ISyE 6740 - Spring 2021
Final Report

Author Name: Ahmed Bilal

Project Title: Credit Card Fraud Detection Classification

Problem Statement
Day-to day credit card transactions continue to grow. Global non-cash transactions increased nearly 14% from 2018–2019 to reach 708.5 billion transactions, the highest growth rate recorded in the past decade[1]. It means customers can spend more money, but meanwhile the risks of credit card fraud are also on the ride. Thus, it is important for credit card companies to identify fraudulent credit card transactions. The goal for this project is to find a good model to identify credit card fraud by comparing different models and parameters.

Data Source
Dataset: https://www.kaggle.com/mlg-ulb/creditcardfraud
The dataset contains Credit Card transactions made in September 2013 by European cardholders. The transactions occurred in two days, where we have 492 frauds out of 284,807 transactions. This dataset was made available by the Université Libre de Bruxelles' Machine Learning Group. Since the transaction dataset consists of real transactions, the dataset has been PCA transformed for the purposes of anonymity. As such, all the dependent variables in the dataset, except for the 'Time' and 'Amount' features are PCA features. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Methodology
Our goal is to find the best classification model for the purposes of fraud detection. In this project, we will consider several classification models, such as Linear and RBF Kernel SVM, KNN, Decision Tree, Random forest, Neural networks, Logistic regression and Isolation Forest. Our process will be as follows:

- Data Cleaning
- Exploratory Data Analysis
  The PCA features tell us that we do not need to worry about issues of collinearity in our dependent variables.
- Train-Cross validation-Test Split
  We will do a train-test split in the start since we do not want to be performing any resampling techniques on the test set. As such, the imbalanced classes on the test and validation tests will remain untouched.
- Upsample/Downsample/SMOTE for Class Balancing and Cross-Validation
  We do not want to undersample for our model. As such, we will choose between oversampling and SMOTE technique depending on their performance on the test set. Since we will be resampling, we will need to be careful about cross-validation. Santos, Miriam et. al 's *Cross-Validation for*

[1]Download the World Payments Report 2020 here. (n.d.). Retrieved May 04, 2021, from https://worldpaymentsreport.com/

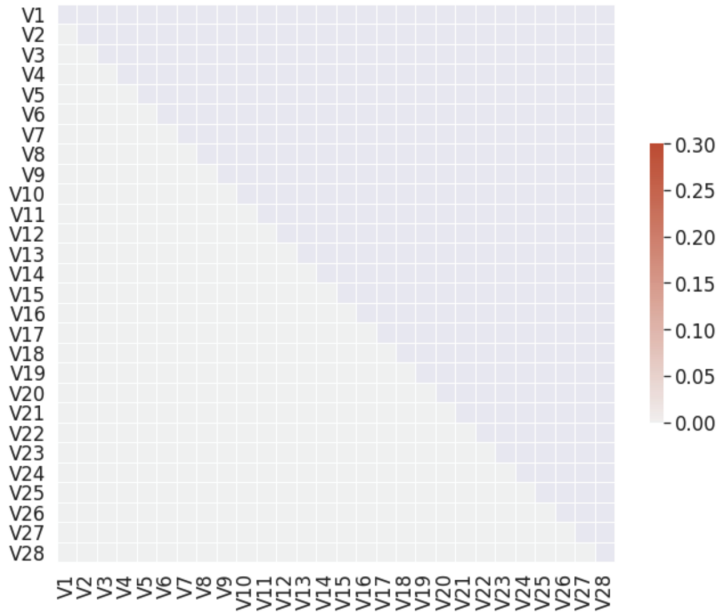
Imbalanced Datasets: Avoiding Overoptimistic and Overfitting Approaches[2] tells us that cross-validation is a standard procedure for performance evaluation, its joint application with oversampling remains an open question for researchers farther from the imbalanced data topic. A frequent experimental flaw is the application of oversampling algorithms to the entire dataset, resulting in biased models and overly-optimistic estimates. As such, we will use Synthetic Minority Oversampling Technique during Cross Validation - perform oversampling for each fold prior to training.

- Feature Engineering/Feature Selection
  The features have already been PCA transformed so far. PCA by itself is not a feature selection technique, but a dimension reduction one. As such, we will use a combined approach of regularization techniques like Lasso with our PCA features for feature selection and dealing with issues of multicollinearity as seen in *Combining least absolute shrinkage and selection operator (LASSO) and principal-components analysis for detection of gene-gene interactions in genome-wide association studies*[3]
- Apply classification algorithms
  Algorithms: Linear and RBF Kernel SVM, KNN, Decision Tree, Random forest, Neural networks, Logistic regression and Isolation Forest with hyperparameter tuning.

Evaluation and Final Results

0.1 Data Exploration

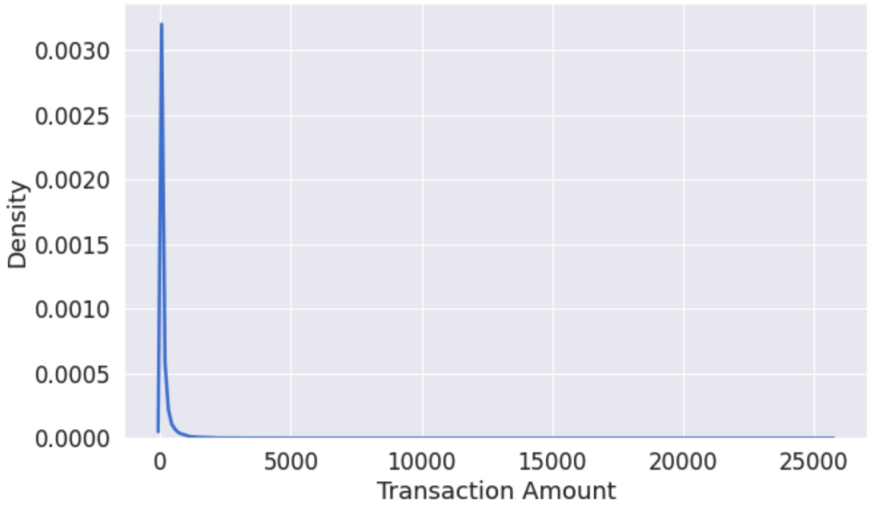
There are no null values in the data set. Further, we expect the PCA transformed variables to not exhibit any correlation with each other. Here we confirm this hypothesis.



