Machine Learning Project

Stock market movements prediction

Abstract

The goal of the project is trying to predict stock market movements.

The main idea is that stock markets nowadays are interlinked, so a change in a particular stock might reflect on different stocks.

The project aims to analyze several stock indices in order to predict whether a specific asset is going to increase or decrease in the following day.

This asset is SP500, which is based on 500 big companies around the world, and is generally used as a market benchmark.

Table of Contents

Data gathering

The first step was data gathering: stock market data is freely available on the web. I took my dataset from Yahoo Finance.

I downloaded data for the following stock markets indices:

- Australia (“^AXJO”)
- Dow jones (“^DJI)
- Frankfurt (“^GDAXI)
- Honk Kong (“^HSI”)
- NASDAQ (“^IXIC”)
- Tokyo (“^N225”)
- Paris (“^FCHI”)
- Standard & Poor’s 500 (“^SP500TR”)

I downloaded data starting from 2010, since I only wanted to focus on recent market movements.

Initial data preprocessing

The first step was some basic data preprocessing.

First of all, I removed all null entries using a simple regex and manual editing of files (since it was just a few files I didn’t deed it necessary to use a more sophisticated approach).

Then I loaded the dataset in Python using *“pandas”* library. This resulted in 8 different *DataFrames*, one for each stock market.

Each dataset had several columns:

- “Date”
- “Open”
  - Opening price of the stock
- “Close”
  - Closing price
- “Adj Close”
  - Adjusted closed price. It is supposed to take more factors into account than Close, but on all datasets I had, there was not difference between the two columns
- ‘High”
  - Highest price reached during that day
- “Low”
  - Lowest price reached during that day
- “Volume”
  - Number of transactions executed on the given day

Then I removed some unneeded columns, in details:

- “Adj Close”, because it was identical to “Close”
- “Volume”, since it had several inconsistencies
  - For some stocks all values were 0, for others there were a bunch of missing values

The last step was renaming the columns in order to facilitate joining all these datasets into a single one (which will be done and explained later).

Feature generation

The next step was feature generation.

I computed several metrics from the data available and added them to each dataset.
Each new metric is generated as a percentage, so all new values are already normalized.

This solved an important issue, since different stocks usually have very different prices, and putting these data directly into the model wouldn't provide reliable results.

To this end, I deleted the original values from the datasets after having generated all the features.

Features

I decided to focus on four different financial metrics.

- Daily returns
- Multiple days returns
- Returns moving average
- Time lagged results

Daily returns

This metric represents the variation of price on a given day compared to the previous day.

For instance, if yesterday's price was 2$ and today's is 3$, then we would have a return value of 0.5, or an increase of 50%.

Multiple days returns

This feature is similar to daily return, but is calculated on a window of several days.

Returns moving average

This feature calculates the average returns of the last n days.

Time lagged results

This feature provides the values of previous n days.

For instance, if n = 1, then each day would also have the results of the previous day.

Selected features

In the end, I decided to select the features which gave me a higher probability of guessing. These are:

- Daily return for each stock
    - These values were also lagged both one and two days back (so, each day also had the returns of the previous two days)
- Multiple days return for each stock
    - Computed on windows of 2 and 3 days
    - The values computed on a window of two days were also lagged one and two days back, analogously to the daily returns
- Returns moving average for each stock
    - Computed on windows of 2 and 3 days

I decided to use this set of features after some experimenting. I tried to use both larger and smaller windows and to lag even the returns moving average, but doing so only decreased my probability of guessing.

Target output

The last feature I generated was a column indicating if the price of my target stock ("SP500TR") was going to increase or decrease in the following day.

This column will be the output of my prediction, and the goal of the project.

Dataset joining

The last step is to join all datasets created into a single one

This step proved a little tricky because stock markets are located on different countries, so they're going to have national holidays on different dates.

To solve this issue, I decided to perform an inner join between the datasets, in order to keep only those days where I have all the information I need.

Thanks to the names I had assigned previously to the columns of each dataset, I simply had to specify on which column to perform the join operation (which was "Date").

If I had assigned different names to the columns I would have had one of the following scenarios:

- All dataset had the same column names:
  - In this case I would have had to specify how to rename the columns for each dataset
  - For example, *"Close"* would have needed to be renamed to *"Close_dji"* or *"Close_ixic"*, depending from which dataset it came from.
- All datasets had different column names:
  - I would have had to specify different names for the join column
  - For example, I would have needed to specify that *"Date_dji"* and *"Date_ixic"* referred to the same concept.

