Udacity Machine Learning Nanodegree Capstone Project: Stock Prediction with SageMaker's DeepAR Algorithm

Date: 09-23-20

Author: John Farrell

BACKGROUND - STOCK PREDICTION

Prediction of stock trends has long been of interest to investment and trading firms around the world. Entire professions are dedicated to analyzing stock trends and valuations in order to predict which direction that stock will move in the next day, month, or year(s).

On a personal level, I have always been interested in how the investment world works. My background is in chemical engineering and materials science, but I've always been a "numbers guy". I've even successfully taken and completed the Level I Exam towards becoming a Chartered Financial Analyst (CFA), the highest certification someone in the investment industry can have. For me, this project is an opportunity to utilize my new skills in machine learning towards a problem that I've always been very interested in solving.

As we all know, stock prediction is difficult. Like any challenging problem, there are a number of different ways to go about trying to solve this. There are three main avenues used to traditionally value a company (stock): discounted cash flow analysis (DCF), comparable company analysis, and precedent transaction analysis[1]. The exact definitions of each of these analysis techniques is not the purpose of this project, so I'll only provide a very brief summary of each method.

DCF utilizes forecasted unlevered free cash flows of the business discounted back to today's value using the firm's weighted average cost of capital. Additionally, a form of discounted cash flow, discount dividends, can be used as well. Comparable company analysis is a relative valuation method that uses trading multiples (Price/Earnings, Price/Book Value, Price/Sales, EV/EBITDA, etc.) to compare a business with other similar businesses. Precedent transaction analysis compares a business to other similar businesses that have been recently sold, merged, or acquired.

In addition to the fundamental valuation techniques described, a number of analysts will use technical analysis, or the study of historical market data, to value equities[2]. The basis for this strategy is that price movement is not a random walk. Rather, current prices reflect all available information to the market. Forecasted prices can then be estimated using patterns recognized within an equity's chart.

The top indicators in technical analysis are[3]:

- Lagging trend indicators
    - Is the movement up, down, or sideways over time?
- Lagging mean reversion indicators
    - How far a price swing will stretch before a retracement.
- Leading relative strength
    - Oscillations in buying/selling pressure.
- Leading momentum indicators
    - Speed of price change over time.
- Leading/lagging volume indicators

- Utilize trade volumes to indicate whether bulls or bears are in control.

With the advent of machine learning techniques coming to the forefront in the last decade, we can now attempt to use computational power to better predict the direction of stock movements. In this project I will be using Amazon's DeepAR algorithm to attempt to predict stock prices of a select few stocks for a given forecasted time period. This algorithm is a supervised learning method for forecasting scalar time series using recurrent neural networks (RNNs)[4].

Alternatively, traditional computational forecasting methodologies like autoregressive integrated moving average (ARIMA) or exponential smoothing (ETS) are possible. These models work great on individual time series that an implementer looks to project into the future. However, if looking to project multiple related time series, the DeepAR model outperforms these two methods. Additionally, the DeepAR model starts to outperform the ARIMA and ETS methods when the dataset to be trained contains hundreds of related time series. Further reading on how this model works can be found in the AWS Documentation.

For the purposes of this problem, I will only be focused on predicting adjusted close prices for stocks. Adjusted close prices are closing stock prices at the end of each trading day that account for any corporate action such as stock splits, dividends, or rights offerings. Thus, we can compare daily closing stock prices of a stock like General Electric (GE) that pays regular dividends and has split 7 times since it started trading publicly.

The training data fed to this algorithm will be historical adjusted close prices for a stock. The DeepAR model is trained by randomly selecting data from the training dataset. Additionally, the model is given a pair of time windows, a "context length" and a "prediction length", that are adjacent to one another. The context length tells the algorithm how far to look into the past while the prediction length dictates how far to predict into the future. Figure 1 below displays this approach. The orange areas represent the context length. The blue areas represent the prediction length.

