A Comparative Analysis of Stock Price Prediction Using a Manual-Based and Library-Based Support Vector Machine (SVM) Model

For the degree of

B.Sc. in Computing Science

Griffith College Cork

June 10th, 2025.

Under the supervision of:

Atif Atif

Table of Contents

Abstract

In the past few years, both experts and retail investors have been trying to predict stock prices movement. This project examines how well machine learning models can use technical indicators, SMA, EMA, RSI, and MACD to predict the short-term direction, either up or down, of stock prices. An interesting aspect is the comparison between a manual SVM built using NumPy and standard library-based models from the scikit-learn library like the Logistic Regression, Random Forest and K-Nearest Neighbors.

This project involved building a clean dataset by using historical price data from Apple (AAPL) with features added by using technical indicators. A prediction label was made to categorize the next day's price path. After data preprocessing and normalization, it was used to train both the manual and library-based models. The models were tested under different train/test splits from 60/40 to 90/10 to see their learning behavior.

This project uses real-world historical stock data from certain companies and evaluates each model's predictive accuracy with different split/test structure. The transparency of the manual SVM implementation is emphasized, which informs us more on how decision boundaries are

formed. Visual and quantitative comparisons of the model performances were made, and the results are saved to CSV files and visualized through plots.

The model's performance was compared both visually and quantitatively, and the results were saved to csv files and graphs were plotted to help with the analysis. One important observation was how the model was sensitive to different training sizes and data splits. Also, the manual SVM did better when it was trained on bigger datasets. This shows how important training data size and feature significance in financial predictions.

The results suggest that while the manual SVM offers valuable educational understanding, the library-based models are generally more accurate and can be used in more situations. This project not only demonstrates how machine learning can be used in real life finance, but it helps students and aspiring data scientists understand how classification algorithms work when applied time-series financial data.

Introduction

During my final year, I wanted my project to show something that combines my growing interest in data science and analysis, and my curiosity about financial markets. After considering some ideas, I decided to focus on using technical indicators and machine learning, especially Support Vector Machines (SVM) to predict short-term price movements, and it took a while to make this decision.

Initially, I researched other ideas about financial forecasts and retail investors behavior, but I was not able to find realistic datasets that would help me implement techniques taught in my machine learning module. With the help of my supervisor, I finally decided to use technical indicators like SMA, EMA, RSI and MACD which generated using real historical stock price data sourced from Kaggle. Traders often use these indicators, and I thought it would be exciting to see how machine learning models like SVM, Logistic Regression, Random Forest and KNN could use these indicators to predict future price movements.

This topic interested me because it had both theoretical and practical depth. There are many factors that can affect how the stock market moves. Machine learning cannot be 100% accurate, but it can be used to find trends in large amounts of data. Choosing this path also aligned with the modules I took in my final year, most especially Machine Learning and Data

Analysis and Visualization, which gave a great chance to use what I learned in class with a real dataset.

Another important part of this project was learning how to use NumPy to implement a manual version of SVM from scratch. I wanted to learn more about how this could work without using built-in libraries to deepen my understanding of how these algorithms work. It turned out to be a difficult and rewarding part of the project as it took multiple attempts to work correctly, especially while tuning the decision boundary and managing various train/test split ratios. I generated visualizations, kept track of the model performances across the different splits and saved predictions to CSV files, aiming to keep my project as practical as possible.

I learned how to use optimization methods like Stochastic Gradient Descent (SGD) in a practical way. I also learned how important it is to scale features and distribution data are when trying to improve the performance of a model. When built-in models are used, results are gotten faster, but with manual implementation, each line of the code helps you understand how the model works. This hands-on experience greatly increased my confidence in working with basic machine learning principles.

Although, this was a technical project, it also helped me to be patient, fix bugs and helped in my decision making. I had to test different layouts and go back to fix parts that didn't work which slowed down my progress but also helped me grow. The combination of theoretical research, code implementation and visual analysis in this project was a great way to round up my degree.

This report records the motivation, tools, methodology, challenges and results that went into making the project. Before getting this final version, there was a lot of process full of various tests, mistakes and adjustments that helped me grow both technically and personally.

Literature Review

Before I started working on the model and structure of this project, I researched different sources to learn more and have a better understanding of how stock market prediction has been approached in recent works, especially by using technical indicators and machine learning models.

The stock dataset sourced from Kaggle (Kaggle, 2023) was utilized as the basic data foundation for the purpose of conducting experimental research. The use of technical indicators and the training of predictive models were both made possible by this dataset, which provided comprehensive historical stock information.

The manual implementation of the Support Vector Machine was made easier by NumPy (NumPy Developers, 2024). NumPy was able to facilitate the implementation of numerical computations. The array operations and mathematical functions that it offered were crucial in the process of handing large datasets and performing efficient calculations.

A GitHub repository that really helped me during my research was Rudrendu Paul's project on "Support Vector Machine for Predicting Stock Market Daily Trend." This project majorly focused on using Support Vector Machine to predict daily movements in the S&P500. This project was helpful to me because it showed how SVM can be directly used in the finance market, and how features could be selected based on technical indicators. While my project used a simplified manual implementation of SVM using Stochastic Gradient Descent (SGD), it was encouraging to see that this approach worked in real-life application and has been used in open-source projects. This inspired me to structure my own SVM class using NumPy, adjusting it to fit binary classification of price movements.

To evaluate the effectiveness of the manual SVM, I utilized scikit-learn, which is a robust machine learning package (Scikitlearn Developers, 2024). It provided standardized implementations of various algorithms, which made it possible for a comparative analysis of model accuracies and efficiency.

The repository's way of preprocessing and labeling data also helped me figure out how to describe the binary classification task. I chose a simpler structure because it fits the scope of my project, but seeing how it was used in the real world helped me understand how theoretical models can be used in the real world. It also changed how I structured my data flow, especially the strategy of figuring out if the price will go up or down the next day.

To get a foundational understanding of the technical indicators I used, I watched a YouTube video, "The Only Technical Analysis Video you will Ever need" by the Trading Channel. It explained the concepts of SMA, EMA, MACD and RSI, and made them easy to understand and able to use in my code. The video clarified how these indicators show signs on potential buy or sell moments. It also encouraged my decision to calculate and visualize the indicators like sma_5, ema_7, and rsi_9 in the beginning of my project.

To facilitate the implementation of technical indicators into my project, I made use of the open-source Python library known as ta which I got from a GitHub repository, "Technical Analysis Library using Pandas and NumPy" by Dario Lopez Padia (bukosabino). This library includes functions for generating SMA, EMA, RSI and MACD. The ta library made it possible for me to implement these formulas in a programming manner with few errors and consistent outcomes. This was helpful during the feature engineering stage, when I needed to generate multiple indicators in a short amount of time to examine their relationship with the movement of the stock price. The availability of such tools reflects the efforts made by the developing community to make financial analytics more accessible to scientists and researchers.

In addition, the visual illustrations in the video helped me understand how the indicators behave when trends are going up or down. This helped me understand my own data visualization and how I explained their role in model performance. The video helped change the way I thought about indicators from being just numbers but as signals traders often respond to. This made it easier to explain their use as machine learning features.

I also utilized the use of Matplotlib (Hunter, 2024) and Seaborn (Seaborn Developers, 2024) for my data visualization. The use of these libraries made it possible to generate insightful plots such as the decision boundaries, and trend analysis which played a crucial role in the interpretation of the performance of the machine learning models ad the behavior of technical indicators.

Another GitHub repository that helped me is "Stock Price Prediction using LSTM and Technical Indicators" by Tejas Linge. This repository used LSTM which I didn't implement, but it was nice to see how the author organized their dataset using multiple indicators before it was passed to the model. This reassured me that using multiple indicators as an input function was effective.

Although I did not use LSTM for my project, seeing this implementation encouraged me to use technical indicators in a classification setup. I learnt a lot from how the data was prepared, normalized and put into the model. I still maintained the standard machine learning rules, but working with time-series specific models made me more aware of how my project could improve in future.

These resources helped me gain a better understanding of the methodology, indicators and the machine learning model I implemented. I thought about their approaches and used them to

suit the scope for my project. It was also good to know there are resources and other people that had some similar ideas and were able to implement them.

Finally, looking at existing projects and learning materials really helped me better understand how my project fits better in the bigger context of financial prediction using machine learning. Although, the methods used in my project were limited, especially the use of technical indicators and the implementation of both manual and the built-in models, which are consistent in the industry. These sources encouraged me to work with my own ideas and gave me a clearer understanding of the pros and cons of the different modeling strategies.

System Design and Methodology

To efficiently predict how stock prices will prove in short term, I structured the whole process into different data science stages which includes data collection, feature engineering, EDA, modeling and evaluation. These steps were made to fit with practical machine learning processes, while making it easy to understand and developing hands on algorithm development.

Overview of the Approach

The main goal of this project is to use machine learning models and technical indicators to predict short-term price movement. Firstly, I began by gathering historical stock data sourced from Kaggle, making new features using some common trade indicators and setting up the prediction function as a binary classification problem. The process involved training multiple models, both library based and manual models, and then testing them using different train/test splits.

Dataset and Preprocessing

The dataset used in this project was sourced from a Kaggle notebook “Stock Market Data Analysis.” It includes daily historical stock prices for different companies. I focused mainly on Apple (AAPL) for this project because of its steady price movement.

I loaded the AAPL subset into a pandas data frame and carried out the following steps to make sure the data was suitable for machine learning:

1. **Missing values**: Rows that had any missing values were removed particularly the rows that were affected by the calculation of the indicators, SMA, EMA, RSI and MACD.
2. **Sorting and Alignment**: To avoid data loss during train/test splits, the data was sorted in ascending order.
3. **Feature Normalization**: To make models like SVM and KNN perform well, z-score normalization was used to make sure that indicator values were all the same.

These steps made sure that the data that was loaded into the model was clean and correctly structured for both training and testing purposes.

After splitting the main file, a little filtering step removed any rows with missing values. The resulting clean data was then saved as *AAPL_filtered_stocks.csv*, which then served as an input to *indicators_AAPL.py*.

Before focusing on Apple (AAPL) and Netflix (NFLX), the original *stock.csv* file, which already contained daily data for multiple tickers, was automatically split into one csv file for each ticker. This was done in the *main.py* script, which filters the *stock.csv* file by the Ticker column and saves the files as AAPL_data.csv, NFLX_data.csv, GOOG_data.csv and MSFT_data.csv. After that, only the AAPL and NFLX were used for further preprocessing.

A binary target variable called '*price up'* was created to show if the price went up the next day:

df['price_up'] = (df['Close'].shift(-1) > df['Close']).astype(int)

This made the problem a classification function, where the model predicts 1 if the price is expected to go up, and predicts 0 if the price is expected to drop.

Using AAPL as the target dataset was a good choice because it is well liked by investors, has a stable trading rate, and covers a large part of the market, making it a good example to use when testing the predictive models. I later added NFLX data to the model to make it more complete, but AAPL was still the main focus for most of the modelling and testing process.

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) was very important in transforming the raw historical price data into a more clear understanding of patterns, trends and relationships. The EDA was done

in a different script called *eda.py* file, whose main goal was to show how the price of Apple (AAPL) stock will change over time and how the technical indicators changed with it. This visual representation helped to ensure data quality, understand trends and decide which of the indicators will be best for making predictions. It also proved the decision to use the technical indicators as feature inputs instead of the prices.

Some of the EDA explored are:

- **Line Plot of the Closing Price**: To see the general trends and how prices changed I plotted the closing price over time. This helped me see any important trends going up or down.

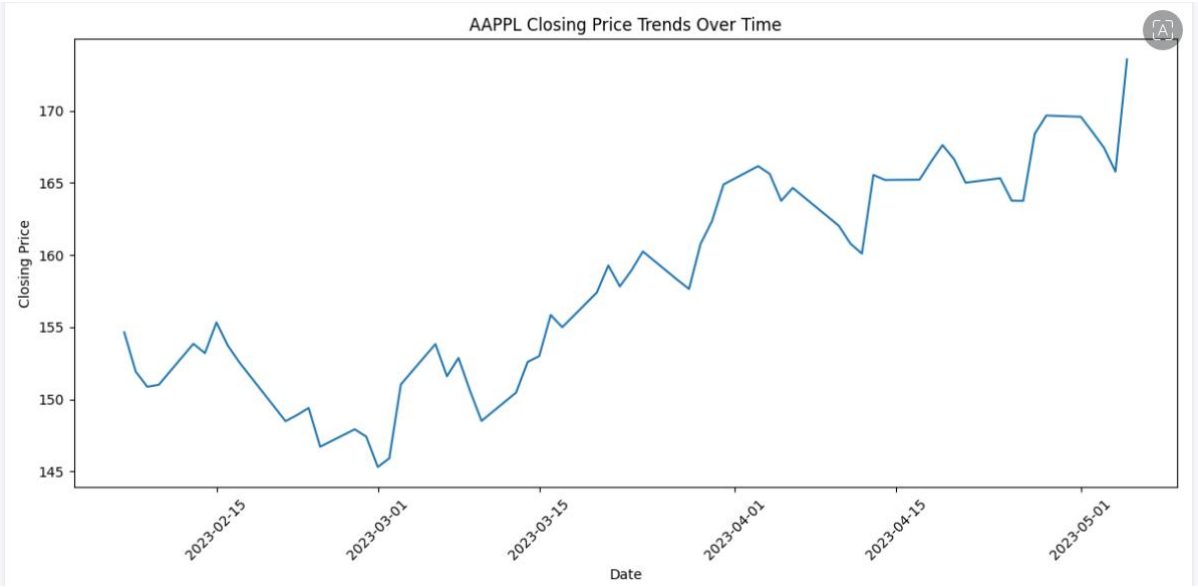

Figure 1: The trend of AAPL closing prices

- **Histogram of Closing Prices**: A histogram was plotted to better understand the daily distribution of closing prices. This showed that the prices were grouped into specific groups, and the KDE curve showed what the main trend was.

Figure 2: The distribution of the closing price of AAPL with a KDE overlay.

- **Correlation Heatmap**: The correlation heatmap was used to show the relationship between the closing price and the technical indicators. This helped check for related or unnecessary features.

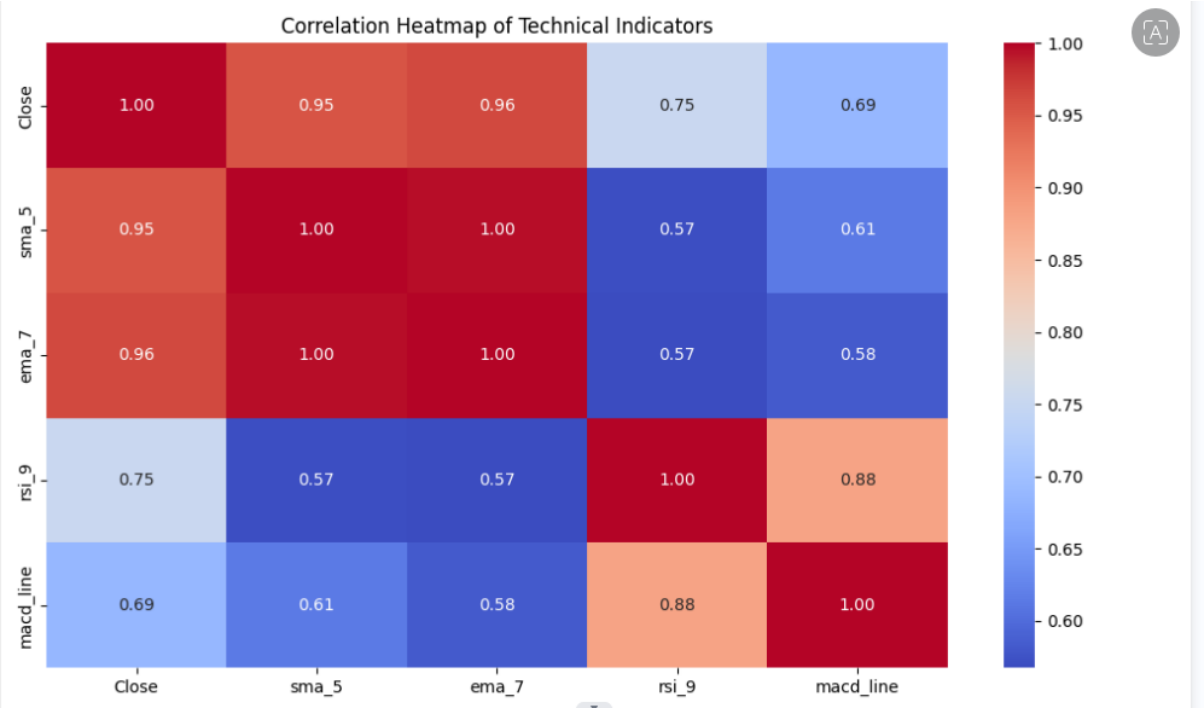

Figure 3: The correlation between technical indicators and closing price

- **Price Trend with SMA and EMA:** A line plot was made together with the actual closing price, the 5-day Simple Moving Average (SMA) and the 7-day Exponential

Moving Average (EMA). This was helpful to see how the short-term trends evolved over time and how the technical indicators tracked the actual price changes. It also confirmed that the features can help the models find signals in the data.

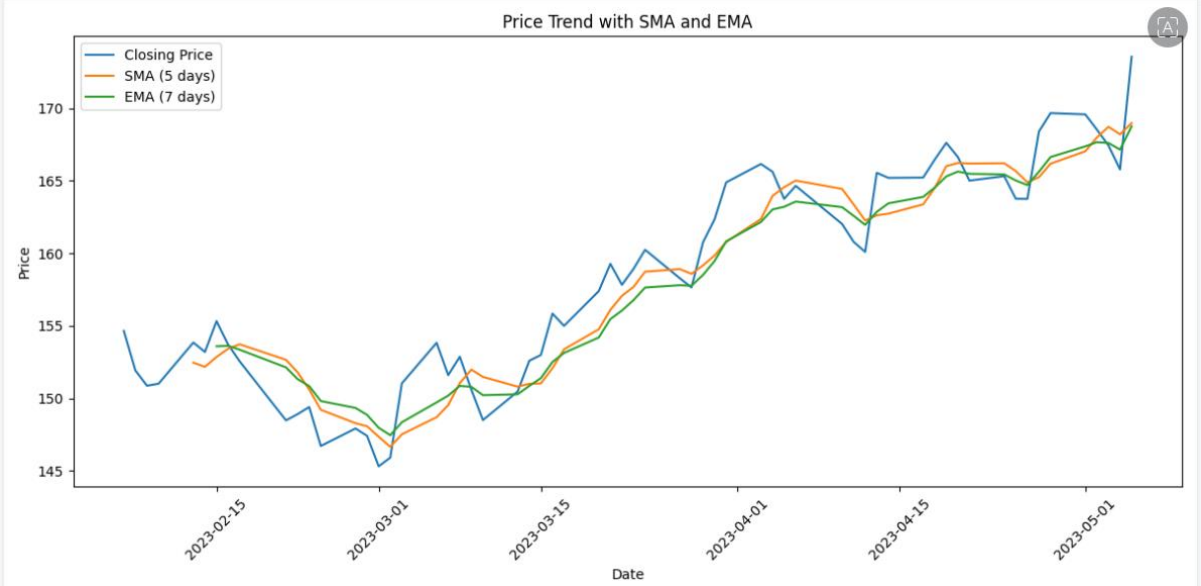

Figure 4: This shows how the closing price fits with SMA and EMA trends.

- **RSI Plot**: I plotted a graph for the Relative Strength Index (RSI) over time. The 30 (oversold) and 70 (overbought) levels were shown with horizontal lines. This helped me understand when the stock has hit a turnaround zone.

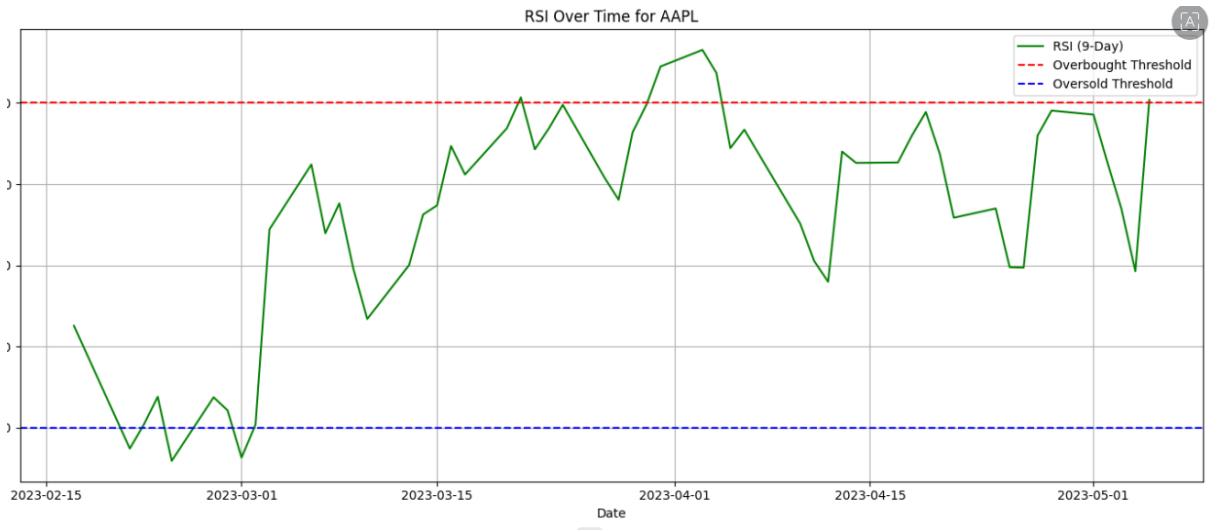

Figure 5: This shows the overbought and oversold condition

- **MACD vs Signal Line**: I plotted a graph of the MACD Line and the Signal Line to evaluate convergence/divergence points, which are often used to predict when momentum will change.

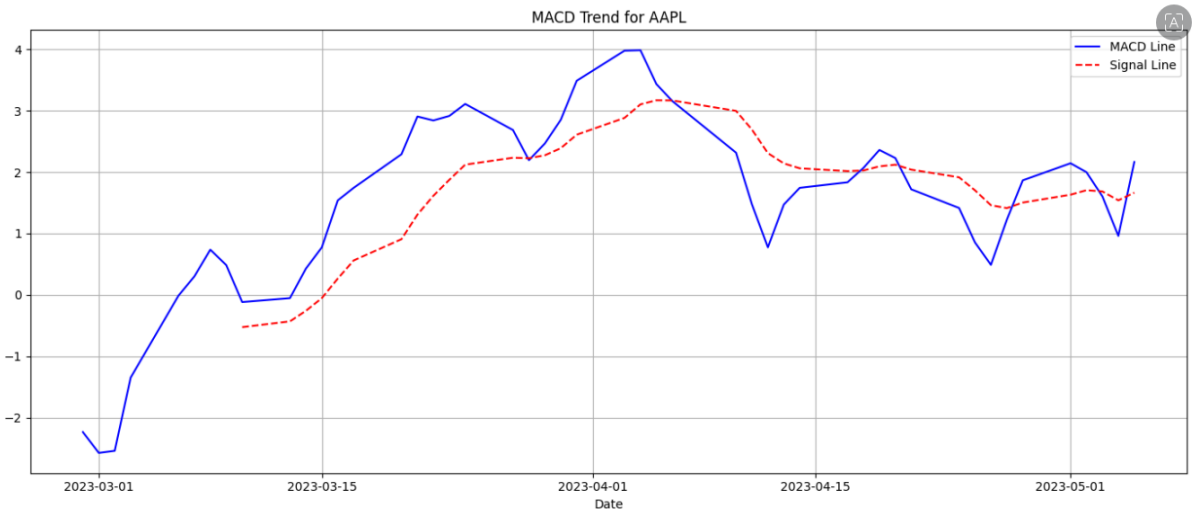

Figure 6: A plot of the MACD and Signal line showing the trend changes.

Overall, the EDA helped to confirm which of the indicators changed in a meaningful way and which ones were strongly linked to the price. This step made sure I wasn't just putting features into models, but I chose the input features based on evidence of signal strength. It also helped me have a stronger knowledge base on the project's foundation with visual analysis.

All EDA plots such as the closing price, histogram, correlation heatmap, RSI and MACD were generated using *stock_indicators_AAPL.csv*. Although, similar process could be used for *stock_indicators_NFLX.csv*, the main EDA plots in this document focuses on AAPL.

Feature Engineering

I generated a set of technical indicators to give the models significant input:

- **Simple Moving Average (SMA):** A 5-day average that gets rid of short-term noise.
- **Exponential Moving Average (EMA):** A 7-day average that gives recent prices more weight.
- **Relative Strength Index (RSI):** This measures movement and can now show overbought or oversold conditions.

- **Moving Average Convergence Divergence (MACD):** This shows changes in trend direction and momentum.

The ta library was used to compute the technical indicators, which were then added to the dataset and used as the main set of features to train the models. Each indicator was selected for how easy it was to understand. While there are other more complex indicators like Bollinger Bands, I decided to focus on models with more clarity and transparency over feature depth. These 4 indicators are generally used in technical analysis, which makes it easier to understand and align with real world strategies.

Each of these indicators were evaluated using only past data points in order to avoid data leakage. This is very important for time series forecasts because using future data would make the model have less accurate predictions.

In addition to AAPL, the same technical indicators were generated for Netflix (NFLX). A separate file called *indicators_NFLX.py*, uses the ta library to calculate sma_5, ema_7, rsi_9, macd_line and macd_signal, and then writes the improved dataset to *stock_indicators_NFLX.csv*. Later on, NFLX was used as a second test to make sure that our model logic still worked with NFLX.

Development Environment and Dependencies

All scripts were developed in Python 3.11.5 using a virtual environment. Key libraries used include Pandas, NumPy, ta, Matplotlib, seaborn and scikit-learn.

Modeling Approaches

To evaluate how well the different machine learning methods could learn from technical indicators, I used two different modeling techniques. The first used the built-in classifiers from the scikit-learn library, while the other focused on building a manual Support Vector Machine (SVM) from scratch using NumPy.

Built in models using scikit-learn

Four built-in models were selected based on their relevance in classification problems and are commonly used in tasks involving financial prediction tasks:

- Support Vector Machine (SVM)

- Logistic Regression
- Random Forest Classifier
- K-Nearest Neighbors (KNN)

Each model used the same technical indicators, sma_5, ema_7. Rsi_9 and macd_line, as its input. These indicators are meant to transform price data into trends, momentum and signals that are easy to understand.

As part of this method, I saved the predicted values for each of the models and generated a graph to visualize their performance. The prediction results were saved to a csv file and created a plot to show each of the models performed across the different train/test splits. I used accuracy as the main metric for comparison. Since the main goal of the project was to predict the stock price movement accuracy provided a direct way to test how regular the models made the right predictions.

Manual SVM Implementation

I implemented a custom Support Vector Machine which used NumPy and trained it with Stochastic Gradient Descent (SGD). I decided to choose SGD because it is a commonly used optimization method for linear models like SVM. It works by changing the model's weights repeatedly based on the training case, which helps strengthen the concept of SVM margins. While there are other methods like Sequential Minimal Optimization (SMO), SGD was used for my project because it fits better with the goals and timeline of my project.

In the manual implementation, only two features, sma_5 and ema_7, were used to visualize the decision boundary in a two-dimensional space. The SVM was trained on standardized features X_train and y_train labels, that were converted from (0,1) to (-1, 1).

For the final testing, a decision boundary was plotted using the 90/10 train/test split. This made it easier to see how well the model divided the two classes, if price went up or dropped, which proved that the custom implementation worked as expected.

The next section, Implementation and Results, shows how this theoretical design was translated into Python code and what accuracy we actually got.

Train/Test Split and Evaluation Strategy

To see the stability and performance of the model, I evaluated the manual SVM over different train/test splits:

- 60/40
- 70/30
- 80/20
- 90/10

The idea behind using different ratios was to replicate situations where access to data changes, which is a common problem in the real world. In some situations, data scientists may only have a limited amount of historical data which can be used for training, while in other situations, they may have access to large datasets. Evaluating how the model works with both more and less training data helped me understand how sensitive manual implementation is to the amount of data.

For each of the splits, the model was trained to know how accurate the model was on the test set. The manual SVM was only tested mainly using accuracy, which was measured by the number of times it predicted price changes accurately. This fit in with the classification task, *price_up* label, that was set up earlier.

The results were saved to a csv file, and a graph was plotted to visualize each of the split size performance, which helped to clearly compare the accuracy across the different split ratios.

These tests showed that adding more training data does not always make the model perform better. It can depend a lot on how well the data is organized and how the model interacts with it. Testing with different splits also showed how important it is not to depend on just one evaluation measure when drawing conclusions from machine learning tests.

Visualization Tools

Two key plots were generated to help illustrate how the model performed during testing:

- **The Accuracy Comparison plot**: This is a line graph that shows how the manual SVM accuracy changed for each of the four train/test splits. It shows a visual comparison and performance in the different data availability. By plotting the

accuracy scores on a single axis, it makes it easier to see which of the configurations gave the most balanced result and which one had trouble with overfitting or underfitting.

- **The Decision Boundary plot**: It is a two-dimensional decision boundary plot that shows how the manual SVM divided two classes using the sma_5 and ema_7 features. This plot is very helpful because it shows how the model's margin was formed and how well it separates the feature space. It gives a visual representation of potential model weaknesses.

These visuals were not only essential for communicating the results but also for validating the manual implementation to work correctly. For example, the decision boundary plot helped make sure that the manual SVM was correctly learning from the data. If the plot has shown a random class separation, it would mean there was a problem with the learning process of feature scaling. Also, the accuracy plot helped to show how changes in data split size affected the performance, and whether more training always led to better results.

Overall, the visualizations helped to have better understanding of how the model works by acting as a connection between raw numerical results and the model behavior. They gave immediate input during the development and helped make sure that the SVM, both the built-in and manual, was not just generating results but also doing so in a way that can be trusted and understood. The visual confirmation also aligned with the project's goal of combining transparent behavior of the model with practical finance predictions.

Implementation and Results

This section will explain in detail how the two models, manual SVM and the built in scikit-learn model were implemented and tested. It will also discuss the results, the model's accuracy across the train/test splits and the understanding gained from the visualizations like the accuracy plot and decision boundary plot.

Built In Model using scikit-learn

To set up a baseline for comparison, I first used a standard classification using the scikitlearn library which has been outlined in the previous section. The four indicators SMA, EMA, RSI and MACD were used to train the model. The train_test_split function was used to divide the dataset into parts, then the accuracy of each model was found using the unseen test data. The

models were selected because it works well in supervised classification task and easy to use and test with scikit-learn.

This setup made it possible to directly compare various models and verify if the technical indicators could accurately predict the short-term direction of prices, whether it is going up or down.

For every model, SVM, Logistic Regression, Random Forest and KNN, predictions with the labels were saved to files named *model_predictions_AAPL.csv* and *model_predictions.NFLX.csv*. A combined accuracy comparison of all libraries-based models was saved in *model_comparison_AAPL.csv* for Apple and *model_comparison_NFLX.csv* for Netflix. Also, for the manual SVM, train/test split accuracies were saved in *the manual svm_split accuracy.csv file*, and its end class results were saved in *svm_predictions_AAPL.csv*. The csv files makes it possible to compare the numbers without having to rerun the whole process.

Manual SVM Implementation Using NumPy

An important part of this project was creating a manual Support Vector Machine classifier using NumPy. A challenging part of this project was getting the optimization process to work with Stochastic Gradient Descent (SGD). Some key steps implemented are:

- Making a class SimpleSVM with new fit() and predict() methods
- Two features sma_5, ema_7, selected to train the model and easier to visualize.
- Normalizing the features using z-score normalization before training the model.
- Mathematically, SVM works better with -1 and 1, so converted the target labels from (0, 1) to (-1, 1).

The implementation process needed several iterations while trying to fix the bugs. One of the challenging parts was making sure that the weight updates and margin calculations worked right, but it helped me understand how margin-based classifiers work in the back.

Training Strategy and Evaluation

To see how universal the model was, I ran my manual SVM model through some different train/test split ratios, 60/40, 70/30. 80/20 and 90/10. For each of the split:

- The model was trained again from scratch.
- The predictions were made on the test set.

- The accuracy was calculated by comparing the predicted and actual values.
- The results were saved to a csv file for documentation and visualization.

This method allowed me to test the model's performance changes based on the amount of training data that is available, which is similar to how things work in the real world, where the amount of data can vary significantly.

Results Overview

The performance of each of the manual SVM models across the splits are:

Train/Test splits	Accuracy (%)
60/40	44
70/30	67
80/20	38
90/10	75

These values were saved in a csv file named manual svm_split accuracy.csv. Based on the table above, the 90/10 split produced the best accuracy for the manual SVM, probably due to the larger training set providing more information for the model to learn from.

Visual Results

Two key plots were generated:

- **Accuracy Comparison Plot**: This is a line graph that shows how the accuracy changed between the different train/test splits.

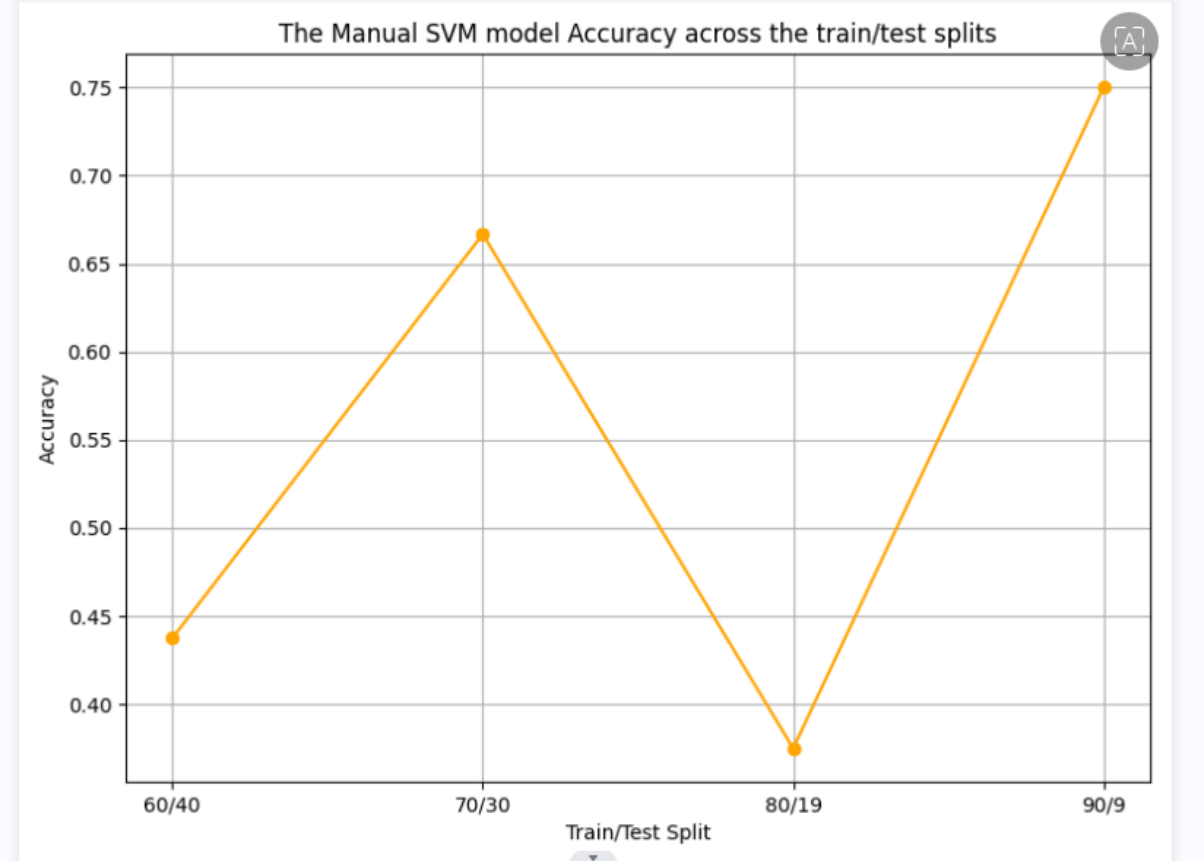

Figure 7: The Accuracy comparison plot across the different train/test splits

- **Decision Boundary Plot**: A two-dimensional decision boundary plot was created using the 90/10 split to show the groups are divided using the SVM.

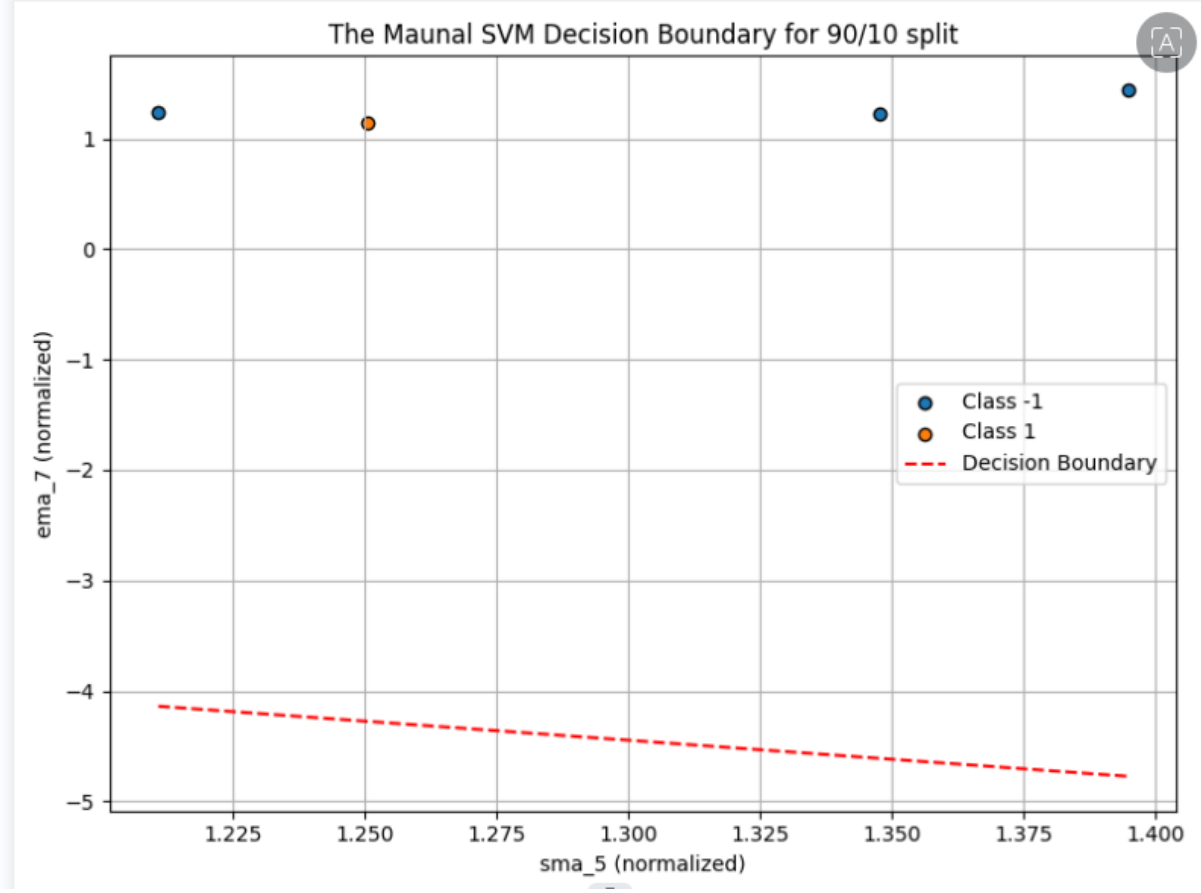

Figure 8: The Decision boundary of the manual SVM using 90/10 split, which illustrates how the model separates the 2 classes based on sma_5 and ema_7 features.

These plots provided both validation and understanding. The decision boundary showed whether the model could clearly differentiate between trends going up or coming down, while the accuracy plot showed how well the model was doing overall.

Discussion

This section explores about how the model works, why certain features were chosen and the overall learning experience during the project as a whole. The results from the technical indicators and machine learning models gave different outcomes, they provided important insights into what works well and what does not work well, especially when working with small financial datasets. This part of the documentation reflects on the models tested, how they performed with different data splits, and the technical lessons drawn from building a manual SVM model from scratch.

Performance Summary

The Support Vector Machine (SVM) and Logistic Regression models gave the most reliable and consistent results from all the built-in models that were tested with scikit-learn. With the manual SVM, the 90/10 split achieved the best accuracy with a 75% score. This was fascinating because success on smaller test sets is often inconsistent. Although, it appears the manual model learned the margin better because it had more data to train on, even when the test set was smaller. On the other hand, the 80/20 split had the lowest accuracy with an accuracy score of 38%. This shows that performance can change a lot with small changes in data availability. The 70/30 split performed okay with a 67% accuracy score, showing that while it wasn't the best, it made a solid training and testing balance.

Models like Random Forest and KNN were less dependable across the splits. This may be because the technical indicators used gave better signals, continuous trends and momentum patterns which linear models like SVM or Logistic Regression are better at detecting, rather than the clear class separations that tree-based or neighbour-based models like.

Observations on the Manual SVM

While the manual SVM did not perform better than the built-in model, the process of building it from scratch using NumPy taught us some important insights. A major observation was seeing how sensitive the models was to hyperparameters like the number of epochs and the learning rate. Sometimes little changes in the values caused some changes in accuracy. This showed that the optimization process needed to be tuned carefully.

Furthermore, visualizing the decision boundary made the idea come to life. The two-dimensional representation of how the SVM told the difference between bullish and bearish moves helped to strengthen the mathematical idea behind margin-based classification. It was also clear why feature selection was so important, models trained with simple features like sma_5 and ema_7 made limits that were easy to see, but they might not be very good at generalizing. This observation emphasized the well-known trade-off between how easy a model is to understand and how well it can predict the future.

Overall, the manual implementation of the SVM helped me learn more about the margin-based algorithms and how they need balanced and well-structured data. It also made it easier to understand the depth behind the scikit-learn's simplicity and how important it is for the algorithms to be stable when working with financial datasets.

The SVM used only two features sma_5 and ema_7, which restricted its ability to learn deeper relationships but made it easy to visualize the decision boundary. This showed that models are often less accurate when simple and easy to understand and interpret.

Lessons Learned

In this project, I learned that model choice is more than just picking a complex algorithm. If the features are clean and well normalized, a simple linear classifier like SVM or Logistic Regression may work best.

Using Stochastic Gradient Descent (SGD) manually was another important lesson. It helped me understand what's really going on in a model when it changes weights and how learning rate and epochs directly affect the boundary drawn between classes. I also learned how important z-scores normalization can be. SVM and KNN would have had worse performance because the size of data affects how they work.

Advantages and Disadvantages in Using Technical Indicators

Traders like to use technical indicators like the SMA, EMA, RSI and MACD because they are commonly used in trade settings. They served as accessible and significant features that helped find trends and momentum shifts. These indicators provided structure and historical context, which made them more useful for time-series classification tasks.

However, it became clear during this project that these signs are not enough to fully explain how the market works. Using established features that are easy to understand and trying to make custom features that might show better progress or momentum is a trade-off. I might add volume-based indicators like Bollinger bands or combining indicators in future work to see higher order interactions.

Also, technical indicators often use set time periods like 5-day SMA or 7-day EMA, which might not always show how stocks move in real time. Including adaptive techniques or meta-features that change based on volatility might make performance more stable. As part of future experiments, it might be useful to use feature methods to find ways that signs like RSI crosses and MACD signal changes can work together.

Why Visualizations are useful

Visualization helps to understand the models and indicators better. Plots like the Decision Boundary, the Accuracy Comparison and the SMA/EMA trends were used throughout to

validate that the models behaved as expected. Without the visualizations, it would have been more difficult to understand the results of some splits.

I was able to have a better understanding of how price changes worked with the help of EDA plots like the RSI line and the MACD signal also helped me interpret the behaviour of price changes. It would have been more difficult to tell how the model was aligning with the actual trends without the plots.

During development, these plots were used as checks to make sure that the models were not only doing well numerically, but also behaving as expected in real world.

What I would improve

If I were to revisit this project in future, I would look into a longer training window and add time-based cross validation, which better simulates how things work in the real world of finance distribution. I would also try the model on unseen time ranges to see how well it generalizes to future market behaviour. This is better than using random splits which can cause data leaks in time-series data.

Additionally, enhancing the manual SVM by adding soft margin SVM reasoning and adding more signs to derived metrics. This would allow better handling of conflict between the bullish and bearish classes.

Finally, adding more advanced feature selection methods like correlation-based filtering could help make the model work better and reduce noise. These methods might help find the most useful indicator and discard the ones that contribute little to the classification task. Given that the financial data is always changing and has a lot of dimensions, this could help with both speed and processing.

Limitations

This project shows that both built-in and the manual implementation using machine learning models can be used to predict short-term stock prices accurately. Although, there were some limitations that affected the results:

- **Limited Dataset Scope**: The project briefly tested the Netflix (NFLX) dataset, but the main focus of analysis, feature engineering and manual model training was the Apple (AAPL) dataset. It is difficult to generalize the results across other sectors or

industries because the study mainly focused on a single stock. More tests with different companies would help confirm if the model is reliable.

- **Limited Time Range:** The historical data only covered a short time period. A larger and more varied time period with different economic phases might show more about how reliable the model can be.
- **Manual SVM Feature Limitation:** The SVM that was manually implemented used only two features for visualization to be easy. This helped explain the decision boundaries but was not as good at prediction compared to the models that were trained on all four indicators.
- **Simplified Model Scope:** This project did not explore other advanced models like XGBoost and LSTM that are built to predict time series data. The dataset is a time series data, but the project focused on a classification problem using custom technical indicators.
- **Hyperparameter Sensitivity:** The learning rate and epoch count was very important to the manual SVM model. The hyperparameters were adjusted manually which might have not led to the best result.
- **Approach based only on Technical Indicators**: There were no basic indicators like earnings or sentiment analysis included, all features used were only technical. This left out important market moving signals.
- **Computational and Time Limits:** The implementation was limited by academic timeline and computational tools available. This made it difficult to run larger experiments.

Regardless of all these limitations, the project still gives important information about how machine learning models work with financial data, mainly when it comes to technical indicators and manual algorithm implementation.

Future Work

This project has successfully explored the use of technical indicators and machine learning models to classify short-term stock trends. However, the work can be extended in a number of areas in future versions.

Adding information for more than one company can be a next step for future development. By using statistics from many companies in different industries, the model can be tested to

see how well it works in other industries as well. This would show more clearly that technical indicators and classification models are differ types of market situations.

Another valuable improvement will be to introduce more data types like basic indicators like earnings, P/E ratios and sentiment-based features like news headlines and Twitter scores would also be a good idea. These sources of data show how think about the market and macroeconomic context, both which have a big effect on price changes. Combining them with technical indicators could make them more accurate and easier to understand.

In this project engineered label like bullish or bearish were used. In future, it might be helpful to reframe the problem as a regression task so that we can predict how prices or returns will actually change. Long Short-Term Memory (LSTM) networks and other more advanced time aware models could be looked into. The purpose of these models is to find trends in sequential data. This could mean better results on stock data where time dependencies are important.

From a systems perspective, another area to explore is the development of a user-friendly application, which is discussed in detail in the following section. This could be done with tools like Streamlit or Flask, which does not require a complex frontend setup. This would make the project easier to use and could serve as the foundation of personal investment tracking apps to be made.

One more extension is to make feature selection and hyperparameter setting automatic. For this task, factors like the rate of learning and certain features had to be changed manually. Using Grid Search, Bayesian Optimization or AutoML tools in future versions could help make more reliable models and reduce the amount of trial and error that needs to be done manually.

Furthermore, the integration of live market data through APIs like Alpha Vantage or Yahoo Finance API would also make it possible to make predictions in real time. This would change the project from historical analysis to a fully functional support tool for investors. Combined with alert systems or trade signals, the application could be used as a simple financial assistant.

Finally, later versions might look into model explainability methods like Local Interpretable Model-agnostic Explanations (LIME), to interpret which of the features are causing each

prediction. This would be useful for transparency but also for building trust in the system and among users.

In summary, this project serves as a solid foundation for more advanced research or experimentation in financial prediction systems. The model can be improved to make it useful for investors or researchers by adding more data types, larger datasets better automation and modern architectures.

Potential Frontend and Deployment

This project focused mostly on implementing the backend and technical analysis. However, a simple frontend interface could make it a useful tool for retail investors. As a possible next step, you could make a simple web platform that lets users add csv files of stock, see technical indicators and receive real-time trend predictions from the trained models.

Frameworks such as Flask or Streamlit would be well suited for this. Streamlit in particular makes it easy to quickly create live data apps in Python, which works with the codebase that already exists. This would allow people interact with the analysis graphically, like by plotting the SVM decision boundary or checking how the MACD and RSI indicators match up with the predictions, without having to run Python scripts or handle raw data manually.

A frontend like this would also make the project more useful for learning. Users could test different stocks from their own accounts or look at how different indicators affects requirements. This would help them learn more about technical analysis. Dropdown options like date-range filters and auto generated insights could make the tool more accessible to non-technical users.

In the longer term, deploying the app on a web server or in the cloud could make it available in future as a personal finance assistant or learning tool. Due to lack of time, deployment was not possible for this research project. However, the backend design already works with lightweight interfaces, so it would not take much work to extend the project into a live system. This path also lets you add live stock data APIs and user login systems, which will make the project more useful in the real world and allow it to grow.

Critical Reflection and Personal Learning

This project was not only an academic requirement, but also as a significant opportunity for both personal and professional development. My knowledge in machine learning and data analysis from the module provided me with the opportunity to apply theoretical knowledge to a real-world financial problem and I was able to find a solution to it. I became more aware of the extent to which I have developed both technically and personally by reflecting on the entire process, beginning with the planning stage and continuing through the documentation stages.

When I first started working on this project, I was familiar with how machine learning could be utilized in the financial sector was limited to a level. Although I had the opportunity to learn about ideas like Support Vector Machine (SVM) from lectures, I had never really applied one manually or used in a professional setting. Making the decision to use NumPy to develop a manual SVM turned out to be the rewarding aspect of this project. I was compelled to engage with the mathematical logic behind the algorithm, decision boundaries and convergence. Although the manual process was more challenging and time consuming, it helped me have a deeper understanding of gradient descent as well as the impact of learning rate, epochs, and feature normalization has on the training results.

