Comparative Study of Supervised Learning Methods for Credit Card Fraud Detection

Au Liang Jun (*e0203163@u.nus.edu*), Dexter Wah Yixiang (*e0203109@u.nus.edu*), Joycelyn Ng (*e0007771@u.nus.edu*), Kenny Ng Jian Liang (*e0004448@u.nus.edu*), Liew Jia Hong (*e0176269@u.nus.edu*), Yan Hong Yao Alvin (*e0052842@u.nus.edu*)

National University of Singapore

CS3244 Machine Learning

13 November 2018

Abstract

Despite implementation of fraud analytics and Europay, MasterCard and Visa (EMV) technology, credit card fraud rates have risen over the years. In this study, we compare the effectiveness of supervised learning methods in detecting fraudulent credit card transactions with the use of an appropriate performance metric. We envision that findings from this comparative study could help credit card companies improve their fraud detection technology, and could be extended to detect other forms of fraud.

1 Introduction

1.1 Motivation

As cashless payments grow increasingly popular, the number of reported credit card fraud cases has likewise been on the rise. Financial loss attributed to credit card fraud is also estimated to exceed US$35.5 billion in 2020 according to a 2014 Nilson Report [1]. To prevent huge losses for companies as well as to safeguard the personal data and financial assets of consumers, it is imperative to be able to reliably detect credit card fraud.

1.2 Guidelines

The focus of this study will be to examine the effectiveness of various resampling techniques and supervised learning algorithms for credit card fraud detection, as opposed to maximising the performance of a single supervised learning algorithm. Through this study, we hope the answer following guiding questions:

1. Which is the most appropriate supervised learning algorithm for identifying fraudulent transactions and why?
2. What techniques can be applied to address the problem of class imbalance?
3. Is it possible to amalgamate the merits of various supervised learning algorithms to develop a more powerful model capable of detecting fraudulent transactions?
4. Are we able to apply a similar workflow and approach on other classification problems?

For this study, we will work on a credit card transaction dataset from the Université Libre de Bruxelles (ULB) Machine Learning Group, made available through the Kaggle platform [2]. Although the dataset is hosted on Kaggle, it is without an associated Kaggle competition. Hence, there is no specified performance metric and we will have to define an appropriate performance metric to evaluate model performance.

2 Dataset and Related Work

2.1 Dataset

The dataset we are working with contains 284807 actual credit card transactions. Each transaction has 30 features and is classified as either fraudulent or non-fraudulent. Apart from the transaction amount and time elapsed since the first transaction, all other features are anonymised for confidentiality of payment records. The features have been preprocessed with Principal Component Analysis (PCA), a dimensionality reduction procedure that reduces a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. For every fraudulent transaction present, it is known that there are 588 non-fraudulent transactions, highlighting the presence of a large class imbalance.

2.2 Related Work

2.2.1 Research

While credit card fraud detection is a popular use case for machine learning in practice, research studies that compare and consolidate various supervised learning paradigms for fraud detection are relatively limited. Moreover, many research work on credit card fraud detection methods focus on a single resampling technique to preprocess the data [3]. This study aims to fill the gap by investigating the effectiveness of different resampling techniques as well as supervised learning paradigms in detecting fraudulent transactions, in hope of providing better direction for future research work.

2.2.2 Kaggle Kernels

We note that existing work in Kaggle kernels often do not account for the temporal (time-dependent) nature of the transactions when partitioning the dataset into training, validation and test sets [4]. By partitioning the dataset randomly, the temporal sequence of the transactions is not preserved. It is highly likely that the models built have benefited from data snooping, as traits of the test set could be embedded within the training set. As a result, model performance on the test set may not be representative of the model's actual performance in the real-world forecasting environment.

We also notice that existing work often do not define a suitable performance metric when measuring the success of the models. One example is the ill-suited use of classification accuracy as a performance metric [5]. The use of classification accuracy as a metric for such imbalanced datasets is likely to result in arbitrarily high scores that is not reflective of the model's actual predictive power.

Our work aims to address these limitations in existing work to break new ground. The transactions in the dataset will be arranged in sequential order and partitioned accurately such that the temporal sequence of the transactions is preserved. An appropriate performance metric will also be defined to evaluate model performance.

