Group 2: Daniel Rozenzaft, Yaroslava Shynkar, Jonathan Kelaty
Project: Stock Movement Prediction

TradeAide: Automated Stock Market Prediction

Checklist

- ✔ Project report
- ✔ GitHub Link
- ✔ Video

Problem Description

For our final project, we will be using various machine learning algorithms to predict future stock prices. We will also be using reinforcement learning to predict the optimal times to buy and sell stock shares. We accomplished this by applying LSTM, SVM, RF, and Q-Learning to daily stock price data.

The challenge in this application lies in the unpredictability of the stock market. Due to the complex, highly volatile nature of the market, many existing models fail to take into account all of the variables that affect price movements, and thus struggle to adapt to changing market conditions. Massive representative samples—high frequency trading data collected over long periods of time—create very large state spaces.

Existing literature employs a wide array of methods for local extrema prediction, including (and certainly not limited to) support vector machines (SVMs), random forest (RF), reinforcement learning (RL), and long short-term memory neural networks (LSTMs). We implemented several of these and compared the price and investment strategy predictions that they produced. We also like to quantify the performance-boosting impact of sentiment analysis techniques on company news or tweet data; integrated sentiment analysis metrics—based on financial news title and content data—as another feature for our models. For our novel approach, we made key additions to our Q-Learning algorithm, including a volatility weighting and risk-setting mechanism. As for LSTM we compare whether the true and predicted data depict the same trend (e.g. the values go up or down for a certain day). The models that are able to predict the stock movement are labelled as safe and could be integrated for the portfolio of a certain company, while the models that cannot pick up the trend with high confidence are suggested to

be monitored by humans, or by assuring that the trader will not be able to trade more than ⅓ of its stock at a time.

Team Roles

The division of roles in our project was primarily based on algorithms: each of us worked simultaneously and (mostly) individually on incorporating one of the following approaches with our data: RL (Daniel), SVM and RF (Jonathan), and LSTM (Yaroslava). After creating our initial implementations, we collaborated on merging our algorithms with the NLP, fleshing it out as much as we could before the final deadline.

Support Vector Machines (SVM) and Random Forest (RF) for Stock Price Prediction

Hiba Sadia et al. found that SVM and RF were two strong predictors of stock price trends. Thus, we elected to run each of these models on our data.

Approach

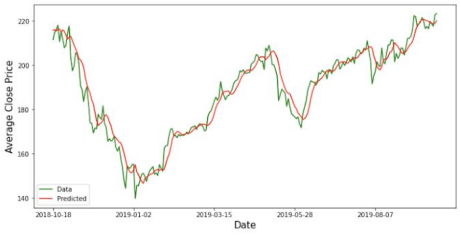
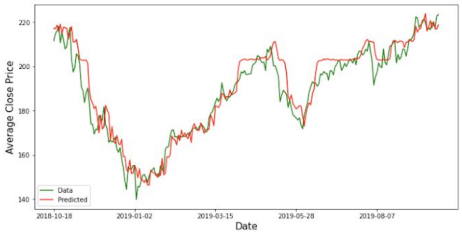
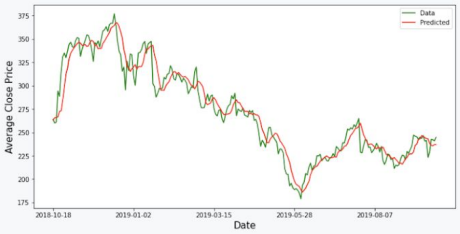
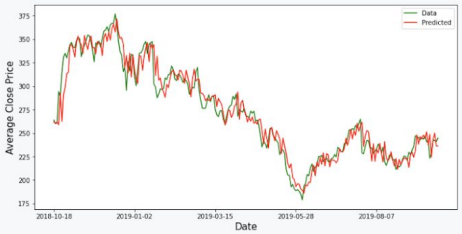
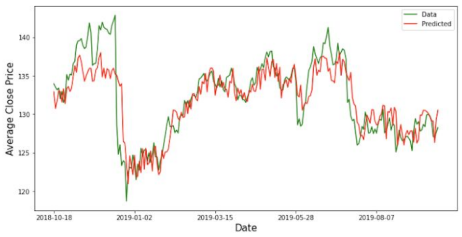
We utilized the standard implementation of SVM and RF provided by sklearn. The main implementation detail that needed to be figured out was the feature set, which would be used for training and prediction. We incorporated the following features in our SVM and RF predictors:

1. Running [up to] 5 day average of the adjusted close price
2. Running [up to] 5 day average of the news sentiment
3. Volatility–ratio of previous two days' adjusted close price

We also considered other metrics, including price trend, trading value and volume, and relative strength (RSI). Ultimately, our performance could have seen some improvement with these added features. We highlight this area for future work.

Evaluation and Results

We evaluated each of our models on the same set of stocks: Apple (AAPL), Tesla (TSLA), and Johnson & Johnson (JNJ). The table below shows our results for both algorithms across our range of stocks. Our main takeaway was that the SVM model's predicted stock values formed a smooth approximation, which closely mapped to the actual stock prices. RF was far more sensitive to the rapidly changing day-to-day stock values, while arguably more closely resembling real-world behavior.

Ticker	SVM	RF
AAPL		
TSLA		
JNJ		

We evaluated the error in our predictive models using the difference between the predicted and actual stock prices. One minor improvement that we considered was incremental training: adding the previous instance to our training set in order to improve future predictions.

An issue that we observed was that our models did not generalize well enough when we tested them on newer training data. When we observed the data, we noticed a general trend of rapid stock price increases over the past few years. Most of the training data came from prices recorded several years before that; in essence, our models were making predictions for an entirely different market. As a general intuition, as Tesla's popularity has increased in mainstream media and culture, investors are likely to view it as a safer stock, and therefore treat it differently over time. This would effectively invalidate certain historical data. Another market trend that deviated from previous patterns was the COVID recession, beginning in March 2020. SVM and RF failed to generalize to this data: our training data did not train the model to predict prices during a global pandemic! To clarify, the historical data could still be considered useful, but it likely requires a more sophisticated feature set, which would treat certain trends with more nuance and take other external factors into consideration.

