Requirements:

- The application needs to be user-friendly.
- The application needs to provide fast response to the user requests.
- The application should have an acceptable level of accuracy in the stock trend prediction and in the classification of the tweets' sentiment.

Workflow

Data Collection

Tweet Sentiment Classifier

The raw tweets were scraped with the Python library *snscrape*. Using the functions offered by this library, we were able to scrape all the tweets that contains certain keywords, published in a specific period. For each stock, we used as keywords the company name and the ticker name. The tweets downloaded have the following structure:


```
{
    "Account_Name":"Sayo473",
    "Number_Follower":30,
    "Text":"The past year has been tough for Apple (NASDAQ:AAPL) ...
    "Datetime":1642462696000
 }
```


We decided not to include fields such as the number of likes, retweets, or comments because we are downloading the tweets in real time, so all the previous mentioned fields would have small or zero values.

Stock Trend Predictor

Regarding the stock trend predictor, we scraped financial historical data with Yahoo Finance Python API. The data downloaded have the following structure:


```
{
    "Date":"2017-01-03T00:00:00.000Z",
    "Open":28.95,
    "High":29.08,
    "Low":28.69,
    "Close":29.04,
    "Volume":115127456,
    "Currency":"USD",
    "Stock":"AAPL"
}
```


Data Pre-processing

To obtain a good sentiment analysis on the tweets we must apply some pre-processing routine, because frequently in the text of the tweets are present noise words which could decrease the level of accuracy of our classification model. Moreover, we decide to remove tweets which we considered irrelevant for our purposes, such as not popular tweets for example. After numerous analysis we have noticed that the number of

tweets removed during the pre-processing phase significantly improved the performance of the classification model.
We do the following steps:

1. Put the text of the tweets in lower case.
2. Remove not English tweets and the tweets that do not contains keywords (ex. Remove all tweets published by accounts that contains the keyword in the username).
3. Removing hashtags, mentions, link, images, and emoji.
4. Remove all the punctuation.
5. Remove the extra-spaces.

For example:
Raw tweet:
$AMZN NEW ARTICLE : Amazon accused of anti-union tactics in New York https://t.co/XJSo4tOk7f Get all the latest $AMZN related news here : https://t.co/LlLESJ6Mlc https://t.co/CQ5swq3h72
Processed tweet:
amzn new article amazon accused of antiunion tactics in new york get all the latest amzn related news here

In order to apply classification algorithms to the tweets' text we must represent them as numerical vectors. To do so we follow the following steps:

- Extract from the text a matrix of token counts by using the *CountVectorizer* function provide by the Sklearn library.
- Transform a count matrix to a normalized tf or tf-idf representation. The goal of using *tf-idf* instead of the raw frequencies of the occurrences of a token in each document is to scale down the impact of tokens that occur very frequently in each corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus.

To compute the daily sentiment of the tweets related to a stock, we have given them a different weight, based on the number of followers of the user who has written it. We use an empirical formula which gives a higher importance to popular tweets, because

the number of followers of a user indicates how many persons can he reach with his opinion and how fast it will propagate. The formula which we use is the following:

$$tweet\_weight = \frac{\#\ follower \cdot 1000}{200000}$$

To compute the *polarity score* of the day we consider only tweets with positive or negative polarity, they will respectively increase or decrease the whole amount and then we compute the mean, which will be used by the prediction model to determine the stock's price trend.

Training the Models

Tweet Sentiment Classifier

For the tweet sentiment classifier, we decided to test 7 different classifiers:

- RandomForestClassifier
- XGBClassifier
- AdaBoostClassifier
- KNeighborsClassifier
- LogisticRegression
- MultinomialNB
- BernoulliNB

As specified in the dataset section, to have a more complete financial vocabulary, we used the Kaggle dataset to train our sentiment classifier model.
The performances were evaluated with a 10-fold cross validation with a train test split of 0.8 and 0.2.

Algorithm	Class	Precision	Recall	F Score	Accuracy
RandomForestClassifier	Negative	0.96	0.82	0.88	0.96
	Neutral	0.96	0.98	0.97	
	Positive	0.96	0.95	0.96	
XGBClassifier	Negative	0.92	0.77	0.84	0.94
	Neutral	0.93	0.98	0.96	
	Positive	0.97	0.92	0.94	
AdaBoostClassifier	Negative	0.90	0.61	0.73	0.89
	Neutral	0.86	0.98	0.92	
	Positive	0.95	0.79	0.86	
KNeighborsClassifier	Negative	0.71	0.30	0.42	0.73
	Neutral	0.71	0.95	0.81	
	Positive	0.85	0.41	0.55	
LogisticRegression	Negative	0.92	0.64	0.76	0.92

	Neutral	0.90	0.98	0.94	
	Positive	0.95	0.88	0.91	
MultinomialNB	Negative	0.96	0.20	0.33	0.80
	Neutral	0.78	0.96	0.86	
	Positive	0.85	0.66	0.74	
BernoulliNB	Negative	0.73	0.40	0.52	0.83
	Neutral	0.86	0.89	0.88	
	Positive	0.80	0.84	0.82	

As visible in the table, the results were very high. When using a Kaggle dataset, we are using a dataset that is tuned to realize classification models so if we want to have more realistic results, we should try these models with reals tweets.

