CU Boulder Capstone L3Harris Adversarial Data Attack Detection Machine Learning Model

Sarthak Shukla, Andrew Wu, Tim Wilson, Ethan Olander
Veda Jammula, Portia Bhattacharjee

May 10, 2023

Abstract

The research conducted in this white paper explores L3Harris Technologies' collaboration with the Senior Capstone project team from University of Colorado at Boulder to develop a dependable and resilient machine learning (ML) model for monitoring satellites, rockets, and space debris by manipulating DJIA, Russell 2000, and Nasdaq proxy data. The main objective is to alleviate the workload of orbital analysts while improving prediction accuracy. The proposed ML model is expected to exhibit resistance against various types of attacks, possess error tolerance, and be easily interpreted by analysts unfamiliar with the model. The ultimate outcome will be a modified neural network model that is both explainable and robust enough to function in adversarial environments. Additionally, the testing injection framework utilized can be replicated and customized to fit the specific conditions and frameworks of L3Harris' ML model customers, providing a valuable resource for future endeavors. The project is consistent with the company's corporate strategies, which prioritize the use of optimal solutions, and includes the establishment of a stand-in domain using stock market data for modeling space orbit detection.

1 Introduction

Our capstone project involved collaborating with L3Harris, a leading defense technology and aerospace company. As part of their operations, L3Harris manages an immense amount of data on objects in space, including satellites, rocket bodies, and space debris. To maintain accurate awareness of these objects' locations, and to make predictions on them, L3Harris relies on machine learning models that process location data at regular intervals.

However, the importance of such models has raised concerns about the possibility of bad data being fed into them. This could enable adversaries to manipulate information about satellites or other space objects for their own purposes, posing a significant security threat. Additionally, ensuring accurate location information is critical when planning for new launches.

Given these challenges, L3Harris tasked us with working towards a solution. Due to the proprietary nature of this data and model, we needed to identify appropriate substitutes for both. Our approach involved using an LSTM model and publicly available stock market data for our simulation. We then simulated continuous retraining using modified sliding windows, referred to as window parsing, and applied various methods to detect anomalies and outliers in the data.

To test the effectiveness of our approach, we generated anomalous data and evaluated how well our outlier detection methods identified it, as well as how our model's predictions improved after removing it. Our findings indicated that our methods were effective in identifying and reporting unexpected data that did not match expected patterns.

Overall, our work has the potential to help L3Harris address potential data quality issues they may face while maintaining accurate awareness of objects in space. Furthermore, our approach could be adapted to other scenarios where machine learning models are at risk of adversarial data attacks.

2 Methods

2.1 Model and Data Selection

Due to our lack of access to the orbital data, our research was conducted on stock market indices. Although they are different types of data, the unpredictable time series nature of the stock data makes it a reasonable proxy. To achieve our purposes of data attack detection we conducted our research using this stock data, as well as a model predicting the stocks' close prices.

The data chosen for our model includes the indices: Nasdaq, Russell 2000, and the Dow Jones Industrial Average (DJIA). Each of these indices are contained in CSV (comma-separated values) files in a folder titled "stock_data" in the root directory of our project, which can be seen on our GitHub project repository. The respective CSV files in this folder are: "djia_2012.csv", "nasdaq_all.csv", and "russel2000_all.csv".

- DJIA CSV: data from 10/31/2022 - 11/10/2022 for date and the Dow Jones Industrial Average value.
- Nasdaq CSV: data from 02/05/1971 - 11/09/2022 for date, open price, close price, high value, low value, adjusted close price, and volume.
- Russell 2000 CSV: data from 09/10/1987 - 11/09/2022 for date, open price, close price, high value, low value, adjusted close price, and volume.

These indices were chosen because a team member had already downloaded and formatted these CSVs from previous unrelated class work. Thus, the choice of our data was due to having readily-available and accurate data, which in turn facilitated our project setup.

The models that were considered for the evaluation of our data manipulations were Recurrent Neural Networks (RNNs), Convolution Neural Networks (CNNs), and Long Short-Term Memory model (LSTM). Initially, CNNs were ruled out because of

the higher capabilities of RNNs in this scenario. However, deciding between RNNs and LSTMs became difficult. Ultimately, the machine learning model chosen for evaluation was the Long Short-Term Memory model.

RNNs are designed to manage sequential data, in which prior events highly influence a prediction of future events. Through the method of backpropagation, an error is passed from the output layer to the input layer, which allows the error between predicted and actual values to be minimized over time using a gradient. The gradient will indicate a general direction to move through the error surface by calculating partial derivatives considering all parameters. However, moving back between several layers and the need for a long chain rule to travel through all parameters in previous layers could lead to a vanishing gradient[1].

This issue of the vanishing gradient is why the LSTM model is the best choice for stock prediction. The LSTM model was developed to memorize information for longer periods of time and is defined as a "deep learning technique for time series predictions"[1]. Therefore, a LSTM is essentially a RNN, however, it contains a memory cell at each point where a recurrent node would be placed in a RNN[10]. The basics of the LSTM are that it is composed of input layers, output layers, hidden layers, and a cell state. The LSTM does not change the flow of information by using solely linear interaction in managing cell state. Furthermore, the LSTM gate mechanism is used to only pass specific information and is also used to delete/modify cell state information[1]. Therefore our research uses the LSTM model for stock prediction due to its ability to memorize and use previous information, as well as avoiding the vanishing gradient issue that is present in RNNs.

2.2 Data Pipeline Overview & Project Architecture

The general flow of the project can be seen in the architecture diagram below. A summary of the process is as follows:

1. Stock data enters the pipeline and is manipulated to become corrupted to simulate adversarial attacks.

2. Following, the data sets with anomalies are continuously retrained using window parsing methods and a judgment window.

3. The training window is trained with anomaly and adversarial attack detection and then run on the judgment window. The methods used are K-Nearest Neighbors, Local Outlier Factor, Hurst Exponent, and Hole Detection.

