Supervised Learning

Beat US Stock Market (2019 edition)

Turma 3 - Grupo 22

Catarina Fernandes - up201806610@fe.up.pt

Diogo Almeida - up201806630@fe.up.pt

Pedro Queirós - up201806329@fe.up.pt

Specification of the work to be performed

Context

The algorithmic trading space is buzzing with new strategies. Companies have spent billions in infrastructures and R&D to be able to jump ahead of the competition and beat the market. Still, it is well acknowledged that the buy & hold strategy is able to outperform many of the algorithmic strategies, especially in the long-run. However, finding value in stocks is an art that very few mastered, can a computer do that?

The Problem Specification

The objective of a supervised learning model is to predict the correct label for newly presented input data.

Our goal is to correctly label stock as buy-worthy or not. In order to accomplish it, we'll use the entries from the Beat US Stock market (2019 edition) data set, which contains the 10-K filings of 638 Tech Companies, to train and test our predictive model.

As we intend to approximate a mapping function from input variables to discrete output variables we can classify this problem as a **classification problem**.

Related work with references to works found in a bibliographic search

- Exercises from the practical classes: https://moodle.up.pt/mod/resource/view.php?id=154306
- Beat US Stock market (2019 edition): https://www.kaggle.com/cnic92/beat-us-stock-market-data
- Data Preprocessing: https://moodle.up.pt/pluginfile.php/211571/mod_resource/content/0/IART_Lecture5c_MachineLearning_DataPreprocessing.pdf
- Machine learning tools: https://moodle.up.pt/pluginfile.php/211545/mod_resource/content/0/IART_Lecture5b_MachineLearning_Tools.pdf
- Machine learning classification: https://moodle.up.pt/pluginfile.php/213487/mod_resource/content/0/IART_Lecture5d_MachineLearning_Classification.pdf
- Decision Trees - https://scikit-learn.org/stable/modules/tree.html
- K-Nearest Neighbor (K-NN): https://www.techopedia.com/definition/32066/k-nearest-neighbor-k-nn
- Support Vector Machine: https://www.techopedia.com/definition/30364/support-vector-machine-svm
- Multi-layer Perceptron: https://www.techopedia.com/definition/20879/multilayer-perceptron-mlp

Description of the tools and algorithms to use in the assignment

Development Environment:

- Jupyter Notebook
- Spyder (For algorithm testing)

Libraries/Tools:

- Pandas (Data Extraction);
- NumPy and SciPy (Data Manipulation);
- Seaborn and MatPlotLib (Data Visualization);
- Scikit-Learn (Learning Models).
- Imbalanced-learn (Oversampling and Undersampling)

Data Analysis & Pre-Processing

When we first looked at the data, it looked pretty decent, it had no empty cells and no wrong data. All we had to do was remove the stocks' names column, which was useless.

We then studied the influence of each feature in the price variation of the stocks of the companies and removed the columns that have a correlation greater than 95% with other columns (48). This way we only have the features that provides us with relevant information, so the models can be processed faster.

Furthermore, we also noticed that the dataset has more stocks that increased in value than stocks decreased in value. With that said, our models will naturally be better prepared for stocks that have gone up in price.

