



Stock Price Predictor with SageMaker DeepAR

Machine Learning Engineer Nanodegree
Capstone Project

Shamus Clifford
February 2020

I Definition

Project Overview

The stock market is a place where you can buy, sell, and trade stocks. A stock is a share of a company. So, buying a stock in a company is buying part of that company. If you buy a stock and the price of the company goes up, you will make money. If the price of that same company goes down, you will lose money. Being able to predict whether the price of a stock will rise or fall would be lucrative to any investor.(Amadeo 2019)

The **Random Walk Hypothesis** is a financial theory that states stock market prices are random, that the changes in price are unpredictable.(Smith 2019) The Random Walk Hypothesis is consistent with the **Efficient Market Hypothesis** (EMH). EMH is an investment theory that states that the price of a stock reflects all information made available to market participants at any given time.(Van Bergen 2012) Based on Random Walk and EMH, it would be impossible for an investor to predict future prices.

I disagree with these theories. I believe an investor, armed with machine learning models, can predict the price of a stock better than the market can. Technical analysis or “identifying trading opportunities by analyzing statistical trends gathered from trading activity,” can help an investor predict future prices.(Hayes 2019) Machine learning models can aid in technical analysis.

Problem Statement

I plan on proving that EMH is false, that the market is not efficient and an investor can predict the price of a stock, meaning a stock's future price is not totally random as the Random Walk Hypothesis suggests. I will analyze a market sectors price movements over a 5 year span to predict the stock price in the future (1 day, 7 days, 28 days). Success will be measured against the Naive Method. The Naive Method “forecasts the next values using the last value observed.”(Ramírez 2017) Using the Naive Method, the price of the stock on January 3 would be the same price it was at the end of January 2. This method works well for a random walk forecast.(Hyndman & Athanasopoulos 2018) I will test this against the AWS DeepAR Forecasting algorithm which is a “supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN).”(Mishra 2019) Using DeepAR, I will predict stock prices better than the Naive Method.

Metrics

The evaluation metric will be Mean Absolute Percentage Error (MAPE). I will compare the MAPE for the Naive Method and compare it to the DeepAR model. The method with a smaller MAPE did a better job at predicting the future stock price.(Halimawan & Sukarno 2013) MAPE is a statistical measure of how accurate a forecast system is which makes it a great metric for the accuracy of a stock price predictor.

$$MAPE = \frac{1}{n} \sum_{t=1}^n \left| \frac{A_t - F_t}{A_t} \right|$$

The benchmark model will be the Naive Method. The Naive Method sets all forecasts to be the value of the last observation (the previous days adjusted close price). This method works well for random walk forecasts.(Hyndman & Athanasopoulos 2018)

II Analysis

Data Exploration

For this project, I used stocks from the Information Technology Sector. DeepAR performs better when stocks belong to a cluster.(Das 2018). The list of stocks I intended to use were:

- [Apple \(AAPL\)](#)
- [Microsoft \(MSFT\)](#)
- [Intel \(INTC\)](#)
- [Cisco \(CSCO\)](#)
- [Adobe \(ADBE\)](#)
- [Salesforce \(CRM\)](#)
- [NVIDIA \(NVDA\)](#)
- [Accenture \(ACN\)](#)
- [PayPal \(PYPL\)](#)
- [Oracle \(ORCL\)](#)

These stocks were chosen from the [Vanguard Information Technology ETF](#). I chose the top ten weighted stocks in the fund (ignoring Visa and Mastercard which I do not believe fit perfectly in the Information Technology sector). Historical data for these stocks comes from Yahoo Finance. I will use the daily adjusted close price of each stock over a 5 year span (January 1, 2014 - December 31, 2018) as training data and 1 year of test data (January 1, 2019 - December 31,

2019). I chose a 5 year span because I wanted DeepAR to detect any trends in seasonality of the stocks.

