PYTHON

Forecasting S&P 500 using Machine Learning

lcampos
(https://quantdare.com/author/lcampos/) *20/12/2017*

Is it possible to foresee the future movements of a stock? Let's use Machine Learning techniques to predict the direction of one of the most important stock indexes, the S&P 500.

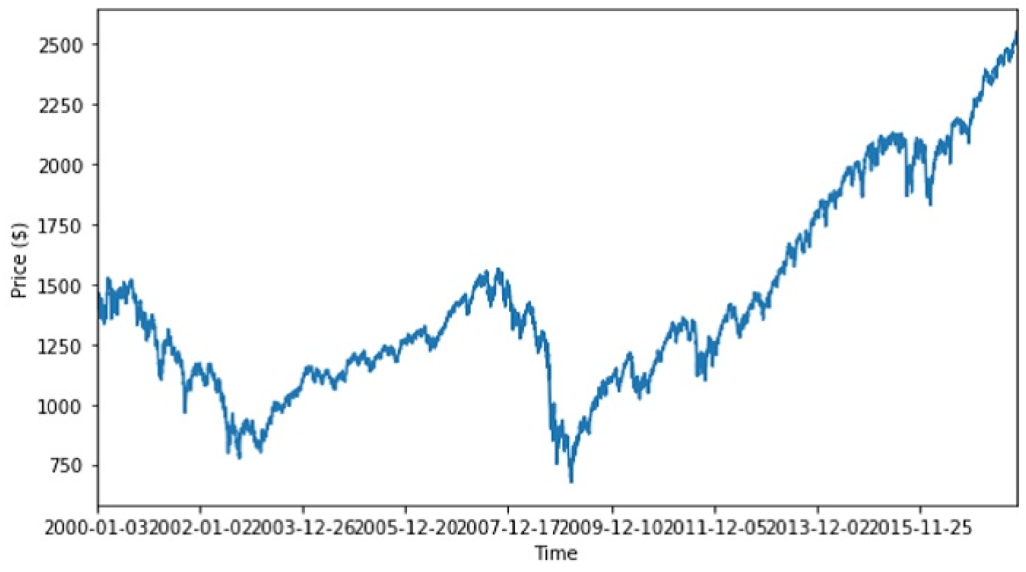
Pregaming

The Standard & Poor's 500 (S&P500) is a stock market index based on the capitalization of the 500 largest American companies. It is an index widely traded through index funds and ETFs, which allow us to speculate its future value.

We are going to build a Machine Learning model that takes information from the past price of the stock, and returns a prediction on the movement of that stock price the next day. The final goal is to make trading profits when investing in the direction of our prediction.

It's all about data

First things first: data. Due to the recent shutdown Yahoo! Finance API, which used to be the most widely-used free finance data provider, some extra work had to be done to find a proper data source. Luckily, we came across Alpha Vantage, an open finance data provider with a nice Python API that besides naked price data, provides very useful trading technical indicators.

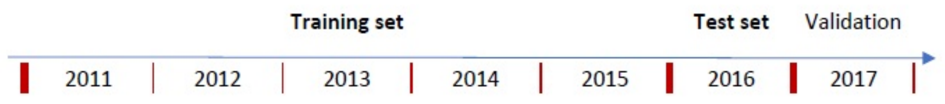


In order to feed our Machine Learning models, both the naked price and a range of different technical indicators computed over it have been chosen:

- Simple Moving Average.
- Weighted Moving Average.
- Exponential Moving Average.
- MACD (moving average convergence divergence).
- Momentum.
- Stochastic Oscillator.
- RSI (Relative Strength Index).
- ADX (Average directional movement index).
- Larry William's R%.
- A/D (Accumulation/Distribution) Oscillator.
- CCI (Commodity Channel Index).
- Aroon values.
- Bollinger bands.
- On Balance Volume.

Those are going to be our features. In order to feed the models properly, we have scaled them using a scikit-learn standard scaler. On the other hand, our label is going to be the movement of the close price the day after: 0 if the price goes down and 1 if it goes up.

For training, testing and validating, data has been split into the following sets:



The key point here is to make sure all the samples, in both test and validation sets, are later in time than those of the training set. That way we make sure the model is not fed with future information.