3 Methodology

3.1 Metric Definition

In this section, we explain why classification accuracy, despite being a popular metric for classification tasks, is not a suitable performance metric for imbalanced datasets. Thereafter, we introduce the F_β score, the main performance metric that we use to assess and tune our models in place of classification accuracy.

3.1.1 Classification Accuracy

	Predicted Positive	Predicted Negative
Actual Positive	*True Positive (TP)*	*False Negative (FN)*
Actual Negative	*False Positive (FP)*	*True Negative (TN)*

Figure 1: Confusion Matrix for General Binary Classification Task

$$Classification\ Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Classification accuracy measures how well a model is able to correctly predict the class of the instances, by finding the ratio of the number of correctly predicted instances (TP+TN) to the total number of instances (TP+TN+FP+FN). In the context of credit card fraud detection, it measures how well a model is able to correctly predict a transaction as fraudulent or non-fraudulent.

	Predicted Positive	Predicted Negative
Actual Positive	*0*	*10*
Actual Negative	*0*	*5880*

Figure 2: Confusion Matrix for Imbalanced Dataset with All Instances Predicted Negative

$$Classification\ Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{5880}{5890} = 99.8\%$$

When the classes are severely imbalanced as illustrated in Figure 2, models with poor predictive power can yield a deceptively high classification accuracy. The dataset we are working with contains 588 non-fraudulent transactions for every fraudulent transaction present. It is thus possible to achieve an arbitrarily high classification accuracy of over 99.8%, by not learning from the data and always predicting the transactions as non-fraudulent. For this reason, we conclude that it is inappropriate to use classification accuracy as a performance metric to evaluate model performance.

We thus propose the use of the F_β score, which considers two important aspects of a classification model, namely sensitivity and precision.

3.1.2 Sensitivity

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

Sensitivity measures how well a model is able to correctly predict positive instances as positive, by finding the ratio of correctly predicted positive instances (TP) to the total number of positive instances (TP+FN). In this context, it measures how well a model is able to correctly predict a fraudulent transaction as fraudulent. In order to maximise true positives and minimise false negatives, we should seek to achieve high sensitivity scores for our models.

3.1.3 Precision

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

Precision measures how accurate a positive prediction is, by finding the ratio of correctly predicted positive instances (TP) to the total number of predicted positive instances (TP+FP). In this context, it measures how accurate the prediction of a fraudulent transaction is. We should also seek to achieve high precision scores for our models, so as to maximise true positives and minimise false positives.

3.1.4 F_β Score

$$F_\beta = \frac{(1 + \beta^2) \times Precision \times Sensitivity}{(\beta^2 \times Precision) + Sensitivity}$$

Instead of taking the simple average of sensitivity and precision, the F_β score finds a weighted harmonic mean of sensitivity and precision based on a predefined β value. The formulation of the F_β score allows for greater emphasis to be placed on sensitivity than precision by pre-defining the value of β to be more than 1.

While the aim is to maximise both sensitivity and precision, we believe it is more important to identify every possible fraudulent transaction, even if it means inadvertently having some false positives. To place greater emphasis on sensitivity than precision and for ease of computation, we define β as 2 when evaluating our models.

3.2 Resampling Techniques

Resampling techniques are employed to balance the number of instances for both classes in the training set. We aim to investigate the feasibility of using the nearest neighbours principle to perform resampling. Thus, resampling techniques which are based on this principle are explored - namely Synthetic Minority Oversampling Technique (SMOTE), which oversamples the minority class, and NearMiss, which undersamples the majority class.

3.2.1 SMOTE

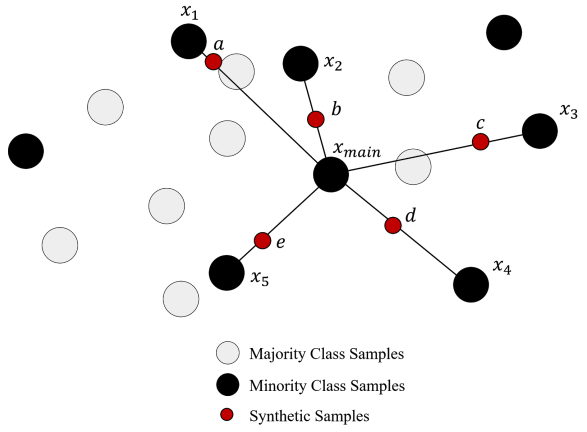

Figure 3: Visualisation for SMOTE

SMOTE is an oversampling algorithm which creates synthetic instances of the minority class via linear interpolation between the nearest minority class instances [6]. As illustrated in Figure 3, an arbitrary line is first drawn between a randomly selected instance of the minority class represented by x_{main}, to each of its five nearest neighbours from the same class, represented by x_i where $i = 1, ..., 5$. A new instance is then randomly generated along each of the lines, represented by a, b, c, d, e.

