Report
Topic: Stock price prediction and selection based on risk minimization

Amogha | 150040102

Aim:

To explore various models for effectively predicting the stock price of an equity.

Data used:

We have used the past 15 years microsoft stock price-volume data for some of the equities traded. This data includes the adjusted open price, close price, high, low, volume traded, split and dividend data. All these have been inputted as a CSV file to our code.

Adjusted prices are prices amended to include any distributions and corporate actions such as stock splits (splitting one stock into two which would halve the price), dividends (giving stockholders cash as a fraction of profits) that occurred at any time before the next day's open.

Interesting characteristics of this problem

1. Predicting multiple outputs: We will predict the adjusted close prices for 7 days after the last input date.
2. Extracting and engineering the input data as opposed to being given input data.
3. We used time series data.

Analysis of Problem

This is a regression problem (as opposed to a classification problem) because we are predicting daily Adjusted Close prices for a stock.Also we have trained and used a Neural network These prices are continuous.

It's not immediately obvious what kind of model will be best.

Characteristic of problem:

- Time-series data.
- Noisy data
- Datapoints (prices of different stocks) are not independent of each other -> Naive Bayes is not appropriate
- Many features. (Daily open, high, low, adjusted close for many stocks) ->
- Regression problem (**continuous output**).
- Training cost or time: it is not critical to keep this lower than 12 hours because we are predicting daily prices based

Metrics

We will measure performance as the root mean squared percentage error (difference between the stock's actual and predicted Adjusted Close prices).

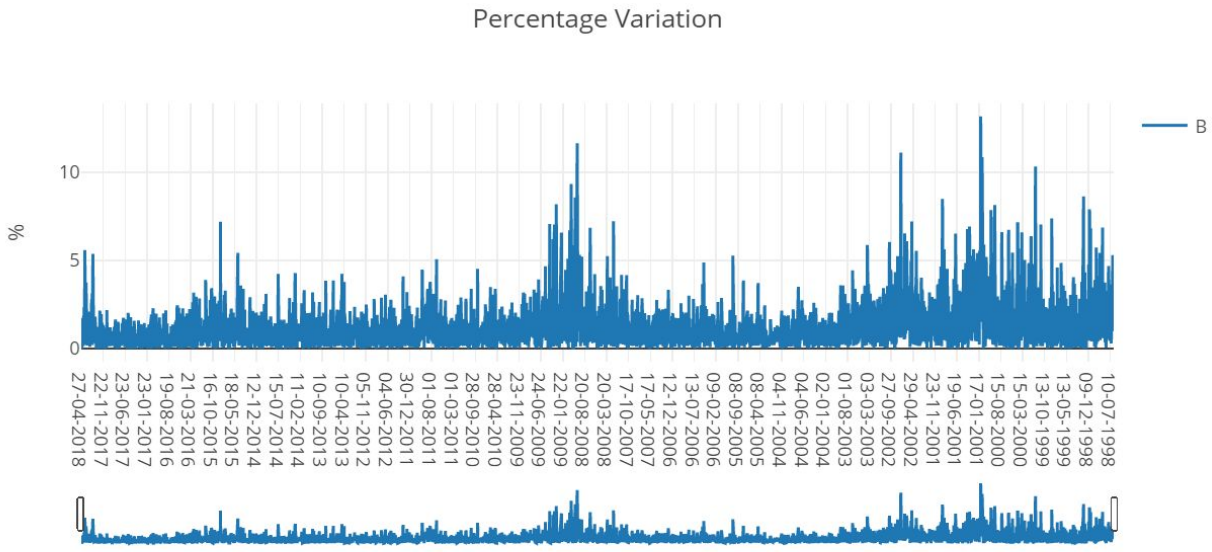
Reasoning:

1. This represents the error between the actual price and the predicted price.
2. We have to square it and then take the square root because if we don't square it, errors from overestimates and underestimates will cancel each other out.

We will also informally consider the range of root mean squared percentage error as a secondary metric - we want a model with lower error variance because a series of small good trades can be more than cancelled out by a single large-magnitude bad trade

We will not consider transaction costs (you have to pay every time you trade and that will reduce profits).

Percentage variation of stock prices



Techniques

1. Time-series train-test split
    - We will train our model on what we'll call the training set, a subset of the data that we have.
    - To make sure our model generalises, we need to test it on some data it has not seen before and evaluate how well it does predicting on that data.
    - To do this, we need to set aside data for testing our model - the test set.
    - Because our data is time series data (there is some ordering to it and the ordering influences prices), we cannot shuffle the data.
    - If the data were shuffled, e.g. the adjusted close price for 1 Nov 2015 might be in the training set. We might then be asked to predict the adjusted close prices for the 7 days after 31 Oct 2015, which would include the price for 1 Nov 2015 which we'd have seen before.
    - So we cannot use sklearn's `train_test_split` function which automatically shuffles the data. Instead, we have written our own function.

2) Time-series cross-validation

- But testing on only one test set and training on only one training set isn't robust enough. What if the test or training sets we choose have special characteristics that aren't common to other datasets?

