Credit Card Fraud Detection

MA902 Task : Project Essay

by Yaswanth
Supervisor: Vernitski Alexei
Department:Mathematics
University: University of Essex
April 29, 2021

Contents

1 Abstract

A key challenge for a credit card company is to be able to detect a fraud transaction, but due to imbalance and skewness in data it is very difficult to detect them. Inner card fraud and external card fraud are the two forms of credit card fraud. Inner card fraud is usually the result of collusion between merchants and cardholders who use false transactions to defraud banks. External card fraud is characterised by the use of a stolen, forged, or counterfeit credit card to make purchases, or the use of cards to obtain cash in concealed ways, such as purchasing costly, small-quantity goods or goods that can be easily converted to cash [10]. So, the main objective of this research project is to provide solution for detecting fraudulent transactions with the help of Machine Learning and Data Science by using Resampling techniques. Our model pipeline primarily consists of five tasks; Data Collection, Data Analysis & Pre-processing, Resampling, Model Building and finally Model Evaluation.

2 Introduction

Online transactions have increased as a result of digitization. Credit cards are one way of doing online money transaction, but billions of dollars of loss is caused every year by fraudulent credit card transactions. Any credit card fraud detection system's main goal is to identify unusual activities and report them, while allowing regular transactions to be processed automatically. Financial institutions have been entrusting this role to rule-based systems that use expert-written rule sets for years. However, they are gradually turning to a machine learning approach, which can significantly enhance the process. Machine learning tools have higher precision and return more relevant results than rule-based solutions because they consider several additional variables [2]. There are two types of machine learning methods used in fraud detection: supervised and unsupervised learning. A model that is trained on labelled data and learns from previous examples is said to be supervised learning. Unsupervised Learning is a machine learning methodology in which the model does not need the users' supervision. Instead, it allows the model to work independently to discover previously undetected trends and knowledge. It is primarily concerned with unlabeled data. We will be using Supervised learning here as we have labelled data. The design of efficient and effective fraud detection algorithms or models is key for reducing the fraud losses. So, we will build a pipeline with four different algorithms and compare them with evaluation metrics to see which are working better. We will be using python for building our Machine Learning models. Python's simplicity and stability, as well as access to excellent libraries and frameworks for Artificial Intelligence and machine learning (ML), platform independence, and a large community, make it the best choice for machine learning and Data science projects.

3 Methodology

Here we will look into data collection, description, analysis and pre-processing (distributions, correlations), resampling methods (Under-sampling, Over-sampling), Model building (Data splitting, Hyper-parameter tuning, Random Search Cross Validation) and Model Evaluation with metrics like Accuracy, Precision, Recall and F1-score. As we are using python for this project we first need to download all the required libraries for our project (code in Appendix: line 1-30).

3.1 Data Collection

Data scientists need vast quantities of data in order to implement and create new research ideas. However, important business data is not always publicly accessible. So, we get our data from Kaggle in CSV format (www.kaggle.com/mlg-ulb/creditcardfraud) (code in Appendix: line 33). Kaggle is a web-based data-science environment, where users can find and publish data sets, explore and create models.

3.2 Data Description

The data includes credit card purchases made by European cardholders in September 2013. This data set contains 492 frauds out of 284,807 transactions that occurred over the course of two days. The data set is extremely unbalanced, with frauds accounting for 0.172 percent of all transactions.

It only has numerical input variables that have been transformed using Principle Component Analysis (PCA). PCA is a dimensionality reduction technique and it extracts dominant patterns/features from the data [13]. We do not have original features or further background details about the data due to confidentiality concerns. The principal components obtained with PCA are features V1, V2,... V28; the only features not transformed with PCA are 'Time' and 'Amount. The seconds elapsed between each transaction and the first transaction in the dataset are stored in the feature

'Time'. The transaction Amount is represented by the function 'Amount', which can be used for example-dependent cost-sensitive learning. The response variable is called 'Class,' and it has a value of 1 when there is fraud and 0 when there is not. [1]

Attributes	Type	Description
V1, V2,... V28	Numerical	Principal components obtained with PCA
Time	Numerical	Seconds elapsed between each transaction and the first transaction in the data
Amount	Numerical	Transaction Amount
Class	Numerical	Response Variable to determine if a transaction is fraud

Table 1: Data Description

3.3 Data Analysis & Pre-processing

Data Pre-processing is very important step in building a Machine Learning Model as the data in the real world is quite dirty. It has inconsistencies, noise, duplicate & missing values and many more. Fortunately, there are no duplicate & null values in our data (code in Appendix: line 36).

3.3.1 Distributions

From figure 1 (code in appendix: line 43-46) we can clearly see that most of the transactions are non-fraud. We could get a lot of errors if we use this data as the basis for our predictive models, and our algorithms would probably over-fit because it will "assume" that most transactions are not fraudulent. But we don't want our model to make assumptions; we want it to spot trends that indicate fraud! So we will be using resampling techniques like under-sampling and over-sampling for solving this problem. I will explain more about it in the next pages.