3.2.2 NearMiss

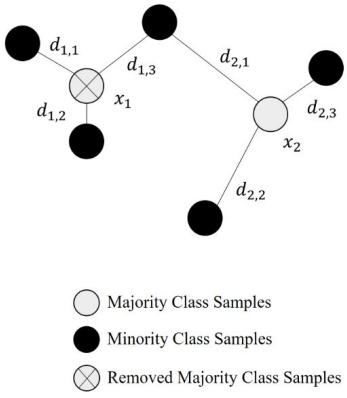

Figure 4: Visualisation for NearMiss

NearMiss is an undersampling method that involves elimination of instances from the majority class which are nearest to instances from the opposite class [7]. For this study, we implement the NearMiss algorithm to remove instances from the majority class with the smallest average distance to three nearest instances from the minority class. Figure 4 illustrates this point, an instance from the majority class, x_1 is deleted instead of x_2, since $\frac{d_{1,1}+d_{1,2}+d_{1,3}}{3} < \frac{d_{2,1}+d_{2,2}+d_{2,3}}{3}$, where $d_{i,j}$ is the Euclidean distance between two instances for $i = 1, 2$ and $1 \leq j \leq 3$.

3.3 Preprocessing

3.3.1 Data Partitioning

Given the temporal nature of the transactions, we paid special attention when partitioning the dataset into training, validation and testing sets. In order to prevent data snooping and to simulate the real world forecasting environment accurately, we have arranged the instances in the dataset in chronological order and included only the earlier sequential instances for training, and the later instances for validation and testing respectively.

3.3.2 Data Normalisation

With 30 numerical features of varying scale and range, all features in the dataset were normalised to adhere to a common distribution of mean zero and unit variance. This data normalisation step was performed to standardise distances between instances, to allow each feature to contribute proportionately to the final distance in classifiers that depend on a distance metric to generate the classification boundary. One such classifier would be the *k* Nearest Neighbours algorithm, which is one of the supervised learning methods that we examine in this study.

3.4 Supervised Learning Algorithms

Supervised learning algorithms that are well-established for binary classification tasks are implemented and studied. These algorithms include Logistic Regression, *k* Nearest Neighbours, Naive Bayes, Decision Tree, Random Forest, Single Layer Perceptron and Multi Layer Perceptron. An overview of these algorithms and their implementations is available under Appendix B.

3.4.1 Hyperparameter Tuning

Hyperparameters of the various models are tuned on the validation data to ensure that the models have better fit. For instance, in the implementation of *k* Nearest Neighbours, the number of neighbours that yields the best performance on validation data is chosen as the final hyperparameter, *k*. Likewise, for Random Forest, the number of decision trees that generates the best performance when tested on validation data is chosen as the final hyperparameter.

3.4.2 L2 Regularisation

L2 regularisation, which penalises excessively large weights [8], is also performed to control for overfitting. The Multi Layer Perceptron, with high expressive power, is likely to overfit the training data. With that in mind, we applied L2 regularisation when training the Multi Layer Perceptron model and found that applying an L2 penalty parameter of 0.0002 was able to generate the best performance on the validation set.

3.5 Model Stacking

3.5.1 Overview

To explore if the merits of individual supervised learning algorithms can be amalgamated, we attempted model stacking, a model ensembling method employed in competitive machine learning. Model stacking hinges on the intuition that different models may perform well for different regions in the multidimensional space of features. By learning these regions, it is

possible to determine which model's prediction should be retained under particular circumstances [9].

