Design of a Simple End-to-End Fraud Detection System

Ryan Tandy

1 Preface

The following piece of work is a technical report for a simple end-to-end project of a fraud detection system. The project was designed to help me build a variety of technical skills. The skills include:

- Introduction to Heroku model deployment.
- Introduction to Git/Github.
- Command Line Interface skills - Virtual Environments, Git/Heroku.
- Python coding skills - PEP 8 style, modularity, functional coding, database interaction, Plotly Dash.
- Pandas/SQLite Data management - importing and exporting datasets, data manipulation, saving model training work using parquet/pickle files.
- Directory management.
- Model validation - Bootstrapping, sources of error, data leakage.
- Algorithm selection - What algorithms are most appropriate for the given data?
- Metric selection - What is the best way to evaluate model performance for the use case?
- Introduction to Density Estimation and generative models.
- Technical report writing.
- Debugging skills.

1.1 Project Information

The app for the project can be found at "fraud-system.herokuapp.com". The Github repository address is "https://github.com/ryantdata/Fraud-Detection-System/tree/main".

The graphs will appear when there are 10 observations to display.

The ratio of fraud occurrence in the app has been vastly increased to demonstrate the dashboard. The real occurrence of fraud is approximately 1 in 600 transactions; the occurrence in the app has been increased to 1 in 5. As such, the static metrics do not estimate the dynamic metrics.

The generation of new transactions is set to 1 per second. Dash callbacks are slow as the functions are calculated server side instead of client side. The way around this is to use clientside callbacks written in Javascript so that the functions can be computed locally - however - I did not want to get involved with Javascript for this project. I will likely revisit the project to implement this if I want to develop my Javascript skills.

Research of the domain was not carried out as I feel confident in my ability to perform domain specific research. I desired to keep the size of the project small so that I could complete it within a reasonable amount of time while focusing on the skills that I wanted to develop.

2 Problem Statement

Fraud is a thorn in the side for any business and can be both costly and disruptive. If fraud occurs, it is on the business to foot the bill as it is the responsibility of the business to prevent fraud from occurring. Therefore, it is vital that companies have a means to detect fraud when it happens. For larger companies, the number of transactions they process exceeds their ability to manually check whether the transaction is legitimate or not. As a result, these companies must rely on automated systems which can detect fraud in real-time and take the appropriate action to prevent the fraud. Hence, a well designed fraud detection system can be a valuable asset to a company.

A well designed system should:

- Automatically prevent a transaction from occurring if it is thought to be fraudulent.
- Maximise the number of fraudulent transactions which the system detects and blocks.
- Minimise the number of legitimate transactions that the system mistakenly detects as fraud and blocks.

2.1 Metric

A natural metric for this problem is a Modified Youden's Index (MYI) defined as

$$\text{MYI} = -1 + w_1(\text{Recall}) + w_2(\text{Precision})$$

where w_1 and w_2 are weights and MYI is identical to Youden's Index when $w_1 = w_2 = 1$. Recall measures the proportion of fraud that is detected by the system and Precision measures the proportion of blocked transactions which were fraud. Recall and Precision are defined as

$$\text{Recall} = \frac{TP}{TP + FN}, \qquad \text{Precision} = \frac{TP}{TP + FP}$$

where TP is the number of True Positives corresponding to the number of fraud transactions blocked, FN is the number of False Negatives corresponding to the number of fraud transactions missed by the system and FP is the number of False Positives corresponding to the number of legitimate transactions mistakenly blocked by the system.

3 System Overview

The system is made up of the following components:

- Generative Gaussian Mixture Model (GGMM) - generates synthetic transaction data (used to simulate a real-time data feed into the detection system.
- SQLite Database - stores the generated transactions.
- Predictive Logistic Model - predicts whether a transaction is fraudulent or not.
- Plotly Dash Application - calculates dynamic model metrics and displays them in real-time in a browser dashboard.
- Heroku Application - hosts the system online for public viewing.

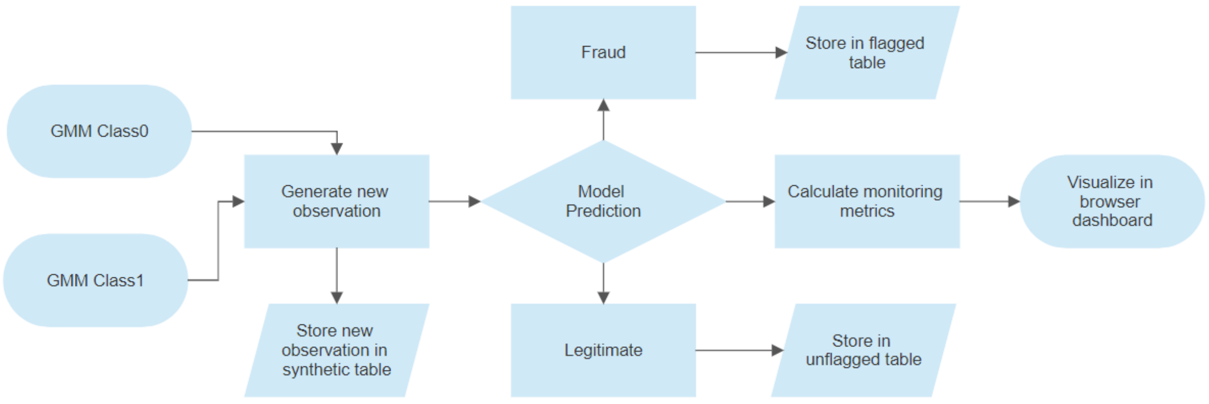
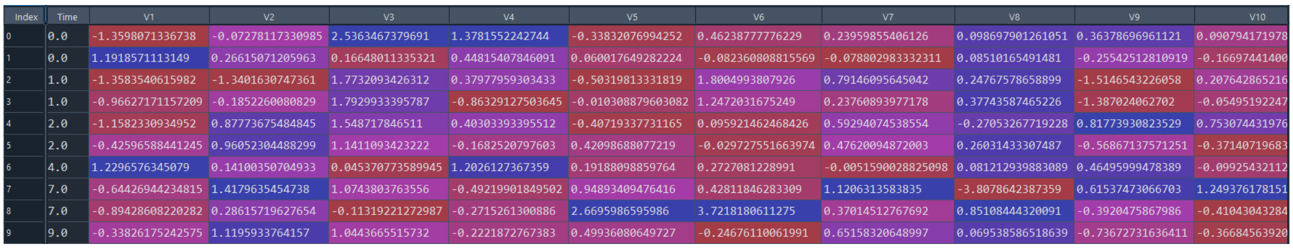