	Ticker	Revenue	Revenue Growth	Cost of Revenue	Gross Profit	SG&A Expense	Operating Expenses	Operating Income	Earnings before Tax	Net Income	...	EPS Diluted Growth
0	INTC	7.084800e+10	0.1289	2.711100e+10	4.373700e+10	6.750000e+09	2.042100e+10	2.331600e+10	2.331700e+10	2.105300e+10	...	1.2513
1	MU	3.039100e+10	0.4955	1.250000e+10	1.789100e+10	8.130000e+08	2.897000e+09	1.499400e+10	1.430300e+10	1.413500e+10	...	1.6100
2	AAPL	2.655950e+11	0.1586	1.637560e+11	1.018390e+11	1.670500e+10	3.094100e+10	7.089800e+10	7.290300e+10	5.953100e+10	...	0.2932
3	MSFT	1.103600e+11	0.1428	3.835300e+10	7.200700e+10	2.222300e+10	3.694900e+10	3.505800e+10	3.647400e+10	1.657100e+10	...	-0.3446
4	HPQ	5.847200e+10	0.1233	4.780300e+10	1.066900e+10	4.859000e+09	6.605000e+09	4.064000e+09	3.013000e+09	5.327000e+09	...	1.2027
...	...	...	...	...	...	...	...	...	...	...	...	...
633	TRNS	1.551410e+08	0.0781	1.177000e+08	3.744100e+07	2.841500e+07	2.841500e+07	9.026000e+06	7.948000e+06	5.922000e+06	...	0.2656
634	TSRI	6.499000e+07	0.0386	5.460910e+07	1.038090e+07	9.471523e+06	9.471523e+06	9.093770e+05	8.672080e+05	4.862080e+05	...	0.7857
635	TZOO	1.113220e+08	0.0450	1.226800e+07	9.905400e+07	8.182300e+07	9.081600e+07	8.238000e+06	8.286000e+06	4.661000e+06	...	0.3704
636	WSTG	1.814440e+08	0.1300	1.545240e+08	2.692000e+07	2.031900e+07	2.276500e+07	4.155000e+06	5.117000e+06	3.538000e+06	...	-0.3097
637	WTT	5.278800e+07	0.1456	2.862100e+07	2.416700e+07	1.790100e+07	2.281000e+07	1.357000e+06	8.300000e+04	3.500000e+04	...	1.0000

```python
clean_data = ten_k_fillings_data.drop(columns=["Ticker"])

# Create correlation matrix
corr_matrix = clean_data.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# Find features with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(upper[column] > 0.95)]

# Drop features
clean_data.drop(to_drop, axis=1, inplace=True)

print("{} Dropped columns: {}".format(len(to_drop), to_drop) )
```

```
48 Dropped columns: ['Cost of Revenue', 'Gross Profit', 'Earnings before Tax', 'Net Income', 'Net Income Com', 'EPS Diluted',
'Weighted Average Shs Out (Dil)', 'Profit Margin', 'EBITDA', 'EBIT', 'Consolidated Income', 'Earnings Before Tax Margin', 'Net
Profit Margin', 'Total current assets', 'Total assets', 'Total current liabilities', 'Total liabilities', 'Other Liabilities',
'Operating Cash Flow', 'Capital Expenditure', 'Free Cash Flow', 'priceToSalesRatio', 'ebitperRevenue', 'grossProfitMargin', 'pr
etaxProfitMargin', 'netProfitMargin', 'eBITperRevenue', 'quickRatio', 'operatingCashFlowSalesRatio', 'Revenue per Share', 'Oper
ating Cash Flow per Share', 'Free Cash Flow per Share', 'Cash per Share', 'Shareholders Equity per Share', 'Price to Sales Rati
o', 'Current ratio', 'SG&A to Revenue', 'Capex to Revenue', 'Return on Tangible Assets', 'Invested Capital', 'Average Receivabl
es', 'Average Payables', 'Days Sales Outstanding', 'Days Payables Outstanding', 'ROE', 'Capex per Share', 'EPS Diluted Growth',
'Weighted Average Shares Diluted Growth']
```

Oversampling & Undersampling

To try to make the dataset balanced, we created 2 more sets from the original: one oversampled and the other undersampled.

To finalize the pre-processing stage, we also scaled/normalized the data since some of the algorithms used require it, like for example K-Nearest Neighbors (which needs to calculate distances).

```
os = SMOTE(random_state=1)
us = RandomUnderSampler(random_state=1)

os_inputs, os_labels = os.fit_resample(inputs_train, labels_train)
print(Counter(os_labels))

us_inputs, us_labels = us.fit_resample(inputs_train, labels_train)
print(Counter(us_labels))
```

```
Counter({1: 338, 0: 338})
Counter({0: 140, 1: 140})
```

```
scaler = StandardScaler()
scaler.fit(inputs_train)

inputs_train = scaler.fit_transform(inputs_train)
inputs_test = scaler.fit_transform(inputs_test)

scaler.fit(os_inputs)
os_inputs = scaler.fit_transform(os_inputs)

scaler.fit(us_inputs)
us_inputs = scaler.fit_transform(us_inputs)
```

Train Test Split & Classification

From the datasets we split 75% for training and 25% for testing with the stratify parameter to guarantee equal class ratio for both train and test. For the classification we used the following algorithms:

- Decision Trees
- Support Vector Machines
- K-Nearest Neighbors
- Multilayer Perceptron

We used K-Fold cross validation with 10 splits and grid search to do hyperparameter tuning for all algorithms.