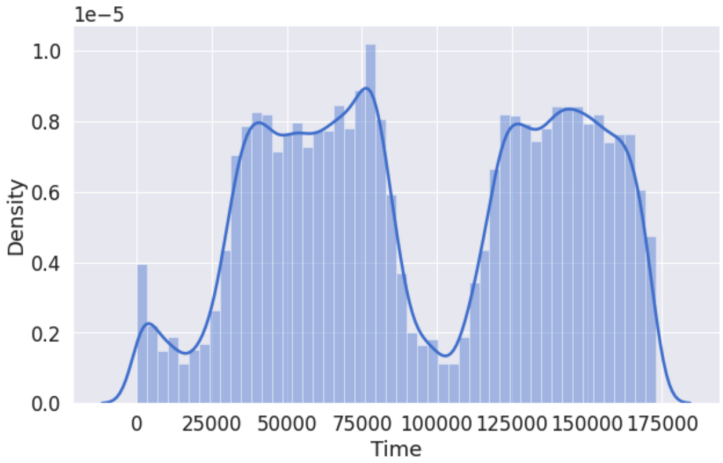
[2]Santos, M. S., Soares, J. P., Abreu, P. H., Araujo, H., amp; Santos, J. (2018). Cross-validation for imbalanced datasets: Avoiding overoptimistic and overfitting approaches [research frontier]. IEEE Computational Intelligence Magazine, 13(4), 59-76. doi:10.1109/mci.2018.2866730

[3]D'Angelo GM, Rao D, Gu CC. Combining least absolute shrinkage and selection operator (LASSO) and principal-components analysis for detection of gene-gene interactions in genome-wide association studies. BMC Proc. 2009;3 Suppl 7(Suppl 7):S62. Published 2009 Dec 15. doi:10.1186/1753-6561-3-s7-s62

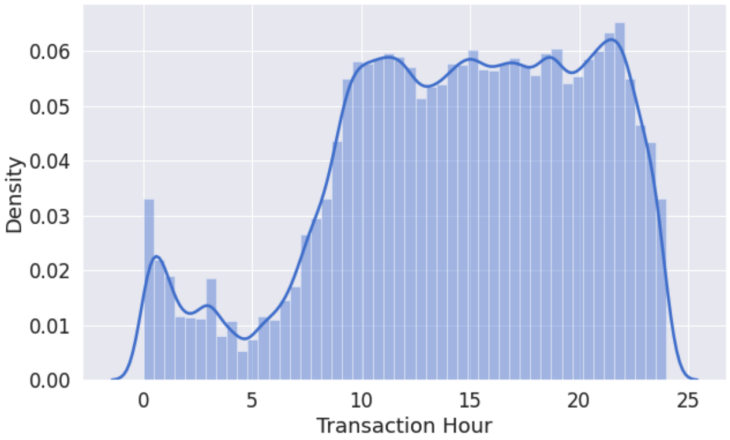
Using KDE for density distribution, we can see that the large majority of transactions were small amounts with large outliers.



The Time variable reflects time elapsed since first transaction in seconds.



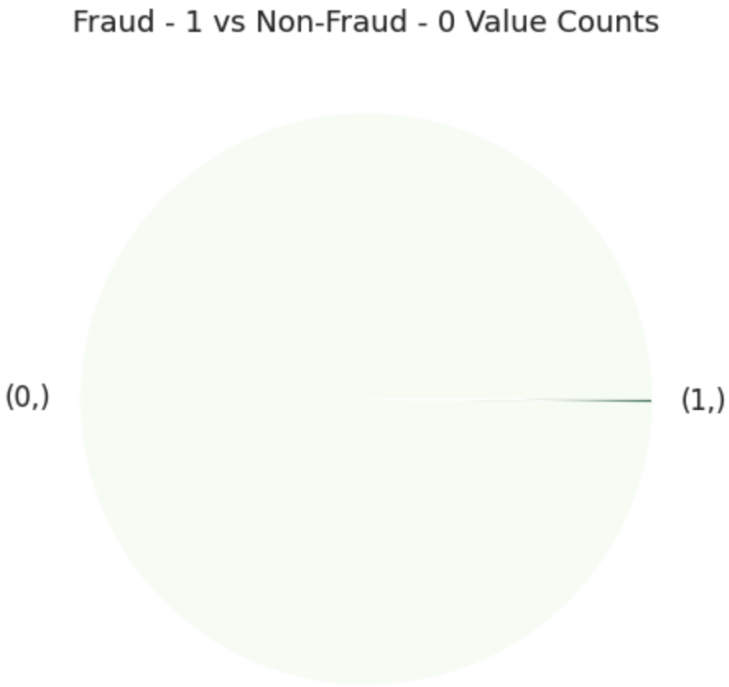
This doesn't seem very useful on it's own. Let's transform it so it reflects daily time of the transaction.



These hours do not correspond to actual hours of transactions since we do not know what the initial transaction hour was, but we still get an idea of transactions that happened within the same hour in a day. The histogram shows us that there are some peak hours grouped together with a high number of transactions and a group of hours with a low number of transactions as per our intuition about credit card transactions.
Now we will normalize the Transaction Hour and Transaction Amount features to fall in line with the PCA features which had to be normalized before any PCA Transformation. However, we will normalize the Test and Train sets separately in order to prevent any sort of leakage of test data distribution in the training data[4].