Figure 1: System Schematic

The schematic for the system is shown in Figure 1. There are two GGMM's; the `Class0` GGMM generates only legitimate transactions and the `Class1` GGMM generates only fraudulent transactions. A new transaction is generated once per second and a random number generator determines which model is used to generate the new observation. The new observation is stored in the `synthetic` table of the `transactions.db` database and also fed into the Logistic model. The logistic model makes a prediction as to whether it thinks the transaction is fraudulent or not, then the system stores the observation in the `flagged` table if the model believes it is fraudulent - or - in the `unflagged` table otherwise. The dynamic metrics in the dashboard are then updated and displayed.

4 Data Exploration

The supervised learning data contains 284807 observations and 31 features. The data has been anonymised - a standard practice when handling sensitive data - so the capacity for early insights and feature selection/engineering is severely limited.



Index	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	0.0	-1.3598071336738	-0.07278117330985	2.5363467379691	1.3781552242744	-0.33832076994252	0.46238777776229	0.23959855406126	0.098697901261051	0.36378696961121	0.090794171978
1	0.0	1.1918571113149	0.26615071205963	0.16648011335321	0.44815407846091	0.060017649282224	-0.082360808815569	-0.078802983332311	0.08510165491481	-0.25542512810919	-0.16697441400
2	1.0	-1.3583540615982	-1.3401630747361	1.7732093426312	0.37977959303433	-0.50319813331819	1.8004993807926	0.79146095645042	0.24767578658899	-1.5146543226058	0.207642865216
3	1.0	-0.96627171157209	-0.1852260080829	1.7929933395787	-0.86329127503645	-0.010308879603082	1.2472031675249	0.23760893977178	0.37743587465226	-1.387024062702	-0.05495192247
4	2.0	-1.1582330934952	0.87773675484845	1.548717846511	0.40303393395512	-0.40719337731165	0.095921462468426	0.59294074538554	-0.27053267719228	0.81773930823529	0.753074431976
5	2.0	-0.42596588441245	0.96052304488299	1.1411093423222	-0.1682520797603	0.42098688077219	-0.029727551663974	0.47620094872003	0.26031433307487	-0.56867137571251	-0.37140719683
6	4.0	1.2296576345079	0.14100350704933	0.045370773589945	1.2026127367359	0.19188098859764	0.2727081228991	-0.0051590028825098	0.081212939883089	0.46495999478389	-0.09925432112
7	7.0	-0.64426944234815	1.4179635454738	1.0743803763556	-0.49219901849502	0.94893409476416	0.42811846283309	1.1206313583835	-3.8078642387359	0.61537473066703	1.249376178151
8	7.0	-0.89428608220282	0.28615719627654	-0.11319221272987	-0.2715261300886	2.6695986595986	3.7218180611275	0.37014512767692	0.85108444320091	-0.3920475867986	-0.41043043284
9	9.0	-0.33826175242575	1.1195933764157	1.0443665515732	-0.2221872767383	0.49936080649727	-0.24676110061991	0.65158320648997	0.069538586518639	-0.73672731636411	-0.36684563920

Figure 2: Shows the first few rows and columns of the anonymised transaction data.

The head of the data is shown in Figure 2. The data was checked for missing values, but none were missing. Had there been missing values - imputing the mean would be a good strategy for this purely numerical dataset with a large number of observations.

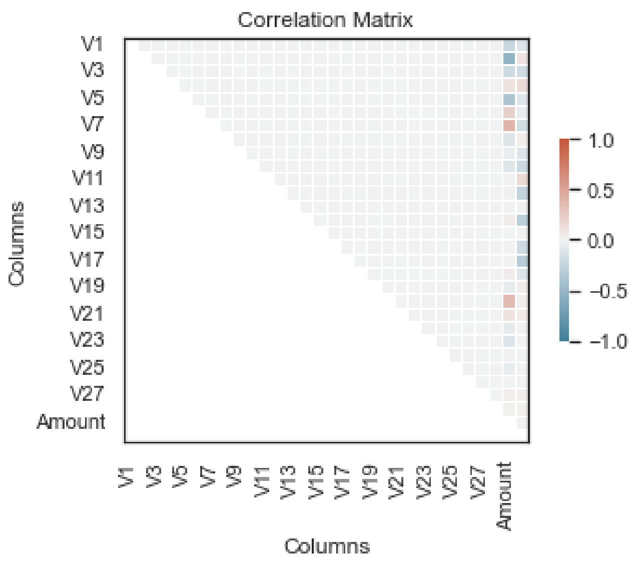


Figure 3: Shows a heatmap of the correlation strength between the features.

