Machine Learning Engineer Nanodegree

Capstone Project: Investment and Trading Capstone Project

Igwebuike Onyeka Daniel
January 31st, 2021

I. Definition

Project Overview

In very recent times, finance industries and firms have used trading models to predict possible future outcomes, and study market movements for better and more profits. These financial activities have also risen in numbers in terms of amount of transactions with the rapid global economic developments, and thus making their trend more complex to monitor.

> *"Forecasting time series is a long-standing research problem that is applicable to econometrics, marketing, astronomy, ocean sciences, and other domains. Similarly, networks are the subject of active research with broad relevance to areas such as transportation, internet infrastructure, signalling in biological processes, and online media. In this paper, we are concerned with forecasting among a network of time series with mutual influences. Tools for tackling this problem will help answer questions about complex systems evolving over time in the above application domains and beyond."* Alasdair Tran, 2021[1]

Leveraging on the immense amount of data available, machine learning models can use knowledge of past sequences to predict possible future results, using one of many machine learning algorithms available. Just as in many other fields of science, research and development, machine learning has proven also to be a great tool for financial modelling and analysis and more include portfolio optimization, stock betting and predictions.

> *"Before the emergence of efficient machine learning algorithms, researchers in China and elsewhere generally used various statistical and econometric methods to build prediction models for research. Conventional statistical and econometric models require linear models and cannot be used to predict and analyse financial products before transforming nonlinear models to linear models. As an important branch of machine learning algorithms, neural networks (NNs) have the following advantages compared to conventional statistical methods: they are numeric, data-driven and adaptive*." Pengfei Yu, 2019[2]

The aim of this project is to develop a model that learns from past available data and predict possible future outcomes at given time intervals.

Problem Statement

The challenge in trading predicting models is usually in the quickly varying values, which must be studied with temporal sequence as with most time-series predictions. Such models require special algorithms, because the conventional algorithms like linear regression, would not be able to take care of the presence of outliers or seasonal sporadic changes in trend.

Here -in this project- the aim is to make as accurate as possible 'tomorrow's' market price (financial outcome) from how the trend has been for a period of about 5-6 years, taking into consideration the market price of each day. This might sound a bit straight forward, however, when we compare it to predictions like value of house prices based on certain features, which uses a linear regression approach to predict the most likely outcome, given some feature values, we find out that such approach with a varying "unpredictable" event might not yield desired outcomes. This is largely because unlike the house prices, which depends on known contributing factors, stock market prices (just like weather changes) can't be predicted as linear, because they are not based on 'known' features or factors.

There are a lot of factors affecting how a market price might go up (or down), sometimes these factors might be speculated, other times it might be as a result of chain of events, and such variations are usually not what conventional algorithms are built on.

I sourced my data from Yahoo, taking data of stock market prices from five major stocks viz: Google, Apple Inc, Oracle, SPY, and IBM. Dating from January 1st 2008 (2008.01.01), to December 31st 2020 (2020.12.31), a period of ten (13) years. Using the Adjacent Close (Adj Close) column of this data, my aim was to build and train a model that would learn from these data and make close predictions of future outcomes.

For such a task, special algorithms or approach is needed. I considered a few, which includes using an Artificial Neural network (an RNN or LSTM), which makes use of recursive training, where a prediction not only depends on its immediate past, but also far into the past, a model can learn, and such learning is fed back into the system for better future outcomes.

However, after various searches and tutorials, and with the new knowledge and skill I learnt using AWS, I decided to try two dynamic and powerful algorithms, the AWS in-built XGBoost (eXtreme Gradient Boosting) and the external FbProphet. These algorithms perform great because of verse performance with various distributions, datatypes and their results with variety of machine learning problems.

The best approach would be to use these algorithms and compare their outputs (predicted values) with the test data, and against a known benchmark, and the run diagnosis to calculate for RMSE, and further testing for measure of performance. R^2 can also be calculated; R-squared (R^2), which is the proportion of variation in the outcome that is explained by the predictor variables. In multiple regression models, R^2 corresponds to the squared correlation between the observed outcome values and the predicted values by the model. The Higher the R-squared, the better the mode

The best approach is to build a model using one of the known algorithms, split the data into train and test groups, and train this model with the input train-group data, and then use the model to make predictions, and compare these predictions with the test data.