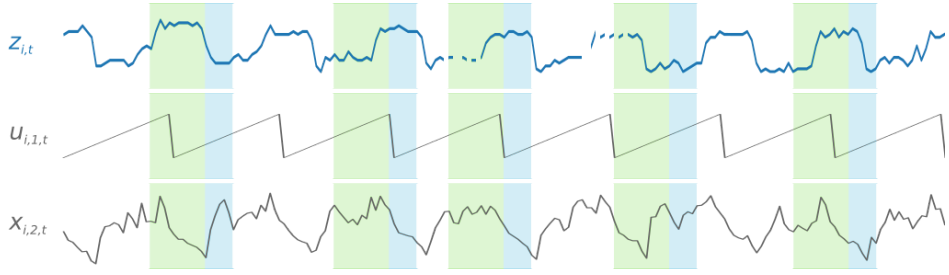

Figure 1. DeepAR depiction of training on adjacent context and prediction lengths.

Optional test data can be fed to the DeepAR model as well. The test dataset is used to evaluate the model (similar to a validation dataset in other supervised learning models). For this stock prediction problem, I will utilize the previous 3 years of historical stock data minus the prediction

length as my training dataset. I will then include the prediction length data in my test dataset to determine how well the model fits our data. However, to perform a proper inference, I'll have to test the trained model with data it has never seen before. To do so, I'll provide the model with the subsequent prediction length adjusted close prices to the end of our test data and without a target for the model to evaluate against.

Datasets & Inputs

I used an API to capture data provided by quandl. Free stock prices of over 2,000+ stocks are provided through this website up until 2018. Open, high, low, and close prices, adjusted prices, volumes, and split ratios are all provided. As noted above, I'm only be focused on adjusted close prices for this project. I'll use my custom API key to pull the data from quandl's database. The pulled data is already tabularized into a dataframe for us. See below for an example of an output of the data:

Date	Open	High	Low	Close	Volume	Ex-Dividend	Split Ratio	Adj. Open	Adj. High	Adj. Low	Adj. Close	Adj. Volume
1962-01-02	75.00	76.2500	74.25	74.75	21600.0	0.0	1.0	0.329505	0.334997	0.326210	0.328407	2073600.0
1962-01-03	74.38	74.3800	73.75	74.00	14800.0	0.0	1.0	0.326781	0.326781	0.324014	0.325112	1420800.0
1962-01-04	74.00	74.6200	72.50	73.13	18400.0	0.0	1.0	0.325112	0.327836	0.318522	0.321290	1766400.0
1962-01-05	73.13	73.2500	70.00	71.25	27300.0	0.0	1.0	0.321290	0.321817	0.307538	0.313030	2620800.0
1962-01-08	71.25	71.2500	69.00	71.13	31000.0	0.0	1.0	0.313030	0.313030	0.303145	0.312503	2976000.0
...	...	...	...	...	...	...	...	...	...	...	...	...
2018-03-21	13.66	13.9600	13.57	13.88	64989359.0	0.0	1.0	13.660000	13.960000	13.570000	13.880000	64989359.0
2018-03-22	13.75	13.7900	13.32	13.35	70929333.0	0.0	1.0	13.750000	13.790000	13.320000	13.350000	70929333.0
2018-03-23	13.40	13.4499	13.02	13.07	82930120.0	0.0	1.0	13.400000	13.449900	13.020000	13.070000	82930120.0
2018-03-26	13.23	13.2395	12.73	12.89	101095809.0	0.0	1.0	13.230000	13.239500	12.730000	12.890000	101095809.0
2018-03-27	12.92	13.7200	12.82	13.44	153476613.0	0.0	1.0	12.920000	13.720000	12.820000	13.440000	153476613.0

Table 1. Sample stock data pulled from quandl

Benchmark Model

For this project, I'll use a Naive method to compare our model against. In the Naive method, forecasts for every new time period correspond to the last observed value. It is described by the following equation[5]:

$$\hat{Y}(t+h|t) = Y(t)$$

Evaluation Metrics

I will use the mean absolute percentage error (MAPE) to evaluate the accuracy of my model and the Naive method. The MAPE provides a suitable indication of how well the model predicts the output over the entire prediction time horizon. It is calculated as follows[6]:

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

where A_t is the actual value, F_t is the forecasted value, and n is the number of periods.