The correlation matrix is shown in Figure 3. Most variables are uncorrelated or very weakly correlated with each other, so the assumption of independence between the predictors holds very well. There are some features that appear uncorrelated with the target. However, since the features are anonymised, it is risky to remove those features as non-negligible interactions may inadvertently be removed in the process. Due to this issue, all features were kept for model training.

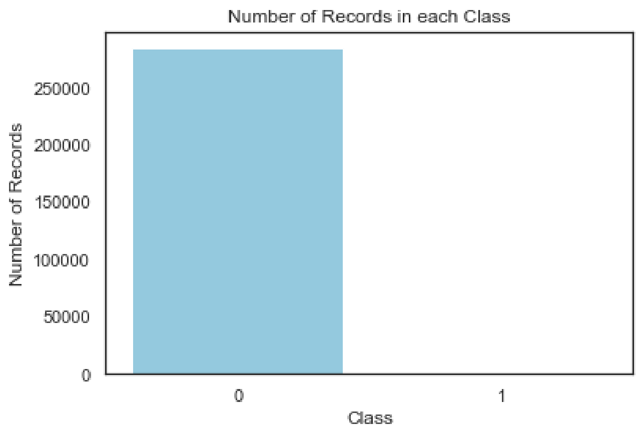


Figure 4: Shows a barchart counting the number of observations in each class.

The number of observations for each class are shown in Figure 4. The classes have a large imbalance; the ratio of `class1` to `class0` is 1:587.

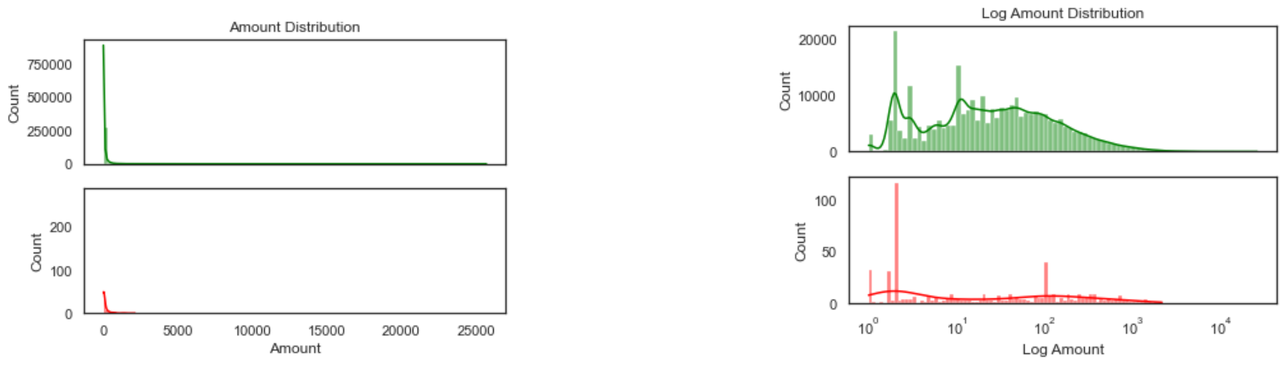


Figure 5: Show the Amount feature and log-transform of the Amount feature.

The distributions of most features look uninteresting and do not need transforming prior to model training. However, the `Amount` feature is exponentially distributed, so a log-transform is necessary for this feature prior to model training. The distribution and log-transformed distribution for the `Amount` feature are shown in Figure 5. The `mean` and `std` for the Amount column for each class are

$$\mu_0 = 88.29, \qquad \sigma_0 = 250.11, \qquad \mu_1 = 122.21, \qquad \sigma = 256.68.$$

The variance is similar, but there is a substantial difference in the means.

5 Database Schematic

The schematic for the database shown in Figure 6 shows two primary tables and 5 tables which are subsets of the same data in one of the primary tables. The `labelled` table contains the supervised learning data with which to train and assess the machine learning models. The data is randomly split into the `training`, `validation`, and `test` tables which contain 50%, 25%, and 25% respectively of the `labelled` data. The `synthetic` table contains synthetic data generated by the generative GGMM models. The `flagged` and `unflagged` tables store the observations from the `synthetic` data according to whether the final model predicted them as fraudulent or not. This would serve as data for future assessment and improvements of the system.

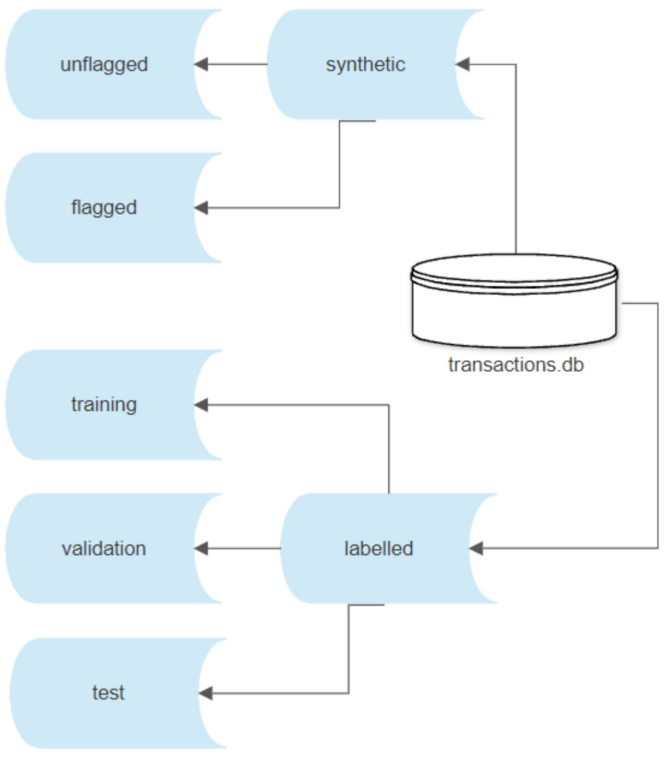


Figure 6: Database Schema.