Long Short-Term Memory for Stock Price Prediction

Long Short-Term memory (LSTM) is an artificial recurrent neural network that can learn the order dependence in a sequence. Predictions made by LSTM are dependent on previous data. LSTM consists of a cell, an output gate and a forget gate. LTSM can learn the important input because of its gate controllers. It is commonly used in language and speech recognition, as well as time series analysis. In our project we used LSTM to predict the adjusted closing prices of the market.

Related Work

Nabipour et al. compared predictions made by algorithms like Decision Tree, SVM, RF, ANN, K-NN, Logistic Regression, and LSTM. They found that for continuous data, LSTM and RNN tend to outperform other models. The authors inputted ten technical indices (SMA, WMA, MOM, STC, STK, RSI, SIG, LWR, ADO, CCI) into their models to make predictions. The LSTM was trained based on the data of up to 30 trading days. The following hyperparameters were used: "Hidden Layer Neuron Count" = [500], "Number of training days" = [1,2,5,10,20,30], "Neuron Type" = "LSTM", "Activation Function" = ["Softmax", "Tanh"], "Optimizer" = "Adam", "Learning rate" = [0.00005], "Beta_1" = [0.9], "Beta_2" = [0.999]. The following hyperparameters were used for training stop conditions: "Early stopping", "Monitoring Parameter" = "validation data accuracy", "Patience" = [100], "Max Epochs" = [10000]. All coding is done using Scikit Learn and Keras library. The evaluation shows that LSTM model's accuracy is 0.86 for the diversified financial sector, 0.87 for the mineral sector, and 0.83 for the petroleum sector.

Wei Du also used LSTM to predict stock prices, establishing that historical information is the basis of investment decisions. The attention layer should be used to overcome the limitations of LSTM (e.g. lag of prediction). The author proposed an attention-based LSTM model that consists of four layers: the input layer, hidden layer, attention layer, and output layer. The input layer prepares the data for LSTM. The hidden layer connects the LSTM unit to the line model network. The attention layer takes into consideration the weights of the feature vector. The output layer shows the computed results. For training, small batch gradient descent was used to accelerate the speed of convergence.

Bing et al. extracted Twitter data and applied sentiment analysis techniques in order to predict stock market movements. The authors classified tweets (a combination of several words and phrases) into Positive+, Positive, Neutral, Negative, and Negative– sentiment categories. They applied the chi-square test and adjusted residual to identify the patterns between public opinion and stock market prices. After text extraction and processing, the authors were able to predict IT industry stock prices with an accuracy of 76.12 percent.

Approach

To analyze stock movements, we picked three companies each representing different industries. We used daily closing price data from Tesla (auto manufacturing), Apple (technology) and Johnson & Johnson (pharmaceutical and consumer healthcare) to train and evaluate our models. We extracted prices ranging from January 1, 2015 until December 31, 2019. Finally, we matched company tickets to a financial news dataset in order to compute a single news sentiment value. The focus section on the NLP component further details this process.

The first step to create the LSTM model is to prepare the data (input layer) so that the hidden layer can be implemented. The prepForLSTM() function creates the aggregated input. Our input layer is built on the previous three weeks of pricing data, aiming to predict the next day's price. The columns used to create the preparation layer are "open", "high", "sentiment", and "adj close." The processed data is scaled and split into a train set and test set.

We used the hyperparameters outlined by *Nabipour et al.* to train our LSTM model. We had to limit the maximum number of epochs to 50, as the time for training would scale with the number of epochs allowed. We attempted to reproduce the results of the state-of-the-art to the extent that

we could. We ran the Sequential model with parameters “Neuron type” = “LSTM”, ‘monitor’ = “val_loss”, “verbose” = 1, “patience” = 100 epochs, “learning rate”= 0.00005, “beta_1” = 0.9, “beta_2” = 0.999, “batch_size” = 8, “validation_split” = 0.2, “callbacks” = “early stopping”. Overall, our sequential LSTM model had a total number of parameters equal to 1,010,501.

In designing the code, we used the Keras and Scikit-Learn libraries, as well as the references provided by Geron in *Hands-On Machine Learning*.

Experiments and Evaluation

We trained our LSTM model to predict the adjusted closing price values of APPL, TSLA, and JNJ. Our results are explained below.

LSTM Model for Apple

In order to obtain the optimal model, we used the hyperparameters described in the last section. As a result, the loss function used in the regression problem is Mean Absolute Error (MAE). MAE loss is the average of absolute differences between the actual and predicted values. The calculated MAE for APPL was 0.0067. Mean Squared Error (MSE) loss was also calculated as the average of the squared differences of actual and predicted data. For AAPL, the final MSE equated to 9.4305e-05. Other error values follow:

Final value of cost function for cross-validation data

val_loss = 0.0149

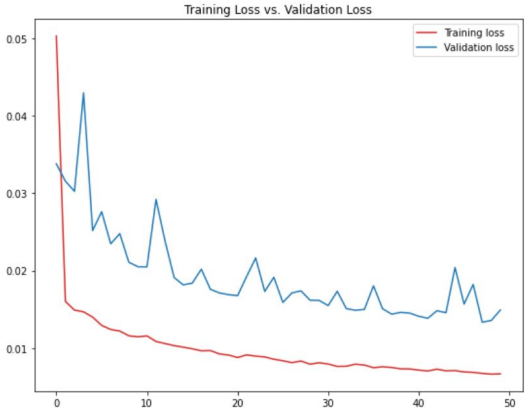
MSE of cross-validation data:

val_mse = 3.6786e-04

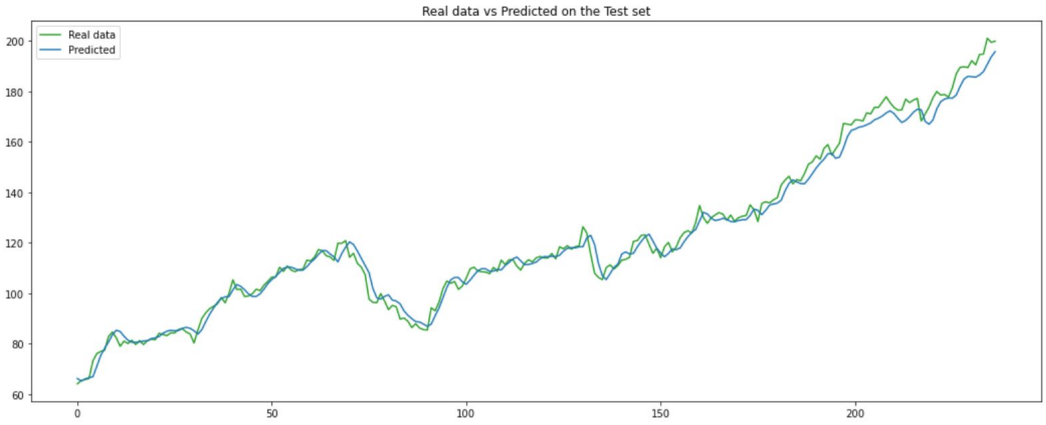
MAE of cross-validation:

val_mae = 0.0149

The graph below shows the training loss and validation loss of the LSTM model when run on APPL.



From this graph, we can clearly see that validation loss is higher than training loss. The prediction on the test dataset gave us a Root Mean Square Deviation (RMSE) equal to 3.60. The relationship between scaled true and predicted data is shown below.



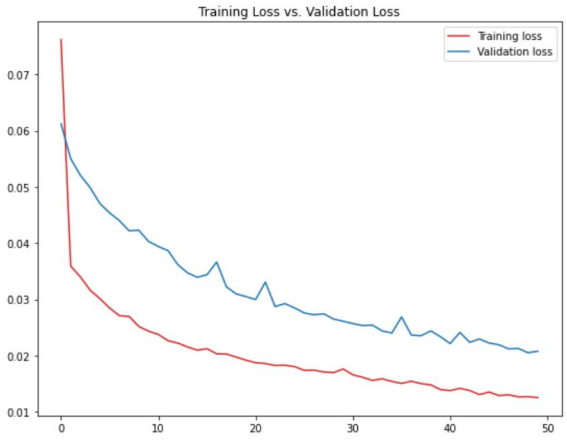
Here, we see that the predicted data (blue color) slightly underestimated the stock price values, predicting prices lower than the actual data.

Our results for train and test data are shown in the following table:

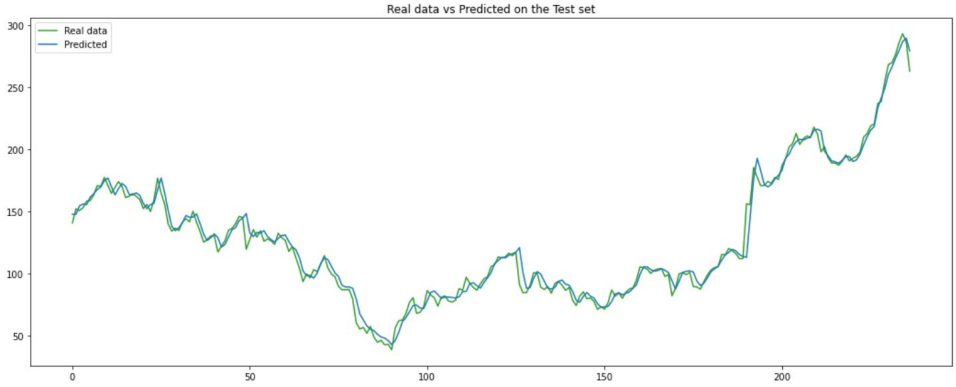
Train Loss	Train MSE	Train MAE	Test Loss	Test MSE	Test MAE
0.0080	0.0001	0.0080	0.0140	0.0003	0.0140

LSTM Model for Tesla

The model was trained on TSLA with the same hyperparameters. Training the model led to an MAE value of 0.125 and an MSE value of 3.2459e-04. Cross-validation led to a data loss of 0.208, an MSE of 9.1789e-04, and an MAE of 0.0208. The following graph illustrates that validation loss is higher than training loss:



After making the prediction on the test set, we obtained a test set RMSE of 6.52. Superimposing the predicted stock prices on the real ones shows that scaled real and predicted data are nearly identical.

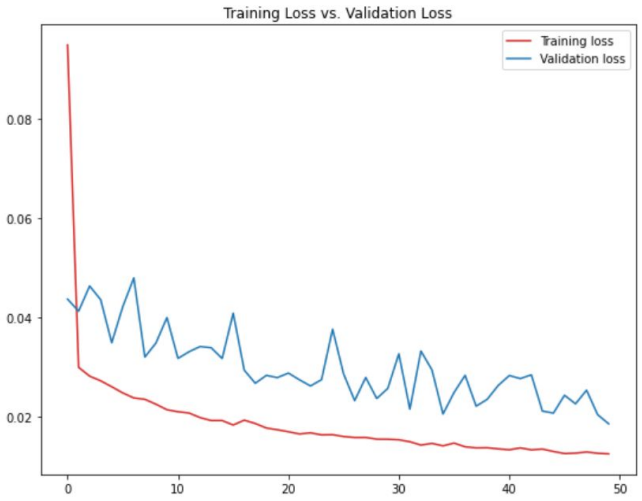


These were our results for the training and test data:

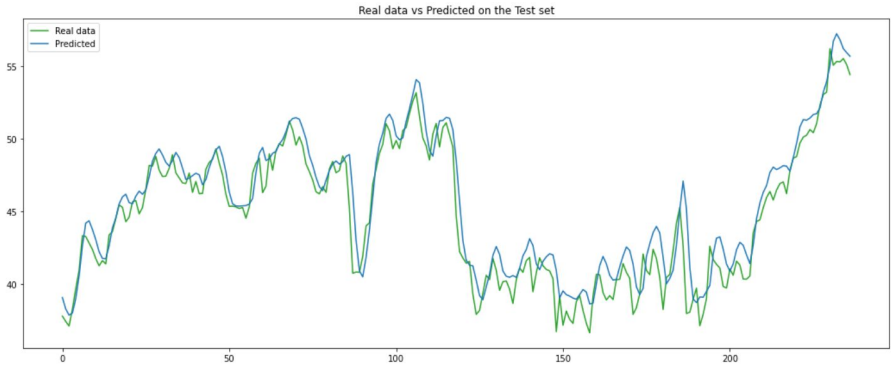
Train Loss	Train MSE	Train MAE	Test Loss	Test MSE	Test MAE
0.0136	0.0004	0.0136	0.0151	0.0005	0.0151

LSTM Model for Johnson & Johnson

The JNJ model was trained using the same hyperparameters as the previous two. Training the model led to an MAE of 0.125, an MSE of 2.9869e-04, a cross-validation loss of 0.0185, a cross-validation MSE of 8.5810e-04, and a cross-validation MAE of 0.0185. The JNJ model also exhibited a higher validation loss than training loss.



