HarvardX PH125.9x Capstone Project:CYO Credit Card Fraud

David Mc Manus

11/20/2022

Contents

Introduction

In their Seventh Report on Card Fraud, the European Central Bank estimated that "the total value of fraudulent transactions using cards issued within SEPA and acquired worldwide amounted to €1.87 billion in 2019." [1]. While only comprising 0.036% of the total value of transactions, or 36 cents in every €100, fraud has real world implications for ordinary people. Whilst the effects on a victim of fraud may range from financial to mental and emotional, the larger society also bears the cost, through fees and other related charges.

As banks and other card providers strive to detect and reduce the amount of fraudulent transactions, another issue must be given consideration, particularly in the post covid push to online banking and contactless payments, the effects of false positives. If a customer is repeatedly having issues paying online or has their card denied too often, they will change their shopping behaviour, or simply resort to using physical money.

To this end, fraud detection systems need to not only be able to detect case of fraud, but also need to keep the number of false positive cases to a minimum.

Aim

Similarly to the IEEE-CIS Fraud Detection [2] competition in 2019, this project will evaluate detection systems based on their AUC-ROC scores.

GLM, Decision Trees, Random Forrest,K Nearest Neighbours, and Keras models will be evaluated, before the model with the highest AUC-ROC score will be tested with a final validation data set.

What is a AUC-ROC score?

AUC stands for Area Under the Curve, and, while there are more variants, we will be focusing on the ROC, which stands for Receiver Operating Characteristics curve. The AUC-ROC is an important evaluation metrics for classification model. [3] To understand the metric, you need to have a basic understanding of a ROC curve, as a AUC-ROC score is essentially a summary of it. [4].

A ROC curve plots the True Positive Rate (y-axis) against the False Positive Rate (x-axis) at a number of different threshold values, and the area under this curve is called the AUC.

Ranging from 0 to 1, an AUC score of 0.5 would mean that a system cannot discern between the classes and a score of 1 meaning the system can predict the class perfectly. Conversely, a score of 0 would mean that the system would predict every positive as negative and vice-versa.

Although there is no definitive "good" AUC score, we could apply Hosmer and Lemeshow's rule of thumb [5] and consider any score above 0.9 to be outstanding discrimination.

Installing and loading Packages

When using the keras package, it is advised to install it on fresh session to avoid any issues. So that's what we'll do first.

Dataset

First published by Pozzolo et al. [6] the data set contains over 280,000 transaction made by European card holders in September 2013. The data has undergone PCA transformation, leaving only numeric variables, while, due to confidentiality issues, no background information or original features of the data is available.

Each entry contains 31 features, V1 through V28, as well as three untouched by the PCA transformation: Time (seconds elapsed between a given transaction and the first in the dataset), Amount, and Class (where 1 represents a case of fraud, and a 0 for legitimate transaction). [7]

Of the 284,807 transactions, only 492 are classed as fraud, accounting for 0.172% of the dataset. This extreme imbalance will need to be addressed, as severely imbalanced datasets can cause serious issues, up to and including algorithms ignoring the minority class entirely. [8]

The dataset can be found here.

Analysing the Data

Dataset Structure

First, lets have a look at the dataset.

Table 1: Dataset Head

Time	V1	V2	V3	V4	V5	V6	V7	V8	V
0	-1.3598071	-0.0727812	2.5363467	1.3781552	-0.3383208	0.4623878	0.2395986	0.0986979	0.36378
0	1.1918571	0.2661507	0.1664801	0.4481541	0.0600176	-0.0823608	-0.0788030	0.0851017	-0.25542
1	-1.3583541	-1.3401631	1.7732093	0.3797796	-0.5031981	1.8004994	0.7914610	0.2476758	-1.51465
1	-0.9662717	-0.1852260	1.7929933	-0.8632913	-0.0103089	1.2472032	0.2376089	0.3774359	-1.38702
2	-1.1582331	0.8777368	1.5487178	0.4030339	-0.4071934	0.0959215	0.5929407	-0.2705327	0.81773
2	-0.4259659	0.9605230	1.1411093	-0.1682521	0.4209869	-0.0297276	0.4762009	0.2603143	-0.56867

Table 2: Dataset Summary

Any NA's
FALSE

As mentioned above, virtually all the features of the data set have under gone transformation, for anonymisation purposes, and are simply numeric values with no other information provided. The data set has no missing data.

Rate of Fraud

Lets look at the amount of fraud in the data set.

Table 3: Number of Real and Fraudulent Transactions

Var1	Freq
'0'	284315
'1'	492

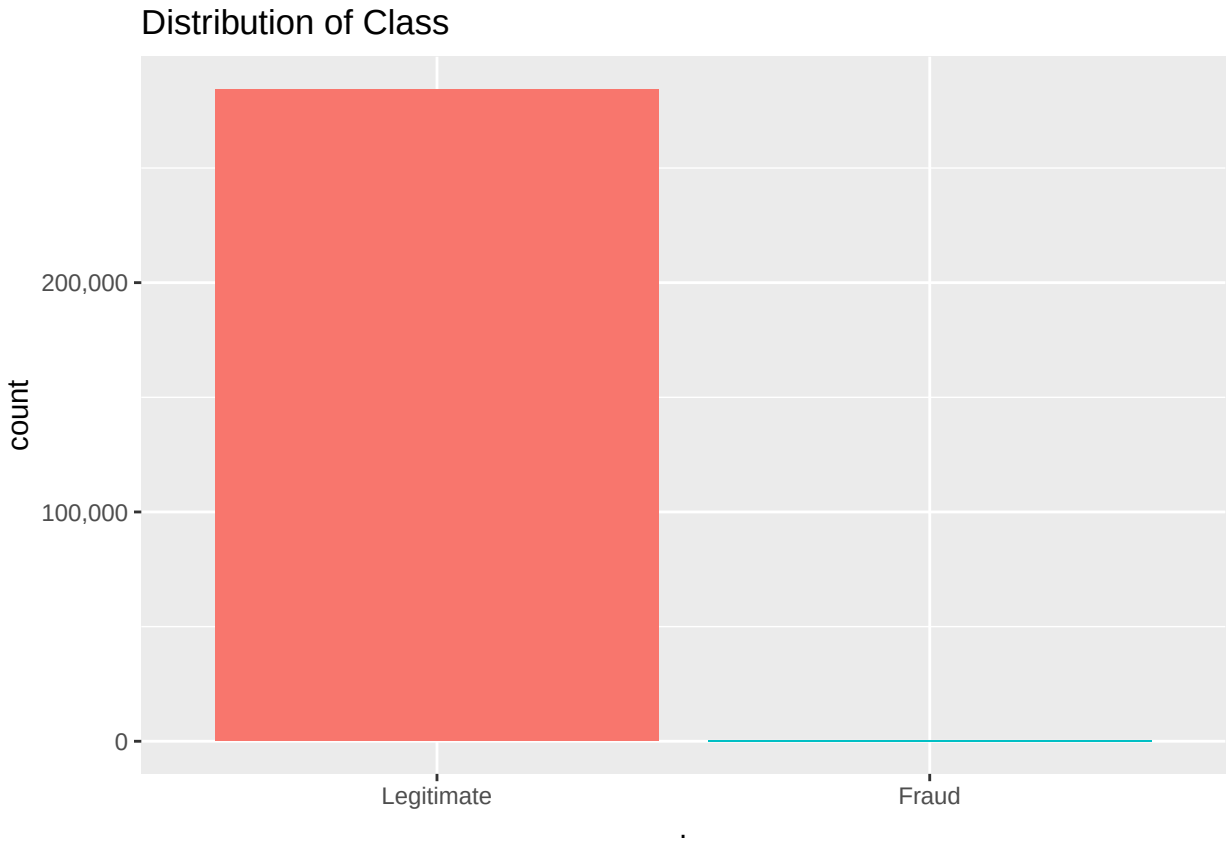
Note:
0 = Legitimate, 1 = Fraud

The set is extremely unbalanced, with less than 0.002% being labeled as fraud. To fully illustrate this imbalance, let's look at simple bar chart to get a better understanding.