One of the most insightful moments was when trying to balance the performance explanation. For instance, in order to see the decision boundary better, I decided to use only two features in the manual SVM, sma_5 and ema_7. However, I realized that this simplicity came at the expense of the accuracy of the predictions. This trade-off showed me that the explanation of models is important, but it must be weighed against the objective of producing accurate predictions in the real world. This made me more thoughtful in how I select features and define model objectives.

My approach to data preprocessing and feature engineering was another important area in which I made significant progress. Initially, I assumed that using technical indicators such as SMA or the RSI index would be enough, but as I progressed through Exploratory Data Analysis (EDA) phase, I saw how important it was to clean data, normalize features and understand each feature's relationship to the target variable. The correlation between indicators, redundancy and timelines were all factors that played a role in whether the model performed well or not. Learning how to deal with these issues gave a better confidence in working with financial data.

I also gained a lot of knowledge about the evaluation methods for the model. Before this project, I mostly depended on random train/test splits most of the time. However, this project made me consider how time-based validation strategies can be more realistic in financial contexts. Additionally, it served as a reminder that accuracy alone is not enough, it is equally important to understand the reason why a model performs bad. The visualizations plotted, the decision boundary in particular helped me check the accuracy with the actual behaviour trend.

Time management was another skill I developed during this project. Initially, I underestimated the amount of time that would be required for each phase, most especially the process of debugging and tuning process involved in the manual SVM. However, as the project progressed, I learned the ability to plan better, allocate time for testing and avoid overcomplicating features.

This project also tested my ability to work independently, troubleshoot technical issues and still maintain focus despite deadlines and academic responsibilities. There were many times I felt frustrated, most especially when the results seemed inconsistent, but working through the problems helped me become more patient and flexible as a data scientist and analyst.

Finally, this project helped me discover a stronger interest in data science and analysis. I have always had interest in business and markets, but this project made me more confident in handling positions in financial analytics or data driven decision making roles after graduation. Working with actual datasets, developing predictive models, and discussing my findings, both in terms of technical logic and business relevance, are all activities I feel more confident and comfortable with.

In summary, this project was much more than a technical assignment but also a personal milestone in my journey towards becoming a data analyst in future. I gained new knowledge, faced real challenges, and now emerge with a better understanding of machine learning as well as myself as a learner. The experience has equipped me with practical tools and a mindset that I know will be beneficial in my future professional journey.

Conclusion

The major aim of this project was to explore how Support Vector Machine (SVM), a type of machine learning model, could be used with technical indicators to predict short-term changes in stock prices. I explored both built in models with scikit-learn and a custom manual implementation using NumPy. The project was done using a real-world data from Kaggle with a major focus on Apple (AAPL) due to its regular trading activity.

This project showed that technical indicators can be used to partly predict the financial market trends. This was done by dataset preparation, feature engineering with technical indicators like SMA, EMA, RSI and MACD. Building a manual SVM from scratch helped me understand how model training works at the computational level especially when Stochastic Gradient Descent (SGD) is used as an optimization method.

The built-in models such as SVM and Logistic Regression worked well and provided a standard measure for comparison. While the manual SVM was more difficult to implement it still performed well with a 90/10 train/test split with an accuracy score of 75%. This showed how important the amount of training data is for margin-based classifiers.

Significantly, the project taught me a lot of important things that can be used in real word applications, like how important it is to be careful with feature normalization, split strategies and how hyperparameters can affect the model performance. Visualizations were also important in this project. They not only communicate the results but also helped understand trends.

Overall, this project was a great way to learn because it combined technical implementation and real financial uses. It helped me grow as a student and an aspiring data analyst, by enhancing my ability in data handling, building models, evaluating and writing reports. Although, there are still some things that could have been improved, the information and results gained during this process will form a strong foundation in finance and machine learning.

References

- Hunter J.D. Dale, Firing E., Drottboom M., 2024. Using Matplotlib.

https://matplotlib.org/stable/users/index.html

- Kaggle. Stock Market Data Analysis

https://www.kaggle.com/code/thesnak/stock-market-data-analysis

- Lopez Padial. D, 2023. Technical Analysis Library using Pandas and NumPy.

https://github.com/bukosabino/ta

- NumPy Developers. NumPy

https://numpy.org/doc/

- Rudrendu Paul, 2022. Support Vector Machine for Predicting Stock Market Daily Trend.

https://github.com/RudrenduPaul/Support-Vector-Machine-SVM-for-Predicting-Stock-Market-Daily-Trend

- Scikitlearn. Supervised Learning

https://scikit-learn.org/stable/supervised_learning.html

- Seaborn. Seaborn: Statistical Data Visualization

https://seaborn.pydata.org/

- Tejas Linge, 2021. Stock Price Prediction using LSTM and Technical Indicators

https://github.com/tejaslinge/Stock-Price-Prediction-using-LSTM-and-Technical-Indicators/blob/main/Price%20Prediction%20using%20LSTM%20and%20TA.ipynb

- The Trading Channel, 2021. The Only Technical Analysis Video You Will Ever Need.

https://www.youtube.com/watch?v=eynxyoKgpng

Appendix

Appendix A: The Manual SVM Implementation (NumPy)

This section includes the Support Vector Machine (SVM) code that was made from scratch using NumPy and Stochastic Gradient Descent (SGD).

lass SimpleSVM:

```
def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):
    self.lr = learning_rate
    self.lambda_param = lambda_param
    self.n_iters = n_iters
    self.w = None
    self.b = None

#
def fit(self, X, y):
    n_samples, n_features = X.shape
    self.w = np.zeros(n_features)
    self.b = 0

    for _ in range(self.n_iters):
        for idx, x_i in enumerate(X):
            condition = y[idx] * (np.dot(x_i, self.w) + self.b) >= 1
            if condition:
                self.w -= self.lr * (2 * self.lambda_param * self.w)
            else:
                self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y[idx]))
                self.b -= self.lr * y[idx]

def predict(self, X):
    approx = np.dot(X, self.w) + self.b
```

return np.sign(approx).flatten()

Appendix B: Technical Indicator with *ta* Library

This shows the use of the *ta* library to generate SMA, EMA, RSI, and MACD indicators from stock price data for NFLX.

#a) Simple Moving Average (SMA) for 5 Days

df['sma_5'] = ta.trend.sma_indicator(close=df['Close'], window=5)

#b) Exponential Moving Average (EMA) for 7 Days

df['ema_7'] = ta.trend.ema_indicator(close=df['Close'], window=7)

#c) Relative Strength Index (RSI) for 9 Days

df['rsi_9'] = ta.momentum.rsi(close=df['Close'], window=9)

#d) Moving Average Convergence Divergence (MACD)

macd = ta.trend.macd(close=df['Close'], window_slow=15, window_fast=5)

df['macd_line'] = macd

df['macd_signal'] = ta.trend.macd_signal(close=df['Close'], window_slow=15, window_fast=5)

Appendix C: Feature Engineering and Label Creation

This code shows how the binary target label *price_up* was generated to show if the price stock will go up the next day.

#If tomorrow's price goes up it is '1', otherwise 0

df['price_up'] = (df['Close'].shift(-1) > df['Close']).astype(int)

#Drop last row as there's no next day and will return NaN

df.dropna(inplace=True)

Disclaimer

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the Degree of Bachelor of Science in Computing at Griffith College Cork, is entirely my own work and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Signed:  **Date: 10/06/2025**