To achieve this, we tested the algorithms also with manually labelled real tweets that were scraped and pre-processed with the modules of our application. The dataset used in this section of the evaluation has 566 neutral labelled tuples, 143 positive labelled tuples and 50 negative labelled tuples. The classifiers results were the following:

Algorithm	Class	Precision	Recall	F Score	Accuracy
RandomForestClassifier	Negative	0.76	0.60	0.68	0.77
	Neutral	0.84	0.83	0.83	
	Positive	0.69	0.82	0.75	
XGBClassifier	Negative	0.81	0.88	0.84	0.85
	Neutral	0.84	0.89	0.86	
	Positive	0.91	0.76	0.83	
AdaBoostClassifier	Negative	1.00	0.51	0.68	0.76
	Neutral	0.72	0.91	0.81	
	Positive	0.72	0.75	0.73	

KNeighborsClassifier	Negative	0.00	0.00	0.00	0.43
	Neutral	0.43	1.00	0.60	
	Positive	0.00	0.00	0.00	
LogisticRegression	Negative	0.86	0.44	0.58	0.65
	Neutral	0.62	0.91	0.74	
	Positive	0.62	0.47	0.53	
MultinomialNB	Negative	0.00	0.00	0.00	0.48
	Neutral	0.54	0.57	0.56	
	Positive	0.44	0.76	0.56	
BernoulliNB	Negative	0.71	0.63	0.67	0.60
	Neutral	0.76	0.40	0.52	
	Positive	0.49	0.86	0.63	

The model that has the best performance is XGBClassifier, a gradient boosted decision trees algorithm, and it was chosen for the application.

Stock Trend Predictor

For the stock trend predictor, we decided to test 6 different classifiers:

- RandomForestClassifier
- XGBClassifier
- AdaBoostClassifier
- KNeighborsClassifier
- LogisticRegression
- GaussianNB

We tried a lot of different combinations of predictors to perform the stock trend classification. We are dealing with a time series, so we focused on traditional methods used to improve time series prediction: we tried including an **exogenous factor,** and different **exponential moving average,** a type of moving average that places a greater weight and significance on the most recent data points.

Regarding the **exogenous factor,** our choice was the *S&P500* index, a financial index that keeps track of the performance of 500 large companies listed on stock exchanges in the United States.
It was chosen because all the companies selected for the application belong to the first 500 most important American companies. We thought that by including this factor we can have, as an input, a value that summarize how similar companies are performing.
However, we were not able to achieve better results with this technique, so it was not included in the final training dataset. It was proven that exogenous factors do not improve the prediction of a model.

Regarding **exponential moving average,** or **EMA,** we exploited the *ewm()* function that Pandas library provides that computes it in the following way:

$y_0 = x_0$

$y_t = (1 - \alpha)y_{t-1} + \alpha x_t$

$\alpha = 2 / (span + 1)$

We initially included 3 different EMAs: we computed it for span values of 3, 5, and 10 days with the values of the **difference between open price and close price** of a stock. We thought that an EMA calculated on these fields will give us a valuable summary of the current trend of a stock. Adding these factors allowed us to achieve better results so they were included in the final training dataset.
Then we also included other 3 EMAs, this time computed on the daily polarity value.
In the following figure is shown the code used to compute the different exponential moving averages.

```
# Compute Exponential Mobile Average (EMA) for stock price daily increments
delta = df['Prev Close'] - df['Prev Open']
df['10 Days Incr EMA'] = np.round(delta.copy().ewm(span=10, adjust=False).mean(), decimals=3)
df['5 Days Incr EMA'] = np.round(delta.copy().ewm(span=5, adjust=False).mean(), decimals=3)
df['3 Days Incr EMA'] = np.round(delta.copy().ewm(span=3, adjust=False).mean(), decimals=3)

# Compute Exponential Mobile Average (EMA) for stock polarity
df['10 Days Pol EMA'] = np.round(df['Polarity'].copy().ewm(span=10, adjust=False).mean(),
decimals=3)
df['5 Days Pol EMA'] = np.round(df['Polarity'].copy().ewm(span=5, adjust=False).mean(),
decimals=3)
df['3 Days Pol EMA'] = np.round(df['Polarity'].copy().ewm(span=3, adjust=False).mean(),
decimals=3)
```