Table 4: Frequency of Real and Fraudulent Transactions

Var1	Freq
'0'	0.9982725
'1'	0.0017275

Note:
0 = Legitimate, 1 = Fraud



The scale of the imbalance is clearly on display. Compared to legitimate transactions, the amount of fraudulent transactions is barely more than a thin line. As mentioned above, this can lead to models ignoring the minority class.

Before building our models, we will use the ROSE package to mitigate this risk.

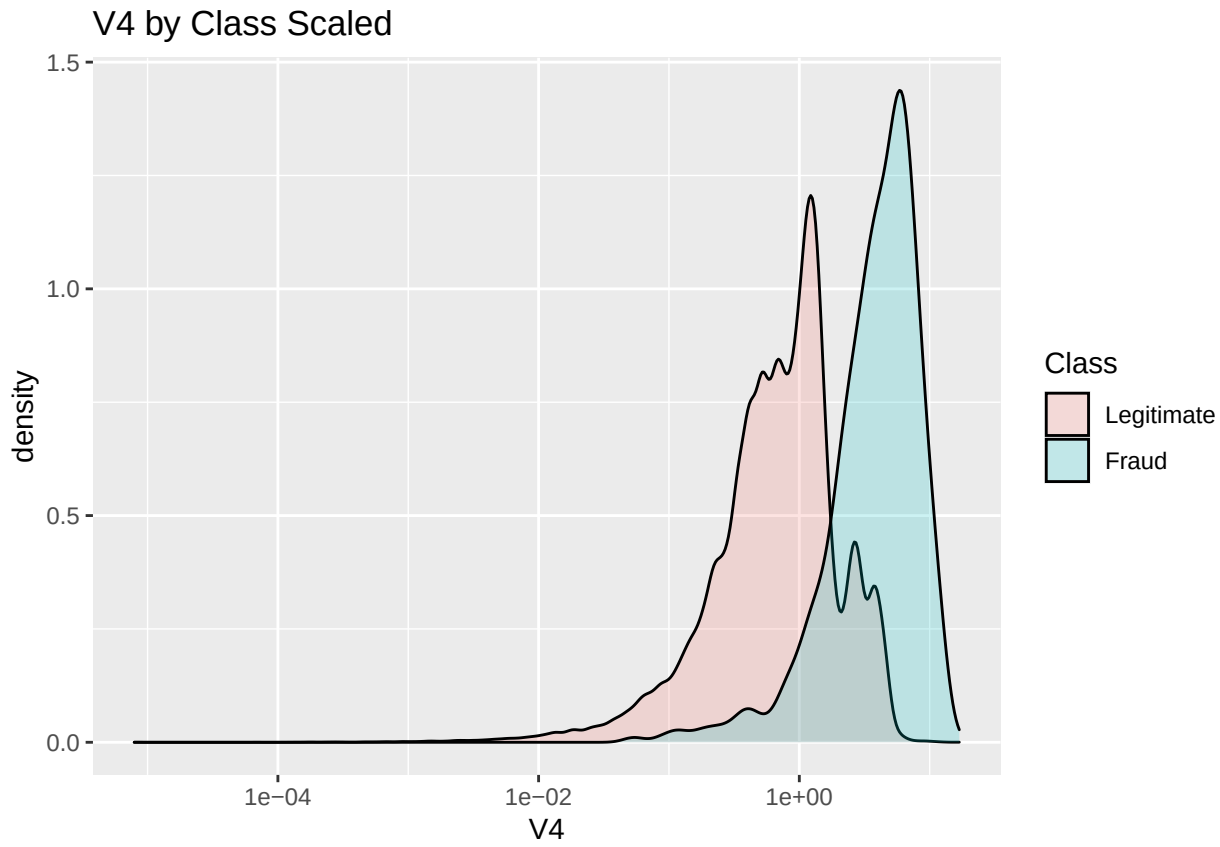
Correlations

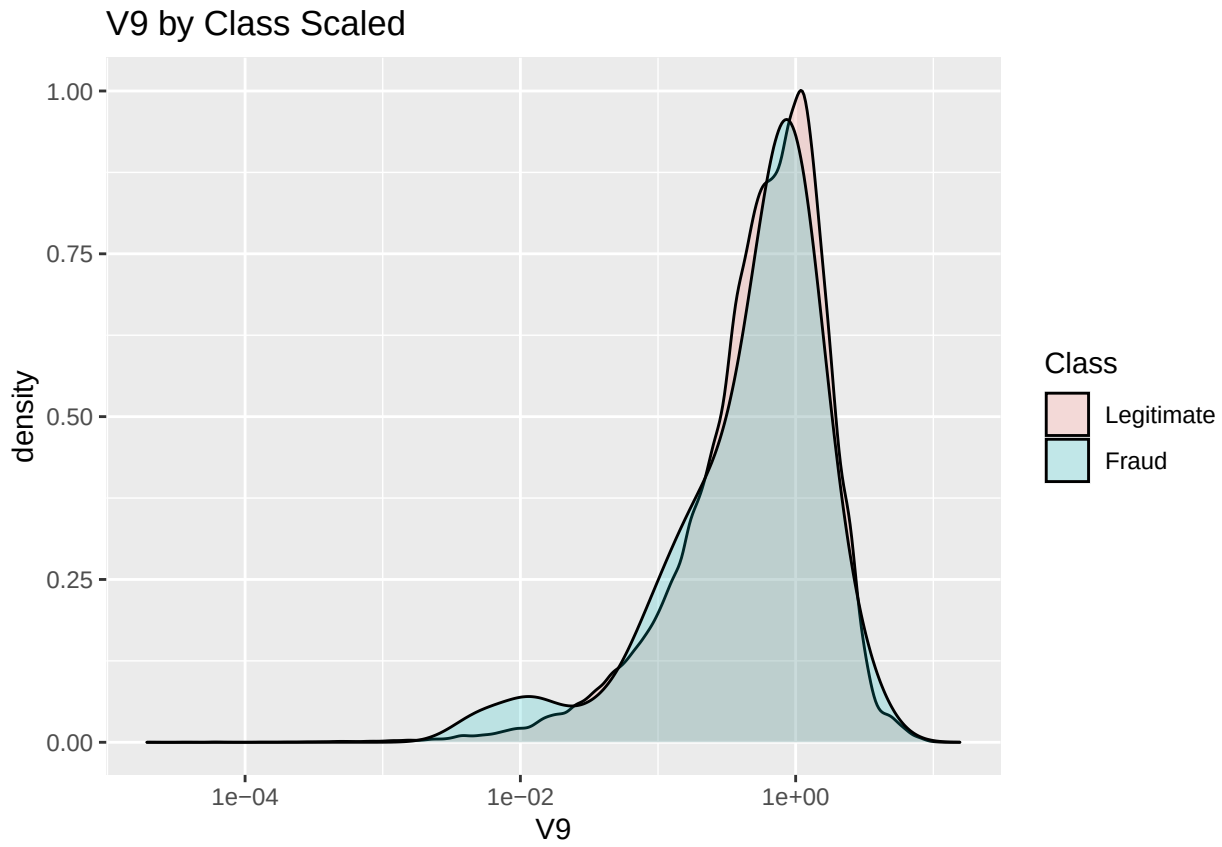
Before delving into the detection systems, it may be worthwhile to visualize some of the features' relationship to fraudulent and legitimate transactions. Apart from Class, the other two features we know are Time and Amount. As we know, Time represents the seconds elapsed between the first transaction and any given transaction, so we shall disregard that variable.