0.2 Class Imbalance Issue

Class Imbalance in the Train Set:



[4]https://machinelearningmastery.com/data-leakage-machine-learning/

Dealing with class imbalance issues when applying our models:

1. We do not want to undersample and lose data points
2. We do not want to oversample all the data points originally because then when we do *leave one participant out cross-validation* to assess model performance in the training set, the cross-validation success methods will not be very successful[5]:

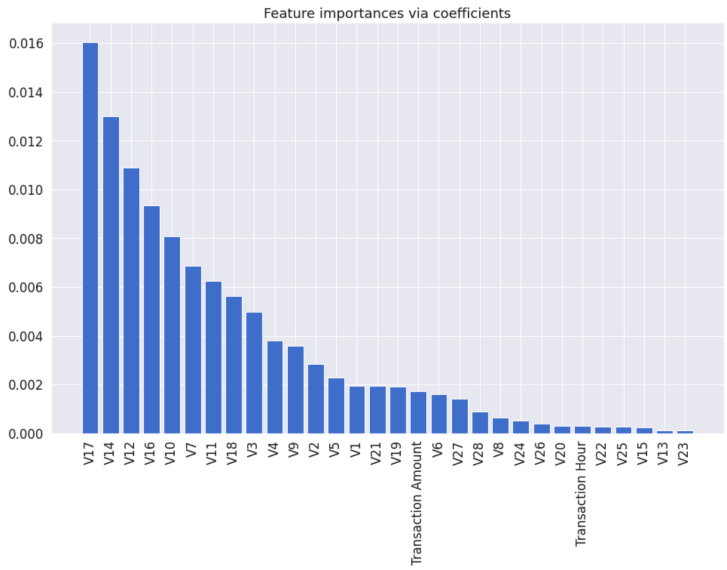
As such our approach will be to perform SMOTE which interpolates the minority class, instead of just duplicating it like oversampling does. This does, to an extent deal with the issue of duplication of data points in the cross validation validation and training folds. However, in order to further make our cross validation results more accurate and prevent overfitting of our models, we will use the following approach:
We will start with the original unbalanced dataset and perform cross-validation first. For each iteration in the cross-validation, first we will exclude the sample to be used for the validation set for that iteration, and then we will oversample the minority class using SMOTE in the remaining set for that iteration. We can use this technique with k-fold cross-validation as well.

0.3 Feature Selection

For the feature selection stage, we will not be oversampling the training data with SMOTE as Rok Blagus1 and Lara Lusacorresponding in *SMOTE for high-dimensional class-imbalanced data*[6] argue that feature selection before SMOTE (Synthetic Minority Oversampling Technique) is preferred. This is because most variable selection methods assume that the samples are independent and Oversampling the minority class with SMOTE violates the independence assumption. As such we will be performing the variable selection methods on the original unbalanced dataset.

0.3.1 Feature Selection - Embedded Method: Lasso with Cross Validation

We use 10 fold cross-validation to find the best model with an alpha value: $1.7e-05$. We then fit the the model to the training data and find the variables in order of the largest coefficients and plot them below.



[5]https://www.marcoaltini.com/blog/dealing-with-imbalanced-data-undersampling-oversampling-and-proper-cross-validation

[6]Blagus, R., amp; Lusa, L. (2013). SMOTE for HIGH-DIMENSIONAL class-imbalanced data. BMC Bioinformatics, 14(1). doi:10.1186/1471-2105-14-106

0.3.2 Feature Selection - Embedded (Ensemble) Method: Random Forest Regressor

We fit a Random Forest Regressor with max_depth=2,n_estimators=100, and oob_score=True to our training data. The decision trees in the random forest uses the most important features to predict the response variable and like other embedded methods penalizes features that are not helpful in predicting the response variables. We then select the variables with the highest feature importance and plot their importance coefficients in order.

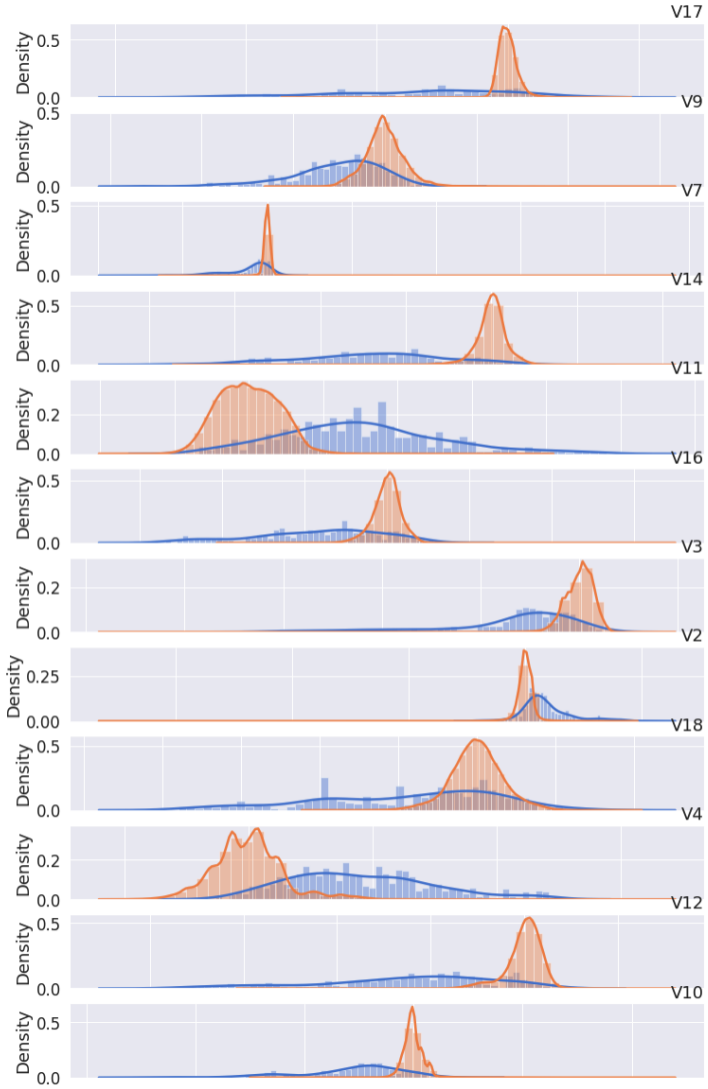
0.3.3 Feature Selection - Wrapper Methods: Forward selection and Backward Selection

