PREDICTING EXTREME EVENTS IN APPLE STOCK PRICES

Contents

Executive summary

This report investigates the prediction of extreme price changes (±2%) in Apple's stock using machine learning models: a Random Forest (RF) and a Temporal Convolutional Neural Network (TCNN). While results showed modest predictive ability, particularly with the RF using resampling and the TCNN using weighted loss, challenges such as overfitting, class imbalance, and the inherent complexity of financial data limited overall performance. Future work should focus on advanced feature engineering, model enhancements, and developing reinforcement learning-based trading strategies. With these improvements, there is potential to build a model that meaningfully contributes to forecasting extreme stock price movements.

Introduction

This report presents an analysis of predicting extreme price changes in Apple's stock using machine learning models. An extreme event is defined as a daily price change exceeding ±2% compared to the previous day's closing price. These events are inherently rare and influenced by complex, interdependent factors, making accurate prediction a significant challenge. The primary focus is on developing and evaluating two models: a Random Forest (RF) and a Temporal Convolutional Neural Network (TCNN). The report details the data preparation process, provides an overview of the two modelling approaches, and compares their performance against established baseline benchmarks to assess predictive power. The report also identifies key areas for potential improvement, particularly focusing on enhancing the TCNN model. The improvement section explores various strategies applied to refine the TCNN's performance. Finally, the conclusion summarises the findings, discusses limitations, and reflects on the challenges of predicting extreme stock price movements.

Data Processing

Effective preprocessing of stock price data was critical to ensure that the models received clean, well-structured, and appropriately scaled input.

Dataset overview

The dataset was sourced using the Yahoo Finance API via the yfinance library, covering the period from January 1, 2015, to January 31, 2024 (inclusive). The dataset consisted of daily stock price data for Apple, with the following key features:

Open: Opening price of the stock.

High: Highest price of the stock within the day.

Low: Lowest price of the stock within the day.

Close: Closing price of the stock.

Volume: Number of shares traded within the day.

Adjusted Close: The closing price adjusted for dividends, stock splits, and other corporate actions.

The daily return was calculated to capture daily price movements using the following formula:

$$Daily\ Return = \frac{Adj\ Close_t - Adj\ Close_{t-1}}{Adj\ Close_{t-1}}$$

An extreme event was defined as any day when the absolute daily return exceeded ±2%. A binary flag, “Extreme_Event”, was created where:

1 indicates an extreme event (absolute daily return > 2%).

0 indicates no extreme event.

To frame this as a next-day prediction problem, the Extreme_Event column was shifted by one day. This ensured that for each row, the model will use today's features to predict if an extreme event occurs tomorrow.

Handling the non-trading days

The stock market is closed on Saturdays and Sundays, resulting in gaps within the dataset. Two approaches were considered:

- Option 1: Add missing weekend dates and forward-fill values with the last known price.
- Option 2: Omit non-trading days entirely from the dataset.

The first approach was rejected, as including stagnant prices could introduce unwanted noise and distort model learning. Additionally, including weekends would provide no additional predictive value for validation or testing since prices remain unchanged. Therefore, non-trading days were completely omitted, ensuring the dataset reflected actual market activity.

Train-Validation-Test Split

To respect the time-series nature of the data and avoid data leakage, the dataset was split chronologically as follows:

- 70% Training
- 15% Validation
- 15% Testing

The percentage split was based on the total number of trading days in the time window

Lagged Variables (for Random Forest Model)

To enable the Random Forest model to capture temporal patterns, lagged variables were generated. For each day, the preceding 10 days of data were used to form a sequence. This resulted in lagged features such as:

Open_t-1, Open_t-2, ..., Open_t-10

High_t-1, High_t-2, ..., High_t-10

(similarly for other features)

The target variable for each sequence was the Extreme_Event on the immediately following day. This structure provided the model with historical context to learn from past patterns.

Feature Scaling (for the tCNN model)

Given that neural networks are sensitive to feature magnitudes, it was essential to standardise the data. The StandardScaler was applied to ensure features had zero mean and unit variance, promoting

faster convergence during training. The scaler was fitted on the training data to prevent data leakage. The same scaling transformation was then applied to the validation and test datasets to ensure consistency.

Note: Feature scaling was not applied to the Random Forest model, as it is not sensitive to feature magnitudes.

Addressing Class Imbalance (SMOTE Resampling)

The dataset was highly imbalanced, with extreme events being relatively rare. To mitigate this, SMOTE (Synthetic Minority Over-sampling Technique) was applied only to the training data. This technique generates synthetic samples for the minority class, reducing bias towards the majority class. Timing of Resampling: Resampling was conducted after scaling. This ensured that the generated samples respected the underlying feature distributions while maintaining temporal integrity. This preprocessing pipeline ensured that the data was clean, structured, and appropriately balanced, setting a solid foundation for model training and evaluation.

Model 1: Random Forest

Baseline Model

After preprocessing the data and generating lagged variables, the initial step was to fit a baseline Random Forest model. This model was trained without any additional tuning or resampling, providing an initial benchmark for performance evaluation. The validation dataset was used to assess this baseline accuracy, giving a sense of the model's initial capability in handling the task.

Addressing Class Imbalance

Given the significant imbalance in the dataset (with extreme events being relatively rare), SMOTE (Synthetic Minority Over-sampling Technique) was applied only to the training data. This technique synthetically generates new instances of the minority class, helping the model better learn from underrepresented patterns.

Hyperparameter Tuning

To optimise model performance, GridSearchCV was employed to explore a range of hyperparameters:

- n_estimators: Number of trees in the forest ([50, 100, 200, 300]).
- max_depth: Maximum depth of each tree ([5, 10, 20, None]).
- max_features: Number of features considered for each split (["sqrt", "log2"]).