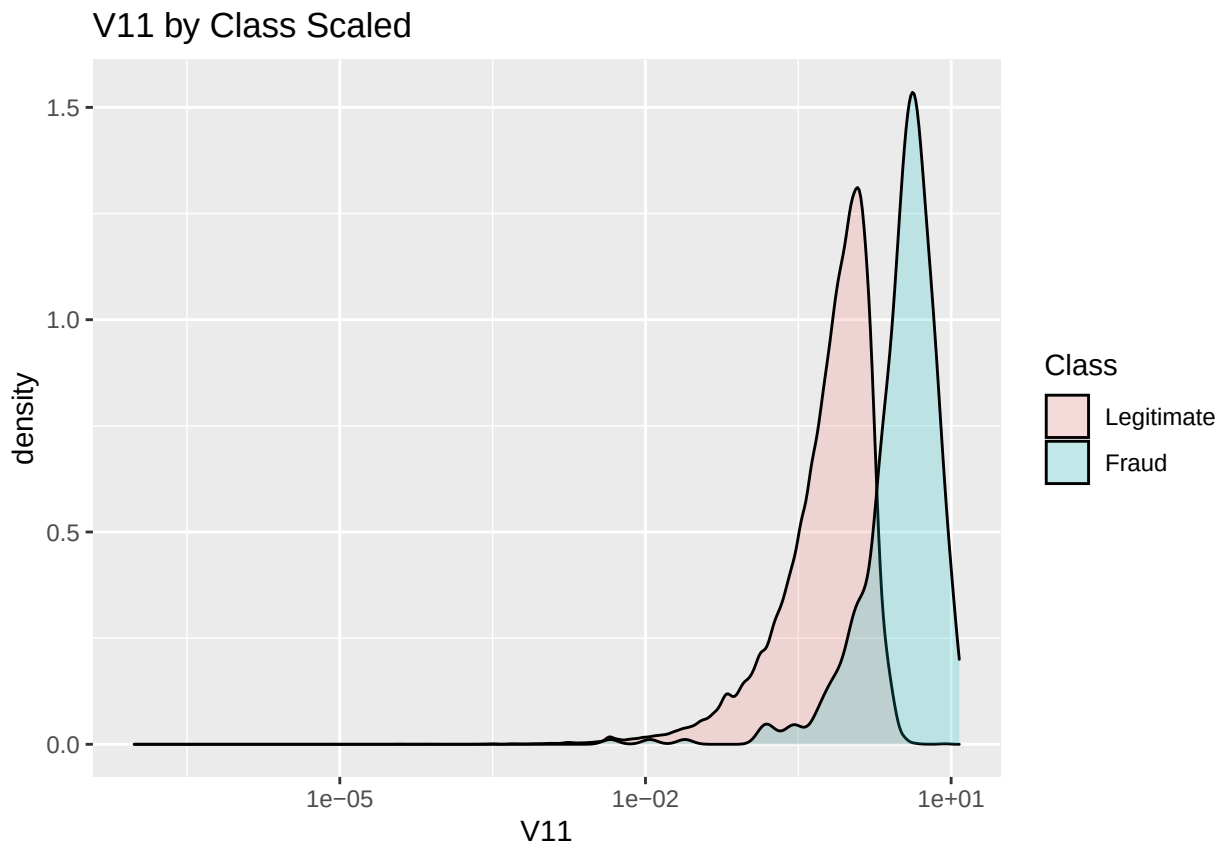
For the sake of brevity, we will also only look at a small sample of the remaining features. This sample will be chosen to best illustrate all of the features.

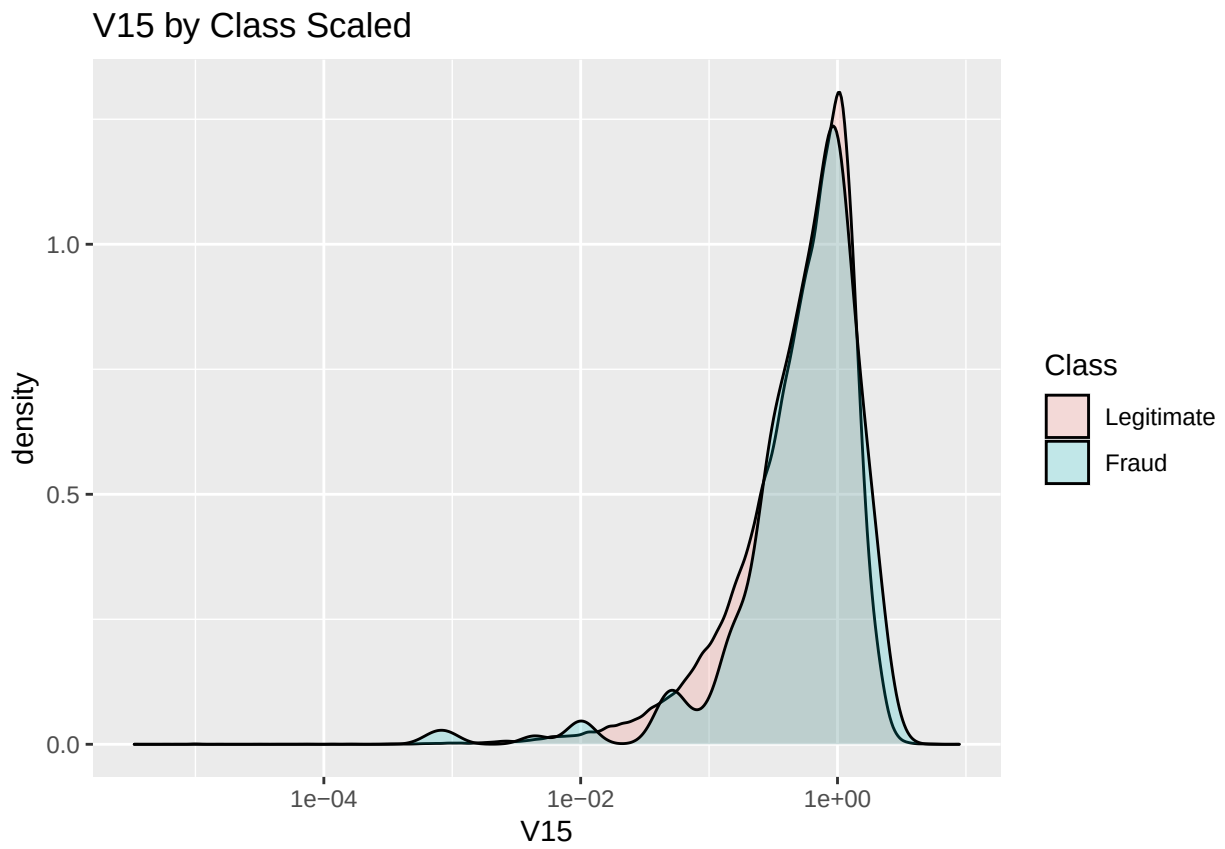
It appears that fraud tends to favour lower and higher amounts, whereas the legitimate transactions are more evenly spread. This may indicate a deliberate system of testing if the transaction will be processed, using a small amount, then trying to maximize their return with a large amount. This, however, is purely speculation, albeit based on a known trend. [9]

Lets look at 4 other features.


V4 by Class Scaled
1.5
1.0
0.5
0.0
density
1e−04
1e−02
1e+00
V4
Class
Legitimate
Fraud


V9 by Class Scaled
1.00
0.75
0.50
0.25
0.00
density
1e−04
1e−02
1e+00
V9
Class
Legitimate
Fraud


V11 by Class Scaled
1.5
1.0
0.5
0.0
density
1e−05
1e−02
1e+01
V11
Class
Legitimate
Fraud



Where as V9 and V15 are almost perfectly matching, and V4 and V11 display a higher degree of distinction, all the remaining features fall in-between the two extremes.

Conclusion

The data set is clean, with a single redundant feature, Time.

The data set is extremely unbalanced, which if left unaddressed, may cause problems when training our models.