Metrics

I used the Adjacent Closing column of the stock tables to train and test my data. Three datasets were generated using a method, which takes the entire column and outputs three sets of data for training, validation and testing.

After building and training the model, testing was performed by making predictions and using the sklearn '*mean_squared_error*' method to compare and know how well the model did.

Well, these can give you different insights into the model's errors. If '*y*' is the target, '*p*' is the prediction and ….

$$\textit{Error, } e = p - y$$

- Mean Squared Error: MSE= mean(e^2): Measures additive bias in the error. It measures the square deviation in the prediction of the errors.
- Root Mean Squared Error: RMSE = √ mean(e^2): Root square is taken to make the units of the error be the same as the units of the target. This measure gives more weight to large deviations such as outliers, since large differences squared become larger and small (smaller than 1) differences squared become smaller.

I used the sklearn method to calculate for both MSE and RMSE for the xgboost algorithm. Fbprophet has a Diagnostics library which contains a performance_metrics method. This method returns a pandas data-frame which contains all common errors associated with the model.

The RMSE and MSE are the two errors I calculated and used for analysis, although there are several more errors that can be calculated for regression models.

- R^2, coefficient of determination: $R^2 = \sum (y - p)^2 / \sum (y - x)^2$: The closer to 1 the better. This is a measure of the ratio of variability that your model can capture vs the natural variability in the target variable. This is however affected by the number of independent variables or features to the model.

In practice usually there is the use of a combination of MSE, R^2 and RMSE if there are no outliers in the data.
These metrics can be calculated easily using the sklearn library.

II. Analysis

Data Exploration

First is the importation of all the needed libraries for the project, which is used to import the needed dataset. I generated my dataset from online Yahoo Finance source using *pandas-datareader*, passing arguments of the start and end dates ranging from January 2008 to December 2020:

- *df_ORCL = web.DataReader('ORCL', data_source="yahoo", start=start_date, end=end_date)*

Date	High	Low	Open	Close	Volume	Adj Close
2008-01-02 00:00:00	22,82	22,38	22,55	22,49	42775700	19,25546
2008-01-03 00:00:00	23,11	22,43	22,43	23,11	42045400	19,78629
2008-01-04 00:00:00	22,88	21,79	22,77	22,03	44738600	18,86162
2008-01-07 00:00:00	22,48	21,79	21,96	22,25	40045600	19,04998
2008-01-08 00:00:00	22,32	21,14	22,24	21,15	43551600	18,10818
2008-01-09 00:00:00	21,69	21,15	21,34	21,61	47973000	18,50203
2008-01-10 00:00:00	21,8	21,07	21,46	21,68	42107400	18,56195
2008-01-11 00:00:00	21,69	21,06	21,53	21,1	46251400	18,06538
2008-01-14 00:00:00	22,12	21,1	21,44	22,06	48826100	18,8873
2008-01-15 00:00:00	22,05	21,3	21,76	21,31	39579500	18,24517
2008-01-16 00:00:00	22,33	20,85	21,03	21,92	76869400	18,76744
2008-01-17 00:00:00	21,95	21,3	21,9	21,41	52783800	18,33078
2008-01-18 00:00:00	21,83	21,17	21,35	21,58	51486500	18,47634
2008-01-22 00:00:00	20,79	19,68	20,28	20,22	66962800	17,31193
2008-01-23 00:00:00	20,68	19,52	19,58	20,61	56349500	17,64584
2008-01-24 00:00:00	20,97	20,15	20,6	20,61	43764900	17,64584
2008-01-25 00:00:00	21,08	20,22	21	20,28	42303500	17,36331

Table 1. df_ORCL head table

A larger amount of data makes it better for the training model and thus results. These data sources include (but not limited to) Google, Yahoo, Bloomberg etc. However, for this project, I have used Yahoo data source.

I chose the Oracle data and used same data across the models I created, this was so I could compare models across same data. The data was daily trading activities for a twelve (13) years period -2008 to 2020. This dataset is imported (from Yahoo) as Pandas DataFrames, with seven columns [‘High’, ‘Low’, ‘Open’, ‘Close’, ‘Volume’, ‘Adj Close’]. These are financial stock prices for the highest, lowest, opening, closing, number of volumes, and adjacent closing respectively. I chose the ‘Adj Close’ column as my data column, on which the prediction is done, and the target prediction are the next day values of the ‘Adj Close’ column. The Adjacent Close is a bit like the Close column, with some major difference,

> *The closing price is simply the cash value of that specific piece of stock at day's end while the adjusted closing price reflects the closing price of the stock in relation to other stock attributes. In*

general, the adjusted closing price is considered as a more technically accurate reflection of the true value of the stock. (Bischoff, 2019)[4]

Labelling was not needed as this is a Time-Series, the predictions were simply the next day values.