One of the most important steps in analysis is to understand the distributions of the features and checking for outliers. But we do not remove outliers here, as they are fraud values, not errors in data. In order to implement a PCA transformation, the features need to be previously scaled. So, as we know that the features from V1 to V28 are from PCA transformation, we do not need look at their distributions. From the distributions of

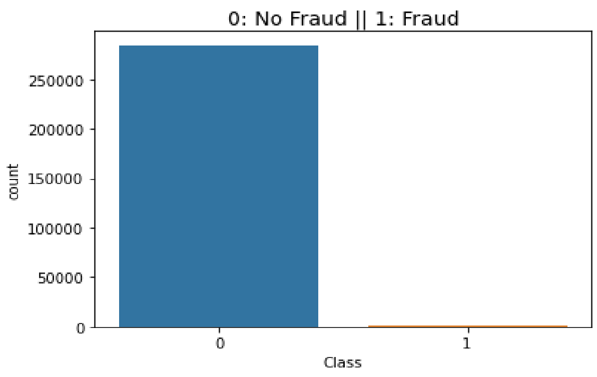

Figure 1: Class Distribution

transaction time and transaction amount from figure 2 (code in appendix: line 48-58) we can clearly see that they are skewed and the transaction time ranges from 0 to 160,000. This becomes a problem for many Machine Learning Algorithms except tree based ones, as they use euclidean distance in their computations. So, we need to standardize both time and amount(code in Appendix: line 60-64). We will be using Robust scaler in python for standardizing these two features, as it is robust to outliers.

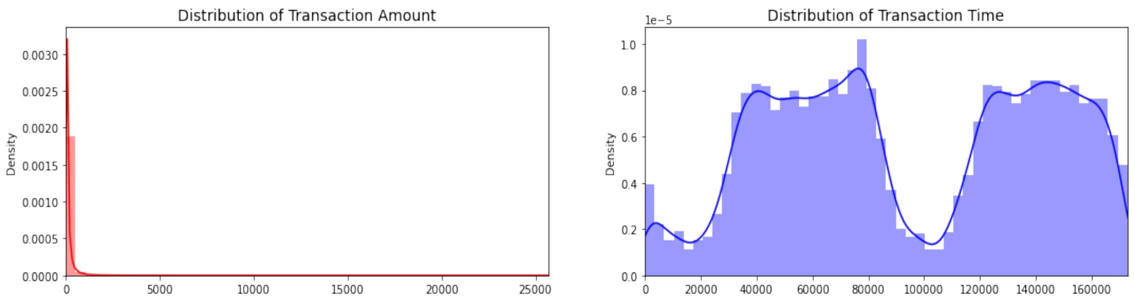

Figure 2: Time & Amount Distributions

3.3.2 Correlations

The degree to which two variables are related is measured by correlation. A correlation analysis can yield three outcomes: a positive correlation, a negative correlation, or no correlation. A positive correlation is when two variables shift in the same direction. A negative correlation is a relationship between two variables in which an increase in one variable causes the other to decrease. When there is no relationship between variables, it is called a zero correlation. In Figures 3, 4 and 5 we can see the Pearson correlation heat maps of actual data, under-sampled data and over-sampled data respectively. Having high correlations and multi-collinearity between variables is a problem [9] as it weakens the predictive strength of your model by reducing the accuracy of the estimate coefficients.