Although the Amount by Class Scaled plot appears to support the idea of fraudsters first attempting a small transaction before making a large one, due to the significant overlap between the two classes, without any additional data identifying the card used, this information will be unlikely to be useful in designing our models.

This is also true of the other variables, V1-V28, when taken individually, without additional information, there is a prohibitive overlap between legitimate and fraudulent transactions.

For that reason, we must rely on approaches that can accommodate this ambiguity and lack of additional information.

Creating Training, Test, and Validation sets

Before moving on to building and testing our models, there is an issue which needs to be addressed first. As mentioned before, the data set is extremely unbalanced, which can cause serious issue during training.

However, in order to help preserve real world validity, both the validation and test sets will remain unbalanced.

We will keep 20% of the entire data set for Validation, and 20% of the remaining entries for our test set. After seperating the test and validation sets, we will then use the ROSE package on the remaining entries to re-balance our training set.

ROSE

In order to rebalance our training set, we can use one of two common approaches, Random Oversampling and Random Undersampling. Simply put, undersampling involves randomly removing instances of the majority class, where as oversampling increases the amount of the minority class in the set. This can be achieved in two ways, by duplicating existing instances, or by creating new, synthetic entries.

In this project, we will use the ROSE (Random Over Sampling Examples) package to address this issue. The ROSE package is specifically designed to resolve binary classification imbalances, and generates synthetic samples to achieve this.

More information on ROSE may be found here

Lets look at the amount and percentage of fraud in the data.rose set.

Table 5: Number of Real and Fraudulent Transactions

Var1	Freq
'0'	91162
'1'	91113

Table 6: Frequency of Real and Fraudulent Transactions

Var1	Freq
'0'	0.5001344
'1'	0.4998656

The data.rose set is now balanced, and we have mitigated the risk that the models may simply ignore cases of fraud during training.

Changing levels

For the sake of later ease, we should change the levels of class. To make our tables a bit easier to read, instead of 0 and 1, we will change them to "Good" and "Fraud".

At the same time, and more importantly, as we want to detect fraud, we can relevel them so our sensitivity and specificity scores are accurate.

Detection systems

In Reproducible Machine Learning for Credit Card Fraud Detection - Practical Handbook; Le Borgne, Siblini Lebichot, and Bontempi [10] use decision tree, linear regression, random forest, XGBoost systems, before moving on to deep learning approaches. In this project, we will use a similar approach, first we evaluate a GLM, a Decision Tree, a Random Forest, and a KNN3 system. We will then look at a Keras Sequential Model approach.

Setting TrainControl

As we will be utilizing the caret package in some of our models, one of it's major benefits is the ability to change the model with ease, we will first set a traincontrol.

glm

The first system we will test is a Generalized Linear Model. After fitting, model will be used to predict the Class for the test set.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud  Good
##      Fraud    64   469
##      Good     15 45022
##
##                Accuracy : 0.9894
##                  95% CI : (0.9884, 0.9903)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2068
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.810127
##             Specificity : 0.989690
##          Pos Pred Value : 0.120075
##          Neg Pred Value : 0.999667
##              Prevalence : 0.001734
##          Detection Rate : 0.001404
##    Detection Prevalence : 0.011696
##       Balanced Accuracy : 0.899908
##
##        'Positive' Class : Fraud
##
```

Method	AUC	Sensitivity	Specificity
glm	0.9767752	0.8101266	0.9896903

The GLM method gives an AUC score of 0.977. Although there is no "rules", this is a very good score to start with. The model resulted in 15 cases of fraud going undetected, or false negatives, and 469 false positives, or legitimate transactions being flagged as fraud.

Decision Tree

Next we will try a Decision Tree approach, widely used in classification problems [11] For this we will use the rpart function in the caret package.

This can be achieved by simply changing our model's method from "glm" to "rpart".

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud  Good
##      Fraud    63   949
##      Good     16 44542
##
##                Accuracy : 0.9788
##                  95% CI : (0.9775, 0.9801)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1126
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.797468
##             Specificity : 0.979139
##          Pos Pred Value : 0.062253
##          Neg Pred Value : 0.999641
##              Prevalence : 0.001734
##          Detection Rate : 0.001382
##    Detection Prevalence : 0.022208
##       Balanced Accuracy : 0.888304
##
##        'Positive' Class : Fraud
##
```

Method	AUC	Sensitivity	Specificity
glm	0.9767752	0.8101266	0.9896903
Decision Tree - rpart	0.8882948	0.7974684	0.9791387

Unfortunately, using rpart has resulted in an AUC score of 0.888, and the GLM remain the best performing model. The rpart model has produced 16 false negatives and 949 false positives, one more false negative than the glm model, but over double the false negatives.

Random Forest

Moving on from a decision tree, we shall now evaluate a random forest approach. We will utilize the ranger package for this model.

For the sake of processing speed, the number of cross validations from 10 to 2.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud  Good
##      Fraud    58    27
##      Good     21 45464
##
##                Accuracy : 0.9989
##                  95% CI : (0.9986, 0.9992)
##     No Information Rate : 0.9983
```

```
##     P-Value [Acc > NIR] : 0.0001164
##
##                   Kappa : 0.7068
##
##  Mcnemar's Test P-Value : 0.4704864
##
##             Sensitivity : 0.734177
##             Specificity : 0.999406
##          Pos Pred Value : 0.682353
##          Neg Pred Value : 0.999538
##              Prevalence : 0.001734
##          Detection Rate : 0.001273
##    Detection Prevalence : 0.001865
##       Balanced Accuracy : 0.866792
##
##        'Positive' Class : Fraud
##
```

Method	AUC	Sensitivity	Specificity
glm	0.9767752	0.8101266	0.9896903
Decision Tree - rpart	0.8882948	0.7974684	0.9791387
Random Forest - Ranger	0.9283625	0.7341772	0.9994065

The ranger model has given us an AUC score of 0.928, with 21 missed case of fraud and only 21 legitimate transactions being flagged as fraud.

However, the GLM is still the best performing model.

KNN3

Next we shall build and evaluate a KNN3 model.

KNN3 was designed to address some of the issues encountered when using KNN as a binary classifier.[12]

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud  Good
##      Fraud    44    22
##      Good     35 45469
##
##                Accuracy : 0.9987
##                  95% CI : (0.9984, 0.9991)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 0.005774
##
##                   Kappa : 0.6063
##
##  Mcnemar's Test P-Value : 0.111961
##
##             Sensitivity : 0.5569620
##             Specificity : 0.9995164
##          Pos Pred Value : 0.6666667
```

```
##          Neg Pred Value : 0.9992308
##              Prevalence : 0.0017336
##          Detection Rate : 0.0009655
##    Detection Prevalence : 0.0014483
##       Balanced Accuracy : 0.7782392
##
##        'Positive' Class : Fraud
##
```

Method	AUC	Sensitivity	Specificity
glm	0.9767752	0.8101266	0.9896903
Decision Tree - rpart	0.8882948	0.7974684	0.9791387
Random Forest - Ranger	0.9283625	0.7341772	0.9994065
KNN3	0.8094973	0.5569620	0.9995164

The KNN3 has produced an AUC score of 0.809, with 35 false negatives and 22 false positives.

Keras

What are Keras and TensorFLow2?

TensorFlow2 is an open source library for machine learning and deep neural networks developed by the Google Brain Team. Keras is the API of TensorFlow2, although it can run other backends. It allows for the use of CPU or GPU processing, whilst still being user-friendly.

Keras can be used to build two types of models, a sequential model and a functional API model.

Sequential models are best to use when there one input and one output tensor, whereas a functional API approach can be applied to multiple input and output tensor problems [13], and a tensor is a "generalization of vectors and matrices to potentially higher dimensions." [14]

To put it simply, as we are working with single input and output tensors, i.e. our train/test/validation set being our single input and the predicted class being the single output, the sequential model approach is appropriate.