These procedures were applied to all 3 datasets (original, undersampled and oversampled), so we can have more results to compare.

```
dt_classifier = DecisionTreeClassifier(random_state=0)

dt_tuned_parameter = {'criterion': ['gini','entropy'],
                      'splitter': ['best', 'random'],
                      'max_depth': [1, 2, 3, 4, 5],
                      'max_features': [1, 2, 3, 4, 'sqrt', 'auto','log2']}

dt_grid_search = GridSearchCV(dt_classifier,
                              param_grid=dt_tuned_parameter,
                              scoring='precision_weighted',
                              cv=10)

dt_grid_search.fit(inputs_train, labels_train)
print('Best score: {}'.format(dt_grid_search.best_score_))
print('Best parameters: {}'.format(dt_grid_search.best_params_))
```

Results Analysis - Decision Trees

```
Decision Trees - Original
=====================================================
Best score: 0.7419896790748586
Best parameters: {'criterion': 'gini', 'max_depth': 2,
=====================================================
TRAIN
Accuracy Score: 0.7447698744769874
Precision Score: 0.742152466367713
Confusion Matrix:
[[ 25 115]
 [  7 331]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.78      0.18      0.29       140
         buy       0.74      0.98      0.84       338

    accuracy                           0.74       478
   macro avg       0.76      0.58      0.57       478
weighted avg       0.75      0.74      0.68       478

=====================================================
TEST
Accuracy Score: 0.7125
Precision Score: 0.7410071942446043
Confusion Matrix:
[[ 11  36]
 [ 10 103]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.52      0.23      0.32        47
         buy       0.74      0.91      0.82       113

    accuracy                           0.71       160
   macro avg       0.63      0.57      0.57       160
weighted avg       0.68      0.71      0.67       160
```

```
Decision Trees - Undersampled
=====================================================
Best score: 0.6998667169603702
Best parameters: {'criterion': 'entropy', 'max_depth':
=====================================================
TRAIN
Accuracy Score: 0.33472803347280333
Precision Score: 0.8571428571428571
Confusion Matrix:
[[136   4]
 [314  24]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.30      0.97      0.46       140
         buy       0.86      0.07      0.13       338

    accuracy                           0.33       478
   macro avg       0.58      0.52      0.30       478
weighted avg       0.69      0.33      0.23       478

=====================================================
TEST
Accuracy Score: 0.54375
Precision Score: 0.8703703703703703
Confusion Matrix:
[[40  7]
 [66 47]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.38      0.85      0.52        47
         buy       0.87      0.42      0.56       113

    accuracy                           0.54       160
   macro avg       0.62      0.63      0.54       160
weighted avg       0.73      0.54      0.55       160
```

```
Decision Trees - Oversampled
=====================================================
Best score: 0.729232735992201
Best parameters: {'criterion': 'gini', 'max_depth': 4,
=====================================================
TRAIN
Accuracy Score: 0.6631799163179917
Precision Score: 0.7959866220735786
Confusion Matrix:
[[ 79  61]
 [100 238]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.44      0.56      0.50       140
         buy       0.80      0.70      0.75       338

    accuracy                           0.66       478
   macro avg       0.62      0.63      0.62       478
weighted avg       0.69      0.66      0.67       478

=====================================================
TEST
Accuracy Score: 0.6125
Precision Score: 0.6946564885496184
Confusion Matrix:
[[ 7 40]
 [22 91]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.24      0.15      0.18        47
         buy       0.69      0.81      0.75       113

    accuracy                           0.61       160
   macro avg       0.47      0.48      0.47       160
weighted avg       0.56      0.61      0.58       160
```