Machine learning

After having generated the dataset, I applied several machine learning algorithms to try and predict whether the price for the following day was going to increase or decrease.

I selected 5 amongst the most popular machine learning algorithms, specifically:

- Random forests
- K-nearest neighbors
- Support vector machines
- Adaptive boosting
- Gradient tree boosting

Then I split each dataset in two halves: the first one was the training set, the second one the testing set.

Finally, I executed all algorithms on the two sets and printed how many times my prediction was correct.

Cross validation

I also decided to cross validate my predictions, but at the same time I didn't want to take data randomly from the dataset, because time ordering is an important factor in the domain of the project.

So I decided to split the dataset in folds, smaller datasets which go up to a specific date.

In my case, I split the dataset in 5 folds, where the first one contains the first fifth of my original data, the second one contains the first two fifths, and so on…

On each dataset I then applied all machine learning algorithms and printed the results.

Results

The results, as I expected, were quite disappointing.

On the whole dataset I managed to obtain at most a precision of 56%, with the random forest algorithm.

We also need to remember that I'm trying to predict only if the price is going to increase or decrease, and not by how much, so the problem is classification and not regression.

This means that by tossing a coin I would have a 50% probability of guessing; my results were barely more efficient then a coin toss.

This was actually expected, as I'm going against the Efficient Market Hypothesis, which states that stock prices reflect all possible available information, thus including this very algorithm, if it would have been successful (More info: https://en.wikipedia.org/wiki/Efficient-market_hypothesis).

I also noticed that on a small dataset (namely, the first two folds) the scores of my predictions are even below 50%, so I concluded that my algorithm is inefficient and unreliable with few data available.

On the other hand, on the whole dataset I managed to obtain only positive scores (greater than 50%), which means that my algorithm isn't wrong, even though the results obtained are not reliable enough to be used in a real-world scenario.

This is not only because of the probability of guessing being very low, but also because it doesn't reflect by how much it's going to change.

For instance, the price could increase slightly and then decrease severely, and if only the increase is predicted it would still lead to a loss.

Model simplifications

Finally, we also have to remember that the project is making several simplifications on the actual model of the stock market.

First of all, I'm considering only closing prices of stocks; this means that I'm also assuming that assets will open at the same prices they had closed, which in a real-world scenario doesn't happen.

Then I'm also ignoring both volume of transactions and high and low prices reached during each day.

I decided not to use those values because I noticed that they decreased the probability of guessing, but they still hold information on the market.

Lastly, I'm also assuming that all the information needed to predict the stocks are inside the market itself, which is almost never the case. Stocks vary also according to news and other events around the world, which are not contained in the market itself.

Final considerations

I also noticed that, even on small datasets, I have sometimes a high score of correct predictions (even up to 59% on the second fold, using SVMs).

Since those values stabilize on a lower value (around 54%) on the whole dataset, I believe the higher rate could be caused by some accidental form of overfitting on the first parts of the dataset.

Results obtained

Here are reported the results obtained on the whole dataset:

- Fold 1, from 2011-01-31 to 2012-06-20 (273 values)

Random forest:	0.481751824818
K-nearest neighbor:	0.481751824818
Support vector machines:	0.554744525547
Adaptive boost:	0.510948905109
Gradient tree boost:	0.467153284672

- Fold 2, from 2011-01-31 to 2013-10-24 (546 values)

Random forest:	0.498168498168
K-nearest neighbor:	0.461538461538
Support vector machines:	0.589743589744
Adaptive boost:	0.501831501832
Gradient tree boost:	0.549450549451

- Fold 3, from 2011-01-31 to 2015-03-25 (819 values)

Random forest:	0.541463414634
K-nearest neighbor:	0.521951219512
Support vector machines:	0.578048780488
Adaptive boost:	0.534146341463
Gradient tree boost:	0.531707317073

- Fold 4, from 2011-01-31 to 2016-08-03 (1092 values)

Random forest:	0.571428571429
K-nearest neighbor:	0.5
Support vector machines:	0.545787545788
Adaptive boost:	0.554945054945
Gradient tree boost:	0.527472527473

- Fold 5, from 2011-01-31 to 2017-12-12 (1365 values)

Random forest:	0.560761346999
K-nearest neighbor:	0.531478770132
Support vector machines:	0.544655929722
Adaptive boost:	0.512445095168
Gradient tree boost:	0.544655929722