The final training data set has the following schema:

Date	Prev Close	Prev Volume	Polarity	10 Days Incr EMA	5 Days Incr EMA	3 Days Incr EMA	10 Days Pol EMA	5 Days Pol EMA	3 Days Pol EMA
2021-01-04	3256.929932	2957200.0	39.350200	-18.070	-18.070	-18.070	64.340	59.712	54.622
2021-01-05	3186.629883	4411400.0	49.959878	-29.943	-39.837	-50.720	61.725	56.461	52.291
2021-01-06	3218.510010	2655500.0	22.887350	-14.953	-9.058	0.890	54.664	45.270	37.589
2021-01-07	3138.379883	4394800.0	15.317456	-13.707	-8.739	-3.605	47.510	35.286	26.453
2021-01-08	3162.159912	3514500.0	21.230500	-10.277	-4.106	0.777	42.732	30.601	23.842
...	...	...	...	...	...	...	...	...	...
2022-01-10	3251.080078	2329300.0	17.298125	-16.919	-21.893	-22.987	20.988	18.621	17.651
2022-01-11	3229.719971	4389900.0	31.623548	-10.568	-8.592	-2.488	22.922	22.955	24.637
2022-01-12	3307.239990	3140300.0	7.483831	5.397	20.019	37.376	20.115	17.798	16.061
2022-01-13	3304.139893	2501500.0	12.161957	-0.559	4.226	5.008	18.669	15.919	14.111
2022-01-14	3224.280029	2609400.0	26.118981	-15.135	-24.093	-37.861	20.023	19.319	20.115

We trained the stock predictor model using 1 years of historical data of each selected stock along with the correspondent daily sentiment. We cannot perform a ‘standard’ cross validation due the fact that we are using a time series, so data cannot be random sampled because we cannot use future information to predict the past.

We trained the data with the first 70% of the data and we used the last 30% for the test set, then with an 80% - 20% split and finally with a 90% - 10% split. In the following table are shown the results of the tests, using different combination of attribute measures.

Algorithm	Attribute Selection	Class	Precision	Recall	F Score	Accuracy
RandomForestClassifier	StandardScaler	Down	0.52	0.39	0.44	0.54
		Up	0.55	0.67	0.60	
XGBClassifier	StandardScaler	Down	0.54	0.36	0.43	0.55
	PCA	Up	0.55	0.71	0.62	
AdaBoostClassifier	StandardScaler	Down	0.57	0.42	0.49	0.57
	PCA	Up	0.57	0.71	0.64	
KNeighborsClassifier	StandardScaler	Down	0.48	0.40	0.44	0.50
	PCA	Up	0.52	0.59	0.55	
LogisticRegression	StandardScaler	Down	0.33	0.01	0.01	0.52
		Up	0.52	0.99	0.68	
GaussianNB	StandardScaler	Down	0.60	0.05	0.10	0.53
	SelectKBest(5)	Up	0.53	0.97	0.68	

All the models are not accurate, however AdaBoostClassifier was chosen for the application.

Possible Improvements

To conclude, the nature of the problem makes it almost impossible to obtain a highly precise classifier model but there are still some considerations that can be done regarding possible improvements.
We are computing the daily polarity with our sentiment classifier, so if it misclassifies some tuples the error will be propagated to the stock trend predictor. So, it is probable that the training dataset of the stock trend predictor, in particular the daily polarity field, is not highly reliable. We might obtain a more accurate sentiment classifier using deep learning techniques like neural network.

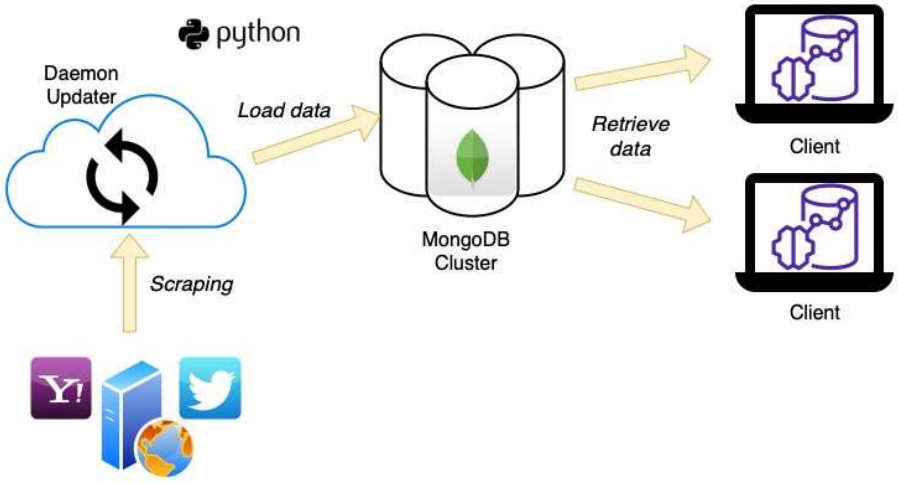
The lexicon used by Twitter's users can change over time and it probably will decrease the accuracy of the Tweet Sentient Classifier: a possible way to improve our models is by triggering a re-train whenever a concept drift is detected.