5.1 Dataset Creation

The `Amount` column of the dataset was log-scaled. The `labelled` data was split into the `training`, `validation` and `test` sets. Then each set was standardized individually. Note that standardizing the data prior to splitting the data would result in data leakage as every set would be standardized with the same mean and variance despite containing different data.

6 Generative Gaussian Mixture Model (GGMM) Training

6.1 Metric

The Bayesian Information Criterion (BIC) was chosen to evaluate the performance of the GGMM's. The BIC is defined as

$$\text{BIC} = k\ln(n) - 2\ln(\hat{L})$$

where k is the number of parameters estimated by the model, n is the number of observations in the dataset, and $\hat{L} = p(x|\hat{\theta}, M)$ is the maximum value of the likelihood function given the parametrised model $M(\theta)$ on the observed data x. The BIC does not give information on the absolute quality of the model fit, but only on the fit relative to other models, so serves as an effective comparison measure.

The choice for the BIC over the AIC (Akaike Information Criterion) is because the first term in the BIC $k\ln(n)$ depends on the number of observations as well as the number of parameters, whereas the equivalent term in the AIC $2k$ depends only on the number of parameters. Since the number of observations in the dataset is large - leading to the possibility of a large amount of clusters within the data - there is need for a measure which more aggressively penalizes the degrees of freedom. Hence, why the BIC is preferred over the AIC in this case.

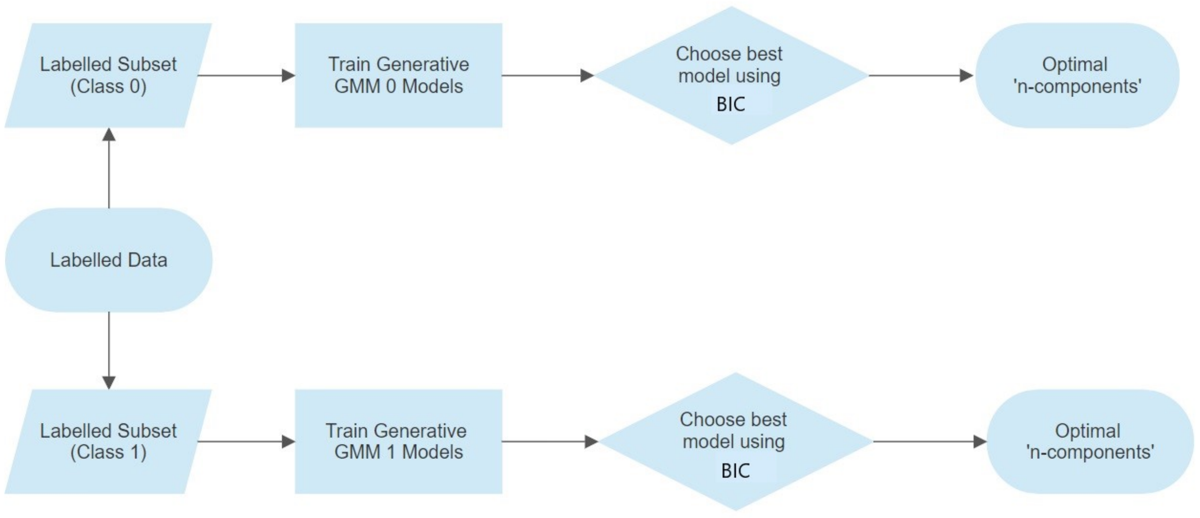

Labelled Subset (Class 0)
Train Generative GMM 0 Models
Choose best model using BIC
Optimal 'n-components'
Labelled Data
Labelled Subset (Class 1)
Train Generative GMM 1 Models
Choose best model using BIC
Optimal 'n-components'

Figure 7: GMM Model Training Flowchart

The GGMM training is shown in Figure 7. The entire supervised learning dataset was split into two subsets - divided by class label - then a GGMM was trained on each subset to model the underlying distribution of the data. The `n_components` hyper-parameter - which controls the number of clusters fit by a model - was tuned by calculating the BIC for each model and choosing the value of `n-components` which minimizes the BIC. Since the sample size for `class0` is very large, the `covariance_type` was set to `spherical` - which calculates a single variance for each cluster - so that training time was significantly decreased. After an approximation was found, the chosen model was retrained using the `full` type, which calculates the full covariance matrix for each cluster. For `class1`, the type was set to `full`.