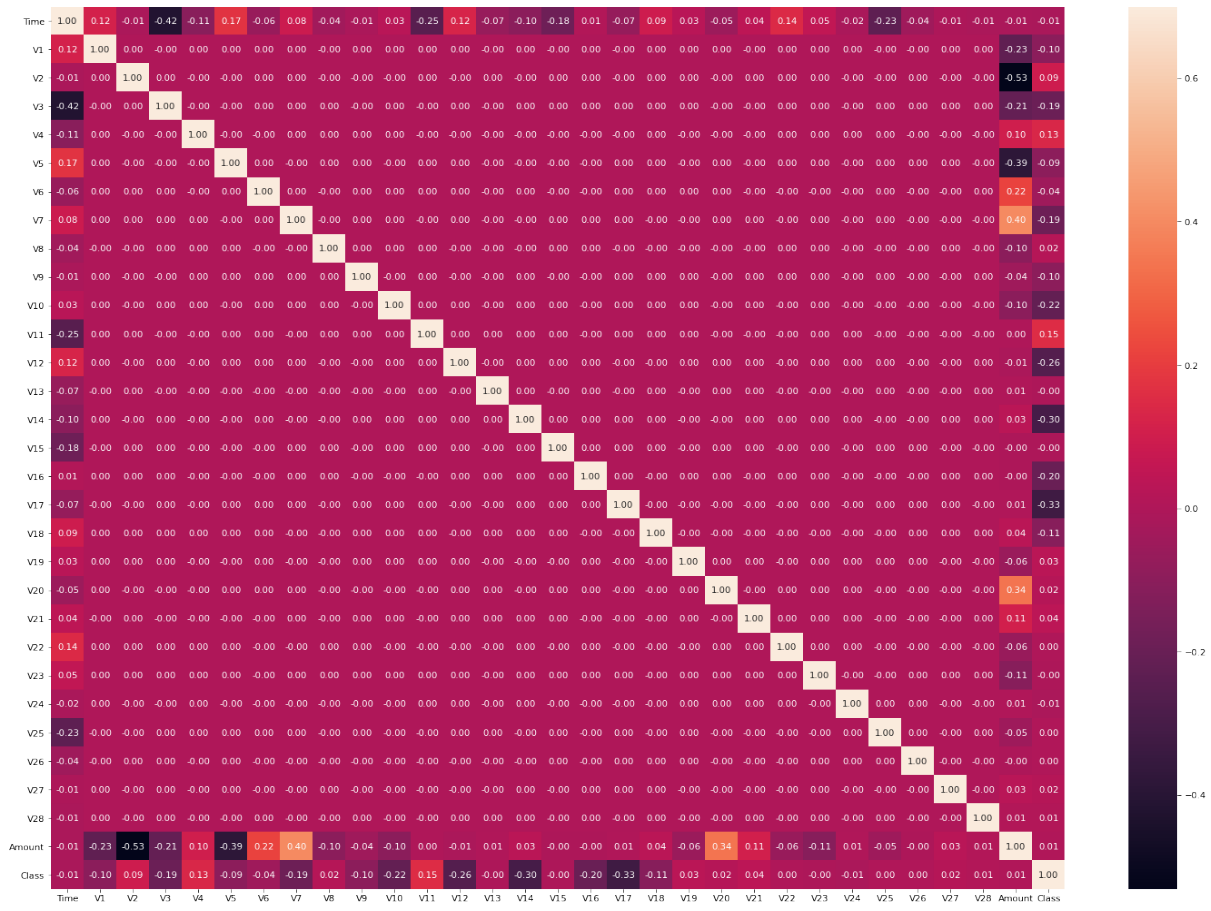

Figure 3: Correlation Heat Map of Actual Data

3.4 Resampling

Resampling is the process of generating a new transformed version of the training dataset with a different class distribution for the selected examples. Over-sampling and Under-sampling are the two major methods to random resample for imbalanced classification [2]. As you can see from figure 3 (code in appendix: line 38-41) there are no high correlations between the features. So, we are not dropping any features when giving the data to the models for predicting the output.

3.4.1 Under-sampling

In order to better align the class distribution, undersampling technique exclude examples from the training dataset that belong to the majority class, such as reducing the skew from imbalanced distribution to 1:1 class distribution. In python we use NearMiss for Under-sampling the data. After performing under-sampling we will look at correlations between features. As you can see from figure 4 (code in appendix: line 108-111) we have high correlations between many of the features from V1 to V18. So, we are dropping features (code in appendix: line 153) that have correlations greater than 0.7 and comparing the evaluation metrics of the models.

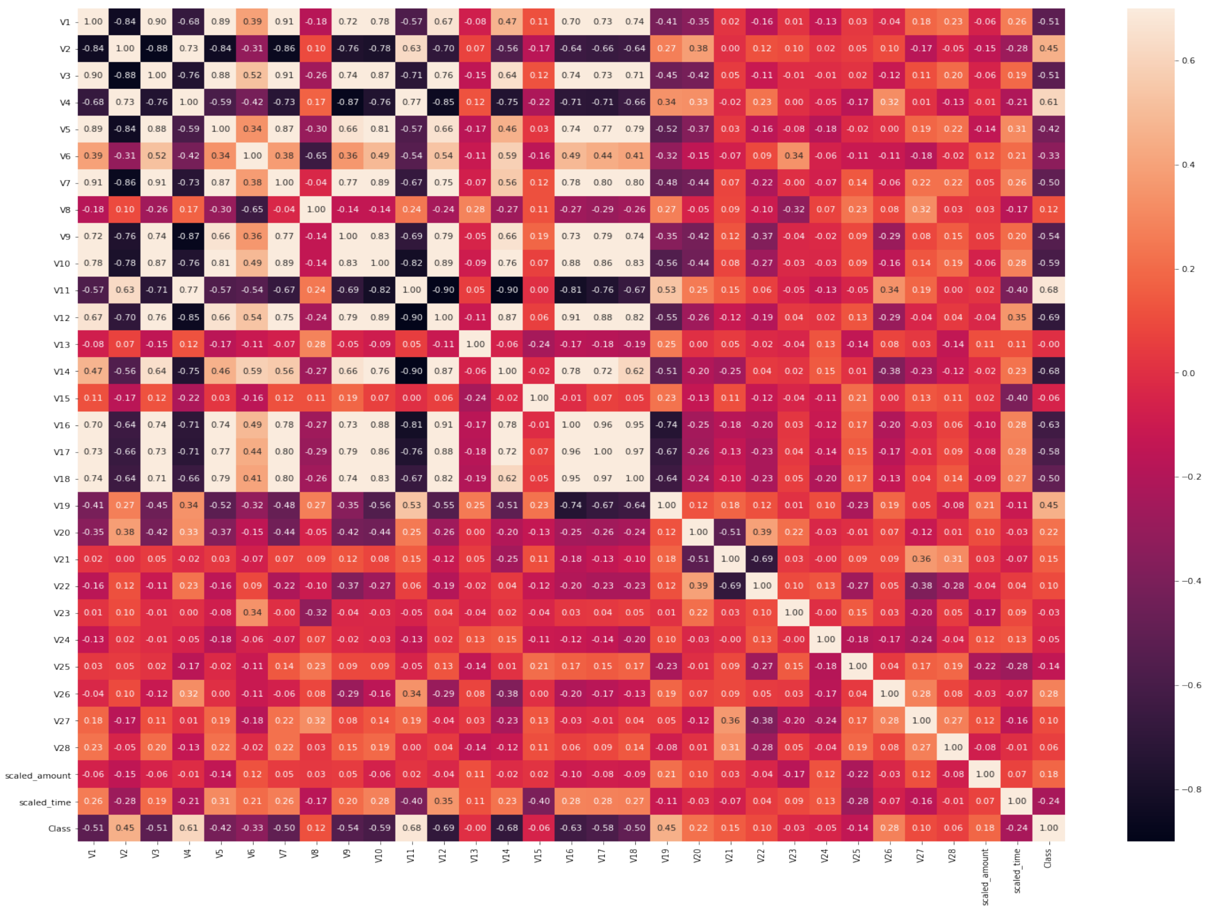

Figure 4: Correlation Heat Map of Under-Sampled Data

3.4.2 Over-sampling

Random oversampling is the process of randomly selecting and replacing examples from the minority class and adding them to the training data set to get 1:1 class distribution [14]. In python we use Synthetic minority over-sampling technique (SMOTE) for oversampling the data. This algorithm helps to overcome the over-fitting problem posed by random oversampling [3]. As you can see from figure 5 (code in appendix: line 118-121) we have high correlations between many of the features from V1 to V18. So, we are dropping features(code in appendix: line 156) that have correlations greater than 0.7 and comparing the evaluation metrics of the models.