Implementation

The whole web-application is implemented in *Python* exploiting the *SK-learn* library for the Machine Learning purposes. The training, the cross-validation analysis of our models is implemented using the *Jupiter* notebook. The application is developed using *PyCharm*.

The client-side of the application is developed using the *Dash* library, which permits to develop modern reactive dashboard for the data analysis application.

We decide to use *MongoDB Atlas,* a document database which gives the possibility to use a cluster of three servers in the cloud to store our data. We decide to use a database to store the information which we scrape from the web to speed up the time to retrieve data for our analysis, in this way we can delegate the scraping functionality to the daemon process.



Regarding the **Jupyter notebook** we have two modules:

- *sentiment_analysis.ipynb*
  In which there is the code to train and save the model to classify the tweets polarity.
- *training_stock_predictor.ipynb*
  In which there is the code to train and save the model to classify the stocks trend.

The **application** consists of five different packages:

- *classification:*

In this package there are functions to apply the models to live data retrieved from the database

- *collecting:*
  In this package there are all the functions to scrape data from the web and to interact with the MongoDB.
- *common:*

In this package there are useful constants and utility functions.

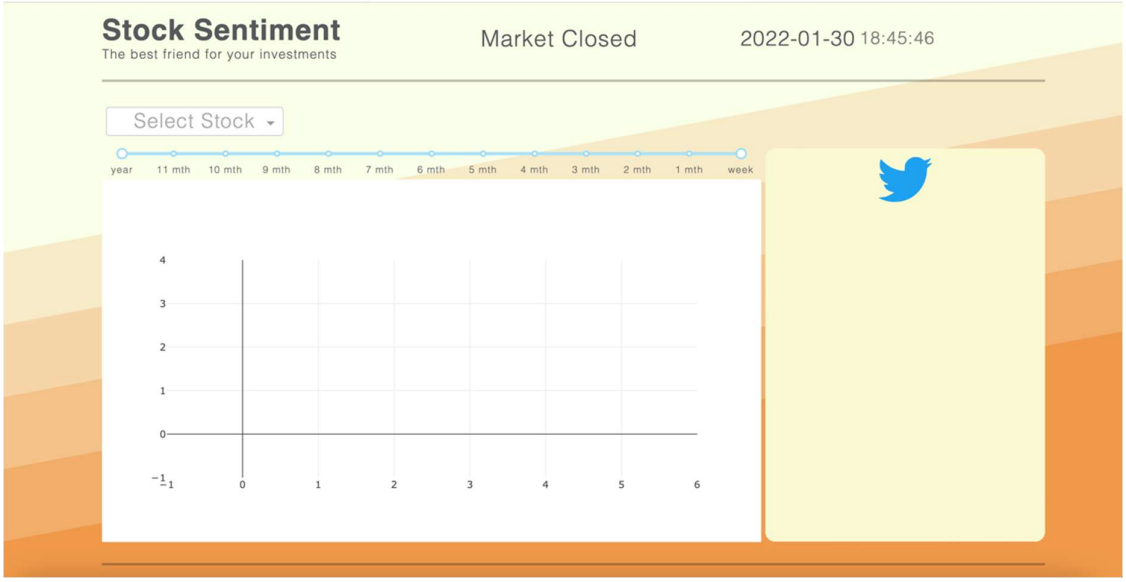
- *model:*

In this package there are a function which we use to add label by hand to the tweets, used to preparing the test set for the classification model.

- *preprocessing:*
  In this package there are functions to clean the tweets from the noise words and to add a weight the tweets based on the popularity of the user

Usage

Stock Sentiment is a user-friendly web-application, the first view of the application shows *dropdown* menu where the user can select one of the available companies.



In the header the user can visualize the status of the market and the current time (EST time-zone).

After having chosen one company the application will show to the user:

- Live market stock's price.
- A chart where is represented the historical trend of the selected stock.
- Live tweets with the respective polarity sorted by weight.
- The prediction of the stock trend.
- Recent financial news relative to the selected stock.

The user can analyse the period which he prefers by moving the slider on the top of the graph.