LSTM (Lucrative Stock Trading Machine)

Using Deep Learning to Predict Stock Price Movement

Machine Learning Capstone Project

Sloan Austermann

June 2020

1 Definition

1.1 Project Overview

This purpose of this project is to create a deep learning algorithm that can learn patterns in from S&P500 price and volume data to determine which stocks in the index will have the highest chance of positive future daily returns and which stocks have the highest change of negative daily returns. This problem has historical roots in the trading of financial security, particular over short-term time horizons. There are two philosophies of analysis in the world of investing and trading, fundamental analysis and technical analysis. Technical analysis uses heuristics or mathematical calculations based on the price, volume or open interest of a security. Fundamental Analysis, on the other hand, is a method of measuring a security's intrinsic value through the examination of economic and financial factors. Macroeconomic factors like the interest or tax rates or microeconomic factors like the allocation of capital are both taken into consideration in attempt to calculate a stock's intrinsic value[1]. Momentum, for example, is one of the most widely known and used technical indicators because of its ability to achieve abnormal returns (returns above the market), over varying time intervals.

A 2015 study looked at momentum strategies on Global Multi-Asset data from 1800-2015 found that this strategy is not only effective on Equities but also Forex, Bonds, Commodities and other asset classes[2]. There many other technical indicators that all fall under the four main types of indicator: Trend, Momentum, Volume and Volatility. Relative Strength Index is an example of a momentum indicator, the Moving Average Convergence Divergence (MACD) is an example of a trend indicator, and Bollinger Bands are an example of a volatility indicator[3]. All of these indicators have proven to have success in trading strategies that executed over time frames of a few weeks to a few months. Fundamental analysis has is backed by economic theory and has

[1] Investopedia, Technical Analysis, 2020.

[2] Christopher C. Geczy, Mikhail Samonoc, 215 Years of Global Multi-Asset Momentum, 2015.

[3] Harry Nicholls, 7 Popular Technical Indicators and How to Use Them, 2018.

been proven to be successful over long time periods. On the other hand, High Frequency Trading happens incredibly fast, executing trades in microsecond time intervals purely based off of price and volume data. The strategy being implemented in this project will attempt to use deep learning to find a successful strategy using a combination of factors over a medium range time frame of a 30-120 days.

As an undergraduate at the University of Notre Dame, I studied Mathematics and Economics, focusing on a concentration in financial econometrics. Studying economics through this financial lens taught me to appreciate the power of the stock market as a data processing machine. This academic background inspired and motivated the idea to apply machine learning to the financial market data in an attempt to better understand its seemingly enigmatic behavior.

1.2 Problem Statement

Since we are working with time-series data, the machine learning model developed in this project will be designed to solve a forecasting problem. Specifically, since we are working with stock price data, the model will attempt to predict the movement of the asset prices using a look-back window of historical data leading up to the time of prediction. The goal of applying machine learning trading is to determine which stock prices will increase in the future and which stock prices will decrease in the future. If the historical price data is one-hot-encoded to categorize the data into three categories based off of some daily return threshold, T, with all stocks that have a return greater than T over the prediction period categorized as a buy, stocks with a return below -T over the prediction period categorized as a sell, and everything else categorized as a hold. Assigning these values to each security for each day over the range of the historical data will turn our problem into one of classification. Rather than building a regression algorithm to predict the stock price of each security, we can classify them as buy, sell or hold based on whether we believe the price will go up or not.

Each security will have a target label (buy, hold, sell) for each day. The inputs for that label will be the n-day window of OHLCV and technical indicator data. The label will be determined based off of the price movement during the prediction window. The inputs for day n will be the sequence of day from n-50 to n-1. The label for day n will be:

> **buy,** if price (n+k) /price(n) > T.
>
> **sell,** if price(n+k)/price(n) < -T.
>
> **hold,** otherwise.

For some return threshold T and prediction window length k. The if the model will be tasked to accurately classify each stock's return over the prediction window as either greater than the threshold, less than the negative threshold or in between.

1.3 Metrics

Accuracy is the primary metric used to evaluate the performance of the model. Ultimately, the most important part of building an algorithmic trading machine is pick the right trades. If the model picks profitable trades more often than not, then the model will be working sufficiently. Accuracy can often be a misleading metric for model performance if there are imbalances in the data set. However, because the data was processed to evenly balanced across the target groups, accuracy will work perfectly for this project.

In addition to accuracy, the 'test:RMSE' of the model will also be used as a loss function in order to determine the model's performance throughout the training process.This metric is defined as the "root mean square error between the forecast and the actual target computed on the test set"[4] This is the best metric to determine the model's performance on the test data and takes into account imbalances in the data. Furthermore, this metric is also used to optimize the model's performance through hyperparameter tuning in Amazon's SageMaker. The Hyperparameter Tuner used in this project evaluated the model and optimized the

[4] Amazon Web Services, Tune a Deep AR Model (2020).

hyperparameters with the objective of minimizing the root mean square error between the forecasted targets and the actual targets in the test set.