Recipe

The first step in our model is to centre and square the data. For this, we will use the recipes package.

Baking

Now we have our recipe, we can bake the data. At the same time, we will separate Class from the rest of the data, as keras needs to have the "target" as a separate input.

Unfortunately, we must also convert Class to numeric. Once again, 0 will represent a legitimate transaction, and 1 a case of fraud.

Model

Keras models work by passing the model through a number of layers. For each layer, the number of units, or, to think of it another way, the neurons in the brain, must be stipulated. The input shape must also be stated for the first layer. We also must state the metric, or metrics, we wish to evaluate the model by.

Weights

Keras also allows for preforming actions such as saving the weights during training, by using a callback. This functionality allows for the projects to start where they left off, instead of always having to all over again. We will use one to save the weights for each feature.

Training

We can fit our model by using the fit function, and include our callback to save the weights.

Loading Weights

Once saved, we can load the weights into our model.

Predict

And now, we can make predictions using the test set.

Results

```
## Confusion Matrix and Statistics
##
##
##         0     1
##   0 45491    43
##   1     0    36
##
##                Accuracy : 0.9991
##                  95% CI : (0.9987, 0.9993)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 6.756e-06
##
##                   Kappa : 0.6257
##
##  Mcnemar's Test P-Value : 1.504e-10
##
##             Sensitivity : 0.455696
##             Specificity : 1.000000
##          Pos Pred Value : 1.000000
##          Neg Pred Value : 0.999056
##              Prevalence : 0.001734
##          Detection Rate : 0.000790
##    Detection Prevalence : 0.000790
##       Balanced Accuracy : 0.727848
##
##        'Positive' Class : 1
##
```

Using the Keras approach gives us a AUC score of 0.9995, outperforming the glm model by 0.0225. This approach gives us 0 false positives, a perfect specificity score of 1, while only catching 36 cases of fraud, less than half of the cases.

Method	AUC	Sensitivity	Specificity
glm	0.9767752	0.8101266	0.9896903
Decision Tree - rpart	0.8882948	0.7974684	0.9791387
Random Forest - Ranger	0.9283625	0.7341772	0.9994065
KNN3	0.8094973	0.5569620	0.9995164
keras Sequential Model	0.9995278	0.4556962	1.0000000

Conclusion

Reviewing our results, we can see two stand out models, the glm and keras, with auc scores of 0.9767752 and 0.9995278, respectively. Where as across all our models, the specificity score were all very high, ranging from 0.979 to a perfect score of 1, sensitivity never rose about 0.82 (glm) and fell as low as 0.4556 (keras).

Considering this, we should look a bit closer at their results to get a better understanding of how our two top perfroming models did.

Method	AUC	Sensitivity	Specificity	FP	FN
keras Sequential Model	0.9995278	0.4556962	1.0000000	0	43
glm Model	0.9767752	0.8101266	0.9896903	469	15

Note:
FP = False Positivies, FN = False Negatives

We can see that the glm approach gives 15 false negatives and 469 false positives, where as the keras model results in 0 false positives and 43 false negatives. The keras model's extremely high AUC-ROC score is due to how well it identified the legitimate purchases, but its ability to detect fraud is very poor. It was only able to correctly identify 36 out of the 79 fraud cases in the test set.

Despite this, the systems are being evaluated based off their AUC-ROC scores, and as the highest scoring model, the keras sequential model will be used for the final validation.

Final Validation

Preprocess Validation Set

We need to bake our validation set, and create our target.

loading weights

We will also reload the weights from earlier.

Making Predictions

We now make our predictions for the validation set.

Results

```
## Confusion Matrix and Statistics
##
##
##          0     1
##   0 56851    41
```

```
##   1    12    58
##
##                Accuracy : 0.9991
##                  95% CI : (0.9988, 0.9993)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 2.866e-07
##
##                   Kappa : 0.6859
##
##  Mcnemar's Test P-Value : 0.00012
##
##             Sensitivity : 0.585859
##             Specificity : 0.999789
##          Pos Pred Value : 0.828571
##          Neg Pred Value : 0.999279
##              Prevalence : 0.001738
##          Detection Rate : 0.001018
##    Detection Prevalence : 0.001229
##       Balanced Accuracy : 0.792824
##
##        'Positive' Class : 1
##
```

Method	AUC	Sensitivity	Specificity
glm	0.9767752	0.8101266	0.9896903
Decision Tree - rpart	0.8882948	0.7974684	0.9791387
Random Forest - Ranger	0.9283625	0.7341772	0.9994065
KNN3	0.8094973	0.5569620	0.9995164
keras Sequential Model	0.9995278	0.4556962	1.0000000
Final Validation - Keras Sequential	0.9139254	0.5858586	0.9997890

Our Keras sequential model has produced an auc score of 0.9139254 on our final validation set. The model increased its sensitivity by 0.13, but specificity fell by 0.000221. There were 12 false positives and 41 false negatives. This is emblematic of a known issue concerning using AUC-ROC with imbalanced data, a small number of misclassifications can result in a large change in the score [15]

If we compare this result to our other models' test set scores, it would rank mid table, after glm and ranger and above rpart and knn3.

However, if we go by Hosmer and Lemeshow's rule of thumb[5], this would still be considered to be outstanding.

Conclusion and Discussion

This project set out with the aim of evaluating 5 different credit card fraud detection systems (glm, rpart, ranger, knn3, and a keras Sequential Model) based on their AUC-ROC scores.

To address for an extremely imbalanced dataset, we used the ROSE package to get an almost perfect balance between legitimate and fraudulent transaction for our training set. However, this 50/50 balance is not the only option. Mark Bentivegna [16], for example, in an effort to limit the number of false positives, used a 2:1 ratio of legitimate to fraud. This is an area that appears to not have a consensus, and is open to investigation.

On the test set, the keras Sequential model performed the best, with a close to perfect 0.999527, however, when tested on the validation set, this score dropped to below 0.92.

The extremely high initial score the keras model achieved, and the subsequent drop on the validation set, can be explained by how well it classified legitimate transactions and the impact that a small number of misclassification has when dealing with an imbalanced set, such as our test and vaidation sets.

As such, lets have another look at all our models' results, including the number of false positives and false negatives.

Method	AUC	Sensitivity	Specificity	FP	FN
keras Sequential Model	0.9995278	0.4556962	1.0000000	0	43
glm Model	0.9767752	0.8101266	0.9896903	469	15
ranger Model	0.9283625	0.7341772	0.9994065	27	21
keras Sequential Model - Final Validation	0.9139254	0.5858586	0.9997890	12	41
rpart Model	0.8882948	0.7974684	0.9791387	949	16
knn3 Model	0.8094973	0.5569620	0.9995164	22	35

Note:
FP = False Positivies, FN = False Negatives

Only considering the results from the test set for the moment, lets look at the numbers of false positives and false negatives. At this point it may be beneficial to remind ourselves of what False Positives and Negatives represent. Every False Positive is a customer being stopped while attempting to purchase something, and every False Negative is a customer being charged for something they did not purchase. Each of scenario will have different impacts on both the customer and the card issuer.

The ranger model produced 27 false positives, meaning that for over 45000 legitimate purchases, under 30 were incorrectly stopped. Compare this to the glm's 469, while still only a small percentage of the tens of thousands of good transactions, it is considerably higher. The glm model, however, out performed the ranger in identifying cases of fraud by 6, 15 to 21. This leads to the question, does 6 cases of fraud outweigh almost 450 irritated customers?

Based solely on the AUC-ROC score from the test set, the keras sequential model was the best, less than 0.0005 away from perfect discrimination. But yet it allowed almost three times the number of fraudulent transactions to go undetected as the glm model did.

In their 2019 competition, the IEE-CIS [2] equated a higher AUC-ROC score with a better customer experience. In their example of a legitimate purchase being denied, they failed to consider the inverse scenario. If a customer has to repeatedly dispute charges on their card, will they still have a positive experience?

