CQF Final Project Report

Siqi He

January 2023

Introduction: Problem Statement and Process Design

In this study, the Long Short-Term Memory (LSTM) model is applied to predict the trend movements of Tesco stock prices. Trends are identified as daily logarithmic returns movements based on adjusted close prices. The target labels are determined according to a threshold above which returns are labelled as "positive" or else "negative". In the model development process, small positive returns below 0.1% are classified as negative. The primary data source for the experiment the Yahoo Finance API, where we extracted TESCO daily price data across the period 01/01/2014 to 25/12/2023, which was further split into training set, development set, and test set.

In addition to an end-to-end application of the LSTM framework in financial time series forecast for completion of the CQF final project, this study attempts to provisionally examine several other focuses. First, as deep learning is inherently data-hungry, we explore whether having a large breadth of produces good model performance when training an LSTM. Second, we experiment with several methods of feature selection in the context of a large initial feature set and produce candidate feature sets upon which different LSTM networks are calibrated. In particular, we refrain from using traditional linear selection tools (such as PCA and regression with penalties) as they fundamentally (to varying degrees) assume linear underlying relationships between the features and the target. We believe that excessively removing variables on a linear basis defeats the purpose of a deep learning algorithm, which is built to detect complex non-linear relationships with minimal human interventions. Instead, we explore other supervised and unsupervised classification techniques for dimensionality and feature reduction. Third, exhaustive model architectures of different depth are built and tested before tuning hyperparameters of high potential candidate architectures.

The feature engineering step consists of three parts. First, we take advantage of the Pandas-TA library to extract a large set of technical indicators, spreading across categories including candles, momentum, overlap, performance, statistics, trends, utility, volatility, volume. Next, several other data sources are added as candidate inputs to the LSTM model, including Sterling-dollar daily adjusted closing rate, FTSE100 daily adjusted closing price, CPI month-on-month percentage change as an indicator of inflation which was a prominent theme in the UK economy in the past years. The Fama French 3-factors for the European market was also added to the dataset. Furthermore, we calculate a series of generic indicators across multiple lookback periods, such as returns, moving averages, gaps, and Bollinger Bands. The Yang-Zhang drift independent volatility estimator was also constructed using the algorithm described in their paper (2000). Due to the disadvantage of having no professional subscriptions, features such as financial ratios and sentiment data were not part of our initial feature pool.

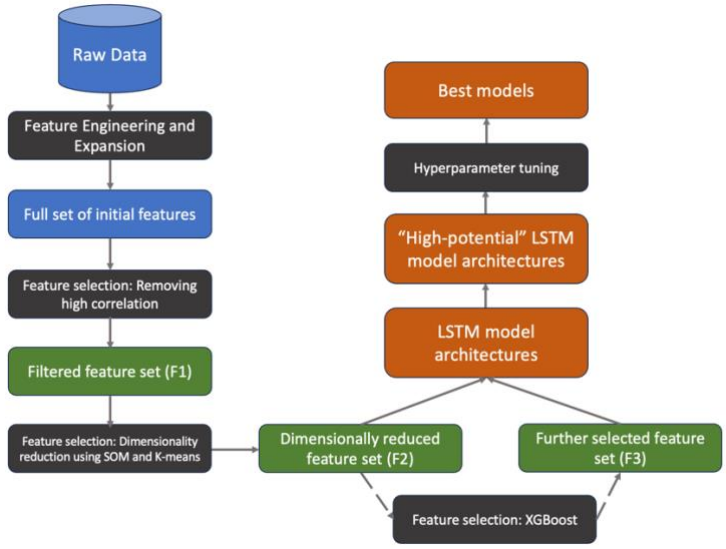
All features were cleansed accordingly including removing invalid features, addressing missing values and different lookback lengths. After exploratory data analysis, we scale the data using a customised scaler and splits the dataset into a training, development, and test sets. In the feature selection stage, we apply a combination of filter and embedded methods. First, multi-collinearity was addressed with an iterative algorithm that identifies the most frequently appearing features in high-correlated pairs and removes them with the overarching goal of preserving as many relevant features as possible in the candidate pool. Next, two unsupervised machine learning algorithms are collaboratively applied for dimensionality reduction: K-means clustering and Self-organising maps (SOM). Subsequently, features that are identified by both algorithms to be in the same 'cluster' are removed, keeping only those that have the shortest Euclidian distance from the K-means cluster centroid. Moreover, we further use supervised ensemble learning methods, XGBoost, to extract smaller feature sets based on the outputs of K-means and SOM. We fit an

XGBoost tree classifier on the dimensionally reduced training set and use the best performing models to identify important features for classifying the target label, based on ranking of feature importance during training and Shap analysis results. The three stages outlined above produces three feature sets, all of which are explored in parallel for model calibration.

After feature selection, we comprehensively experiment with an extensive range of LSTM model architectures using the training set and development set for all candidate sets produced in the previous step. While the baseline model has only one LSTM layer with dropout layer, the alternative model architectures tested range from having 1 to 3 layers with dropouts as well as dense layers before output. Additionally, we also test the ability of dense layers placed at the beginning of the sequential model to perform feature extraction (input compression) functionalities. To do this, we re-introduce the filtered feature set (after removing only highly correlated pairs) and use them to train model architectures with dense layers placed before LSTM layers. These fully connected dense layers allow for the possibility to algorithmically reduce feature dimensions in a non-linear fashion, while being and organic part of the wider LSTM entity. For each candidate feature set we provisionally identify three 'high potential' model architectures after an initial trial run based on values of validation metrics. These models are then proceeded to the hyperparameter tuning stage using the Bayesian Optimisation tuner.

Finally, back testing is carried out using a trading strategy with the out-of-sample test dataset, where a cumulative profit and loss over the period is used as a metric for model performance in a simplified real-world market setting.

A flowchart of the processes outlined is presented below.



Survey of Relevant Literature

The complex and evolutionary nature of financial markets has rendered the forecasting of financial data an empirically challenging task both in academia and industry. The rise of deep learning, however, has brought about a tide of emerging studies exploring the power of artificial neural networks to financial time-series prediction. The ability of deep learning algorithms to identify abstract features and patterns from data with limited human interventions makes it undeniably attractive compared to conventional linear models in machine learning (Chong et al.,

2017). In particular, the long-short term memory (LSTM) network has received much focus for its suitability for sequential models as it can selectively "remember" and "forget" information from the past, hence potentially aligning with the volatile nature of market movements.

A plethora of studies have sought to forecast stock price using the LSTM algorithm. For instance, Dezsi and Nistor (2015) comparatively studied the LSTM with classical econometric models on the daily returns of Romanian stocks with data spanning across 2001-2016. Chen et al. (2015) produced 3-day earning predictions of Chinese stocks using an LSTM-based approach using 15 years of data from 1990 to 2015. In many of such papers, the authors have highlighted that the success of a deep neural network should not be taken for granted but requires the need for careful design of the model components. In the input front, data pre-processing and feature selection are crucial steps in the development lifecycle to improve learning efficiency. For example, principal component analysis (PCA) has been employed as a common dimensionality reduction method (Tsai and Hsiao, 2010; Lin et al., 2009), as well as other approaches such as hybrid selection techniques (Lee, 2009), stepwise regression (Jeon et al., 2018), and cluster methods such as self-organising maps (Huang and Tsai, 2009). On the model development front, a careful selection of model parameters such as the number of units, hidden layers and learning rates is just as important for yielding satisfactory outcomes (Hussain et al., 2008).

Overall, due to the nature of deep learning to extract trends from a given dataset, it is difficult to design a one-size-fits-all approach to building an LSTM model. It is thereby no surprise that all such studies embark on an explorative rather than prescriptive nature, where the authors experiment with new combinations of information sources and toolkits for the problem in question. For instance, many studies experimented with hybrid variations of LSTM models, combining it with techniques such as transfer learning and paragraph vectors for analyse textual information (Kraus and Feuerriegel, 2018; Zhu et al., 2017; Akita et al., 2016). Other studies integrated LSTM with other methods in deep learning. Wu et al. (2021) and Liu et al. (2017) both leveraged the CNN-LSTM a framework where the convolutional layers are used to perform feature extraction in the dataset before input into the LSTM model for training. Zhou (2019) combined LSTM with Multi-layer perceptron (MLP) to predict next month returns with monthly returns and accounting data. Furthermore, results of classical machine learning models such as logistic regression and Gradient boosted trees, and ARIMA are often applied as a benchmark to evaluating sequential deep learning model performance (Selvin et al., 2018; Krauss et al., 2017).

Data Extraction and Feature Engineering

Our raw dataset consists of Tesco daily price and volume data spanning across 1st January 2015 to 25th December 2023. The choice of a 9-year data period (as opposed to 5) is to account for null value observation removals due to lookback/lookforward periods in feature extension, as well as the train-dev-test split, to ensure that the cleansed training set still covers a reasonable length of time.

Tesco is a multinational grocery store and general merchandise retailer based in the United Kingdom founded in 1919. According to Wikipedia figures, it is the third-largest retailer in the world in terms of gross revenues. Its stores have the strongest presence in the United Kingdom (where it claims a market share of approximately 28.4%) followed by Ireland, and several other countries in Europe. In the past decades, the firm has expanded from the grocery business to several other domains, such as clothing (Florence and Fred), telecom (Tesco mobile) and banking (Tesco bank).

Due to the absence of professional subscriptions, the initial dataset was extracted using Yahoo Finance with the `yfinance` Python package. We take the "`Adj Close`" price column as the

subject of study, and subsequently adjust the values in "`Open`", "`High`", and "`Low`" according to the difference between "`Close`" and "`Adj Close`" prices:

$$P_{Adj} = P_{Orig} \times \frac{P_{Adj\ Close}}{P_{Close}}$$

We subsequently drop the "`Close`" column and rename the "`Adj Close`" column as "`Close`".

In the first part of the feature engineering process, we add data from several other sources to capture trends in the macro economy and market. There are two main reasons for doing so: First, as a significant stakeholder in the UK market, Tesco is inevitably impacted by the overall shifts happening in its financial market environment. Second, the UK underwent a period of economic downturn with a concurrence of social and political events over the past years, including but not limited to Brexit, Covid-19, turbulence in political leadership and high inflation, all of which have left visible effects on the domestic economy. We attempted to capture these trends with the following indicators:

Indicator	**Data source**	**Manipulation**	**Feature name**
Sterling-dollar daily adjusted closing rate percentage change	Yahoo Finance using `yfinance` Python package	Percentage change of daily adjusted closing rate. Values joined to initial dataset by timestamp index.	"`GBPUSD=X_Close`"
FTSE100 daily adjusted closing price percentage change	Yahoo Finance using `yfinance` Python package	Percentage change of daily adjusted closing rate. Values joined to initial dataset by timestamp index.	"`^FTSE100_Close`"
CPI month-on-month percentage change	Office for National Statistics Consumer Price Inflation tables	CPI Month-on-month percentage change, monthly values extended to all daily timestamps for that month. Values joined to initial dataset by timestamp index.	"`cpi_mom`"
Fama-French 3-factors for the European market	Dartmouth Ken French Database	Values joined to initial dataset by timestamp index.	"`Mkt-RF`" "`SMB`" "`HML`"

The second step of feature engineering is feature extension, where we calculate a variety of technical indicators / estimators using both Python technical analysis packages and self-coded algorithms. First, taking advantage of the Pandas Technical Analysis (TA) library, we call the `.ta.strategy('All')` method to automatically extract 277 features. The full list of features and their calculations can be found in the package documentations of libraries Pandas-TA and TA-Lib. Below are examples of some indicators included in the library.

- Exponential moving average (EMA)
- Fibonacci's weighted moving average (FWMA)
- Moving average convergence divergence (MACD)

- Relative strength index (RSI)
- Commodity channel index (CCI)
- Mass Index

Beyond the features extracted with pre-built libraries, we also calculate several standard key features in financial market analysis across a range of lookback periods, listed below. It is noted that some of such features may share notable similarities with the 277 auto-extracted features. This concern is taken care of in later steps of multicollinearity analysis / removal and dimensionality reduction processes.

Indicator	Calculation (`df`: dataset, `i`=lookback)	Lookback	Feature name in data (`i`=lookback)
Intraday price range: Open – Close	`df['Close'] / df['Open'] - 1`	0	`OC_0`
Intraday price range: High – Low	`df['High'] / df['Low'] - 1`	0	`HC_0`
Intraday price gap	`df['GAP_0'] = df['Open'] / df['Close'].shift(1) - 1`	0	`GAP_0`
Lagged returns	`df['LOGRET'].shift(i)`	1, 2, 3, 4, 5, 21, 63	`LOGRET_LAG_{i}`
Cumulative log returns	`df['LOGRET'].rolling(i).sum()`	1, 2, 3, 4, 5, 21, 63	`CUM_LOGRET_{i}`
Percentage change of close price	`df['Close'].pct_change(i)`	1, 2, 3, 4, 5, 21, 63	`PCHG_{i}`
Percentage change of volume	`df['Volume'].pct_change(i)`	1, 2, 3, 4, 5, 21, 63	`VCHG_{i}`
Scaled moving average closing price	`df['Close'] / df['Close'].rolling(i).mean() - 1`	1, 2, 3, 4, 5, 21, 63	`MA_SCALED_{i}`
Scaled moving average of volume	`df['Volume'] / df['Volume'].rolling(i).mean() - 1`	1, 2, 3, 4, 5, 21, 63	`VMA_SCALED_{i}`
Rolling mean of intraday price range: Open – Close	`df['OC_0'].rolling(i).mean()`	1, 2, 3, 4, 5, 21, 63	`OC_{i}`
Rolling mean of intraday price range: High – Low	`df['HC_0'].rolling(i).mean()`	1, 2, 3, 4, 5, 21, 63	`HC_{i}`
Rolling mean of intraday price gap	`df['GAP_0'].rolling(i).mean()`	1, 2, 3, 4, 5, 21, 63	`GAP_{i}`
Volatility	`df['LOGRET'].rolling(i).std()`	1, 2, 3, 4, 5, 21, 63	`STD_{i}`

Bollinger bands (Upper band)	`df['Close'].rolling(i).mean() + df['Close'].rolling(i).std() * multiplier`	1, 2, 3, 4, 5, 21, 63	`UB_{i}`
Bollinger bands (Lower band)	`df['Close'].rolling(i).mean() - df['Close'].rolling(i).std() * multiplier`	1, 2, 3, 4, 5, 21, 63	`LB_{i}`

Furthermore, we add two indicators for weekdays. Incorporating weekday in the feature set allow us to capture potential weekday-specific patterns such as end-of-week effects. We use the sine and cosine transformations on values of 1-7 (Monday – Sunday). Replacing numeric assignments or excessive dummy variables with this continuous representation effectively captures the cyclical and periodic nature of weekday variable, encouraging the learning algorithm to pick up on the underlying patterns.

Indicator	Calculation	Lookback	Feature name in data
Weekday indicator: sine transformation	See function `transform_days_column`	N/A	`dsin`
Weekday indicator: cosine transformation	See function `transform_days_column`	N/A	`dcos`

Lastly, we build an algorithm for the Yang-Zhang drift independent volatility estimator in accordance with the formula described in their published paper (Yang and Zhang, 2000). The Yang-Zhang volatility estimator is a historical volatility estimator that allows for the drift and the opening price jumps, accounting for intraday price movements commonly found in stock price data. A strength of the estimator is that only public available price data "`Open`", "`Close`", "`High`", and "`Low`" is needed for its computation, all of which are readily contained in our dataset.

Indicator	Calculation	Lookback	Feature name
Yang-Zhang volatility estimator	See function `drift_independent_vol_estimator`	21, 63	"`yz_tesco_{lookback}`"

With all features engineered, we check for missing values. From examining the head and tail of the dataset, we can see that a significant portion of missing values are present due to lookback / look-forward periods applied in feature extension. Moreover, several columns also contain a large number of missing values, amounting to the hundreds and even thousands. To address all such issues, we perform the following data cleansing steps.

First, we remove all features where more than 10% of the observations contain null values. Then, we remove observations at the start and end of the data period with consecutive null values as a result of missing data (in case of the Fama-French factor where data was only available up to 31st October 2023) or due to applying lookback / look-foreard periods in feature extension. Finally, we apply forward fill to all other missing values in the middle of the dataset (only 6 missing values remain after the first two steps, which we deem to be sufficiently immaterial hence not needing further investigation). The forward fill method is chosen over other methods (backward fill and interpolation) for the following reasons.

- Forward fill does not assume knowledge of any future information as missing values are replaced by the most recent information available up to the current point in time.

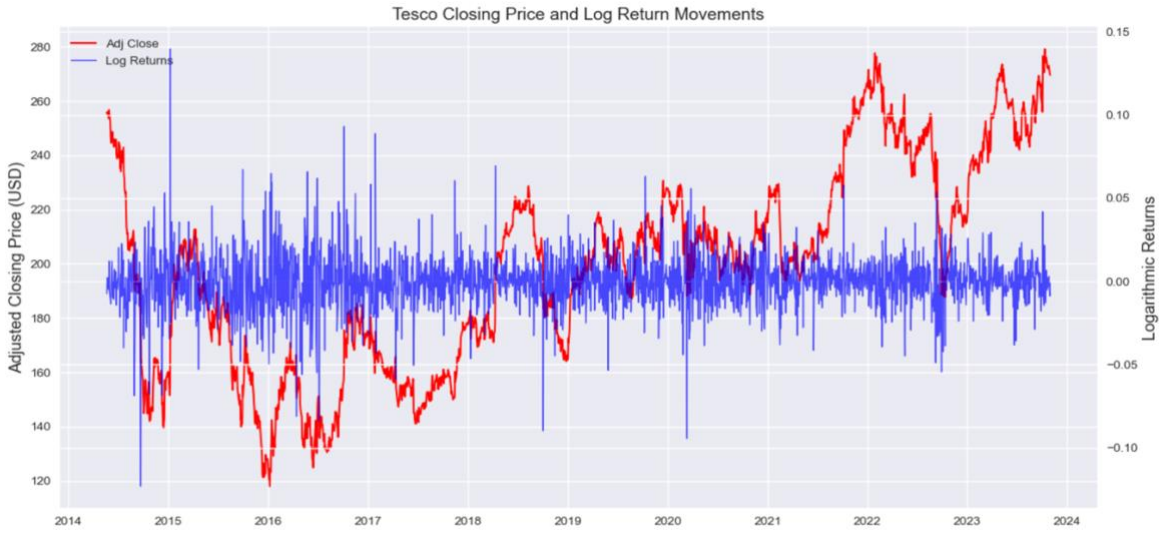
- Backward fill can introduce lookahead bias and data leakage since it brings forward information which should be unknown at the current timestamp.
- Forward fill considers the possibility of delays in data reporting and processing in market movements. The method gives leeway for such latencies by always reflecting up-to-date market conditions while preserving the effects on the market.
- Interpolation methods has a smoothing effect of diminishing the significance any triggers or events, thus potentially causing loss of crucial information.

Finally, we remove all features generated in the TA library that has homogenous values across all timestamps as they demonstrate no movements throughout our examined period and hence very unlikely to have predictive power for binary target values. The full dataset has 354 features and 2387 daily observations across dates 21st May 2014 to 31st October 2023.