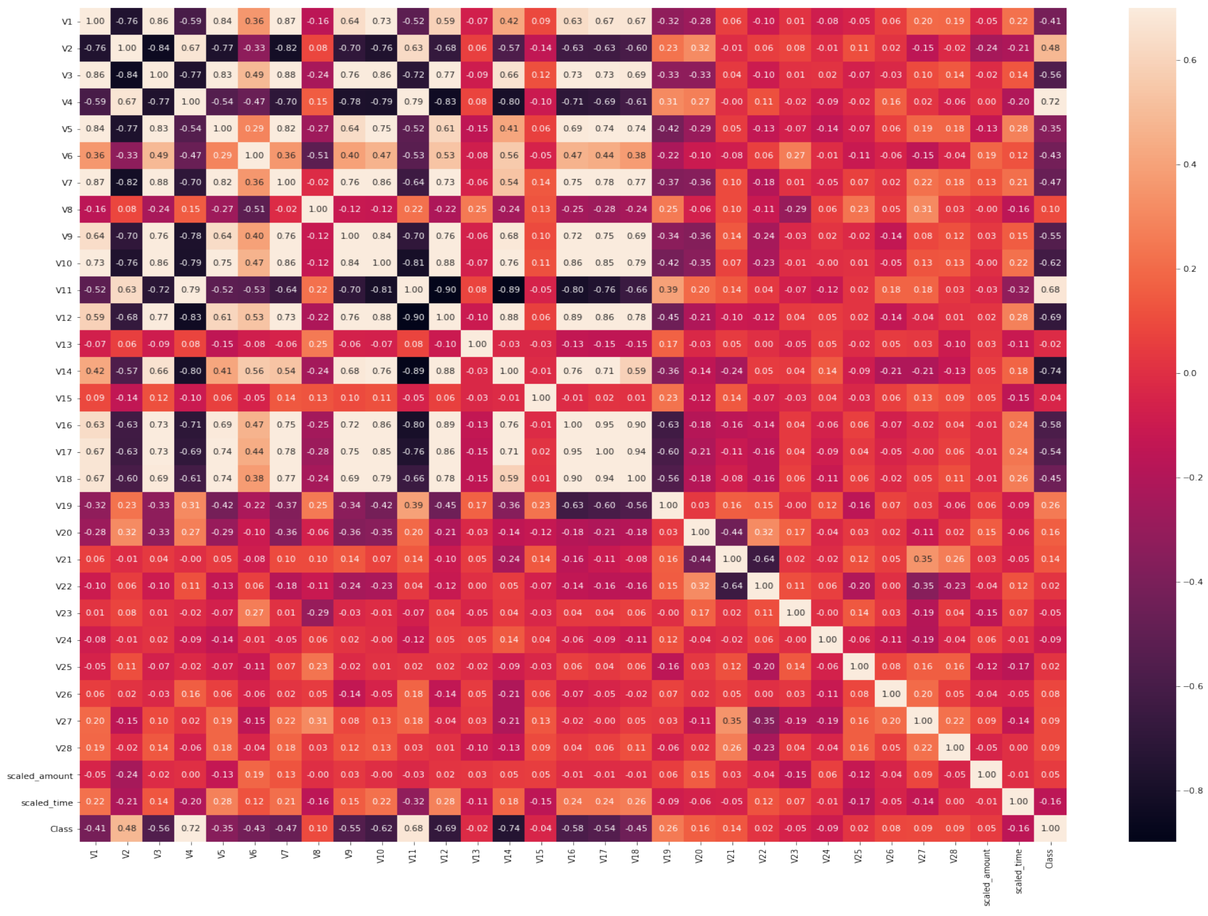

Figure 5: Correlation Heat Map of Over-Sampled Data

3.5 Model Building

In this step we will split the data into train and test, then fit the train data with four different models(Logistic Regression, K Nearest Neighbour classifier, Support Vector Classifier, Extreme Gradient boost/XG boost). We will use Random search cross validation for hyper-parameter tuning the models, and finally predict/classify the 'Class' feature. We will also know more about the machine learning algorithms that we are using here.

3.5.1 Splitting the data

In machine learning, splitting the data into training and testing sets is common practice [8]. We do this because attempting to test your models using data that you have previously trained on is unrealistic. The whole idea of a machine learning model is to be able to predict the data that hasn't been used/seen before. We need to split the data first and then use under-sampling or over-sampling method. This eliminates leaking the information from test set into the training set. In python we do the splitting with 'train_test_split' function present in scikit-learn package.

3.5.2 Hyper-Parameter Tuning

When building an ML model, we will be given design options on how to describe the architecture of our model. In typical machine learning fashion, we will ask the machine to conduct this exploration and automatically pick the best model architecture. The parameters that characterise the model architecture are known as hyper-parameters, and the process of finding the ideal model architecture is known as hyper-parameter tuning. We will be using Random Search Cross Validation for selecting the right hyper parameters. (code in appendix: lines 158-205)

3.5.3 Random Search Cross Validation

Random search is the most effective way to find an optimal set of hyper-parameters for a machine learning model. It draws randomly from a collection of hyper parameter distributions to train and test a series of models. After training N different versions of the model with different randomly chosen hyper-parameter combinations, the algorithm selects the most efficient version of the model, leaving you with a model trained on a near-optimal set of hyper-parameters.