4. Anomalous entries are deleted and replaced with interpolated values and the cleaned data judgment window is returned back to the pipeline.

5. The cleaned data set now runs through the PyTorch LSTM model for predictions.

6. All results of data manipulations and the LSTM model run are simultaneously outputted to the UI, which is a Streamlit data science dashboard.

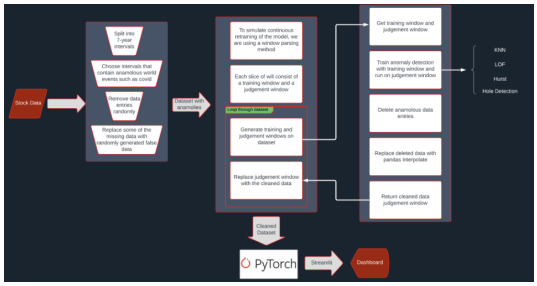

Figure 1: Project Architecture Diagram

The project architecture can be viewed in the GitHub repository link in the appendix. The root of the project is the *L3Harris-Senior-Capstone* folder. In this folder, we have the *LSTM_model* folder, *stock_data* folder, Pipfile, Pipfile.lock, and a README.md. So the *LSTM_model* contains all the data manipulations, the pipeline, the LSTM model simulation, and the Streamlit UI pages.

Inside the *LSTM_model* folder there is *Data_Manipulations* folder, *pages* folder, Home.py, LSTM_sim.py, Pipeline.py files, as well as a README.md. The README describes the project architecture and how to run the dashboard. The *pages* folder contains the 3 pages for Nasdaq, Russell 200, and DJIA where the pipeline and model simulations are run. Furthermore, *Data_Manipulations* folder contains all the data manipulations and simulated adversarial attack Python class files. The LSTM model and simulation run are contained in the LSTM_sim.py file. Moreover, the Pipeline.py file runs all of the data manipulations in our pipeline with simulated retraining of the data through window parsing techniques.

Window parsing refers to something similar to the sliding window technique for nested loops. We first slice the data set into two windows; a training window and a testing/judgment window. The sizes of the windows can be adjusted. After training our detection methods on the training window, we judge the judgment window. Anomalous values are found and replaced with Scipy's interpolate function. Then we replace the judgment window with our cleaned judgment window and slide both windows forward to judge the next set of points. This method does assume that there is an initial training window's worth of data that is correct. This sliding window simulates continuous retraining on our model and acts as the distinction between Novelty Detection and Outlier Detection. Since our model is a secondary component to the bulk of our work, we don't need to actually continuously retrain it. The sliding window simulates how data will come in in real-time and we can save resources by not retraining our model.

This project used the dependencies of NumPy, Pandas, Streamlit, Pylab-SDK, MatPlotLib, SciKit-Learn, SciKit-Learn-Extra, Seaborn, and TorchVision. All these dependencies are managed using PipEnv, which is a virtual environment through which Pip dependencies can be installed. The PipEnv does not have a Python version requirement, however, it does need python versions above 3.6. For Python versions below 3.10, StreamLit will have issues on local-host launch, therefore version 1.2.2 of URLLib3 is installed. With Python versions 3.10 or above, the URLLib3 dependency in the PipFile will need to be deleted so that StreamLit can use a more up-to-date version of URLLib. Dependencies are listed and easily managed in the PipFile, and when *pipenv install* is run, it installs

the dependencies from Pipfile.lock.

2.2.1 Hole Detection

One metric we can judge incoming data on is auxiliary data related to the entry. This can easily reveal whether the data in question should be accepted or not. In the case of a stock market, we can look at the date that data is coming in. Each weekday should have one entry associated with it so we can look for dates that overlap, missing dates, or dates that are weekends. If a data entry is coming in on a Saturday or Sunday, this clearly shouldn't be accepted since stock markets are closed on the weekend. This can be applied to other data sets that have data inputted periodically or more generally, data whose entry must follow certain rules. In this model, we checked for duplicate dates, missing dates, and weekend dates.

2.2.2 Local Outlier Factor

Local Outlier Factor, or LOF, is a novelty detection algorithm implemented in Scikit-Learn. It takes a set of points to build a model from, along with a parameter, k, to specify how many neighbors are used. When used to classify a new point as either an outlier or not, it computes the distance from the new point to the nearest k points the model was built on. It uses these distances to estimate the local density around the new point. If that local density is substantially lower than that of the k nearest neighbors, it is classified as an outlier. In our testing, we found that LOF performed best when we used k equal to the number of points the model was built on.

2.2.3 Hurst Exponent

The Hurst Exponent is a measure of long-term memory of a series. It measures the correlation between data and a delayed copy of that data. It returns a value between 0 and 1. A value between 0.5 and 1 means the data follows a trend. A value of 0.5 means the data follows a geometric random walk. A value between 0 and 0.5 means the data is mean reverting or in other words, returns to an average. In this project we applied the Hurst exponent to successive windows and used the difference between exponents to see whether the trend was changing too fast. When applied to stocks, this measure is more useful for trading since it largely measures linear trends, but it can become particularly powerful when applied to transformed data. If data is expected to be periodic in some form, applying a transform could convert the data into a more linear form in which a Hurst exponent could accurately detect if the data is straying course. In this algorithm we used Hurst as a visual signal rather than pruning values like the other methods. We display green for negligible change in the exponent, yellow for significant changes, and red for extreme changes.