- To make our evaluation more robust so we choose the best model, it's better if we can run multiple train-test cycles.
- To do this, we wrote the function `execute()`. In this function, we set a number of train-test cycles (`steps`), a total length of the train-test data (`periods` data-points) and a number of datapoints between the starting points of each consecutive train-test cycle (`buffer_step`).

We performed the following steps:

1) Import data (CSV) and format it as a Pandas Dataframe
2) Create features dataframe: Select features we wanted to use and put it into a separate data-frame
3) Create target dataframe (Prices for 7 days following the last date provided in the features).
4) Split into training and testing sets. (**No shuffle because we are dealing with time series data**.)
5) Train chosen classifier.
6) Predict test target.
7) Evaluate test target and print evaluation metrics.

Observations:

Days after last training date	Mean Root mean squared daily percentage error (across 8 distinct train-test sets)
1	1.306
2	1.954
3	2.354
4	2.747
5	3.039
6	3.267

7	3.434

Mean R2 score: 0.838.

Neural Network (had 15 layers) and SVM

Days after last training date	Mean Root mean squared daily percentage error (across 8 distinct train-test sets)
1	26.317
2	26.391
3	26.455
4	26.495
5	26.537
6	26.591
7	26.693

Mean R2 score: -11.83

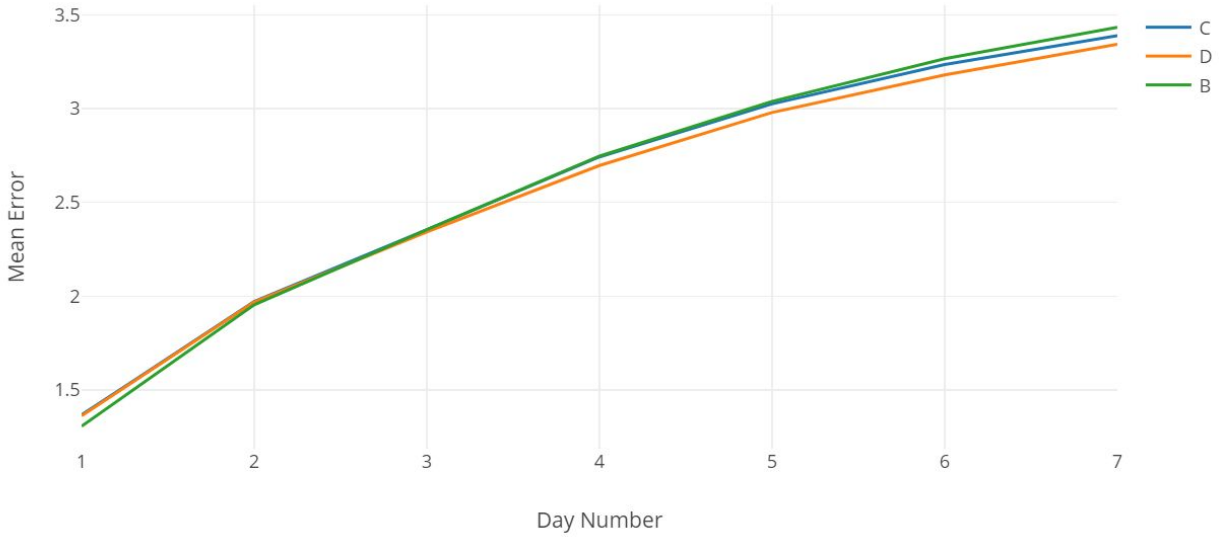
The Linear Regression did surprisingly well, with a mean R2 score above 0.836 overall for 7-day predictions and a mean RMS percentage error of under 5% for forecasts 7 days away.

The SVM regression did horribly - it had a negative mean R2 score (-11.834) and negative median R2 score, which means it was worse than guessing randomly. It had a mean RMS percentage error of over 24% for all number-of-days ahead predicted.

Why 7 days?

Day to predict	7d (used)	10d	14d	21d	30d	100d
1	1.306	1.366	1.361	1.1584	1.162	1.924
2	1.954	1.971	1.969	1.639	1.676	2.768
3	2.354	2.355	2.343	1.904	1.951	3.370
4	2.747	2.743	2.697	2.193	2.214	3.890
5	3.039	3.026	2.980	2.434	2.480	4.355
6	3.267	4.236	3.180	2.646	2.706	4.769
7	3.434	3.390	3.344	2.837	2.913	5.163

Number of days used



We can see that mean RMS percentage error is slightly smaller in one instance (using 10d instead of 7d to predict precisely 7 days ahead), but otherwise that mean RMS percentage error is greater as the number of days of data given increases.

Challenges faced:

1. The model has to be run for dates not within the training set for the model to be 'fair'. But given there may be big shifts in how people view the markets from year to year, it may be hard for the model to generalise from one year to the next.
2. Some of the companies' stock prices are volatile so they may be harder to predict.

Further Implementations possible:

1) Try better SVM regressors than the default one.
2) Try predicting the stock prices for every minute rather than just predicting the close values for each day.

References used :

→ Google.com

→ A paper on time series data

→ Python official website for module tutorials

→ Stack Exchange for random implementation doubts