IT1244 Project Report: Credit Card Approval Dataset

Team 8

Chia Kwang Yang
Lee Weng Heng, Jon
Lim Zhi Yong
Qiu Zi Hao

Introduction

Credit cards are one of the most popular financial instruments issued by banks for items or services using credit. Banks spend a significant amount of resources to accept or reject people applying for credit cards, mostly for a variety of reasons. Using machine learning, we can cut down on the amount of resources for such an important financial instrument. This project allows us to segment customers in our dataset to accept and reject people applying for the credit cards. This project is relevant to our module as we can use simple machine learning methods taught in this course and incorporate other methods that may be relevant and more efficient.

There has been several works done to improve the efficiency in processing credit card applications. These examples include:

- Automating the application process for processes such as data entry and document classification
- Streamlining the application process by requesting fewer documents and allowing documents to be submitted online
- Using predictive analysis to predict which applications are most likely to be approved and prioritise those applications

However, these works consisted of several drawbacks, such as:

- Inaccurate data when using predictive analysis
- Human intervention still required for complex or high-risk applications

Dataset

The dataset itself consists of two files, “application” and “credit_record” csv. The application dataset consists of records of people that want to apply for a credit card. There are 18 columns with different features of a customer, with examples of columns like income and house type. The credit_record dataset only consists of 3 datasets, the id, month_bal and the status column.

There were a few issues with the dataset itself. Firstly, if we were to take a look at the application dataset itself, we realise that the application dataset does not have any column for classification. This means that the dataset needs some form of classification, which leads to our first method of using clustering to self-define these clusters. The second issue is that we would need to find a way to join the application dataset with the credit_record dataset. However, not every id in the application dataset is present in the credit_record dataset. Lastly, some columns in the dataset are categorical and therefore would need to be encoded. Therefore, we have decided to preprocess the dataset as such:

For the application dataset:

- We fill in missing data in ‘job’ column based on their ‘income_type’
- Convert ‘own_car’ and ‘own_realty’ from alphabetic data to numeric data
- Applied logarithmic scaling to ‘income’
- Convert ‘birth_day’ and ‘employment_length’ in terms of years
- Convert ‘income_type’, ‘education_level’, ‘family_status’, ‘house_type’ and ‘job’ columns to one-hot encoding

For the credit_record dataset:

- We first group the dataset in terms of the ID column.
- We then extract features from the status column to count the number of Cs, the number of Xs and the months late.
- Finally, we merge the dataset with the application.

Methods

We will split our technical approach into two to solve the problem. Firstly, we need to cluster the dataset so that we can properly cluster the dataset into different k groups. We use the k-means clustering algorithm for the first part of the problem.

For the classification part of the issue, we evaluate the credit card dataset using 3 different non-deep learning methods. These 3 methods are logistic regression, decision trees and gradient boosting (specifically, XGBoost). Decision trees and gradient boosting are methods not taught in this course, and therefore we will explain these methods.

The reason that we do not consider Deep Learning is because we want to consider the interpretability of the methods. Even though deep learning provides us better results as compared to other methods, we have to consider the context of the dataset. In the financial industry, there is a lot of red tape (Hubbis, 2015) and therefore we will have to explain the model's decision in accepting or rejecting the client. Deep Learning is highly accurate, but is not very interpretable compared to traditional methods. This is not the case with the other methods that we use in this project, as the other methods are interpretable. We are able to get the weights of each column in Logistic Regression to see the importance of each column, and we are able to get the feature importance of each column for Decision Trees and Gradient Boosting.

In the file 'train.ipynb', we first used the 'train_test_split' method to implement a 70/30 split of the dataset into training and testing data. For simplicity, we also implemented a function 'result' which takes in the training and testing data as well as the model. The function trains the model using the training data followed by predicting the probability of whether a randomly chosen client will be accepted or rejected based on the test data.