2.2.4 KNN Unsupervised Learning

K-Nearest Neighbors (KNN) unsupervised learning algorithm was implemented using a Kaggle Notebook describing anomaly detection methods for stock market indices. This method uses a distance-based approach to calculate the anomaly score of the data entries, without having to use a ML model. The means of this Kaggle note-

book is to identify anomalous trading days in stock market data.[4]

The algorithm uses SciKitLearn Nearest Neighbors module and SciKit Spatial distance module, along with NumPy. Using $k = 4$ neighbors, the nearest neighbors are evaluated with KNN, and the distance metric is set to Euclidean distance, while the neighbor values are fitted along to the percent change of close price between the previous and current day. With the anomaly scores, a color map of the scores in accordance with each data point will be generated[4]. From this, we have calculated the anomaly scores of all our data entries in the data set, and these values will be displayed in red circles on the plot of all data entries. Following the plotting, the anomalous entries will be pruned from the data set and the deleted values will be interpolated for a better fit.

2.3 Simulated Data Attacks

Data attacks were simulated by randomly generating intervals within each stock market index data file to add random noise to. These intervals were of a random size up to 342 or the average length of a stock market crash. Gaussian noise was used to apply one of five levels of noise: 100, 200, 300, 400, or 500. The random noise value was generated with a mean of 0 and one of these five standard deviations.

Gaussian noise was used to add noise that would be model random variables, as well as be independent and uncorrelated with the data. This way the simulated data attack findings are not specific to just the proxy stock market data. As there was no trend or bias to be assumed for all three stock market indices, we set the mean to 0. The standard deviations allowed for a larger range of noise levels to be simulated, creating more extreme fluctuations in the data. The noise primarily needed to be the right size to be able to be seen in the large data frames.

2.4 LSTM Close Price Prediction Model

The LSTM stock close price predicition model was found on Kaggle as well. The dependencies used to run the model are: NumPy, Pandas, Matplotlib, SKLearn metrics, SKLearn preprocessing, Torch, Torch.NeuralNetwork, Torch.AutoGrad[9].

For our purposes, the LSTM_sim.py file contains the LSTM model build as well as the simulation run process. Therefore, there is an LSTM class as well as a LSTM_sim class contained in the file. The LSTM class uses the neural network module of PyTorch to build the model itself. The model is built with one input layer, 32 hidden dimensions, 2 num layers (stacking 2 RNNs on top of each other), and 1 output layer. This LSTM class sets up all the layers as well as initializes the hidden state with zeroes and initializes the cell state. Moreover, this model creates a truncated backpropogation through time[9].

After the cleaned data frame has been passed into the simulation model, the data frame and the time v.s close price plot are displayed side-by-side in the dashboard. The LSTM model will first fill in missing values using a scalar fit min-max transformation. An important aside is that the LSTM model should not have to fill missing values because our data manipulations will

return an error-free cleaned data set. Now the LSTM model class will be called and generated in the simulation class. Before being trained, the train and test sets are created using NumPy and PyTorch. Then, the model is trained over 100 epochs where the forward pass creates a prediction on the training set while outputting the mean-squared-error for each 10th epoch. On each step the the gradient is zeroed out, followed by a backwards pass, and finally the model parameters get updated. After the model training, the training loss is visualized (outputted to the dashboard) and predictions are made and then inverted. The root mean square error is calculated for train and test scores and the prediction is visualized for the entirety of the time series against the actual, observed data. This final graph is also outputted to the Streamlit dashboard[9].

3 Results

3.1 Data Manipulations

3.1.1 Hole Detection

Hole Detection was applied to all data sets and correctly found holes and anomalous values we had added to the data set. However, it also misidentified holidays as holes. We didn't account the holiday days each year and this resulted in more holes being detected than usual. Depending on the year, these could range in number from 8 to 1. These are influenced by world events such as in 2001 when 9/11 closed the stock markets for 4 days or Hurricane Sandy.

3.1.2 Local Outlier Factor

LOF was applied to the original NASDAQ, Russel 2000, and DJIA data sets, and again to the tainted versions of each. Compared to the other methods, LOF was prone to over identifying outliers. When run on the uncorrupted data, this method identified 43 outliers for the NASDAQ, 29 for the Russel 2000, and 47 for the DJIA, out of the 1762 data points. This means an accuracy greater than or equal to 97.56% for the NASDAQ, 98.35% for the Russel 2000, and 97.33% for the DJIA.

However when tainted data was introduced, this method performed considerably worse. For the tainted NASDAQ 74 outliers were identified, for the Russell 2000 119 were identified, and for the DJIA 31 were identified. Contrary to expectations, in one of the cases the number of detected outliers actually decreased. This means that it wasn't able to effectively differentiate normal data from tainted data.

3.1.3 Hurst Exponent

While the Hurst Exponent didn't return any quantitative data, it correctly identified areas of caution in the data set. It identified whenever noise was introduced and highlighted areas where KNN and LOF found anomalies.