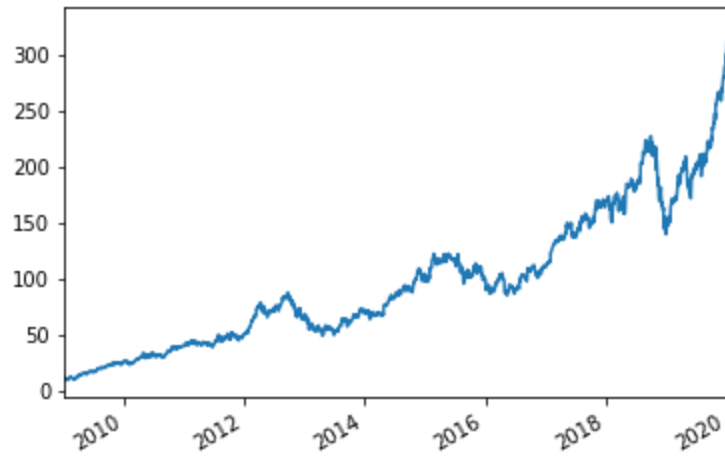
After several attempts to create an accurate DeepAR model, I decided to increase the timespan from 5 years to 10 years in order to help DeepAR detect seasonality trends. I also adjusted the list of stocks to ensure that they all had 10 years worth of data. This forced me to drop PayPal which does not have 10 years of historical data. In addition, I added 11 new stocks to the stocks that I looked at. DeepAR performs better with more related time series. Finally, I added the Volatility Index as a dynamic feature to help DeepAR perform better. The [Volatility Index \(VIX\)](#) measures how volatile the market is as a whole. Higher volatility would generally lead to greater changes in prices. The list of stocks I added are:

- [IBM \(IBM\)](#)
- [Texas Instruments \(TXN\)](#)
- [Qualcomm \(QCOM\)](#)
- [Automatic Data Processing \(ADP\)](#)
- [Intuit \(INTU\)](#)
- [Micron Technology \(MU\)](#)
- [Advanced Micro Devices \(AMD\)](#)
- [Auto Desk \(ADSK\)](#)
- [HP \(HPQ\)](#)
- [Amazon \(AMZN\)](#)
- [Google \(GOOG\)](#)

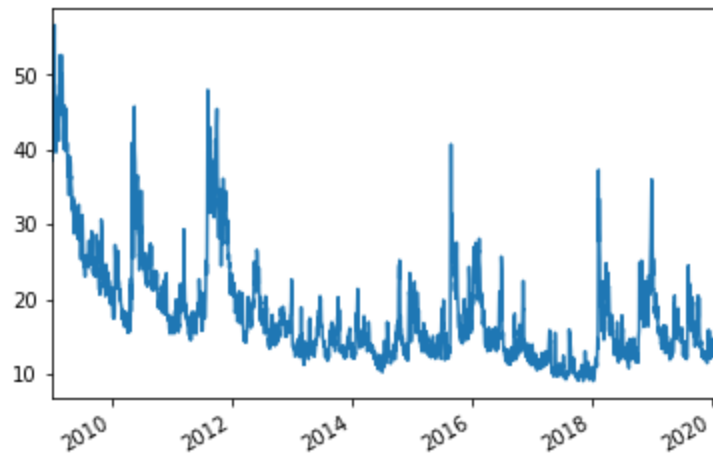
This data was gathered from [Yahoo Finance](#) using [RapidAPI](#) for data access. You must subscribe to the [Yahoo Finance API](#) which does require a credit card. The API comes in a Freemium model where you get 500 monthly API calls for free.

Exploratory Visualization

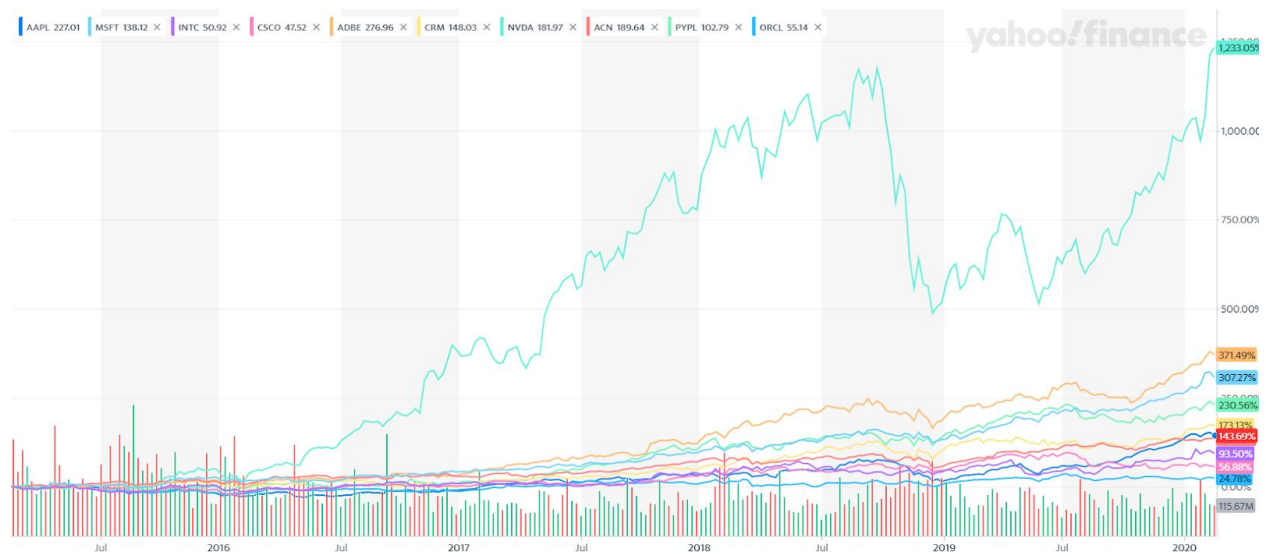
Fortunately, financial data is well documented and visualized already. You can go on Yahoo Finance and create beautiful charts quite simply. So, the data visualization part of this project for me was done as a sanity check. I was making sure that my time series was formatted correctly. This was done by simply plotting the time series. For example, here is the chart for Apple (AAPL)