We will now explain the clustering algorithm and the 3 different methods used in classification for our project and how we implemented them in our code. We mostly used supervised learning methods.

K-Means Clustering

K-means clustering is an unsupervised learning method where similar data points are identified and grouped into clusters such that data points within each cluster are similar. It is an iterative process that first initialises k centroids randomly. Then, we will assign each datapoint to the closest centroid. After all data points have been assigned, within each cluster, we will calculate the new centroid of the assigned data points. Repeat the process until there is no change to the centroids. Once the algorithm has converged, the data points will have been grouped into k clusters based on their similarity to each other.

Logistic Regression

The logistic regression model uses a logistic function, given by the sigmoid function $g(x) = \frac{1}{1+e^{-x}}$, to model the relationship between the input features and the probability that the output variable is true. The sigmoid function maps any real-valued number to a probability value between 0 and 1. The model is trained using a dataset that contains input features and output labels. During training, the algorithm adjusts the model parameters to minimise the difference between the predicted probabilities and the actual output labels. Once the model is trained, it can be used to predict the output probability of new input data. If the predicted probability is greater than a certain threshold value (typically 0.5), the output is predicted to be true, and if it is less than the threshold, the output is predicted to be false. One of the downsides of logistic regression is that it may not perform well on data with non-linear boundaries, which decision trees are more suitable for.

We used the 'scikit-learn' library to import 'LogisticRegression' from the module 'linear_model'. For our large dataset, we went for a higher 'max_iter' value of 20000 as the solver was unable to converge with the set amount of 100 iterations. Then, we ran our 'result' function to obtain the classification report.

Decision Trees (not covered in IT1244)

Decision trees are graphical representations of a series of decisions and their possible consequences. It consists of nodes, branches, and leaves, where each node represents a decision based on one or more input features, each branch represents a possible outcome of that decision, and each leaf node represents the final prediction or decision. The process of building a decision tree involves recursively splitting the data into smaller subsets based on the values of the input features, with the goal of maximising the information gain or minimising the impurity at each split. One advantage of decision trees is that they are easy to interpret and visualise. However, decision trees are prone to overfitting, especially when the data has a large number of features. Using methods such as gradient boosting can help mitigate the issue (Analytixlabs, 2022).

To implement the decision tree, we used 'RandomForestClassifier' from the module 'tree' in 'scikit-learn'. We went for a max_depth value of 8 (which is defined by the number of layers between the root node and the leaf node), as we wanted our model to converge.

Gradient Boosting (not covered in IT1244)

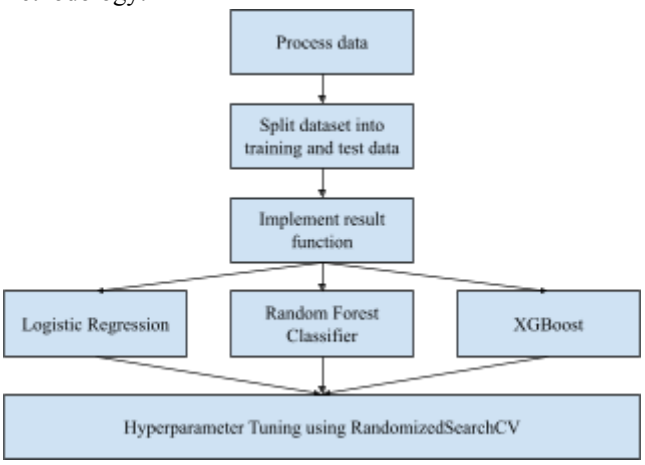
Gradient boosting works by combining multiple weak learners, usually decision trees, into a strong predictor. The algorithm starts with a single tree and iteratively adds new trees, with each new tree trained to correct the errors of the previous ones. At each iteration, the algorithm calculates the gradient of the loss function with respect to the predictions, and trains a new tree to minimise this gradient. One key advantage of XGBoost is its ability to handle high-dimensional data with many features. It can automatically handle feature selection, allowing it to perform well on large datasets. It also "does not require optimization of the parameters or tuning", and is also able

to control the problem of overfitting especially for our large dataset (Simplilearn, 2023).