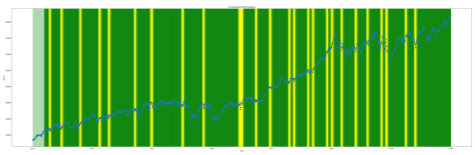

Figure 2: Flags for untainted Dow Jones data

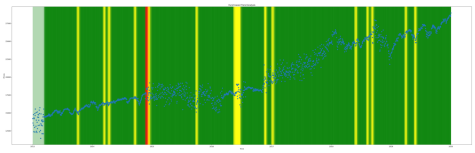

Figure 3: Flags for tainted Dow Jones data

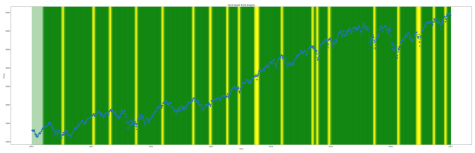

Figure 4: Flags for untainted NASDAQ data

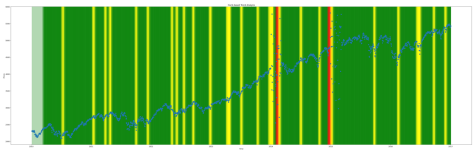

Figure 5: Flags for tainted NASDAQ data

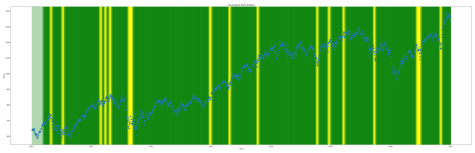

Figure 6: Flags for untainted Russell data

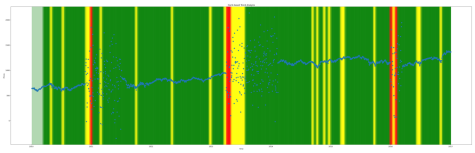

Figure 7: Flags for tainted Russell data

3.1.4 KNN Unsupervised Learning

Each run of KNN was applied to the original data and tainted/simulated attack data for Nasdaq, Russell 2000, and DJIA stock indices. Below are the KNN results which include a color map of the anomalies, the top 5 anomaly scores in the data set, and a graph of the outliers mapped onto close price v.s. time of the data set for all 3 types of data sets per the stock index.

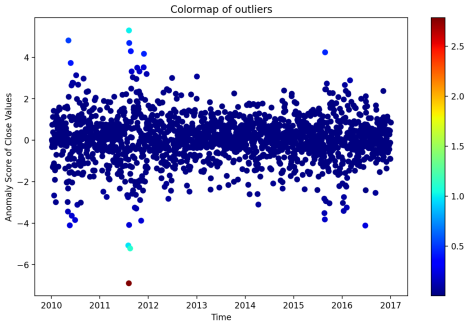

Figure 8: Color-map of anomaly scores for each data point in untainted Nasdaq data

Top 5 Largest Anomaly Scores of Data Entries

Date	Close	Anomaly_Score
2011-08-08 00:00:00	-6.8994	2.7841
2011-08-18 00:00:00	-5.218	1.1125
2011-08-09 00:00:00	5.2946	1.0019
2011-08-04 00:00:00	-5.0753	0.9697
2010-05-10 00:00:00	4.8123	0.5196

Figure 9: Top 5 anomaly scores out of all data points in untainted Nasdaq data

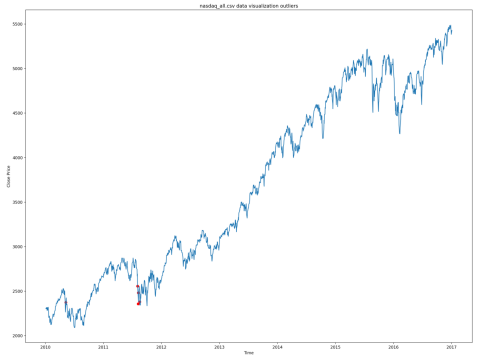

Figure 10: KNN identified outliers mapped onto close price vs. time graph for untainted Nasdaq data

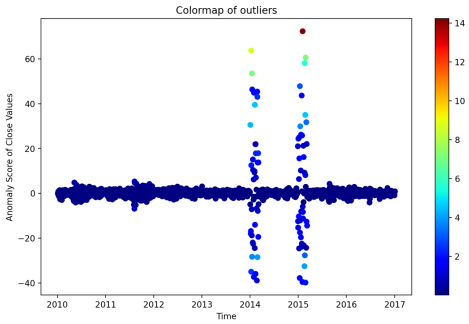

Figure 11: Color-map of anomaly scores for each data point in TAINTED Nasdaq data

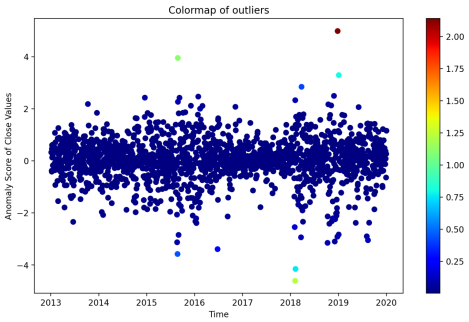

Figure 14: Color-map of anomaly scores for each data point in untainted DJIA data

Top 5 Largest Anomaly Scores of Data Entries

Date	Close	Anomaly_Score
2011-08-08 00:00:00	-6.8994	2.7841
2011-08-18 00:00:00	-5.218	1.1125
2011-08-09 00:00:00	5.2946	1.0019
2011-08-04 00:00:00	-5.0753	0.9697
2010-05-10 00:00:00	4.8123	0.5196

Figure 12: Top 5 anomaly scores out of all data points in TAINTED Nasdaq data

Top 5 Largest Anomaly Scores of Data Entries

Date	Close	Anomaly_Score
2018-12-26 00:00:00	4.9846	2.1401
2018-02-05 00:00:00	-4.6049	1.2163
2015-08-26 00:00:00	3.9516	1.1071
2019-01-04 00:00:00	3.2925	0.7966
2018-02-08 00:00:00	-4.1493	0.7607

Figure 15: Top 5 anomaly scores out of all data points in untainted DJIA data

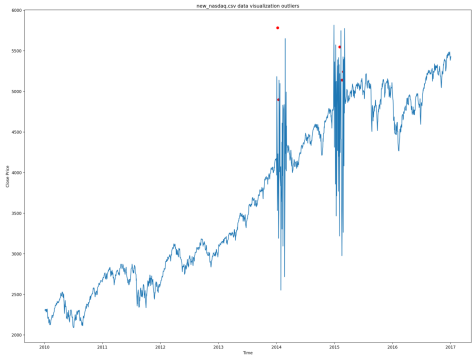

Figure 13: KNN identified outliers mapped onto close price vs. time graph for TAINTED Nasdaq data

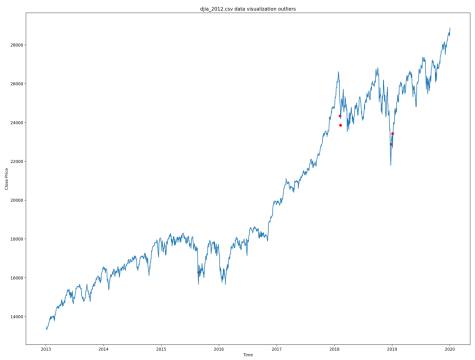

Figure 16: KNN identified outliers mapped onto close price vs. time graph for untainted DJIA data