Results Analysis - Support Vector Machines

```
Support Vector Machines - Original
=====================================================
Best score: 0.7126628935827595
Best parameters: {'C': 100, 'gamma': 0.0001, 'kernel':
=====================================================
TRAIN
Accuracy Score: 0.7468619246861925
Precision Score: 0.737417943107221
Confusion Matrix:
[[ 20 120]
 [  1 337]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.95      0.14      0.25       140
         buy       0.74      1.00      0.85       338

    accuracy                           0.75       478
   macro avg       0.84      0.57      0.55       478
weighted avg       0.80      0.75      0.67       478

=====================================================
TEST
Accuracy Score: 0.7
Precision Score: 0.7152317880794702
Confusion Matrix:
[[  4  43]
 [  5 108]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.44      0.09      0.14        47
         buy       0.72      0.96      0.82       113

    accuracy                           0.70       160
   macro avg       0.58      0.52      0.48       160
weighted avg       0.64      0.70      0.62       160
```

```
Support Vector Machines - Undersampled
=====================================================
Best score: 0.679890873015873
Best parameters: {'C': 10, 'gamma': 'scale', 'kernel':
=====================================================
TRAIN
Accuracy Score: 0.700836820083682
Precision Score: 0.9079497907949791
Confusion Matrix:
[[118  22]
 [121 217]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.49      0.84      0.62       140
         buy       0.91      0.64      0.75       338

    accuracy                           0.70       478
   macro avg       0.70      0.74      0.69       478
weighted avg       0.79      0.70      0.71       478

=====================================================
TEST
Accuracy Score: 0.5375
Precision Score: 0.7746478873239436
Confusion Matrix:
[[31 16]
 [58 55]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.35      0.66      0.46        47
         buy       0.77      0.49      0.60       113

    accuracy                           0.54       160
   macro avg       0.56      0.57      0.53       160
weighted avg       0.65      0.54      0.56       160
```

```
Support Vector Machines - Oversampled
=====================================================
Best score: 0.7632888533225565
Best parameters: {'C': 100, 'gamma': 'auto', 'kernel':
=====================================================
TRAIN
Accuracy Score: 0.8200836820083682
Precision Score: 0.9809160305343512
Confusion Matrix:
[[135   5]
 [ 81 257]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.62      0.96      0.76       140
         buy       0.98      0.76      0.86       338

    accuracy                           0.82       478
   macro avg       0.80      0.86      0.81       478
weighted avg       0.88      0.82      0.83       478

=====================================================
TEST
Accuracy Score: 0.64375
Precision Score: 0.7745098039215687
Confusion Matrix:
[[24 23]
 [34 79]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.41      0.51      0.46        47
         buy       0.77      0.70      0.73       113

    accuracy                           0.64       160
   macro avg       0.59      0.60      0.60       160
weighted avg       0.67      0.64      0.65       160
```