3.5.4 Logistic Regression

This is a technique that can be used for traditional statistics as well as Machine Learning. It is one of the most basic Machine Learning algorithms and despite it's name it is used for classification. It can be expressed as

$$log(P(X)/(1 - P(X))) = \beta_0 + \beta_1 X$$

With the exception of the response variable being binomial, the technique is very similar to multiple linear regression. It is used to predict the likelihood of a specific class or event. Training a model with this algorithm does not require a lot of computing power [11]. But it requires the features to be independent and have no multi co-linearity among them. It also assumes linearity of independent variables.

3.5.5 K Nearest Neighbour classifier

It classifies the new data based on the similarity measures of the available data. The similarity measure corresponds to any functions such as distance. Classification is done depending on the majority of data points in the neighbourhood. Euclidean distance function is used to measure the distance between the data points. [7]

$$D = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Where D is the distance function and p & q are the data points, respectively.

3.5.6 Support Vector Machine Classifier (SVM)

It is a supervised machine learning algorithm that can be used for classification and regression. The concept behind Support Vector Machine's is to find a hyper-plane that divides a data set into two groups as best as possible. It usually works better than Logistic Regression as it uses 'kernel trick'. By projecting data into a higher dimension, it converts linearly inseparable data into linearly separable data. But it is less effective with noisier data set and requires large amount of memory and computational time [15]

3.5.7 Extreme Gradient boost Classifier

It is a tree based ensemble model that uses gradient boosting. In boosting the models are designed in a sequential order, minimising previous model errors, while increasing the effect of high-performance models. To reduce errors in sequential models, gradient boosting uses the gradient descent algorithm. In most of the cases ensemble models like boosting and bagging works way better than a single model [6] So, we are using this model to get better accuracy, but it takes lot of time for computation.

3.6 Model Evaluation

This is the final step in the Machine Learning pipeline where we use classification performance metrics like Accuracy, Precision, Recall and F1 score for evaluating our models. For getting these metrics confusion matrix is very useful. It gives the number of false positives (Incorrectly Classified Fraud Transactions), false negatives (Incorrectly Classified Non-Fraud Transactions), true positives (Correctly Classified Fraud Transactions), and true negatives (Correctly Classified Non-Fraud Transactions) [5].

		Predicted	
		Yes	No
Actual	Yes	True Positive	False Negative
	No	False Positive	True Negative

Table 2: Confusion Matrix

(1) Accuracy indicates how often the classifier is correct.

$$\text{Accuracy} = \frac{\text{TruePositive} + \text{TrueNegative}}{Total} \tag{1}$$

(2) Recall is the ratio of true positives to actual results. It is the amount of fraud cases our model is able to detect.

$$Recall/Sensitivity = \frac{TruePositive}{TruePositive + FalseNegative} \tag{2}$$

(3) Precision is the ratio of true positives to predicted results. As the name says, it gives how precise our model is in detecting fraud transactions

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \quad (3)$$

(4) Comparing two models with low precision and high recall, or vice versa, is difficult. So, we use F1 score. It is a tool for assessing both recall and precision at the same time.

$$F_1 = \frac{\text{2TruePositive}}{\text{2TruePositive + FalsePositive + FalseNegative}} \quad (4)$$

4 Experiments

In this step we will fit the train data without and with resampling (Over sampling & Under sampling) with four different models by dropping & not dropping the correlated variables and use actual test data for evaluating them with metrics accuracy, precision, recall and F1-score. Below tables show the results of models with all the scenarios. (code in appendix: lines 87-102)

Model	Accuracy	Precision	recall	F1Score
Logistic Regression	61.37	55	40.7	32.6
KNN	54	51.4	56.7	54.4
SVM	59.43	58.32	54.81	55.83
XG boost	60.34	54.32	58.32	58.45

Table 3: Evaluation metrics of models with actual data

Model	Accuracy	Precision	recall	F1Score
Logistic Regression	83	62	86	72.1
KNN	73.6	32.3	72.81	44.31
SVM	84.3	58.3	83.5	68.82
XG boost	72.2	33.2	80.32	46.71

Table 4: Evaluation metrics of models trained on under-sampled data including correlated features

Model	Accuracy	Precision	recall	F1Score
Logistic Regression	85	63	88	73.4
KNN	74.6	40.3	73.81	51.68
SVM	88.3	60.3	84.5	70
XG boost	75.2	34.2	82.32	48.81

Table 5: Evaluation metrics of models trained on under-sampled data (correlated features dropped)

Model	Accuracy	Precision	recall	F1Score
Logistic Regression	74	68	70	69
KNN	97	88.2	84	85.7
SVM	76	67	69	67.9
XG boost	98	99	94.3	94.4

Table 6: Evaluation metrics of models trained on over-sampled data including correlated features

Model	Accuracy	Precision	recall	F1Score
Logistic Regression	75	69.8	71.7	70.5
KNN	98	88.7	87.8	88.3
SVM	77	68.6	70.6	69.34
XG boost	99	99	94.3	96.45

Table 7: Evaluation metrics of models trained on over-sampled data (correlated features dropped)

5 Conclusion

A comparative study has been made with four different classifiers/models with over sampling & under sampling with correlated features, over sampling & under sampling without correlated features, and with actual data. With actual data the evaluation metrics are very less which is to be expected due to data imbalance. With under-sampled data we can see that the precision is very less, which means there are high number of false positives. So, our model is unable to detect for a large number of non fraud transactions correctly and instead, miss classifies those non fraud transactions as fraud cases. Imagine that people that were making regular purchases got their card blocked due to the reason that our model classified that transaction as a fraud transaction, this will be a huge disadvantage for the financial institution. The number of customer complaints and customer dissatisfaction will increase. This might be due to over fitting as the under-sampled data is very less for the model to train and it discards potentially useful data.