We used 'XGBClassifer' from the 'XGBoost' library. We used a 'n_estimator' value of 100, a 'learning_rate" of 1.0, a "max_depth" of 1 and an "objective" of "binary:logistic".

We then followed up by using **RandomisedSearchCV** to find the best hyperparameters for each of the models. This method is the randomised version of GridSearchCV, which helps us to "automate the tuning of hyperparameters" to enhance model performance (Great Learning Team, 2023). RandomisedSearchCV tries random combinations of parameters provided to find the best hyperparameters and is known to deliver similar results to GridSearchCV.

The following flow chart is a summary of our methodology:



Results & Discussions

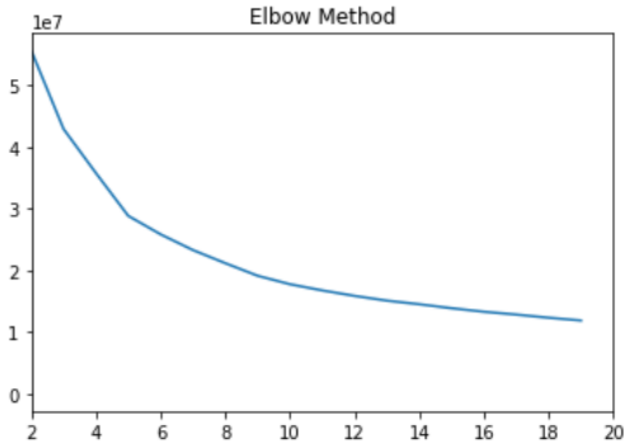
Firstly, let us discuss the results of the K-means clustering algorithm.

K-Means Clustering

For the clustering algorithm itself, we measure the error of the clustering using the within sum of squares (WSS), which is the distance between each datapoint and its closest cluster. We calculate the WSS for every k value from 2 to 20, and plot the error for each k value. The elbow method suggests that there is a huge fall for the first few values of k, and eventually reaches diminishing returns.

Moreover, the clustering algorithm itself has a high level of error. The WSS is high throughout all values of k, and could be attributed to the size of the dataset with the dataset having well over 438000 data points.

Lastly, we used k++means to initialise the cluster instead of typical k-means, as it would allow better clustering of data points instead of having the clusters be very near each other, which might cause huge inaccuracies. The plot of the elbow method is shown below.



For each of our models, we obtained the classification report to find the precision, recall, F1-score and support.

We evaluated each model by obtaining the classification report with the following values: precision, recall, F1-score and support.

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$F1\,Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Support is a measure of how frequently the collection of items occurs together as a percentage of all transactions.

We tried not to use accuracy as it only takes into account the true positives (TP) and true negatives (TN), which is not what we are most concerned about. The cost of having a false positive (FP) is higher than a false negative (FN) as we can deny the loan to a person who is not a defaulter, but we cannot give a loan to a person who is a defaulter. Therefore, we should seek to minimise the number of FP, and consequently, maximise the precision. The higher the precision, the better. Additionally, accuracy is generally not a good indicator of model performance when our dataset has a class imbalance (i.e. the likelihood of whether a loan is given is very different).