The dataset is imported in pandas dataframe form, and is indexed by datetime, and this is a suited form for the processes that would be carried out. The data is checked for null values, every null value must be dropped or filled with the best suited value, in most general cases, the mean value is used to replace the nulls, however, this is not a good practice for time series data, as this might throw off the model predictions, thus the best practice is to use either the previous day value or the next day value. The correct datatype for the dataset is also checked, for stock prediction the datatype should be float since the figures represent financial values.

All other columns are dropped, leaving just the 'Adj Close' column, and this column is plotted to visualize the form and nature of the data. This same data is used similarly for both the fbprophet and the xgboost algorithms.

The Data pre-processing is a bit different for the two algorithms being used for project.

FbProphet Data Pre-processing:

After data cleaning and pre-processing, the data for fbprophet algorithm is renamed to 'df_fbp' to reflect its representation while containing same data. The fbprophet algorithm requires a two-columns dataframe with column-names ('ds' and 'y'), where 'ds' represents the column for dates, and 'y' represents the column for the data we need to train for forecasting. The algorithm would only accept data in this format, and this must be adhered to for this algorithm.

To get these columns, the dataframe needs only naming of the 'Adj Close' to 'y' to get the first column. Since our data only has one column, we need a second column for 'ds'. This is obtained by getting the date values from the index (since our index are the dates needed). Simply running this code creates a new column of date values from the index, and names the new column as 'ds'.

- *df['ds'] = df.index.date*

XGBoost Data Pre-processing:

I used same data for both algorithms, however renamed them accordingly to reflect which algorithm is being used. Unlike the fbprophet which uses just one column along with a date column, the xgboost algorithm requires a target column since the time series is a type of non-linear regression. For the target column, I used the 'shift()' method of pandas dataframe to obtain values of the next day from the 'Adj Close' column, since that is our target prediction (we are trying to predict next day values), and these values are assigned to a new column. Using the 'shift()' function to create values means there would be a NaN-value at the end of the data sequence, I used pandas forward-fill function (*'fillna(method=ffill)'*) to fill the NaN-value. I must note here that it is always important to check for NaN-values, and I did the checking for nan both for fbprophet and xgboost algorithms.

Here the naming of the columns is of no significant importance since they can take any name unlike in fbprophet where they must follow a particular naming. The dataframe which now has two columns for the data and target values are separated into 'X' and 'y', where 'X' represents the data and 'y' represents the target values. These two sets of values are what the algorithm would be training.

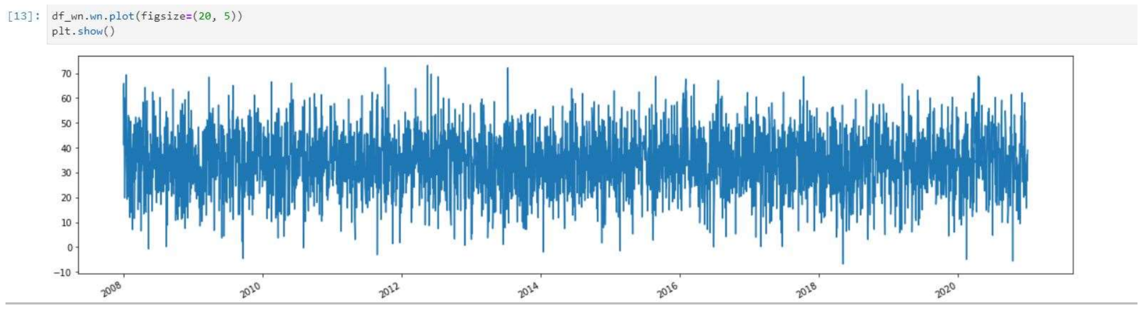
I used the sklearn ‘train_test_split’ function to split the data into groups of training and testing, following best practices I further split the data into a third validation group, thus there is the Train, Test and Validation groups.