After making predictions on the test dataset, we calculated a test RMSE of 1.49. Once again, we saw that the scaled actual prices are less than scaled predicted prices.

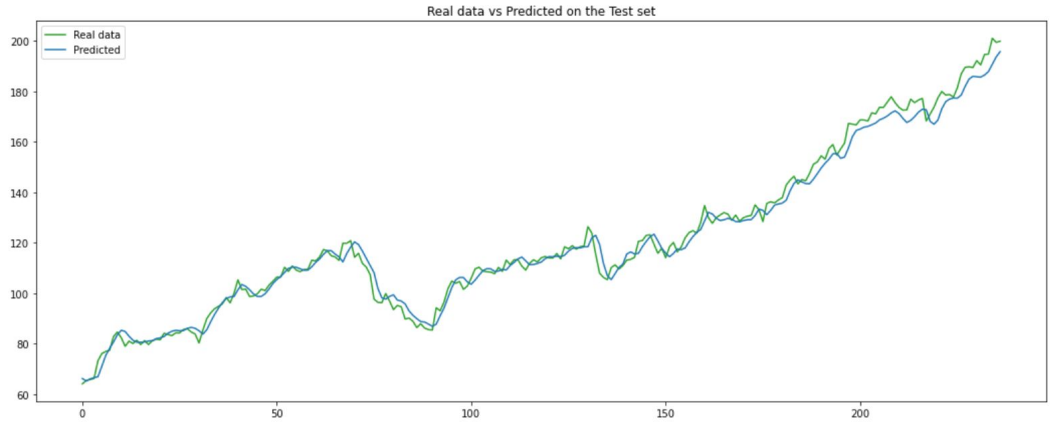


Finally, our results for train and test data follow:

Train Loss	Train MSE	Train MAE	Test Loss	Test MSE	Test MAE
0.0136	0.0004	0.0136	0.0195	0.0007	0.0195

Discussion of Results

Results for the true and predicted adjusted close values for Apple look very promising. Even though we had to scale the data for the LSTM, we can still see from the graph in the previous section that the model is able to predict the trend (whether the price will increase or decrease) with high precision:



Since the model underestimates prices, our LSTM-informed buying and selling recommendations would prevent a trader from losing money.

Results from Tesla predictions show that scaled predicted values are higher than scaled real values. This occurs mainly because we trained our model using daily prices dating between 2015 and 2019. Over that period, the Tesla stock's price doubled. When run on 2020 testing data, the model expected the trend to continue to be positive. This should be taken into consideration when using the LSTM to create a TSLA portfolio. TSLA's high volatility complicates the prediction process. In our project's reinforcement learning component, we devised a novel method to weight our agent's decisions on stock volatility. This will be discussed in the RL section. Another cause of error may be that TSLA's media coverage tends to be biased in favor of the company. We discuss possibilities for future work on combating bias in sentiment values in the NLP Focus section. For the moment, we would not recommend using LSTM results to make automated suggestions for buying and selling TSLA stock. We would prefer to wait until it is able to predict trends without being as sensitive to volatility.

Results from predictions on Johnson & Johnson also reveal that the scaled predicted values exceed the scaled actual values. This could harm bullish traders, who could conclude a false rise in the stock price from our LSTM results. In our RL section, we discuss our implementation of a user-set risk score, allowing the Q-Learning algorithm to match their behavior and preferences.

Lessons Learned

LSTM performs well for both continuous and binary data. ANN performance is far more dependent on whether labels are binary or continuous (performance is much stronger on binary data). Early stopping is a great way to stop iteration as update values approach zero. We also learned how to perform supervised learning on stock price data, and how to analyze news using

TextBlob and FinBERT. FinBERT uses dictionaries to train its transformer, allowing programmers to select a dictionary and retrain the transformer to detect specific language and sentiment patterns.

Challenges and Unachieved Goals

Even though the LSTM model performed quite well, it was hard to execute the hyperparameter tuning operation. The runtime for 1000 epochs with batch size equal to 4 is around 30 minutes. While trying to list all hyperparameter combinations—including the number of training iterations (trading days), different batch sizes, and numerous activation functions—the kernel would shut down after running a certain number of computations. Another challenge was implementing the FinBERT transformer for news sentiment analysis. It requires cuda, which is not available for Intel processors. Running FinBERT without parallel computing took several hours without showing results. Training the ANN to predict price trends posed another challenge. The model performed poorly, and hyperparameter tuning returned an optimal model with very low accuracy. Therefore, the ANN was largely excluded from this report. To compensate, we created an ANN model (ANN_predict_sentiment.ipynb) to predict news sentiment using moving averages, close values, adjusted close values, and open values. Using this network, we can fill in missing sentiments and transform all of our data into the ten technical indicators in an effort to improve predictions in the future.

LSTM: Conclusion

Our findings showed that training an LSTM with a news sentiment feature made stocks from some stocks (namely tech) easier to predict. Therefore, LSTM is a strong predictor of safe

stock prices, and thus is capable of creating automated suggestions on when to buy and sell certain stocks. Future iterations of our model could represent the scaled predicted values using categorical values: +1 (prices will rise), -1 (prices will fall), and 0 (prices will not move, or stay very close together). Other opportunities for future work could involve experimenting with the layers used for LSTM, or finding a dataset that can more accurately predict (with less bias) the sentiments of market makers. Another path highlighted is to create the Trend Deterministic Data Preparation Layer that consists of ten technical indices, volatility and news sentiment components and converting them into binary values to obtain a better prediction model.