Let's analyse the precision of each method.

	precision	recall	f1-score	support
0	0.43	0.18	0.25	32175
1	0.78	0.92	0.84	99393
accuracy			0.74	131568
macro avg	0.60	0.55	0.55	131568
weighted avg	0.69	0.74	0.70	131568

The above is the classification report for our logistic regression model. It has a precision of 0.43 in determining whether or not to accept the credit card application. Evidently, the model is not very good. This is why we

considered other methods like decision trees and gradient boosting.

	precision	recall	f1-score	support
0	0.99	0.48	0.64	32175
1	0.86	1.00	0.92	99393
accuracy			0.87	131568
macro avg	0.92	0.74	0.78	131568
weighted avg	0.89	0.87	0.85	131568

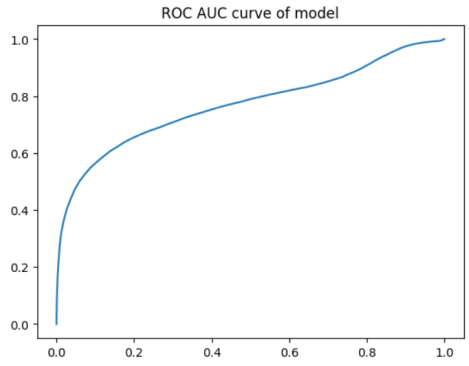
The classification report for Random Forest Classifier is much better. The precision for both outcomes is close to 1.

	precision	recall	f1-score	support
0	0.88	0.84	0.86	32175
1	0.95	0.96	0.96	99393
accuracy			0.93	131568
macro avg	0.92	0.90	0.91	131568
weighted avg	0.93	0.93	0.93	131568

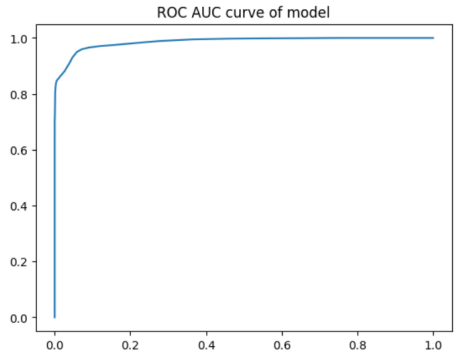
Similarly, the precision of XGBoost is quite solid with values close to 1 as well.

We then plotted various ROC-AUC curves that tell us how efficient each model is.

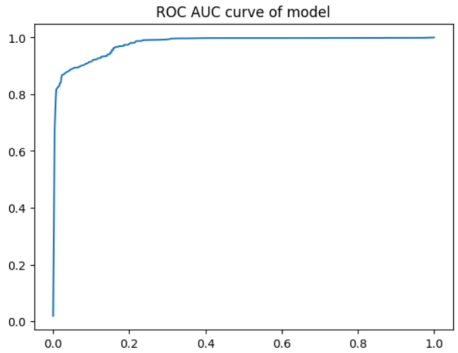
Logistic Regression



Random Forest Classifier



XGBoost



Lastly, we also did experiments with the hyperparameters of each of the models. We found out that with hyperparameter tuning, logistic regression improves slightly when the learning rate of logistic regression is changed. However, for the other two models, changing different values such as the learning rate and the depth of the model itself only led to the model overfitting. With that, we have decided to go for a baseline gradient boosting model instead since it performed the best without overfitting.

References

Simplilearn. (2023, February 23). *What is XGBoost? An Introduction to XGBoost Algorithm in Machine Learning.* Simplilearn.com. Retrieved March 31, 2023, from https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article

Analytixlabs.co (2022, October 19). *A Detailed Guide On Gradient Boosting Algorithm With Examples.* Retrieved March 31, 2023, from https://www.analytixlabs.co.in/blog/gradient-boosting-algorithm/

Great Learning Team. (2022, June 13). *Hyperparameter Tuning with GridSearchCV.* Great Learning Blog: Free Resources what Matters to shape your Career! Retrieved March 31, 2023, from https://www.mygreatlearning.com/blog/gridsearchcv/

Association rule. GeeksforGeeks. (2023, January 11). Retrieved March 31, 2023, from https://www.geeksforgeeks.org/association-rule/

Hubbis. (2015, June 29). *Cutting through the red tape.* Asian Wealth Management and Asian Private Banking. Retrieved March 31, 2023, from https://www.hubbis.com/article/cutting-through-the-red-tape