And here is the chart for the Volatility Index (^VIX)



Here is an example of a chart I made on [Yahoo Finance](#). This chart is great because you can visualize all the stocks and see their percentage return on the right side over a 5-year span.



The important thing from this chart is that all these stocks seemed to increase in price over the time span and at a similar rate, aside from NVDA (1,233.05% return over 5 years!!!). DeepAR works best with time series that are similar. These time series are. Each stock tends to move with the others, they generally increase or decrease in price at the same time.

Algorithms and Techniques

The proposed solution to the problem is to apply DeepAR forecasting which is successful when forecasting related time series.

To do this, we will pull data from the Yahoo Finance API, create an individual time series for each stock in the dataset and convert that dataset to a JSON file for DeepAR to train on. The time series data will include the date and adjusted close price every day in the time frame.

The data fed to the DeepAR model must be formatted as a JSON file. It must have a start date, which is indicated by the first day in the time series. It must have a time series of instances which is a list of adjusted close prices from the start date until the end of the training or test period. Finally, the dynamic features are added. The dynamic feature I used was the Volatility Index. The dynamic feature spanned the entire time frame for both the train and test periods.

Benchmark

The benchmark model will be the Naive Method. The Naive Method sets all forecasts to be the value of the last observation (the previous days adjusted close price). This method works well for random walk forecasts.(Hyndman & Athanasopoulos 2018)

The results of the benchmark model are shown in the table below. They describe the MAPE of each stock for each time frame.

	1-day	5-day	10-day	20-day	30-day
AAPL	1.583156	3.897945	4.719671	5.292743	6.133637
MSFT	1.567856	2.595607	3.753417	5.610161	8.773028
INTC	0.067835	0.531528	1.375553	5.933489	8.047588
CSCO	0.063129	0.562921	1.610361	1.584302	1.818353
ADBE	1.325948	2.418356	3.253070	4.715763	6.825946
CRM	0.750015	2.543084	3.128785	3.696259	4.874571
NVDA	0.187209	2.202002	2.797383	2.710498	6.369267
ACN	0.195766	1.519735	2.113438	2.454919	3.220082
IBM	0.827720	1.497187	2.530976	4.322330	7.276478
ORCL	0.387953	1.033144	1.543485	1.470943	1.798118
TXN	0.269728	0.713068	1.012157	1.983948	2.120542
QCOM	0.293092	1.344411	2.800502	2.300370	2.034214
ADP	0.931001	1.744926	3.053382	3.616114	4.576559
INTU	2.349019	3.020656	4.266938	5.714441	7.573863
MU	1.303894	1.706716	1.706113	3.243080	2.722242
AMD	0.878106	0.724618	2.422516	2.652458	5.404541
ADSK	1.289812	2.118122	2.590943	4.286728	6.105720
HPQ	1.290017	2.914830	4.045674	4.238825	5.149035
AMZN	0.787011	1.035106	1.319676	2.598741	5.291595
GOOG	0.781872	2.200980	3.645856	3.999907	5.241635

As you can see, the Naive Method is pretty accurate when forecasting 1-day into the future but gets less accurate as the time frame increases.

III Methodology

Data Preprocessing

Once the data was received from Yahoo Finance, the data needed to be preprocessed. I did this in a few steps. First, I had each stock's data put in a list. The data came in as a string which needed to be parsed. I parsed each item in the list and pulled out the price data.

```
price_data = [json.loads(d.text)['prices'] for d in data]
```

Then, I put the data in a Pandas DataFrame. The data was in chronological order which needed to be reversed. The dates came in as timestamps which I formatted to be as a datetime object. The resulting DataFrame was a time series for each stock. Each DataFrame was put in a list for easy access.