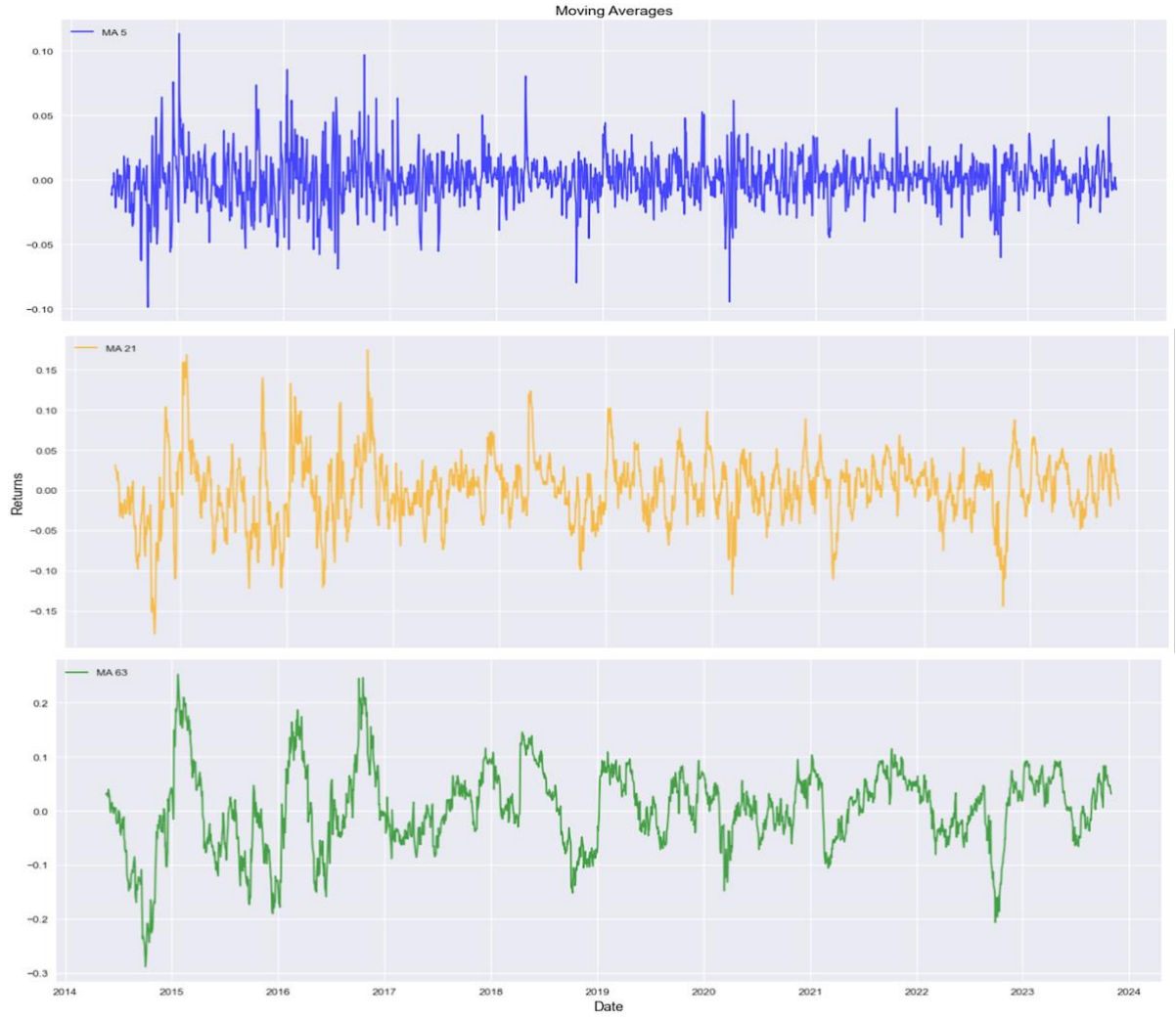
Exploratory Data Analysis (EDA) and Data Pre-processing

There are two primary purpose of exploratory data analysis (EDA). First, we wish to develop an understanding of the internal structure of our data, such as their value range, correlations, and identifiable patterns. Second, an appropriate choice of scaler should be chosen in accordance with our findings. Third, to provide information on the need for applying a filter method in feature selection for removing multicollinearity. As the initial feature set is rather large, we break down exploratory data analysis into several steps and approach the feature set from sub classes and categories.

First, we take a look at the key columns that are most relevant to our stated problem definition, namely the closing price and daily logarithmic returns. The graphs below plot the movement of adjusted closing price along with 10-day exponential moving average and logarithmic returns across our selected period respectively. We can see that there seem to be a presence of outliers as the stock price movements a have been quite volatile, featuring several drastic increases and drops at various points of time.



Next, we visualise the moving averages across several lookback periods (5-day, 21-day and 63-day).



As the candidate features generated by the TA library is quite numerous, we break them into sub-categories according to their strategy class for closer inspection: volatility, volume, trend,

statistics, momentum, overlap, and candlestick. For each class, we plot the values across our time period to visualise their movements and value range. Moreover, we also compute the Z-score for values in each field to identify the presence of outliers (3 standard deviations away from the mean). A Shapiro Wilk test ($p<0.05$) is then performed for each feature to evaluate whether the feature values approximate a gaussian distribution. A full list of feature graphs and test results can be found in the Jupyter notebooks attached.

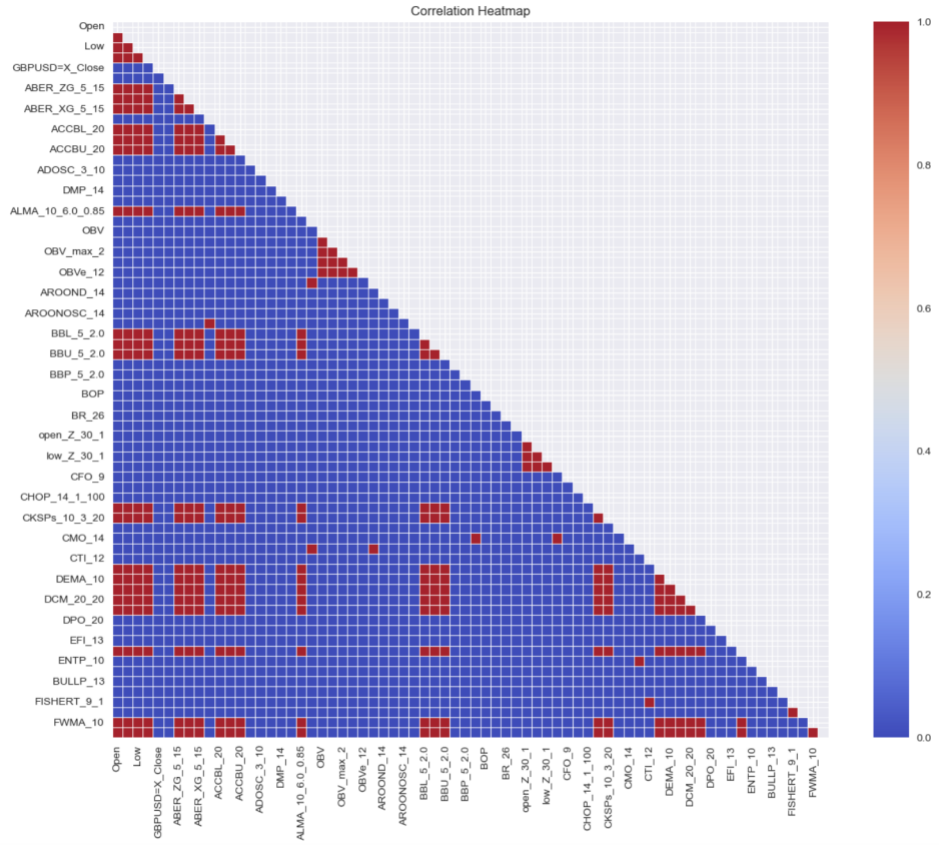
The Shapiro Wilk test outcomes indicates that almost no features resemble a normal distribution. Meanwhile, the calculated Z-scores show that 75 out of 354 features contain outliers that are more than 3 standard deviations from the mean. This renders both the Standard Scaler and Min Max Scaler to be inadequate for the data structures, as the former assumes normal distribution within each feature values, and the latter is particularly poor at handling outliers with all values map linearly onto the range of [0, 1]. Therefore, we select the Robust Scaler, which overcomes both extreme values and non-normal underlying distributions. The Robust Scaler scales data using the median and inter quantile range (IQR), hence also preserves the relative spread of raw values. The formula for the Robust Scaler works by first subtracting values by the median and then dividing it by its IQR.

$$X_{scaled} = \frac{X - X_{median}}{IQR_X}$$

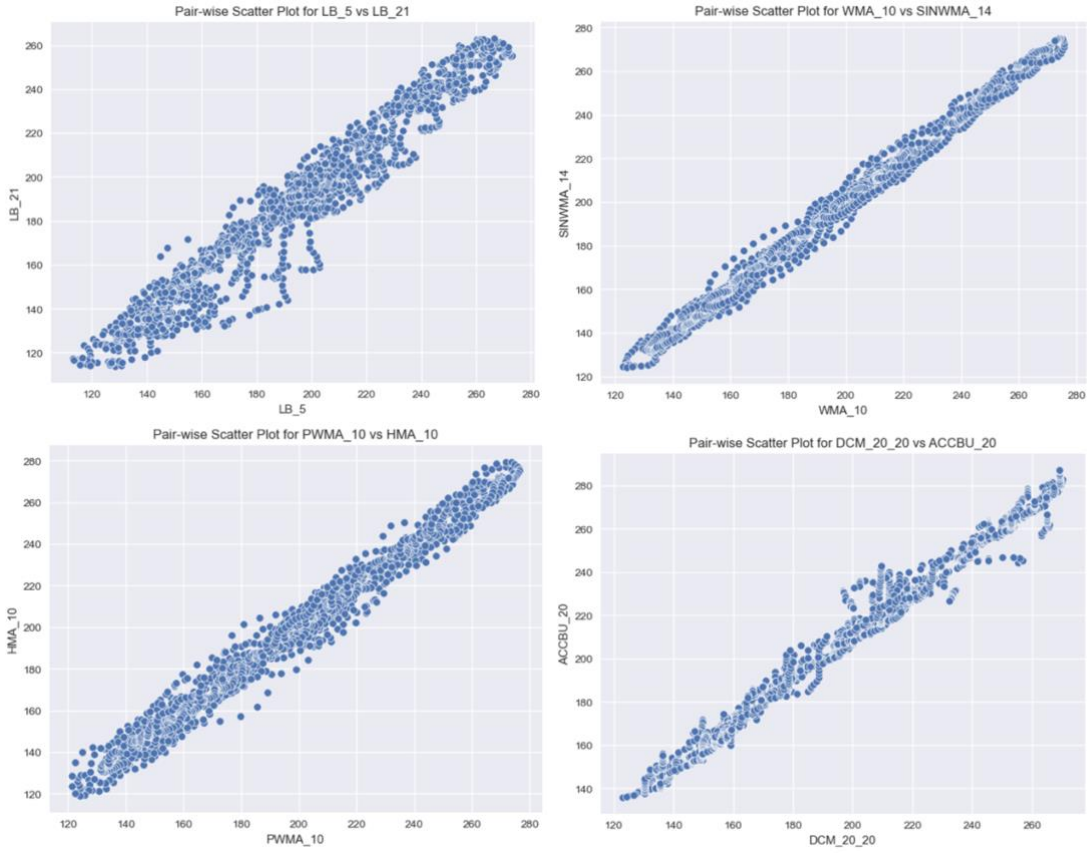
However, it was noted from EDA that categorical features have strong presence in the dataset. All candlestick features computed by the TA library (despite categorical in nature) are represented by large values with minimum and maximum of -100 and 100. In this case, using the Robust Scaler can result in information loss of original categories and can skew the results in later stages of analysis.[1] To address this, we use the Min-Max Scaler for all categorical fields to map their values to the range of [0, 1]. Moreover, it was decided that the numerical weekday representations `dsin` and `dcos` should be left unscaled as they already lie in a reasonable range of [-1, 1] as per our initial trigonometry transformation.

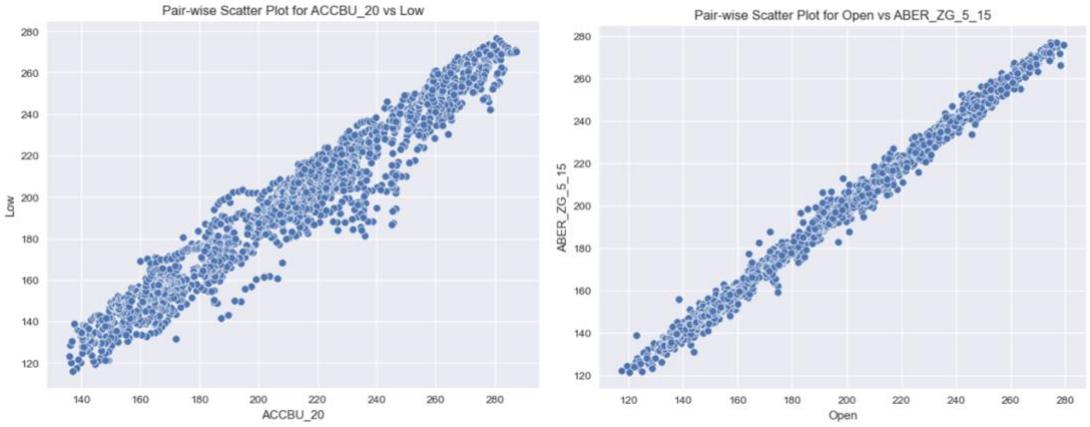
The next step is to check for multicollinearity. As the set is too large to fit onto an eligible correlation heatmap, we split the data into 4 groups to visualise their correlations. Categorical variables are excluded from the view. Note that this is only a tentative view of the multicollinearity pairs, as by definition splitting into groups we are unable to view the relationship between every single pair, and not all feature labels are displayed on the graphs due to space limitations. Nonetheless, it does indicatively show us that highly correlated pairs are present in the dataset. The first out of four correlation matrices (threshold is set at $\rho=0.9$) is presented below (the rest can be found in Jupyter notebooks attached).

[1] It was found that applying the Robust Scaler to candle stick features result in their strong dominance in dimensionality reduction stages, in one trial, close to 70% of selected features were in the candlestick class.



To take a closer look at correlated features, we produce scatter plots of a few highly correlated pairs.





From the clearly linear distributed scatters, we can infer that multicollinearity must be removed as part of the feature selection process.

To wrap up this section, we split the data into three sets: training, development, and test set, at a ratio of 6:2:2. Then we fit the customised scalers to our training set, and then use it to transform all three sets. A temporary raw label is created by taking the logarithmic return field values and shifting them by -1, so that the value on day *n* is the return of day *n+1*. This label is set tentatively as in the model building phase we will be experimenting with various threshold values for distinguishing positive and negative returns.

The shapes of the datasets are summarised in the table below.

Dataset	Shape	Dataset	Shape
`X_train_df_scaled`	1431, 354	`Y_train_df_scaled`	1431, 1
`X_dev_df_scaled`	477, 354	`Y_dev_df_scaled`	477, 1
`X_test_df_scaled`	478, 354	`Y_test_df_scaled`	478, 1

Feature Selection: Overview

Feature selection for this project undertakes an experimentative stance rather than a definitive attitude. In other words, we select several sets of candidate features to be used for model calibration. This approach is warranted by several rationales. First, acknowledging that the unique strength of deep learning models to capture and abstract intricate patterns and relationships from complex datasets, we are aware that aggressive feature reduction may prevent the algorithm from performing to its full potential, such as identifying multivariate dependencies between features that are undetectable by classic feature selection algorithms. Second, there is no universally applicable framework for feature selection in a deep learning context for financial time series, hence it's more prudent from a study point of view to explore a range of candidate models and compare performance than to restrict the scope to a pre-defined feature set which may be suboptimal or inappropriately selected. Third, as the initial feature spans across many categories, it is difficult to know definitively up-front which ones the actual LSTM algorithm will deem relevant during the model training phase. Hence, the decision is made to downstream varying degrees of "autonomy" to the LSTM algorithm to evaluate appropriateness of different feature sets. We hope that this could reduce any biases introduced by excessive human intervention and enable the algorithm to autonomously select features from a generous pool.

Additionally, the feature selection methods in this study deliberately avoids linear methods such as principal component analysis and regularised regression. Instead, we apply non-linear

techniques that assumes no inherent linear relationships to preserve any non-linear interactions which may not be discernible with linear techniques. In general, the steps in this section a "conservatism", where we seek to prevent overboard premature discarding of valuable information. All feature selections below are performed using the training set.

Feature selection: Filtering high correlations.

As examined in EDA, high correlation (>0.9) is present amongst the feature set. Thus, our first step is to make sure that the data does not contain such highly correlated pairs as part of the filter method in feature selection. We run an initial correlation matrix computation and look at the pairs identified.

	f1	f2	rho
0	Open	High	0.996928
1	Open	Low	0.997241
2	Open	Close	0.995802
3	Open	ABER_ZG_5_15	0.994482
4	Open	ABER_SG_5_15	0.994004
...	...	...	...
3020	CUM_LOGRET_21	PCHG_21	0.997631
3021	VMA_SCALED_21	VMA_SCALED_63	0.933151
3022	UB_21	LB_21	0.921389
3023	CUM_LOGRET_63	PCHG_63	0.994169
3024	HC_63	STD_63	0.917560

The dataset contains a total of 3024 pairs where the correlation coefficient is greater than 0.9. A further inspection is made to see the number of pairs a feature is associated with.

```
Number of features with high correlations: 191
Features and the number of high-corr pairs they are associated with:
{'QTL_30_0.5': 76, 'VIDYA_14': 76, 'IKS_26': 76, 'MEDIAN_30': 76, 'SUPERT_7_3.0': 75, 'CKSPs_10_3_20': 75, 'Open': 74, 'SSF_10_2': 74, 'SMA
_10': 74, 'SINWMA_14': 74, 'RMA_10': 74, 'PWMA_10': 74, 'T3_10_0.7': 74, 'OHLC4': 74, 'MIDPRICE_2': 74, 'MIDPOINT_2': 74, 'MCGD_10': 74, 'L
R_14': 74, 'KCUe_20_2': 74, 'SWMA_10': 74, 'TRIMA_10': 74, 'TEMA_10': 74, 'KCLe_20_2': 74, 'High': 74, 'LB_21': 74, 'UB_21': 74, 'LB_5': 7
4, 'UB_5': 74, 'LB_4': 74, 'UB_4': 74, 'LB_3': 74, 'UB_3': 74, 'LB_2': 74, 'UB_2': 74, 'ZL_EMA_10': 74, 'WMA_10': 74, 'VWMA_10': 74, 'VWAP_
D': 74, 'KCBe_20_2': 74, 'WCP': 74, 'KAMA_10_2_30': 74, 'DCU_20_20': 74, 'ABER_XG_5_15': 74, 'ACCBL_20': 74, 'ACCBM_20': 74, 'ACCBU_20': 7
4, 'ALMA_10_6.0_0.85': 74, 'BBL_5_2.0': 74, 'BBM_5_2.0': 74, 'BBU_5_2.0': 74, 'Close': 74, 'Low': 74, 'LDECAY_5': 74, 'DEMA_10': 74, 'DCL_2
0_20': 74, 'DCM_20_20': 74, 'EMA_10': 74, 'ABER_ZG_5_15': 74, 'FWMA_10': 74, 'HA_open': 74, 'HA_high': 74, 'HA_low': 74, 'HA_close': 74, 'H
ILO_13_21': 74, 'HL2': 74, 'HLC3': 74, 'HMA_10': 74, 'JMA_7_0': 74, 'ITS_9': 74, 'ABER_SG_5_15': 74, 'HWM': 73, 'HWL': 73, 'HWMA_0.2_0.1_0.
1': 73, 'CKSPl_10_3_20': 71, 'HWU': 71, 'AO_5_34': 13, 'MACD_12_26_9': 13, 'STCmacd_10_12_26_0.5': 13, 'SMI_5_20_5': 12, 'SMIs_5_20_5': 11,
'RSX_14': 11, 'KST_10_15_20_30_10_10_10_15': 11, 'TSI_13_25_13': 11, 'ISA_9': 10, 'RSI_14': 10, 'CMO_14': 10, 'MACDs_12_26_9': 10, 'QQE_14_
5_4.236_RSIMA': 10, 'APO_12_26': 9, 'PPO_12_26_9': 9, 'MA_SCALED_3': 8, 'QQE_14_5_4.236': 7, 'D_9_3': 7, 'LB_63': 7, 'high_Z_30_1': 7, 'MA_
SCALED_5': 7, 'TSIs_13_25_13': 7, 'TOS_STDEVALL_U_2': 6, 'LOGRET': 6, 'PCTRET_1': 6, 'CUM_LOGRET_1': 6, 'PCHG_1': 6, 'MA_SCALED_2': 6, 'TOS
_STDEVALL_LR': 6, 'MA_SCALED_63': 6, 'TOS_STDEVALL_L_1': 6, 'BIAS_SMA_26': 6, 'TOS_STDEVALL_L_2': 6, 'TOS_STDEVALL_U_1': 6, 'TOS_STDEVALL_U
_3': 6, 'ZS_30': 6, 'close_Z_30_1': 6, 'low_Z_30_1': 6, 'TOS_STDEVALL_L_3': 6, 'MA_SCALED_21': 6, 'MA_SCALED_4': 6, 'COPC_11_14_10': 6, 'FI
SHERT_9_1': 5, 'PCHG_21': 5, 'open_Z_30_1': 5, 'SLOPE_1': 5, 'CUM_LOGRET_21': 5, 'PPOs_12_26_9': 5, 'CUM_LOGRET_2': 4, 'PCHG_2': 4, 'OBVe_1
2': 4, 'KSTs_9': 4, 'SQZPRO_20_2.0_20_2_1.5_1': 4, 'OBV': 4, 'OBV_min_2': 4, 'OBVe_4': 4, 'ENTP_10': 4, 'SQZ_20_2.0_20_1.5': 4, 'OBV_max_
2': 4, 'STOCHk_14_3_3': 4, 'STOCHd_14_3_3': 4, 'MOM_10': 3, 'FISHERTs_9_1': 3, 'PCHG_3': 3, 'TRUERANGE_1': 3, 'CUM_LOGRET_3': 3, 'K_9_3':
3, 'VAR_30': 2, 'UB_63': 2, 'WILLR_14': 2, 'TRIX_30_9': 2, 'CUM_LOGRET_63': 2, 'STDEV_30': 2, 'HC_1': 2, 'VMA_SCALED_4': 2, 'HC_4': 2, 'CUM
_LOGRET_4': 2, 'PCHG_4': 2, 'ISB_26': 2, 'HC_0': 2, 'MAD_30': 2, 'CTI_12': 2, 'ROC_10': 2, 'PGO_14': 2, 'HC_21': 1, 'HC_3': 1, 'NATR_14':
1, 'VMA_SCALED_5': 1, 'HC_5': 1, 'STD_4': 1, 'STD_5': 1, 'CUM_LOGRET_5': 1, 'PCHG_5': 1, 'VMA_SCALED_21': 1, 'VMA_SCALED_63': 1, 'PCHG_63':
1, 'HC_63': 1, 'VMA_SCALED_3': 1, 'PDIST': 1, 'PVI_1': 1, 'RVGI_14_4': 1, 'STOCHRSId_14_14_3_3': 1, 'STOCHRSIk_14_14_3_3': 1, 'STCstoch_10_
12_26_0.5': 1, 'TRIXs_30_9': 1, 'STC_10_12_26_0.5': 1, 'RVGIs_14_4': 1, 'OC_0': 1, 'ABER_ATR_5_15': 1, 'OC_1': 1, 'GAP_0': 1, 'GAP_1': 1,
'VTXP_14': 1, 'PVT': 1, 'ATRr_14': 1, 'STD_63': 1}
```

A total of 191 features are associated with at least a pair of high correlation, with their frequencies ranging from 76 (highest) to 1 (lowest). However, we are aware that not all 191 features need to be dropped to eliminate high correlations. For instance, if feature 'A' is highly correlated with features 'B', 'C' and 'D', then it makes sense to only remove feature A to effectively eliminate all three pairs of high correlations. Hence, we write an algorithm to iteratively identify the feature that contribute to the highest number of highly correlated pairs, remove it from our feature set, and then recalculate a correlation matrix after its removal. In this way, we slowly eliminate all highly correlated pairs shown above by removing the least number of features in total.