PREPROCESSING

APIs related to historical stock price data pull very organized datasets as shown in Table 1 above. Thus, a simple call to the API will extract the data we need. I plotted visuals primarily to observe the general behavior of one or many stocks. The code below captured the initial data, provided a start and end year for the data I fed to my model, and then plotted the adjusted close price of this stock.

```
stocks = ['GE']

# Define years of interest to model against
start_year = '1992'
end_year = '1994'
data = quandl.get("WIKI/"+stocks[0])

# Plot stock price of individual stock for Adjusted Close price and given
years to train against
plt.plot(data['Adj. Close'][start_year:end_year])
```

The chart below displays the general stock data for General Electric (GE) between the years I plan to train my model on: 1992 to 1994.

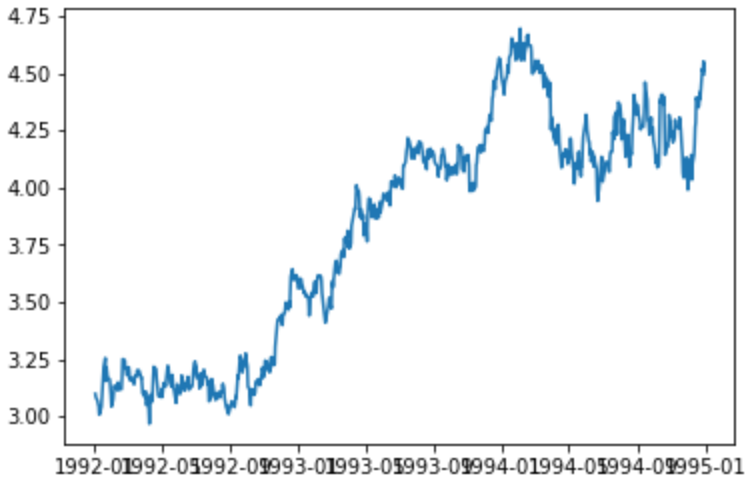

Figure 2. GE adjusted close stock price for 1992 to 1994

The data captured using my API from quandl needs to be preprocessed in order to feed it into the DeepAR model. First, I chose to separate the data by year into a series. Thus, I end up with a series of daily adjusted close stock prices for a stock separated in a Pandas series by year. This series looks like the following:

```
[Date
 1992-01-02    3.098627
 1992-01-03    3.083664
                  ...
 1992-12-31    3.560810
 Name: Adj. Close, Length: 254, dtype: float64,
 Date
 1993-01-04    3.560810
 1993-01-05    3.602457
                  ...
 1993-12-31    4.487893
 Name: Adj. Close, Length: 253, dtype: float64,
 Date
 1994-01-03    4.450662
 1994-01-04    4.434400
                  ...
 1994-12-30    4.497406
 Name: Adj. Close, Length: 252, dtype: float64]
```

As discussed earlier, the DeepAR algorithm can accept train and test channels of datasets. For this example, the training data will be all of the prices from 1992 to the end of 1994 minus the prediction length (e.g. 7 days). The test data will be all of the adjusted close price data from 1992 to 1994. The following graph shows the output for one year (1992) of my train/test split code that separated the time series into training and test data.