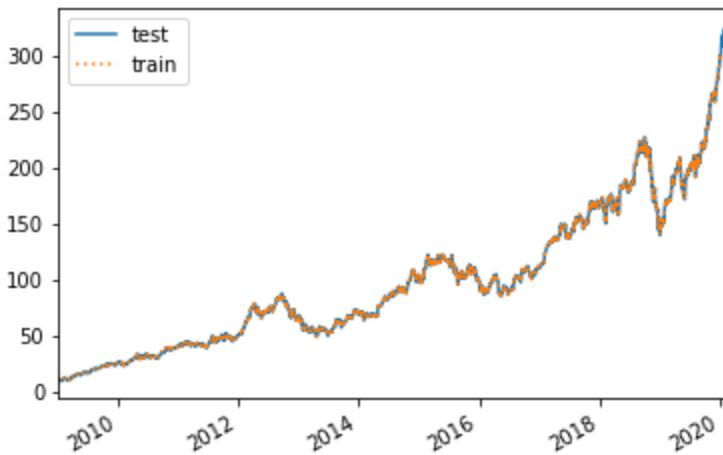
```
time_series = []
for stock in price_data:
    data = []
    index = []
    for day in stock:
        if 'adjclose' in day:
            data.append(day['adjclose'])
            index.append(datetime.fromtimestamp(day['date']))
    data.reverse()
    index.reverse()
    time_series.append(pd.Series(data=data, index=index))
    print(len(data))
```

The only data I needed from Yahoo Finance was the date and adjusted close price for each day in the time frame. In the end, I had 2,802 days in each time series and 21 time series (including VIX), so a total of 58,842 data points.

Once the data was formatted, I split the data into a training and test set. The training set used 10 years of data minus the last 30 days. The test set used the full data set. The prediction length I chose for DeepAR was 30 days so I took out the last 30 days in our training set. I kept the 30 days in the test set so we can evaluate the model on the test set of data. Since the data was in a list of DataFrames, it was quite simple to create the training set.

```
prediction_length = 30
time_series_training = []
for ts in time_series:
    time_series_training.append(ts[:-prediction_length])
```


As a sanity check, I created a chart to compare the train and test data. As expected, there was a small period of time at the end of the chart where there was test data but no training data.



Finally, the data was converted to JSON format. DeepAR expects to see data in JSON format. The data needs to come from a JSON file S3. DeepAR also accepts two data channels: train and test. Naturally, the training dataset will be described on the train channel and the optional test dataset will be described on the test channel. More information about the Input/Output Interface can be found in the [Developer Guide](#). This data was saved locally and uploaded to an S3 bucket.

Implementation

Once the data was preprocessed, training a DeepAR model is straightforward. You create an estimator, specify hyperparameters, and create a training job. You could also create a hyperparameter tuning job with DeepAR. Rather than describe each hyperparameter, I will reference the [DeepAR Hyperparameters](#) and [Model Tuning](#) documentation.

```
s3_output_path = f's3://{bucket}/{prefix}/output'

estimator = Estimator(sagemaker_session=sagemaker_session,
                      image_name=image_name,
                      role=role,
                      train_instance_count=1,
                      train_instance_type='ml.c4.xlarge',
                      output_path=s3_output_path
                      )

hyperparameters = {
    'time_freq': 'D',
    'epochs': '30',
    'prediction_length': str(prediction_length),
```

```

    'context_length': str(prediction_length),
    'num_cells': '40',
    'num_layers': '4',
    'likelihood': 'gaussian',
    'learning_rate': '0.001',
    'early_stopping_patience': '10',
    'dropout_rate': '0.117'
}

estimator.set_hyperparameters(**hyperparameters)

data_channels = {
    'train': train_path,
    'test': test_path
}

estimator.fit(inputs=data_channels)

```

The final implementation of my model is shown above. One important hyperparameter here is the `prediction_length`. This is the length of time that the model will predict. I specified a daily frequency so the model will predict daily values in the future. Earlier, I had specified a prediction length of 30, so this model will produce 30 days of price predictions. Epochs, `dropout_rate`, and `num_layers` were all generated from a tuning job from a previous attempt at creating a successful model. I will describe this further in the [Refinement](#) section.