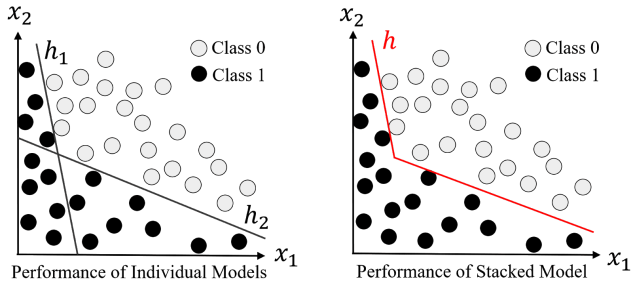

Figure 5: Naive Visualisation for Stacked Model Performance

In Figure 5, we have a naive example of a dataset with two features x_1 and x_2, with class labels 0 and 1. On the left, we observe the respective decision boundaries formed by the hypotheses of two individual models, h_1 and h_2. As depicted, h_1 performs well at classifying instances with high x_2 values, but does poorly for instances with high x_1 values. The converse is true for h_2. By feeding the predictions of individual models (base learners) along with a subset of input features, as input into another individual model (meta learner), the aforementioned trait can be learnt. This produces a stacked model h that merges the best traits of the base learners, resulting in better performance, which is illustrated on the right in Figure 5.

3.5.2 Algorithm

In this section, we elaborate on the algorithm employed to perform model stacking with two base learners.

$$X_{train} = X_{s\_train} + X_{s\_val}$$

The training set X_{train} is split into X_{s_train} and X_{s_val}, to segregate the training data used to train the base learners and the validation data upon which the base learners generate predictions, which are then used as input into the meta learner. This helps to ensure that the base learner predictions provided to the meta learner are reflective of the base learners' performance on unseen data. To balance the tradeoff between having more accurate base learner predictions to train the meta learner and having more training data for the meta learner, a 0.5 split was performed on X_{train}.

$$X_{s\_train} \begin{cases} \text{Base Learner 1} \Longrightarrow h_1 \\ \text{Base Learner 2} \Longrightarrow h_2 \end{cases}$$

The base learners are first separately trained on X_{s_train} to obtain two initial hypotheses h_1 and h_2.

$$X_{s\_val} \begin{cases} h_1 \xrightarrow{h_1(X_{s\_val})} \\ \\ h_2 \xrightarrow{h_2(X_{s\_val})} \end{cases} \text{Meta Learner} \Longrightarrow h_3^*$$

Predictions of the trained base learners on X_{s_val} are obtained and stacked with the original inputs $(h_1(X_{s\_val}) + h_2(X_{s\_val}) + X_{s\_val})$. The stacked set contains $n+2$ features, where n is the number of features in the original dataset. The stacked set is then utilised to train the meta learner.

It should be noted that in conventional implementations of model stacking, it is only crucial for the meta learner to take in predictions of the base learners, and optionally a subset of original input features. The original input features are fed into the meta learner to aid in identification of the region that an input instance resides in. However, to prevent the meta learner from learning only on the original input features and disregarding the base learner predictions, the full set of original input features is often not used.

Given that our dataset has undergone a PCA transformation and is anonymised, it is challenging for us to identify an appropriate subset of features to stack with the base learner predictions. Moreover, as we are performing classification instead of regression, the predictions of the base learners alone are insufficient to inform the metamodel of the region that an input instance resides in. Therefore, in our implementation, we stack all the original input features with the base learner predictions.

$$X_{train} \begin{cases} \text{Base Learner 1} \Longrightarrow h_1^* \\ \text{Base Learner 2} \Longrightarrow h_2^* \end{cases}$$

With the meta learner trained, X_{s_val} no longer needs to be hidden from the base learners. The base learners are retrained on the full training set X_{train} to reduce overfitting and improve performance.

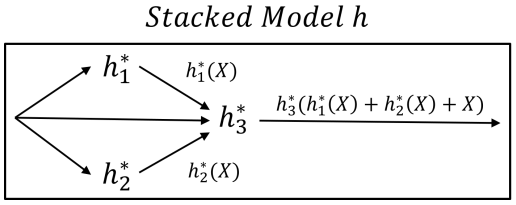

Figure 6: Stacked Model

Finally, combining the retrained base learners with the meta learner, we produce our stacked model as illustrated in Figure 6.

3.5.3 Implementation

We adopted a round-robin approach when experimenting with model stacking instead of a single fixed implementation, to observe the performance of different stacked models with different base and meta learners. In particular, we experimented with the top three performing individual models, with two models chosen as base learners and the other positioned as a meta learner.