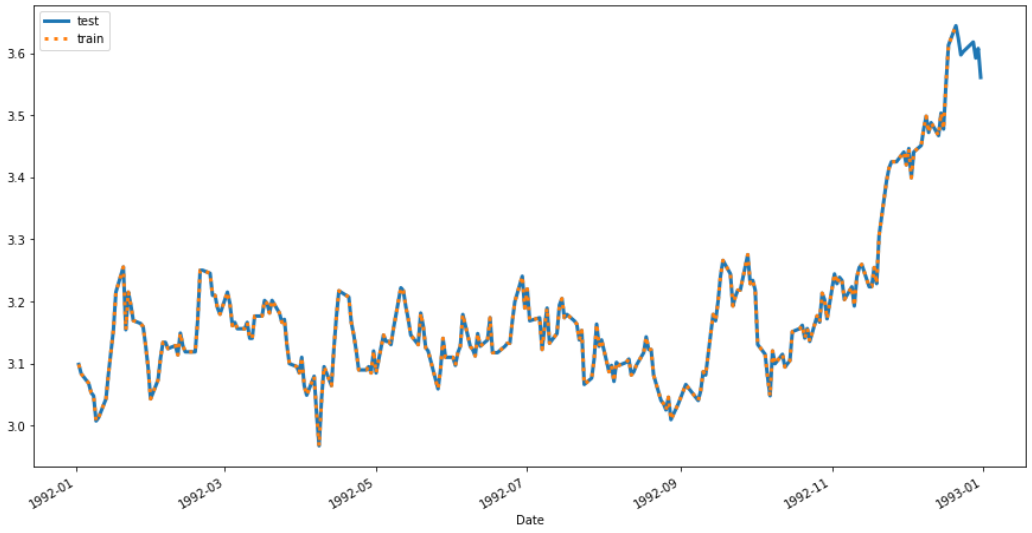

Figure 3. Visual depiction of train and test data for DeepAR algorithm

This data needs to be converted from a Pandas series into a JSON object in order for the DeepAR algorithm to accept it as input. The data takes the following input:

```
{"start": ..., "target": [1, 5, 10, 2], "dynamic_feat": [[0, 1, 1, 0]]}
```

The “start” is a timestamp (‘YYYY-MM-DD HH:MM:SS’), and the “target” list is composed of the adjusted close prices for that specific year. The dynamic_feat is an optional data source that I ignored for the purposes of this solution. An example reformatted JSON line of the GE stock dataset from 1992 to 1994 is below.

```
{'start': '1992-01-02 00:00:00', 'target': [3.0986272202879,
3.0836638677493, 3.0682961002773, ...]}
```

This data was then saved locally and uploaded to S3 for SageMaker to be able to utilize via the following code.

```
# save this data to a local directory
data_dir = 'json_stock_data'

# make data dir, if it does not exist
if not os.path.exists(data_dir):
    os.makedirs(data_dir)
# directories to save train/test data
train_key = os.path.join(data_dir, stocks[0] + '_train.json')
test_key = os.path.join(data_dir, stocks[0] + '_test.json')

# write train/test JSON files
sts.write_json_dataset(time_series_training, train_key)
sts.write_json_dataset(time_series, test_key)

import boto3
import sagemaker
from sagemaker import get_execution_role

sagemaker_session = sagemaker.Session()
role = get_execution_role()
bucket = sagemaker_session.default_bucket()

prefix = 'deepar_' + stocks[0]
```

```
# *unique* train/test prefixes
train_prefix   = '{}/{}'.format(prefix, 'train')
test_prefix    = '{}/{}'.format(prefix, 'test')

# uploading data to S3, and saving locations
train_path  = sagemaker_session.upload_data(train_key, bucket=bucket,
key_prefix=train_prefix)
test_path   = sagemaker_session.upload_data(test_key,  bucket=bucket,
key_prefix=test_prefix)

# check locations
print('Training data is stored in: '+ train_path)
print('Test data is stored in: '+ test_path)
```

Train, Fit, and Deploy

To train the data, I created a container, an estimator object, and hyperparameters for the estimator. Initially, I started with a rough estimate for what the hyperparameters should be for this problem. Hyperparameters included:

- Epochs: Maximum number of passes over the training data
- Time frequency: Granularity of the time series data (e.g. hourly, daily, monthly, etc.)
- Prediction length: How far into the future to predict
- Context length: How far in the past to use to train for prediction length
- Number of cells: Number of cells in each hidden layer of the RNN
- Number of layers: Number of hidden layers in the RNN
- Mini batch size: Mini batch size of data to be fed to the model
- Learning rate: Learning rate used during training
- Dropout rate: Dropout rate used during training
- Early stopping patience: Optional, but if set, the training will cease when there isn't an improvement in loss within the specified number of epochs

```
from sagemaker.amazon.amazon_estimator import get_image_uri

image_name = get_image_uri(boto3.Session().region_name, # get the region
                           'forecasting-deepar') # specify image


from sagemaker.estimator import Estimator

# directory to save model artifacts
s3_output_path = "s3://{}/{}/output".format(bucket, prefix)
```

```
# instantiate a DeepAR estimator
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
                      )

# need to feed frequency to hyperparameters
# set to 'D' for daily
freq='D'

# assign context_length. Start with it equal to prediction_length
context_length = prediction_length

hyperparameters = {
    "epochs": "25",
    "time_freq": freq,
    "prediction_length": str(prediction_length),
    "context_length": str(context_length),
    "num_cells": "50",
    "num_layers": "2",
    "mini_batch_size": "128",
    "learning_rate": "0.001",
    "dropout_rate": "0.1",
    "early_stopping_patience": "10"
}

# set the hyperparams
estimator.set_hyperparameters(**hyperparameters)
```