After the model was created, I created a predictor. To use a DeepAR predictor, data must be passed in and the output must be specified. I used a helper function `json_prediction_input` to accomplish formatting the data to pass into the predictor and specifying the output configuration.

```

def json_predictor_input(input_ts, num_samples=30, quantiles=['0.1',
'0.5', '0.9']):

    instances = []

    for k in range(len(input_ts)):
        instances.append(series_to_obj(input_ts[k], vix_time_series))

    configuration = {'num_samples': num_samples,
                    'output_types': ['quantiles'],
                    'quantiles': quantiles
                    }

    request_data = {'instances': instances,
                   'configuration': configuration
                   }

    json_request = json.dumps(request_data).encode('utf-8')

```

```
return json_request
```

The data passed to the model needed to be in JSON format. The request format is specified in the [Developer Guide](#). After the input was generated, I passed it to the predictor to get a prediction based on my model. At that point the response data was decoded and analyzed. More information about the analysis of the results can be found in the [Results](#) section.

Refinement

I faced many challenges and refined my techniques over and over. So much so that I included my attempts on GitHub. I have 3 notebooks in my [other-attempts](#) folder of my GitHub Repository.

My [first attempt](#) was totally mismanaged. I was using a single DataFrame to store all the stock data which was extremely difficult to feed to DeepAR and difficult to read.

My [second attempt](#) saw cleaner code with much more data visualization. In the end, this model did not beat the Naive Method. It actually did quite poorly. In only a few instances did the model produce more accurate predictions than the Naive Method in predicting stock prices.

My [third attempt](#) saw cleaner code (with almost no documentation), 10 years of data, VIX added as a dynamic feature, and hyperparameter tuning. The result: a slightly better performance than attempt 2 but still pretty inaccurate when compared to the Naive Method. However, this model performed better as the time frame expanded. On average, it was more accurate at the 30-day time frame than the Naive Method.

After doing some research after, I realized that I needed to feed my model more time series. With my third attempt I attempted to do this but all I did was feed my model more data. DeepAR performs best when there are many related time series. Ten was not enough. DeepAR did not start to outperform other models until hundreds of related time series were used. Now, I am not attempting to beat the other models here, I am just attempting to beat the Naive Method. So, I doubled the amount of stocks therefore doubling the amount of time series sent to the DeepAR model for training and testing. Again, I saw increased performance in my model, especially as the time frame expanded.

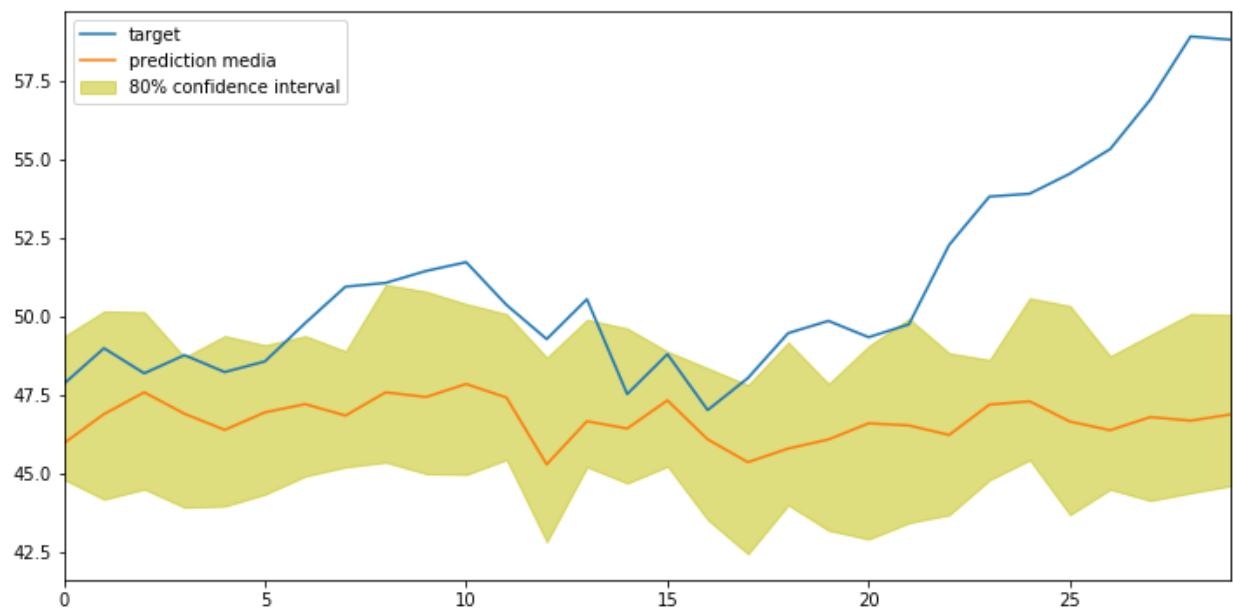
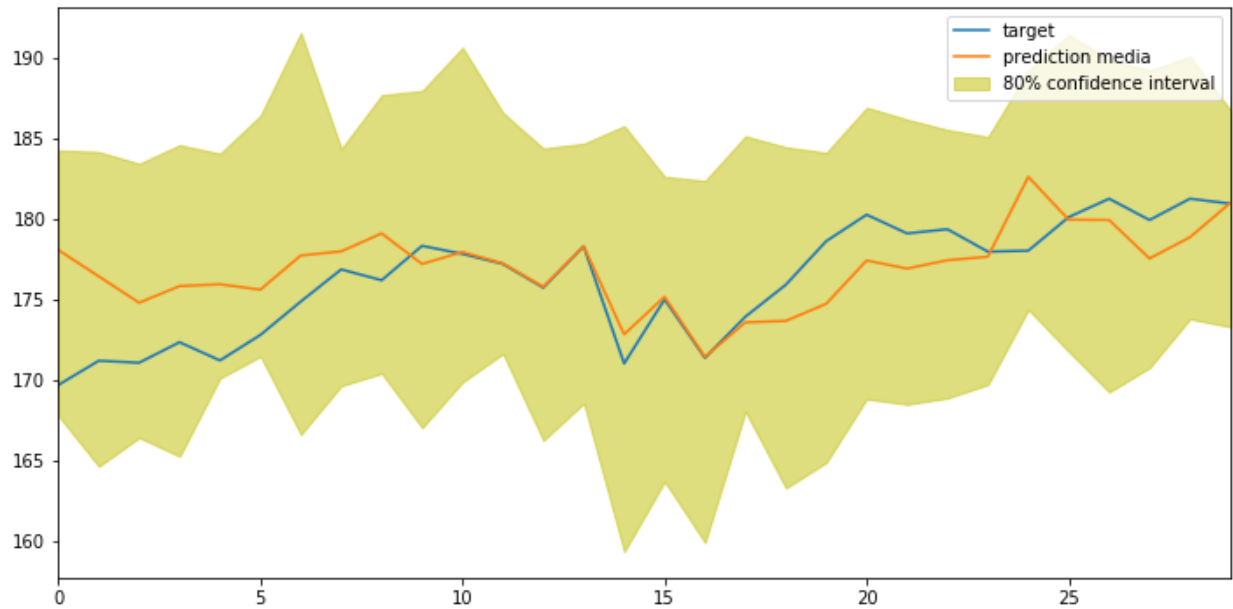
IV Results