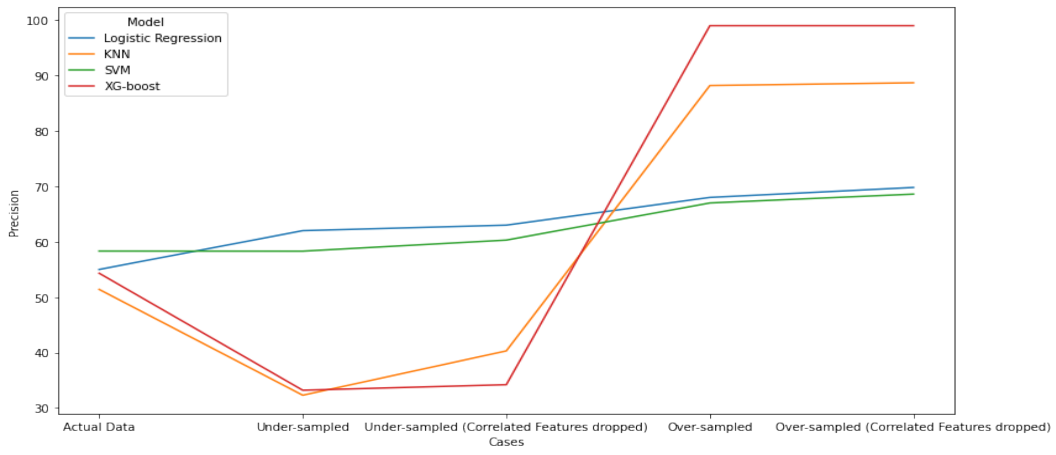

Figure 6: Precision

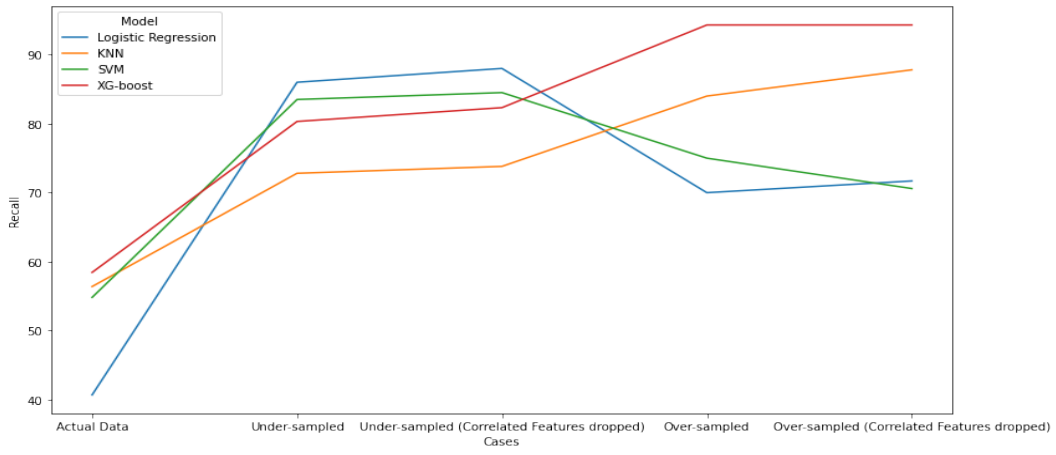

Figure 7: Recall

Logistic Regression and SVM are getting better results with under-sampled data as they work better with less data. We can clearly see that the over sampling is giving good precision and recall values for KNN and XG boost. Overall, we are getting good results with over sampled data and with XG boost as it is an ensemble technique, but it takes a lot of time for execution. In general Under sampling works better with large data (millions of rows) and over sampling works better with medium and small data [12]. This is what we are seeing in our case.

6 Further work

We can further improve the models with deep neural networks, but similar to XG boost it takes a lot of time to execute them [4]. If we have more information about the data instead of features from PCA, we can do more analysis and perform feature engineering based on their dependency with the class variable. Also, having additional features like location might help us get a better understanding of the data.

References

[1] Kaggle credit card fraud detection.

[2] Siddhartha Bhattacharyya, Sanjeev Jha, Kurian Tharakunnel, and J. Christopher Westland. Data mining for credit card fraud: A comparative study. *Decision Support Systems*, 50(3):602–613, 2011. On quantitative methods for detection of financial fraud.

[3] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[4] Sushmito Ghosh and Douglas L Reilly. Credit card fraud detection with a neural-network. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, volume 3, pages 621–630. IEEE, 1994.

[5] Amalia Luque, Alejandro Carrasco, Alejandro Martín, and Ana de las Heras. The impact of class imbalance in classification performance metrics based on the binary confusion matrix. *Pattern Recognition*, 91:216–231, 2019.

[6] Ahmedbahaaaldin Ibrahem Ahmed Osman, Ali Najah Ahmed, Ming Fai Chow, Yuk Feng Huang, and Ahmed El-Shafie. Extreme gradient boosting (xgboost) model to predict the groundwater levels in selangor malaysia. *Ain Shams Engineering Journal*, 2021.