- *X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.33)*
- *X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.33)*

Exploratory Visualization

It is important to visualize the data, this is to see what nature the data takes, and be sure that white noise is absent in the data being used.

White Noise
In signal processing, white noise is a random signal having equal intensity at different frequencies, giving it a constant power spectral density. The term is used, with this or similar meanings, in many scientific and technical disciplines, including physics, acoustical engineering, telecommunications, and statistical forecasting. White noise refers to a statistical model for signals and signal sources, rather than to any specific signal. White noise draws its name from white light, although light that appears white generally does not have a flat power spectral density over the visible band. (Wikipedia, 2021) [3]



```
[13]: df_wn.wn.plot(figsize=(20, 5))
      plt.show()
```

Fig. 1. White Noise sample

It is important that the data does not have the nature of a white noise, and this is clearly visible in the data visualisation. Thus, after dropping the unneeded columns, a visualization of our data column is done to see what it looks like. It is also a good practice to make a copy of the original data, and probably rename the data for good referencing.

```
[61]: ##visualizing the data
      df_fbp.plot(figsize=(20,5))
      plt.xlabel("Year")
      plt.ylabel("Adj Close price")
      plt.title("Adj Close")
```

```
[61]: Text(0.5, 1.0, 'Adj Close')
```

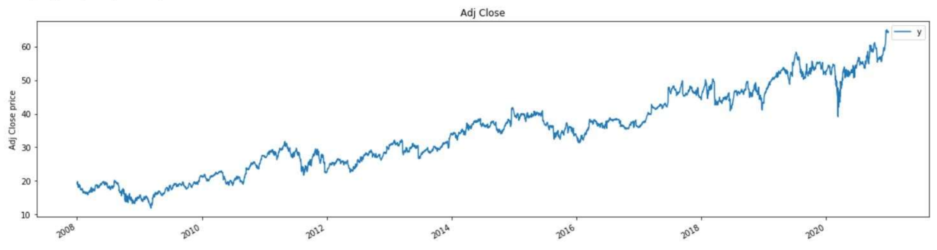


Fig. 2. Here the data has been renamed to suit what it would be used for which in this case is for fbprophet (db_fbp), although it's same data which would be used for both fbprophet and xgboost algorithms.

From this visualization, a steady trend of increase is seen across the dataset along the date range given. There were some years that had lower closings than the previous year, however, generally considering all years, the prices have been in a rising direction.

I mostly used the matplotlib functions for most of the plots, and a few seaborn plots where necessary.

Algorithms and Techniques

I used two different algorithms for my project, this was to see the difference in their approaches and in their outcomes. I used the FbProphet and AWS in-built XGBoost algorithms.

FbProphet: Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

We now have our data dataframe with its columns rightly named and ready for training.

XGBoost: XGBoost is a special type of Gradient Boosting designed to be efficient and scalable. It is an assemble of Decision Trees, where new Trees fix errors of previous Trees already part of the model, and Trees are added until there are no further improvements able to be made to the model. Although this algorithm is mainly for Tabular data, it can also handle Time Series data.

The next step is employing AWS Sagemaker in-built functions for training and prediction. To successfully achieve this, our data is first saved locally, and uploaded to AWS using S3-bucket, and then the data can be accessed from the bucket location. In uploading the data to s3, a specific format has to be followed, the data should not have headers and also the target has to be concatenated with the data, with the target being the first column, this is required by the algorithm provided by Amazon.

AWS Sagemaker uses containers to construct an image for the training, within this container is passed the algorithm which would be used for the training.

SageMaker provides containers for its built-in algorithms and prebuilt Docker images for some of the most common machine learning frameworks. (Amazon, 2021)[5]

The AWS helps a lot with easing the training process. Using the sagemaker estimator function, and setting all necessary parameter, an Estimator is built around the image container created for the xgboost algorithm. These are the parameters needed for an estimator:

- Container: the image name of the training container
- role: the IAM role to use (our current role in this case)
- train_instance_count: the number of instances to use for training
- train_instance_type: the type of instance to use for training
- output_path: where to save the output (the model artifacts)
- sagemaker_session: the current SageMaker session

The AWS Sagemaker platform also provides hyperparameters and hyperparameter tuning option also. I used both to get the best results in training the model. The hyperparameter tuning chooses from multiple hyperparameters settings' options and outputs the best choice.