In their 2015 paper discussing the use of AUC-ROC in relation to radiography, Halligan, Altman, and Mallett [17] pointed out that the costs of misclassifications is not equal for false positives and false negatives. In the context of cancer diagnosis, false positives can result in additional unnecessary medical proceedures, not to mention the emotional impact, a false negative could be the difference between life and death.

While the consequences are by no mean as severe, false positives and false negatives in fraud detection are also not equal. As described by the IEE-CIS[2], when a legitimate transaction is falsely flagged as fraud, the card holder may suffer embarrassment, it may be a cause of anxiety in the future, and may lead to a change of shopping habits, i.e. not using the card in the future. This in turn could lead to a drop in revenue for the card issuer.

However, cases of fraud that go undetected have consequences that can lead to similar end results. If a fraudulent transaction isn't flagged as such, the card holder will be billed and will need to contest the charge, if the notice it. If they are successful, the card issuer will have to cover the cost, if they are the card holder will be out of pocket. There is also the time, effort, and resources devoted to this process for both parties. In either case, the customer satisfaction, and therefore the likelihood of the customer to continue using the credit card, is reduced, and thus the card issuer's revenue. While both false positives and false

negatives have an impact, the case for false negatives to be given a higher level of importance is not hard to make.

It is undeniable that false positives have a negative inpact on users, however its size may be overstated. As Laura Glahder Lindberg, Mette Svendsen, Mikala Dømgaard & John Brodersen[17] found in their study of false positives in breast cancer screenings, despite the impact on their lives, the women "expressed gratitude for the screening programme" after the inaccuracy was reveled. It is possible that the customers may have a similar view in relation to credit cards, particularly when an abnormal purchase is attempted. Instead of being irritated, a customer may feel protected. This view may be nurtured further as mitigation techniques continues to develop and improve, such as two-factor authentication, and the impact of a blocked transaction may be reduced from leaving a store empty handed to simply having to tap a pop-up your phone.

Ultimately, this is a decision for the card issuers, and, in turn, their customers. What amount of blocked legitimate transactions are customers willing to tolerate in order to not be the victim of fraud? Or inversely, how much fraud,along with its associated losses, are customers willing to accept in exchange for never having a legitimate transaction blocked?

Through the results for the glm, ranger and keras models, the limitations of the AUC-ROC score are on full display. The scores also raises questions about how misclassifications should be treated, should both false positives and negatives be equal, or should they be weighted? As Lobo, Jiménez-Valverde, and Real[16] point out, the AUC-ROC does not allow for this and weighs both equally. Many alternatives to AUC-ROC have been proposed, from a "Net Benefit" approach[18] to simply also reporting Sensitivity and Specificity scores[19].

As the debate around AUC-ROC's validity continues, perhaps it is time to re-evaluate the impact of false positives and false negatives, not only in the context of fraud detection, but more generally.

References and Acknowlegements

References

1. https://www.ecb.europa.eu/pub/cardfraud/html/ecb.cardfraudreport202110~cac4c418e8.en.html
2. https://www.kaggle.com/competitions/ieee-fraud-detection/
3. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
4. https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/
5. https://www.statology.org/what-is-a-good-auc-score/
6. Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. Calibrating Probability with Undersampling for Unbalanced Classification. In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2015
7. https://www.openml.org/search?type=data&sort=runs&id=1597&status=active
8. https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/
9. https://www.ssb.bank/media/cms/Small_Charges_288C66DA6984D.pdf
10. https://fraud-detection-handbook.github.io/fraud-detection-handbook/Chapter_5_ModelValidationAndSelection/Introduction.html
11. https://deepai.org/machine-learning-glossary-and-terms/decision-tree
12. https://daviddalpiaz.github.io/stat432sp18/supp/knn_class_r.html

13. https://tensorflow.rstudio.com/guides/keras

14. https://www.geeksforgeeks.org/introduction-tensor-tensorflow

15. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/

16. https://towardsdatascience.com/precision-vs-recall-evaluating-model-performance-in-credit-card-fraud-detection-bb24958b2723

17. Laura Glahder Lindberg, Mette Svendsen, Mikala Dømgaard & John Brodersen, Better safe than sorry: a long-term perspective on experiences with a false-positive screening mammography in Denmark, 2012

18. Halligan S, Altman DG, Mallett S. Disadvantages of using the area under the receiver operating characteristic curve to assess imaging tests: a discussion and proposal for an alternative approach. Eur Radiol. 2015 Apr;25(4):932-9. doi: 10.1007/s00330-014-3487-0. Epub 2015 Jan 20. PMID: 25599932; PMCID: PMC4356897.

19. Lobol, Jorge M., Jiménez-Valverde, Alberto, and Real, Raimundo, AUC: a misleading measure of theperformance of predictive distribution models, Global Ecology and Biogeography, 2007.

Acknowledgements

The code to import the dataset was provided by Adam Kariv and Rufus Pollock, and can be found here

Code

Package Install

Keras Install

```
if(!require(keras))install.packages("keras", repos="https://cran.rstudio.com/")
library(keras)
install_keras(envname = "r-reticulate")
```

Package Install and Loading

```
# Installing Required Packages
if(!require(jsonlite))install.packages("jsonlite", repos="https://cran.rstudio.com/")
if(!require(tidyverse))install.packages("tidyverse", repos = "https://cran.rstudio.com/")
if(!require(tidyr))install.packages('tidyr', repos = "https://cran.rstudio.com/")
if(!require(dplyr))install.packages('dplyr', repos = "https://cran.rstudio.com/")
if(!require(kableExtra))install.packages("kableExtra", repos = "https://cran.rstudio.com/")
if(!require(ggplot2))install.packages("ggplot2", repos = "https://cran.rstudio.com/")
if(!require(ROSE))install.packages("ROSE", repos = "https://cran.rstudio.com/")
if(!require(caret))install.packages('caret', repos = "https://cran.rstudio.com/")
if(!require(pROC))install.packages('pROC', repos = "https://cran.rstudio.com/")
if(!require(ranger))install.packages('ranger', repos = "https://cran.rstudio.com/")
if(!require(recipes))install.packages("recipes", repos = "https://cran.rstudio.com/")
if(!require(scales))install.packages("scales", repos = "https://cran.rstudio.com/")



# tensorflow/kears install
```

```
library("jsonlite")
library("tidyverse")
library("tidyr")
library("dplyr")
library("kableExtra")
library("ggplot2")
library("ROSE")
library("caret")
library("pROC")
library("ranger")
library("recipes")
library("scales")
library("keras")
library("tensorflow")


set.seed(1, sample.kind = "Rounding")
set_random_seed(1)
```

Dataset

```
#Creating data set
#Creating data set
json_file <- 'https://datahub.io/machine-learning/creditcard/datapackage.json'
json_data <- fromJSON(paste(readLines(json_file), collapse=""))


# print all tabular data(if exists any)
for(i in 1:length(json_data$resources$datahub$type)){
  if(json_data$resources$datahub$type[i]=='derived/csv'){
    path_to_file = json_data$resources$path[i]
    data <- read.csv(url(path_to_file))

  }
}

#tidy workspace
rm(json_data, json_file, path_to_file, i)
```