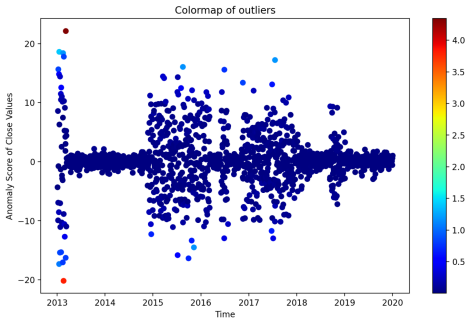

Figure 17: Color-map of anomaly scores for each data point in TAINTED DJIA data

Top 5 Largest Anomaly Scores of Data Entries

Date	Close	Anomaly_Score
2013-03-08 00:00:00	22.1228	4.3417
2013-02-19 00:00:00	-20.167	3.8091
2013-01-17 00:00:00	18.6073	1.3921
2013-02-15 00:00:00	18.3883	1.1731
2017-07-19 00:00:00	17.2152	1.1731

Figure 18: Top 5 anomaly scores out of all data points in TAINTED DJIA data

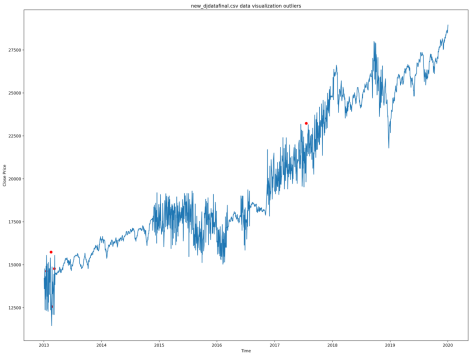

Figure 19: KNN identified outliers mapped onto close price vs. time graph for TAINTED DJIA data

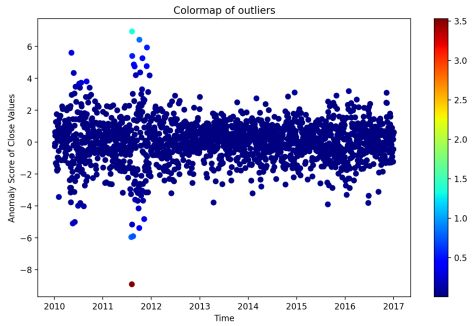

Figure 20: Color-map of anomaly scores for each data point in untainted Russell 2000 data

Top 5 Largest Anomaly Scores of Data Entries

Date	Close	Anomaly_Score
2011-08-08 00:00:00	-8.9095	3.5273
2011-08-09 00:00:00	6.9436	1.3372
2011-10-04 00:00:00	6.4234	0.817
2011-08-04 00:00:00	-5.95	0.7859
2011-08-18 00:00:00	-5.8975	0.7334

Figure 21: Top 5 anomaly scores out of all data points in untainted Russell 2000 data

Figure 22: KNN identified outliers mapped onto close price vs. time graph for untainted Russell 2000 data

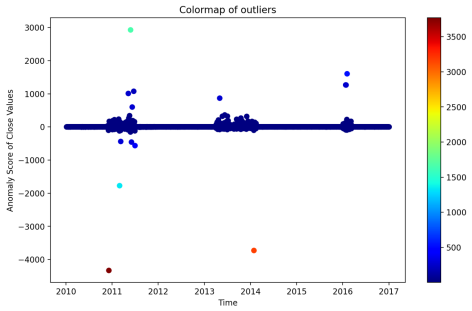

Figure 23: Color-map of anomaly scores for each data point in TAINTED Russell 2000 data

Top 5 Largest Anomaly Scores of Data Entries

Date	Close	Anomaly_Score
2010-12-06 00:00:00	-4,332.0936	3,767.8015
2014-01-28 00:00:00	-3,729.468	3,165.1759
2011-05-27 00:00:00	2,927.8789	1,662.6433
2011-03-01 00:00:00	-1,773.1482	1,334.0316
2016-02-05 00:00:00	1,603.6748	528.4821

Figure 24: Top 5 anomaly scores out of all data points in TAINTED Russell 2000 data

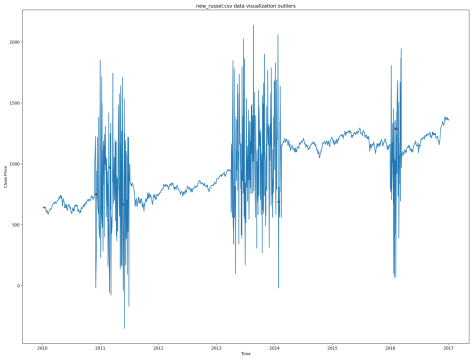

Figure 25: KNN identified outliers mapped onto close price vs. time graph for TAINTED Russell 2000 data

3.2 LSTM Model Predictions

This section will display LSTM close price prediction model results for the 3 types of data sets (original/untainted, tainted, tainted but cleaned with our data manipulations) for Nasdaq, DJIA, and Russell 2000. Each graph will display the observed data vs. the predicted data for close price plotted against time. Additionally, the root mean square error values for train and test scores will be given in figure captions.