This progress of iterative process is shown below as part of the notebook output.

```
feature removed: 1. QTL_30_0.5  | pairs remaining: 2958
feature removed: 2. MEDIAN_30  | pairs remaining: 2883
feature removed: 3. VIDYA_14  | pairs remaining: 2809
feature removed: 4. IKS_26  | pairs remaining: 2736
feature removed: 5. CKSPs_10_3_20  | pairs remaining: 2665
feature removed: 6. SUPERT_7_3.0  | pairs remaining: 2595
feature removed: 7. LB_3  | pairs remaining: 2527
feature removed: 8. LB_21  | pairs remaining: 2460
feature removed: 9. JMA_7_0  | pairs remaining: 2394
feature removed: 10. ITS_9  | pairs remaining: 2329
feature removed: 11. KAMA_10_2_30  | pairs remaining: 2265
feature removed: 12. KCBe_20_2  | pairs remaining: 2202
feature removed: 13. LDECAY_5  | pairs remaining: 2140
feature removed: 14. LB_5  | pairs remaining: 2079
feature removed: 15. HA_high  | pairs remaining: 2019
feature removed: 16. HA_close  | pairs remaining: 1960
feature removed: 17. Open  | pairs remaining: 1902
```

……

```
feature removed: 138. HWL  | pairs remaining: 17
feature removed: 139. ISB_26  | pairs remaining: 16
feature removed: 140. KSTs_9  | pairs remaining: 15
feature removed: 141. K_9_3  | pairs remaining: 14
feature removed: 142. LOGRET  | pairs remaining: 13
feature removed: 143. MAD_30  | pairs remaining: 12
feature removed: 144. OBV  | pairs remaining: 11
feature removed: 145. OC_0  | pairs remaining: 10
feature removed: 146. PVI_1  | pairs remaining: 9
feature removed: 147. QQE_14_5_4.236  | pairs remaining: 8
feature removed: 148. RVGI_14_4  | pairs remaining: 7
feature removed: 149. SQZPRO_20_2.0_20_2_1.5_1  | pairs remaining: 6
feature removed: 150. STC_10_12_26_0.5  | pairs remaining: 5
feature removed: 151. STD_4  | pairs remaining: 4
feature removed: 152. STOCHRSId_14_14_3_3  | pairs remaining: 3
feature removed: 153. TOS_STDEVALL_L_3  | pairs remaining: 2
feature removed: 154. VMA_SCALED_21  | pairs remaining: 1
feature removed: 155. ZS_30  | pairs remaining: 0
```

In total, 155 features were dropped from the initial feature set, leaving 199 features remaining. We refer to this feature set "`set_hc`" in later sections. "`set_hc`" is one of our candidate feature sets to be used for model calibration.

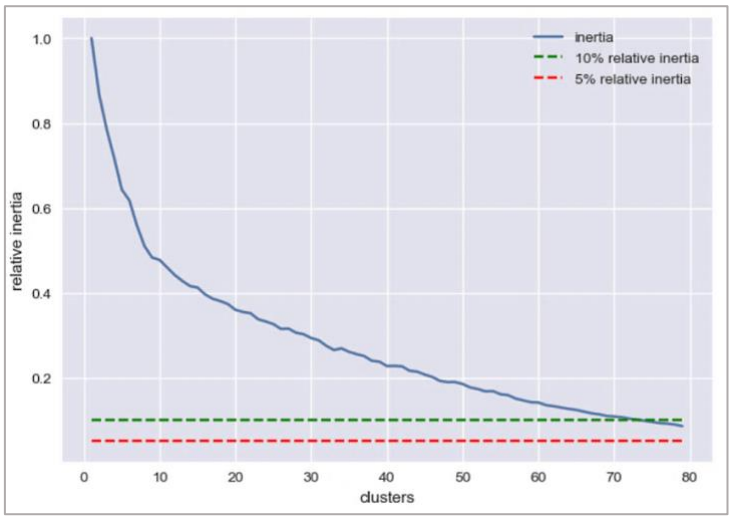
Feature selection: Dimensionality reduction with SOM and K-means clustering.

Next, we leverage unsupervised dimensionality reduction techniques to further narrow down the candidate pool. The Self-Organising map (SOM) and K-means clustering unsupervised learning method are jointly applied to our dataset. Both the methods are centroid-based clustering techniques commonly used for dimensionality reduction. Due to the large set of features (many of which are different variations of particular indicators), we expect that the K-means and SOM clusters should have a decent amount of overlap, i.e., some features that are allocated to the same cluster by the K-means algorithm should also be classified in the same cell in the SOM. Our approach is to find such overlaps and reduce dimensionality by dropping features identified to be similar by both methods. This technique is aligned with our 'conservativism' in the feature selection phase mentioned previously.

The mechanics of this iterative feature reduction process is described as follows. First, we obtain the results of the SOM and K-Means clustering as two collections, each collection containing a number of clusters computed by the respective algorithms. Each cluster contains all features sorted into that cluster by the respective K-Means and SOM model. In the K-Means clusters, features are ranking from closest to furthest from the centroid. Then, we iterate through all clusters in the K-Means cluster collection. For every pair of features in a K-Means cluster, we check if they are also present in the same SOM cluster, if so, we keep only the feature that has the shortest distance from its K-Means centroid. If not, both features are retained. The skeleton logic is described in the pseudo-code as below.

```
# Declare selected feature list as empty list
# for every cluster_k in the K-Means collection

  # If cluster_k contains <2 elements, continue
  # Else, get the first feature f_k_1 in cluster_k

    # Identify the cluster_s in SOM_collection that contains f_k_1
            # Check every feature f_k_n+1 at index 1+ in K-Means cluster_k,

              # If f_k_n+1 exists in the SAME cluster_s
                      # Add f_k_1 to selected feature list
                # Else, add both f_k_1 and f_k_n+1 to selected feature list

                # Remove both features from cluster_k in K-means collection and SOM collection
                        (so that they won't be checked again in later iterations)
```

We produce an elbow plot to obtain the optimal number of clusters for K-Means clustering, setting an exploratory range from 1-80.



The plot indicatively suggests obtaining 10% relative inertia the model requires approximately at least 70 clusters.

Next, we train a Self-Organising Map with a map size of 7×7.[2] Several combinations of hyperparameters were tested, we decide on a learning rate of 0.4 and a sigma value of 1 to minimise the quantisation error. Below is a visualisation of the SOM.

[2] Various map sizes were tried in the experimental process, including 8×8 which is the recommended size as $(5\times199^{1/2})^{1/2}$= 8.398. However, as our selection algorithm only reduces features classified into the same cluster by both unsupervised learning methods, we believe that a smaller 7×7 map size is compensated by the in-built conservatism. Moreover, map sizes of 8×8 and above tend to generate larger selected feature sets after reduction, amounting to 110+ which we deem to be excessive for our problem context.


```
[ 20000 / 20000 ] 100% - 0:00:00 left
quantization error: 12.675531656805957
topographic error: 0.19597989949748743
```

Although the quantisation error seems to be relatively large, this is intuitively aligned with the high dimensionality of input data.

A sample of the K-Means clustering results is presented here. As shown, features are grouped by clusters and sorted by their Euclidean distance to the cluster centroid. The full output can be found in the Jupyter notebook outputs attached.

	cluster	feature	distance
106	0	J_9_3	7.351783e+00
191	0	WILLR_14	9.368179e+00
76	0	CCI_14_0.015	1.037229e+01
86	0	EBSW_40_10	1.070161e+01
168	0	STOCHRSIk_14_14_3_3	1.167086e+01
71	0	BBP_5_2.0	1.462504e+01
158	0	SMIo_5_20_5	1.499664e+01
176	0	UO_7_14_28	1.710237e+01
85	0	DPO_20	2.577468e+01
51	1	CDL_UPSIDEGAP2CROWS	2.041658e+00
14	1	CDL_EVENINGDOJISTAR	2.110470e+00
10	1	CDL_COUNTERATTACK	2.209673e+00
2	1	CDL_2CROWS	2.395787e+00
11	1	CDL_DARKCLOUDCOVER	2.395787e+00
15	1	CDL_EVENINGSTAR	2.454691e+00
44	1	CDL_STALLEDPATTERN	2.483620e+00
36	1	CDL_ONNECK	2.650568e+00
7	1	CDL_ADVANCEBLOCK	2.980378e+00
48	1	CDL_THRUSTING	3.363216e+00
41	1	CDL_SHOOTINGSTAR	4.020991e+00
19	1	CDL_HANGINGMAN	4.798490e+00
59	1	THERMOI_20_2_0.5	9.420180e+00
57	1	SQZ_OFF	1.680212e+01
178	2	VCHG_2	1.457144e-15
16	3	CDL_GAPSIDESIDEWHITE	2.279278e+00
49	3	CDL_TRISTAR	2.445572e+00
40	3	CDL_SEPARATINGLINES	2.501521e+00
52	3	CDL_XSIDEGAP3METHODS	2.512799e+00
47	3	CDL_TASUKIGAP	2.831349e+00
165	17	STD_3	1.201169e+01
166	17	STD_5	1.201169e+01
116	18	LOGRET_LAG_5	1.729905e-15
169	19	STOCHd_14_3_3	7.728483e+00
82	19	CTI_12	1.029214e+01
121	19	MFI_14	1.078925e+01
190	19	VTXP_14	1.092801e+01
193	19	close_Z_30_1	1.107641e+01
66	19	AROONOSC_14	1.124194e+01
67	19	AROONU_14	1.394995e+01
89	19	FISHERTs_9_1	1.425831e+01
108	19	KVO_34_55_13	1.783699e+01
63	19	ADOSC_3_10	2.170878e+01
122	20	Mkt-RF	1.198011e-15
123	21	NATR_14	1.028342e+01
69	21	ATRr_14	1.241867e+01
101	21	HC_5	1.682125e+01
164	21	STD_21	1.725111e+01
171	21	THERMOma_20_2_0.5	1.883953e+01
146	22	PVOL	1.100100e+01
188	22	VMA_SCALED_63	1.100100e+01
162	23	STDEV_30	9.529326e+00
184	23	VHF_28	9.529326e+00
187	24	VMA_SCALED_5	7.280204e+00
186	24	VMA_SCALED_3	1.586829e+01
180	24	VCHG_3	1.897159e+01

The final set of features after the dimensionality reduction algorithm consists of 96 features. We can see that clearly the unsupervised clustering algorithms has successfully removed indicators which we expect to display similar behaviour. For example, the feature set now has only 4 candlestick features from the original selection of 53. Scaled volume moving average now only has values for 5-day. The full set of output features is listed below.

```
['AD', 'ADX_14', 'AROOND_14', 'ATRr_14', 'BBB_5_2.0', 'BBP_5_2.0', 'CDL_GAPSIDESIDEWHITE', 'CDL_INNECK', 'CDL_RICKSHAWMAN', 'CDL_UPSIDEGAP2
CROWS', 'CFO_9', 'CHOP_14_1_100', 'CKSPl_10_3_20', 'CMF_20', 'CTI_12', 'DMN_14', 'DMP_14', 'EFI_13', 'ER_10', 'FISHERTs_9_1', 'GAP_1', 'GAP
_2', 'GAP_21', 'GAP_3', 'GAP_4', 'GAP_63', 'GBPUSD=X_Close', 'HC_1', 'HC_2', 'HML', 'INC_1', 'J_9_3', 'KURT_30', 'KVOs_34_55_13', 'LOGRET_L
AG_1', 'LOGRET_LAG_2', 'LOGRET_LAG_21', 'LOGRET_LAG_3', 'LOGRET_LAG_4', 'LOGRET_LAG_5', 'LOGRET_LAG_63', 'MACDh_12_26_9', 'MASSI_9_25', 'MF
I_14', 'Mkt-RF', 'NATR_14', 'OC_1', 'OC_2', 'OC_21', 'OC_4', 'PCHG_2', 'PCHG_4', 'PCHG_63', 'PDIST', 'PGO_14', 'PPOh_12_26_9', 'PSARaf_0.02
_0.2', 'PSL_12', 'PVOL', 'PVO_12_26_9', 'PVOh_12_26_9', 'PVR', 'PVT', 'QS_10', 'RVGIs_14_4', 'RVI_14', 'SMB', 'SMIo_5_20_5', 'SMIs_5_20_5',
'SQZ_OFF', 'STCstoch_10_12_26_0.5', 'STDEV_30', 'STD_2', 'STD_3', 'STD_63', 'STOCHd_14_3_3', 'SUPERTd_7_3.0', 'THERMO_20_2_0.5', 'THERMOs_2
0_2_0.5', 'UI_14', 'VCHG_1', 'VCHG_2', 'VCHG_21', 'VCHG_3', 'VCHG_4', 'VCHG_5', 'VCHG_63', 'VHF_28', 'VMA_SCALED_5', 'VTXM_14', 'WILLR_14',
'^FTSE_Close', 'cpi_mom', 'dcos', 'dsin', 'vol_yz_tesco_21']
```

We refer to this feature set "`set_ks`" in later sections. "`set_ks`" is one of our candidate feature sets to be used for model calibration.

Feature selection: Further reduction using XGBoost feature importance and Shap.

After dimensionality reduction with unsupervised learning methods, the feature set is still abundant. Acknowledging the potential drawbacks of large feature sets, in particular the possible risk of overfitting when the sample size isn't sufficient to train all relevant parameters, we incorporate another tier of non-linear feature selection based on `set_ks`, to create a smaller input feature set. We exploit the XGBoost technique to further reduce the feature set to a smaller subset. The XGBoost is an ensemble algorithm commonly used for feature selection as it allows the users to view the relative importance of each feature during model training. XGBoost is also a non-linear model with the power to capture complex patterns and relationships within the dataset.

The following procedures were performed.

First, set a base threshold for the target labels. To do this, we take the raw Y values (logarithmic returns) constructed in feature engineering, and apply a threshold of 0.1%, where anything above this threshold will be classified as positive (1) and below classified as negative (0). We notice slight class imbalances (see below), which are addressed with class weights.

Class	Ratio	Weight
0	0.917308	
1	1.099078	0.8346

Second, we fit the training data using an XGBoost classifier applying the class weights constructed. Parameter `col_sample_bytree` (sample ratio of features used when building each tree) is set at 0.8 with maximum tree depth of 5 and gamma value of 0.1 to prevent overfitting.

Third, we rank feature importance for the model from highest to lowest and apply a cut-off value of 0.1, i.e., features with an importance of below 0.1 are dropped from further selection.

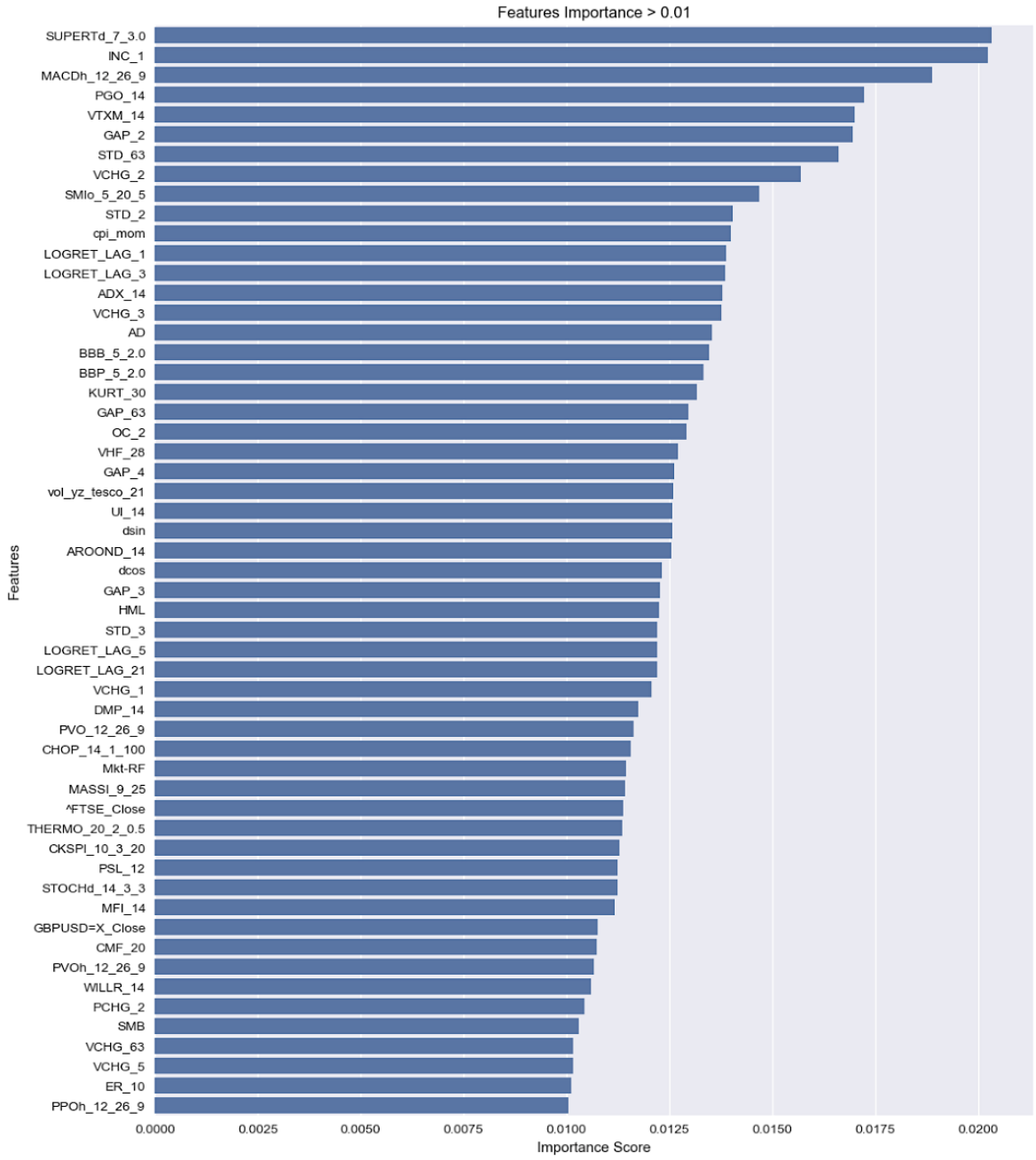
Fourth, we run a Shap (Shapley Additive Explanations) analysis on the XGBoost classifier. The Shap values are based on Shapley values in game theory and represents the contribution of feature in the classifier. The Shap summary plot visualises the association between feature values with their effects on the output. Based on the plot, we manually pick out features that appear to have low impacts on the Shap values. This is evaluated by two factors: the mean absolute Shap value and the distribution of the feature high and low values in relation to the positive and negative Shap value directions (X-axis). Recognising that we did not tune the hyperparameters of the XGBoost model, leeway is given to features that appear to have low mean absolute Shap values but seem to distinguish well between high and low values when impacting the positivity and negativity.