Results Analysis - K-Nearest Neighbors

```
K-Nearest Neighbors - Original
=====================================================
Best score: 0.7179400084297429
Best parameters: {'n_neighbors': 18, 'p': 1, 'weights':
=====================================================
TRAIN
Accuracy Score: 0.7635983263598326
Precision Score: 0.7622377622377622
Confusion Matrix:
[[ 38 102]
 [ 11 327]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.78      0.27      0.40       140
         buy       0.76      0.97      0.85       338

    accuracy                           0.76       478
   macro avg       0.77      0.62      0.63       478
weighted avg       0.77      0.76      0.72       478

=====================================================
TEST
Accuracy Score: 0.7125
Precision Score: 0.7310344827586207
Confusion Matrix:
[[  8  39]
 [  7 106]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.53      0.17      0.26        47
         buy       0.73      0.94      0.82       113

    accuracy                           0.71       160
   macro avg       0.63      0.55      0.54       160
weighted avg       0.67      0.71      0.66       160
```

```
K-Nearest Neighbors - Undersampled
=====================================================
Best score: 0.6750608208870592
Best parameters: {'n_neighbors': 16, 'p': 2, 'weights'
=====================================================
TRAIN
Accuracy Score: 0.6694560669456067
Precision Score: 0.8333333333333334
Confusion Matrix:
[[ 95  45]
 [113 225]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.46      0.68      0.55       140
         buy       0.83      0.67      0.74       338

    accuracy                           0.67       478
   macro avg       0.65      0.67      0.64       478
weighted avg       0.72      0.67      0.68       478

=====================================================
TEST
Accuracy Score: 0.6375
Precision Score: 0.8235294117647058
Confusion Matrix:
[[32 15]
 [43 70]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.43      0.68      0.52        47
         buy       0.82      0.62      0.71       113

    accuracy                           0.64       160
   macro avg       0.63      0.65      0.62       160
weighted avg       0.71      0.64      0.65       160
```

```
K-Nearest Neighbors - Oversampled
=====================================================
Best score: 0.8231892163864876
Best parameters: {'n_neighbors': 1, 'p': 1, 'weights':
=====================================================
TRAIN
Accuracy Score: 1.0
Precision Score: 1.0
Confusion Matrix:
[[140   0]
 [  0 338]]
Classification Report:
              precision    recall  f1-score   support

      ignore       1.00      1.00      1.00       140
         buy       1.00      1.00      1.00       338

    accuracy                           1.00       478
   macro avg       1.00      1.00      1.00       478
weighted avg       1.00      1.00      1.00       478

=====================================================
TEST
Accuracy Score: 0.60625
Precision Score: 0.7450980392156863
Confusion Matrix:
[[21 26]
 [37 76]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.36      0.45      0.40        47
         buy       0.75      0.67      0.71       113

    accuracy                           0.61       160
   macro avg       0.55      0.56      0.55       160
weighted avg       0.63      0.61      0.62       160
```

Results Analysis - Multilayer Perceptron

```
Multilayer Perceptron - Original
=====================================================
Best score: 0.7409533850365232
Best parameters: {'activation': 'logistic', 'alpha': 0
0.25, 'solver': 'adam'}
=====================================================
TRAIN
Accuracy Score: 0.7824267782426778
Precision Score: 0.7839805825242718
Confusion Matrix:
[[ 51  89]
 [ 15 323]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.77      0.36      0.50       140
         buy       0.78      0.96      0.86       338

    accuracy                           0.78       478
   macro avg       0.78      0.66      0.68       478
weighted avg       0.78      0.78      0.75       478

=====================================================
TEST
Accuracy Score: 0.69375
Precision Score: 0.7758620689655172
Confusion Matrix:
[[21 26]
 [23 90]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.48      0.45      0.46        47
         buy       0.78      0.80      0.79       113

    accuracy                           0.69       160
   macro avg       0.63      0.62      0.62       160
weighted avg       0.69      0.69      0.69       160
```

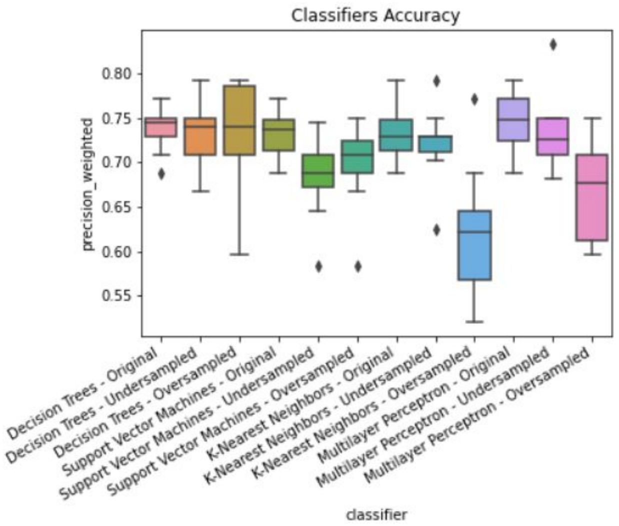
```
Multilayer Perceptron - Undersampled
=====================================================
Best score: 0.7051463195387344
Best parameters: {'activation': 'tanh', 'alpha': 0.000
0.25, 'solver': 'adam'}
=====================================================
TRAIN
Accuracy Score: 0.6715481171548117
Precision Score: 0.9248826291079812
Confusion Matrix:
[[124  16]
 [141 197]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.47      0.89      0.61       140
         buy       0.92      0.58      0.72       338

    accuracy                           0.67       478
   macro avg       0.70      0.73      0.66       478
weighted avg       0.79      0.67      0.68       478

=====================================================
TEST
Accuracy Score: 0.56875
Precision Score: 0.8142857142857143
Confusion Matrix:
[[34 13]
 [56 57]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.38      0.72      0.50        47
         buy       0.81      0.50      0.62       113

    accuracy                           0.57       160
   macro avg       0.60      0.61      0.56       160
weighted avg       0.69      0.57      0.59       160
```

```
Multilayer Perceptron - Oversampled
=====================================================
Best score: 0.8286409331661215
Best parameters: {'activation': 'relu', 'alpha': 0.000
0.25, 'solver': 'lbfgs'}
=====================================================
TRAIN
Accuracy Score: 0.7803347280334728
Precision Score: 1.0
Confusion Matrix:
[[140   0]
 [105 233]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.57      1.00      0.73       140
         buy       1.00      0.69      0.82       338

    accuracy                           0.78       478
   macro avg       0.79      0.84      0.77       478
weighted avg       0.87      0.78      0.79       478

=====================================================
TEST
Accuracy Score: 0.6125
Precision Score: 0.7741935483870968
Confusion Matrix:
[[26 21]
 [41 72]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.39      0.55      0.46        47
         buy       0.77      0.64      0.70       113

    accuracy                           0.61       160
   macro avg       0.58      0.60      0.58       160
weighted avg       0.66      0.61      0.63       160
```

Oversampled and Undersampled data sets’ problems

As we can see in the graph, the undersampled data set usually underperforms, which might be because of the lack of data, since we removed a lot earlier.

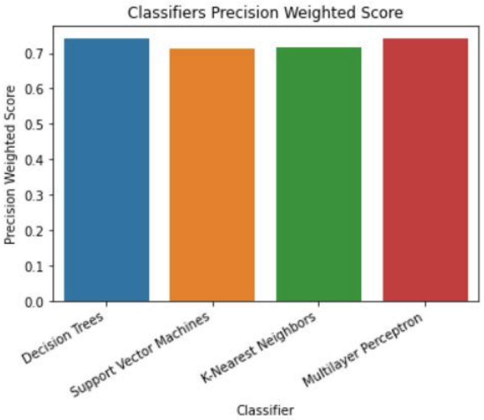
As for the oversampled set, although it makes some models achieve higher scores, it also makes them overfit, since we can see in the graph that the scores vary too much.



Best Model

To choose the best model, we first need to decide which metric (accuracy, precision, recall and F1) is the best for this context. Since the dataset is imbalanced, we can rule out accuracy, leaving us with 3 metrics, and from those 3 we can certainly say that precision is the most important since there is a cost associated with every **buy** prediction the model makes.

With that said, although all of the algorithms and resulting models (for the original dataset) presented us with good results (>70%), we can say that Multilayer Perceptron is the best performing model (74%), since it has the best precision/recall score.




```
TEST
Accuracy Score: 0.69375
Precision Score: 0.7758620689655172
Confusion Matrix:
[[21 26]
 [23 90]]
Classification Report:
              precision    recall  f1-score   support

      ignore       0.48      0.45      0.46        47
         buy       0.78      0.80      0.79       113

    accuracy                           0.69       160
   macro avg       0.63      0.62      0.62       160
weighted avg       0.69      0.69      0.69       160
```

Conclusion

When we first started this project we were going for an accuracy score of about 50-60%, our first experimentations with even the simplest algorithms (like Decision Trees and K-Nearest Neighbors) quickly exceeded our expectations, with results already close to 70%. With all the research and development we did to both the data and the algorithms we managed to get the accuracy of all the used algorithms to over 70%, which is very decent.

With this project we learned that Machine Learning algorithms are very powerful tools that can predict with decent accuracy even the very unpredictable thing that is the Stock Market. Although we achieved good results, these could've been better if more data had been provided, seeing that we only had 638 companies for 100+ features.

Finally, we had the opportunity to see how data analysis/preprocessing and other techniques like k-fold cross-validation and hyperparameter tuning are fundamental in guaranteeing a good Supervised Learning Model.