So far we have only used embedded methods to reduce our features, instead of simply selecting features by their individual predictive ability, we want to confirm that we are not leaving out any feature a part of an important subset of features. As such, we will use sequential feature selection with a Lasso estimator in forward and backward selection to form feature subsets by using greedy algorithm with cross validation. We get similar subsets from the forward and backward selection methods:

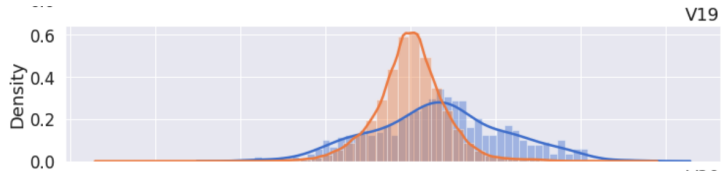
```
Features selected by forward sequential selection: ['V1' 'V2' 'V3' 'V4' 'V5' 'V6' 'V7' 'V9' 'V10' 'V11' 'V12' 'V14'
'V16'
 'V17' 'V18']
Features selected by backward sequential selection: ['V1' 'V2' 'V3' 'V4' 'V5' 'V6' 'V7' 'V9' 'V10' 'V11' 'V12' 'V14'
'V16'
 'V17' 'V18']
```

0.3.4 Feature Selection - Class Distribution

We will now visually inspect our selected variables from the above methods. We can observe that almost all our variables exhibit significant differences in their distributions of the Fraud and Non-Fraud classes.



In addition, we find that out of all the variables we have discarded so far, only variable 'V19' exhibits a slight deviation in distributions of the different classes so we will add it back for now.



This gives us the following set of variables that we will be using for our models:

```
['V17', 'V9', 'V7', 'V14', 'V11', 'V16', 'V3', 'V2', 'V18', 'V4', 'V12', 'V10', 'V19']
```

0.4 Model Selection

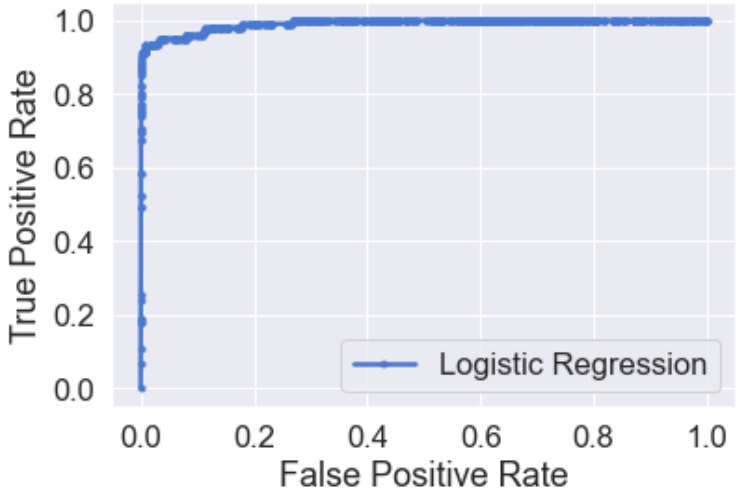
In order to chose the right models, we want to get an idea of the separability of our classes, as such we will use TSNE technique which converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data[7]
TSNE Visualization (Fraud - Blue, Non-Fraud - Red):

The TSNE visualization shows us that the decision boundary will be very fragmented and complex as Fraud data points seems to be spread all over. As such, in order to form such a complex decision boundary, we will first try to estimate it directly. Therefore, we will start by building a Logistic Regression model to represent such a complex decision boundary.

0.4.1 Logistic Regression

Logistic regression uses the sigmoid function - an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1 - to estimate class probabilities. For the logistic regression, we will use parameter C as our regularization parameter. Parameter C is the inverse of the regularization strength. Similar to support vector machines, smaller values specify stronger regularization. Regularization strength penalizes the magnitude of the parameters in order to reduce overfitting. As such, we will only use parameter C for Hyperparameter Tuning. From Grid Search cross validation result, the best C we get is 1. The following is the plot of ROC curve for this model.

[7]Sklearn.manifold.tsne¶. (n.d.). Retrieved May 04, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

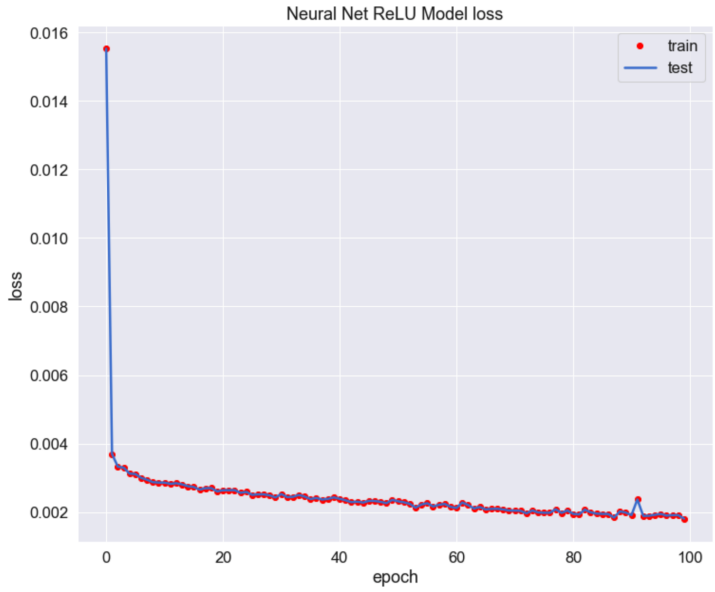


The recall score for training dataset with SMOTE cross validation is 0.895, while for test dataset, we get a recall rate of 0.93 with a misclassification rate of 26% of normal transactions as fraudulent ones. This is unusual. Test accuracy should not be significantly higher than the train dataset since the model is optimized for the latter. One of the reason is that maybe that train/test split percentage is too high. So we use 60% of the data to train on and 40% to evaluate the model. Then the recall scores for test dataset and training dataset are 0.92 and 0.9 respectively with an optimal C for 3. This is a more reasonable result. The difference may be due to randomness, especially due to the fact that the fraudulent transactions are so few. As such, we can expect some variance by chance in the recall rates due to the specific distribution of fraudulent transactions in either of the dataset. We will make one more attempt at predicting the decision boundary directly by using Neural Networks.

