Machine Learning Engineer Nanodegree

Capstone Project

Jean Gagnaire

July 25th, 2020

I. Definition

Project Overview

The goal of this project is to explore what can be done with Machine Learning applied to stock prices prediction. Portfolio managers have been performing statistical analyses of financial markets since they exist, but Machine Learning brought a new perspective to the field, with the ability to couple high-performance computing to statistics. A stock owner or fund manager could be using ML for regression, to predict the future value of an asset, or for classification, to investigate the risk attached to a financial product.

Moreover, prices in financial markets are driven by offer and demand: public feeling about a company or a country can change the price of stocks or government bonds. This is a field where sentiment analysis can be applied, through Natural Language Processing of press articles for instance.

Problem Statement

I will explore several applications of Machine Learning to financial data:

- I will first work on a regression model able to predict future prices for a given stock. The result of this first part will be a Python script, performing regression on a stock's historical data, and providing the user with a prediction as accurate as possible for the next day or more.

- I will then select and train a classifier, to predict a stock price increase on the next day. The work resulting from this will be a Python script, taking the name of a stock as input. The script would then classify the stock as "price will increase tomorrow" or not.

Metrics

→ The quality of the regression models explored to predict the price of a stock will be measured with metrics that should be interpreted against the target value:

- **Mean Squared Error** (MSE):

$$\frac{\sum_{i=1}^{n} (y_i - \hat{y})^2}{n}$$

with y_i the true adjusted close price, $\hat{y}$ the prediction and n the number of samples.

- **Mean Absolute Error** (MAE):

$$\frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}|$$

... but also with others normalized metrics, that can be compared across different models:

- **Standard-Deviation Normalized Root Mean Squared Error** (SD-NRMSE):

$$NRMSE = \frac{RMSE}{\sigma}$$

σ the standard deviation of adjusted close values, and $RMSE$ calculated with:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{y})^2}{n}}$$

- If the model is able to explain all of the variation in the adjusted close price, i.e. if $\hat{y}$ equals y, NRMSE will be zero
- If the model is able to explain the variation partially, SD-NRMSE will be around 1
- A larger ratio would indicate that predictions produced by the model would be quite far from their truth labels.

- **Mean Absolute Percentage Error** (MAPE):

$$\frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i-\hat{y}}{y_i}\right|$$

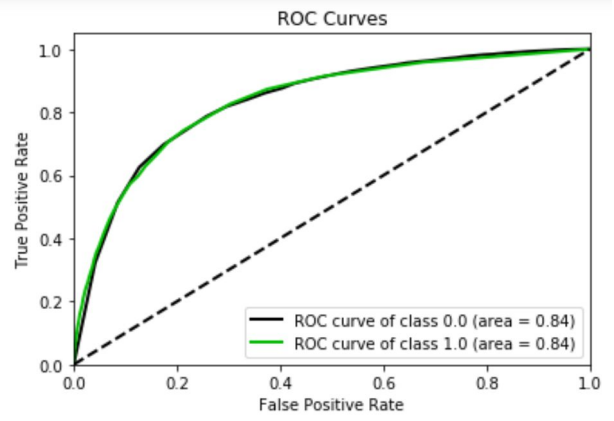
→ The classifiers developed to predict an increase in stock prices will need **high precision**, to avoid false positives as much as possible. A false positive in our use case could lead the user to lose money by incorrectly labeling a stock as “price will increase tomorrow” (class 1). Our classifiers will also need **good recall scores**, meaning they would be able to actually find the stock prices that will increase tomorrow, and not just label everything as “no increase tomorrow” (class 0). The right indicator for high precision and good recall is the **F-beta score with a beta of 0.5** so that precision is favored over recall. These three indicators have values between 0 and 1, and the higher precision, recall and F-beta score, the better the model.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$F_\beta = \frac{(1+\beta^2)\cdot(\text{precision}\cdot\text{recall})}{(\beta^2\cdot\text{precision}+\text{recall})}$$ with β = 0.5

The quality of the evaluated classifiers will also be measured by plotting their **Receiver Operating Characteristic (ROC) curves**, and by calculating the **Area Under the Curve (AUC)**.



AUC always has a value between 0 and 1, and a high AUC means that the model would be able to differentiate the two classes well.