Hyperparameters:

- max_depth=5,
- eta=0.2,
- gamma=4,
- min_child_weight=6,
- subsample=0.8,
- objective='reg:squarederror',
- early_stopping_rounds=20,
- num_round=500)

Now that we have our estimator object completely set up, it is time to create the hyperparameter tuner. To do this we need to construct a new object which contains each of the parameters we want SageMaker to tune. In this case, we wish to find the best values for the `max_depth`, `eta`, `min_child_weight`, `subsample`, and `gamma` parameters. Note that for each parameter that we want SageMaker to tune we need to specify both the type of the parameter and the range of values that parameter may take on.

In addition, we specify the number of models to construct (`max_jobs`) and the number of those that can be trained in parallel (`max_parallel_jobs`). In the cell below we have chosen to train `20` models, of which we ask that SageMaker train `3` at a time in parallel. Note that this results in a total of `20` training jobs being executed which can take some time, in this case almost a half hour. With more complicated models this can take even longer so be aware!

HyperparameterTuner:

- estimator: the estimator object to use as the basis for the training jobs.
- objective_metric_name: the metric used to compare trained models.
- objective_type: whether we wish to minimize or maximize the metric.
- max_jobs: the total number of models to train
- max_parallel_jobs: the number of models to train in parallel
- hyperparameter_ranges = {
    - 'max_depth': IntegerParameter(3, 12),
    - 'eta'   : ContinuousParameter(0.05, 0.5),
    - 'min_child_weight': IntegerParameter(2, 8),
    - 'subsample': ContinuousParameter(0.5, 0.9),
    - 'gamma': ContinuousParameter(0, 10)}

Benchmark

For a Benchmark Model, I used an xgboost stock prediction model, [XGBoost for stock trend & prices prediction | Kaggle]().

The model uses a time period of years 2010 to 2018, using New York Stock Exchange dataset

My target is based on rmse value of the benchmark xgboost model, the approaches are similar, however I would be using an in-built AWS implemented xgboost algorithm, which has a lot of function already defined.

I used this example because it uses a similar model from the xgboost algorithms, I based my comparism on it due to the algorithm I used. This benchmark xgboost model used a regression approach and gave a final rmse value of 2.8859.

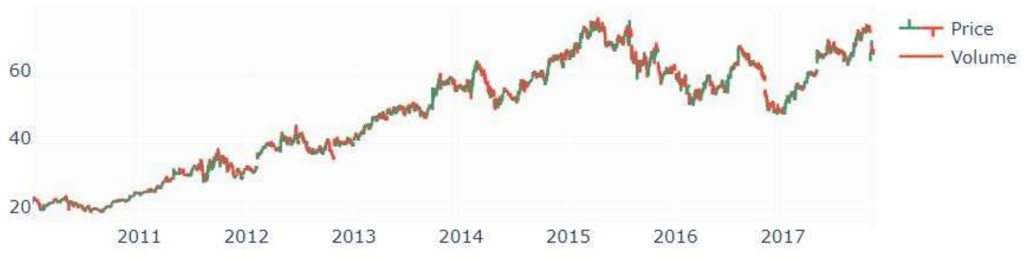


Fig.3.Benchmark Data

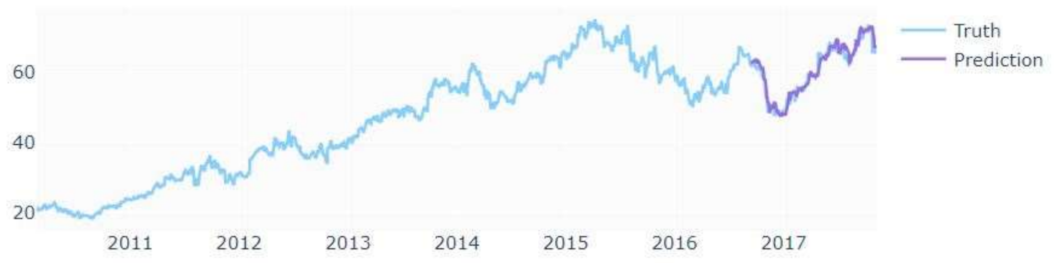


Fig.4.Benchmark Prediction

III. Methodology

Data Pre-processing

The fbprophet is the first algorithm to be applied. Fbprophet is specially for time series forecasting, and the data should be in pandas dataframe format. The fbprophet just takes two columns for its prediction, and these columns must be also named as 'ds' and 'y' for date and data inputs respectively. For this algorithm there is no selection of features, because generally, time series and predictions use just a single series of data to predict the most likely next step.