0.4.2 Neural Networks

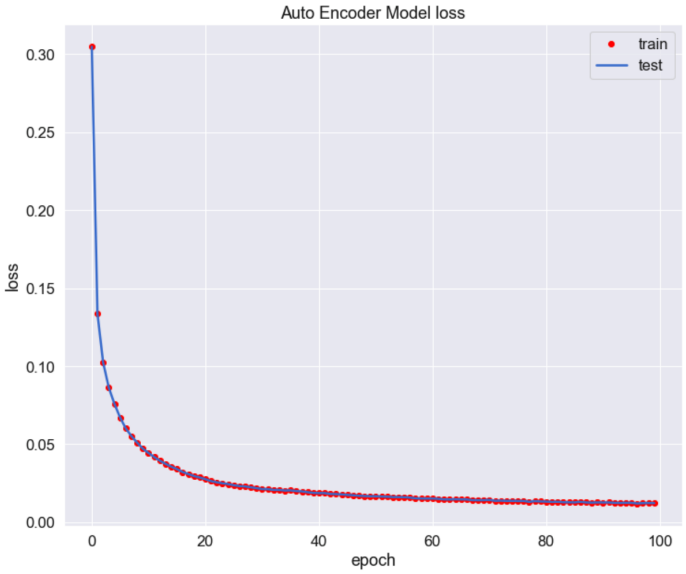
We will be using the Pytorch library to build and train our neural network. This involves converting our training and test data into Tensors in order for us to build a neural network for it.

- **ReLu Neural Network**:
  Our model consists of 4 hidden layers with 26,26,26,and 16 neurons respectively for each of the layers. For all layers, except the final one, we will be using a ReLU activation function. Since we are trying to create a complex decision boundary as shown by the TSNE visualization, we will not be using a linear activation function. Further, for non-linear activation functions, we initially tried sigmoid and hyperbolic tangent function, but found that the functions exhibited poor performance due to their tendency to restrict all large values to 1.0 and small values to -1 and 0 for tanh and the sigmoid function respectively. As such, the layers were unable to capture a lot of gradient information in our models due to this saturation effect and low sensitivity to function inputs. Hence, we opted to use a Rectified Linear Activation Function with stochastic gradient descent with backpropogation of errors, making it more sensitive to the activation input and able to capture detailed gradient information in each of the layers. For the final output layer, we used a sigmoid function to restrict values between 0 and 1 while retaining as much information about input values as possible.
  For our loss function, we opted for the Binary Crossentropy Loss BCELoss function to correspond with our final sigmoid layer of output. For each epoch in our training, the BCELoss function increases exponentially in order to punish extremely wrong predictions more strongly than those that are less wrong. For the optimizer function, we will be using the Adam optimizer that uses stochastic gradient descent with a small learning rate of $1e-3$. With 100 epochs, we see that the error function on the SMOTE balanced training data reduces very significantly from the initial initial epochs, with minute improvements in accuracy with the subsequent epochs.



However, even though this performance on the upsampled training data set seems promising, we observe poor recall rate of 0.72 for the fraud transactions on the unbalanced test data set as the model is unable to predict fraud transactions well.

– **Autoencoder Neural Network**:
We build a FeedForward Autoencoder neural network using the ReLU activation function for the encoding and decoding layers, 'mean squared error' loss function and the Adam optimizer. In order to train the network, we run it on the non-fraud transactions only, and then test graph its error scores on SMOTE up-sampled training data. As such, we observe a similar, albeit smoother pattern of error reduction with subsequent epochs.

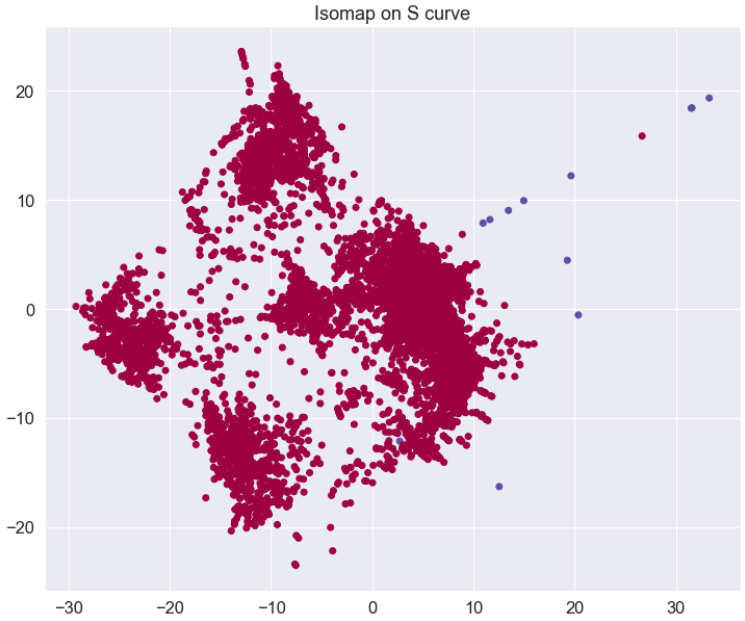


After applying it on the test dataset, we are able to get a recall rate of 0.89.