I trained my model on these initial hyperparameters, deployed the endpoint, and observed the trained model performance on the test data. The model was terrible at predicting the test data's performance based on my predicted data visualizations (more on that later). Thus, after trial-and-error attempts at tuning the hyperparameters myself, I determined it would work best to set up a hyperparameter tuning job.

HYPERPARAMETER TUNING/REFINEMENT

To tune the hyperparameters of my model, I had to provide upper and lower bounds to the dynamic features I wanted to investigate: epochs, learning rate, and dropout rate. Additionally, I

provided a goal for the tune job to measure the model performance against: minimizing mean_wQuantileLoss of the test data.

To execute the tuning job, train the model with the optimal hyperparameters, and deploy the model to an endpoint, I used the following code:

```
from sagemaker.tuner import IntegerParameter, ContinuousParameter,
HyperparameterTuner

hyperparameter_tuner = HyperparameterTuner(estimator = estimator, # The
estimator object to use as the basis for the training jobs.
            objective_metric_name = 'test:mean_wQuantileLoss', # The metric
used to compare trained models.
            objective_type = 'Minimize', # Whether we wish to minimize or
maximize the metric.
            max_jobs = 9, # The total number of models to train
            max_parallel_jobs = 3, # The number of models to train in
parallel
            hyperparameter_ranges = {
                     'dropout_rate': ContinuousParameter(0.1, 0.2),
                     'learning_rate': ContinuousParameter(0.0005, 0.05),
                     'epochs': IntegerParameter(1, 50)
                                          })

# tell S3 where the data will be and what type of format to expect
s3_input_train = sagemaker.s3_input(s3_data=train_path,
content_type='json')
s3_input_test = sagemaker.s3_input(s3_data=test_path, content_type='json')

# fit
hyperparameter_tuner.fit({'train': s3_input_train, 'test': s3_input_test})

# output configuration of best_training_job. Use SageMaker interface to
find actual parameters.
hyperparameter_tuner.best_training_job()

# establish new estimator from best_training_job (commented out previous
estimator)
estimator_best =
sagemaker.estimator.Estimator.attach(hyperparameter_tuner.best_training_job
())
```

```
%%time

# create a predictor
predictor = estimator_best.deploy(
    initial_instance_count=1,
    instance_type='ml.t2.medium',
    content_type="application/json" # specify that it will accept/produce
JSON
)
```

The DeepAR predictor delivers a predicted probability distribution for each time series input into the algorithm upon prediction. Additionally, the output is delivered as a JSON file. To determine how well the model fit the trained data, I passed the data back into the model (again, converting it to JSON for proper input acceptance). To extract a meaningful prediction of the data, I provide the proper quantiles of interest (0.1, 0.5, and 0.9) to a function that converts my training time series into JSON format. Then I was able to plot the median predicted adjusted close price for the probability distributions. The 0.1 and 0.9 quantiles allowed me to plot an 80% confidence interval around the median predicted values.

To convert the data into JSON to feed into the predictor, I used the following function.

```
def json_predictor_input(input_ts, num_samples=50, quantiles=['0.1', '0.5',
'0.9']):
    """
    Accepts a list of input time series and produces a formatted input.
       :input_ts: An list of input time series.
       :num_samples: Number of samples to calculate metrics with.
       :quantiles: A list of quantiles to return in the predicted output.
       :return: The JSON-formatted input.
    """
    # request data is made of JSON objects (instances)
    # and an output configuration that details the type of data/quantiles
we want

    instances = []
    for k in range(len(input_ts)):
        # get JSON objects for input time series
        instances.append(series_to_json_obj(input_ts[k]))

    # specify the output quantiles and samples
    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
```

Taking this JSON formatted training time series data and feeding it back into the predictor, I was able to then gather the predicted price data for comparison. The DeepAR output is also JSON, so a helper function that decoded the output was also written so as to extract the 0.1, 0.5, and 0.9 predicted prices.