II. Analysis

Data Exploration

We can query Yahoo! Finance API for historical market data for a specific stock. The time series for obtained in CSV format can then be loaded in a pandas DataFrame, using the dates as index instead of traditional integers:

```
df = pd.read_csv('historical_data_AI-PA.csv',
                 index_col=0,
                 parse_dates=True,
                 infer_datetime_format=True)
```

```
df.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
2000-01-03	34.854301	36.306599	34.771301	35.061798	10.226519	904282.0
2000-01-04	35.061798	34.999500	32.613701	33.505798	9.772677	1381445.0
2000-01-05	32.779701	33.402100	32.281700	33.194599	9.681908	853763.0
2000-01-06	32.758900	36.223598	32.696701	35.580399	10.377778	1387137.0
2000-01-07	35.580399	37.136398	34.958000	35.144798	10.250728	2198233.0

Rows are then ordered from older to more recent. The “Adj Close” column is the one we are interested in predicting: the value it contains corresponds to the Close price, i.e. the amount of money at which the last transaction of market day occurred, modified to take into account stock splits and dividends paid by the company.

General information about each columns can be displayed using the `describe()` function:

```
df.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	5267.000000	5267.000000	5267.000000	5267.000000	5267.000000	5.267000e+03
mean	65.154315	65.766419	64.550407	65.181575	50.084645	1.257315e+06
std	26.437988	26.582026	26.277434	26.441206	31.948313	7.275060e+05
min	27.530800	28.087299	26.804600	26.970600	7.866548	0.000000e+00
25%	38.716299	39.138500	38.380001	38.770450	18.792883	8.208425e+05
50%	60.484501	61.058498	59.993599	60.505001	42.880104	1.094817e+06
75%	86.474750	87.168801	85.711201	86.457001	76.446960	1.493460e+06
max	140.500000	140.699997	139.800003	140.300003	137.140625	1.014686e+07

This shows us two interesting elements that we should pay careful attention to:

- **our set does not contain a lot of data points**: indeed, there is one point per market day, and the oldest quote available on the Euronext market we work with on this project happened on March 1st 1990.

- **our dataset will require scaling** of its values before applying any ML algorithm: the Volume column has a very different value range than the rest of the dataset.

A common saying about finance datasets is that they are very clean and usually have very few missing values. We can investigate this by counting the NaNs in our dataset:

```
df.isna().sum().sum()
```

```
30
```

30 missing values is not a lot for a total of 5267 data points having 6 columns each.

We can then add technical analysis features to our feature set, using the `ta` Python package:

```
df = ta.add_all_ta_features(df, open='Open', high='High', low='Low', close='Close', volume='Volume')
```

Once this line returns, our dataset will contain the features shown in the screenshot below. These features are traditional statistical indicators used to analyze current trends in all kinds of contexts.

```
df.columns
```