Xgboost**:** Using the in-built aws xgboost is quite an easy and fast approach. The XGBoost is mainly used for supervised and regression learnings, however, it is also known for its robust performance across all Machine Learning situations, and we would see how it performs in this time series prediction. Also, this needs no features selection, however, it is up to the individual to choose the best possible column to represent what outcome or prediction needed.

Implementation

FbProphet Training

A model is created by calling the 'Prophet' function and assigning this to a model named "model". The fbprophet does not necessarily need the splitting of the dataset into train and test sets, thus the entire dataset is passed into our newly created model –"model" using the '.fit()' method, the model is trained.

FbProphet Testing

After training, predictions are made. To make predictions, a new dataframe is created for future dates using the fbprophet 'make_future_dataframe' method. The number of days required for the prediction is passed into this method. This new dataframe is passed to the our trained "model" using a 'predict()' method. The result is future prediction of the number of days inputted.

Depending on results from predictions made, fbprophet also offers hyperparameter tuning to select the best possible combinations for best prediction results.

XGBoost Training

First is to confirm and ensure that our data is in the csv-format needed by Sagemaker.

- *s3_input_train = sagemaker.TrainingInput(s3_data=train_location, content_type='csv')*

- *s3_input_validation = sagemaker.TrainingInput(s3_data=val_location, content_type='csv')*

This is the training which applies the hyperparameter tuning initially set

- *xgb_hyperparameter_tuner.fit({'train': s3_input_train, 'validation': s3_input_validation})*

After the training, the method 'best_training_job()' is applied to the tuned parameters, which selects the best parameters, this is then applied to the Estimator. This is done since there would be a testing of the performance of our trained model

- *xgb_hyperparameter_tuner.best_training_job()*

- *xgb_attached=sagemaker.estimator.Estimator.attach(xgb_hyperparameter_tuner.best_training_job())*

XGBoost Testing

After obtaining our best performing model, a test is carried out on the test data to see the how well the model performs on new data. For testing, I also used Sagemaker's 'transformer' function. A prediction is carried out on the Test data, and then downloaded to local storage for viewing and analysis. This is achieved by building a transformer object from the fit xgb model. Then next is to use sagemaker to run batch transform jobs on test data stored in the s3 bucket

- *xgb_transformer = xgb.transformer(instance_count = 1, instance_type = 'ml.m4.xlarge')*

- *xgb_transformer.transform(test_location, content_type='text/csv', split_type='Line')*

Refinement

For both algorithms, I used hyperparameter tuning to obtain best performing models. I have stated earlier the codes and process of the XGBoost algorithm.

The FbProphet algorithm is a bit similar but with different code, which might appear easier and simpler than the AWS Sagemaker hyperparameter tuning. Fbprophet uses cross-validation approach as a metric to analyse and compare results. The aim o cross-validation is to test the trained model's ability in predicting test data. In its hyperparameter tuning, parameters are evaluated on rmse average over a 30-days horizon. There several input parameters that can be tuned for better results, some of these parameters are:

- changepoint_prior_scale: It determines the flexibility of the trend

- seasonality_prior_scale: This parameter controls the flexibility of the seasonality

- holidays_prior_scale: This controls flexibility to fit holiday effects

- seasonality_mode: For seasonal effects

There is a further tuning with the best parameters for improved results. This was achieved by iterated training for different values of the parameters '*changepoint_prior_scale*' and '*seasonality_prior_scale:*', and outputting the values that produced the best results, then using these values in a new model (now "model2") training.

IV. Results

Model Evaluation and Validation

In total I ended with four models, two (“*model*” and “*model2*”) for the fbprophet algorithm and two (“*xgb*” and “*xgb_hyperparameter_tuner*”) for the xgboost. There were differences in the results from these models. I made final comparisms using their rmse and mse values.

FbProphet Results

The results (the RMSE values) of “model” and “model2” are compared. There were little but significant difference between these two models. After a period of 365-days, “model” had an MSE of 72.116952 and RMSE of 8.492170, while “model2” had an MSE of 46.822840, and RMSE of 6.842722. The difference becomes more significant when comparing their MSE values as the number of days of prediction increases.