[7] J Pamina, Beschi Raja, S SathyaBama, MS Sruthi, Aiswaryadevi VJ, et al. An effective classifier for predicting churn in telecommunication. *Jour of Adv Research in Dynamical & Control Systems*, 11, 2019.

[8] Richard R Picard and Kenneth N Berk. Data splitting. *The American Statistician*, 44(2):140–147, 1990.

[9] Mary Ann Schroeder, Janice Lander, and Stacey Levine-Silverman. Diagnosing and dealing with multicollinearity. *Western journal of nursing research*, 12(2):175–187, 1990.

[10] Aihua Shen, Rencheng Tong, and Yaochen Deng. Application of classification models on credit card fraud detection. In *2007 International Conference on Service Systems and Service Management*, pages 1–4, 2007.

[11] Jack V Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology*, 49(11):1225–1231, 1996.

[12] Gary M Weiss, Kate McCarthy, and Bibi Zabar. Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs? *Dmin*, 7(35-41):24, 2007.

[13] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[14] Bee Wah Yap, Khatijahhusna Abd Rani, Hezlin Aryani Abd Rahman, Simon Fong, Zuraida Khairudin, and Nik Nik Abdullah. An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets. In *Proceedings of the first international conference on advanced data and information engineering (DaEng-2013)*, pages 13–22. Springer, 2014.

[15] Jian-Pei Zhang, Zhong-Wei Li, and Jing Yang. A parallel svm training algorithm on large-scale classification problems. In *2005 International Conference on Machine Learning and Cybernetics*, volume 3, pages 1637–1641. IEEE, 2005.