4 Results Analysis

4.1 Results

The results obtained for the implemented supervised learning methods are illustrated as follows. A more detailed tabulation of results is available under Appendix A.

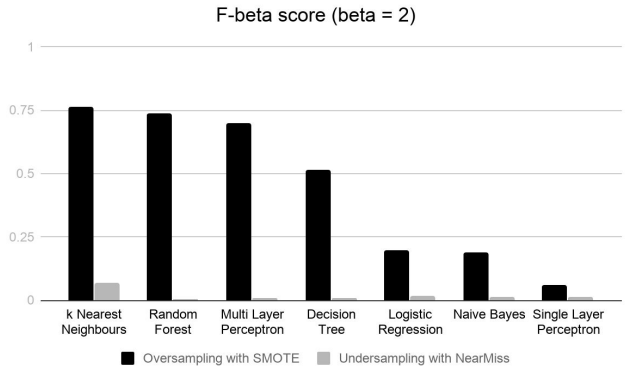

Figure 7: Comparison of F_β Scores Obtained

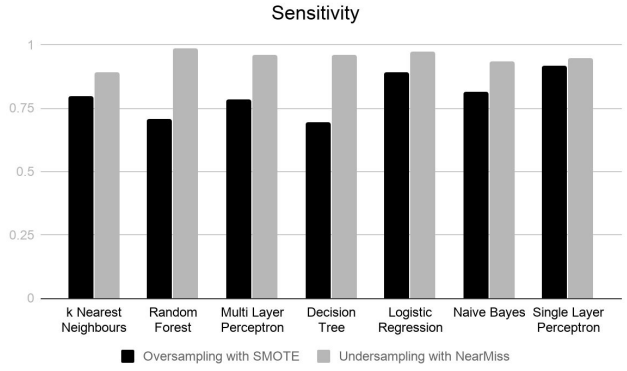

Figure 8: Comparison of Sensitivity Scores Obtained

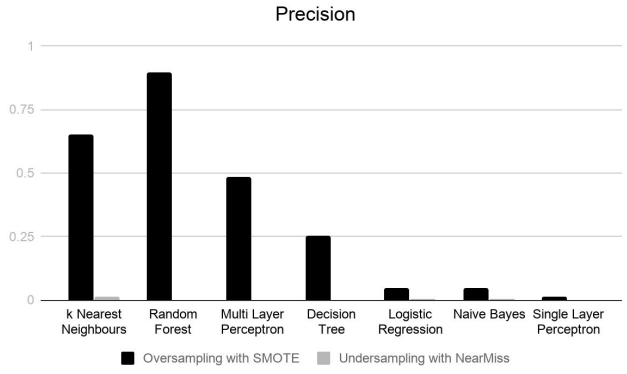

Figure 9: Comparison of Precision Scores Obtained

4.2 Discussion

4.2.1 Sampling Techniques

When comparing results obtained from oversampling the minority class and that from undersampling the majority class, we observe that models yield higher precision and F_β scores when trained on the oversampled data. However, models tend to yield higher sensitivity scores on undersampled data.

The higher F_β scores on oversampled data support our hypothesis that having more data enhances learning. SMOTE's method of generating synthetic instances between existing instances of the minority class allows models to better recognise the range of traits that fraudulent transactions typically possess, resulting in higher precision and F_β scores. On the other hand, undersampling with NearMiss reduces the amount of majority class instances dramatically. With fewer majority class instances, it is likely that the models learn the specifics of this smaller set of majority class instances, and thus would overfit. The poor generalisation on unseen test data results in many non-fraudulent transactions being incorrectly predicted as fraudulent, which translates to low precision scores and consequently low F_β scores.

Higher sensitivity scores on undersampled data is likely attributed to the retention of all information about the minority class during undersampling. With all information about the minority class retained and less information about the majority class, the models are likely to perform better at predicting the minority class, which translates to high sensitivity.

4.2.2 Supervised Learning Algorithms

When comparing the results obtained on oversampled data for the different models, we observe that *k* Nearest Neighbours yields the highest F_β score of 0.7653 out of all the algorithms implemented.

Despite being relatively simple, k Nearest Neighbours outperforms other algorithms. This is likely attributed to fraudulent transactions sharing similar traits, as fraudsters are likely to follow a modus operandi. This results in the instances forming clusters in multidimensional space. Utilising the nearest neighbours principle to make distance-based predictions is thus likely to be effective.

The good performance of k Nearest Neighbours can also be attributed to oversampling with SMOTE. Oversampling with SMOTE modifies the Euclidean distance between test samples and the minority class. The test samples are on average closer to the synthetic samples than to the original samples from the minority class, and hence *k* Nearest Neighbours is likely to be able to better differentiate the classes in the test set and perform better.

Moreover, with over 200000 instances and only 30 features, it is unlikely that *k* Nearest Neighbours will suffer from the curse of dimensionality.

We observe that Random Forest and Multi Layer Perceptron yield decent F_β scores on oversampled data. The moderate performance is likely due to the models being more complex and expressive, and hence being able to better approximate the target function and yield good predictions on the test data.

We also note a leap in F_β score from Decision Tree to Random Forest. This is an indication that bagging, which involves averaging multiple hypotheses, is effective in improving model performance. This is consistent with our knowledge that averaging multiple hypotheses in ensemble learning can lower the variance of the learned hypothesis, in turn producing a model that generalises better on unseen test data.

Finally, we notice that Logistic Regression, Naive Bayes and Single Layer Perceptron yield low F_β scores. The performance of these methods hinge on assumptions of linear separability and conditional independence. The poor performance could indicate that these assumptions are false. Furthermore, the dataset is likely to be too complex and the simple hypotheses of these models are unable to capture the target function well.

4.2.3 Stacked Models

We observe an improvement in F_β scores for stacked models as compared to individual models (Appendix A). A closer look at sensitivity and precision scores reveals consistently high sensitivity and precision scores of greater than 70% for all three stacked models, suggesting an improved balance of sensitivity and precision.

However, the F_β scores achieved by the stacked models exceed the best performing individual model, *k* Nearest Neighbours, by no more than 1%. We attribute this observation to two possible causes. Firstly, the meta learner receives a stacked set of original input features and base learner predictions, however, it is unable to place sufficient weight on the base learner predictions and fails to leverage on their strengths. This effect counteracts the benefits of model stacking, as the meta learner receives only half the number of instances it would receive if model stacking was not performed. Secondly, our base learner hypotheses could be lacking diversity, thus incorrectly identifying mostly the same instances. As such, stacking would yield only a small improvement, as there is little individual strengths for the meta learner to learn from.

While more research has to be conducted to fully explore the potential of stacking, the improved performance is an indication that model stacking is a viable enhancement to existing supervised learning methods for credit card fraud detection.

5 Limitations and Future Work

5.1 Validation

Due to lack of computational resources and time, we have validated the models based on hold-out validation instead of cross-validation. Ideally, we would like to apply *k*-fold cross validation, which averages the hold-out validation error obtained across the *k* folds, to have a better estimate of how the models would perform on unseen test data. Our group would also like to highlight that for imbalanced temporal datasets, resampling techniques should be applied to the reduced training sets for each nested cross-validation iteration, so as to prevent data snooping.

5.2 Feature Engineering

Due to anonymity of features, we did not carry out feature engineering for this comparative study. Engineering new features could potentially improve model performance, but without knowledge of these attributes, it is challenging to engineer meaningful features. Should these attributes be made known, future work on the dataset could consider engineering new features, for example, an aggregated feature based on time of transaction, transaction amount and transaction type [10].

5.3 Resampling Techniques

By randomly selecting minority class instances to generate synthetic data, SMOTE assumes uniform importance across all minority class instances [11]. However, some instances within the minority class may pose greater learning difficulty to classifiers than others. Adaptive Synthesis Sampling (ADASYN), which adaptively generates minority class instances according to their distribution, is more selective and can be considered in future work to yield an enhanced resampled dataset [12].

5.4 Deep Learning Methods

Due to the lack of computational resources and time, deep learning methods were not extensively explored in this comparative study. With advancement in deep learning algorithms for temporal datasets, future work on credit card fraud detection could consider implementing Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM) units to achieve enhanced performance. RNN with LSTM units is likely to give favourable performance as the model takes into account long and short-term past transactions, and would be able to leverage on the fact that fraudulent transactions tend to occur in sporadic bursts.

5.5 Model Stacking

Due to time constraints, model stacking was only briefly explored with the top three performing individual models. Ideally, all models should be considered, as weak models could possibly offer diversity that the meta learner might be able to exploit. Other means of introducing diversity, such as increasing the number of base learners and randomly exposing a subset of features to the base learners, can also be considered for future work. Most importantly, as covered in results analysis, the number of input features fed into the meta model should be fine-tuned.

6 Conclusion

Through this comparative study, we identify that oversampling the minority class yields better results than undersampling the majority class. With more data to train on, supervised learning algorithms have more information to learn from and can better estimate the target function.

In terms of model performance, we observe *k* Nearest Neighbours surpassing all other individual supervised learning algorithms. The exceptional performance suggests that leveraging on the clustering of fraudulent transactions in multidimensional space and distance based models can lead to better performance.

In addition, we observe that model stacking is a viable enhancement for credit card fraud detection. Learning from a stacked set of original input features and base learner predictions helps a stacked model to amalgamate the merits of individual supervised learning algorithms and achieve improved performance.

The above results suggest that oversampling, *k* Nearest Neighbours and model stacking are methods that credit card companies and researchers could look into, to further innovate resampling and supervised learning methods used for credit card fraud detection.

Furthermore, the workflow used in this study of defining a performance metric, performing resampling techniques to overcome class imbalance, experimenting and optimising various supervised learning methods, can be applied to other datasets that are also highly imbalanced, such as detection of rare diseases and customer attrition.

7 Acknowledgements

We would like to thank Professor *Kan Min-Yen* and the teaching staff of CS3244 Machine Learning at the National University of Singapore for their invaluable help and advice.

We would like to acknowledge the *scikit-learn* and *imblearn* packages in Python, which we have used for the implementation of models and resampling methods discussed in this study.

We would like to credit *Ben Gorman* for his article on *A Kaggler's Guide to Model Stacking in Practice*, which provided us with knowledge on the model stacking paradigm as well as inspiration to develop associated illustrations presented in this report.

We would also like to credit *Aleksey Bilogur* for his kernel on *Oversampling with SMOTE and ADASYN*, and *Indresh Bhattacharya* for her article on *SMOTE and ADASYN (Handling Imbalanced Data Set)*, which provided us with insights to resampling techniques as well as the inspiration to develop corresponding illustrations presented in this report.

8 References

All references are arranged in chronological order as they appear in this report.

[1] The Nilson Report. (2015, August 4). *Global Card Fraud Losses Reach $16.31 Billion — Will Exceed $35 Billion in 2020 According to The Nilson Report*. Retrieved from Businesswire: https://www.businesswire.com/news/home/20150804007054/en/Global-Card-Fraud-Losses-Reach-16.31-Billion

[2] Machine Learning Group - ULB. (2018). *Credit Card Fraud Detection*. Retrieved from Kaggle: https://www.kaggle.com/mlg-ulb/creditcardfraud

[3] Mary Frances Zeager, A. S. (2017 (17)). *Adversarial learning in credit card fraud detection*. Systems and Information Engineering Design Symposium (SIEDS), 112-116.

[4] Currie32. (2016). *Predicting Fraud with TensorFlow*. Retrieved from Kaggle: https://www.kaggle.com/currie32/predicting-fraud-with-tensorflow

[5] Rathee, A. (2016). *Achieving 100% accuracy*. Retrieved from Kaggle: https://www.kaggle.com/arathee2/achieving-100-accuracy

[6] Nitesh V. Chawla, K. W. (2002). *SMOTE: Synthetic Minority Over-sampling Technique*. Retrieved from Journal of Artificial Intelligence Research: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume16/chawla02a-html/chawla2002.html

[7] Show-Jane Yen, Y.-S. L. (2006). *Under-Sampling Approaches for Improving Prediction*. ICIC, 731-740.

[8] Corinna Cortes, M. M. (2009). *L2 Regularization for Learning Kernels*. UAI, 109-116.

[9] Güneş, F. (2017, May 18). *Why do stacked ensemble models win data science competitions?* Retrieved from SAS Blog: https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions/

[10] Alejandro Correa Bahnsen, D. A. (2016 (51)). *Feature engineering strategies for credit card fraud detection*. Expert Systems With Applications, 134-142.