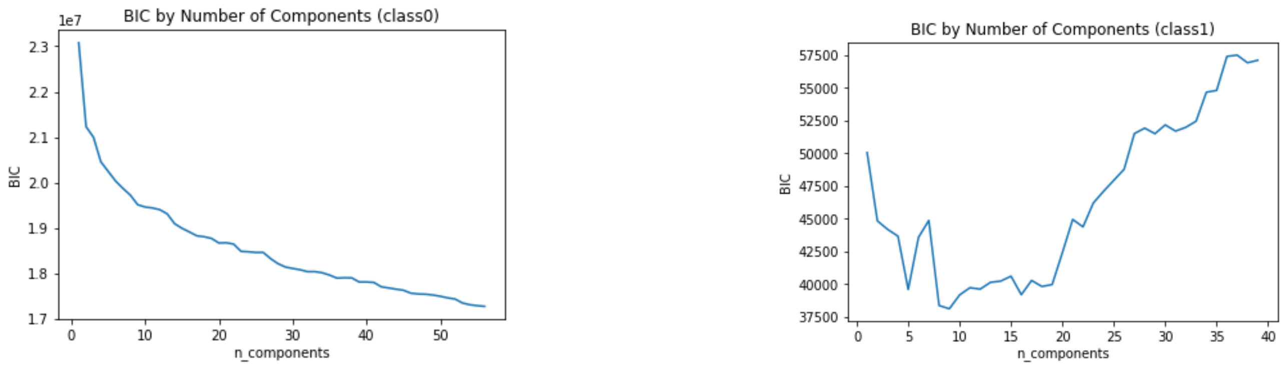


Figure 8: BIC curve for GGMM model selection for each class.

The BIC for the `class0` data kept decreasing with no signs of finding a minimum for `n-components`> 50, as shown in Figure 8. The BIC for `class1` found a minimum at `n-components`$= 9$ before dramatically increasing afterwards.

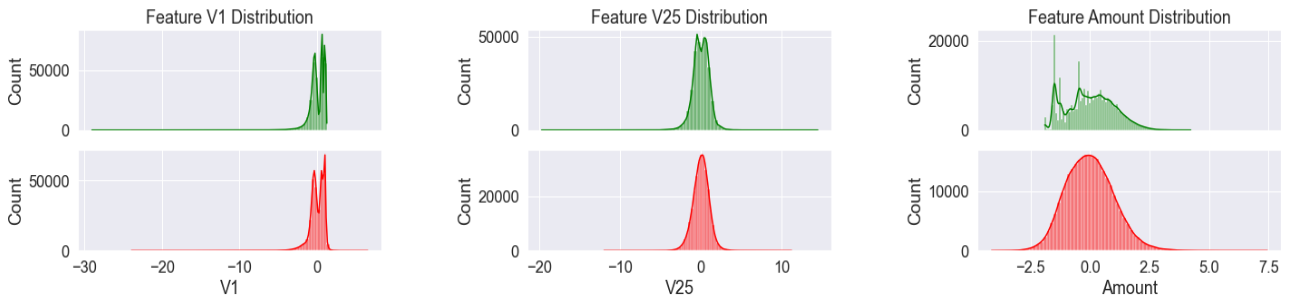


Figure 9: Examples of univariate distribution comparison for the real and synthetic data. The green distributions represent the real data and the red distributions represent the synthesized data. The synthetic distributions closely match the real data in most cases, but the model struggles to mimic the Amount feature.

The training time for `class0` began to increase substantially towards the end of the training session. Therefore, `n-components`$= 50$ was chosen and a synthetic sample produced by the model was compared with the real data to visually inspect the quality of the data generation. Examples of this comparison are shown in Figure 9. The synthetic distributions for each feature closely match the distributions of the real data. However, the model has difficulty modelling the `Amount` feature - as shown in the right plot. If the quality of synthetic data was critical, then one possibility would be to model the feature independently. However, the quality is good enough for the scope of this project so the model was accepted.

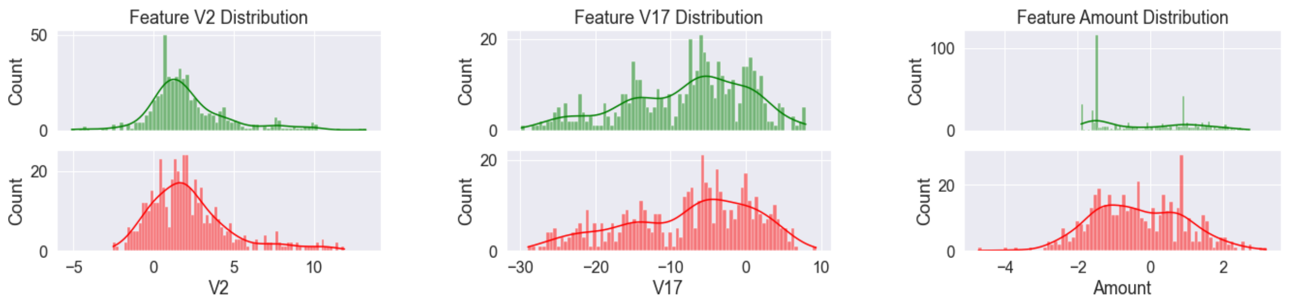


Figure 10: Examples of univariate disitrbution comparsion for the real and synthetic data. The green distributions represent the real data and the red distributions represent the synthesized data.

Like in the previous case, a synthetic sample was generated using the `class1` GGMM and then compared with the real data. Examples from the visual inspection can be seen in Figure 10. Again, the model mimics the data very well in all cases apart from the `Amount` feature. Therefore, the model was accepted.

6.1.1 Aside

Another option to assess the similarity of the synthetic data to the real data is to use the Kolmogorov-Smirnov Test, which compares the cumulative distributions and calculates a p-value corresponding to the likelihood that the distributions are identical; though as the sample size becomes large, the test becomes over-zealous and rejected the null hypothesis even if the distributions are very close to identical. This is why a visual inspection was preferred given the large dataset.