The following features were manually selected and dropped.

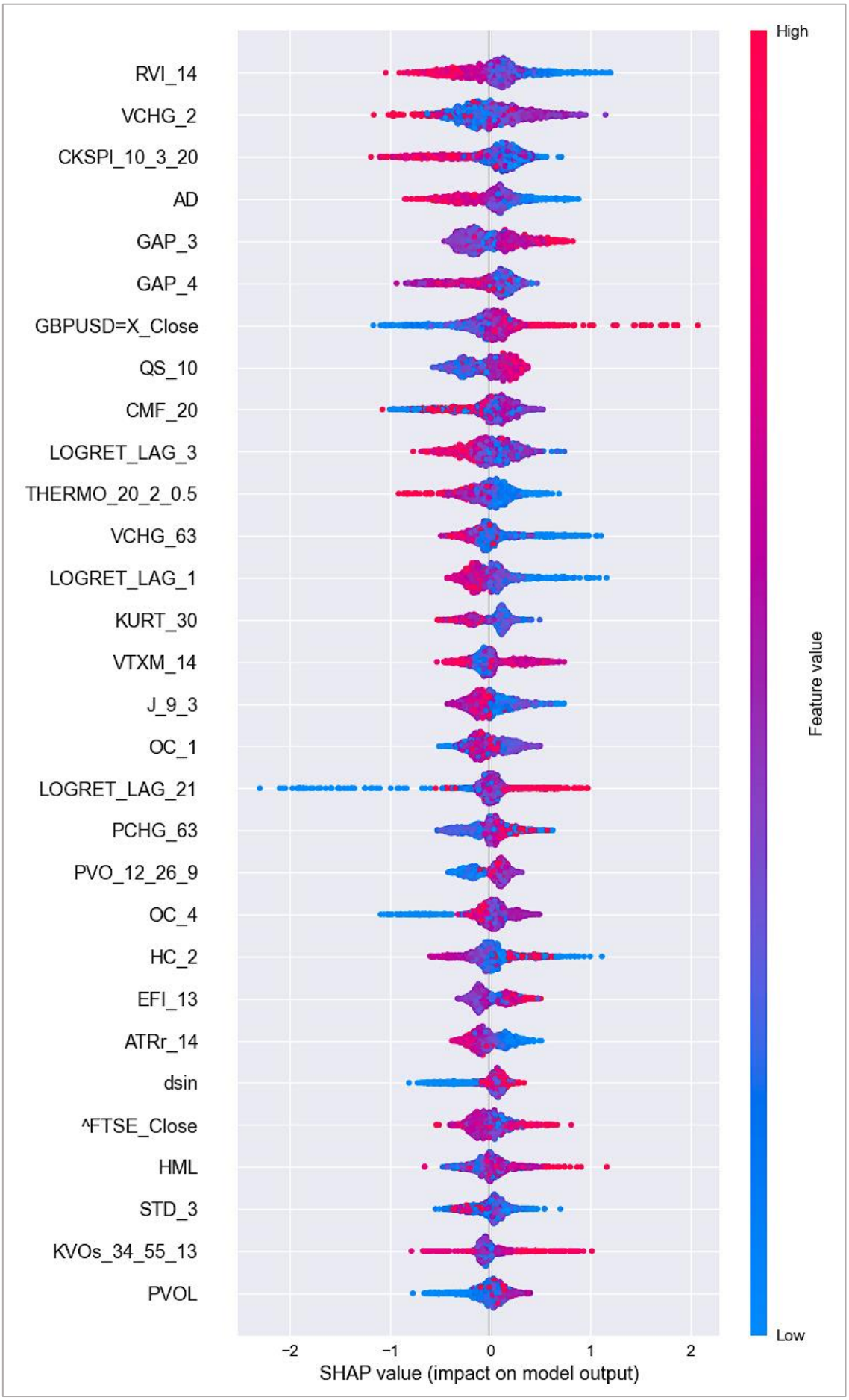
```
Features dropped from SHAP analysis:

['cpi_mom', 'FISHERTs_9_1', 'OC_2', 'PSARaf_0.02_0.2', 'PCHG_4', 'AROOND_14', 'INC_1', 'MFI_14', 'STOCHd_14_3_3', 'STCstoch_10_12_26_0.5',
'PSL_12', 'dcos', 'PVR', 'SUPERTd_7_3.0', 'SQZ_OFF', 'CDL_UPSIDEGAP2CROWS', 'CDL_RICKSHAWMAN', 'CDL_INNECK', 'CDL_GAPSIDESIDEWHITE', 'CFO_
9', 'EFI_13', 'STD_2', 'LOGRET_LAG_5', 'GAP_21', 'GAP_1', 'VCHG_3', 'VCHG_5', 'ADX_14', 'GAP_63', 'SMIs_5_20_5', 'SMIo_5_20_5', 'VMA_SCALED
_5', 'CHOP_14_1_100', 'PGO_14']
```

The final subset of features is the intersect of those retained by both evaluations: feature importance and Shap values. 37 features were selected, referred to as "`set_xg`" in later sections. "`set_xg`" is one of our candidate feature sets to be used for model calibration.

The horizontal bar plot for feature relative importance during XGBoost model training above cut-off threshold is shown below.



Below is the Shap summary plot for the top 30 features with highest Shap values (subset due to spacing limitation), plots of the full set of features can be found in the Jupyter notebook results attached.


RVI_14
VCHG_2
CKSPI_10_3_20
AD
GAP_3
GAP_4
GBPUSD=X_Close
QS_10
CMF_20
LOGRET_LAG_3
THERMO_20_2_0.5
VCHG_63
LOGRET_LAG_1
KURT_30
VTXM_14
J_9_3
OC_1
LOGRET_LAG_21
PCHG_63
PVO_12_26_9
OC_4
HC_2
EFI_13
ATRr_14
dsin
^FTSE_Close
HML
STD_3
KVOs_34_55_13
PVOL
−2
−1
0
1
2
SHAP value (impact on model output)
High
Feature value
Low

Model development: Architecture shortlisting.

We perform model building on three features sets: `set_hc`, `set_ks`, and `set_xg`. The first stage is to investigate an extensive range of model architectures. Specifically, we assess a total of 27 architectures, incorporating a range of LSTM layers and pre-output layer configurations. Below is a full list of the architectures explored with notes on the datasets they are calibrated on.

Additionally, we also experiment with the addition of dense layers positioned at the top of the sequential model for feature extraction, as an alternative to the dimensionality procedures undertaken previously. For feature sets KS and XG, we add the option of having 2 or 3 dense layers atop of the first LSTM layer and set the hidden unit size accordingly to reduce input dimensions. In our trial run for architecture shortlisting, the first pre-LSTM dense layer is set to have the same number of units as the feature size. Where three dense layers are present, second pre-LSTM dense layer is configured with 80 hidden units and the third pre-LSTM dense layer is configured with 40 units. Where only two dense layers are present, the second dense layer is configured with 40 units. The numbers are strategically chosen to maintain high-level consistency between LSTM layer inputs across candidate feature sets (the XG feature set has an input of 37 features).

The following list of model architectures are initially tested.

Model ID	Architecture	Dataset	Shortlisted
1 (Baseline)	1 LSTM layer with dropout layer	KS	
1	1 LSTM layer with dropout layer	XG	
2	0 pre-LSTM dense layer 1 LSTM layer with dropout 1 pre-output dense layer	KS, XG	XG
3	0 pre-LSTM dense layer 1 LSTM layer with dropout 2 pre-output dense layer	KS, XG	
4	0 pre-LSTM dense layer 2 LSTM layer with dropout in between 0 pre-output dense layer	KS, XG	XG
5	0 pre-LSTM dense layer 2 LSTM layer with dropout in between 1 pre-output dense layer	KS, XG	
6	0 pre-LSTM dense layer 2 LSTM layer with dropout in between 2 pre-output dense layer	KS, XG	
7	0 pre-LSTM dense layer 3 LSTM layer with dropout in between 0 pre-output dense layer	KS, XG	XG
8	0 pre-LSTM dense layer 3 LSTM layer with dropout in between 1 pre-output dense layer	KS, XG	
9	0 pre-LSTM dense layer 3 LSTM layer with dropout in between 2 pre-output dense layer	KS, XG	
10	2 pre-LSTM dense layer 1 LSTM layer with dropout 0 pre-output dense layer	KS, HC	
11	3 pre-LSTM dense layer 1 LSTM layer with dropout	KS, HC	

	0 pre-output dense layer		
12	2 pre-LSTM dense layer 1 LSTM layer with dropout 1 pre-output dense layer	KS, HC	HC, KS
13	3 pre-LSTM dense layer 1 LSTM layer with dropout 1 pre-output dense layer	KS, HC	
14	2 pre-LSTM dense layer 1 LSTM layer with dropout 2 pre-output dense layer	KS, HC	HC
15	3 pre-LSTM dense layer 1 LSTM layer with dropout 2 pre-output dense layer	KS, HC	
16	2 pre-LSTM dense layer 2 LSTM layer with dropout in between 0 pre-output dense layer	KS, HC	
17	3 pre-LSTM dense layer 2 LSTM layer with dropout in between 0 pre-output dense layer	KS, HC	
18	2 pre-LSTM dense layer 2 LSTM layer with dropout in between 1 pre-output dense layer	KS, HC	
19	3 pre-LSTM dense layer 2 LSTM layer with dropout in between 1 pre-output dense layer	KS, HC	
20	2 pre-LSTM dense layer 2 LSTM layer with dropout in between 2 pre-output dense layer	KS, HC	
21	3 pre-LSTM dense layer 2 LSTM layer with dropout in between 2 pre-output dense layer	KS, HC	HC
22	2 pre-LSTM dense layer 3 LSTM layer with dropout in between 0 pre-output dense layer	KS, HC	
23	3 pre-LSTM dense layer 3 LSTM layer with dropout in between 0 pre-output dense layer	KS, HC	KS
24	2 pre-LSTM dense layer 3 LSTM layer with dropout in between 1 pre-output dense layer	KS, HC	
25	3 pre-LSTM dense layer 3 LSTM layer with dropout in between 1 pre-output dense layer	KS, HC	
26	2 pre-LSTM dense layer 3 LSTM layer with dropout in between 2 pre-output dense layer	KS, HC	KS
27	3 pre-LSTM dense layer 3 LSTM layer with dropout in between 2 pre-output dense layer	KS, HC	

The baseline model consists of only 1 LSTM layer with dropout layer, built upon the KS dataset. The model is fitted with 15 epochs and a patience of 10, with 20 hidden units. The training output of the baseline model is shown here.

```
Now fitting model: LSTM_model_arch_1
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 LSTM1 (LSTM)                (None, 20)                9360

 Dropout1 (Dropout)          (None, 20)                0

 Output (Dense)              (None, 1)                 21

=================================================================
Total params: 9381 (36.64 KB)
Trainable params: 9381 (36.64 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/50
12/12 [==============================] - 1s 21ms/step - loss: 0.7922 - accuracy: 0.4830 - precision: 0.4475 - recall: 0.5550 - val_loss: 0.73
85 - val_accuracy: 0.4934 - val_precision: 0.4706 - val_recall: 0.7583
Epoch 2/50
12/12 [==============================] - 0s 11ms/step - loss: 0.7073 - accuracy: 0.5206 - precision: 0.4798 - recall: 0.5721 - val_loss: 0.71
80 - val_accuracy: 0.5022 - val_precision: 0.4669 - val_recall: 0.5355
Epoch 3/50
12/12 [==============================] - 0s 10ms/step - loss: 0.6927 - accuracy: 0.5376 - precision: 0.4952 - recall: 0.5550 - val_loss: 0.71
88 - val_accuracy: 0.5110 - val_precision: 0.4615 - val_recall: 0.3412
Epoch 4/50
12/12 [==============================] - 0s 10ms/step - loss: 0.6786 - accuracy: 0.5759 - precision: 0.5364 - recall: 0.5364 - val_loss: 0.72
30 - val_accuracy: 0.5175 - val_precision: 0.4698 - val_recall: 0.3318
Epoch 5/50
12/12 [==============================] - 0s 10ms/step - loss: 0.6719 - accuracy: 0.5780 - precision: 0.5387 - recall: 0.5395 - val_loss: 0.73
24 - val_accuracy: 0.5110 - val_precision: 0.4565 - val_recall: 0.2986
Epoch 6/50
12/12 [==============================] - 0s 10ms/step - loss: 0.6645 - accuracy: 0.6021 - precision: 0.5665 - recall: 0.5550 - val_loss: 0.74
62 - val_accuracy: 0.5066 - val_precision: 0.4624 - val_recall: 0.4076
Epoch 7/50
12/12 [==============================] - 0s 10ms/step - loss: 0.6578 - accuracy: 0.6057 - precision: 0.5622 - recall: 0.6233 - val_loss: 0.74
73 - val_accuracy: 0.5154 - val_precision: 0.4702 - val_recall: 0.3744
Epoch 8/50
12/12 [==============================] - 0s 11ms/step - loss: 0.6484 - accuracy: 0.6227 - precision: 0.5890 - recall: 0.5798 - val_loss: 0.75
32 - val_accuracy: 0.5110 - val_precision: 0.4709 - val_recall: 0.4597
Epoch 9/50
12/12 [==============================] - 0s 11ms/step - loss: 0.6379 - accuracy: 0.6291 - precision: 0.5889 - recall: 0.6264 - val_loss: 0.76
73 - val_accuracy: 0.4956 - val_precision: 0.4574 - val_recall: 0.4834
Epoch 10/50
12/12 [==============================] - 0s 10ms/step - loss: 0.6280 - accuracy: 0.6284 - precision: 0.5839 - recall: 0.6527 - val_loss: 0.77
13 - val_accuracy: 0.5285 - val_precision: 0.4907 - val_recall: 0.4976
Epoch 11/50
12/12 [==============================] - 0s 11ms/step - loss: 0.6340 - accuracy: 0.6326 - precision: 0.6036 - recall: 0.5736 - val_loss: 0.79
38 - val_accuracy: 0.4912 - val_precision: 0.4644 - val_recall: 0.6493
Epoch 12/50
 7/12 [================>.............] - ETA: 0s - loss: 0.5889 - accuracy: 0.6741 - precision: 0.6130 - recall: 0.7666Restoring model weight
s from the end of the best epoch: 2.
12/12 [==============================] - 0s 11ms/step - loss: 0.6052 - accuracy: 0.6603 - precision: 0.6092 - recall: 0.7178 - val_loss: 0.79
11 - val_accuracy: 0.5175 - val_precision: 0.4815 - val_recall: 0.5545
Epoch 12: early stopping
```

	Dataset	Model_Architecture	Loss	Accuracy	Precision	Recall	Validation_Loss	Validation_Accuracy	Validation_Precision	Validation_Recall
0	High-corr filtered	LSTM_model_arch_1	0.605165	0.660284	0.609211	0.717829	0.791142	0.517544	0.481481	0.554502

The training records show that the baseline model at the 7th epoch has a validation loss of 0.7911, and validation accuracy of 0.5175.

Next, we carryout extensive model architecture testing. Each candidate model architecture listed in the table above is tested for the corresponding feature sets with epoch of 50 and patience of 10. We present the results of initial runs in summary data frames and for each dataset below. Three top-performing architectures are shortlisted for further hyperparameter tuning.

When shortlisting top candidates, several factors were considered. Priority is given to models with smaller (or similar level of) validation loss benchmarked against the baseline model. Secondary priority is given to models where the validation metrics (loss, accuracy, precision and recall) do not have any significantly out-of-range figures, i.e., values significantly below 0.5. We recognise that the initial runs are based on untuned hyperparameters hence the results are only indicative of model architecture performance. Therefore, we choose three candidate models for each dataset for further refinement. The shortlisted architectures have been marked in the summary table above.

Model architecture initial run for `set_xg`, in ascending order of validation loss.

Dataset	Model_Architecture	Loss	Accuracy	Precision	Recall	Validation_Loss	Validation_Accuracy	Validation_Precision	Validation_Recall
XGBoost	LSTM_model_arch_4	0.658053	0.613475	0.580906	0.556589	0.718918	0.506579	0.465686	0.450237
XGBoost	LSTM_model_arch_7	0.666759	0.606383	0.576792	0.524031	0.733486	0.497807	0.471519	0.706161
XGBoost	LSTM_model_arch_5	0.719356	0.579433	0.589655	0.265116	0.748150	0.491228	0.434783	0.331754
XGBoost	LSTM_model_arch_2	0.673665	0.583688	0.542398	0.575194	0.756880	0.502193	0.472222	0.644550
XGBoost	LSTM_model_arch_9	0.662511	0.602837	0.565891	0.565891	0.767336	0.467105	0.459596	0.862559
XGBoost	LSTM_model_arch_8	0.647282	0.621277	0.583710	0.600000	0.786677	0.473684	0.461126	0.815166
XGBoost	LSTM_model_arch_10	0.677223	0.556028	0.513828	0.547287	0.813225	0.489035	0.465625	0.706161
XGBoost	LSTM_model_arch_6	0.659748	0.620567	0.591362	0.551938	0.830245	0.475877	0.455128	0.672986
XGBoost	LSTM_model_arch_1	0.634198	0.634752	0.589532	0.663566	1.022237	0.506579	0.468750	0.497630
XGBoost	LSTM_model_arch_3	0.412617	0.788652	0.740638	0.827907	1.158542	0.480263	0.453901	0.606635

Model architecture initial run for `set_hc`, in ascending order of validation loss.

	Dataset	Model_Architecture	Loss	Accuracy	Precision	Recall	Validation_Loss	Validation_Accuracy	Validation_Precision	Validation_Recall
1	High-corr filtered	LSTM_model_arch_12	0.621878	0.643972	0.667447	0.441860	0.680536	0.574561	0.545946	0.478673
4	High-corr filtered	LSTM_model_arch_15	0.695336	0.542553	0.000000	0.000000	0.689996	0.537281	0.000000	0.000000
13	High-corr filtered	LSTM_model_arch_24	0.697391	0.545390	0.520000	0.080620	0.690161	0.539474	1.000000	0.004739
8	High-corr filtered	LSTM_model_arch_19	0.702665	0.536170	0.480519	0.172093	0.690207	0.535088	0.400000	0.009479
12	High-corr filtered	LSTM_model_arch_23	0.697571	0.532624	0.464646	0.142636	0.691109	0.528509	0.166667	0.004739
6	High-corr filtered	LSTM_model_arch_17	0.696753	0.539007	0.333333	0.007752	0.691212	0.537281	0.000000	0.000000
5	High-corr filtered	LSTM_model_arch_16	0.679260	0.578723	0.545617	0.472868	0.691410	0.567982	0.570000	0.270142
15	High-corr filtered	LSTM_model_arch_26	0.697277	0.523404	0.451957	0.196899	0.691518	0.530702	0.200000	0.004739
16	High-corr filtered	LSTM_model_arch_27	0.694365	0.548936	0.534351	0.108527	0.693570	0.530702	0.466667	0.099526
0	High-corr filtered	LSTM_model_arch_11	0.685495	0.566667	0.527778	0.500775	0.695222	0.504386	0.461929	0.431280
14	High-corr filtered	LSTM_model_arch_25	0.696029	0.543972	0.527778	0.029457	0.695324	0.500000	0.422018	0.218009
10	High-corr filtered	LSTM_model_arch_21	0.694377	0.508511	0.455882	0.384496	0.695336	0.484649	0.434783	0.379147
2	High-corr filtered	LSTM_model_arch_13	0.685344	0.567376	0.597765	0.165891	0.705010	0.543860	0.527273	0.137441
7	High-corr filtered	LSTM_model_arch_18	0.694631	0.536170	0.493506	0.530233	0.712441	0.497807	0.452632	0.407583
9	High-corr filtered	LSTM_model_arch_20	0.696449	0.572340	0.552239	0.344186	0.717235	0.519737	0.463636	0.241706
11	High-corr filtered	LSTM_model_arch_22	0.691395	0.548936	0.508704	0.407752	0.746992	0.482456	0.459283	0.668246
3	High-corr filtered	LSTM_model_arch_14	0.615444	0.653901	0.689157	0.443411	0.773789	0.543860	0.505415	0.663507

Model architecture initial run for `set_ks`, in ascending order of validation loss.