Reinforcement Learning for Optimal Trade Prediction

Reinforcement learning (RL) is built on Markov Decision Processes (MDPs), its mathematical foundation. RL agents make sequential decisions by observing the state of their environment, taking an action, observing its reward, and transitioning into the next state. During the training period, the agent learns an *optimal policy function*, which determines the best action to take, a, given that the environment is in state s. The optimal policy function aims to maximize the long-term reward of each state-action pair. In the automated stock trading application, the agent observes the state of the market and decides to buy, sell, or hold the stock(s) in its portfolio. Its reward function is return on investment (ROI).

Related Work

The state-of-the-art in automated stock trading using reinforcement learning involves deep RL techniques. These include critic-only (the most common), actor-only, and actor-critic approaches. Critic-only approaches, such as the deep Q-learning (DQN) algorithm applied by

Jeong and Kim, are trained using only a single stock or asset. DQN retains the basic premise of the Q-learning algorithm. The agent learns a Q-value matrix, which dictates the expected payoff of taking action *a* while in state *s*, and selects the action associated with the highest value during the testing process. DQN adds a neural network in the Q-value approximation phase: it minimizes the error between estimated and target Q-values across state transitions. Critic-only approaches are practical for discrete action spaces, which we consider in our project.

Rather than learning and minimizing error in a Q-value matrix, actor-only approaches use neutral networks and recurrent RL to learn the optimal policy directly. The policy is a function that informs the probability of taking action *a* (in a continuous action space) while in state *s*. *Deng et al.* have shown that the deep direct reinforcement strategy led to risk-adjusted gains of up to 20 percent on Chinese stock index futures and commodities.

The actor-critic method is the most recent of the three. It combines the actor and critic approaches by learning actor-informed optimal policy probabilities and estimating a critic-informed value function to assess the actions recommended by the actor, as illustrated by Li, Rao, and Shi. In general, actor-critic is able to handle large portfolios as well as other complex use cases, such as game playing. *Yang et al.* expanded on actor-critic by applying Advantage Actor Critic (A2C), which uses an advantage function estimated by the critic network to reduce policy gradient variance. The team combined A2C with Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO) to create an ensemble method that selected the highest-performing of the three, evaluated based on risk-adjusted return.

Approach, Evaluation, Performance, and Results

For our project, we took the critic-only approach, implementing the Q-learning algorithm from scratch and running it on our data. Following the general formula, we initialized a Q-matrix of zeros and updated it with the reward of each iteration. Our state space included three categorical features: price trends (up, same, down), trading volume (low, medium, high), and sentiment (negative, neutral, positive). The latter two were categorized according to the 33rd, 66th, and 99th percentiles of their values in the training data. The rows of the Q-matrix consisted of all three-parameter combinations of states. Our action space consisted of the three possibilities for a stock trading agent: buying, holding, and selling. These were the columns of our Q-matrix. In the cases of buying and selling, our reward function was the next-day profit of the agent's action. In the case of holding, we deviated from this metric (which would have always yielded zero) by creating a novel hyperparameter to represent the user's risk level, L ($0 \leq L \leq 100$). The risk level is essentially a penalty on holding the stock. Higher risk levels bias the training algorithm against holding, while lower risk levels make it more conservative.

The following equations represent the reward function in each of the three cases. The variable C refers to the constant number of contracts traded at a time (set to 1 by default; 1 contract = 100 stock shares). The variable P_t refers to the price of the stock on trading day t.

$$Reward_{buy} = C(P_{t+1} - P_t)$$

$$Reward_{hold} = -L$$

$$Reward_{sell} = C(P_t - P_{t+1})$$

Q-values were updated with each iteration in accordance with the standard formula in Sutton and Barto:

$$Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$$

Here, α represents the learning rate (the factor by which Q-values are modified), R represents the observed reward, γ represents the discount factor (an exponential weight that balances short-term and long-term reward, where $\gamma = 0$ implies that the agent focuses only on immediate reward and $\gamma = 1$ implies that the agent aims for purely long-term gain), and $\max_a Q(S', a)$ equals the current maximum Q-value associated with taking the action a (the highest Q-value between buying, holding, and selling) at the next state S'. Actions were selected based on this greedy maximum with probability $1 - \epsilon$, and selected randomly with probability ϵ.

```
['hold', 'hold', 'buy', 'sell', 'hold', 'hold', 'hold', 'buy', 'sell', 'hold', 'buy', 'buy', 'buy', 'hold', 'sell',
'buy', 'sell', 'buy', 'sell', 'sell', 'buy', 'sell', 'buy', 'sell', 'sell', 'buy', 'sell', 'hold', 'buy', 'buy', 'bu
y', 'sell', 'sell', 'buy', 'sell', 'buy', 'buy', 'hold', 'sell', 'sell', 'sell', 'buy', 'sell', 'hold', 'hold', 'hol
d', 'buy', 'sell', 'buy', 'sell', 'hold', 'buy', 'sell', 'buy', 'sell', 'buy', 'sell', 'hold', 'hold', 'buy', 'buy',
'sell', 'buy', 'buy', 'sell', 'sell', 'buy', 'sell', 'sell', 'hold', 'buy', 'sell', 'buy', 'buy', 'buy', 'hold', 'sel
l', 'sell', 'buy', 'sell', 'sell', 'hold', 'hold', 'buy', 'buy', 'sell', 'buy', 'sell', 'buy', 'sell', 'sell', 'buy',
'buy', 'sell', 'sell', 'hold', 'buy', 'buy', 'sell', 'sell', 'buy', 'sell', 'hold', 'hold', 'buy', 'sell', 'buy', 'bu
y', 'sell', 'sell', 'buy', 'sell', 'hold', 'buy', 'sell', 'buy', 'sell', 'hold', 'buy', 'buy', 'buy', 'sell', 'buy',
'sell', 'buy', 'hold', 'sell', 'sell', 'buy', 'buy', 'hold', 'hold', 'hold', 'sell', 'buy', 'hold', 'sell', 'sell',
'sell', 'buy', 'sell', 'hold', 'buy', 'sell', 'buy', 'sell', 'hold', 'buy', 'buy', 'sell', 'sell', 'buy', 'buy', 'bu
y', 'hold', 'buy', 'hold', 'sell', 'buy', 'sell', 'sell', 'sell', 'buy', 'sell', 'buy', 'buy', 'sell', 'sell', 'sel
l', 'buy', 'sell', 'buy', 'sell', 'buy', 'buy', 'sell', 'buy', 'buy', 'sell', 'buy', 'hold', 'sell', 'buy', 'sell',
'sell', 'sell', 'hold', 'buy', 'buy', 'sell', 'sell', 'hold', 'hold', 'hold', 'buy', 'buy', 'buy', 'sell', 'buy', 'ho
ld', 'hold', 'sell', 'sell', 'sell', 'buy', 'buy', 'sell', 'sell', 'hold', 'hold', 'buy', 'sell', 'buy', 'buy', 'bu
y', 'sell', 'buy', 'hold', 'sell', 'sell', 'buy', 'sell', 'buy', 'sell', 'sell', 'buy', 'buy', 'sell', 'sell', 'buy',
'sell', 'buy', 'sell', 'buy', 'sell', 'hold', 'hold', 'buy', 'sell', 'buy', 'buy', 'hold', 'sell', 'sell', 'buy', 'se
ll', 'hold', 'buy', 'sell', 'buy', 'buy']
```