Model Evaluation and Validation

To evaluate my results, I used a variety of methods. The first method was a chart of the actual stock price, my model's prediction, and my model's 80% confidence interval of the stock price. I have two examples here. One where my model did a good job predicting the stock price and one where my model did a bad job.

This first chart shows the predictions for Automatic Data Processing (ADP). I like this chart because it shows a few things. First, how accurate the model was in analyzing the trends in the market. The ups and downs of this stock were tracked pretty well. It also shows how far off my model was at predicting the price on the first day. In the end, the 30-Day MAPE for ADP was 1.204417%. This is much better than the 4.927399% from the 1-day forecast.

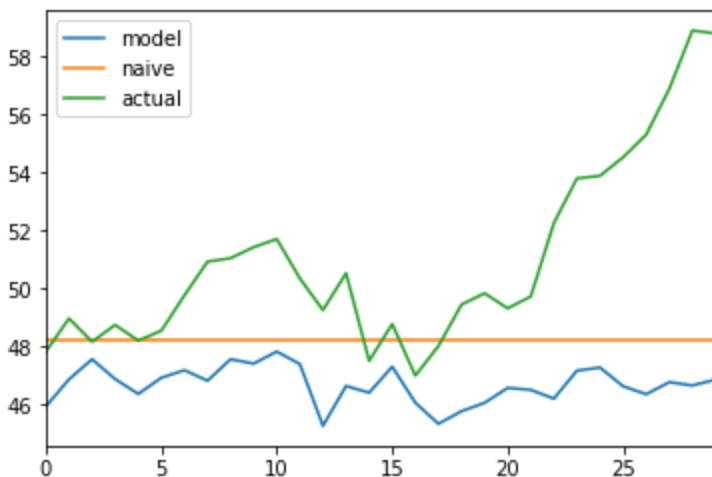
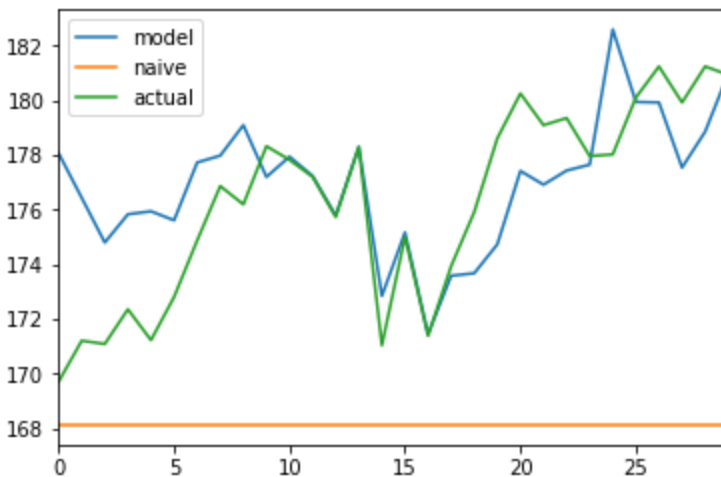
The second chart shows Advanced Micro Devices (AMD). Again, my model did a bad job forecasting the first day. AMD was off by 3.951908% on day one. From there, you can see that my model did a fine job tracking the trends but was still off by a bit, mainly because of the first day. Then, around day 20, the stock began to rise at a much faster rate than many of the other stocks in this project. This seemed to happen based on good news for AMD that it [gained market share in personal computers and servers](#). This goes to show another weakness in my model, it does not take current events into effect. However, because of the news, the Naive Method did a poor job of forecasting as well. In the end, the Naive method was more accurate than my model by 2.700269% at the 30-day time frame.