	Dataset	Model_Architecture	Loss	Accuracy	Precision	Recall	Validation_Loss	Validation_Accuracy	Validation_Precision	Validation_Recall
23	SOM K-means dimentionally reduced	LSTM_model_arch_24	0.699207	0.536879	0.445946	0.051163	0.692527	0.519737	0.466102	0.260664
20	SOM K-means dimentionally reduced	LSTM_model_arch_21	0.697844	0.530496	0.478372	0.291473	0.694225	0.513158	0.465839	0.355450
25	SOM K-means dimentionally reduced	LSTM_model_arch_26	0.697293	0.501418	0.444231	0.358140	0.694361	0.500000	0.469091	0.611374
22	SOM K-means dimentionally reduced	LSTM_model_arch_23	0.716011	0.553901	0.527778	0.235659	0.694394	0.519737	0.484252	0.582938
26	SOM K-means dimentionally reduced	LSTM_model_arch_27	0.712774	0.497872	0.393939	0.181395	0.695358	0.563596	0.581081	0.203791
19	SOM K-means dimentionally reduced	LSTM_model_arch_20	0.687325	0.549645	0.506887	0.570543	0.697345	0.524123	0.487179	0.540284
16	SOM K-means dimentionally reduced	LSTM_model_arch_17	0.699650	0.509929	0.471178	0.582946	0.699502	0.495614	0.425197	0.255924
21	SOM K-means dimentionally reduced	LSTM_model_arch_22	0.698798	0.545390	0.505376	0.291473	0.702117	0.489035	0.454545	0.521327
4	SOM K-means dimentionally reduced	LSTM_model_arch_5	0.667673	0.591489	0.561279	0.489922	0.710798	0.519737	0.468750	0.284360
18	SOM K-means dimentionally reduced	LSTM_model_arch_19	0.675467	0.565957	0.517205	0.768992	0.711360	0.508772	0.479624	0.725118
12	SOM K-means dimentionally reduced	LSTM_model_arch_13	0.671277	0.590780	0.576923	0.395349	0.713394	0.532895	0.492857	0.327014
10	SOM K-means dimentionally reduced	LSTM_model_arch_11	0.683867	0.563830	0.568182	0.193798	0.714000	0.559211	0.546296	0.279621
11	SOM K-means dimentionally reduced	LSTM_model_arch_12	0.669606	0.604965	0.579422	0.497674	0.732153	0.521930	0.484018	0.502370
24	SOM K-means dimentionally reduced	LSTM_model_arch_25	0.692867	0.575177	0.566474	0.303876	0.737006	0.495614	0.445714	0.369668
17	SOM K-means dimentionally reduced	LSTM_model_arch_18	0.676085	0.563121	0.519104	0.610853	0.744365	0.464912	0.447619	0.668246
5	SOM K-means dimentionally reduced	LSTM_model_arch_6	0.660812	0.604255	0.563877	0.595349	0.746138	0.504386	0.475884	0.701422
8	SOM K-means dimentionally reduced	LSTM_model_arch_9	0.671927	0.570213	0.522491	0.702326	0.748040	0.480263	0.450000	0.554502
1	SOM K-means dimentionally reduced	LSTM_model_arch_2	0.652791	0.622695	0.603670	0.510078	0.755914	0.486842	0.449782	0.488152
9	SOM K-means dimentionally reduced	LSTM_model_arch_10	0.653206	0.613475	0.569252	0.637209	0.773238	0.504386	0.475083	0.677725
0	SOM K-means dimentionally reduced	LSTM_model_arch_1	0.582817	0.667376	0.620879	0.700775	0.782809	0.526316	0.487805	0.473934
3	SOM K-means dimentionally reduced	LSTM_model_arch_4	0.628550	0.622695	0.566237	0.748837	0.802761	0.473684	0.454829	0.691943
6	SOM K-means dimentionally reduced	LSTM_model_arch_7	0.668463	0.595035	0.554094	0.587597	0.813724	0.469298	0.449180	0.649289
14	SOM K-means dimentionally reduced	LSTM_model_arch_15	0.570491	0.695745	0.711765	0.562791	0.922974	0.502193	0.467480	0.545024
13	SOM K-means dimentionally reduced	LSTM_model_arch_14	0.562447	0.704965	0.665224	0.714729	1.084482	0.495614	0.463602	0.573460
2	SOM K-means dimentionally reduced	LSTM_model_arch_3	0.236687	0.898582	0.900958	0.874419	1.848139	0.489035	0.453390	0.507109

Model development: Hyperparameter tuning.

With 9 shortlisted model architectures (3 for each feature set), we proceed to hyperparameter tuning for each candidate model. The below table lists all architectures and their respective input datasets examined in the hyperparameter tuning stage.

Architecture ID	Architecture	Feature set
2	0 pre-LSTM dense layer 1 LSTM layer with dropout 1 pre-output dense layer	XG
4	0 pre-LSTM dense layer 2 LSTM layer with dropout in between 0 pre-output dense layer	XG
7	0 pre-LSTM dense layer 3 LSTM layer with dropout in between 0 pre-output dense layer	XG
12	2 pre-LSTM dense layer 1 LSTM layer with dropout 1 pre-output dense layer	HC
14	2 pre-LSTM dense layer 1 LSTM layer with dropout 2 pre-output dense layer	HC
21	3 pre-LSTM dense layer 2 LSTM layer with dropout in between 2 pre-output dense layer	HC
12	2 pre-LSTM dense layer 1 LSTM layer with dropout 1 pre-output dense layer	KS
23	3 pre-LSTM dense layer 3 LSTM layer with dropout in between 0 pre-output dense layer	KS
26	2 pre-LSTM dense layer 2 LSTM layer with dropout in between 2 pre-output dense layer	KS

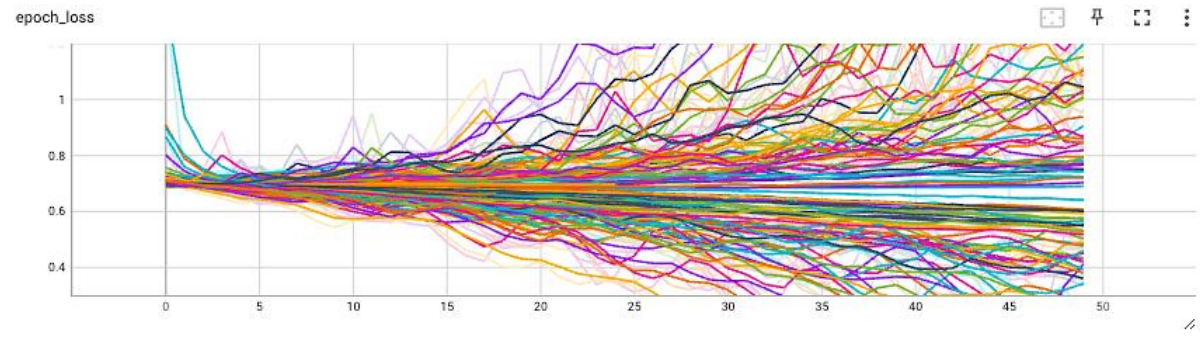
For each shortlisted model, we perform the Hyper Model parameter search using the Bayesian Optimisation Tuner, with 50 trials, 50 epochs and a patience of 10, the same class weights are also applied.

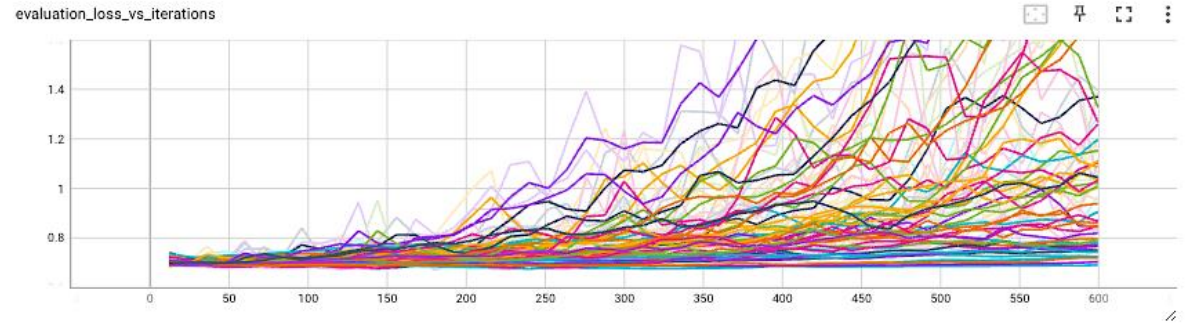
Below are plots of epoch loss/accuracy (for training and validation) and validation loss/accuracy extracted from TensorBoard to visualise the tuning process for each model.

Features set HC:

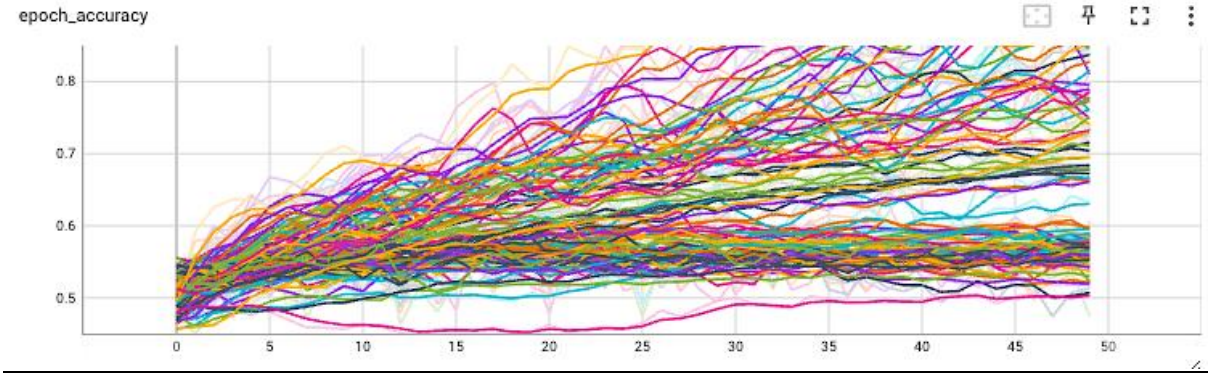
Model architecture 12:

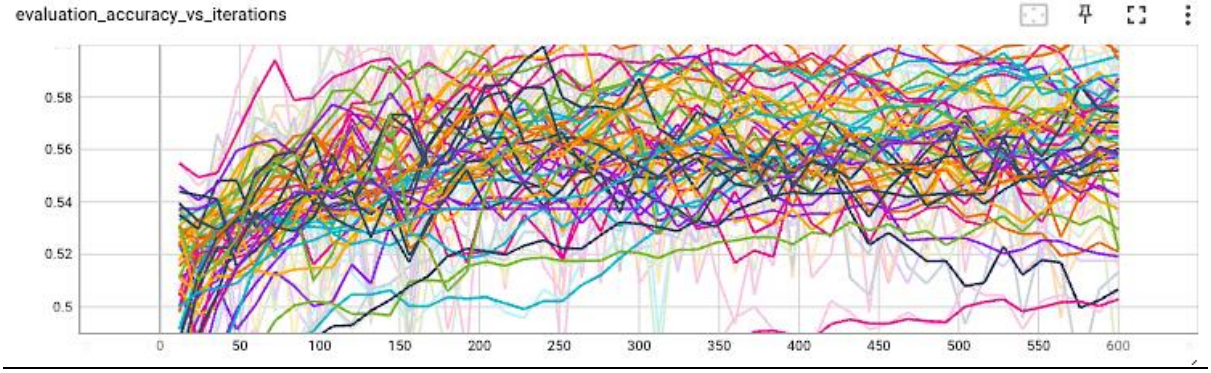
Loss:


epoch_loss
1
0.8
0.6
0.4
0
5
10
15
20
25
30
35
40
45
50


evaluation_loss_vs_iterations
1.4
1.2
1
0.8
0
50
100
150
200
250
300
350
400
450
500
550
600

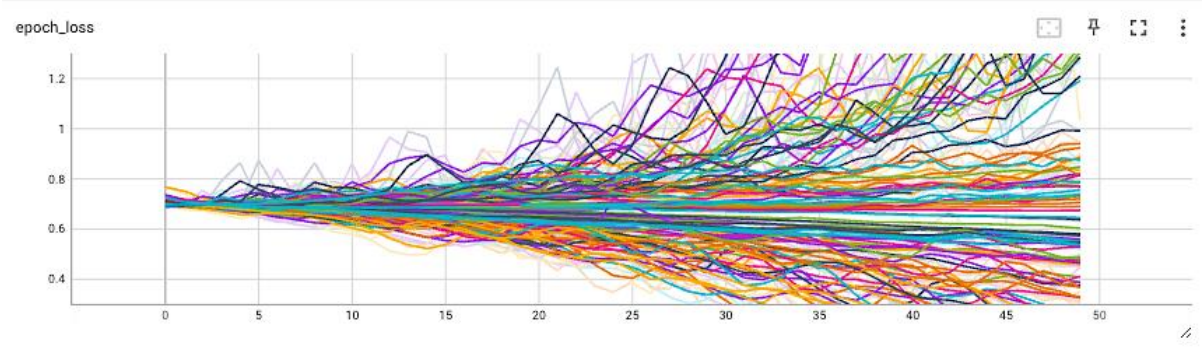
Accuracy:

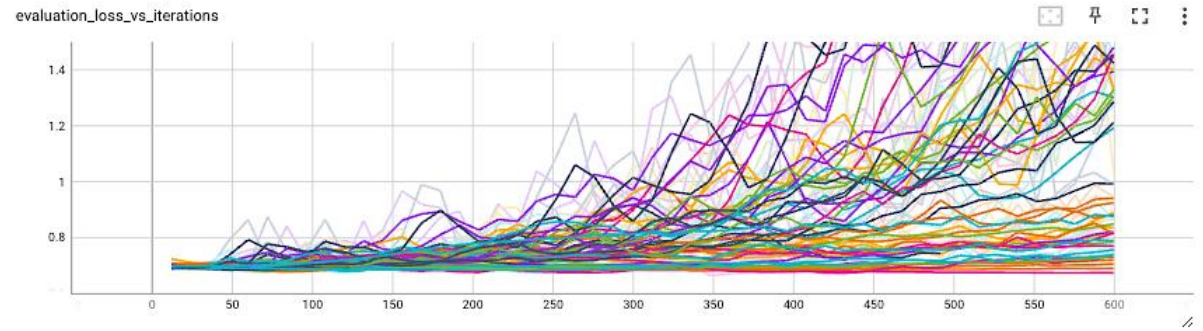

epoch_accuracy
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30
35
40
45
50


evaluation_accuracy_vs_iterations
0.58
0.56
0.54
0.52
0.5
0
50
100
150
200
250
300
350
400
450
500
550
600

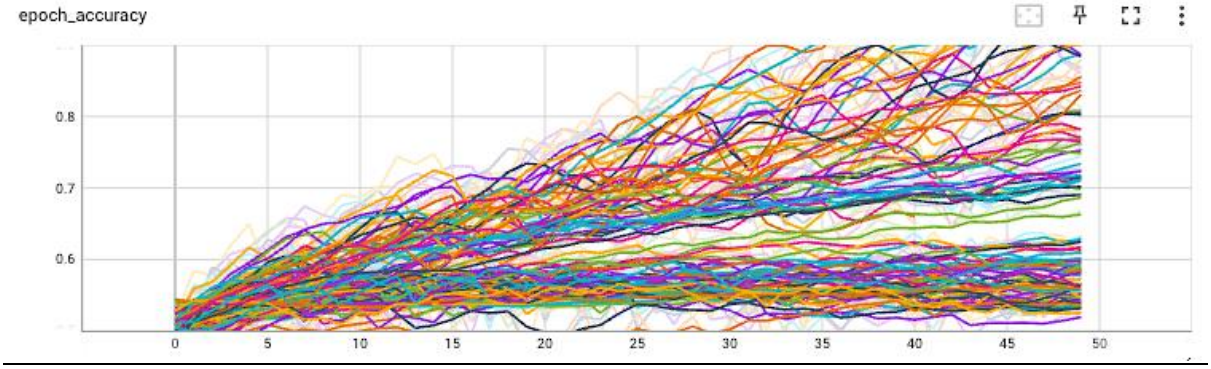
Model architecture 14:

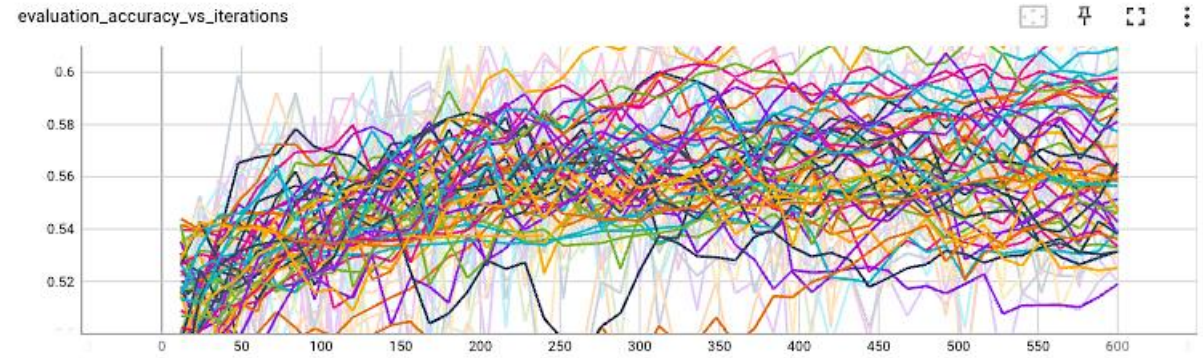
Loss:





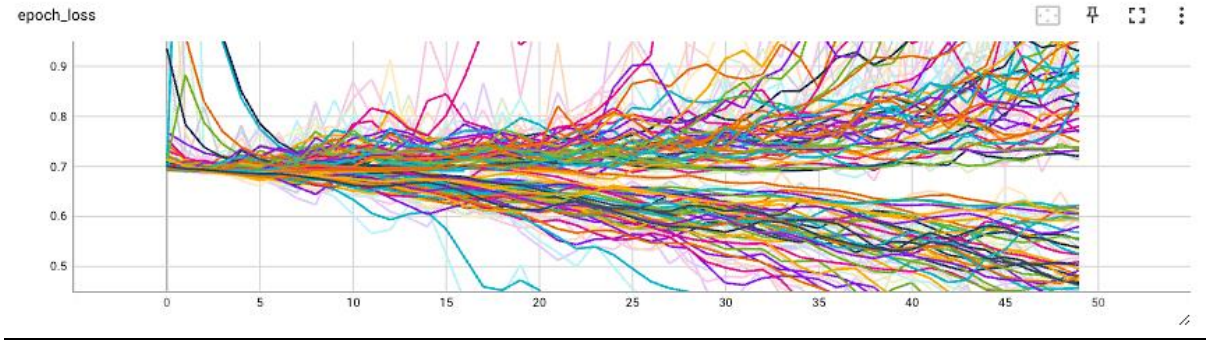
Accuracy:

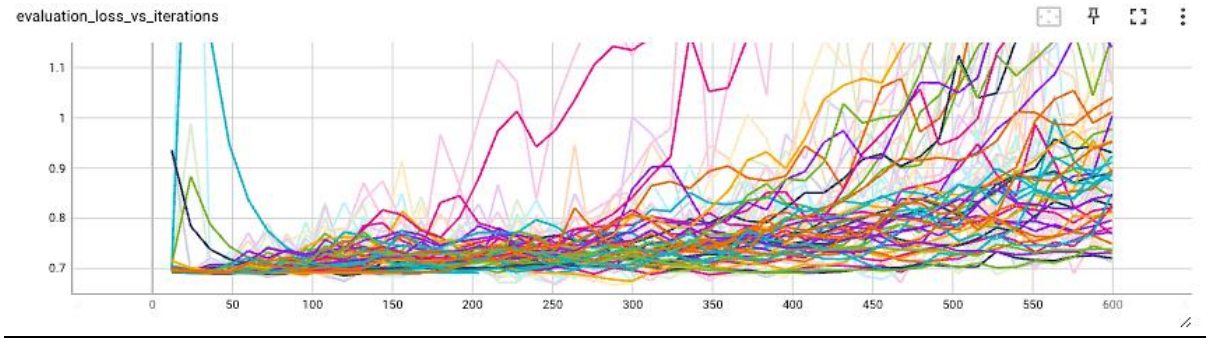




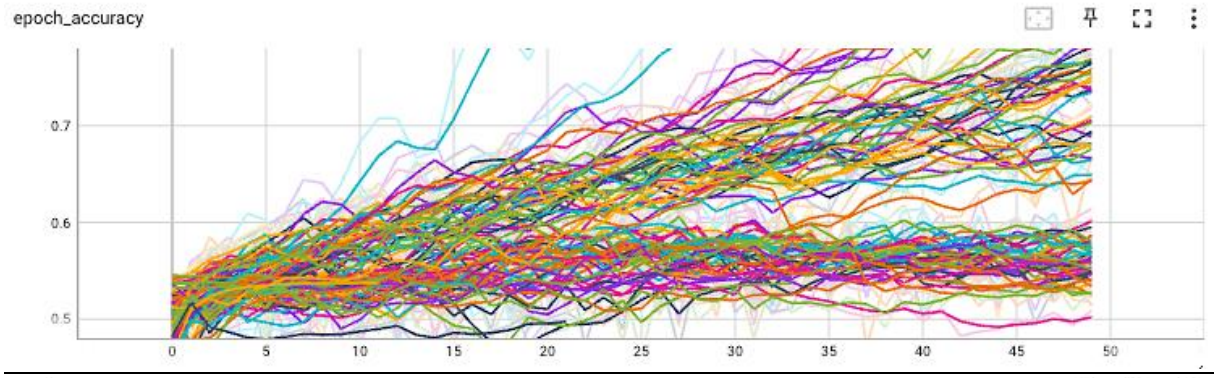
Model architecture 21:

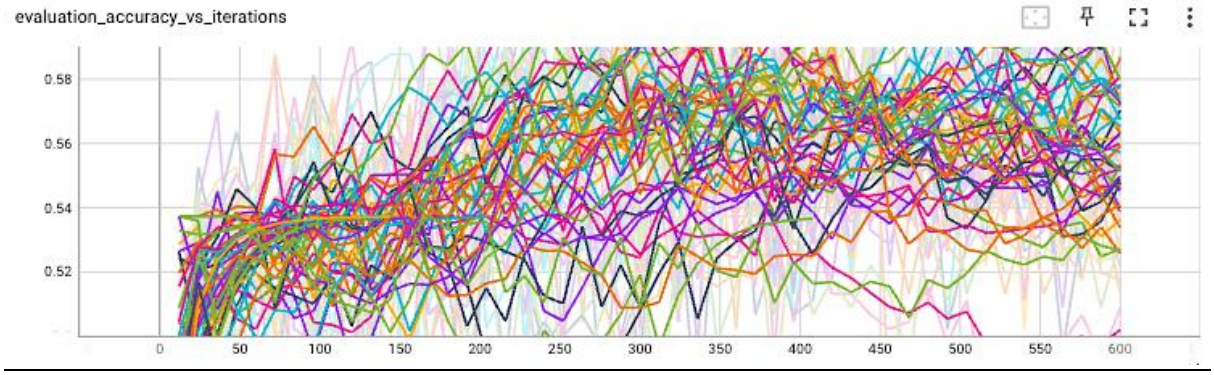
Loss:





Accuracy:



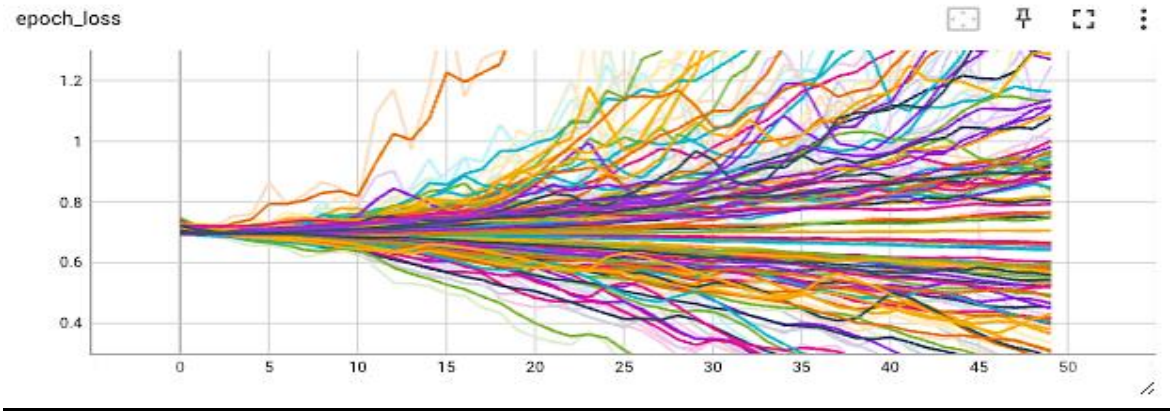


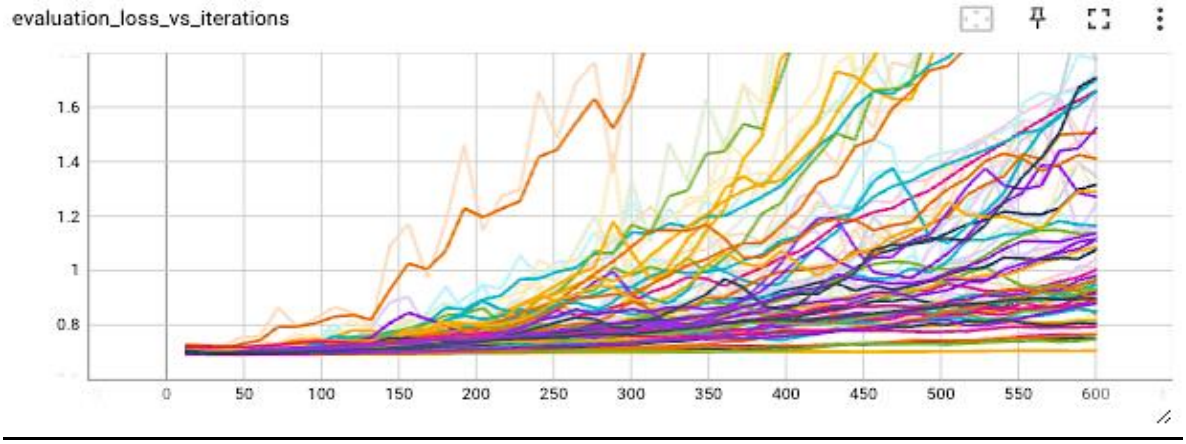
From the plots we can see that some models are clearly displaying trends of overfitting, as the high-level trend shows that evaluation loss display varying rates of increase while training loss generally decreases. It is also interesting that the minimum epoch loss seems to be around 0.7. This trend is also applicable to many models in the KS and XG sets.

Features set KS:

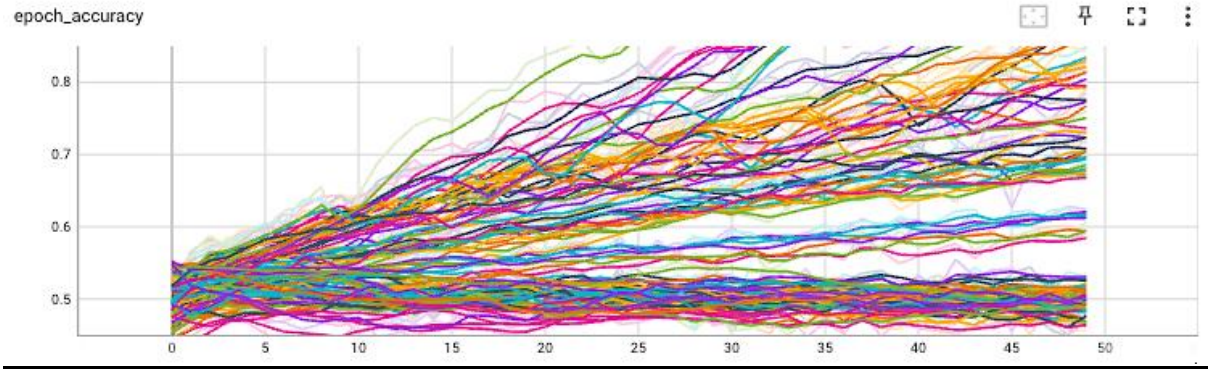
Model architecture 12:

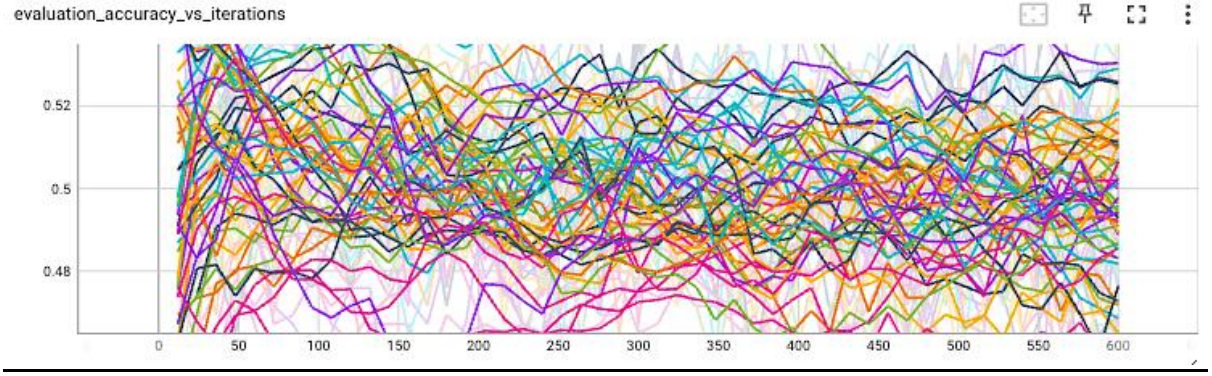
Loss:





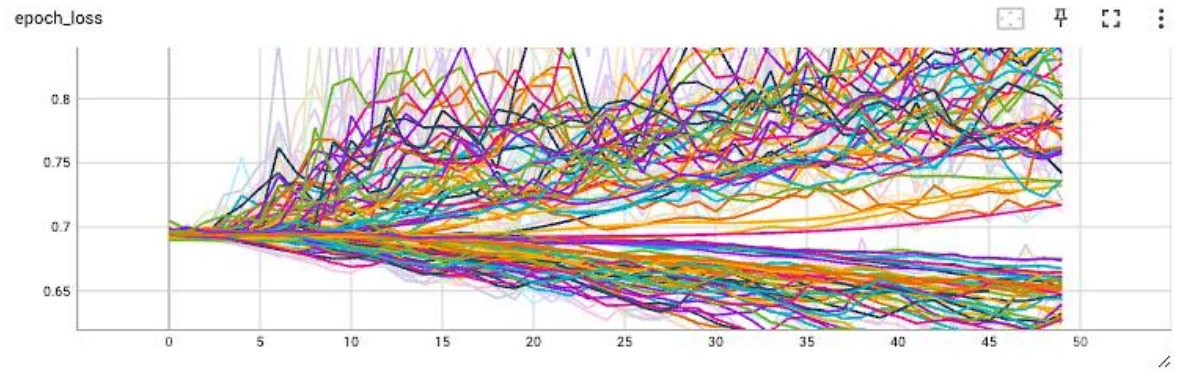
Accuracy:

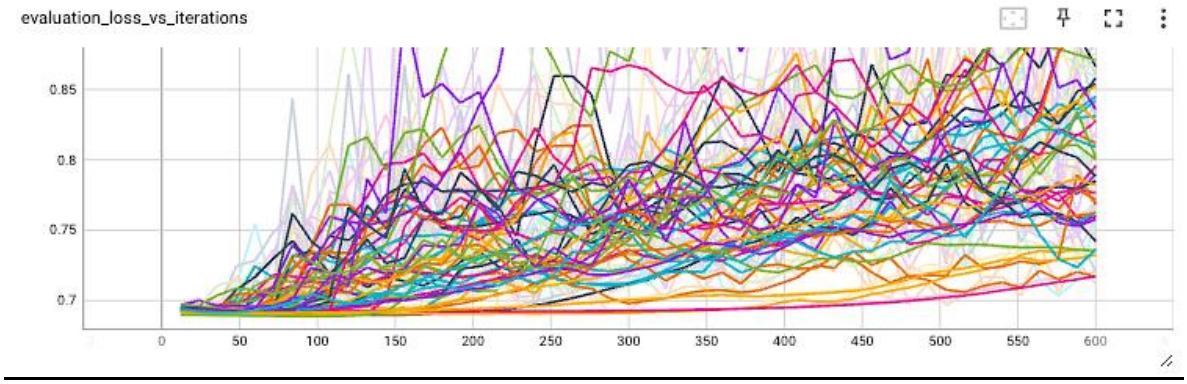




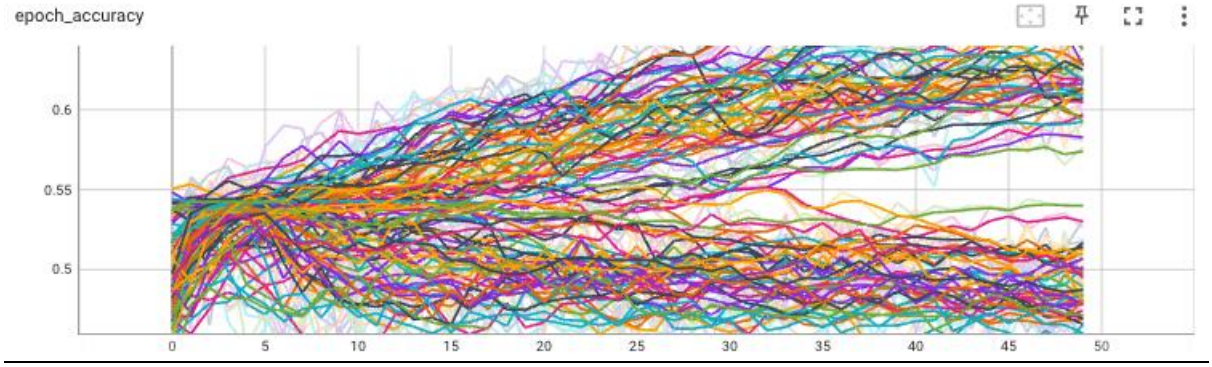
Model architecture 23:

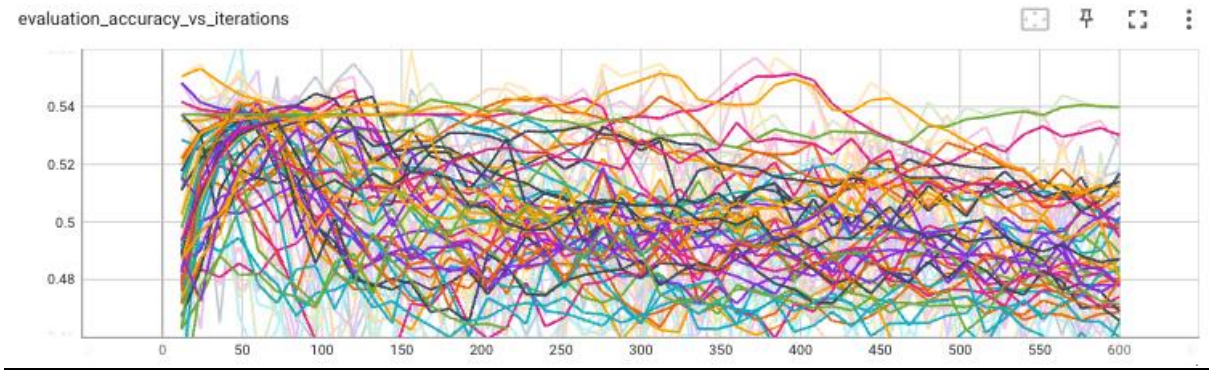
Loss:





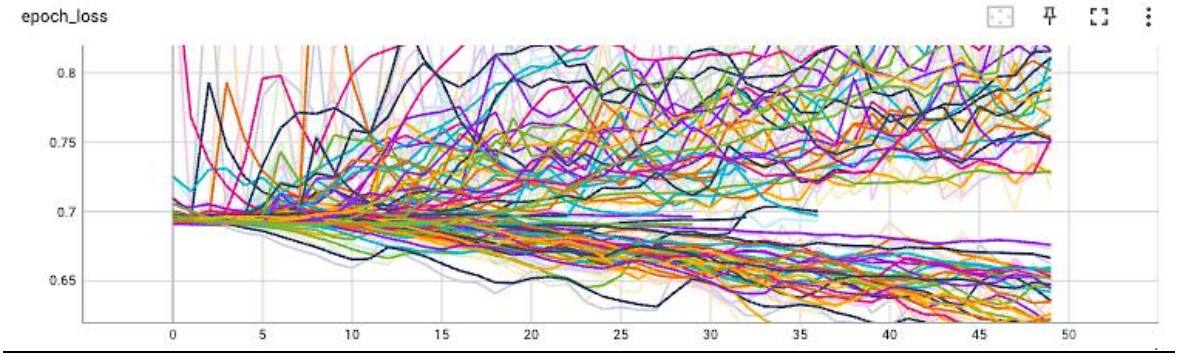
Accuracy:

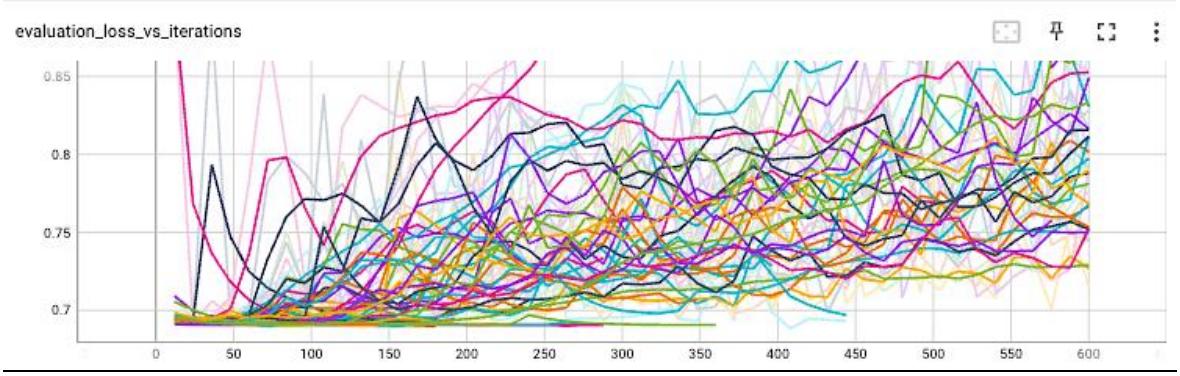




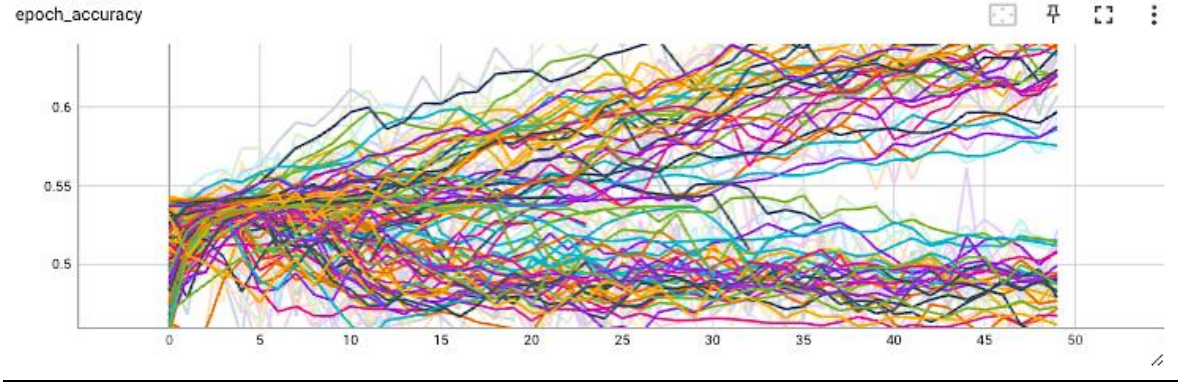
Model architecture 26:

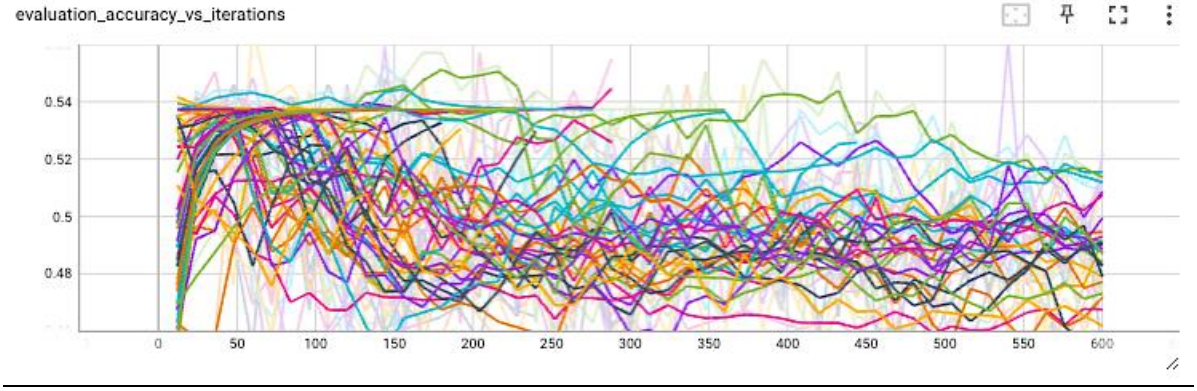
Loss:





Accuracy:



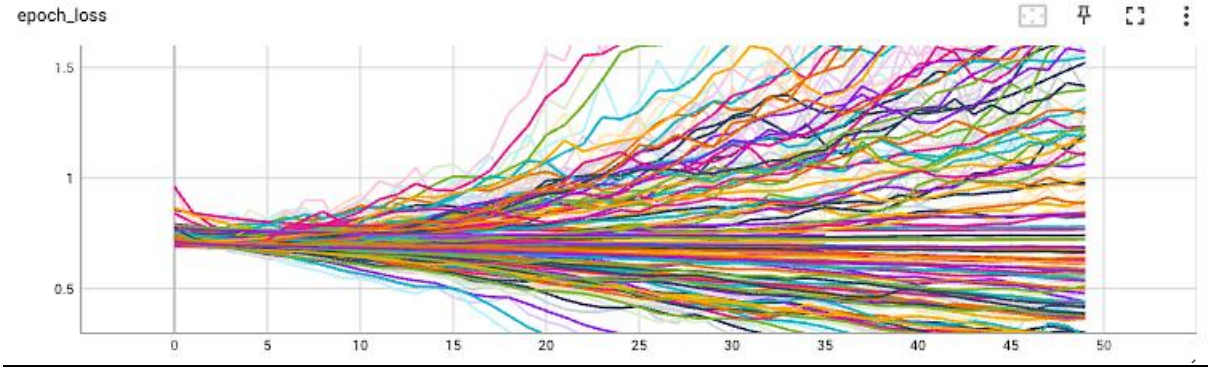


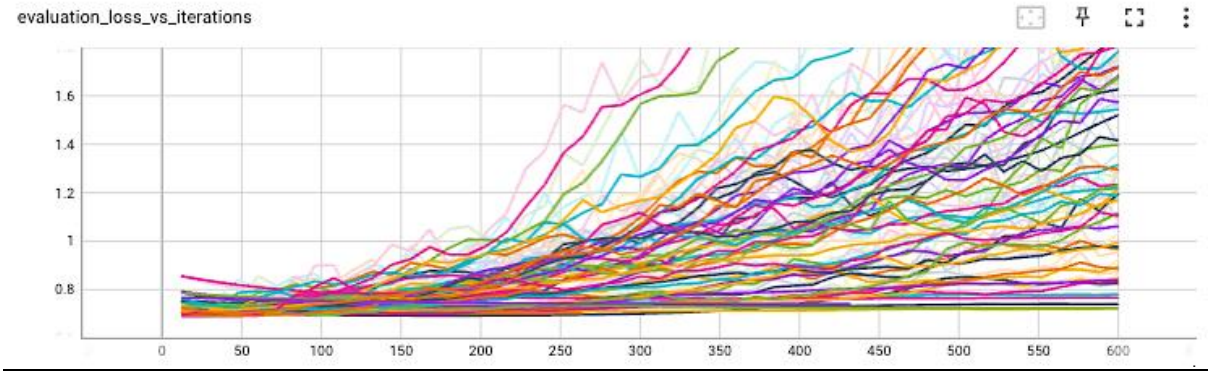
For model architectures 23 and 26, the evaluation loss by iterations displays a much more unstable behaviour, where not only does it increase very early on in the iterations, but also has turbulent movements compared to other architectures.

Features set XG:

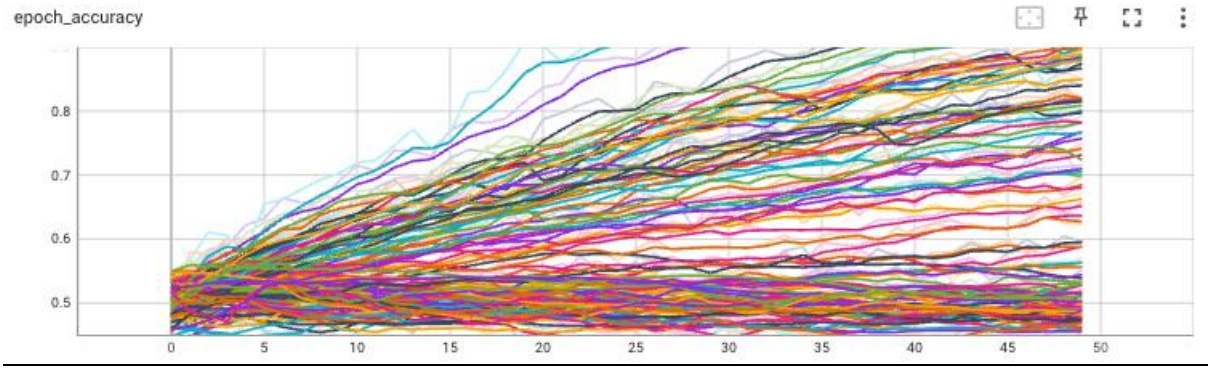
Model architecture 2:

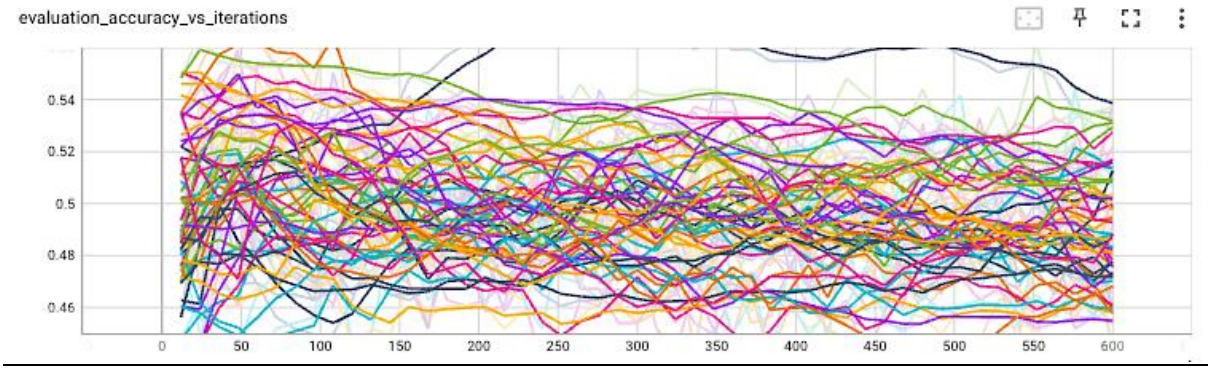
Loss:





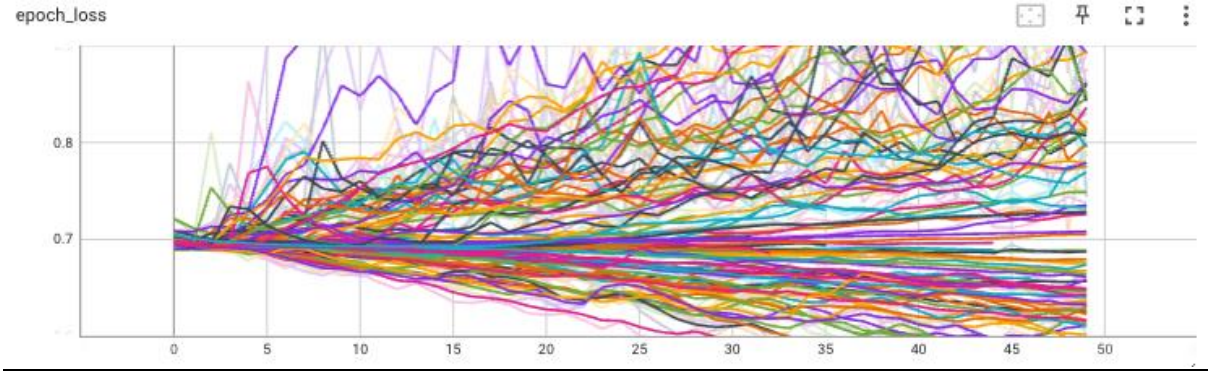
Accuracy:

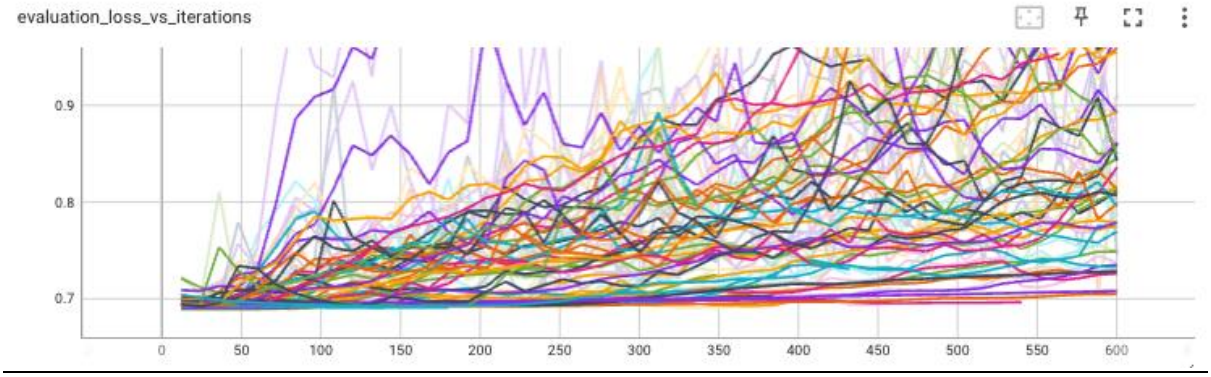




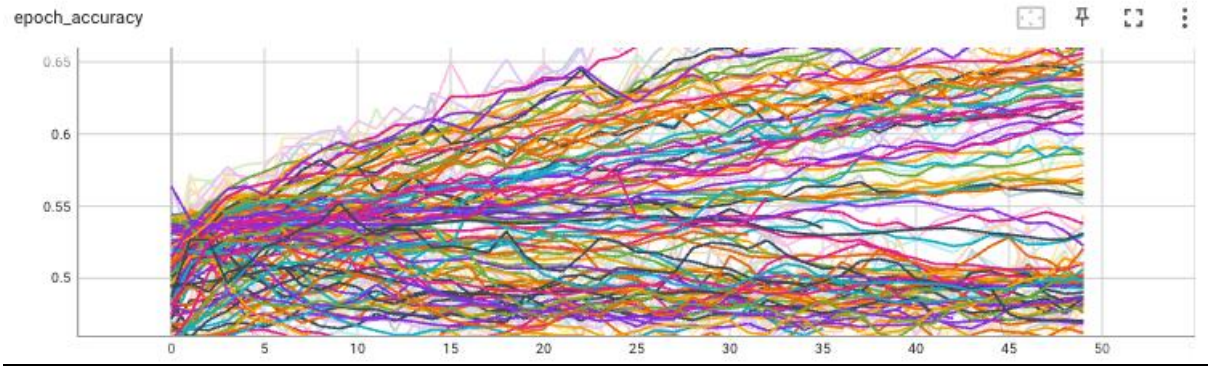
Model architecture 4:

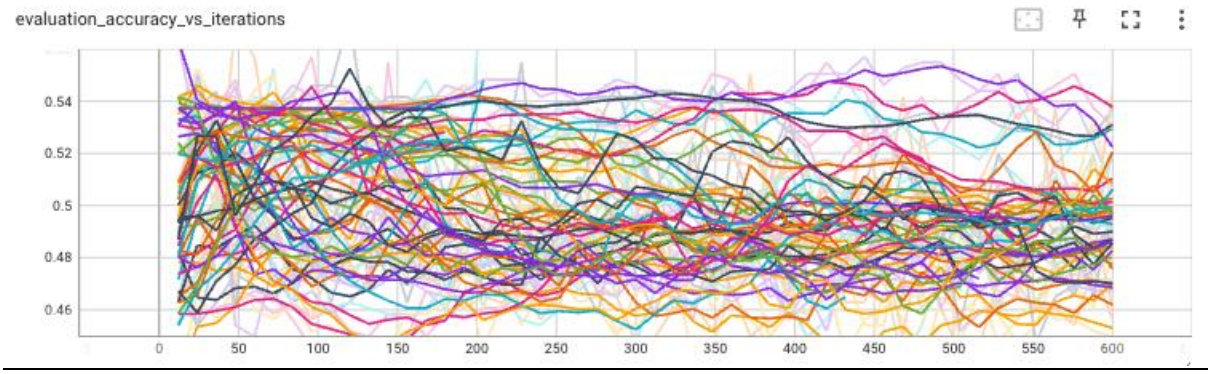
Loss:





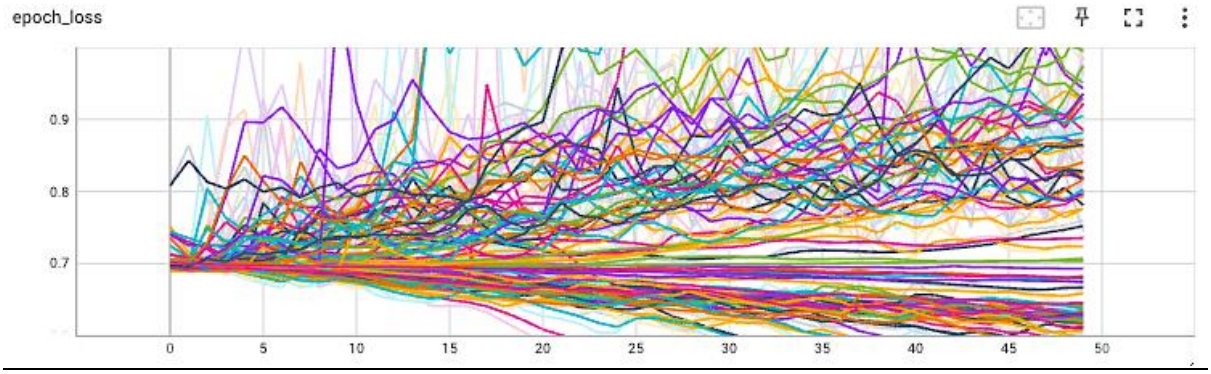
Accuracy:

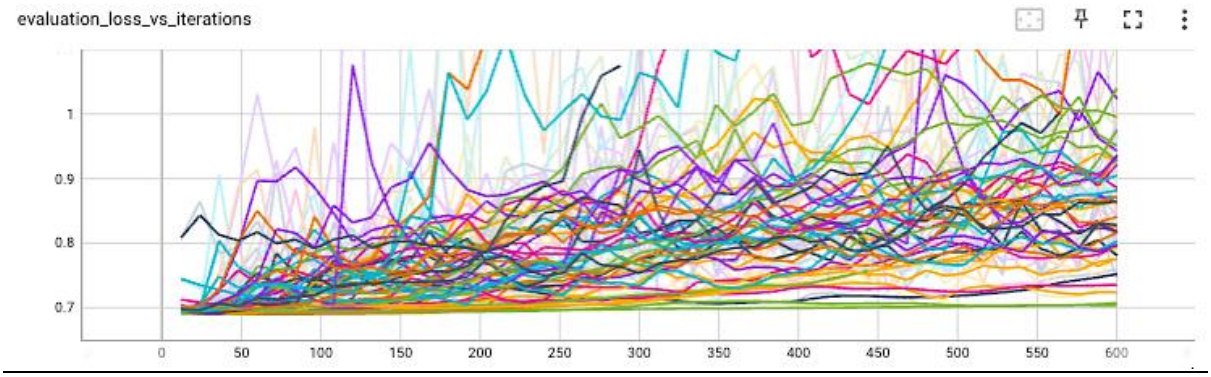




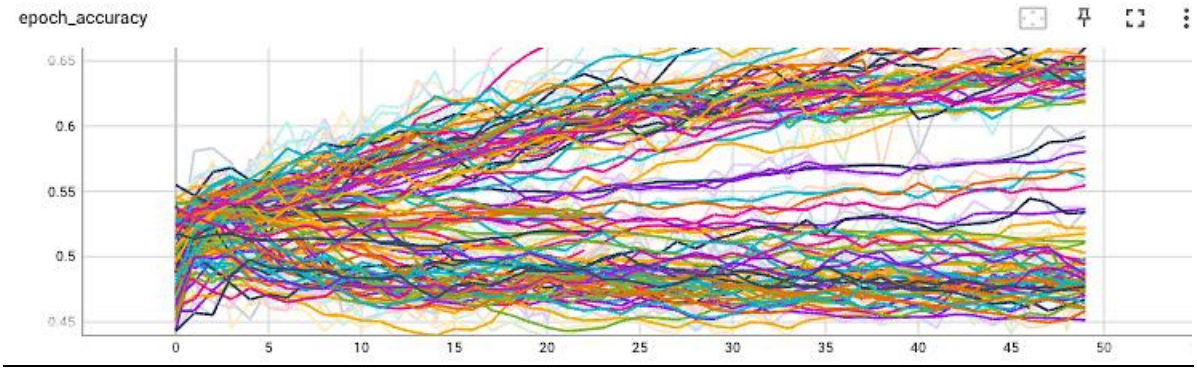
Model architecture 7:

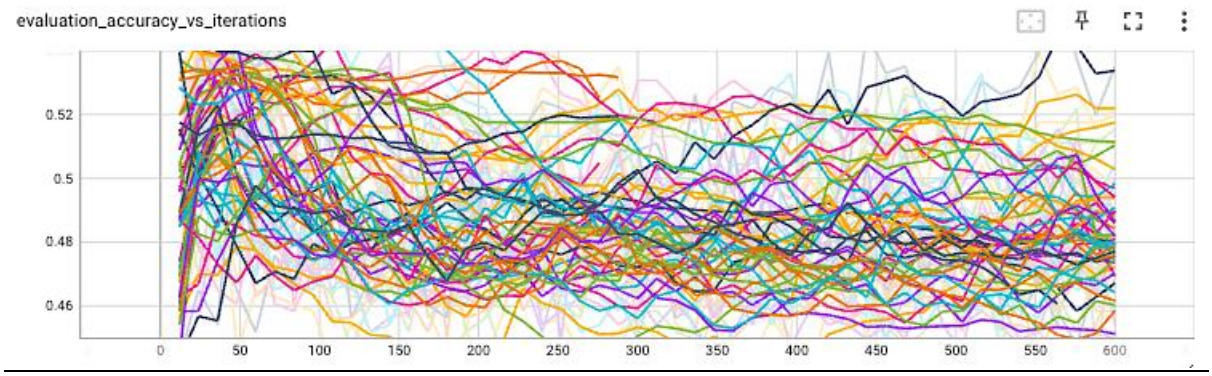
Loss:





Accuracy:



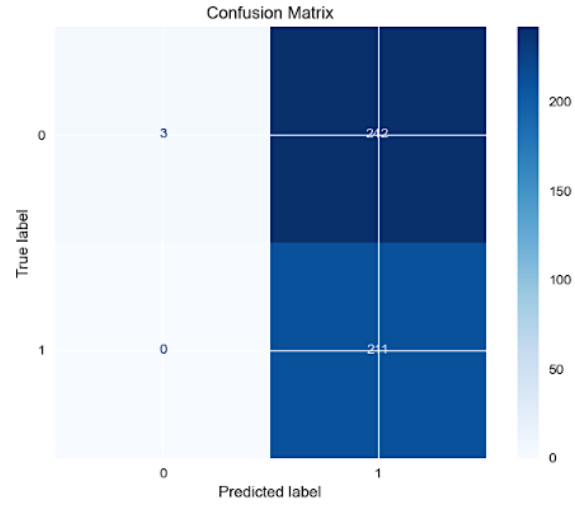

evaluation_accuracy_vs_iterations
0.52
0.5
0.48
0.46
0
50
100
150
200
250
300
350
400
450
500
550
600

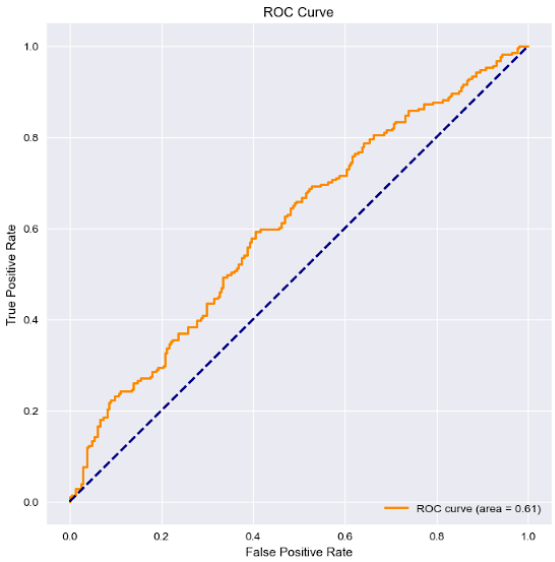
Evaluating model performance.