RESULTS

I was able to plot the median predicted values (0.5 quantile) and 80% confidence interval (0.1 and 0.9 quantile bounds) for GE in 1992, 1993, and 1994 compared to the actual values pulled from the test data. Below is a chart for 1994 that displays this graphical analysis.

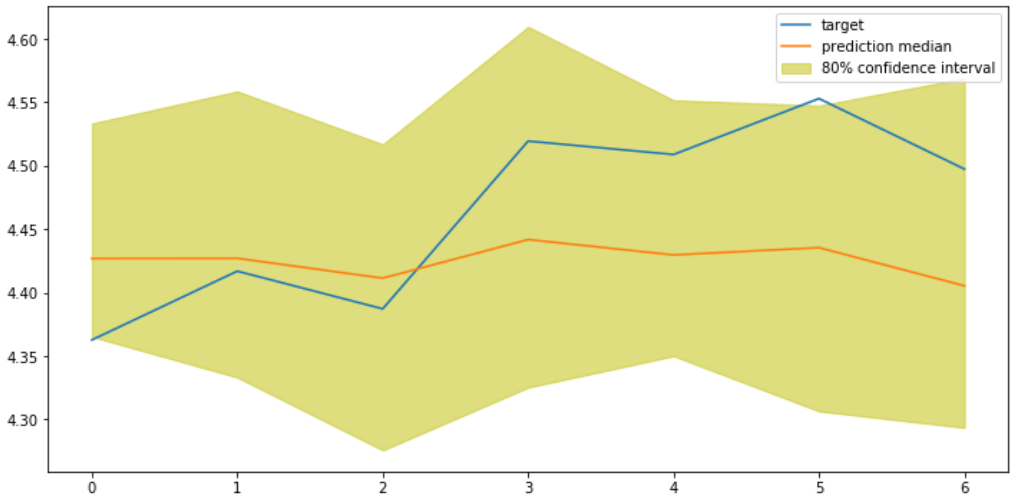

Figure 4. Prediction median values compared to target values of trained data

Visually, it's evident this model can predict the target value relatively well. This type of fit was similar across the other years that the model was trained on (1992 and 1993). However, the high visual accuracy of this graph is somewhat misleading -- this is data that the model has already been trained on. I need to determine how well the model will predict 7 days out on data it has never seen before.

Thus, I created a JSON input object similar to before with the first 7 trading days in 1995, but with no target values specified. This data structure was fed to the predictor, and the median predicted adjusted close prices were extracted and graphed against the real 1995 trading data from those 7 days. The figure below depicts this.

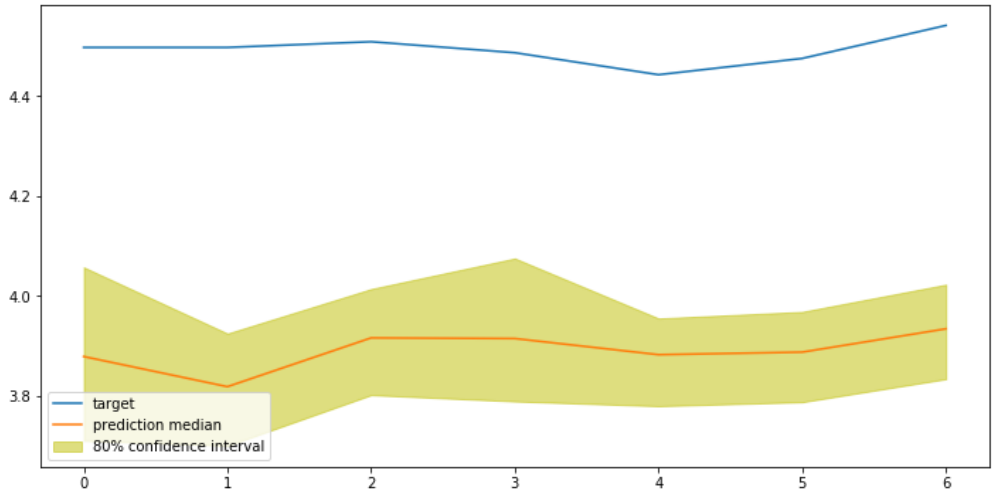

Figure 5. Prediction median values compared to target values of "future" data (1995)

It appears that my predicted value isn't a great predictor of future performance. However, a quantifiable metric will provide a better sense for the actual performance of the model for predicting future stock prices.