Viewing these models’ prediction results graphically, not a lot of difference can be seen in these two results.

	horizon	mse	rmse	mae	mape	mdape	coverage
324	361	46,41055	6,812529	5,794448	0,169128	0,139219	0,22362
325	362	46,04867	6,785917	5,767908	0,168232	0,138166	0,224567
326	363	46,11318	6,790669	5,771364	0,168204	0,138166	0,225785
327	364	46,51611	6,820272	5,800023	0,168774	0,138864	0,22362
328	365	46,82284	6,842722	5,823237	0,168802	0,138166	0,222465

Table 2. Results table of model, displaying mse and rmse values for training with hyperparameter tuning

	horizon	mse	rmse	mae	mape	mdape	coverage
324	361	70,62987	8,404158	6,373881	0,190357	0,127638	0,597538
325	362	71,20876	8,438528	6,391263	0,191118	0,126954	0,599838
326	363	71,34708	8,44672	6,385816	0,190835	0,126954	0,604708
327	364	71,72741	8,469203	6,409529	0,190989	0,127638	0,60579
328	365	72,11695	8,49217	6,440106	0,191018	0,126954	0,603438

Table 3. Results table of model, displaying mse and rmse values for training without hyperparameter tuning

XGBoost Results

The models “*xgb*” and “*xgb_hyperparameter_tuner*” were also compared by their RMSE values.

Comparing these two models with the benchmark model, I can say the models did quite very high a significant high prediction success. My model “*xgb*” without hyperparameter tuning had an output MSE of 0.619825572224999, and an RMSE of 0.3841837399840474. The model “*xgb_hyperparameter_tuner*” which is a hyperparameter-tuned model had an output of MSE 0.6091210861546326 and RMSE value of 0.37102849759819934. The benchmark model had a mean squared error (MSE) of 2.885670142099009.

Also, like the fbprophet algorithm, when the graphs of both model with hyperparameter tuning and model without hyperparameter tuning are plotted, there is no significantly noticeable difference in both graphs.

```
[92]: Text(0.5, 1.0, 'Closing prices and Future predicted prices')
```

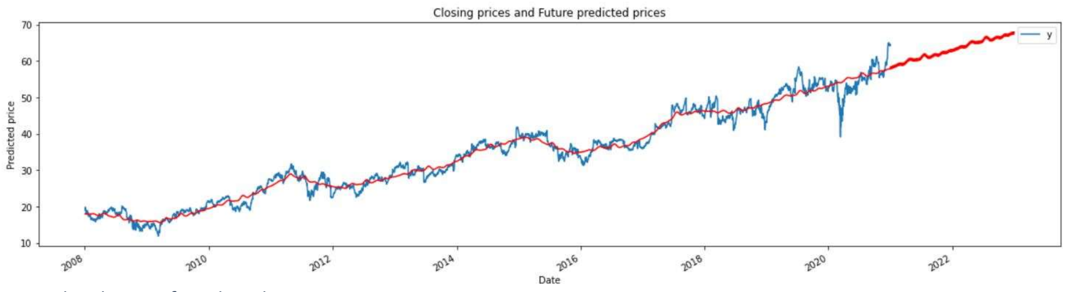


Fig. 5. Graphical view of predicted prices

```
[91]: Text(0.5, 1.0, 'Closing prices and Future predicted prices with parameter tuning')
```

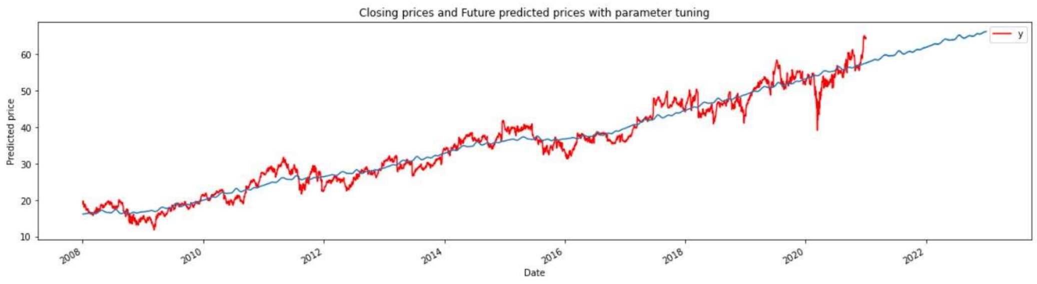


Fig. 6. Graphical view of predicted prices with parameter tuning

However, if we plot a closer sample of these graphs, we can see the difference in their waveforms, with “model2” of the fbprophet having a lower amplitude of variation than “model”.

```
[85]: Text(0.5, 1.0, 'Predicted prices without hyperparameter tuning')
```