There was another chart I created that did a good job of visualizing how my model was doing. I will use the same two stocks, ADP and AMD. Each chart plots the actual stock price, the model's prediction, and the naive prediction. You can see the naive prediction is just a horizontal line because it uses the last known stock price as its prediction.

The first chart shows ADP. You can see how well the model tracks the actual stock price. It is very close to the actual stock price most of the time. In comparison, the Naive Method is very low the entire time. My model correctly predicts the increase in ADP stock price.

In contrast, the second chart shows AMD. My model did not correctly predict the increase in stock price for AMD. It predicted a slight drop in price, however AMD's stock price rose. This was in part due to the good earning's report and increased market share mentioned earlier. If there is a silver lining, you can see some of the trends picked up by my model. For example, from day 5 to 10, my model creates a similar shape and around day 15 my model correctly predicts a quick price increase and sharp decrease.



Justification

Even though charts are great for visualizing how my model performed, the metric I chose for success was Mean Absolute Percentage Error. I summarized MAPE a few ways. First, I found the 1-day, 5-day, 10-day, 20-day, and 30-day MAPE for each stock for both my model and the Naive Method. Then, I found the difference between the two MAPEs. The table below shows the

difference between the MAPEs. A positive MAPE means the Naive Method outperformed the model, while a negative MAPE means my model outperformed the Naive Method.

	1-day	5-day	10-day	20-day	30-day
AAPL	2.163023	1.312661	0.854441	1.497263	1.390035
MSFT	-1.501317	-2.269725	-3.174901	-2.862990	-3.059665
INTC	4.701390	4.183480	2.658724	-0.487917	-2.125374
CSCO	0.972403	1.342056	-0.185055	-0.175312	0.187428
ADBE	-0.853873	-0.467403	-1.266698	-0.654919	-0.843566
CRM	3.493826	2.983137	2.632389	3.141278	2.933124
NVDA	2.671843	1.390822	0.899525	1.816363	1.602795
ACN	3.781085	1.624476	1.331218	0.392363	-1.028817
IBM	0.312681	-0.644690	-1.482563	-1.211309	-1.126729
ORCL	1.603625	1.666676	1.704256	2.104487	1.534530
TXN	-0.040412	0.210533	0.142695	-0.091764	-0.404179
QCOM	1.499519	0.029328	-1.251616	-0.259822	-0.013853
ADP	3.996398	1.240229	-0.944247	-2.313961	-3.372142
INTU	-1.219941	-1.980484	-2.144710	-1.843901	-2.302870
MU	3.081104	1.666073	1.268988	-1.034411	0.454259
AMD	3.073802	2.703362	2.413629	2.618811	2.700269
ADSK	-0.487470	-0.997066	-1.415801	-1.029779	-1.252664
HPQ	0.799318	-1.368393	-2.710408	-2.554618	-2.933682
AMZN	-0.522736	0.214287	-0.284967	0.443845	1.041446
GOOG	2.200146	1.469426	0.979869	1.460053	1.303307

The comparison table has a lot of numbers in it which makes it difficult to read. This table is good for seeing how well the model performed against the Naive Method. You can see which stocks did a decent job (ADP) and which did not (AMD).