An important issue to address, before starting to work with the models, is the imbalance between classes. As there are more days where the prices move upwards, classes should be rebalanced so the models do not hit a higher accuracy by classifying all samples to the most frequent class. To address this problem, several actions might be undertaken:

1. Up-sample: resample the dataset in order to make the number of classes are equal for both groups. This way, a balanced dataset, is achieved without dropping useful information (*down-sample*).
2. Alternative performance metrics: such as accuracy, area under the ROC curve, F1 score...
3. Algorithms penalization: increase the cost of classification mistakes on the minority class (*Penalized-SVM*).

As it is enough to address the problem from just one approach, we have *up-sample* the dataset. Even so, a higher number of metrics have been used to score the models, as it does no damage at all... 🙂

Building the models

There are several algorithms that suit our problem. We have chosen Support Vector Machine (SVM), Random Forest and Neural Network multi-layer perceptron classifiers.

To fine-tune the models, we are going to use hyperparameter optimization, concretely the well-known *scikit-learn* Grid search algorithm. It works by manually specifying a subset of the hyperparameters to test and some performance metric to guide the algorithm (*F1 score* in our case).

After optimization, the best parameters are found to be:

- SVM: 3rd degree polynomial kernel with regularization parameter C 100.
- Random Forest: 10 number of estimators, entropy criterion, 10 maximum depth and 10 minimum samples split.
- Multi-layer perceptron: 50 hidden layer neurons, RELU activation, LBFGS solver, constant learning rate and 200 maximum iterations.

Remember that hyperparameters are optimized against the training data. Now, it's time to check if the models work with the test set:

	Accuracy	F1	AUC ROC
SVM	51.29 %	0.5570	0.5102
Random Forest	47.97 %	0.5188	0.4775
Neural Network	53.14 %	0.4596	0.5364

After refining and scoring the best three models (*SVC*, *RF* and *MLPC*) against the test data, the Support Vector Machines classifier appears to be the most reliable model, being better than random chance all along the three scores. Let's do a last check against the validation set:

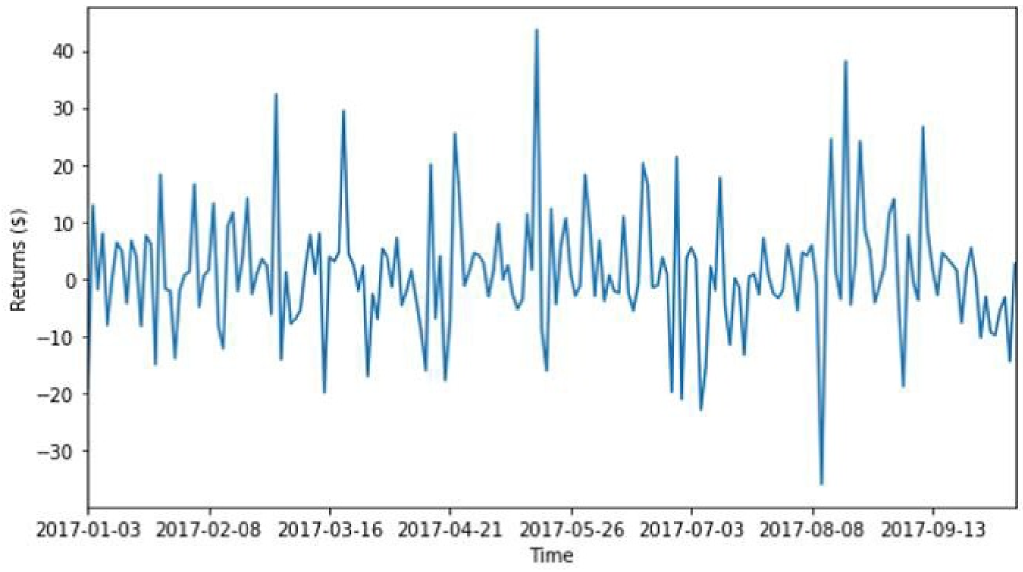
	Accuracy	F1	AUC ROC
SVM Classifier	54.92 %	0.6329	0.5286