7 Model Training

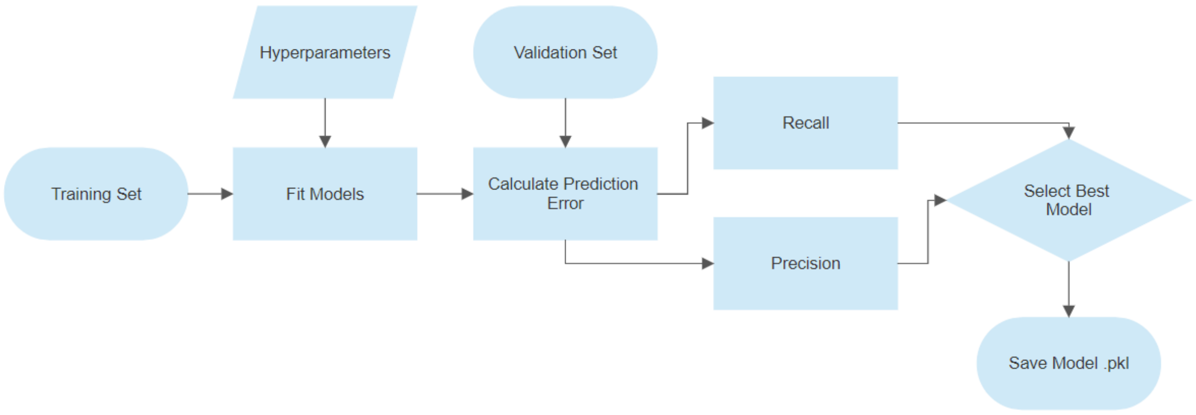


Figure 11: Model Training Flowchart.

The work-flow for the predictive model training process is shown in Figure 11. N models are fit with unique sets of hyper-parameters and then trained on the `training` data. The `validation` set is used to calculate the Precision and Recall for each model and then the MYI from section 2.1 is calculated. The model which maximises the MYI is chosen. Maximising the MYI corresponds to minimising the conditional test error given by

$$\min\{\mathrm{Err}_{\tau,\nu,i}\} = \min\{1 - \mathrm{MYI}_i|\tau,\nu\}$$

where τ is the `training` set, ν is the `validation` set, and i is the i^{th} trained model. Since the validation set is used for model selection, the value for $\mathrm{Err}_{\tau,\nu}$ is biased and so the true conditional test error must be estimated using a second independent test set, i.e. $\mathrm{Err}_{\tau,t}$ where t is the `test` set.

7.1 Algorithm Selection

Logistic models are both fast and robust to overfitting when used with regularization. Logistic models have a tendency to be high bias and low variance, but offer good performance on a wide-range of datasets. Therefore, a Logistic model was chosen to get a baseline model performance on the dataset.

XGBoost models are well suited binary classification tasks with imbalanced class sizes. The algorithm works by assigning increasingly bigger weights to misclassified observations. This property of the algorithm increases the visibility of the minority class - however - regularization techniques must be used as the same property can lead to overfitting of the data. The XGBoost algorithm is much slower on large datasets, but was expected to produce the best performing model. Hence, the algorithm was chosen to model the data.

7.1.1 Aside

Neural networks are another possibility for the choice of algorithm, but outside of the scope of this project so are not considered.

7.2 Logistic Model Training

The `newton-cg` solver was chosen as it converges the fastest when the number of features in the dataset is low, which is useful since the data contains only 31 features, but almost 300,000 observations. The parameter `C` - which controls the strength of regularization - was used along with `L2` regularization, the only type of regularization available with the `newton-cg`. In total, 101 models were trained. Most of the models had identical metrics and so visualizations of the results are not informative, so are omitted. These results are contained in the `pqdata\logreg_metrics.pq` file.

The following table contains the configuration and metrics for the best performing logistic model according to the modified Youden's Index.

`C`	γ	Recall	Precision	Best Threshold	Confusion Matrix
0.001	2.144	0.788	0.781	0.02	$\begin{pmatrix} 71064 & 25 \\ 24 & 89 \end{pmatrix}$

The best model is saved to the pickle file `pklmodels\lr_model.sav`.

7.3 XGBClassifier Model Training

A randomized grid search was performed over a discretized 8-dimensional hyper-parameter space where the dimensions are:

	`learning_rate`	`min_split_loss`	`max_depth`	`min_child_weight`
Range	[0.01, 0.3]	[0, 20]	[2, 15]	[1, 20]

	`max_delta_step`	`reg_lambda`	`reg_alpha`	`n_estiamtors`
Range	[0, 20]	[0, 20]	[0, 20]	[25, 400]

In total, 362 models were trained; the best performing model has the following hyper-parameter choices and performance metrics:

`learning_rate`	`min_split_loss`	`max_depth`	`min_child_weight`
0.222	0.4	14	2

`max_delta_step`	`reg_lambda`	`reg_alpha`	`n_estimators`
8.5	4.8	1.9	182

MYI	Recall	Precision	Best Threshold	Confusion Matrix
2.384	0.823	0.915	0.25	$\begin{pmatrix} 71062 & 10 \\ 23 & 107 \end{pmatrix}$

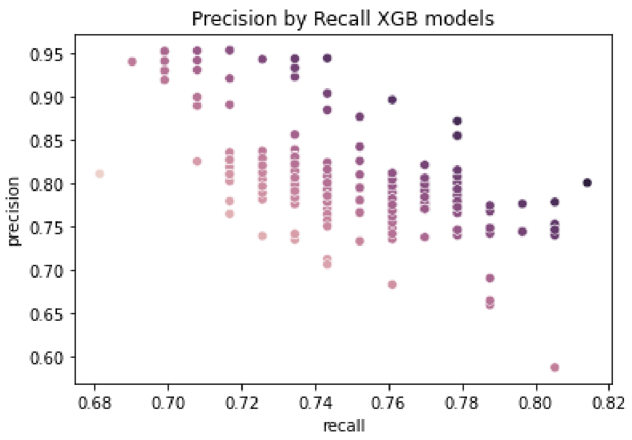


Figure 12: XGB Trained Models Performance Metrics. Higher MYI values are represented as a darker shade.