A Appendix

```python
# Importing required Libraries
# For reading the data
import pandas as pd
# For plotting the graphs
import seaborn as sns
import matplotlib.pyplot as plt
# for scaling the data
from sklearn.preprocessing import RobustScaler
# Machine Learning Algorithms
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
import xgboost
# For Evaluation Metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.metrics import precision_score,recall_score,
    f1_score
from sklearn import metrics
# For Splitting the data
from sklearn.model_selection import train_test_split
# Cross Validation
from sklearn.model_selection import RandomizedSearchCV
# For under-sampling
from imblearn.under_sampling import NearMiss
# For over sampling
from imblearn.over_sampling import RandomOverSampler
# For removing warnings
import warnings
warnings.filterwarnings("ignore")

# Reading the data
data = pd.read_csv('creditcard.csv')

# For checking null values
data.info() # Fortunately there are no null values

# Correlation plot
plt.figure(figsize=(25,20))
sns.heatmap(data.corr(),vmax=.7,cbar=True,fmt=".2f",annot=True)
plt.savefig('correlation actual.png', bbox_inches="tight")

# Class Distribution plot
sns.countplot('Class', data=data)
plt.title('0: No Fraud || 1: Fraud', fontsize=14)
plt.savefig('Class Distribution.png', bbox_inches="tight")

# Time and Amount distribution plots
```

```python
fig, ax = plt.subplots(1, 2, figsize=(18,4))
amount_val = data['Amount'].values
time_val = data['Time'].values
sns.distplot(amount_val, ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize
    =14)
ax[0].set_xlim([min(amount_val), max(amount_val)])
sns.distplot(time_val, ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Time', fontsize=14)
ax[1].set_xlim([min(time_val), max(time_val)])
plt.savefig('Distributions.png', bbox_inches="tight")

# Scaling the data
rob_scaler = RobustScaler()
data['scaled_amount'] = rob_scaler.fit_transform(data['Amount'].
    values.reshape(-1,1))
data['scaled_time'] = rob_scaler.fit_transform(data['Time'].
    values.reshape(-1,1))
data.drop(['Time','Amount'], axis=1, inplace=True)

# Dividing target and independant variables
X = data.drop(['Class'],axis=1)
y = data['Class']

# Model Building
# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)

# KNN
classifier1 = KNeighborsClassifier(n_neighbors = 1)
classifier1.fit(X_train,y_train)
# XG boost
classifier2 = xgboost.XGBClassifier()
classifier2.fit(X_train,y_train)
# SVM Classifier
classifier3 = SVC()
classifier3.fit(X_train,y_train)
# Logistic Regression
classifier4 =LogisticRegression()
classifier4.fit(X_train,y_train)

# We predict the test data with all the models
# Predicting the output
y_pred = classifier1.predict(X_test)

# We will get evaluation metrics for all the ML models
# Evaluation metrics
# Confusion matrix
```

```
print(confusion_matrix(y_test, y_pred))
# Accuracy
print('Accuracy:', metrics.accuracy_score(y_test, y_pred))
# F1 score
print("F1 score:",round(f1_score(y_test,y_pred,average='weighted
    '), ndigits = 3))
# Precision
print("precision score:",round(precision_score(y_test,y_pred,
    average='weighted'), ndigits = 3))
# Recall
print("recall score:",round(recall_score(y_test, y_pred,average=
    'weighted'), ndigits = 3))

# Implementing Undersampling for Handling Imbalanced
nm = NearMiss()
X_und,y_und=nm.fit_sample(X_train, y_train)
data1 = pd.concat([X_und, y_und], axis=1)
# Correlation plot for Under-sampled data
plt.figure(figsize=(25,20))
sns.heatmap(data1.corr(),vmax=.7,cbar=True,fmt=".2f",annot=True)
plt.savefig('correlation Under sample.png', bbox_inches="tight")

# Implementing Oversampling for Handling Imbalanced
os =  RandomOverSampler(random_state=42)
X_res, y_res = os.fit_sample(X_train, y_train)
data2 = pd.concat([X_res, y_res], axis=1)

# Correlation plot for Over-sampled data
plt.figure(figsize=(25,20))
sns.heatmap(data2.corr(),vmax=.7,cbar=True,fmt=".2f",annot=True)
plt.savefig('correlation Over sample.png', bbox_inches="tight")

# Under-sampling model building
# KNN
classifier1 = KNeighborsClassifier(n_neighbors = 1)
classifier1.fit(X_und,y_und)
# XG boost
classifier2 = xgboost.XGBClassifier()
classifier2.fit(X_und,y_und)
# SVM Classifier
classifier3 = SVC()
classifier3.fit(X_und,y_und)
# Logistic Regression
classifier4 =LogisticRegression()
classifier4.fit(X_und,y_und)

# Over-sampling model building
# KNN
classifier1 = KNeighborsClassifier(n_neighbors = 1)
```

```python
classifier1.fit(X_res,y_res)
# XG boost
classifier2 = xgboost.XGBClassifier()
classifier2.fit(X_res,y_res)
# SVM Classifier
classifier3 = SVC()
classifier3.fit(X_res,y_res)
# Logistic Regression
classifier4 =LogisticRegression()
classifier4.fit(X_res,y_res)
# We find evaluation metrics similar to how we did with actual
    data. I am not mentioning the code again, as the report
    becomes too lengthy.

# dropping correlated features undersampling
data_undersampling = data.drop(['V1','V2','V3','V4','V5','V9','
    V10','V11','V12','V14','V16','V17','V18'], axis=1, inplace=
    True)

# dropping correlated features oversampling
data_oversampling=data.drop(['V1','V2','V3','V4','V5','V9','V10'
    ,'V11','V12','V14','V16','V17','V18'], axis=1, inplace=True)

# Hyper -parameter tuning

# SVM
classifier3 = SVC()
# Taking the Hyper parameters
parameters = [{'C': [10, 100, 1000,1500], 'kernel': ['rbf'],'
    gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]}]
# Applying Random Search to find the best parameters
random_cv =RandomizedSearchCV(estimator = classifier3,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           n_jobs = -1)
# Fitting the train data
random_cv.fit(X_train,y_train)
# Getting the best parameters
random_cv.best_estimator_

# XG Boost
classifier2 = xgboost.XGBClassifier()
# Taking the Hyper parameters
n_estimators = [50,100, 500, 900, 1100, 1500]
max_depth = [1,2, 3, 5, 10, 15]
booster=['gbtree','gblinear']
learning_rate=[0.05,0.1,0.15,0.20]
min_child_weight=[1,2,3,4]
```

```
base_score=[0.25,0.5,0.75,1]

# Define the grid of hyperparameters to search
hyperparameter_grid = {
    'n_estimators': n_estimators,
    'max_depth':max_depth,
    'learning_rate':learning_rate,
    'min_child_weight':min_child_weight,
    'booster':booster,
    'base_score':base_score
    }

random_cv = RandomizedSearchCV(estimator= classifier2,
            param_distributions=hyperparameter_grid,
            cv=5, n_iter=50,
            scoring = 'accuracy',n_jobs = 4,
            verbose = 5,
            return_train_score = True,
            random_state=42)
# fitting the train data
random_cv.fit(X_train,y_train)
# Finding the best parameters
random_cv.best_estimator_

# We build models with new data the and find evaluation metrics similar to how we found them with the actual data above.

# Precision and Recall Plots
# metrics of all the models with all the cases
data1 = {'Model':['Logistic Regression','KNN','SVM','XG-boost','Logistic Regression','KNN','SVM','XG-boost','Logistic Regression',
              'KNN','SVM','XG-boost','Logistic Regression','KNN','SVM','XG-boost','Logistic Regression','KNN','SVM','XG-boost'],
        'Cases':['Actual Data','Actual Data','Actual Data','Actual Data','Under-sampled','Under-sampled','Under-sampled',
             'Under-sampled','Under-sampled (Correlated Features dropped)','Under-sampled (Correlated Features dropped)',
             'Under-sampled (Correlated Features dropped)','Under-sampled (Correlated Features dropped)',
             'Over-sampled','Over-sampled','Over-sampled','Over-sampled',
             'Over-sampled (Correlated Features dropped)','Over-sampled (Correlated Features dropped)',
            'Over-sampled (Correlated Features dropped)','Over-sampled (Correlated Features dropped)'],
        'Precision':
```

```
    [55,51.4,58.32,54.32,62,32.3,58.3,33.2,63,40.3,60.3,
         34.2,68,88.2,67,99,69.8,88.7,68.6,99],
         'Recall':
    [40.7,56.4,54.83,58.45,86,72.81,83.5,80.32,88,73.81,
         84.5,82.32,70,84,75,94.3,71.7,87.8,70.6,94.3]
         }

# dataframe
df = pd.DataFrame (data1, columns = ['Model','Cases','Precision'
    ,'Recall'])

# Precision
plt.figure(figsize=(13,7))
sns.lineplot(x = 'Cases', y = 'Precision', hue = 'Model', data=
    df)
plt.savefig('Precision.png', bbox_inches="tight")

# Recall
plt.figure(figsize=(13,7))
sns.lineplot(x = 'Cases', y = 'Recall', hue = 'Model', data=df)
plt.savefig('Recall.png', bbox_inches="tight")
```
Listing 1: Code