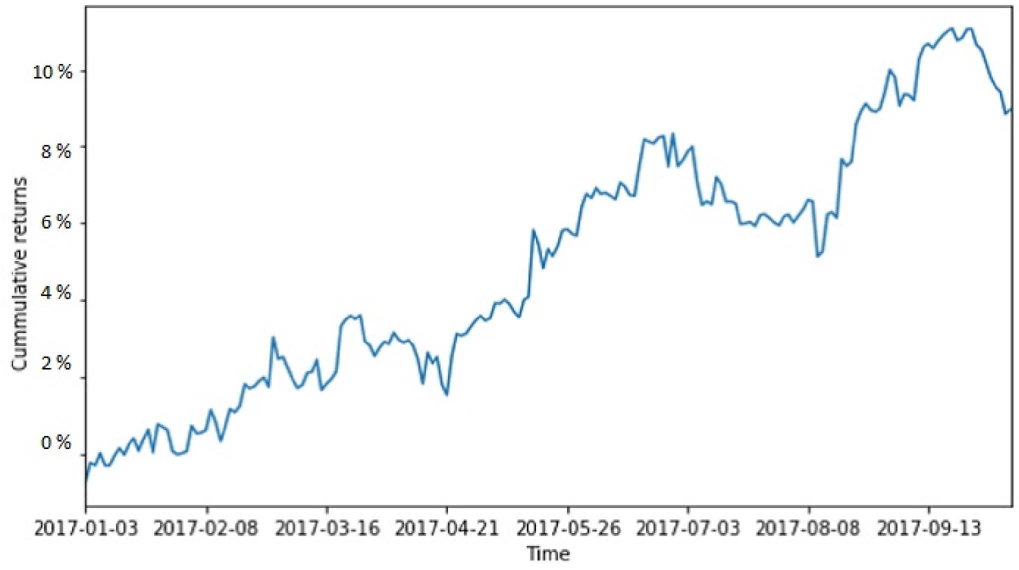
Looks like we have achieved a decent performance with our *Support Vector*

Machines classifier! As a final step, let's implement it with real trading signals.

Playground

We are building a trading model that takes a *long* position (buy) if it predicts the price is going up and a *short* position (sell) if it predicts a downwards movement of the index.





As it can be seen, if the model would have been in charge of a trading system, during the period from *2017-01-01* to *2017-09-05* we would have obtained a performance of *9.92 %*.

Due to the great performance of the S&P500, the returns are pretty similar to those we would have obtained by merely holding the stock (and we are not taking into account fees…). The most interesting application of a long/short strategy like this one is that we are able to obtain positive returns during periods of continuous loses.

You can take a look at the code on my GitHub profile. Thanks for reading!

RELATED POSTS

(https://quantdare.com/hierarchical-clustering-of-etfs/)

(https://quantdare.com/hierarchical-clustering-of-etfs/)
(HTTPS://QUANTDARE.COM/CATEGORY/PYTHON/)

Hierarchical clustering of Exchange-Traded Funds

(https://quantdare.com/hierarchical-clustering-of-etfs/)

pmercatoris
(https://quantdare.com/author/pmercatoris/)

(https://quantdare.com/risk-parity-in-python/)

(https://quantdare.com/risk-parity-in-python/)
(HTTPS://QUANTDARE.COM/CATEGORY/RISK-MANAGEMENT/)

Risk Parity in Python

(https://quantdare.com/risk-parity-in-python/)

fjrodriguez2
(https://quantdare.com/author/fjrodriguez2/)

(https://quantdare.com/world-connections-indexes/)

(https://quantdare.com/world-connections-indexes/)
(HTTPS://QUANTDARE.COM/CATEGORY/PYTHON/)

World Connections using financial indexes

(https://quantdare.com/world-connections-indexes/)

psanchezcri
(https://quantdare.com/author/psanchezcri/)

(https://quantdare.com/calculate-monthly-returns-with-pandas/)

(https://quantdare.com/calculate-monthly-returns-with-pandas/)
(HTTPS://QUANTDARE.COM/CATEGORY/PYTHON/)

Calculate monthly returns…with Pandas

(https://quantdare.com/calculate-monthly-returns-with-pandas/)

mgreco
(https://quantdare.com/author/mgreco/)

ADD A COMMENT

Message