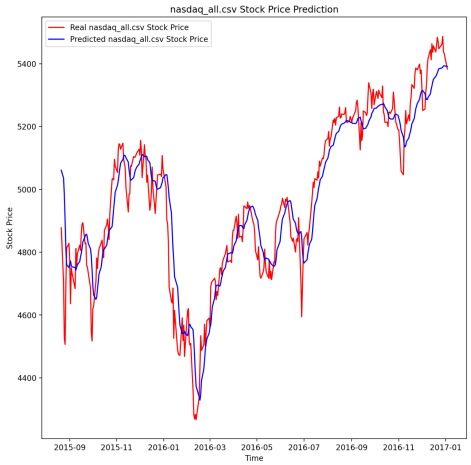

Figure 26: Untainted Nasdaq dataset: Train score = 49.49 RMSE; Test score = 89.12 RMSE

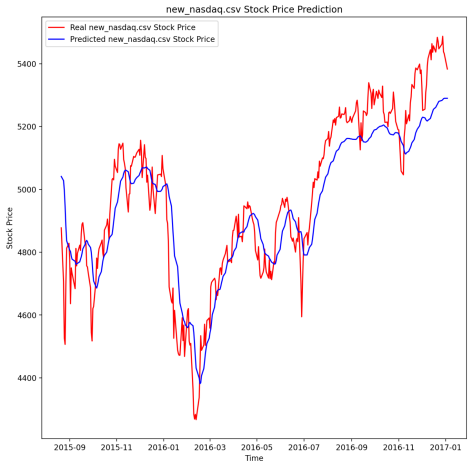

Figure 27: Tainted Nasdaq without cleaning: Train score = 203.44 RMSE; Test score = 110.44 RMSE

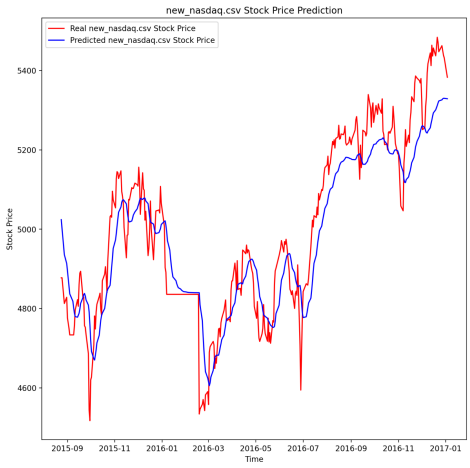

Figure 28: Tainted Nasdaq with cleaning:Train score = 130.15 RMSE; Test score = 90.89 RMSE

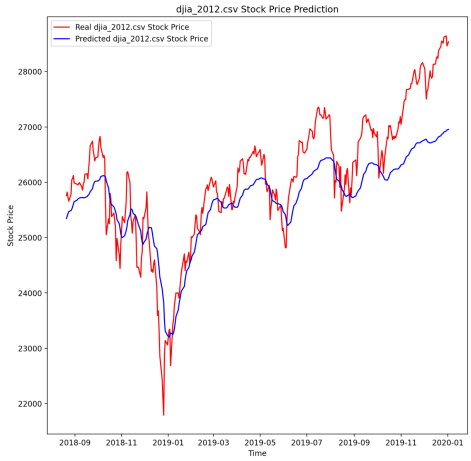

Figure 29: Untainted DJIA: Train score = 235.69 RMSE; Test score = 685.74 RMSE

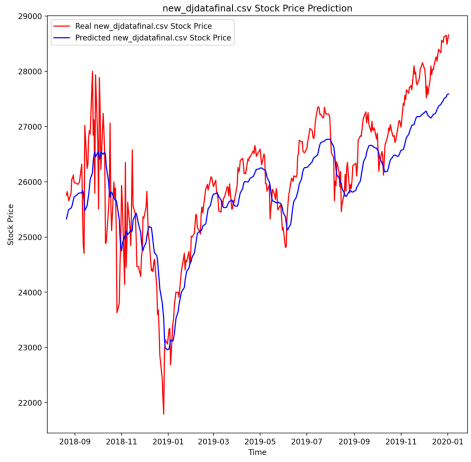

Figure 30: Tainted DJIA without cleaning:Train score = 583.16 RMSE; Test score = 600.84 RMSE

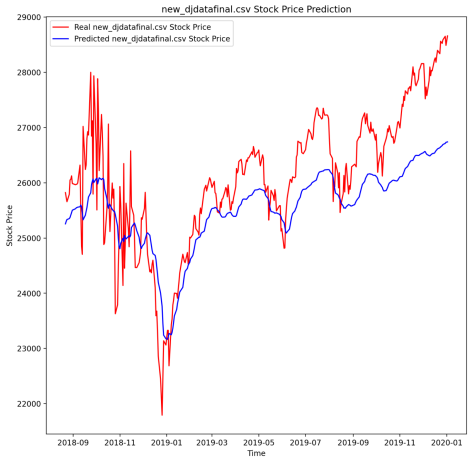

Figure 31: Tainted Nasdaq with cleaning:Train score = 558.45 RMSE; Test score = 861.06 RMSE

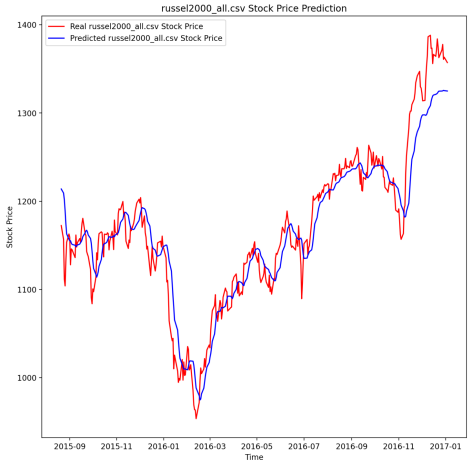

Figure 32: Untainted Russell 2000:Train score = 17.19 RMSE; Test score = 28.52 RMSE