Further consideration in model evaluation could be taken by back testing the output of the model's trade picks in a simulated training test. This technique can be implemented using the Zipline python library, in which the models forecasted targets can be used to simulate a real trading algorithms[5]. This performance of this algorithm can also be used as a performance metric. The ultimate task would be to build a model with optimal risk/return profile and beat the market return without taking on too much additional risk.

[5] Zipline.io, Risk and Performance Metrics (2016).

2 Analysis

2.1 Data Exploration

IThe data used for this project is OHLCV data for the stocks trading on the S&P 500 index. This data was pulled from Yahoo Finance using the python library pandas-datareader. The data I collected and the code for I used to mine it are included in the GitHub Repo associated with this submission. The data is stored in the 'stocks_dfs' folder and the code used to get the data is in get_data.py and was used as part of a Sentdex's tutorial on python for financial analysis[6]. Because not all of the data sets were properly formatted, the web scraping process was gathered 491 individual data frames of OHLCV stock data. This data is structed in a panel and hierarchically indexed by date and stock ticker. The outer most index is the daily date time index starting at Janurary 1st, 2010 ("2010-01-01") and ending on January 1st, 2020 ("2020-01-01"). The second level index was the categorical ticker name. In total, across the 491 stocks in the data set there are 1,177,323 observations over the 10-year period.

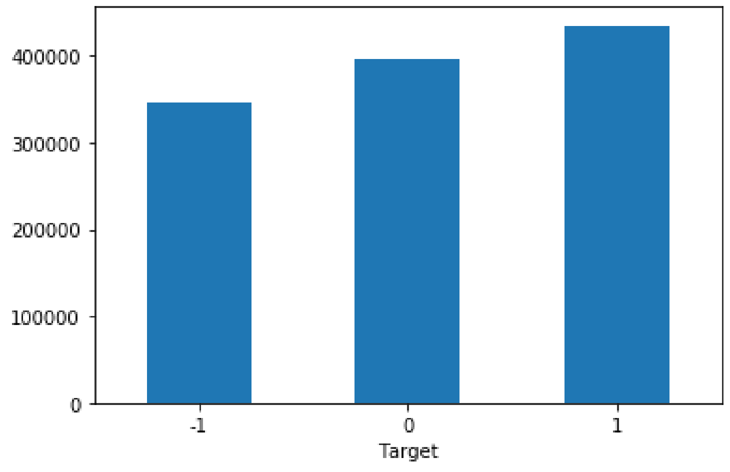
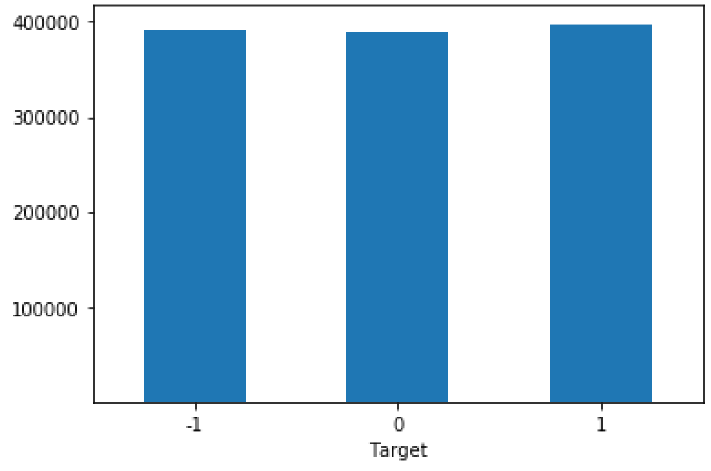
All features within the data set are quantitative variables other than the categorical ticker labels in the second level index. The primary data used for the feature sets in this project are the Adjusted Close price and Volume data, indexed as previously described. Other features used as inputs in the model are in the form of technical indicators calculated from each stock's Adjusted Close price data. The technical indicators used in this project are Momentum, Bollinger Bands, MACD, RSI and the rolling mean and standard deviation, all of varying time frames. The following table displays the summary statistics for the price, volume and 25-day indicators in the feature set used to train the model.

[6] Python Programming For Finance, Getting all company pricing data in the S&P 500 (2017)

	Adj Close	Volume	50MA	50STD	50UBB	50LBB	50RSI	50Mom
count	1.17 e+06	1.17 e+06	1.17 e+06	1.17 e+06	1.17 e+06	1.17 e+06	1.17 e+06	1.17 e+06
mean	7.65 e+01	4.87 e+06	7.53 e+01	2.98 e+00	8.13 e+01	6.93 e+01	5.31 e+01	2.20 e+00
std	1.34 e+02	1.14 e+07	1.32 e+02	5.94 e+00	1.42 e+00	1.21 e+02	8.99 e+00	1.60 e+01
min	4.95 e-03	0.00 e+00	0.00 e+00	0.00 e+00	0.00 e+00	-7.6 e+01	0.00 e+00	-8.5 e+02
max	3.89 e+03	8.60 e+08	3.71 e+03	2.93 e+02	3.96 e+03	3.51 e+03	1.00 e+02	6.80 e+02

2.2 Exploratory Visualization

The following plots show the distribution of the target values in the data set. Originally, the positive and negative daily return thresholds were set to the same absolute value of 0.025. However, due to the general tendency for stock prices to increase, there was a positive skew in the target distribution (Figure 1). In order to correct for this and have a more evenly distributed target class, the daily return thresholds were adjusted accordingly to limit the number of **buys** and increase the number of **holds**. Choosing a positive threshold of 0.027 and a negative threshold of -0.022, the targets were much more evenly distributed (Figure 2).