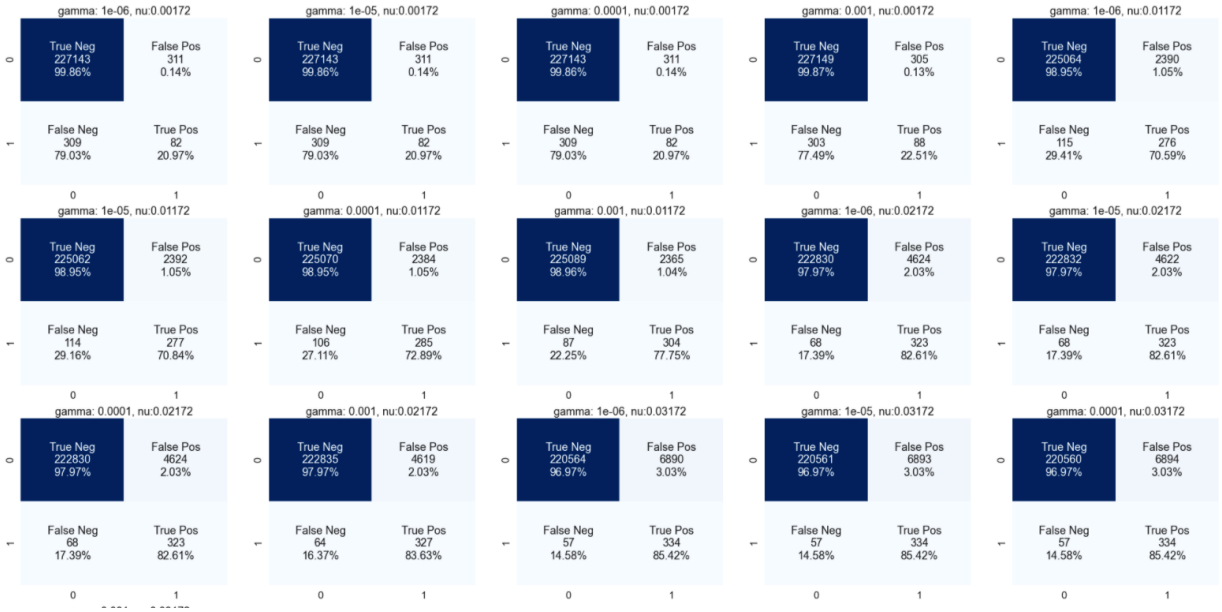
0.4.3 One-Class SVM

In order to get an intuition for the feasibility of using a geometric approach for anomaly detection, we will use Isomap which uses the geodesic distances based on the pairwise distance between data points

to create a low dimensional embedding, instead of t-distribution probabilities like t-SNE.
Isomap Visualization (Fraud - Blue, Non-Fraud - Red):

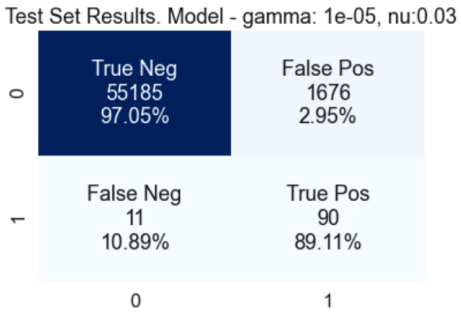


This Isomap low-dimensional embedding gives us reason to believe that a geometric approach through the One-Class SVM with the non-linear Radial Basis Kernel has a strong chance of creating a hypersphere that separates the fraudulent and non-fraudulent transaction reasonably well. The kernel trick allows us to create a more complex hypersphere decision boundary. Since the hypersphere geometric shape depends on the rbf kernel gamma value and the One Class SVM nu value (fraction of training errors), we will be performing hyperparameter tuning on these two parameters and plotting their confusion matrix results on the training data:



We can observe from these confusion matrices that a nu value of 0.03 with a gamma value in the range of $1e-03$ to $1e-06$ allows us to have the best recall rate of 0.85 with a reasonably low false positive

rate of 3.03% on the training data. We will now use these parameters to predict our test data. After doing so, we are able to get a good recall rate of 0.89 on our test set alongside a low misclassification rate as well.



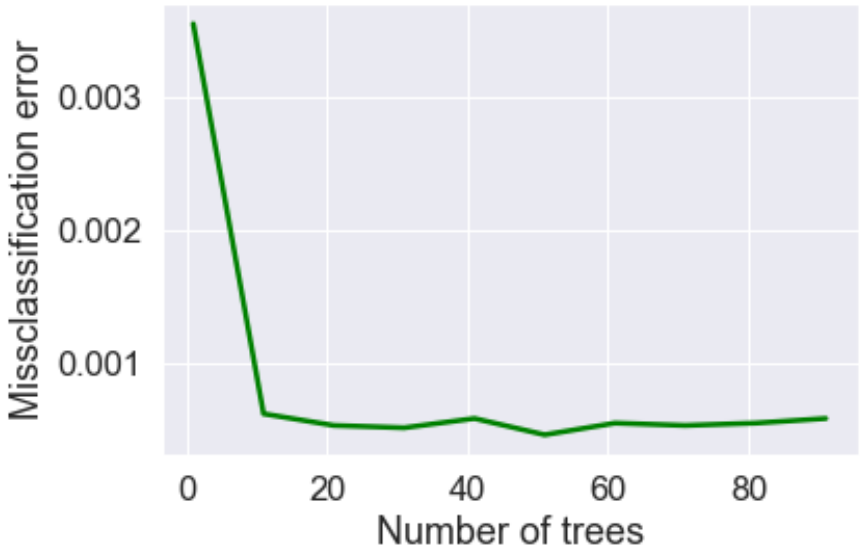
If we increase our nu value to 0.5, we are able to get a recall rate of 0.95. However, this is accompanied by a 25 percent miclassification rate of normal transactions as fraudulent. We will now try another distance based metric - K-nearest neighbors.

0.4.4 K-nearest neighbors

KNN is a non-parametric method used for classification and regression. Its algorithm is to assign a data point a label by taking a majority vote or average value over the K training points closest to this data point. So it's very important to have the right k-value when using this to avoid overfitting and under-fitting of the dataset. Larger k value means smother curves of separation resulting in less complex models, which may cause under-fitting. Whereas, smaller k value tends to overfit the data and ending with complex models.
For this dataset, we use grid search with SMOTE cross validation for hyperparameter tuning to get the best model performance on our training dataset. We choose 'leaf_size','n_neighbors' and 'p' as our tuning parameters. From GridSearch, it can be seen that the best number of leaf_size is 1 while the optimal distance matrix is Manhattan or p = 1(p=2 is euclidean_distance). The best number of k is 2 - this is expected due to the need for a highly complex and noisy decision boundary. After applying it on the test dataset, we get a recall score of 0.89.
Lastly, we will now try ensemble methods by combining several simpler models to see if we can get a better score by using an ensemble approach instead.