Tuning Strategy

The tuning process included the usage of the python function TimeSeriesSplit (5 folds). Traditional k-fold cross-validation randomly shuffles data, which is unsuitable for time-series tasks. TimeSeriesSplit maintains temporal order, ensuring that each fold uses past data for training and future data for validation. This mimics real-world scenarios where models learn from historical data to predict future outcomes. The F1-score was used to balance precision and recall. The best hyperparameters were selected based on the highest cross-validated F1-score.

Final Model Training and Evaluation on the Test Set

Once the optimal hyperparameters were identified, the final model training process involved two steps:

Data Consolidation

The training and validation datasets were merged to maximise data availability so that the model could leverage the maximum amount of historical data for learning.

Retraining with Best Hyperparameters

The model was retrained using the best hyperparameters identified from the grid search.

Evaluation Metrics

The final model was evaluated on the unseen test dataset to ensure a fair assessment of its generalisation capabilities.

Model 2: Temporal CNN

The Temporal Convolutional Neural Network (TCNN) was developed to capture temporal dependencies in the stock price data and predict extreme stock price movements. The TCNN leverages convolutional layers to identify patterns across sequences of historical data.

Data Preparation

The data preparation process for the TCNN involved multiple key steps to ensure that the model received sequences of consistent and scaled inputs.

Sequence Creation

The dataset was transformed into sequences of 10 consecutive days of historical stock data. Each sequence served as input to the model, with the corresponding label indicating whether an extreme event occurred on the next day.

Data Scaling

Neural networks are sensitive to the scale of input data. Therefore, a StandardScaler was applied to ensure each feature had zero mean and unit variance. The scaler was fitted only on the training data to avoid data leakage, and the same transformation was applied to the validation and test sets.

Handling Class Imbalance

As with the Random Forest model, the dataset exhibited class imbalance, with extreme events being relatively rare. To address this there are two options:

1. SMOTE (Synthetic Minority Over-sampling Technique) was applied only on the training dataset to synthetically generate new minority samples and balance the dataset. Resampling was applied after scaling to maintain feature consistency.
2. A class weight was introduced in the Cross-Entropy Loss function, giving more weight to the minority class and encouraging the model to focus on these harder-to-predict events. The

default weighted loss was 4.03 as the majority class was 4.03 time larger than the minority class (In the training dataset)

DataLoader Preparation

The processed data was converted into PyTorch DataLoader objects for efficient batch processing during model training and evaluation.

Model Architecture

The TCNN model architecture was designed to extract temporal patterns from sequential data, leveraging 1D convolutional layers to capture dependencies across the 10-day sequences.

Convolutional Layers

Two 1D convolutional layers were used, progressively expanding the number of filters to capture increasingly complex temporal patterns. The first layer had 32 filters and the second 64 filters, with a kernel size of 3. Padding was applied to maintain the temporal dimension of the sequences.

Activation and Dropout

Each convolutional layer was followed by a ReLU activation function to introduce non-linearity. A Dropout layer was used after the convolutional layers to mitigate overfitting by randomly deactivating a fraction of neurons during training.

Flatten and Dense Layers

The feature maps were flattened before being passed to a fully connected dense layer, which performed the final classification into the two classes (extreme event vs. no event).

Output Layer

The model's output was configured to return raw logits, as required by the Cross-Entropy Loss function.

Model Training

The TCNN was trained using the following strategy:

Loss Function

Cross-Entropy Loss with optional class weighting was used to handle class imbalance, giving greater emphasis to the minority class.

Optimiser

The Adam optimiser was selected for its efficiency in handling sparse gradients and noisy data.

Early Stopping

To prevent overfitting and unnecessary computation, an early stopping mechanism was implemented. The model's performance was monitored using the validation loss, and training was halted if no improvement was observed over 5 consecutive epochs.

Training Process

Each epoch involved two phases:

Training: The model updated its weights using the training data.

Validation: The model's performance was evaluated on the validation data to monitor generalization.

Best Model Checkpointing

The model's parameters were saved whenever the validation loss achieved a new minimum, ensuring the best-performing model was retained for evaluation.

Model Evaluation on the validation set

Following training, the TCNN model was evaluated using the validation set to assess its generalisation capabilities.

Final Model Evaluation on Test Set

Evaluating on the test set could be done with one of the following options:

- Option 1 - Direct Evaluation: Use the trained model as is, based on the best checkpoint determined during validation (via early stopping). The model would then be evaluated directly on the unseen test set without further retraining.
- Option 2 - Retraining with consolidated data: Combine the training and validation datasets and retrain the model to maximise the amount of data used for learning. Importantly, the retraining process would need respect the best early stopping epoch identified during the original training phase, ensuring consistent training duration and avoiding overfitting

For the TCNN model, Option 1 was selected. The reason was to keep the test set completely untouched.

Model Comparison

Performance results

The performance results can be seen in the table below:

Model	Accuracy	Precision	Recall	F1 Score	Confusion Matrix
RF (Baseline)	63.36%	29.49%	25.56%	27.38%	[[188, 55], [67, 23]]
RF (Best Hyperparameters)	59.16%	30.17%	38.89%	33.98%	[[162, 81], [55, 35]]
RF (Best Hyperparameters with Resampling)	68.17%	5.56%	1.11%	1.85%	[[226, 17], [89, 1]]
TCNN	66.97%	38.89%	38.89%	38.89%	[[188, 55], [55, 35]]
TCNN (with Resampling)	72.97%	50.00%	1.11%	2.17%	[[242, 1], [89, 1]]
TCNN (with Default Weighted Loss)	58.26%	34.19%	58.89%	43.27%	[[141, 102], [37, 53]]

Table 1: Model results in the Validation dataset

The TCNN with default weighted loss achieved the highest F1-score (43.27%), indicating the best balance between precision and recall. The plain TCNN (without weighting or resampling) also performed relatively well, with an F1-score of 38.89%, outperforming all Random Forest variations.

All models seemed to have struggled with lower F1-scores, particularly when resampling was applied, where performance dropped dramatically.

Model	Accuracy	Precision	Recall	F1 Score	Confusion Matrix
RF (Baseline)	84.29%	16.67%	2.08%	3.70%	[[278, 5], [47, 1]]
RF (Best Hyperparameters)	82.48%	8.33%	2.08%	3.33%	[[272, 11], [47, 1]]
RF (Best Hyperparameters with Resampling)	73.72%	27.59%	50.00%	35.56%	[[220, 63], [24, 24]]
TCNN	77.64%	9.38%	6.25%	7.50%	[[254, 29], [45, 3]]
TCNN (with Resampling)	85.50%	0.00%	0.00%	0.00%	[[283, 0], [48, 0]]
TCNN (with Default Weighted Loss)	64.95%	18.52%	41.67%	25.64%	[[195, 88], [28, 20]]

Table 2: Model results in the Test dataset

Moving on to the test set, the RF with best hyperparameters and resampling achieved the highest F1-score of 35.56% and a recall of 50%, showing strong performance improvement in detecting extreme events compared to the validation set. The TCNN with default weighted loss came second with an F1-score of 25.64% and a recall of 41.67%, which is quite competitive. The plain TCNN and TCNN with resampling performed poorly, with the resampled version completely failing to predict any extreme events (zero recall).

Model evaluation metrics

Our objective is to predict extreme events. From the above performance metrics accuracy is the metric that should not be used in our case because accuracy also takes into consideration how many times the model correctly predicted a non-extreme event which is not our primary goal. To illustrate this point, the extreme events were 19.8% of all daily events. Thus, a model predicting "non-extreme" for every case would have an accuracy score of 81.2%.

Precision, recall and F1 score are the ones that we need to consider for our scenario. In the code, the F1 score has been prioritised for fine tuning and selections as it combines precision and recall into one score. That said, one may argue that Recall is more important than precision. This is because if we want to predict as many extreme events as possible for trading, the False Negatives will make us miss an opportunity or start losing. This can be much more detrimental compared to a few false alarms. Precision of course still remains relevant however if for example the model predicted an extreme event and this event doesn't occur, then a trader won't be losing much as the stock price will remain at roughly the same levels. F1 score is a score that combines Precision and Recall at an equal proportion so a suggestion would be to use a different score that gives slightly more weight on recall. For instance, F_2 gives 4 times more weight on recall than precision or $F_{1.5}$.

Challenges

In general, predicting extreme events in stock prices is a challenging task. Here are three reasons as to why that is:

1. From a business standpoint, a market share will go up when more people are buying the stock than selling it, and it will go down when more people are selling the stock than buying

it. As a result, this balance can be influenced by almost anything; from political decisions and economic changes to new competitor products/services and investor sentiments. So, both rational and behavioural factors are involved. In any case, these extreme events can occur due to non-linear, interdependent factors that are difficult to quantify and predict.

2. From a statistical point of view, prediction requires separation between signal and noise. Financial data is extremely noisy and often contains random fluctuations that, because of the nature of the event we want to predict (extreme and rare), makes it hard for models to detect and learn from it. Additionally, the fact that financial data is non-stationary (varying around different means and with different standard deviations) means that past patterns may not hold in future occasions.
3. Finally, from a modelling perspective, typical predictive models often rely on observable historical data (like price, volume, and returns) as proxies for the complex, unobservable latent factors driving market behaviour. This approach has some inherent limitations. The main one, in my opinion, is that we are adding an extra point of potential concept drift (i.e., the relationship between features and the target variable may change). For example, events that occurred in a low-interest-rate environment may behave differently when interest rates rise. Another limitation is that models are more likely to overfit to patterns in historical data, mistakenly learning from random noise rather than true underlying relationships and thus, failing to generalise. This is also evident in the tables above, where for most models, the performance metrics were significantly higher on the validation set compared to the test set.

Class imbalance

When class imbalance is present in the data then one wants to find a way to train the model in way so that it understands that the minority class shouldn't get less attention than the majority class. The two most common treatments are resampling techniques (resample the training data to create a more balanced training dataset) and weighted loss (creating a loss function that penalises mistakes on the minority class more than the majority class). The first one was applied for the random forest model and then both were applied for the TCNN model.

SMOTE was chosen as the main resampling technique which creates synthetic samples from the minority class, balancing the datasets. Other techniques were also investigated, but they didn't yield as good results in the validation dataset. The typical minority class over-sampling and majority class under sampling were tried as well as SMOTEENN. SMOTEENN is a hybrid approach that combines oversampling (SMOTE) and undersampling to balance datasets. It first oversamples the minority class (to generate synthetic examples) as normal.

Without resampling:

Both the RF and TCNN models struggled with the class imbalance the dataset exhibited. This is evident from the low recall and precision for the minority class (extreme events). This is particularly clear in the baseline models where the recall for extreme events remained below 10%.

With SMOTE Resampling:

RF with Resampling showed a significant improvement in handling imbalance, achieving 50% recall and an F1-score of 35.56% on the test set. However, this came at the cost of reduced precision (27.59%) and a noticeable increase in false positives. Interestingly, this improvement was not evident if one had only the validation dataset where the resampled dataset had a sharp decline in performance.

TCNN with Resampling, surprisingly, failed to predict any extreme events on the test set. This is evidenced by its precision, recall, and F1-score all being 0%. This suggests that while the model learned to classify the majority class well, it completely ignored the minority class post-resampling. The model likely overfitted to the oversampled data during training and failed to generalise to the real-world distribution in the test set.

Weighted Loss Approach (TCNN only):

The TCNN with weighted loss offered a better balance, achieving a recall of 41.67% and an F1-score of 25.64% on the test set. This suggests that weighting the loss function was effective in making the model more sensitive to the minority class while avoiding the overfitting issues seen with SMOTE resampling.

Model predictability

In general, it is clear that the performance of both models cannot be considered at high levels and improvement is needed if it is to be used for prediction.

All models (apart from the random forest with resampling and hyperparameter tuning) struggled from overfitting. While strategies like early stopping and class weighting reduced overfitting to some extent, it remained a significant challenge. The notable performance drop from validation to test sets reflects how both models struggled to generalise.

Assessing predictive ability

Assessing the predictive power of the model would require us to set some benchmarks/baselines. We will be using the following four benchmarks:

1. Completely random predictions (50% probability to predict an extreme event)
2. Random predictions of extreme events with the same prevalence as in the historical data
3. Always predict “extreme event”
4. The benchmark of 50% which is benchmark that most people intuitively have in a binary classification problem. We will not be considering this in our analysis.

We will assume that the above baselines models/control groups are applied in the consolidated training and validation datasets and will be predicting on the test dataset in order to have a fair competition.

The prevalence of an extreme event if we combine training and validation is 21%. The prevalence of an extreme event in the testing dataset is 15.2%. The testing dataset has 342 observations

Model 1 will have Expected TP: 50% X 52 = 26, Expected FP 50% X 290 = 145

Model 2 will have Expected TP: 21% X 52 = 11, Expected FP 21% X 290 = 61

Model 3 will have Exact TP: 52, Exact FP 290 = 61

Model	Accuracy	Precision	Recall	F1 Score
Random 50%	50.00%	15.20%	50.00%	23.32%
Random with prevalence	70.18%	15.20%	21.00%	17.64%
Always Extreme Event	15.20%	15.20%	100.00%	26.40%

Table 3: Control Group

The Random Forest model with hyperparameter tuning and resampling outperforms all the above. Also, the TCNN model with weighted loss outperforms all apart from the last one. Hence, it is fair to say that compared to the above, the models demonstrate predictive ability for extreme events in stock prices.

Additionally, it is important to note that if these models are to be used for trading, a well-defined betting strategy would also need to be implemented. This involves not just relying on the model's predictions but also determining the amount of shares to buy or sell, as well as setting stop-loss and take-profit boundaries. These strategies can be designed in a way that maximises the expected profit, even if the model's predictive power is limited.

Without such a smart betting approach, the models developed in this assignment should not be used for real-world trading. Identifying an optimal betting strategy could be formulated as a reinforcement learning problem, where the agent learns to maximize long-term returns while managing risk.

Improvement Suggestions

Plan of attack

In this assignment, we are focusing on the modelling phase of a machine learning project. The modelling phase consists of three elements: the code (algorithm/model), the hyperparameters, and the data.

For many years, there has been a strong focus on a model-centric approach (largely influenced by online courses). This approach involves keeping the data unchanged and concentrating on improving the model and its accompanying hyperparameters. However, in our experience, this approach often yields limited performance improvements compared to a data-centric approach.

A data-centric approach focuses on fitting a fair machine learning model and then improving the dataset itself. This could involve data augmentation, feature engineering, or consulting with domain experts. While this approach can result in substantial performance gains, it also requires more research and effort.

Given the limited time available, our approach will be structured as follows:

1. **Error Analysis**:
   Conduct an Exploratory Data Analysis (EDA) on the incorrect predictions of our TCNN model to identify key areas of focus for improvement.

2. **Data-Centric Approach (Feature Engineering):**
   Engineer new features that could help reduce the number of incorrect predictions and enhance overall model performance.

3. **Model-Centric Approach (Hyperparameter Tuning and Architecture Optimisation):**
   Iterate through various hyperparameter combinations and architectures to determine the most effective model configurations.

4. **Validation Set Usage:**
   Use the validation set exclusively for tuning and testing during development. The test set will

only be used at the very end for final model evaluation to ensure that it remains an unbiased measure of model performance.

Error analysis

Firstly, let's take a look at the training data

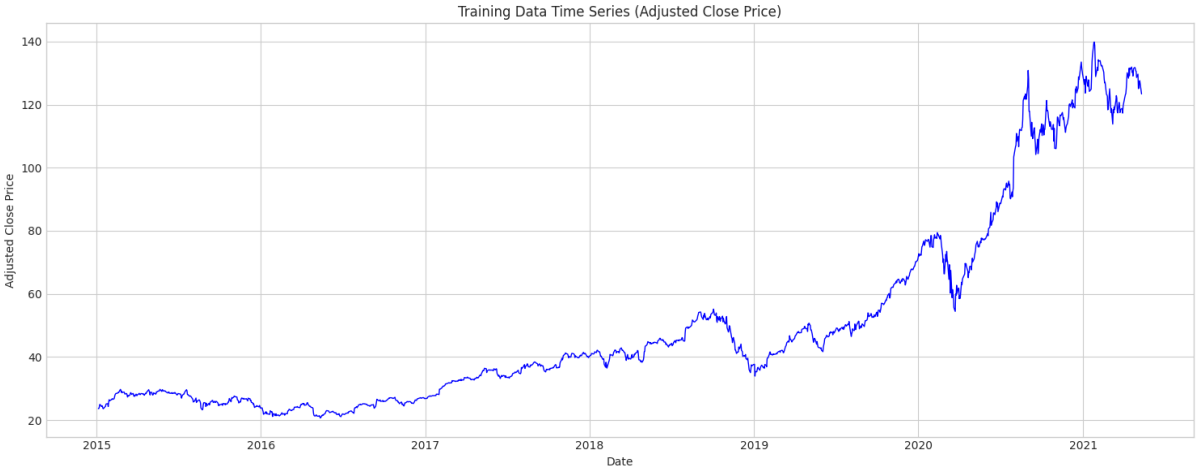

Fig 1: Adjusted Close Price per Date (Training dataset)

We can see that there's an upward trend since 2015 that took off mainly after Feb 2020. Afterwards, let's see the validation dataset and overlay the model predictions

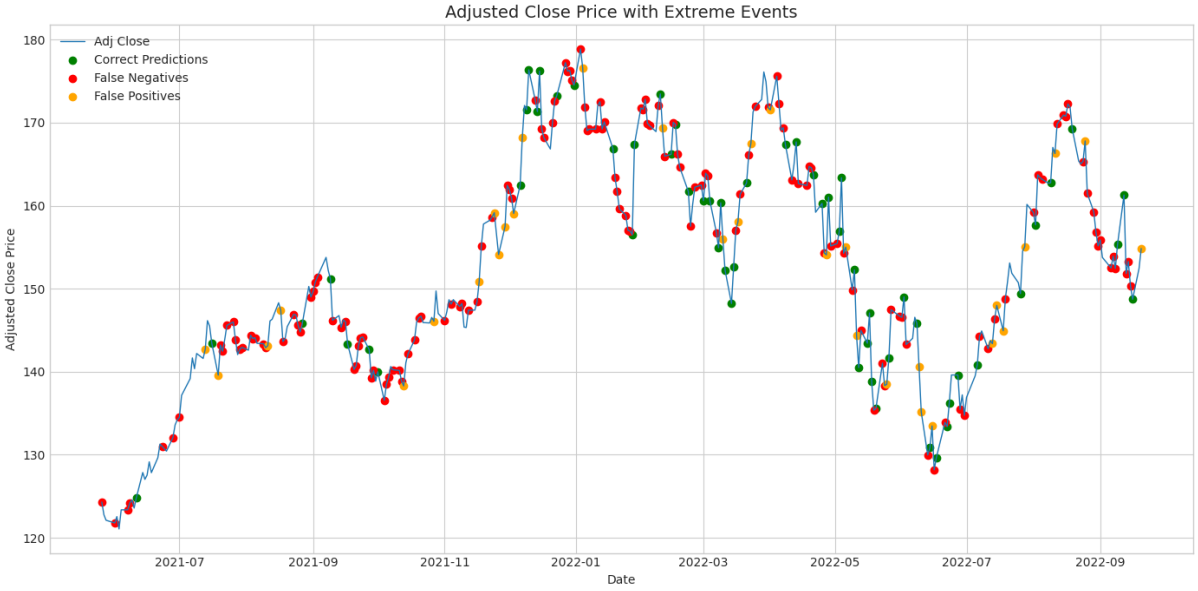

Fig 2: Adjusted Close Price per Date (Validation dataset) with Model Predictions

It seems that the model is struggling when there is an upwards trend or when the trend changes.

Next, we will analyse the distributions of our features at correct vs incorrect predictions

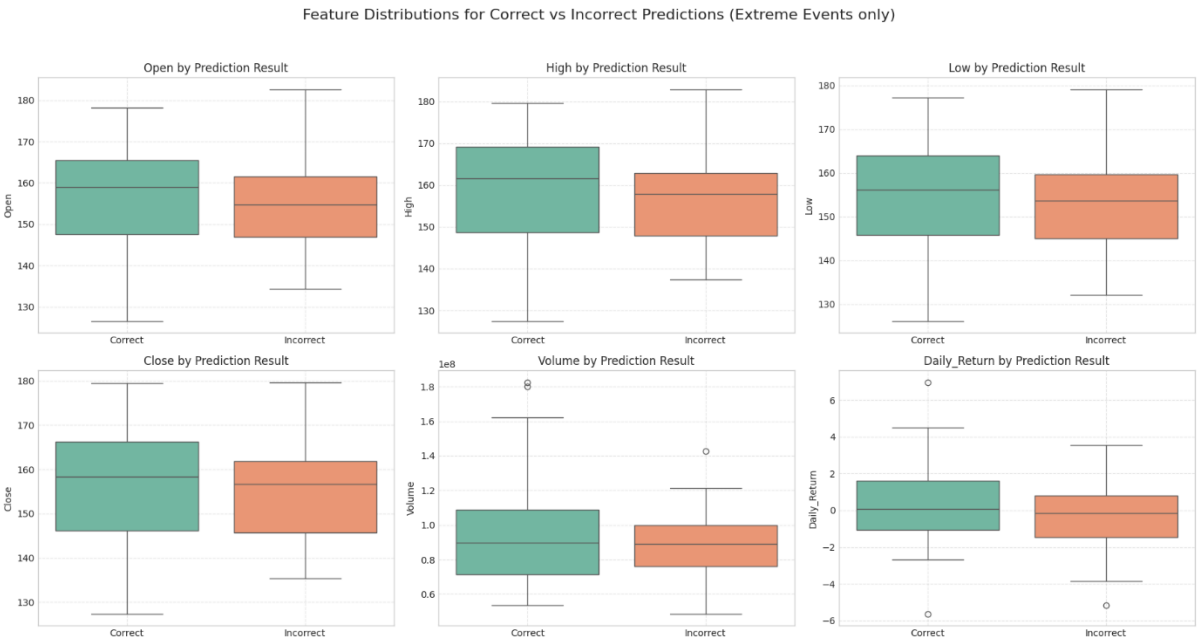

Fig 3: Distributions of our features at correct vs incorrect predictions (Extreme events only)

For some features the distribution seems to be different, for example, the daily return has higher values in the correct predictions vs the incorrect ones. Also, the range in the case of correct predictions is larger compared to the incorrect ones.

Next, we will be looking at the ACF and pACF graphs of the Adjusted Close:

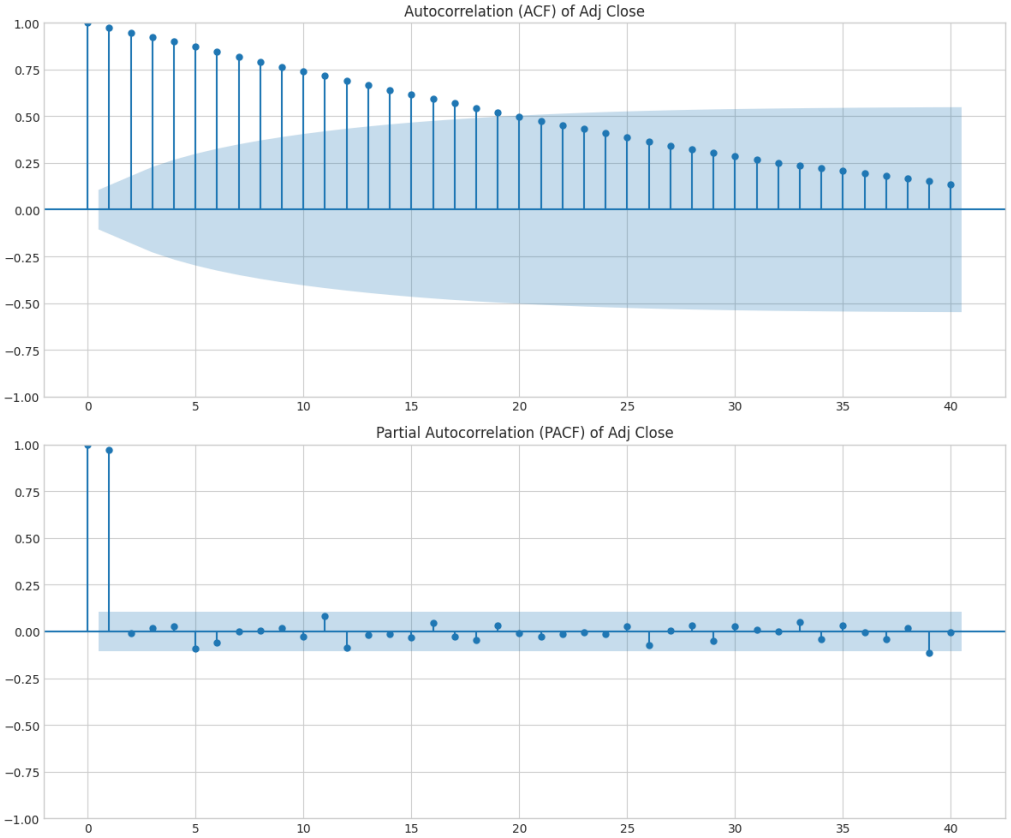

Fig 4: ACF and pACF graphs of Adjusted Close

It seems that the auto correlation diminishes very slowly whereas the partial auto correlation is only for lag 1. We will be plotting these after taking the 1st degree differences:

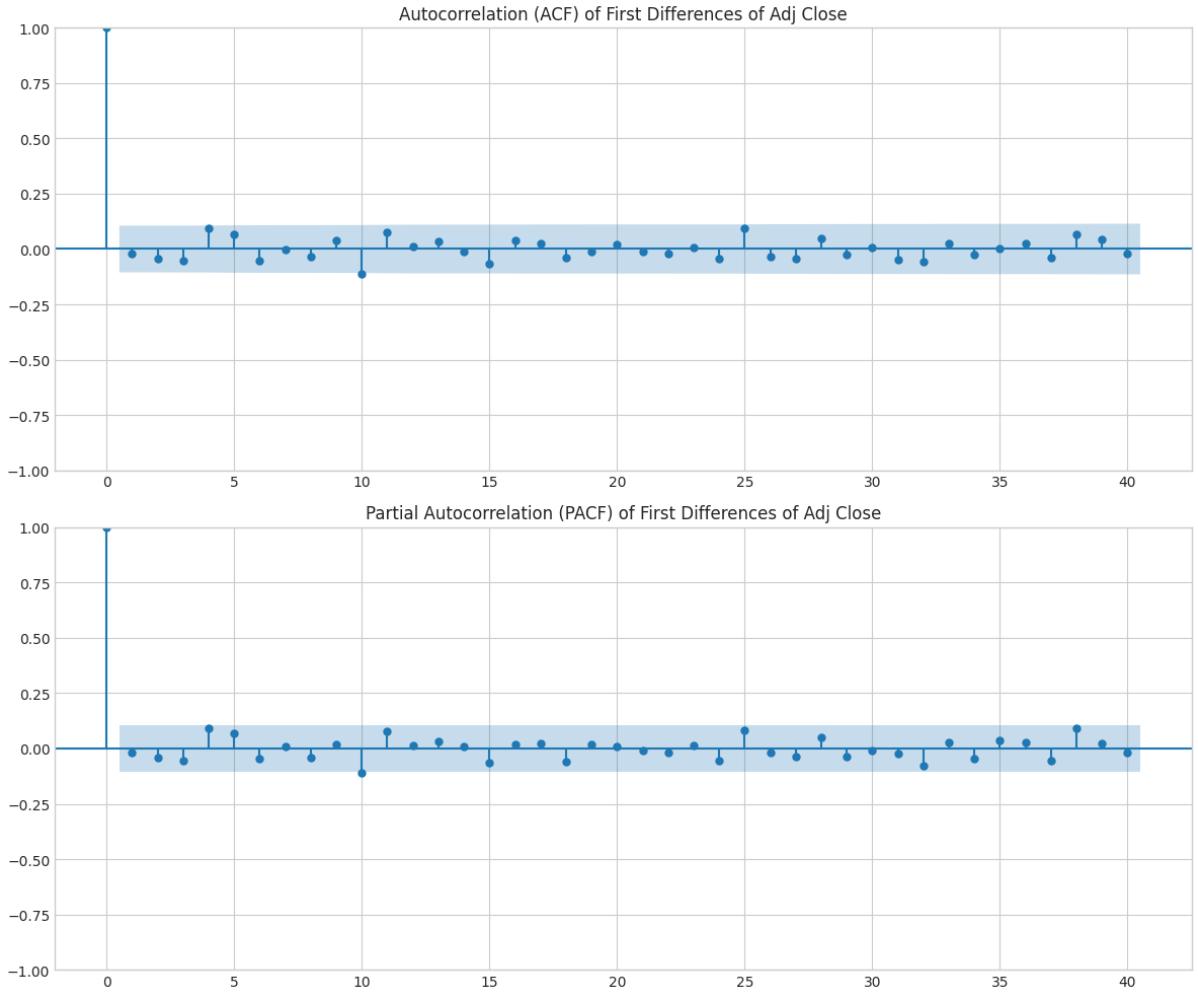

Fig 5: ACF and pACF graphs of Adjusted Close after lag 1 differences

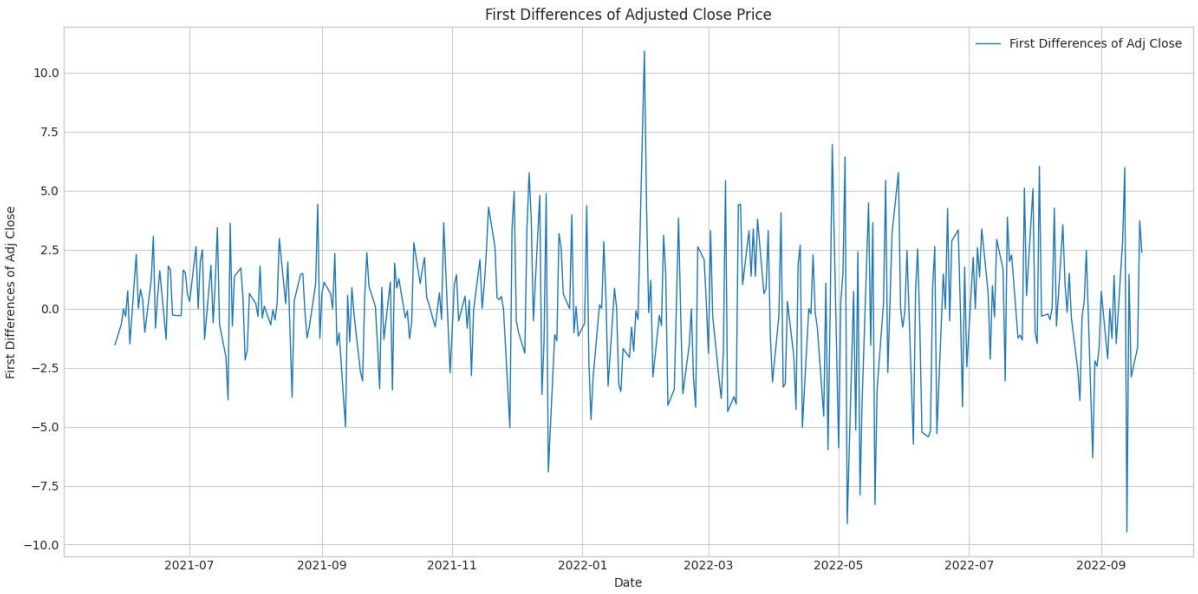

Fig 6: Time Series plot of Adjusted Close after lag 1 differences

From the above, it seems that after taking the first lag differences the time series almost becomes stationary. We say almost as the standard deviation seems to increase as time progresses.