Analysing the Data

```
#check sturcture of dataset

head(data)%>%  #problem with longtable
  kbl(caption = "Dataset Head") %>%
  kable_styling(latex_options = c("striped", "hold_position", "scaled_down"))

summary(data)%>%
  kbl(caption = "Dataset Summary") %>%
```

```
  kable_styling(latex_options = c("striped", "hold_position", "scale_down"))


anyNA(data) %>%
  kbl(col.names = "Any NA's") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)
```

```
table(data$Class) %>%
  kbl(caption = "Number of Real and Fraudulent Transactions") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)%>%
  footnote(general = " 0 = Legitimate, 1 = Fraud")
```

```
data$Class %>%
  as.factor() %>%
  as.data.frame() %>%
  ggplot(aes(., fill = .)) +
  geom_bar(show.legend = FALSE)+
  scale_y_continuous(labels = comma) +
  ggtitle("Distribution of Class")+
  scale_x_discrete(labels = c("Legitimate", "Fraud"))
```

```
#amount v Class
data %>%
  ggplot(aes(Amount, fill = Class)) +
  geom_density(alpha = 0.2)+
  scale_x_log10()+
  ggtitle("Amount by Class Scaled")+
  scale_fill_discrete(labels=c('Legitimate', 'Fraud'))
```

```
#V4, V9, V11, V15 V Class
#V4
data %>%
  ggplot(aes(V4, fill = Class)) +
  geom_density(alpha = 0.2) +
  scale_x_log10()+
  ggtitle("V4 by Class Scaled")+
  scale_fill_discrete(labels=c('Legitimate', 'Fraud'))

#V9
data %>%
  ggplot(aes(V9, fill = Class)) +
  geom_density(alpha = 0.2) +
  scale_x_log10()+
  ggtitle("V9 by Class Scaled")+
  scale_fill_discrete(labels=c('Legitimate', 'Fraud'))

#V11
data %>%
  ggplot(aes(V11, fill = Class)) +
  geom_density(alpha = 0.2) +
  scale_x_log10()+
  ggtitle("V11 by Class Scaled")+
```

```
   scale_fill_discrete(labels=c('Legitimate', 'Fraud'))


#V15
data %>%
  ggplot(aes(V15, fill = Class)) +
  geom_density(alpha = 0.2) +
  scale_x_log10()+
  ggtitle("V15 by Class Scaled")+
  scale_fill_discrete(labels=c('Legitimate', 'Fraud'))
```

Creating Training, Test, and Validation sets

```
# Creating Validation, Train and Test sets

set.seed(1, sample.kind = "Rounding")# if using R 3.5 or earlier, use `set.seed(1)

#Validation set
test_index <- createDataPartition(y = data$Class, times = 1, p = 0.2, list = FALSE)
imbal_set <- data[-test_index, ]
validation <- data[test_index, ]

#train and test sets

test_index <- createDataPartition(y = imbal_set$Class, times = 1, p = 0.2, list = FALSE)
imbal_train <- imbal_set[-test_index, ]
test_set <- imbal_set[test_index, ]

rm(imbal_set, test_index)
```

```
#Need to address the extreme imbalance in dataset
#use ROSE package
#set seed
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`

#over sampling with ROSE

data.rose <- ROSE(Class ~ ., data = imbal_train, seed = 1)$data

rm(imbal_train)
```

```
table(data.rose$Class) %>%
  kbl(caption = "Number of Real and Fraudulent Transactions") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)

prop.table(table(data.rose$Class))%>%
  kbl(caption = "Frequency of Real and Fraudulent Transactions") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)
```

```
#renaming levels and relevel, remove time
#data rose
data.rose$Class <- as.factor(data.rose$Class)
levels(data.rose$Class)[levels(data.rose$Class)=="'1'"]<- 'Fraud'
levels(data.rose$Class)[levels(data.rose$Class)=="'0'"]<- 'Good'

data.rose$Class <- relevel(data.rose$Class, "Fraud")
data.rose <- data.rose %>% select(-Time)

#test set
test_set$Class <- as.factor(test_set$Class)
levels(test_set$Class)[levels(test_set$Class)=="'1'"]<- 'Fraud'
levels(test_set$Class)[levels(test_set$Class)=="'0'"]<- 'Good'

test_set$Class <- relevel(test_set$Class, "Fraud")
test_set <- test_set %>% select(-Time)

#val set
validation$Class <- as.factor(validation$Class)
levels(validation$Class)[levels(validation$Class)=="'1'"]<- 'Fraud'
levels(validation$Class)[levels(validation$Class)=="'0'"]<- 'Good'

validation$Class <- relevel(validation$Class, "Fraud")
validation <- validation %>% select(-Time)
```