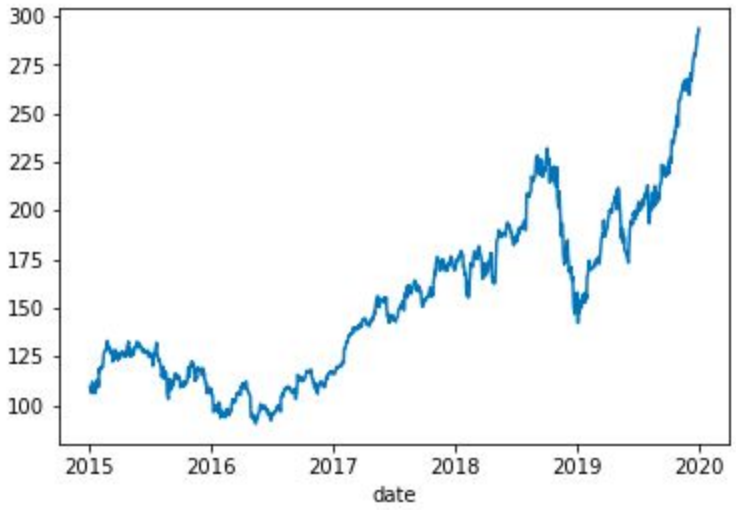
The testing algorithm takes a sequence of actions during each iteration.

The hyperparameter ϵ helps to decrease overfitting by encouraging the training algorithm to explore different paths. Following the decayed-epsilon-greedy method (fewer actions are chosen randomly as the number of iterations increases), we decayed ϵ by raising it to the power of i, the number of trading days elapsed since the start of the training. We found the optimal model for the given stock by employing hyperparameter tuning. We ran models that used all combinations of the learning rates {0.001, 0.01, 0.1}, the risk levels {0, 25, 50, 75, 100}, the initial ϵ values {0.9, 0.99, 0.999}, and the γ values {0, 0.5, 1}, for a total of 3 * 5 * 3 * 3 = 135 iterations. We evaluated each training run using its final ROI at the end of the training period (all trading days between January 1, 2015 and December 31, 2018). The ROI values associated with each hyperparameter were averaged to compute a final "score" for the hyperparameter value.

Many trials suggested that the most consistently high-scoring hyperparameters were $\alpha = 0.001$, $L = 25$, $\epsilon_{initial} = 0.99$, and $\gamma = 0$ when run on most stocks.

After training, the learned Q-value matrix is used by the test function to decide the actions it takes at each trading day. For a given state *s*, the testing algorithm selects an action *a* such that Q(*s*, *a*) is maximized. It then updates *s* to *s'* by calculating the next trend, volume, and sentiment value. Finally, it continues to the next trading day. After traversing the test data set (all trading days in the year 2019), the final ROI is computed. We evaluate testing performance by building out a confidence interval for the sample proportion of the ratio of the final ROI value to the highest ROI achieved in the hyperparameter tuning (model selection) stage. We call this metric the "percent of maximum profit captured," or *p*.

Due to the randomness imposed by ϵ on the learning process, the general sophistication of reinforcement learning agents, and the potential for deriving a false equivalency between the effects of the state of the market on closing prices during the relatively short training period (2015–18) and during the testing period (2019), values of *p* vary substantially, and the 95% confidence interval for *p* often shows a wide range. For some stocks, such as Apple (AAPL), the model yields high returns on the initial investment ($100,000) regardless of hyperparameters, and the range of the confidence interval is relatively small:



AAPL closing price values over time. The agent was trained on 2015–2018 data and tested on 2019 data.

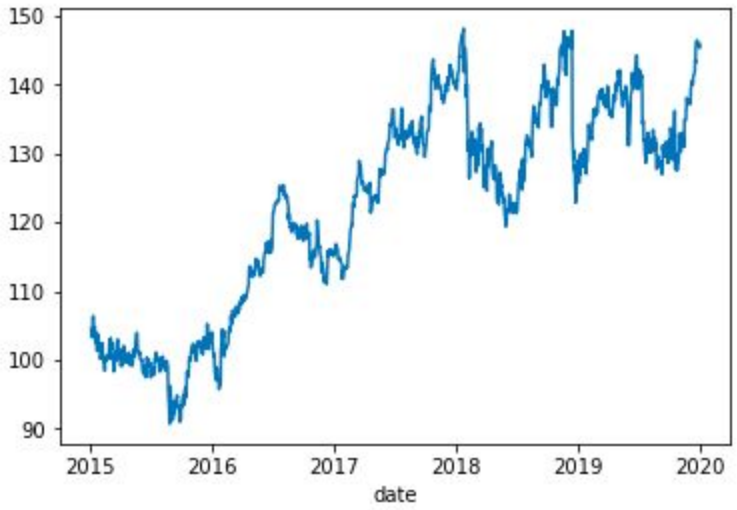
{0.001: 154652.3100789388, 0.01: 152781.55544704862, 0.1: 148304.57543267144} {0: 146179.9233895761, 25: 158830.70367
883754, 50: 153831.40750461156, 75: 152180.99851255064, 100: 148541.03517885564} {0.9: 152137.99933539497, 0.99: 1523
83.44333224825, 0.999: 151216.99829101565} {0.0: 156060.37587483725, 0.5: 149628.11082628038, 1: 150049.95425754122}
{'alpha': 0.001, 'risk level': 25, 'epsilon': 0.99, 'gamma': 0.0}

Final portfolio values for each hyperparameter, averaged over each of its runs (a fraction of the 135 total runs) on AAPL in the hyperparameter tuning stage. The dictionaries represent α, L, $C_{initial}$, and γ, respectively.