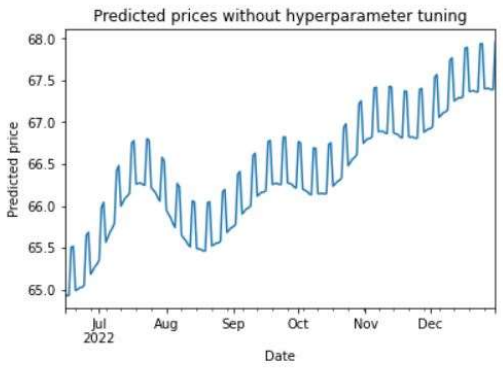


Fig. 7. Predicted prices without hyperparameter tuning

```
[84]: Text(0.5, 1.0, 'Predicted prices with hyperparameter tuning')
```

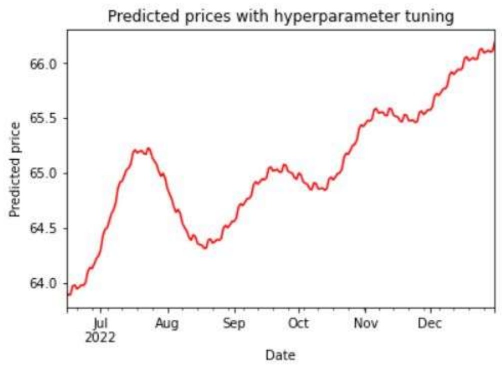


Fig. 8. Predicted prices with hyperparameter tuning

Justification

Considering all models trained and used for prediction, the xgboost models gave better and more accurate precisions, since they have the lowest error values. The error values of the xgboost algorithm with and without hyperparameter tuning is much better than the values of the benchmark model and better than the values from the fbprophet algorithm. However, the fbprophet is easier to implement, and has a lot of metrics to view many sides to the data and results generated.

The fbprophet algorithm results also show that there is a rapid rise in the error values as we proceed more in number of days predicted.

In a lot of cases, reports show that a higher amount of data usually leads to a better prediction, however, I can’t say for sure if the amount of data I used led to a better error rate than the benchmark; I used a period of 13yrs, while the benchmark used a period of 8 years.

model	**rmse**	**mse**
xgb	0.3841837399840474	0.61982
xgb_hyperparameter_tuner	0.37102849759819934	0.60912
Benchmark	1,7	2,89
fbp	72,11695	8,49217
fbp with hyperparam tuning	46,82284	6,842722

Table 4. RMSE and MSE results

V. Conclusion

Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

- *Have you visualized a relevant or important quality about the problem, dataset, input data, or results?*
- *Is the visualization thoroughly analyzed and discussed?*
- *If a plot is provided, are the axes, title, and datum clearly defined?*

Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

- *Have you thoroughly summarized the entire process you used for this project?*
- *Were there any interesting aspects of the project?*
- *Were there any difficult aspects of the project?*
- *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

Improvement

Though I used the xgboost algorithm, I believe that recently, a lot of algorithms do a great job at time series prediction given it’s robust and sporadic nature. Deep Learning and Neural Networks might be very good at such task; however, this is for me to study further and find out.

References

[1]Alasdair Tran, A. M. (2021). *Radflow: A Recurrent, Aggregated, and Decomposable Model.* Australia: Creative Commons Attribution 4.0 International.

[4]Bischoff, B. (2019, March 31). *adjusted-closing-price-vs-closing-price*. From finance.zacks.com: https://finance.zacks.com/adjusted-closing-price-vs-closing-price-9991.html

[2]Pengfei Yu, X. Y. (2019). *Neural Computing and Applications.* Hubei.

[3]Wikipedia. (2021, Feb 5). *Wikipedia*. From wikipedia.org: https://en.wikipedia.org/wiki/White_noise

[5]Amazon. (2021, Feb 1). *Amazon Web Services*. From docs.aws.amazon.com: https://docs.aws.amazon.com/sagemaker/latest/dg/docker-containers.html