To compare the performance of winner models for each combination, we load the best models that were saved in the hyperparameter tuning process and plot their confusion matrix, ROC curve and classification report on the development sets. All models in the XG and KS category perform poorly with the area under the ROC curve in the range 0.49 – 0.52, which makes the model on par with a random coin toss. Majority of models in on the XG and KS feature sets are also weak at identifying negative classes, producing a high volume of false positives (a general problem for all models here indeed). However, judging by the ROC curve all three models in the HC category displays superior performance in contrast to the rest. The results are shown here along with the model's hyperparameters and architecture.

Features set HC:

Model architecture 12:





Model classification Report:

	precision	recall	f1-score	support
0	1.00	0.01	0.02	245
1	0.47	1.00	0.64	211
accuracy			0.47	456
macro avg	0.73	0.51	0.33	456
weighted avg	0.75	0.47	0.31	456

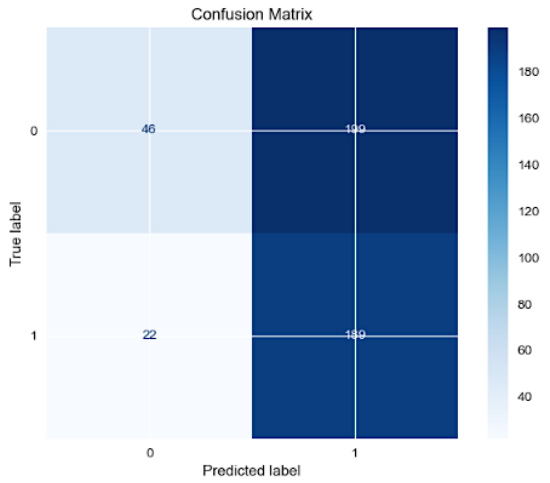
Model parameters and architecture:

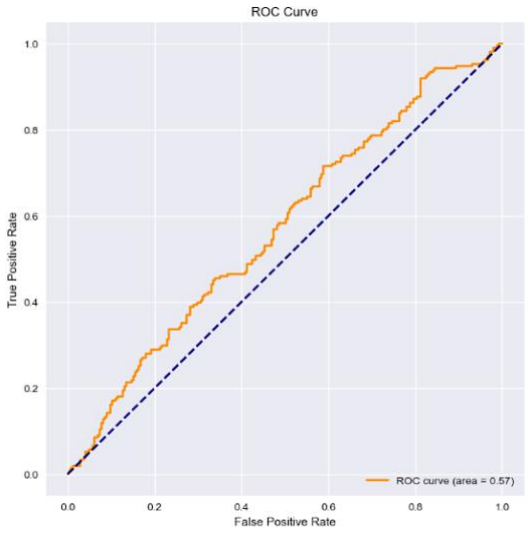
```
Trial 42 summary
Hyperparameters:
Dense_start_units1: 107
Dense_start_units2: 84
Dense_end_units1: 12
LSTM_units_1: 10
Dropout_rate: 0.1
learning_rate: 0.002635725165470654
LSTM_activation1: relu
Dense_start_activation1: elu
Dense_start_activation2: elu
Dense_end_activation1: elu
Score: 0.6699366569519043
```

Layer (type)	Output Shape	Param #
Dense_start1 (Dense)	(None, None, 115)	23000
Dense_start2 (Dense)	(None, None, 68)	7888
LSTM1 (LSTM)	(None, 22)	8008
Dropout1 (Dropout)	(None, 22)	0
Dense_end1 (Dense)	(None, 18)	414
Dense_end2 (Dense)	(None, 8)	152
Output (Dense)	(None, 1)	9

Total params: 39471 (154.18 KB)
Trainable params: 39471 (154.18 KB)
Non-trainable params: 0 (0.00 Byte)

Model architecture 14:






```
Model classification Report:
              precision    recall  f1-score   support

           0       0.68      0.19      0.29       245
           1       0.49      0.90      0.63       211

    accuracy                           0.52       456
   macro avg       0.58      0.54      0.46       456
weighted avg       0.59      0.52      0.45       456
```

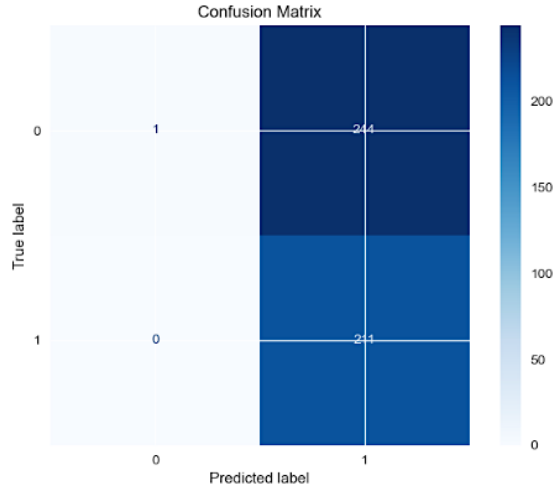
Model parameters and architecture:

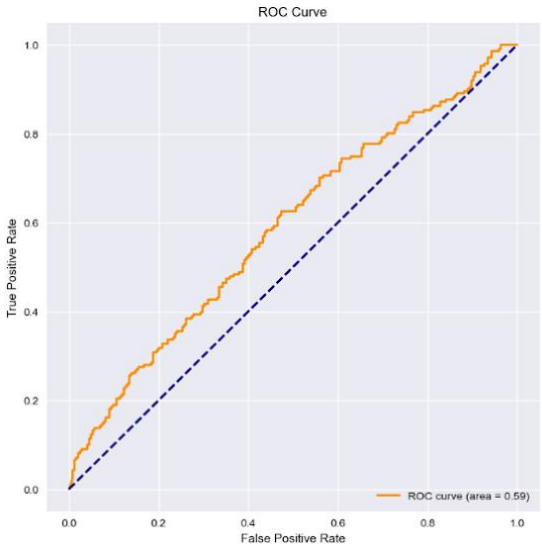
```
Trial 42 summary
Hyperparameters:
Dense_start_units1: 107
Dense_start_units2: 84
Dense_end_units1: 12
LSTM_units_1: 10
Dropout_rate: 0.1
learning_rate: 0.002635725165470654
LSTM_activation1: relu
Dense_start_activation1: elu
Dense_start_activation2: elu
Dense_end_activation1: elu
Score: 0.6699366569519043
```

Layer (type)	Output Shape	Param #
Dense_start1 (Dense)	(None, None, 107)	21400
Dense_start2 (Dense)	(None, None, 84)	9072
LSTM1 (LSTM)	(None, 10)	3800
Dropout1 (Dropout)	(None, 10)	0
Dense_end1 (Dense)	(None, 12)	132
Output (Dense)	(None, 1)	13

```
Total params: 34417 (134.44 KB)
Trainable params: 34417 (134.44 KB)
Non-trainable params: 0 (0.00 Byte)
```

Model architecture 21:





Model classification Report:

	precision	recall	f1-score	support
0	1.00	0.00	0.01	245
1	0.46	1.00	0.63	211
accuracy			0.46	456
macro avg	0.73	0.50	0.32	456
weighted avg	0.75	0.46	0.30	456

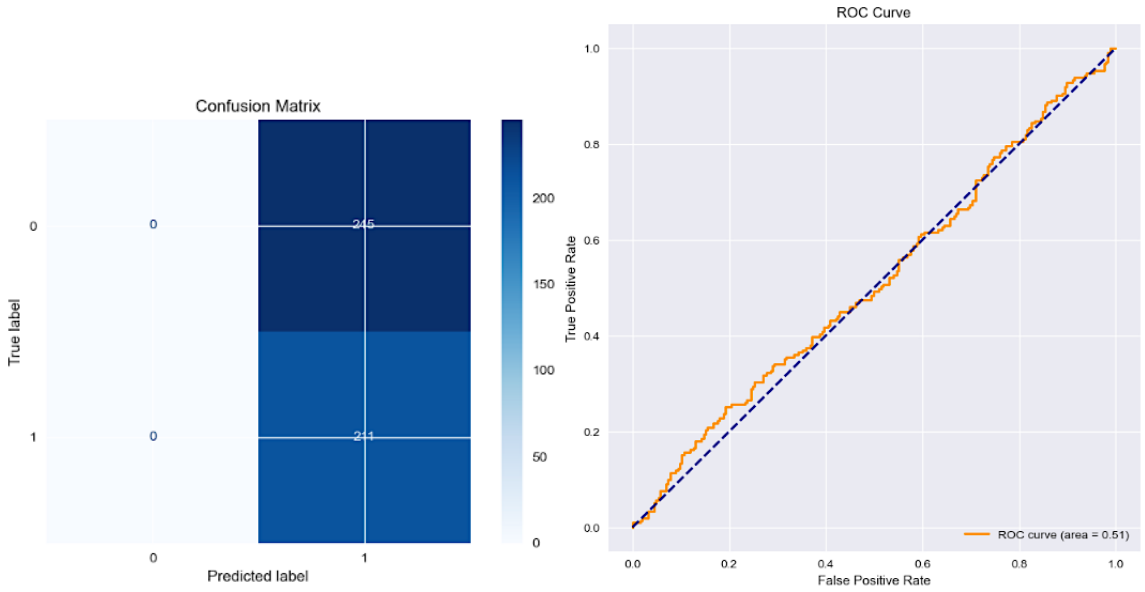
Model parameters and architecture:

```
Trial 11 summary
Hyperparameters:
Dense_start_units1: 195
Dense_start_units2: 81
Dense_end_units1: 26
Dense_end_units2: 13
LSTM_units1: 6
LSTM_units2: 14
Dropout_rate1: 0.2
learning_rate: 0.0016224619152272561
LSTM_activation1: relu
LSTM_activation2: elu
Dense_start_activation1: relu
Dense_start_activation2: elu
Dense_start_activation3: relu
Dense_end_activation1: relu
Dense_end_activation2: elu
Score: 0.6671620011329651
```

Layer (type)	Output Shape	Param #
Dense_start1 (Dense)	(None, None, 195)	39000
Dense_start2 (Dense)	(None, None, 81)	15876
Dense_start3 (Dense)	(None, None, 81)	6642
lstm1 (LSTM)	(None, None, 6)	2112
Dropout1 (Dropout)	(None, None, 6)	0
lstm2 (LSTM)	(None, 6)	312
Dense_end1 (Dense)	(None, 26)	182
Dense_end2 (Dense)	(None, 13)	351
Output (Dense)	(None, 1)	14

Total params: 64489 (251.91 KB)
Trainable params: 64489 (251.91 KB)
Non-trainable params: 0 (0.00 Byte)

We then compare with the **baseline model performance**, shown below.



Model classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	245
1	0.46	1.00	0.63	211
accuracy			0.46	456
macro avg	0.23	0.50	0.32	456
weighted avg	0.21	0.46	0.29	456

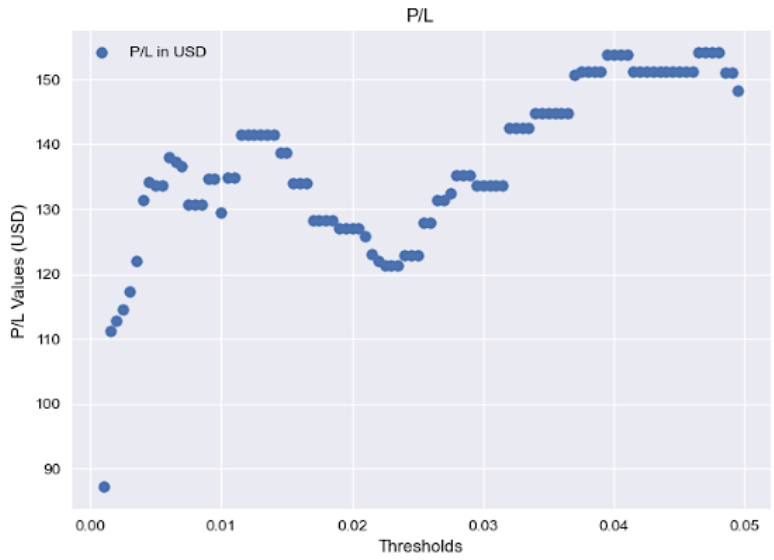
In all three cases of best models, the models seem to discern positive movements accurately, but are exceptionally bad at predicting negative movements. Judging from the outcomes, we choose architecture 14 as the best model. While 12 and 21 has the largest AUC of 0.61 and 0.59,

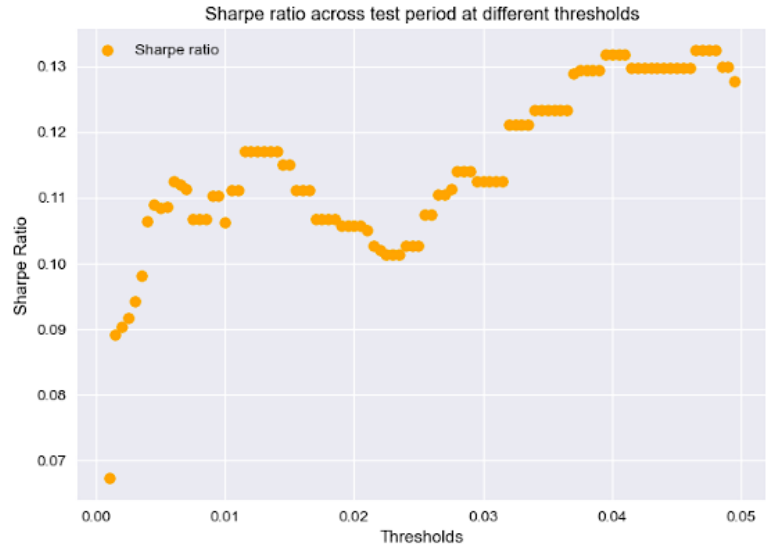
architecture 14 with AUC of 0.57 has nonetheless an edge in predicting non-upward movements, which aligns better with the target labels in the actual dataset. Meanwhile, we can also see that the baseline model is equally weak at making negative predictions with an AUC of 0.51. Comparing with industry standards, an AUC of 0.5-0.7 is considered poor discrimination.

Back testing: Trading strategy on test set at various thresholds.

For back testing, we use the HC test set, which the model has not yet seen in the development and validation process. The period covers dates span across 03-12-2021 to 30-10-2023. For each date, if the model predicts positive movements in the next day, we hold the stock, and if the model predicts non-positive movements, we sell the stock. This is a simplified trading strategy assuming no transaction costs.

Thresholds values from 0.001 to 0.05 were explored at steps of 0.0005. The cumulative P/L and Sharpe ratio are plotted below as scatter plots. We can see that overall; the strategy gives us profits of varying amounts at all thresholds, with a low of 87.17 USD when the threshold is set at 0.001. As all models seem to be producing false positives, it is not surprising that higher thresholds generally give higher profit.





Conclusion

This report has presented the processes and results of the following procedures: First, extensive feature engineering was accomplished via a combination of feature extension and external data sources. Second, exploratory data analysis was carried out comprehensively to understand the data types, value ranges and distributions of the dataset. Third, feature selection was performed sequentially by filtering high-correlations, dimensionality reduction using SOM and K-means clustering, and XGBoost with feature importance and Shap analysis, producing three candidate features sets. Fourth, an extensive range of model architectures were explored for all feature sets, resulting in top three shortlists. Fifth, hyperparameter tuning was conducted using the Bayesian Optimisation Tuner, based on which a winner model was selected after model performance evaluation. Lastly, a trading strategy was devised as a back test for applying the model on unseen data.

Overall, we make several observations / findings. First, the performance of candidate LSTM models seem to generally suffer from low recalls for the negative class and low precision for the positive class, suggesting that the model produces a large proportion of false positives with weak ability to make negative classifications. Second, the fact that models for the HC feature set (features after only removing high correlations) generally perform best suggests that dense layers positioned before LSTM layers can potentially be an effective method for feature extraction as opposed to conventional feature selection methods, which often tend to assume certain relationships when ranking feature importance. Third, the overall poor performance of all models trigger thought on several fronts. We identify below some possible reasons why this is the case.

- Generally, it is agreed that one cannot predict future financial outcomes with past data.
- The period selected for this study is highly volatile with various major social and political events impacting stock movements. The train-dev-test split has hence encapsulated periods where the underlying market conditions and value ranges are highly different.
- Due the absence of professional subscriptions, the data suffers from the absence of various factors that are important to shaping financial market movements, for example sentiment data and financial statement data.

References of papers in literature review.

Akita, R., Yoshihara, A., Matsubara, T. and Uehara, K., 2016, June. Deep learning for stock prediction using numerical and textual information. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)* (pp. 1-6). IEEE.

Chen, K., Zhou, Y. and Dai, F., 2015, October. A LSTM-based method for stock returns prediction: A case study of China stock market. In *2015 IEEE international conference on big data (big data)* (pp. 2823-2824). IEEE.

Chong, E., Han, C. and Park, F.C., 2017. Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Systems with Applications*, *83*, pp.187-205.

Dezsi, E. and Nistor, I.A., 2016. Can deep machine learning outsmart the market? a comparison between econometric modelling and long-short term memory. Romanian Economic and Business Review.

Huang, C.L. and Tsai, C.Y., 2009. A hybrid SOFM-SVR with a filter-based feature selection for stock market forecasting. *Expert Systems with applications*, *36*(2), pp.1529-1539.

Hussain, A.J., Knowles, A., Lisboa, P.J. and El-Deredy, W., 2008. Financial time series prediction using polynomial pipelined neural networks. *Expert Systems with Applications*, *35*(3), pp.1186-1199.

Jeon, S., Hong, B. and Chang, V., 2018. Pattern graph tracking-based stock price prediction using big data. *Future Generation Computer Systems*, *80*, pp.171-187.

Kraus, M. and Feuerriegel, S., 2017. Decision support from financial disclosures with deep neural networks and transfer learning. *Decision Support Systems*, *104*, pp.38-48.

Lee, M.C., 2009. Using support vector machine with a hybrid feature selection method to the stock trend prediction. *Expert Systems with Applications*, *36*(8), pp.10896-10904.

Lin, X., Yang, Z. and Song, Y., 2009. Short-term stock price prediction based on echo state networks. *Expert systems with applications*, *36*(3), pp.7313-7317.

Liu, S., Zhang, C. and Ma, J., 2017. CNN-LSTM neural network model for quantitative strategy analysis in stock markets. In *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017, Proceedings, Part II 24* (pp. 198-206). Springer International Publishing.

Krauss, C., Do, X.A. and Huck, N., 2017. Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. *European Journal of Operational Research*, *259*(2), pp.689-702.

Tsai, C.F. and Hsiao, Y.C., 2010. Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches. *Decision support systems*, *50*(1), pp.258-269.

Selvin, S., Vinayakumar, R., Gopalakrishnan, E.A., Menon, V.K. and Soman, K.P., 2017, September. Stock price prediction using LSTM, RNN and CNN-sliding window model. In *2017 international conference on advances in computing, communications and informatics (icacci)* (pp. 1643-1647). IEEE.

Wu, J.M.T., Li, Z., Herencsar, N., Vo, B. and Lin, J.C.W., 2023. A graph-based CNN-LSTM stock price prediction algorithm with leading indicators. *Multimedia Systems*, *29*(3), pp.1751-1770.

Yang, D. and Zhang, Q., 2000. Drift-independent volatility estimation based on high, low, open, and close prices. *The Journal of Business*, *73*(3), pp.477-492.

Zhuge, Q., Xu, L. and Zhang, G., 2017. LSTM Neural Network with Emotional Analysis for prediction of stock price. *Engineering letters*, *25*(2).

Zhou, B., 2019. Deep learning and the cross-section of stock returns: Neural networks combining price and fundamental information. *Available at SSRN 3179281*.