Since test runs do not contain randomness, they always yield the same final portfolio value for a given Q-value matrix. The agent was trained with the optimal hyperparameters 25 times, and the percent of the maximum profit captured was recorded. The calculated 95% confidence interval for p in these 25 trials follows:

95% CI: [0.51, 0.87]

The model reached “medium” performance with Johnson and Johnson’s stock (JNJ). Note that the best-performing hyperparameters are different:



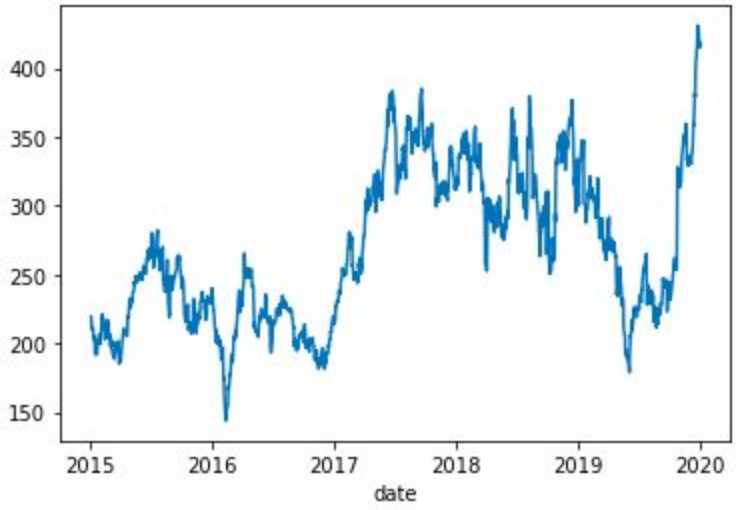
JNJ's clear upward closing price trend between January 2015 and December 2018 is reversed in 2019, replaced with large volatility: an immediate fall, rise, and fall. This may have caused the agent to overfit to the training data, leading to lower profit and *p*, as shown below.

```
{0.001: 106312.89074367947, 0.01: 107210.82482231988, 0.1: 106148.53251139323} {0: 105481.36935763889, 25: 107019.372
19125252, 50: 107524.07576949509, 75: 106321.5572781033, 100: 106440.70553249783} {0.9: 106627.02406141491, 0.99: 105
786.33461846245, 0.999: 107258.88939751519} {0.0: 106119.35668945312, 0.5: 106452.17949761284, 1: 107100.71189032661}
{'alpha': 0.01, 'risk level': 50, 'epsilon': 0.999, 'gamma': 1}
```

The following 95% CI for *p* was computed using the optimal parameters shown in the previous test, {0.01, 50, 0.999, 1}, over 25 training/testing runs:

```
95% CI: [0.3, 0.69]
```

A highly volatile stock like Tesla (TSLA) created even more confusion:



TSLA is widely known for its extreme volatility. Our agent was faced with a tough challenge!

A much higher risk level (75) proved to be a better match for TSLA's unpredictability:

```
{0.001: 117605.90766059028, 0.01: 119398.64352756077, 0.1: 117490.8001030816} {0: 119997.62787995515, 25: 116323.8895
5575449, 50: 116876.84987386067, 75: 120105.88763201678, 100: 117521.33054380064} {0.9: 118412.19740125869, 0.99: 118
831.17570665147, 0.999: 117251.97818332248} {0.0: 118538.82059733073, 0.5: 118173.53156195747, 1: 117782.99913194444}
{'alpha': 0.01, 'risk level': 75, 'epsilon': 0.99, 'gamma': 0.0}
```

The following 95% CI for *p* was computed using the optimal parameters shown in the previous test, {0.01, 75, 0.99, 0}, over 25 training/testing runs:

```
95% CI: [0.21, 0.59]
```

Volatility Weighting: A Novel Strategy

To combat the issue of volatility and help our agent predict sudden drastic price movements before they happened, we devised a novel volatility weighting method. First, we created a 5-day volatility column, which calculated the standard deviation of the previous five closing price values for each trading day (beginning with the fifth trading day). As the training algorithm iterated over the trading days, it tracked the average volatility over time for each state-action pair. After iteration completed, we computed a "volatility-risk factor" (VRF), a rational number between 1 and 2, for each Q-matrix value. The VRF scales based on the numerical distance between the risk level and the average volatility, according to the following equation:

$$VRF_{s,\,a} = 1 + \frac{100 - |\,L - V_{s,\,a}\,|}{100}$$

where L is the global risk level parameter and $V_{s,\,a}$ is the average volatility of closing prices on trading days associated with the state-action pair $(s,\,a)$. Volatilities closer to the risk level receive a scaling close to 2, while volatilities that are very far from the risk level are scaled by a value equal or slightly greater than 1. The values of the VRF matrix were multiplied cell-wise by those of the learned Q-matrix, yielding the weighted Q-matrix on which the model was tested. Unfortunately, repeated trials showed that the performance gain from including the volatility-weighted Q-matrix was somewhat marginal. Future work could approach the volatility issue with a more technical lens, targeted more specifically for the stock prediction application.

Other Challenges and Future Goals

Despite their wide ranges, the means of our resulting *p* values ended up comparing somewhat favorably with the final cumulative return rates attained by *Yang et al.* (shown below):

Fig. 5. Cumulative return curves of our ensemble strategy and three actor-critic based algorithms, the min-variance portfolio allocation strategy, and the Dow Jones Industrial Average. (Initial portfolio value $1,000,000, from 2016/01/04 to 2020/05/08).