Detection systems

```
## Set traincontrol
trainControl <- trainControl(method = "cv",
                             number = 10,
                             classProbs = TRUE,
                             summaryFunction = twoClassSummary
)
```

```
#GLM 11:36 lasts less than 2 minutes
glm_fit <- caret::train(
  Class ~ .,
  data = data.rose,
  trControl = trainControl,
  method = "glm",
  family = "binomial",
  metric = "ROC"
)

predict_test <- data.frame(
  obs = test_set$Class,
  predict(glm_fit, newdata = test_set, type = "prob"),
  pred = predict(glm_fit, newdata = test_set, type = "raw")
)
predict_test$pred <- relevel(predict_test$pred, "Fraud")
glm_auc <- twoClassSummary(data = predict_test, lev = levels(predict_test$obs))
```

```r
#record results
glm_cf <- confusionMatrix(predict_test$pred, test_set$Class)
glm_cf
auc_results <- tibble(Method = "glm", AUC  = glm_auc[1], Sensitivity = glm_auc[2], Specificity = glm_au

auc_results %>%
  kbl(label = "Area Under Curve Results") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)
```

```r
#Decision Tree
#10=15 min
rpart_fit <- caret::train(Class ~ .,
                    data = data.rose,
                    method = "rpart",
                    trControl = trainControl,
                    metric = "ROC"
                    )


predict_test <- data.frame(
  obs = test_set$Class,
  predict(rpart_fit, newdata = test_set, type = "prob"),
  pred = predict(rpart_fit, newdata = test_set, type = "raw")
)
predict_test$pred <- relevel(predict_test$pred, "Fraud")
rpart_auc <- twoClassSummary(data = predict_test, lev = levels(predict_test$obs))
rpart_cf <- confusionMatrix(predict_test$pred, test_set$Class)
rpart_cf
auc_results <- bind_rows(auc_results, tibble(Method = "Decision Tree - rpart", AUC  = rpart_auc[1], Sens

auc_results %>%
  kbl(label = "Area Under Curve Results") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)
```

```r
trainControl2 <- caret::trainControl(method = "cv",
                              number = 2,
                              classProbs = TRUE,
                             summaryFunction = twoClassSummary,
                             verbose = FALSE
)

tuneGrid <- data.frame(
  splitrule = "gini",
  min.node.size = 1
)

ranger_fit <- caret::train(Class ~ .,
                     data = data.rose,
                     method = "ranger",
                     trControl = trainControl2,
                     metric = "ROC",
                     verbose = FALSE)
```

```
predict_test <- data.frame(
  obs = test_set$Class,
  predict(ranger_fit, newdata = test_set, type = "prob"),
  pred = predict(ranger_fit, newdata = test_set, type = "raw")
)

predict_test$pred <- relevel(predict_test$pred, "Fraud")
ranger_auc <- twoClassSummary(data = predict_test, lev = levels(predict_test$obs))

ranger_cf <- confusionMatrix(predict_test$pred, test_set$Class)
ranger_cf
auc_results <- bind_rows(auc_results, tibble(Method = "Random Forest - Ranger", AUC  = ranger_auc[1], Se

auc_results %>%
  kbl(label = "Area Under Curve Results") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)
```

```
knn3_cv <- knn3(Class ~ ., data.rose)

predict_test <- data.frame(
  obs = test_set$Class,
  predict(knn3_cv, newdata = test_set, type = "prob"),
  pred = predict(knn3_cv, newdata = test_set, type = "class")
)

predict_test$pred <- relevel(predict_test$pred, "Fraud")
knn_auc <- twoClassSummary(data = predict_test, lev = levels(predict_test$obs))

auc_results <- bind_rows(auc_results, tibble(Method = "KNN3", AUC  = knn_auc[1], Sensitivity = knn_auc[1

knn_cf <- confusionMatrix(predict_test$pred, test_set$Class)
knn_cf
auc_results %>%
  kbl(label = "Area Under Curve Results") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)
```

```
#using recipes to center and scale
rec_obj <- recipe(Class ~., data = data.rose)%>%
  step_center(all_predictors(), -all_outcomes())%>%
  step_scale(all_predictors(), - all_outcomes())%>%
  prep(data = data.rose)
```

```
#baking the data.rose and test set, and seperating class
x_train <- bake(rec_obj, new_data = data.rose) %>% select(-Class)
x_test <- bake(rec_obj, new_data = test_set) %>% select(-Class)

y_train <- ifelse(pull(data.rose, Class)== "Fraud", 1, 0)
y_test <- ifelse(pull(test_set, Class)== "Fraud", 1, 0)
```

```
seq_mod <- keras_model_sequential()
#add layers
seq_mod %>%
  #first layer
  layer_dense(
    units = 80,
    activation = "relu",
    input_shape = ncol(x_train)
  ) %>%
  #add dropout to prevent overfitting
  layer_dropout(rate = 0.1)%>%
  #second layer
  layer_dense(
    units = 75,
    activation = "relu"
  )%>%
  layer_dropout(rate = 0.1)%>%
  #third layer
  layer_dense(
    units = 35,
    activation = "relu"
  )%>%
  layer_dropout(rate = 0.1)%>%
  #fourth layer

  layer_dense(
    units = 16,
    activation = "relu"
  ) %>%

  layer_dropout(rate = 0.1) %>%
  layer_dense(
    units = 12,
    activation = "relu"
  )%>%

   layer_dropout(rate = 0.1) %>%
  layer_dense(
    units = 4,
    activation = "relu"
  )%>%
  #output layer
  layer_dense(
    units = 1,
    activation = "sigmoid" # 'softmax', 'elu', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard
  )%>%
  #compile
  compile(
    optimizer = optimizer_adamax(), #learning_rate = 0.0075, decay = 0.05
    #adam 0,8598, SGD 0-9471, RMSprop 0.9541, adadelta 0.9541 , adagrad  0.9596, adamax 0.9604 , nadam
    loss = "binary_crossentropy", #with adamax - binary_crossentropy  , categorical_crossentropy error,
    metrics = list(metric_auc(),
```

```
                  metric_false_negatives(),
                  metric_false_positives(),
                  metric_true_negatives(),
                  metric_true_positives()) #'accuracy', 'binary_accuracy
  )

#https://www.rdocumentation.org/packages/kerasR/versions/0.8.1/topics/keras_compile
```

```
##create cp path
checkpoint_path <- "training_2/cp-list{epoch:04d}.ckpt"
checkpoint_dir <- fs::path_dir(checkpoint_path)
batch_size <- 50
##cp callback
cp_callback <- callback_model_checkpoint(
  filepath = checkpoint_path,
  verbose = 1,
  save_weights_only = TRUE,
  save_freq = 5*batch_size
)
```

```
history <- fit(
  object = seq_mod,
  x = as.matrix(x_train),
  y = y_train,
  batch_size = 50,
  epochs = 10,
  validation_split = 0.20,
  callbacks = cp_callback
)
```

```
latest <- tf$train$latest_checkpoint(checkpoint_dir)

load_model_weights_tf(seq_mod, latest)
```

```
keras_predict <- seq_mod %>%predict(as.matrix(x_test)) %>% '>'(0.5) %>% k_cast("int32")
keras_table <-table(as.numeric(keras_predict[,1]), as.numeric(y_test))
```

```
keras_auc <- auc(as.numeric(keras_predict[,1]), as.numeric(y_test))
#keras_auc #0.9995
keras_table <-table(as.numeric(keras_predict[,1]), as.numeric(y_test))

keras_cf <- confusionMatrix(keras_table, positive = "1")
```

```
keras_cf

auc_results <- bind_rows(auc_results, tibble(Method = "keras Sequential Model", AUC  = keras_auc[1], Se

auc_results %>%
  kbl(label = "Area Under Curve Results") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)
```

```
keras_v_glm <- tibble(Method = "keras Sequential Model",
                      AUC  = keras_auc[1],
                      Sensitivity = keras_cf$byClass[1],
                      Specificity = keras_cf$byClass[2],
                      FP = keras_cf$table[2,1],
                      FN = keras_cf$table[1,2])

keras_v_glm <- bind_rows(keras_v_glm,
                         tibble(Method = "glm Model",
                                AUC  = glm_auc[1],
                                Sensitivity = glm_cf$byClass[1],
                                Specificity = glm_cf$byClass[2],
                                FP = glm_cf$table[1,2],
                                FN = glm_cf$table[2,1]))



keras_v_glm %>%
  kbl(label = "keras v glm") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)%>%
  footnote(general = " FP = False Positivies, FN = False Negatives")
```

Keras

Final Validation

```
x_val <- bake(rec_obj, new_data = validation) %>% select(-Class)

y_val <- ifelse(pull(validation, Class)== "Fraud", 1, 0)
```

```
latest <- tf$train$latest_checkpoint(checkpoint_dir)

load_model_weights_tf(seq_mod, latest)
```

```
val_predict <- seq_mod %>% predict(as.matrix(x_val)) %>% '>'(0.5) %>% k_cast("int32")
val_auc <- auc(as.numeric(val_predict), as.numeric(y_val))

val_table <-table(as.numeric(val_predict), as.numeric(y_val))
val_cf <- confusionMatrix(val_table, positive = "1")
```

```
val_cf

auc_results <- bind_rows(auc_results, tibble(Method = "Final Validation - Keras Sequential", AUC  = val

auc_results %>%
  kbl(label = "Area Under Curve Results") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)
```

Conclusions and discussions

```
results_FP_FN <- tibble(Method = "keras Sequential Model",
                        AUC  = keras_auc[1],
                        Sensitivity = keras_cf$byClass[1],
                        Specificity = keras_cf$byClass[2],
                        FP = keras_cf$table[2,1],
                        FN = keras_cf$table[1,2])

results_FP_FN <- bind_rows(results_FP_FN,
                           tibble(Method = "glm Model",
                                  AUC  = glm_auc[1],
                                  Sensitivity = glm_cf$byClass[1],
                                  Specificity = glm_cf$byClass[2],
                                  FP = glm_cf$table[1,2],
                                  FN = glm_cf$table[2,1]))

results_FP_FN <- bind_rows(results_FP_FN,
                           tibble(Method = "ranger Model",
                                  AUC  = ranger_auc[1],
                                  Sensitivity = ranger_cf$byClass[1],
                                  Specificity = ranger_cf$byClass[2],
                                  FP = ranger_cf$table[1,2],
                                  FN = ranger_cf$table[2,1]))

results_FP_FN <- bind_rows(results_FP_FN,
                           tibble(Method = "keras Sequential Model - Final Validation",
                                  AUC  = val_auc[1],
                                  Sensitivity = val_cf$byClass[1],
                                  Specificity = val_cf$byClass[2],
                                  FP = val_cf$table[2,1],
                                  FN = val_cf$table[1,2]))


results_FP_FN <- bind_rows(results_FP_FN,
                           tibble(Method = "rpart Model",
                                  AUC  = rpart_auc[1],
                                  Sensitivity = rpart_cf$byClass[1],
                                  Specificity = rpart_cf$byClass[2],
                                  FP = rpart_cf$table[1,2],
                                  FN = rpart_cf$table[2,1]))


results_FP_FN <- bind_rows(results_FP_FN,
```

```
                          tibble(Method = "knn3 Model",
                                 AUC  = knn_auc[1],
                                 Sensitivity = knn_cf$byClass[1],
                                 Specificity = knn_cf$byClass[2],
                                 FP = knn_cf$table[1,2],
                                 FN = knn_cf$table[2,1]))

results_FP_FN %>%
  kbl(label = "Results including False Positives and Negatives") %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = F)%>%
  footnote(general = " FP = False Positivies, FN = False Negatives")
```