Data-Centric approach

We will create features based on the above that will try to capture as much information as possible

- Simple Moving Average (SMA): A rolling average of the closing price over the past N days, used to gauge short-term price trends.
- Exponential Moving Average (EMA): A weighted moving average that applies more significance to recent prices, making it more responsive to short-term price changes.
- Bollinger Bands: Uses a moving average and standard deviation of prices to create upper/lower bands, helping identify volatility and potential overbought/oversold conditions.
- Average True Range (ATR): Measures the average range of price movements over a set window, capturing market volatility.
- Temporal Indicators (weekday, month start/end, etc.): Encodes calendar-based patterns (e.g., Mondays or month-end effects) that can affect stock returns.
- Volume-Weighted Average Price (VWAP): Combines price and volume to show the average price a stock has traded at throughout the day.
- Volume Spike: Compares current volume to a rolling mean of past volumes, highlighting unusually high trading activity.
- Lag Features (Close_Lag, Daily_Return_Lag): Shifts price or return data from previous days into current features, enabling the model to see historical context.
- World Events (e.g., Covid, Apple_Event): Flags for major external or company-specific events that may influence price movements.
- Rolling Standard Deviation (std rolling): Tracks the variability in Close or Daily_Return over a recent window, indicating volatility.
- Average Directional Index (ADX): Evaluates the strength of a price trend by comparing positive and negative directional movements.
- Relative Strength Index (RSI): Gauges momentum by comparing recent gains vs. losses, often used to spot overbought or oversold conditions.
- Rate of Change (ROC): Captures the percentage change in price over a set window, measuring momentum or speed of price changes.
- Trend Slope: Fits a linear regression to recent closing prices to identify the slope (rate of change) within a given window.
- EMA Diff: Calculates the difference between consecutive EMAs, highlighting acceleration or deceleration in price movement.

Model Centric approach

The Temporal Convolutional Neural Network (TCNN) architecture consists of two 1D convolutional layers designed to capture temporal patterns from sequential stock data. The first convolutional layer applies num_filters filters with a specified kernel_size and dilation, ensuring the receptive field covers temporal dependencies. The second convolutional layer doubles the number of filters and increases the dilation rate to capture longer-term dependencies.

ReLU activations are used after each convolutional layer for non-linearity.

A dropout layer is applied to prevent overfitting.

The output is flattened and passed through a fully connected layer, producing logits for the two classes (extreme event vs. no extreme event).

The hyperparameter grid that was used for model fine tuning is the following:

- **num_filters** ([32, 64, 128]): Controls the number of filters in the convolutional layers, affecting the model's capacity to capture complex patterns.
- **kernel_size** ([3, 5, 7]): Determines the size of the convolutional window, influencing how many time steps are considered when detecting patterns.
- **dilation** ([1, 2, 4, 8]): Controls the spacing between kernel elements, allowing the model to capture longer-term dependencies.
- **dropout_rate** ([0.2, 0.3, 0.5, 0.6]): Helps prevent overfitting by randomly dropping neurons during training.
- **learning_rate** ([0.01, 0.001, 0.0005, 0.0001]): Defines the step size for weight updates during training, impacting convergence speed and stability.
- **weight_minority** ([2, 4.06, 5, 7, 10]): Adjusts the class weighting in the loss function to address class imbalance and emphasize the minority class.

This grid was designed to balance model complexity, generalisation, and convergence.

Results

Unfortunately, the additional features and hyperparameters didn't yield an improve on the model's performance.

These are the stats in the validation and test dataset:

Dataset	Accuracy	Precision	Recall	F1 Score	Confusion Matrix
Validation	55.62%	34.27%	67.78%	45.52%	[[122, 117], [29,61]]
Test	65.44%	19.44%	44.68%	27.10%	[[193, 87], [26,21]]

Table 3: Final results after model improvement

Best parameters identified:

- Num of filters: 32
- Kernel size: 3
- Dilation: 1
- Dropout rate: 0.6
- Learning rate: 0.001
- Weight minority: 4.06

Conclusion

Reflecting on the project

This project set out to tackle the challenging task of predicting extreme price changes in Apple's stock using machine learning models, specifically a Random Forest (RF) and a Temporal Convolutional Neural Network (TCNN). Despite the inherent complexity of financial forecasting, the models demonstrated some encouraging (but modest) signs of predictive ability, particularly when benchmarked against random baselines. The RF model with resampling and TCNN with weighted loss stood out, showing improved recall and F1-scores, suggesting that the models were able to capture meaningful patterns in the data.

While the models alone did not achieve outstanding results, it's important to note that model predictions are just one part of a successful trading strategy. Equally critical is the design of a smart trading framework, which involves determining position sizes and setting stop-loss and take-profit boundaries. This opens up an exciting opportunity to explore reinforcement learning, where an agent could learn optimal trading actions based on the model's predictions, maximizing long-term profits while managing risk.

Future Work

To enhance model performance and better integrate predictions into a trading framework, the following steps are recommended:

- Feature Optimisation: Focus on generating more meaningful features. This can be achieved by consulting with experts, conducting research, and generating a broad range of features. Redundant and highly correlated features should then be removed to reduce noise and improve learning efficiency. Feature selection strategies may also be applied.
- Data Augmentation: Explore augmentation techniques, such as oversampling specific periods (e.g., months or days with higher volatility) to improve the model's robustness.
- Trading Strategy Integration: Develop a trading strategy that works alongside model predictions, optimising for risk and reward through reinforcement learning.
- Probability-Based Decisions: Instead of relying solely on binary predictions, leverage the model’s raw probabilities to make more nuanced trading decisions.
- Model Architecture Enhancements: Experiment with deeper networks, additional convolutional layers, or alternative architectures like LSTMs or Transformers to better capture temporal patterns.
- Custom Loss Functions: Design loss functions that place greater weight on correctly predicting extreme events, ensuring the model focuses on these critical cases.
- External Data: Incorporate external factors like macroeconomic indicators or sentiment analysis to provide the model with a richer context.
- Ensemble modelling: Try combining more than one predictive model

To conclude, while these two machine learning models have shown potential in this domain, significant advancements in both data and modelling techniques are required for reliable predictions. Implementing the above strategies, I believe there is potential to develop a model that can meaningfully contribute to predicting stock price movements.