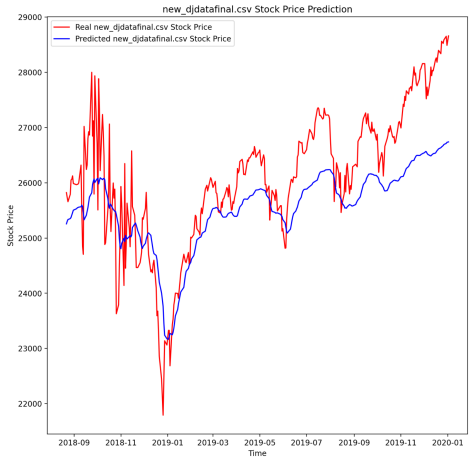

Figure 33: Tainted Russell 2000 without cleaning:Train score = 218.31 RMSE; Test score = 28.52 RMSE

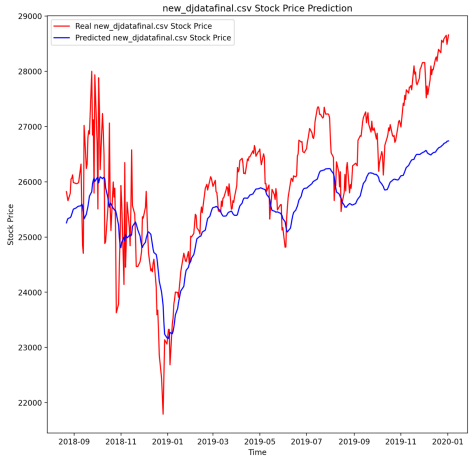

Figure 34: Tainted Nasdaq with cleaning: Train score = 171.42 RMSE; Test score = 49.17 RMSE

We can see from these models' results that the test score significantly declines between the tainted data without our cleaning and the tainted data with our cleaning for Nasdaq and Russell 2000 indices. However, for the DJIA index there seems to be an increase in the test score from the tainted data without cleaning and the tainted data with cleaning. We are not comfortable with the result for DJIA and the issues with this index will be examined in the discussion.

4 Discussion

4.1 Data Manipulations

4.1.1 Hole Detection

Hole detection worked well with the exception of holidays. If these had been hard-coded in this would have performed much better. However, because holidays change each year and closures are dependent on geography and world events, this would be hard to accomplish. One possible alternative solution is to hard code the strict rules such as weekends and duplicate data whilst the "looser" rules are merely detected and a flag thrown, rather than treating all rules the same. In this way, another ML model could be trained to detect violations of more minor rules. Overall, Hole detection performed adequately, but it could definitely be improved.

4.1.2 Local Outlier Factor

Local Outlier Factor did not perform adequately. It succeeded in detecting changes between different patterns or trends, however, it failed to detect many different tainted data points. This method is very susceptible to the data being over-saturated with corrupted data. This happens when a significant amount of the training window contains tainted data, which causes predictions made from that training window to either be meaningless or to classify further tainted data as correct.

4.1.3 Hurst Exponent

Through our experience using Hurst in this model, we recognize that Hurst will not work well for stock data specifically, but would work much better for data that is expected to exhibit consistent behavior in any measurable dimension. In this instance, it was a good indicator of data that could be anomalous, but it did not contribute any exemplary results.

4.1.4 KNN Unsupervised Learning

The KNN method implemented has proven to be successful in discovering outliers in our data set. However, the main limitation of this implementation is that K-neighbors are set to only 4, and with variable data set sizes this would not be applicable to every data set. KNN is an algorithm that is normally implemented as a machine learning supervised learning model. The algorithm is used normally for classification purposes where the value of k dictates how many neighbors will be looked at in comparison to the current point. From this comparison, using distance metrics, the algorithm is able to classify whether a point belongs in a certain group based on its neighbors' groupings[2].

The use of KNN in our research is to iden-

tify outliers. Thus, instead of classifying data points into certain groups, which could be useful in the context of orbital data with multiple characteristics, we classify data entries based on their fit to the general trend of their neighbors. Therefore, a point can be classified as an outlier if it does not follow the same trend as the neighbors using the evaluation metric of Euclidean distance concatenated with the percent change of close price between previous and current data entries. The aforementioned problem of the pre-defined k value could lead to a lack of confidence in our results. The issue is that with $k = 4$, we do not know if this is the most optimal value of k to use for each data set which prompts a need for parameter sweeping over k.

Some interesting results of our implementation of the KNN are listed below:

- Nasdaq and Russell 2000 original data sets both listed August 8th, 2011 as the top anomaly.
- Nasdaq original data set listed August 18th, 2011 as the second highest anomaly score.
- DJIA original data set listed December 26th, 2018 as the highest anomaly score.
- KNN overall tainted data sets performed well catching all the highest anomaly-scored data entries.

August 8th, 2011 is known as "Black Monday" and according to CNN Money reports, this was the worst day for Wall Street since the 2008 financial crisis. The Nasdaq composite dropped by 175 points, DJIA sank by 635 points, and the S&P lost 80 points[7]. Unfortunately, we did not have DJIA data preceding 2012, however if we did, we would most likely see this date as a top anomaly score.

Furthermore, CNN Money reported that the Nasdaq composite took a major hit of 131 points on August 18th, 2011 due to factors such as: poor forecasts for global economic growth, a slowdown in the domestic manufacturing sector, and a U.S. housing report that was published with unanticipated negative consequences[8].

December 26th, 2018 was indeed an anomalous day for the markets. As reported by CNN Business, the Dow had risen 1,086 points as well as 5 percent increases in the S&P 500 and Nasdaq. CNN Business noted that there was no news or indicators that drove this biggest percentage gain of all 3 indices since March 3rd, 2009[5].

4.2 LSTM Model Predictions

The LSTM model was used to test the effects of our data manipulation cleaning methods. We hoped to see a decline in test scores following the LSTM prediction model run from the tainted data set to the tainted data set with cleaning applied. This was the case for the Nasdaq and Russell 2000 tainted data sets, however, the DJIA data set did not follow this expected pattern.

DJIA tainted data set performed with a test score of 600.84 RMSE. The tainted data set without manipulations applied performed with a test score of 861.06 RMSE. We are not comfortable with this result, however, there is a potential reason for it. It should be noted that the original DJIA data set performed with a test score of 685.74.