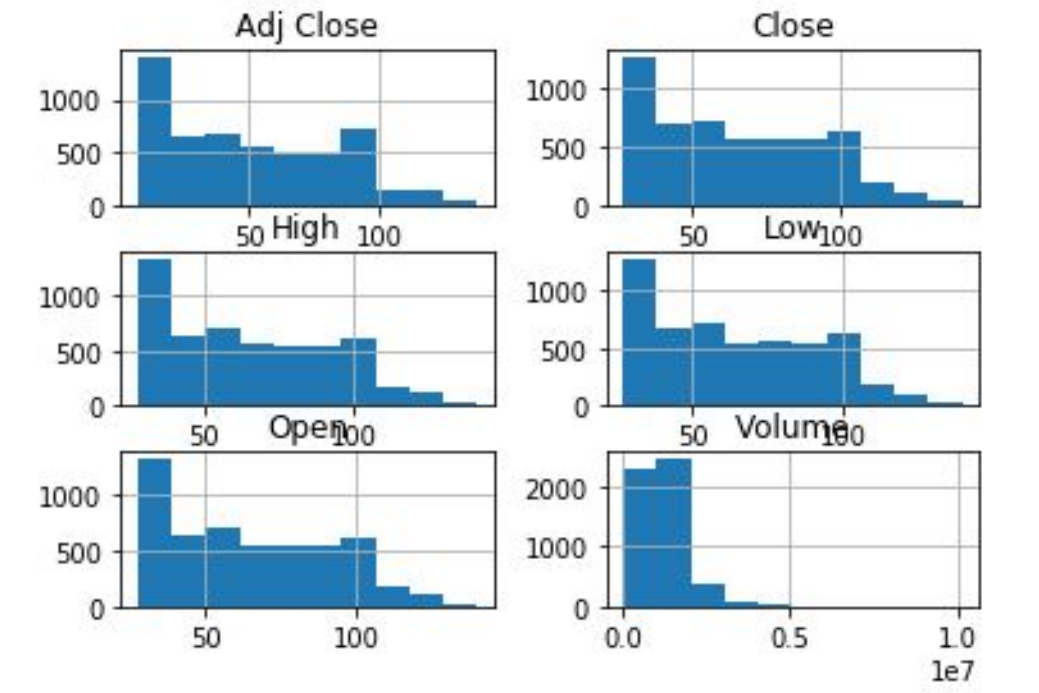
```
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'volume_adi',
       'volume_obv', 'volume_cmf', 'volume_fi', 'momentum_mfi', 'volume_em',
       'volume_sma_em', 'volume_vpt', 'volume_nvi', 'volume_vwap',
       'volatility_atr', 'volatility_bbm', 'volatility_bbh', 'volatility_bbl',
       'volatility_bbw', 'volatility_bbp', 'volatility_bbhi',
       'volatility_bbli', 'volatility_kcc', 'volatility_kch', 'volatility_kcl',
       'volatility_kcw', 'volatility_kcp', 'volatility_kchi',
       'volatility_kcli', 'volatility_dcl', 'volatility_dch', 'trend_macd',
       'trend_macd_signal', 'trend_macd_diff', 'trend_sma_fast',
       'trend_sma_slow', 'trend_ema_fast', 'trend_ema_slow', 'trend_adx',
       'trend_adx_pos', 'trend_adx_neg', 'trend_vortex_ind_pos',
       'trend_vortex_ind_neg', 'trend_vortex_ind_diff', 'trend_trix',
       'trend_mass_index', 'trend_cci', 'trend_dpo', 'trend_kst',
       'trend_kst_sig', 'trend_kst_diff', 'trend_ichimoku_conv',
       'trend_ichimoku_base', 'trend_ichimoku_a', 'trend_ichimoku_b',
       'trend_visual_ichimoku_a', 'trend_visual_ichimoku_b', 'trend_aroon_up',
       'trend_aroon_down', 'trend_aroon_ind', 'trend_psar_up',
       'trend_psar_down', 'trend_psar_up_indicator',
       'trend_psar_down_indicator', 'momentum_rsi', 'momentum_tsi',
       'momentum_uo', 'momentum_stoch', 'momentum_stoch_signal', 'momentum_wr',
       'momentum_ao', 'momentum_kama', 'momentum_roc', 'others_dr',
       'others_dlr', 'others_cr'],
      dtype='object')
```

Open, Close, Low, High, Volume for the CAC40 and SBF120 will also be added to our feature set: the stock we will work with relate to these two major market indices.

Exploratory Visualization

The distributions of each of our initial features can be plotted in histograms. This shows us that Adj Close, Close, High, Open, and Low unsurprisingly have the same value ranges and non-normal distributions. We can also note from the figures below that the Volume feature has a very different value range, and that its distribution is highly skewed to the left.

```
ret = df.hist()
```



Algorithms and Techniques

→ I will apply the following regression algorithms to predict the future price of a stock:

- LinearRegression
- RandomForestRegressor
- KNeighborsRegressor
- Support Vector Regression (SVR)
- KernelRidge regression
- SageMaker's Recurrent Neural Network algorithm DeepAR

The six algorithms will be provided with the same training and testing sets, consisting in time series containing features and target values:

- features are the technical analysis indicators and metrics for the day, as shown above
- target values are constructed from the adjusted close values of the next 7 days:

```
adjclose_df.head()
```