[11] Przemyslaw Skryjomski (2017). *Influence of minority class instance types on SMOTE.* Proceedings of Machine Learning Research, 74:7-21.

[12] Haibo He, Y. B. (2008). *ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced*. IEEE Xplore, 1322-1328.

9 Appendix

Appendix A: Model Performance

Method	**Sensitivity**	**Precision**	F_β ***(β=2)***
Logistic Regression	0.8933	0.0474	0.1956
k Nearest Neighbours	0.8	0.6522	0.7653
Naive Bayes	0.8133	0.0461	0.1878
Decision Tree	0.6933	0.2549	0.5159
Random Forest	0.7067	0.8983	0.7382
Single Layer Perceptron	0.92	0.0130	0.0614
Multi Layer Perceptron	0.7867	0.4836	0.6991

Table 1: Oversampling using SMOTE

Method	**Sensitivity**	**Precision**	F_β ***(β=2)***
Logistic Regression	0.9733	0.0036	0.0176
k Nearest Neighbours	0.8933	0.0151	0.0709
Naive Bayes	0.9333	0.0026	0.0112
Decision Tree	0.96	0.0016	0.0081
Random Forest	0.9867	0.0013	0.0066

Single Layer Perceptron	0.9467	0.0023	0.0113
Multi Layer Perceptron	0.96	0.0022	0.0107

Table 2: Undersampling using NearMiss

Base Learners	**Sensitivity**	**Precision**	F_β ***(β=2)***
k Nearest Neighbours and Random Forest	0.76	0.7917	0.7661
k Nearest Neighbours and Multi Layer Perceptron	0.7467	0.9032	0.7735
Random Forest and Multi Layer Perceptron	0.7867	0.7195	0.7723

Table 3: Comparison of F_β Scores Obtained for Stacked Models

Appendix B: Overview of Supervised Learning Methods

Logistic Regression

Logistic Regression is a probabilistic-based supervised learning algorithm that computes the probability of an instance belonging to the positive class using the sigmoid activation function. For a binary classification task like fraud detection, there is a standard decision threshold of 0.5. That is, if the probability of an instance belonging to the positive class exceeds the threshold of 0.5, that instance will be assigned to the positive class. Otherwise, it will be assigned to the negative class.

k Nearest Neighbours

k Nearest Neighbours is a non-parametric supervised learning algorithm that classifies an instance by the majority vote of its nearest neighbours based on a chosen distance metric. For this study, features are numerical and continuous, hence the Euclidean distance metric is used to compute the distance between instances.

Naive Bayes

Naive Bayes is a simple Bayesian method that makes use of the Bayes theorem when classifying the instances. It estimates the posterior probabilities from the prior probabilities, marginal probabilities and likelihoods. It also naively assumes that the features are conditionally independent of each other.

Decision Tree and Random Forest

Decision Tree is a tree-based model which uses entropy and information gain to identify the best features and split the training data into ideal subsets. Due to the expressiveness of tree-based models, Decision Tree often suffers from high variance and is likely to overfit the data. Random Forest, which utilises feature projection and averages over the multiple hypotheses of several Decision Trees, is implemented to overcome the problem of high variance and to control for overfitting.

Single Layer Perceptron

Single Layer Perceptron, or Perceptron Learning Algorithm, is an iterative learning algorithm that learns linear relationships in the data. In each iteration, a misclassified instance is selected, and the weight is updated to correct for that particular misclassified instance. The weight update typically terminates once the linear separating hyperplane is found.

Multi Layer Perceptron

Multi Layer Perceptron is a type of feedforward neural network that is composed of more than one perceptron, and consists of at least three layers of nodes - an input layer, the hidden layer and an output layer. Each node in the hidden layer typically uses a nonlinear activation function, allowing it to learn non-linear relationships in the data. The architecture of the Multi Layer Perceptron that is used in this study consists of three hidden layers with 20, 20 and 5 nodes respectively. Stochastic gradient descent is used to minimise the training cost.

Appendix C: Source Codes

All codes that are used in this study are available in a Jupyter Notebook via the following link:

https://colab.research.google.com/drive/1SIjfoiI_Fe25Ssp_13VXp_7goBTBxGNf