The Recall and Precision for each of the trained XGB models are shown in Figure 12. The chosen model is the right-most point and corresponds to the highest MYI. Note that many of the dots represent multiple models as Recall and Precision are practically discrete despite being theoretically continuous variables. The raw results are contained in the `pqdata\xgb_metrics.pq` file.

The best XGBoost model outperforms the best Logistic model on the data, so the chosen model for deployment is the best-in-class XGBoost model specified in the previous table. The best model is saved to the pickle file `pklmodels\xgb_model.sav`.

8 Model Assessment

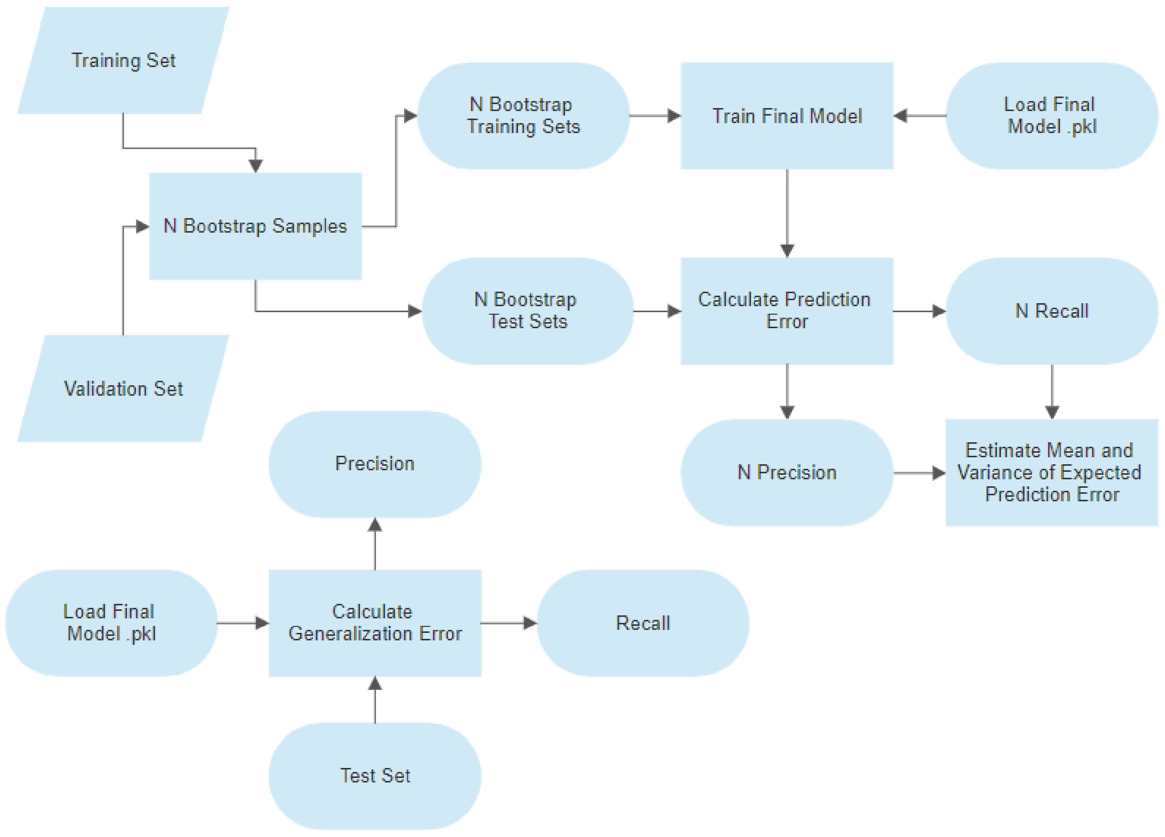


Figure 13: Final Model Assessment Procedure.

The assessment for the final model consists of two parts which are shown in Figure 13. Part one is a bootstrapping procedure. 344 bootstrap samples were created by randomly re-sampling the `labelled` dataset for each bootstrap sample and then randomly splitting it into a train and validation set. For each bootstrap sample, the model training workflow in section 7 was followed. From this, the expected Recall and Precision (expected test error $E[\mathrm{Err}_\tau]$) were calculated with the following:

$$\widehat{E[S(Z)]} = \frac{1}{B}\sum_{b=1}^{B} S(Z^{*b}),$$

where S is either the recall or precision, B is the number of bootstrap samples and Z is the bootstrap sample data. The variance is estimated using

$$\widehat{\mathrm{Var}[S(Z)]} = \frac{1}{B-1}\sum_{b=1}^{B}(S(Z^{*b}) - S^*)^2.$$