0.4.5 Random Forest

Random Forest is a classification algorithm that combines many decision trees. It uses bootstrap method for the training data and grows a decision tree for each batch of bootstrap samples. First we test different number of trees(parameter 'n_estimators') for Random Forest to locate a reasonable range for the number of trees. Our results from the training data show that 50 decision trees achieve the minimum misclassification error. Below is the number of trees with their associated misclassification error on the training dataset.



We select two important parameters: number of trees('n_estimators') and the complexity of the tree('max_depth') to build a series of models and use GridSearchCV to find the best one. The best model from this grid search cross validation SMOTE process gets a recall score of 0.87 for test dataset with the corresponding parameters: $'\max_depth' : 10,' n_estimators' : 30$.
However, when we use this model to predict on the training dataset without SMOTE, we get a recall score of 0.98. It implies that there probably is an overfitting problem for the model. Decision trees do use a greedy algorithm and tend to overfit and we assume the Random Forest ensemble technique is not doing a good enough job of preventing overfitting. As such, we will now try to use and ensemble of non-greedy decision trees to see if we can get better results.

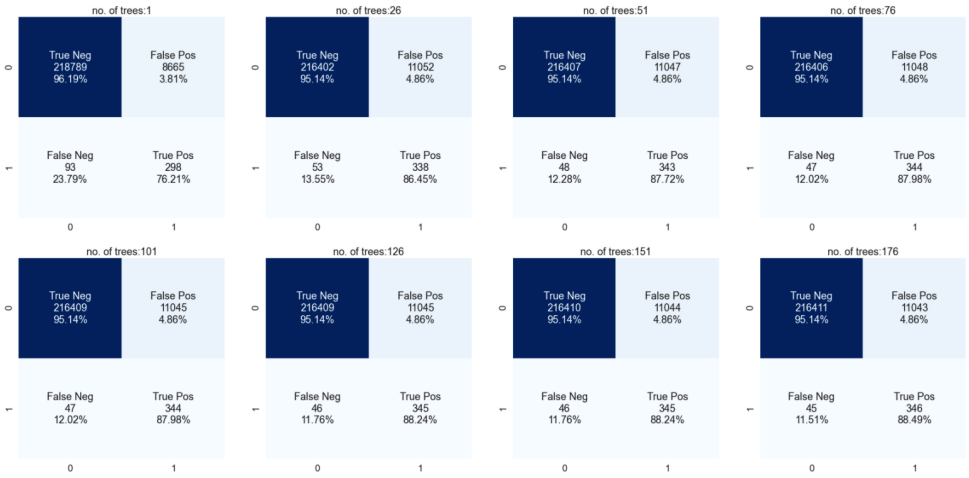
0.4.6 Isolation Forest

The Isolation Forest is similar to the Random Forest, in the sense that they are both an ensemble of multiple decision trees. We will be using the ExtraTreeRegressor decision trees that split on the best split from amongst randomly chosen attributes that have been split on randomly chosen points. Our hope is that by choosing a random decision tree regressor, we will be able to avoid building greedy trees that overfit to the training data and hence, are unable to find the extremely sparse fraud transactions in the test set, which do not share a lot in common with the fraud transaction distribution in the training set. Our Isolation Forest will be using the Bootstrap Aggregated Regressor, and using the anomaly scores for each of the data points to find outliers. We will be using the Isolation Forest in a supervised setting so we will be setting the contamination ratio which corresponds to the amount of anomalies we expect in our data set.

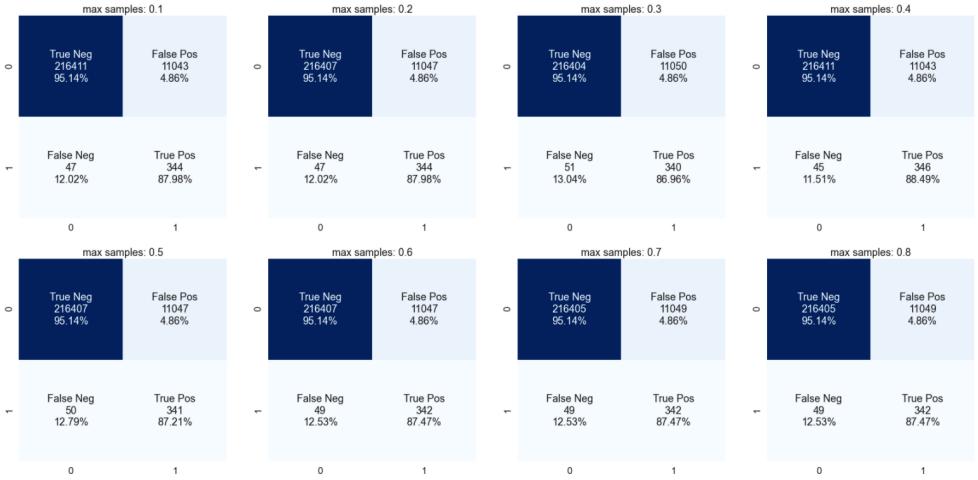
Hyperparameter Tuning Fei Tony Liu and Kai Ming Ting in their paper *Isolation-based Anomaly Detection*[8] claim that the Isolation Forest is pretty robust with respect to its parameter values other than the contamination factor. We find similar results in our results from the training data.

- the parameter *n_estimators* corresponds to the number of regressor trees we will be using in our model. We can see that increasing the number of trees after 26 is not bringing about any significant improvement in our recall score.