Date	AdjClose_D+1	AdjClose_D+2	AdjClose_D+3	AdjClose_D+4	AdjClose_D+5	AdjClose_D+6	AdjClose_D+7
2000-01-03	9.772677	9.681908	10.377778	10.250728	9.893692	9.802925	9.916677
2000-01-04	9.681908	10.377778	10.250728	9.893692	9.802925	9.916677	10.147853
2000-01-05	10.377778	10.250728	9.893692	9.802925	9.916677	10.147853	10.371740
2000-01-06	10.250728	9.893692	9.802925	9.916677	10.147853	10.371740	10.026813
2000-01-07	9.893692	9.802925	9.916677	10.147853	10.371740	10.026813	9.923937

→ The classifiers that we will work with to predict an increase in stock price include:

- LinearSVC
- SGDClassifier
- LogisticRegression Classifier
- Support Vector Machines (SVC)
- KNeighborsClassifier
- RandomForestClassifier
- AdaBoostClassifier with DecisionTreeClassifier as weak learner

They will be provided with the same feature set used for the regression task, but a different target dataset: 1 if an increase of 5% occurred on the next day, 0 otherwise. As this dataset contains a lot more samples from class 0 than class 1, the classifiers will most probably require the `class_weight` parameter to be set to `balanced`.

Date	increase_tomorrow
2020-07-07	0
2020-07-08	0
2020-07-09	0
2020-07-10	0
2020-07-13	0

Benchmark

→ Scikit-Learn's benchmark model `DummyRegressor` will be first executed. This model always predicts the mean of the training set, and will act as a simple baseline to be compared with the other real models.

Below are the results obtained by this benchmark model:

	MSE	SD-RMSE	MAE	MAPE
d+1	0.242	5.106	0.482	62.904
d+2	0.242	5.101	0.482	62.907
d+3	0.242	5.095	0.483	62.909
d+4	0.243	5.087	0.483	62.912
d+5	0.243	5.08	0.483	62.916
d+6	0.244	5.067	0.484	62.919
d+7	0.244	5.057	0.484	62.923

As we can see, the two normalized error measures SD-RMSE and MAPE are both very high.

→ Regarding the classification task, `DummyClassifier` will be used to obtain comparison metrics:

```
Results:
        accuracy: 84.34%
        precision: 5.56%
        fbeta: 0.057
Wall time: 15.6 ms
```

While the accuracy obtained can be surprisingly high, the precision and F-beta scores are both very low.

III. Methodology

Data Preprocessing

For both regression and classification tasks, NaNs and infinity values will be filled by using interpolation, so that the algorithms can be applied without errors.
Data will be scaled between 0 and 1 using MinMaxScaler before being processed.

There will be an extra data preprocessing step for the DeepAR regression model, as it requires its input data to be pre-transformed into a specific JSON format.

An important point of the preprocessing and split of the training and testing set should be noted:

- The time series characteristics of the dataset should be preserved to perform regression correctly, so the regression testing set will be made of more recent days than the regression training set.

- For classification, the time order is no longer important since we are trying to identify correlations between values of features and an event. This means that our full dataset can be shuffled safely, and then split into training and testing sets.

Implementation

→ For the regressors, the SD-RMSE and MAPE metrics are not included in Sklearn, but can be easily implemented with one line of code each:

```
def stdev_root_mean_squared_error(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred)) / np.std(y_true)
```

```
def mean_asbsolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

7 versions of each of the 6 models listed above will be trained to predict stock price for D+1 to 7, and MSE, SD-RMSE, MAE and MAPE will all be computed for each prediction.

→ For the classifiers, accuracy, precision, recall and F-beta scores are all implemented and ready-to-use from Sklearn.

Refinement

→ After applying several regression models and performing backtesting on past market data to evaluate them, I have found that simple models like LinearRegression and KernelRidge perform much better than more complex models like Support Vector Machines regressor, that failed to obtain good metrics.

Something quite trivial but still interesting to observe is that predictions are always better for D+1 than for D+7.
Also, training the regressors on recent data points resulted in producing less error than training them on the full historical dataset: indeed, quite good predictions can be obtained for D+1 by averaging the predictions of 3 KernelRidge models trained on the last 10, 20 and 30 days of the stock being analyzed.

→ Regarding the classification task, I first worked on a dataset made of historical data for a single stock, but soon realized that the algorithms would either:

- easily overfit because the training set was too small (maximum 7 000 points)

- or failed to classify increase on the next day correctly, because the training set had too few samples of this class, despite adjusting their `class_weight` parameter to `balanced`.

Seeing this, I tried to apply the same classification algorithms to a much larger dataset, made of historical data from 96 different stocks, for a total of more than 500 000 data points.

IV. Results

Model Evaluation and Validation

→ For the regression task, I have selected and trained LinearRegression and KernelRidge models, and then made predictions for stocks of 5 very different companies: 3 blue chips, and 2 highly speculative stocks. For each time horizon of prediction, D+1, D+3, and D+4, I have trained 3 LinearRegression and 3 KernelRidge models on the last 10, 20 and 30 days of historical market data.

The best prediction was always from KernelRidge, and was the mean of the 3 predictions from models trained on 10, 20 and 30 days data.

The results appear in the table on the next page, and percentage of errors are calculated below for D+1, D+3, and D+7:

	D+1	D+3	D+7
AI	0.10%	3.30%	6.43%
DBV	4.60%	7.80%	25.08%
ETL	0.27%	0.50%	1.71%
GNFT	4.80%	1.70%	11.46%
SAF	0.01%	1.70%	1.12%

Predictions for all 5 stocks are always below 5% error for D+1, which is quite good. Error is then amplified as the prediction horizon increases.

STOCK: AI.PA	D+1			D+3			D+7		
historical training days	LinearReg forecast	KernelRidge forecast	actual	LinearReg forecast	KernelRidge forecast	actual	LinearReg forecast	KernelRidge forecast	actual
10	128.5866	132.0505		125.7461	127.5578		128.2738	127.9127	
20	125.6467	130.7608		120.7969	126.9991		127.2525	126.9917	
30	120.9018	132.5771	131.6	119.8404	128.7965	132.2	130.6812	126.7069	135.95
mean preds (10,20,30)	125.0450333	131.7961333		122.1278	127.7844667		128.7358333	127.2037667	
full historical data	188.9171	179.142		189.3879	179.4179		193.9827	179.7235	

STOCK: DBV.PA	D+1			D+3			D+7		
historical training days	LinearReg forecast	KernelRidge forecast	actual	LinearReg forecast	KernelRidge forecast	actual	LinearReg forecast	KernelRidge forecast	actual
10	8.6738	8.4919		9.1227	8.3776		10.1039	9.8249	
20	8.9965	8.5126		10.173	8.4563		11.2355	9.0943	
30	9.5222	8.4171	8.1	9.9712	8.722	7.9	13.4223	9.1866	7.49
mean preds (10,20,30)	9.064166667	8.473866667		9.755633333	8.518633333		11.58723333	9.3686	
full historical data	2.4545	5.9898		-6.819	5.5175		12.0053	6.3353	

STOCK: ETL.PA	D+1			D+3			D+7		
historical training days	LinearReg forecast	KernelRidge forecast	actual	LinearReg forecast	KernelRidge forecast	actual	LinearReg forecast	KernelRidge forecast	actual
10	8.2819	8.1451		8.3479	8.1506		8.993	8.7318	
20	8.5647	8.2836		7.8866	7.9956		8.7057	8.6499	
30	9.6481	8.3253	8.274	8.149	8.3917	8.224	6.692	8.31	8.42
mean preds (10,20,30)	8.831566667	8.251333333		8.127833333	8.1793		8.130233333	8.5639	
full historical data	8.0674	7.4575		8.1108	7.3454		8.6177	7.0984	

STOCK: GNFT.PA	D+1			D+3			D+7		
historical training days	LinearReg forecast	KernelRidge forecast	actual	LinearReg forecast	KernelRidge forecast	actual	LinearReg forecast	KernelRidge forecast	actual
10	5.2438	4.9687		4.9669	4.8633		5.5278	5.3444	
20	5.7917	5.1559		5.1604	4.7425		5.2553	4.9039	
30	6.2946	5.0327	4.82	6.0208	4.9011	4.922	5.0028	4.8388	4.512
mean preds (10,20,30)	5.7767	5.052433333		5.3827	4.835633333		5.261966667	5.029033333	
full historical data	-564601300652	4.8142		1.1296	3.9055		-0.1231	2.7121	

STOCK: SAF.PA	D+1			D+3			D+7		
historical training days	LinearReg forecast	KernelRidge forecast	actual	LinearReg forecast	KernelRidge forecast	actual	LinearReg forecast	KernelRidge forecast	actual
10	90.5077	91.1293		92.1622	92.8254		89.3585	91.4452	
20	95.272	94.3826		93.031	89.7753		95.0337	91.6235	
30	82.5447	92.9854	92.84	76.3849	82.9737	90.04	96.885	89.1692	89.74
mean preds (10,20,30)	89.44146667	92.83243333		87.1927	88.5248		93.75906667	90.74596667	
full historical data	136.4871	132.9481		143.6036	133.6821		134.0855	132.4114	

→ The best two classifiers were selected from metrics obtained on the testing set:

```
%%time
rf_model = RandomForestClassifier(class_weight='balanced')
train_eval(rf_model, train_X_scaled, train_y, test_X_scaled, test_y)
```

```
Results:
        accuracy: 97.42%
        precision: 97.37%
        fbeta: 0.657
Wall time: 7min 2s
```

```
%%time
kn_model = KNeighborsClassifier(n_neighbors=8, n_jobs=-2)
train_eval(kn_model, train_X_scaled, train_y, test_X_scaled, test_y)
```

```
Results:
        accuracy: 97.05%
        precision: 93.10%
        fbeta: 0.521
Wall time: 8min 50s
```

Despite very high precision, their F-beta scores were negatively impacted by a lower recall score.

Both models have then been evaluated on stocks that were not part of the training set, and the recall obtained is very close to zero as shown below, meaning that they would need to be retrained on new stocks to be able to make good predictions:

```
Results for RF model:
        accuracy: 98.39%
        precision: 100.00%
        recall: 0.84%
        fbeta: 0.041
```

```
Results for KNN model:
        accuracy: 98.38%
        precision: 50.00%
        recall: 0.84%
        fbeta: 0.039
```

I also evaluated these two classifiers on present day post-close market data, and made predictions for the next day. Both models classified the 96 stocks as “no increase tomorrow”, and 2 stocks got a 5% or more price increase on the next day:

- ALPHA.PA is quite famous for high volatility because of low daily volumes: this certainly makes predicting harder from trends indicators only

- DIM.PA is a Biotech company that recently saw an increase of production because of the Covid-19 outbreak. On the next day, the company published the financial forecasts for their 2020 turnover taking into account the pandemia. This resulted in a 10% stock price increase.

The increase of DIM.PA is typically something that a sentiment analysis model could detect by classifying incoming news articles.

Justification

→ Regarding the regression problem, `DummyRegressor` had an MAPE of more than 62%, versus less than 5% for KernelRidge D+1 predictions averaged from models trained on 10, 20 and 30 days of historical data.

→ The benchmark model DummyClassifier obtained an accuracy of 84.34%, a precision of 5.56% and an F-beta score equal to 0.057, whereas the two KNN and RF classifiers used for final predictions had accuracies of 97.42% and 97.05%, precisions equal to 93.10% and 97.37%, and F-beta scores of 0.521 and 0.657.

These results are much stronger than the two benchmark models. However, predicting the price of a stock is a complex and wide problem, and a solution not taking into account sentiment analysis of news articles could not be considered complete and reliable. Thus, the regression models and classifiers developed in this project can only bring food for thought to a portfolio manager interested in a stock.

V. Conclusion

Reflection

In this project, I have used two different Machine Learning approaches to try to make accurate predictions of the price of a stock: I have first trained and analyzed several regression models, and found a hybrid model capable of predicting a stock price for the next day with less than 5% error. Then, I have considered this problem from the classification angle: I compared 6 algorithms applied to two different datasets, a small one containing data for a single stock, and a larger one, with data from almost a hundred stocks. The best two classifiers obtained good evaluation metrics.

Despite these encouraging results, this project showed me that applying Machine Learning to market data does not produce any magic: indeed, as price movements are caused by investor's decisions to buy and sell a stock, the analyses conducted here revealed insufficient to really solve the initial problem: technical analysis and trend indicators will always include information that is already released publicly and reflected in the current price of a stock.

Improvement

What changes the market valuation of a company is either new information not yet priced, or pure speculation: in both cases, an automatic trading system using Machine Learning models for price prediction would certainly need to incorporate new information to its trading strategy additionally. This can be developed using Natural Language Processing, applied to incoming news articles.