Based on this chart, I summarized the data. I wanted to see which method did better and at what time frames. I did not care about each individual stock. First, I found the sum of the compared

MAPEs at each time frame and printed them. I also found the mean of the compared MAPEs and printed them. This confirmed my analysis that my model did a bad job at predicting short term prices but a good job at predicting longer term prices (30-day). For example, the mean of the 1-day compared MAPEs was 1.486221 and the mean of the 30-day compared MAPEs was -0.265818. So my model was more accurate in the long term than the Naive Method.

V Conclusion

Reflection

This project developed a DeepAR model to forecast stock prices of information technology companies. In order to create this model I had to generate the example data. Generating the example data involved fetching the data from Yahoo Finance, cleaning the data, and preparing the data to send to DeepAR. After the data was generated, I trained the model. To train the model I defined some hyperparameters to create an accurate model. After I had a model, I evaluated it by creating a predictor and comparing my model's predictions against the Naive Method of stock price forecasting. Finally, I deployed my model, created a lambda function, and used API Gateway to create a web app to interact with my model.

For me, cleaning the data was one of the more difficult parts. I could not decide what the best way to store the data was. I eventually decided on making a list of DataFrames for each stock in my dataset. The DataFrame a series of dates and the adjusted close price of the stock on that date. Before settling on this solution, I tried putting the entire dataset into a single DataFrame which turned out to be difficult to manage and made my code unclear.

Another difficult part for me was defining my hyperparameters. I did not know which hyperparameters I should change in order to increase the accuracy of my model. In my third attempt at creating a model, I decided to create a hyperparameter tuning job to determine the best hyperparameters for me. Then, I transferred those hyperparameters over to my final attempt at developing my model.

Improvement

There are many areas where I could improve my model. First, I would add more stocks into my dataset. DeepAR performs best when there are many related time series. Adding more stocks would increase the number of time series and hopefully improve the model. Second, I would add more dynamic features to my model. For example, I would incorporate the 30-day moving average of a stock as a dynamic feature. Adding the moving average may help the DeepAR

model identify when the stock price is above or below average, and help forecast with that in mind.

References

- Amadeo, Kimberly. “Before You Invest in the Stock Market, Make Sure You Know What It Is.” The Balance, The Balance, 25 July 2019, www.thebalance.com/what-is-the-stock-market-how-it-works-3305893.
- Das, Binoy. “Aws-Samples/Amazon-Sagemaker-Stock-Prediction.” GitHub, 20 Nov. 2018, github.com/aws-samples/amazon-sagemaker-stock-prediction/blob/master/notebooks/dbg-deepar.ipynb.
- Halimawan, Alam Akbar, and Subiakto Sukarno. “STOCK PRICE FORECASTING ACCURACY ANALYSIS USING MEAN ABSOLUT DEVIATION (MAD) AND MEAN ABSOLUTE PERCENTAGE ERROR (MAPE) ON SMOOTHING MOVING AVERAGE AND EXPONENTIAL MOVING AVERAGE INDIKATOR.” The Indonesian Journal of Business Administration, vol. 2, 13 Nov. 2013, pp. 1613–1623., <https://media.neliti.com/media/publications/68283-EN-stock-price-forecasting-accuracy-analysi.pdf>.
- Hayes, Adam. “Technical Analysis Definition.” Investopedia, Investopedia, 18 Aug. 2019, www.investopedia.com/terms/t/technicalanalysis.asp.
- Hyndman, Rob J, and George Athanasopoulos. “Some Simple Forecasting Methods.” Forecasting: Principles and Practice, 15 Jan. 2020, otexts.com/fpp2/simple-methods.html.
- Mishra, Abhishek. “Machine Learning in the AWS Cloud: Add Intelligence to Applications with Amazon SageMaker and Amazon Rekognition.” Amazon, John Wiley & Sons, Inc., 2019, docs.aws.amazon.com/sagemaker/latest/dg/deepar.html.
- Ramírez, Rubén Guerrero. Different Methods for Forecasting Time Series Tutorial, 2 Dec. 2017, rstudio-pubs-static.s3.amazonaws.com/336602_0a55f2341eab4637b27766f6619af477.html.
- Smith, Tim. “Random Walk Theory.” Investopedia, Investopedia, 18 Nov. 2019, www.investopedia.com/terms/r/randomwalktheory.asp.
- Van Bergen, Jason. “Efficient Market Hypothesis: Is The Stock Market Efficient?” Forbes, Forbes Magazine, 1 May 2012, www.forbes.com/sites/investopedia/2011/01/12/efficient-market-hypothesis-is-the-stock-market-efficient/.