Accuracy

It is prudent to compare my model to the most basic forecasting model: the Naive method. As mentioned previously, the Naive model predicts the output at a certain time by taking the last observed data point. A comparison of the Naive model vs. our DeepAR algorithm is depicted in the figure that follows.

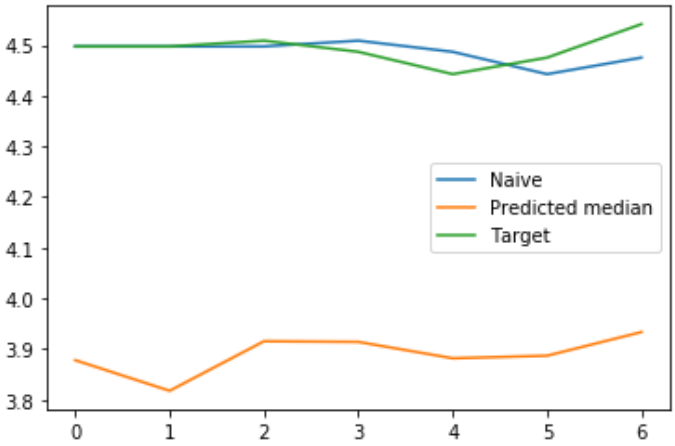

Figure 6. Naive model vs. DeepAR model for 1995 GE first 7 trading days

It is evident that the Naive model actually tracks the target value very closely, while our predicted model is somewhat off the mark. Let's quantify by how much our model is inaccurate vs. the Naive model by using the aforementioned mean absolute percentage error calculation (MAPE).

Table 1. MAPE of 7-day prediction of GE stock based on DeepAR algorithm

	MAPE
Naive, 7-day prediction	0.56%
DeepAR, 7-day prediction	13.4%

I expanded on this finding by looking at short prediction lengths, 3- and 5-day, to see if there would be an improvement in MAPE. The chart below describes the MAPE of the Naive model as well as the DeepAR model over these different prediction lengths.

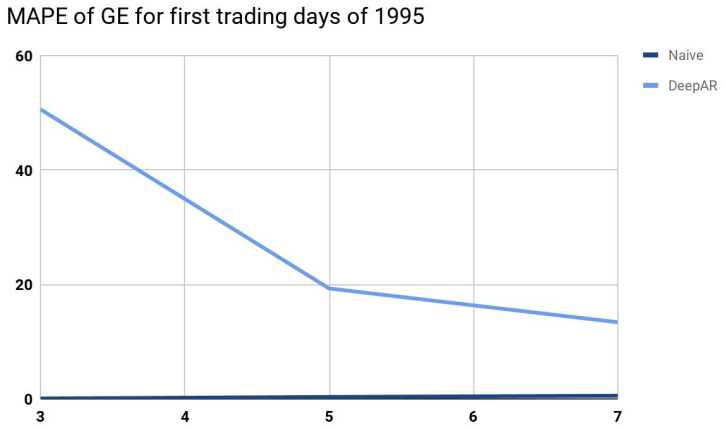

Figure 7. Naive vs. DeepAR MAPE for GE stock for 3-, 5-, and 7- day predictions

Additionally, to check to see if our results for a 7-day prediction length would improve by looking at a different stock, I trained a new model on Microsoft stock (MSFT). After deploying an endpoint predictor, I was able to calculate a new MAPE for the "future" (beginning of 1995) for MSFT.

Table 2. MAPE of 7-day prediction of MSFT stock based on DeepAR algorithm

	MAPE
Naive, 7-day prediction	1.16%
DeepAR, 7-day prediction	25.7%

Conclusions/Future Steps

It's evident that the DeepAR model for a single stock is a pretty poor predictor of adjusted close stock prices. Alternatively, the Naive method appears to perform very well for this small time window. Intuitively, that makes sense -- it's very rare that I'd be able to capture the exact moment in my time series analysis that a stock jumps 40%, for example, in my prediction time window.