The key difference between this result and ours is that it came from an initial portfolio value of $1,000,000, where any positive whole number of shares could be bought at one time. Our agent's major limitation is that it does not optimize the transaction size. It trades one contract at a time by default, and it lacks real functionality to accurately assess what the optimal number of contracts to buy or sell in a given state would be. This probably causes the agent to miss out on a large portion of the available profit. Our primary future goal would be to solve this portfolio optimization problem and incorporate it into our Q-value calculation. Another limitation lies in the simple Q-learning strategy. Employing deep Q-learning—a neural network that would correct Q-value errors using backpropagation—alongside an ensemble strategy would likely result in a significant decrease in the *p* value confidence interval's range and increase in the sample mean. Finally, adding more training and validation data may have helped to improve the Q-value estimates. However, the agent takes a long time to train in its current form, and adding

more data would have further slowed the model selection and testing process. It may have helped to have access to more powerful hardware during the training and validation phase.

Focus: NLP Component

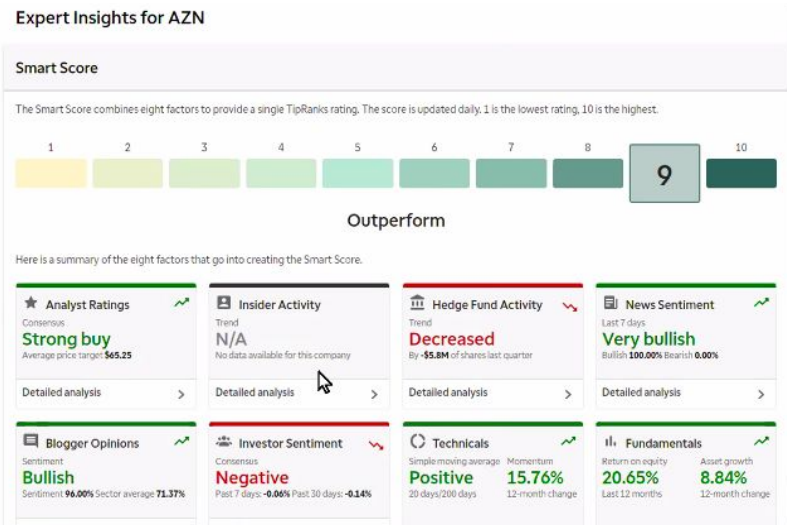
Since our financial news dataset lacked sentiment values, we decided to use the TextBlob package's sentiment polarity function to build out our own news sentiment data, news_sentiment.csv. The news_sentiment dataset computes a single sentiment value for each news article by averaging (equally weighting) the sentiment polarities of its title and content (since it is known that many readers do not read past the headline). All of our models— LSTM, SVM, RF, and RL—incorporate the news_sentiment data. The RL implementation specifically uses a daily sentiment metric, as defined in the news_test script. The daily sentiment function calculates the overall sentiment of a stock on a given day by aggregating the sentiments of the previous week's news articles about that stock and about the market as a whole, weighted by a factor of 0.75 and 0.25, respectively. Individual articles are weighted according to an exponential decay function based on k, the number of days elapsed since the article was published:

$$w = 0.6e^{-\frac{k}{3}}$$

Thus, an article loses influence on the sentiment value as it becomes more outdated. This particular weight function was chosen because it seemed to do a good job of mapping the decay we were aiming for.

Finally, daily sentiment values were computed for each trading day and added alongside the closing price trend and trading volume data. As explained in an earlier section, each of these features was categorized and used to define the states of the market. Thus, our market state space structure consisted of $3^3 = 27$ possible states.

An experienced trader commented that our interpretation of news sentiment and its effect on the market can propagate bias ingrained into the articles themselves. Most of the articles in our dataset seemed to have been written by market analysts, who tend to avoid making negative assessments (recommending that investors sell shares) of companies for fear of losing their inside sources. This recent snapshot from TD Ameritrade's "Smart Score" for AstraZeneca (AZN) illustrates that the opinions of investors and traders may starkly contrast with those of analysts and bloggers:



In the future, we would look to revisit our original idea of using tweets to measure stock sentiment. The average Twitter user is obviously less likely to have personal bias toward a corporation than a market insider. We may also explore tokenizing the articles' content and searching for language that commends corporate management for good decision-making. Our trading consultant suggested that *the presence of explicitly good language* implies more objectivity than *the absence of explicitly bad language*. Our categorization of TextBlob's very small sentiment value decimals probably failed to capture this subtle distinction.

General Conclusion

Despite not yielding high ROI values for all use cases, our Q-Learning solution for predicting optimal stock trades consistently yields positive profit values. In its current form, we would offer it for use by a relatively conservative, low-capital investor who wants to trade one contract at a time. We would also recommend our SVM, RF, and LSTM implementations on our data for an investor seeking to predict future stock price movements to inform their transactions. Future work will be primarily focused on combating bias imposed by stock volatility and bias in financial news.

References

L. Bing, K. C. C. Chan and C. Ou, "Public Sentiment Analysis in Twitter Data for Prediction of a Company's Stock Price Movements," 2014 *IEEE 11th International Conference on e-Business Engineering,* Guangzhou, 2014, pp. 232-239, doi: 10.1109/ICEBE.2014.47.

Deng, Y., Feng, B., Kong,Y., Ren, Z., Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, *28*, 1–12.

Hiba Sadia, K., Sharma, A., Paul, A., Padhi, S., Sanyal, S. (2019). Stock Market Prediction Using Machine Learning Algorithms. *International Journal of Engineering and Advanced Technology*, *8*(4), 25–31.

Jeong, G. and Kim, H. (2018). Improving Financial Trading Decisions Using Deep Q-learning: Predicting the Number of Shares, Action Strategies, and Transfer Learning. *Expert Systems with Applications*, *117*.

M. Nabipour, P. Nayyeri, H. Jabani, S. S. and A. Mosavi, "Predicting Stock Market Trends Using Machine Learning and Deep Learning Algorithms Via Continuous and Binary Data; a Comparative Analysis," in IEEE Access, vol. 8, pp. 150199-150212, 2020. doi: 10.1109/ACCESS.2020.3015966.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). Cambridge, Massachusetts: MIT Press.

Wei, D., "Prediction of Stock Price Based on LSTM Neural Network," 2019 *International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM)*, Dublin, Ireland, 2019, pp. 544-547, doi: 10.1109/AIAM48774.2019.00113.