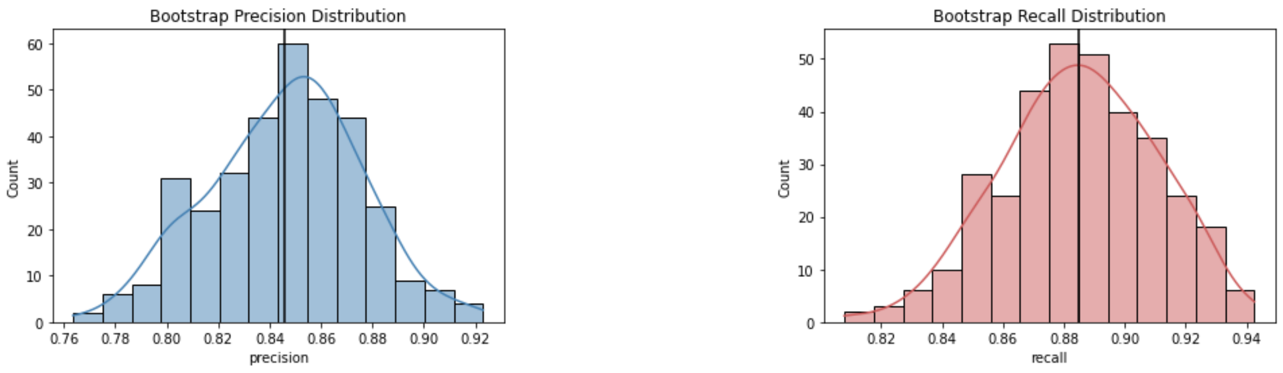


Figure 14: Shows the distribution of Recall and Precision for the 344 bootstrap samples.

The distributions for the Precision and Recall are shown in Figure 14. The distributions are expected to be approximately normal, but the plots show some skewness remains due to the relatively low sample size for Monte-Carlo estimation. The raw bootstrap results are stored in the `pqdata\bootstrap_results.pq` file.

The second part of the model assessment procedure uses the test set to estimate the true conditional test error $\text{Err}_{\mathcal{T},t}$ - i.e. the generalization error for the model. The estimated true conditional test error $\text{Err}^*_{\mathcal{T},t}$ gives the best theoretical point estimate for future model performance given the supervised learning data.

The estimated variables are given in the following table.

$E[\text{Recall}]^*$	$\text{Std}[\text{Recall}]^*$	$\text{Recall}^*_{\mathcal{T},t}$	$E[\text{Precision}]^*$	$\text{Std}[\text{Precision}]^*$	$\text{Precision}^*_{\mathcal{T},t}$
0.885	0.0253	0.854	0.846	0.0296	0.822

8.1 Sources of Bias

The bootstrap samples were created using the whole dataset since the bootstrap samples are only used for assessment and not model selection, so no data leakage has occurred. However, $\text{Err}_{\mathcal{T},t}$ was estimated with the model that was only trained on the `training` set data. Therefore, it is expected that $\text{Err}^*_{\mathcal{T},t}$ is biased pessimistically relative to $E[\text{Err}_{\mathcal{T}}]^*$ obtained from the bootstrapped samples. The deployed model was also trained on the entire dataset afterwards, so it is expected that both $\text{Err}^*_{\mathcal{T},t}$ and $E[\text{Err}_{\mathcal{T}}]^*$ are pessimistically biased relative to $\text{Err}_{\mathcal{T},t}$.

8.1.1 Aside

In reality, 344 bootstrap samples is rather low for a Monte-Carlo estimation of unknown quantities; estimates should use at least 1000 samples for a reliable estimate. However, due to the computational expense and the small scope of the project, it was decided that 344 samples would provide a satisfactory estimate.

9 Dashboard

The dashboard for the system was created using the plotly `dash` library. The dashboard tracks the generated data and various model performance metrics in real time, while also storing the new data in the database.

At the top of the dashboard are the system controls. The `Start` button tells the application to start generating new transactions using the GGMM models and then the application calculates all the metrics and visualizations as well as uploading the new data into the appropriate database tables. This process is shown in section 3. The `Stop` button simply halts all processes. The `Reset` button resets all visualizations, metrics, and deletes all the data stored in the `synthetic`, `unflagged` and `flagged` tables.

Most of the metrics are self-explanatory, but `blocked` and `unblocked` simply refers to whether the system decided to block the transaction or not. The `Static Model Metrics` are derived from the bootstrapped samples and taken from the table in section 8. The confidence intervals are approximate, found with the following formulas:

$$CI_R = [E[\text{Recall}]^* \pm 2(Std[\text{Recall}]^*)], \qquad CI_P = [E[\text{Precision}]^* \pm 2(Std[\text{Precision}]^*)].$$

The estimated `Expected Block Rate` ($E[BR]$) is the percentage of transactions the system is expected to block given the ratio of fraud in the supervised data

$$RF = \frac{F}{L}$$

where F is the number of fraud cases and L is the number of legitimate cases and the estimated performance of the model $E[\text{Recall}]^*$ & $E[\text{Precision}]^*$ - i.e.

$$E[BR] = \frac{TP + FP}{N} = RF^{-1}\left(E[\text{Recall}^*] + \frac{FP}{TP + FN}\right)$$

where N is sample size of the supervised data. The calculation to find TP and FP is as follows:

$$\frac{N}{RF} = TP + FN.$$

Then

$$E[\text{Recall}]^* = \frac{TP}{TP + FN} \qquad \Rightarrow \qquad TP = E[\text{Recall}]^*(TP + FN)$$

and

$$E[\text{Precision}]^* = \frac{TP}{TP + FP} \qquad \Rightarrow \qquad FP = \frac{TP}{E[\text{Precision}]^*} - TP.$$

The estimated `Expected Miss Rate` $E[MR]$ is the percentage of fraudulent transactions expected to go undetected compared to the number of transactions in total, obtained using

$$E[MR] = \frac{FN}{N}.$$

The expected percentage of fraudulent transactions missed in relation to the number of fraudulent transactions is simply $1 - E[\text{Recall}]^*$. This quantity is implicit on the dashboard.

The plot distributions for Recall and Precision are obtained from the bootstrap samples.