Target Distribution			Target Distribution (Balanced)		
-1: 345713	0: 396929	1: 434681	-1: 390748	0: 389090	1: 397485
 *Figure 1, Target Distribution*			 *Figure 2, Target Distribution (Balanced)*		
Positive Threshold **0.025**	Negative Threshold **-0.025**		Positive Threshold **0.027**	Negative Threshold **-0.022**	

The next charts show examples of the raw data scraped from Yahoo Finance. These examples show the ‘Adj Close’ stock price and stock ‘Volume’ traded from ‘2010-01-01’ to ‘2020-01-01’. These data is also colored to show the train-test split that was used for training and evaluating the model. The nine-years spanning the beginning of 2010 to the end of 2019 served as the training set for the model, as shown in blue. All 252 trading days of 2019 are shown in orange.

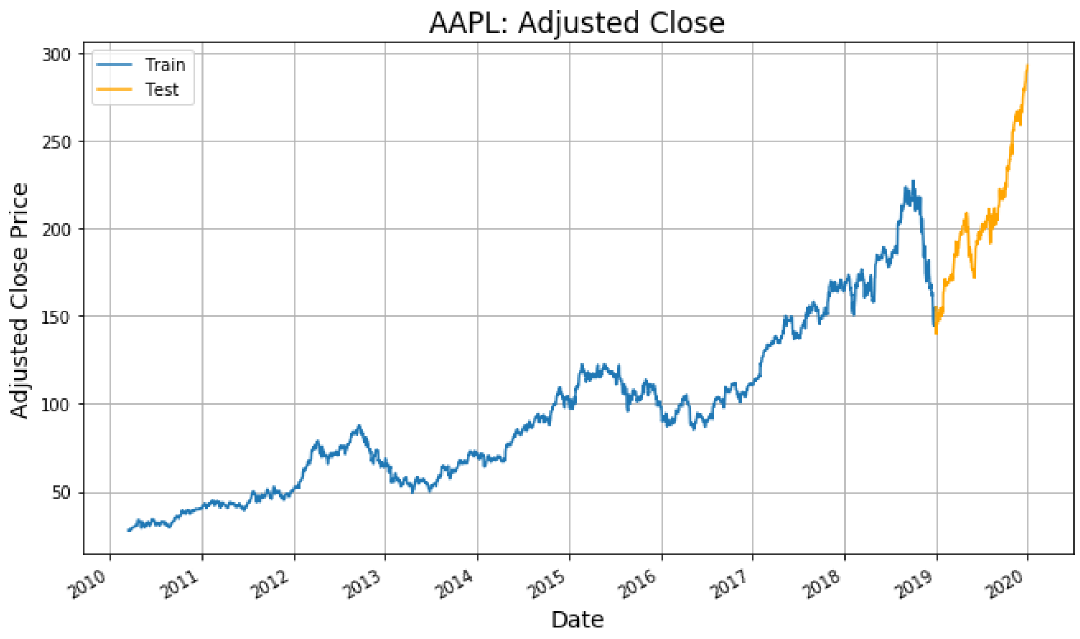


Figure 3, AAPL: Adjusted Close

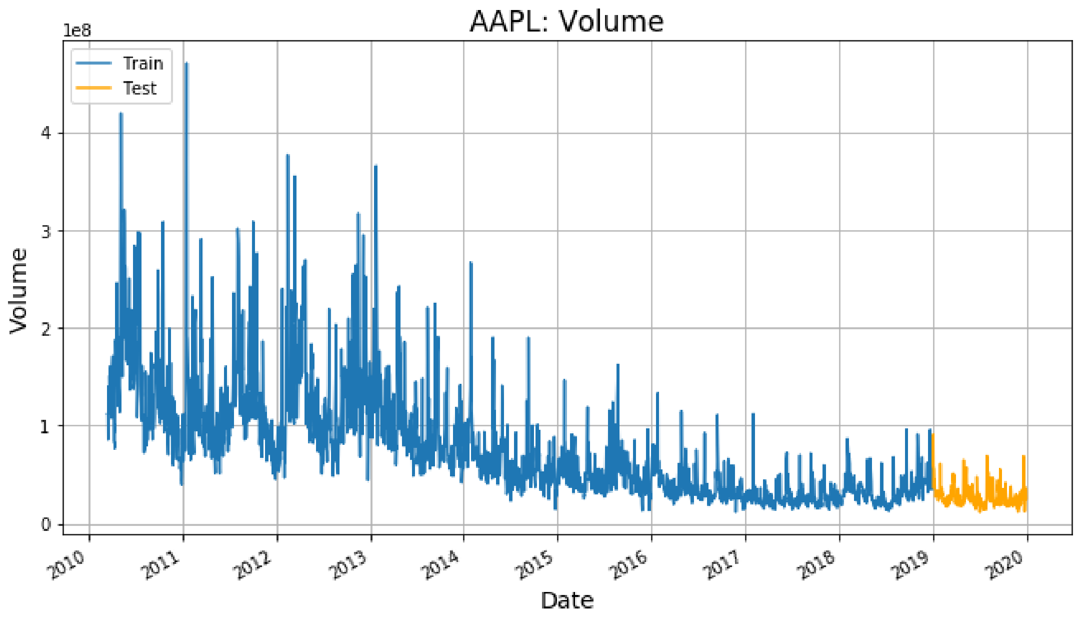


Figure 4, AAPL: Volume

The last set of graphs show examples of the technical indicators calculated from each stocks price data. These collection of these indicators across varying time frames were subjected to principal component analysis in order to reduce the dimensionality of the feature set used for training. All of these graphs are created from the test data from 2019. The graphs are colored red where the target is labeled as **sell**, green where the target is labeled as **buy** and white where the target is labeled as **hold**. *Figure 4* and *Figure 5* show the graphs of the 25 Day and 50 Day Bollinger Bands for AAPL, respectively.

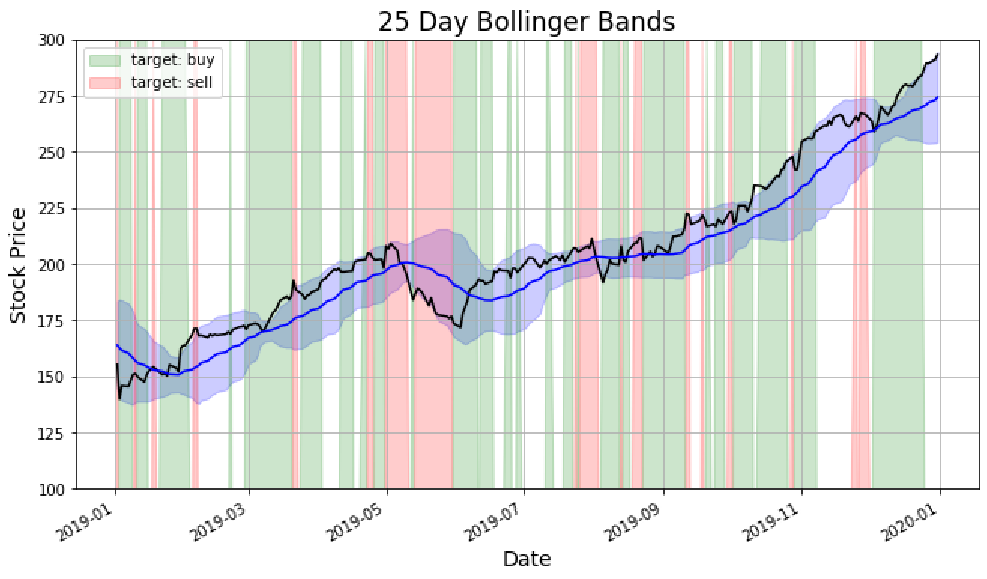


Figure 5, 25 Day Bollinger Bands

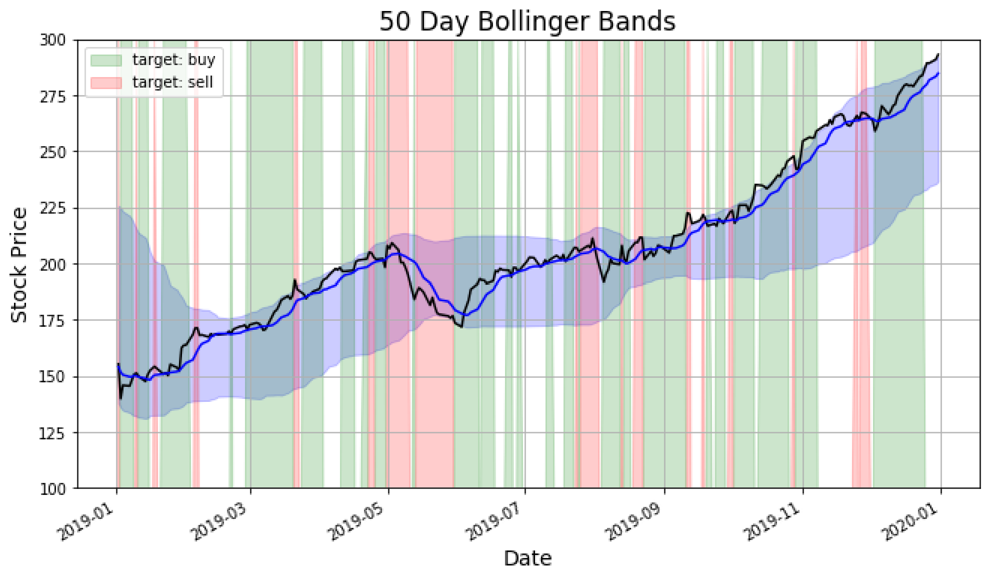


Figure 6, 50 Day Bollinger Bands

The last graphs in this section show the Relative Strength Index and Momentum indicators also calculated from the adjusted close stock prices. *Figure 7* shows the 10, 25 and 50 Day RSI.

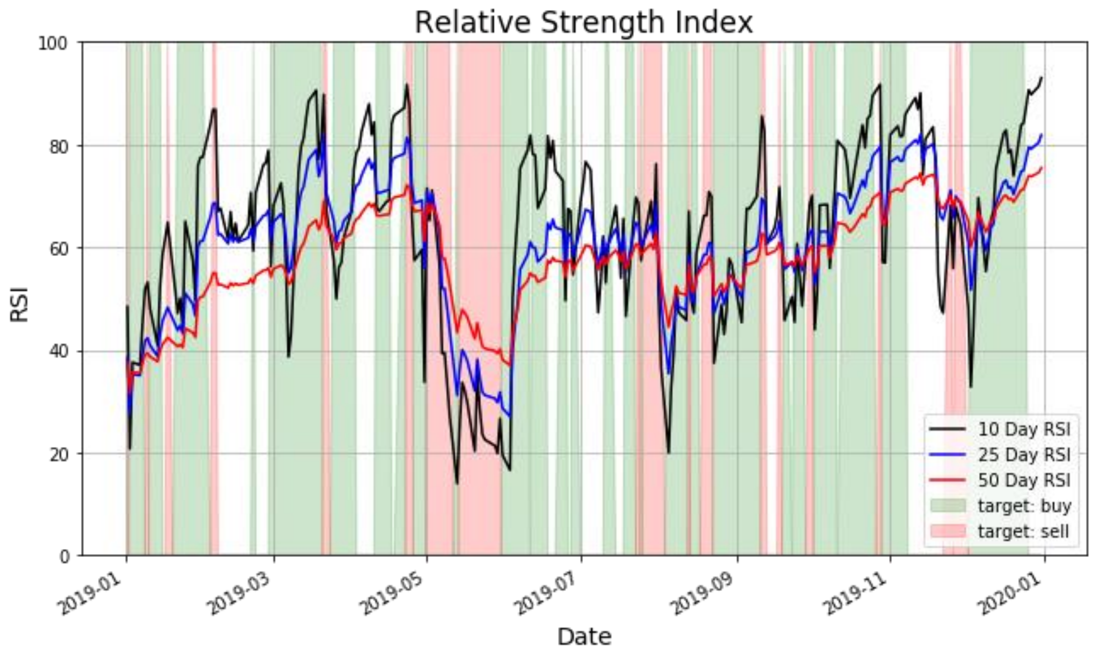


Figure 7, Relative Strength Index

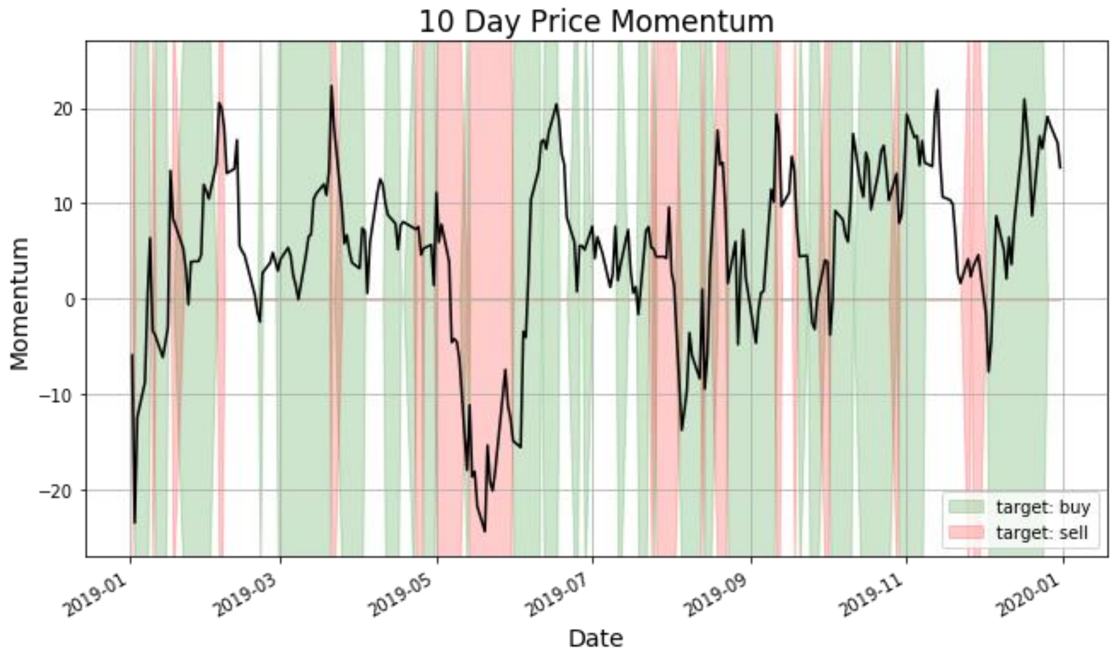


Figure 8, Ten-Day Price Momentum

2.3 Algorithms and Techniques

The primary algorithm used for this project is Amazon SageMaker's DeepAR Forecasting Algorithm. The DeepAR forecasting algorithm is a supervised learning algorithm used for forecasting one-dimensional timeseries using recurrent neural networks. The use of a recurrent neural network for this project allows for one model to be jointly trained across all the individual stock timeseries in our dataset. This is particularly helpful in stock market analysis, in which we have hundreds of related time series. Classical forecasting methods like ARIMA and ETS can only fit to single timeseries while a recurrent neural network can fit to all of the stock data from the S&P 500.

The training input for this algorithm consists of one or more "target" time series. In the context of the project, the target time series are the trading labels **buy**, **hold** and **sell (1, 0, -1)** for each security. Each target time series can be associated with a vector of time-independent categorical features defined as the "cat" field and time-dependent time series defined as the 'dynamic_feat' field. For this project, the ticker labels were converted to unique integer i.d.'s for each stock that were used as categorical inputs in the "cat" field. Furthermore, the price, volume and dimension-reduced indicator timeseries were all used as time dependent dynamic_feat inputs.

Furthermore, there are multiple hyperparameters for the DeepAR model that must be considered to properly apply the algorithm to this particular data set and problem[7]. The training was done with the default *student-T* likelihood function for the estimation of uncertainty in the probabilistic forecasting.

[7] AWS DeepAR Hyperparameters 2010.

Parameter	**Description**
context_length	The number of time-points that the model gets to see before making the prediction
epochs	The maximum number of passes over the training data. The optimal value depends on your data size and learning rate
prediction_length	The number of time-steps that the model is trained to predict, also called the forecast horizon
time_freq	The temporal granularity of the time series in the dataset (D: daily, for this project)
dropout_rate	The dropout rate to use during training. For each iteration, a random subset of hidden neurons are not updated
learning_rate	The learning rate used in training. Typical values range from 1e-4 to 1e-1
mini_batch_size	The size of mini-batches used during training. Typical values range from 32 to 512
num_cells	The number of cells to use in each hidden layer of the RNN

2.4 Benchmark

A 2011 study also attempted to predict the direction of stick price movement using machine learning technique applied to technical indicators of as inputs to their models. They train both vanilla MLP or ANN and a Support Vector Machine on the indicators calculated from Istanbul Stock Exchange data. They experiment resulted in the ANN performing with 75.74% accuracy and the SVM attaining 71.52% accuracy[8]. Since the vanilla ANN has proven to perform well on the problem, that would be a good benchmark model to start with in this project. These models, however, were just conducted with indicators calculated from a sample from the Istanbul Stock Exchange. This project, on the other hand, will be conducted with the S&P 500 index. While this study gives us insight into how well these models can forecast stock price movements, it can only serve as a benchmark. Fundamental differences between the behavior of the stocks that trade on the Istanbul Stock Exchange and the stocks that trade on the NYSE, the NASDAQ and the CBOE BZX Exchange (the exchanges hosing the securities in the S&P500 index) can result in differences in forecasting ability.

[8] Kara, Yakup & Boyacioglu, Melek & Baykan, Omer. (2011). Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange. Expert Systems with Applications. 38. 5311-5319. 10.1016/j.eswa.2010.10.027.

3 Methodology

3.1 Data Preprocessing

All of the code used for data preprocessing done in this project is contained in the python file get_data.py and the iPython notebook file preprocess_data.ipynb. The process of scraping and collecting the data is outlined in section 2.1 Data Exploration. The output of the get_data.py file is a directory of csv files, each containing a data frame of OHLCV timeseries stock data for a particular ticker on the current S&P 500[9] index (as of May 2020).

The preprocess_data.ipynb file uses the pandas library to read the Adj Close data and the Volume data from the OHLCV stock ticker csv files into pandas data frames. Once the price and volume data are loaded into data frames, the technical indicators are calculated from the stock price data. The technical indicators used for this project are rolling means and rolling standard deviations for various time windows ranging from 5 to 50 days. From these statistics, the 50 and 25 day upper and lower Bollinger Bands are calculated. The Relative Strength Index (RSI), MACD and price momentum were also calculated for various time windows ranging from 5 to 50 days. This data frames and stock indicators were iteratively created and saved into a python ordered dictionary, then concatenated into a multi-indexed data frame, hierarchically indexed by datetime, then stocker ticker.

Next, a target was created for each observation (a day of trading for an individual ticker). To do this, the daily returns calculated at each instance in the data set. The function buy_sell_hold was then mapped to these time series to assign a target label to each instance as defined in section 1.2 Problem Statement. If the stock price were to increase by a certain threshold over the next day, it was given the target label: 1 (buy). If the price were to decrease by the threshold, it was given the target label: -1 (sell). Every other instance was labeled as a 0 (hold).

[9] Wikipedia List of S&P 500 companies 2020

Now that we create the target label column, we delete the columns we use to calculate the labels. In addition, we replace all the inf's and NaNs in the data set with 0 to avoid any missing or improper data. This leaves us with the adj close price, volume, calculated indicators and target values in the data set. In total, there are 25 columns of indicator data. Principal Component Analysis is used to compress these 25 factors into just 6 principal components while preserving 95% of the variance in the original 25 indicators. The final data set has the Adj Close, Volume, six principal components and target data hierarchically indexed by date and ticker. This pandas multi-indexed data frame is saved as a csv file in the main project directory.

3.2 Implementation

The entirety of this project was completed within an AWS SageMaker Notebook instance. The models created in the project were trained on an AWS ml.c4.2xlarge virtual machine learning instance. All of the 'deep-ar' notebooks in the MLEND-Capstone-Project GitHub Repository use the same process of implementation, in which we set up the SageMaker session and S3 storage bucket, load and restructure the data, instantiate and train a model, and create and evaluate an endpoint.

Once the SageMaker session is properly launched and the data frame is loaded from the csv file created in preprocess_data.ipynb, the prediction_length, context_length, frequency and training period are defined. The individual ticker time series split into a list of pandas data frames. This list of data frames is used to create a list of dictionary objects with each dictionary entry consisting of data for a particular ticker. First a training set is created that contains the target values and the dynamic_feat set ranging from "2010-01-01" to "2018-12-31". Test data is also created with 10 test windows that extend beyond the last training date. These two lists of dictionaries are then converted to the proper JSON format needed as input in the DeepAR model with the following fields as outlined in the SageMaker documentation[10]:

[10] AWS, Input/Output Interface for the DeepAR Algorithm 2020.

- start: string with the format YYYY-MM-DD HH:MM:SS. The start timestamp can't contain time zone information.
- target: An array of floating-point values or integers that represent the time series.
- dynamic_feat: An array of arrays of floating-point values or integers that represents the vector of custom feature time series (dynamic features). If you set this field, all records must have the same number of inner arrays (the same number of feature time series). In addition, each inner array must have the same length as the associated target value.
- cat: An array of categorical features that can be used to encode the groups that the record belongs to. Categorical features must be encoded as a 0-based sequence of positive integers. For example, the categorical domain {R, G, B} can be encoded as {0, 1, 2}

The data is written to .json files and uploaded to the AWS S3 bucket specified at the beginning of the notebook.

Next a SageMaker Estimator object is instantiated with the latest DeepAR forecasting algorithm as the model image and the ml.c4.2xlarge training instance. Next the hyperparameters listed in section 2.3 Algorithms and Techniques and the proper data_channels are set for the model. Finally the model training begins. Once the model training is complete, the test:RMSE is automatically calculated and reported by the model for review.

While the test metrics are readily available after training, an endpoint needs to be constructed in order to use the model to make predictions. To do this we create a SageMaker RealTime-Predictor object that accepts a json file as input. The function encode_request is used to encode the context window data from a pandas data frame into a json file that is used as the input into the predictor object. The encode_request function is used inside the get_prediction function, that takes the stock ticker, the prediction date, the stock data frame, the predictor object and a list of the names of the features in the dynamic_feat set as inputs and returns a data frame with the actual target and the predicted target for the specified date. Finally, the models performance can be calculated for each stock. The get_accuracy function takes a ticker as an input and calculated the accuracy of the models forecasts for the entirety of the validation data set: ”2019-01-01” – “2019-12-31”. This function uses the get_prediction function to create a list of actual targets and predicted targets for the specified ticker over this date range. Then it uses the accuracy_score function from the sci-kit learn library to calculate the model's accuracy over this period.

3.3 Refinement

The model that was trained throughout this project went through multiple iterations of refinement in an attempt to increase its performance. The model was originally trained with just the **price** and **volume** data as the dynamic features in the dynamic_feat set, the default hyperparameters, a context window of 50 and a prediction window of 1. The model was first trained with just 10 epochs to ensure there were no errors and that the loss decreased with the number of epochs. Once the model was performing as expected, the number of training epochs was increased incrementally.

Each model was tested over the year-long validation set for a selection of with the following results:

Epochs	**Accuracy**	**Test: RMSE**
10	0.5516	0.742048647917
20	0.6500	0.708370957815
30	0.7387	0.689915893223
40	0.7460	0.726889025133
50	0.7460	0.701382485294

Increasing the Epochs of training improved the model significantly up until the 40 Epoch mark. After that the accuracy of the model plateaued at about 74.60% and the RMSE started to increase.

The next step was to include additional data, the stock price indicator data. This data was included in the form of 7 principal components that preserved 95% of the variance from the range of technical indicators calculated from the price data. This model was trained with price, volume and the principal components as the dynamic features in the dynamic_feat set. These additional features only yield a few more correct classifications, resulting in a marginally improved accuracy of 74.76% and a RMSE of 0.720. With this model was used for hyperparameter training in which 30 different models were tested. This model was optimized to find the minimum test:RMSE value. The minimum value of all 30 models test was 0.5435. The hyperparameter tuning process is discussed further in section the next section, 4.1 Model Evaluation and Validation.

4 Results

4.1 Model Evaluation and Validation

The model was tuned with AWS's hyperparameter tuner object. This tuning process searched for the optimal hyperparameters to minimize the Test:RMSE of the model. These are the final hyperparameters and their optimal values.

Hyperparamter	Type	Value
time_freq	FreeText	D
prediction_length	FreeText	1
num_layers	Integer	1
num_dynamic_feater	FreeText	Auto
num_cells	Integer	177
mini_batch_size	Integer	244
learning_rate	Continuous	0.000104
epochs	Integer	41
early_stopping_patience	Integer	40
dropout_rate	Continuous	0.022
context_length	Integer	59
_tuning_objective_metric	FreeText	test:RMSE

The model with these hyperparameters yielded a minimal test:RMSE value of 0.543473. While these values minimized the test:RMSE, the highest average accuracy the model was able to achieve over the sample test stocks was capped at 74.76%. It was unable to predict the stock movements any more accurately with additional model tuning, despite the fall in the RMSE automatically produced by the DeepAR algorithm during the training process.

The robustness of this model is self-evident not only through the process of development but also though the performance. This model was trained through one of the most sophisticated time series forecasting algorithm available. Through the use of AWS infrastructure and computing power, I was able to train the model on 491 stock price data frames scraped from yahoo finance. The use of a deep recursive neural network allowed the model to train on 491 different stock price behaviors over a nine-year period.

In addition to this, the model was refined with a validation set and assessed with a collection with multiple stocks across various sectors of the economy. The model proved to be quite accurate across all of the stocks that it was tested with, achieving a maximum average accuract of 74.66%, with accuracies for individual securities ranging from 66% to 82%.

4.2 Justification

The model successfully solved the problem statement defined at the beginning of the project. The purpose of the project was to train a deep learning model to accurately predict the movement of stock prices, simply using free data scraped from the web. The test:RMSE loss function used to during training was minimized through the hyperparameter tuning process. However, after a certain point, further decreases in the test:RMSE did not result in an improved model accuracy. With regards to the accuracy of the prediction, the model performed well. The model did a relatively good job predicting the stock price movement, predicting the correct trade action [buy, hold, sell] approximately 75% of the time. This performance almost directly achieved the benchmark goal defined in the project proposal. My goal was to achieve a similar performance to the artificial neural network that was able to accurately forecast 75.74% of the stock movements from a sample of the Istanbul Stock Exchange. Not only was the model able to achieve a comparable result, it was able to do so for nearly every stock on the S&P 500 index, sufficiently solving the objective problem of this project.

5 Conclusion

5.1 Summation

This project followed the tradition machine learning workflow. First the problem was identified and defined. The goal of the project was to create a deep learning model to predict the movement in stock prices. Once the problem was properly defined, the next step was to collect the right data. This was done by scraping S&P 500 OHLCV stock data with pandas-datareader from Yahoo Finance. Then the data was preprocessed into a single data frame hierarchically indexed by datetime and ticker. The first 7 principal components of a collection of technical indicators were used as addition data to compliment the price and volume data used as inputs. This addition only marginally increased the performance of the model, most likely because all of the useful signal is already built into the basic price and volume data. The model was able to achieve 75% accuracy over the test data. This however proved to be the maximum accuracy the model was able to achieve after further attempts of development and refinement. Nevertheless, the model was able to successfully and relatively accurately predict the price movement of S&P 500 securities.

5.2 Improvement

While this is a successful start, there is a lot of opportunity for improvement of this model. First, of all, additional data other than price data and technical indicators can be used as inputs into the model to provide more signal for stock price movement. Sentiment analysis of news, twitter and stock twits, for example, all be helpful data sources to find additional price movement signal. Fundamental financial data and macro-economic data could also be used help develop a more accurate model that reflects the larger economic climate in which the trading and stock movements are taking place.

Furthermore, the endpoint and user interface could also use improvement. While it is able to use historical context windows of price and volume data to output a target trading action, this is not helpful in practice. In order to use this model, it must be able to integrate with trading software such as Quantopian, Alpaca or Zorro in order to actually execute real or paper trades.

Furthermore, this model only performs well on the task that was designed for it. While it may accurate predict price movements of a certain threshold, it is not optimized for real-world performance. Further steps in model refinement would include the use of back testing software to such as Quantopian’s Zipline or the Zorro Project’s trading platform interface. Back testing allows for the model to test in real trading environments and evaluate the risk/return profile of the model’s performance. Ultimately, while the model works well for its designed purpose, there is still a lot more work to be done in order to prepare it to profitably trade in live markets. Furthermore, the model will need to learn and predict ‘online’ taking in new data every day and update its parameters with the constantly evolving market.

5.3 Reflection

This project was an incredible challenging yet rewarding endeavor. I am quite satisfied with the results and the process that was done to achieve them. I am very appreciative of the qualityy of the products and services that Amazon Web Services offers. From the state-of-the-art machine learning and deep learning algorithms to the incredible powerful and accessible virtual computers, I would not have been able to undertake such a task with out these tools. I am also appreciative for the Udacity program, of which this capstone project is a part. Within the past two months Udacity has allowed me to develop the skills necessary to complete an entire machine learning project from concept to product. While there is still much more to learn, and much more that can be done to improve upon this project, I think that this project justly portrays the effort and time that I have invested into the Machine Learning Engineer Nano Degree and accurately displays the knowledge and experience that I have collected.

This project has revealed to me the good, the bad and the ugly that goes into the development of a machine learning model. From the acquisition and preprocessing of the data to the tuning and deployment of a working model (as well as all of the coding and development skills that are necessary to pick up along the way), the project has illuminated for me the sheer scale of the complexity and frustration, as well as the pride and elation that are inherent in the development process of a machine learning engineer.