I believe there are a number of reasons that the DeepAR algorithm struggled to produce a model that predicted stock price movement with above average accuracy. Generally speaking, stock price movements are still very difficult to predict for investment professionals. Even for a robust model like DeepAR. Admittedly, the DeepAR documentation claims that the prediction accuracy starts to improve as the number of similar time series starts to increase closer to 300 samples. I think one of the major issues with my data is that while stocks tend to increase over time, there are still many periods of volatility that don't follow any trends from previous data. As this is the data that my model is trained on, my model inherently can't pick up on many patterns given one stock. Additionally, while there is some seasonality to stock performance across industries, I reckon on an individual stock level that observation is less likely to occur as regularly.

The natural progression then would be to think about how to improve the DeepAR model. As mentioned previously, the algorithm works best on a large number of similar time series data of the same length. Naturally, I thought of providing stocks from the S&P 500 as a starting point for this multi-stock portfolio time series prediction approach. These stocks are not necessarily similar in nature (they range in different prices as well as industries), but at least I'd have a larger grouping of stock time series data to provide to our model. Alternatively, I could look at providing stocks from a sector ETF like VTW (Vanguard's Information Technology ETF) that holds stocks all from a very similar industry. Thus, the 328 holdings that are part of that ETF will likely track in the same direction over time as one another. This data might be a better input to the DeepAR algorithm. However, ultimately I decided to start with a very small subset of tech stocks to feed to DeepAR as a starting point to see if there was any improvement over my single stock model.

I went through the same process as I did building a model for a single stock by running a hyperparameter tuning job and feeding the best model to an endpoint predictor for stocks in a portfolio that included GE (carried over from previous stock "portfolio"), AMD, GOOGL, AAPL, AMZN, NFLX, and MSFT. Like my single-stock model, the predictions on test data displayed a high level of accuracy. However, my multi-stock model had a very difficult time predicting the adjusted close prices for each of the stocks in my portfolio at a rate even worse than my single-stock model. The MAPE for the predicted stocks ranged from 17% to 2000%. On a granular level, the model mainly predicted stocks ranging in prices from ~$180 to $300, yet the actual adjusted stock prices ranged from $11 to $1200 per share! It's no surprise the MAPE was so high. Rather than go through displaying the results for this here, the work can be found in my "Alternate Models" directory.

Clearly my hypothesis that we'd see an improvement in MAPE for a portfolio of stocks fed to the model appears to be incorrect. The inaccurate prediction of my model won't be corrected by increasing the number of similar stocks fed to the model or even the number of dissimilar stocks. As noted before, it is evident that stock prediction can be tricky. Moreso, I'm now convinced that DeepAR is not the model to use for this type of stock prediction.

Amazon's DeepAR documentation states that ARIMA and ETS may be better forms of forecasting when there isn't a large amount of time series data available. Future work around this idea would focus on using one of these more traditional forecasting approaches.

CITATIONS

[1] Corporate Finance Institute. "Valuation Methods: The main methods used to value a business." https://corporatefinanceinstitute.com/resources/knowledge/valuation/valuation-methods/. Accessed 20 Sep. 2020.
[2] Stock Charts. "What is Technical Analysis?" *Stock Charts,* https://school.stockcharts.com/doku.php?id=overview:technical_analysis. Accessed 20 Sep. 2020.
[3] Barone, Adam. 10 Jun. 2020. "Top Technical Indicators for Rookie Traders." *Investopedia*, https://www.investopedia.com/articles/active-trading/011815/top-technical-indicators-rookie-traders.asp. Accessed 20 Sep. 2020
[4] Amazon Web Services. "DeepAR Forecasting Algorithm." https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html. Accessed 20 Sep. 2020.
[5] Burba, Davide. 3 Oct. 2019. "An overview of time series forecasting models." https://towardsdatascience.com/an-overview-of-time-series-forecasting-models-a2fa7a358fcb. Accessed 20 Sep. 2020.
[6] Institute of Business Forecasting & Planning. "MAPE (Mean Absolute Percentage Error)." https://ibf.org/knowledge/glossary/mape-mean-absolute-percentage-error-174 Accessed 20 Sep. 2020.