The goal of our tainted data sets is to simulate adversarial attacks and create more anomalous data. However, in this case, the tainted data set performed better than the original data set by a decrease of around 85 RMSE. Therefore, it can be inferred that somehow our simulation of attacks on the data set resulted in more robust and better-performing data. So, when our manipulations are applied to this data set, they remove these anomalous points that made the data fit better in the LSTM model. Therefore, the tainted data set with cleaning will consist of several inaccurate points causing the test error to rise.

5 Future Work

One important step that needs to be taken is mapping out the relationship between how much of the data is corrupted, and to what extent it is corrupted, and the accuracy of the predictions of the model this data is being fed into. Similarly, finding the relationship between missing data and the accuracy of predictions is important. With information about both of these, detection algorithms can be better tuned. It is possible that missing data is harmless while corrupted data can invalidate all of the model's predictions, meaning that detection algorithms should be very strict and remove any potential outliers. The opposite could also be true and the detection algorithms should be very lenient. These results will likely be data set dependent, and would need to be analyzed for each data set in use.

Another detection technique we considered but decided not to pursue was using a Generative Adversarial Network to detect outliers. If fully trained and converged, the discriminator half of a GAN could be an incredibly powerful classifier for incoming data. However we decided against trying this approach because of the time investment. Nevertheless, it is still a serious option worth considering.

Other possible machine learning models include autoregressive integrated moving average models and autoencoders which would work well with Hurst[3]. These models were too time intensive to be pursued but they are worth consideration.

Furthermore, keeping the current implementation of KNN in mind, we could improve the anomaly detection by implementing a supervised learning ML KNN model. Firstly, by running parameter sweeps on the optimal value of k, we can ensure that the KNN model will represent reliable results for any data set inputted. Secondly, incorporating different distance and weighting metrics could improve the reliability and accuracy of our results. For example, the Hassanant distance KNN variation uses the simple design of the normal KNN algorithm, however it calculates distance using maximum and minimum vector points of nearest neighbors in a testing query and uses majority voting rule as an evaluation criteria[2]. Implementing a supervised learning method of this variation of KNN could highly improve our accuracy and allow us to capture more anomalies.

A statistics based approach to detecting outliers is also very possible. Normal distributions could be fit to recent data points and approximations of their derivatives. Then the Probability Density Function of these distributions could be used to score incoming points on how well they

match with previous data. This would be very tunable as the cutoff threshold for when to keep or discard a data point is completely controllable.

6 Appendix

https://github.com/vedajammula/L3Harris-Senior-Capstone.git

6.1 Project Contributions

Andrew built the window architecture as well as Hole and Hurst methods. He also worked on the data pipeline with Sarthak.

Ethan researched different detection methods and implemented LOF.

Sarthak built the data pipeline architecture and implemented it with help from Andrew. He also implemented the KNN unsupervised learning algorithm, as well as implemented the LSTM model. Sarthak provided the formatted CSV data sets for Nasdaq, DJIA, and Russell 2000. He also created the multi-page Streamlit UI dashboard and with help from Andrew, outputted all results of data manipulations and LSTM prediction model runs to the dashboard.

Portia built the simulated data attacks by testing various different noise models, as well as different means and standard deviations (levels of noise).

Veda worked on the simulated data attacks in order to get the final data files ready for the model.

Tim surveyed and tested model baseline assessments providing initial metrics.

6.2 Acknowledgements

We would like to thank our sponsors from L3Harris, Brian Pankau and Patrick Bernard, for their continuous advisement and assistance which led to the successful completion of our project.

Moreover, we would like to thank our teaching assistant Dreycey Albin for his guidance and support throughout the entire project.

Lastly, we owe a big thank you to Alan Paradise, our professor for this year-long capstone project, for providing us the materials, support, and knowledge to effectively plan and implement the project.

References

[1] Hum Nath Bhandari, Binod Rimal, Nawa Raj Pokhrel, Ramchandra Rimal, Keshab R. Dahal, and Rajendra K.C. Khatri. Predicting stock market index using lstm. *Machine Learning with Applications*, 2021. https://doi.org/10.1016/j.mlwa.2022.100320.

[2] Shahadat Uddin et al. Comparative performance analysis of k-nearest neighbour (knn) algorithm and its different variants for disease prediction. *Scientific Reports*, 2022. https:https://doi.org/10.1038/s41598-022-10358-x.

[3] Eryk Lewinson. Introduction to the hurst exponent — with code in python. *Towards Data Science*, 2021. https://towardsdatascience.com/intro

duction-to-the-hurst-exponent-with-code-in-python-4da0414ca52e.

[4] MoreCoding. Anomalydetection. *Kaggle*, 2021. https://www.kaggle.com/code/morecoding/anomalydetection/notebook#Module-9:-Anomaly-Detection.

[5] Sherisse Pham and David Goldman. Dow soars 1,086 points in a miraculous comeback. *CNN Business*, 2011. https://www.cnn.com/2018/12/26/investing/stock-market-today/index.html.

[6] Saif Shabou. Time series with r. 2020. https://s-ai-f.github.io/Time-Series/outlier-detection-in-time-series.html.

[7] Ken Sweet. Dow plunges after s&p downgrade. *CNN Money*, 2011. https://money.cnn.com/2011/08/08/markets/markets_newyork/index.htm.

[8] Ken Sweet. Stocks get demolished. *CNN Money*, 2011. https://money.cnn.com/2011/08/18/markets/markets_newyork/index.htm.

[9] Taron Zakaryan. Predicting stock price using lstm model, pytorch. *Kaggle*, 2021. https://www.kaggle.com/code/taronzakaryan/predicting-stock-price-using-lstm-model-pytorch/notebook.

[10] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2023. https://d2l.ai/chapter_recurrent-modern/lstm.html.