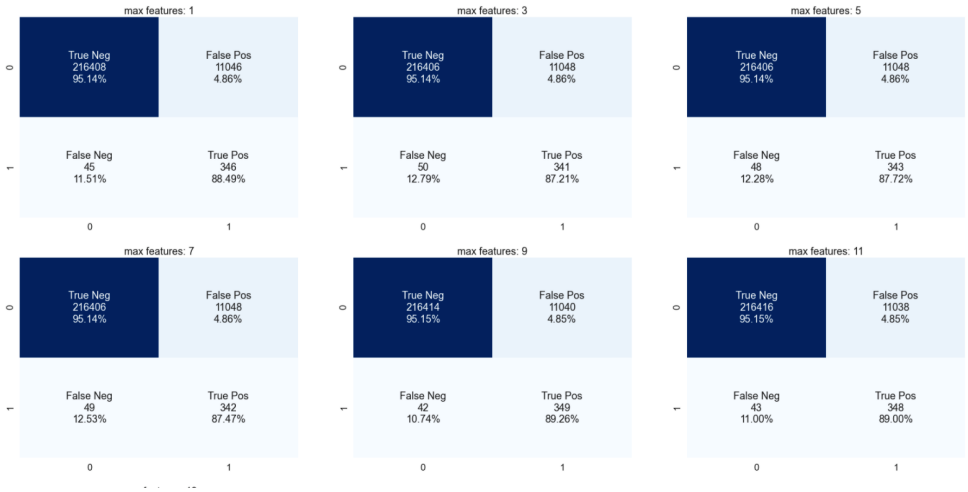
[8]Liu, F. T., Ting, K. M., amp; Zhou, Z. (2012). Isolation-based anomaly detection. ACM Transactions on Knowledge Discovery from Data, 6(1), 1-39. doi:10.1145/2133360.2133363



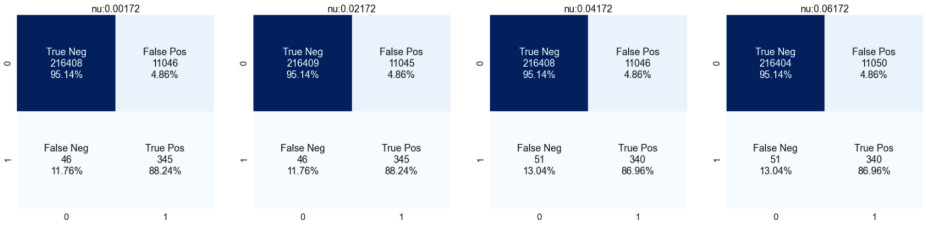
– the parameter *max_samples* corresponds with the percentage of data to sample when building our regression trees. We will try picking sample size corresponding to 10-100% of the training data with replacement. As we can see, the model is pretty robust to the *max_samples* parameter with no meaningful change to the recall rate.



– the parameter $max\_features$ corresponds to the number of features to draw from our training data to train each base estimator. As we can see, the model is pretty robust to the $max\_features$ parameter with no meaningful change beyond 9 maximum features to the recall rate

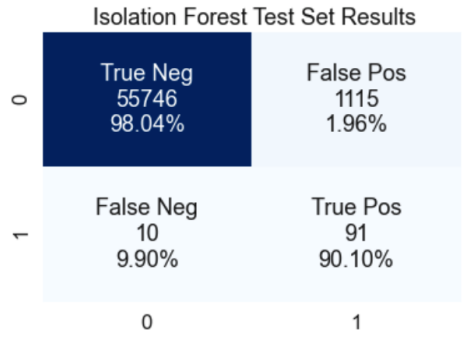


– the parameter *contamination* corresponds to the ratio of outliers in the data. We will start from the actual proportion of outliers in the data and continue to increase the contamination factor. As we can see a *contamination* factor in the range of 0.001-0.02 gives us the best results.



With the following parameter list Isolation forest, we are able to achieve a recall rate of 0.9 on our test set, with the lowest misclassification rate 1.96% of misclassifying normal transactions as fraud.

```
{'bootstrap': True,
 'contamination': 0.02,
 'max_features': 13,
 'max_samples': 1.0,
 'n_estimators': 100,
 'n_jobs': None,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```



Findings
Following are our best performing models along with their respective recall rate on the Test Dataset.

Logistic Regression	0.93
Isolation Forest	0.90

Even though Logistic Regression seems to have a higher recall rate than Isolation Forest, we saw that the recall rate was accompanied with a high misclassification rate of 26% of normal as fraudulent transactions due to its linear decision boundary, compared with Isolation Forest's rate of 1.96%. Moreover, we also observed that Logistic Regression results were highly dependent on the specific distribution of the test dataset. This was not the case with Isolation Forest which was our most robust model in our testing, in addition to being the most computationally efficient due to it not using distance based metrics and directly identifying outliers. Computational efficiency becomes especially important when deploying these algorithms on a large scale to catch credit card fraud in real time. For these reasons, we recommend the use of Isolation Forest for credit card fraud detection amongst all the models we tested.

0.5 Python Packages Used

from string import ascii_letters
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model$_s electionimporttrain_t est_s plit$
from sklearn.preprocessing import StandardScaler
from sklearn.linear$_m odelimportLassoCV$
from sklearn.feature$_s electionimportSequentialFeatureSelector$
from sklearn.feature$_s electionimportSelectFromModel$
from sklearn.ensemble import RandomForestRegressor
import matplotlib.gridspec as gridspec
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import KFold
from sklearn.metrics import recall_score
from sklearn.manifold import TSNE
from sklearn.metrics import precision$_r ecall_c urve$
import matplotlib.patheffects as PathEffects
import torch
import seaborn as sns
sns.set_style('darkgrid')
sns.set_palette('muted')
sns.set_context("notebook", font_scale=1.5,
rc="lines.linewidth": 2.5)
from sklearn.manifold import Isomap
from torch import nn, optim
import torch.nn.functional as F
import torch.utils.data as data_utils
import torchvision
from torchvision import transforms
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import pylab as pl
from sklearn.svm import OneClassSVM
from sklearn.metrics import f1_score
from sklearn.ensemble import IsolationForest
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression