Journal of Science and Technology
ISSN: 2456-5660 Volume 5, Issue 3, May-June 2020

www.jst.org.in *DOI: https://doi.org/10.46243/jst.2020.v5.i3.pp138-146*

Credit Card Fraud Detection Using Machine Learning Classification Algorithms over Highly Imbalanced Data

N.Adityasundar [1], T.SaiAbhigna [2], B.Lakshman[3], D.Phaneendra[4], N.MohanKumar[5]
[1,2,3,4,5]*(Information Technology, Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam, India*
[3]*Corresponding Author: lakshmanlaxman115@gmail.com*

To Cite this Article
N.Adityasundar , T.SaiAbhigna , B.Lakshman, D.Phaneendra and N.MohanKumar, "Credit Card Fraud Detection Using Machine Learning Classification Algorithms over Highly Imbalanced Data", Journal of Science and Technology, Vol. 05, Issue 03, May-June 2020, pp138-146

Article Info
Received: 10-02-2020 Revised: 09-05-2020 Accepted: 12-05-2020 Published: 14-05-2020

Abstract:*Most online customers use cards to pay for their purchases. As charge cards become the most mainstream strategy for installment, instances of misrepresentation relationship with it too increases. The primary goal of this venture is to be ready to perceive false exchanges from non-fake exchanges. In request to do so,primarily,data mining methods are utilized to examine the examples and attributes of deceitful and non-fake transactions.Then,machine learning systems are utilized to foresee the fake and non-fake exchanges automatically. Algorithms LR (Logistic Regression) is used. Therefore, the blend of AI and information mining procedures are utilized to distinguish the fake and non-fake exchanges by learning the examples of the information. Models are made utilizing these calculations and afterward precision,accuracy,recall are determined and an examination is made.*

Keywords: *Credit card, machine learning, fake, transactions.*

I. Introduction

Because of the ascent of E-commerce,there has been a gigantic utilization of charge cards for web based shopping which prompted a high measure of fakes identified with Mastercards. Mastercard misrepresentation had additionally quickened as on the web And disconnected transaction.The need to recognize charge card extortion is fundamental. There Are numerous extortion identification arrangements and programming which forestall cheats in organizations, for example mastercard, retail, web based business, protection, and enterprises.Cardholders ordinarily give the card number, expiry date, and card confirmation number through site or phone.Extortion discovery accept distinguishing misrepresentation as fast as conceivable once it has been committed.It likewise includes checking and dissecting the conduct of different clients so as to assess recognize or maintain a strategic distance from bothersome behaviour.There are various techniques to stop charge card misrepresentationexchanges however are not ready to stop them successfully. A few procedures are planned and actualized to settle Visa misrepresentation identification, for example, counterfeit neural system visit thing set mining, AI calculations, man-made consciousness and information mining.These calculations can separate exchanges which are fake or non-fraudulent.This paper assesses three AI calculation Logistic Regression with various solvers like sag,liblinear and so forth and after wardan examination is made to assess what model played out the best.

1.1 Problem Statement

With the development of internet business sites, individuals and money related organizations do their exchanges that have prompted an exponential increase in the credit card fakes. Deceitful charge card exchanges lead to lost a gigantic measure of cash. The plan of a compelling misrepresentation recognition framework is vital so as to lessen the misfortunes acquired by the clients and money related organizations .Research has been done on numerous models and techniques to forestall and recognize charge card fakes. Some Visa misrepresentation exchange datasets contain the issue of lopsidedness in datasets. A decent extortion discovery framework ought to

have the option to recognize the misrepresentation exchange precisely and should make the recognition conceivable continuously exchange.

1.2 Methodology and Approach

A brief procedure of designing the credit card fraud detection model is explained as follows:

1.2.1 input data: The input dataset is a comma separated values file(csv) containing the fraud dataset, which is highly imbalanced..

1.2.2 Preprocessing of input data: Input dataset is subject to various preprocessing techniques such as filling of missing values, encoding of categorical data and scaling of values in the appropriate range.The data is converted into usable format fit and sampled.

1.2.3 Building individual classifiers on the training dataset: The training dataset is fed to each of the independent base learners and the individual classifiers are built using the training dataset.

1.3 Dataset

The datasets contain transactions made by credit cards in September 2013 by european cardholders[9].This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.It contains only numeric input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset.The feature 'Amount' is the transaction Amount, this feature can be used for example dependant cost sensitive learning. Feature 'Classis the response variable and it takes value 1 in case of fraud and 0 otherwise. [1][2][3][4][5][6][7][8]

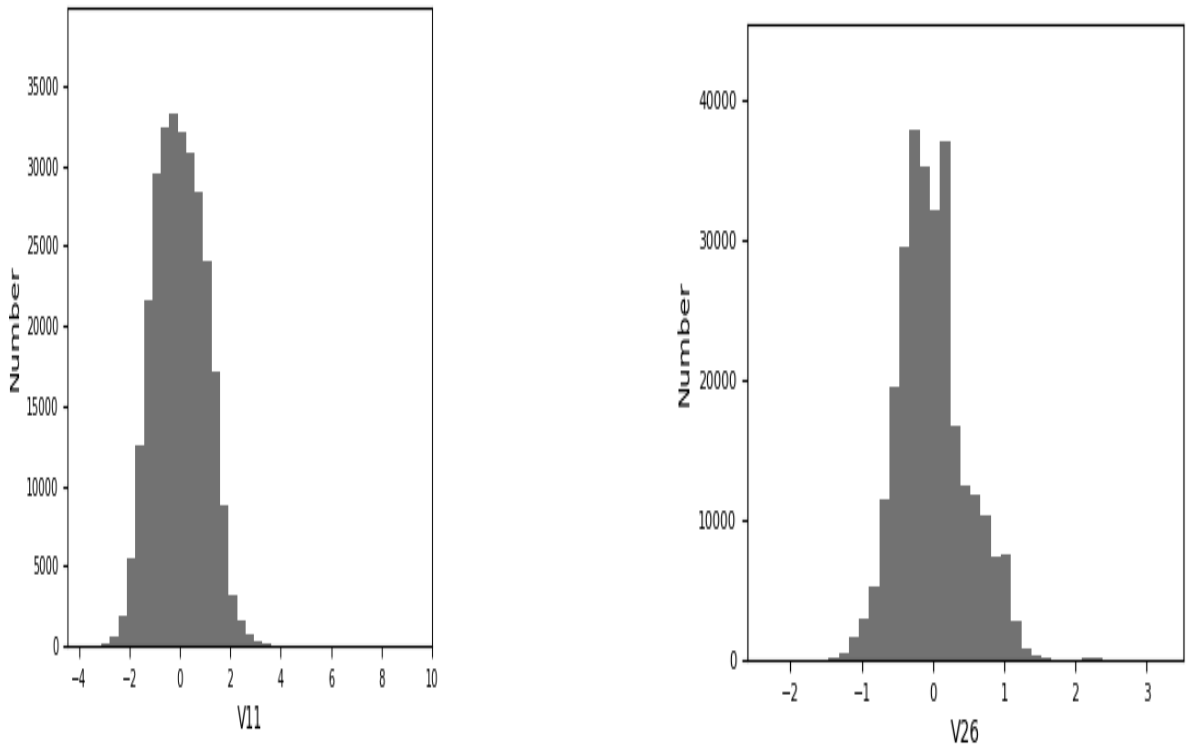

Fig 1 : Histogram of some attributes of dataset (v11,,v26)

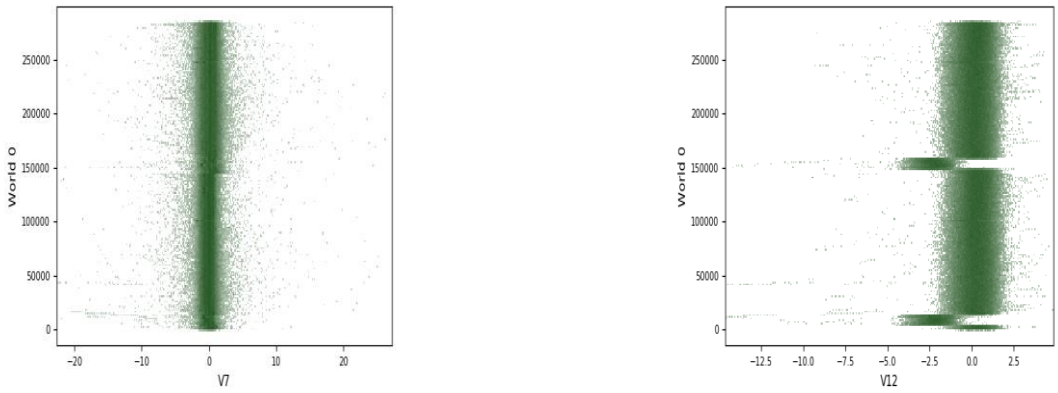

Fig 2 : 2D scatter plot of attributes (v7,v12)

```
''' a
```

	count	mean	std	min	25%	50%	75%	max
Time	284807.0	9.481386e+04	47488.145955	0.000000	54201.500000	84692.000000	139320.500000	172792.000000
V1	284807.0	3.919560e-15	1.958696	-56.407510	-0.920373	0.018109	1.315642	2.454930
V2	284807.0	5.688174e-16	1.651309	-72.715728	-0.598550	0.065486	0.803724	22.057729
V3	284807.0	-8.769071e-15	1.516255	-48.325589	-0.890365	0.179846	1.027196	9.382558
V4	284807.0	2.782312e-15	1.415869	-5.683171	-0.848640	-0.019847	0.743341	16.875344
V5	284807.0	-1.552563e-15	1.380247	-113.743307	-0.691597	-0.054336	0.611926	34.801666
V6	284807.0	2.010663e-15	1.332271	-26.160506	-0.768296	-0.274187	0.398565	73.301626
V7	284807.0	-1.694249e-15	1.237094	-43.557242	-0.554076	0.040103	0.570436	120.589494
V8	284807.0	-1.927028e-16	1.194353	-73.216718	-0.208630	0.022358	0.327346	20.007208
V9	284807.0	-3.137024e-15	1.098632	-13.434066	-0.643098	-0.051429	0.597139	15.594995
V10	284807.0	1.768627e-15	1.088850	-24.588262	-0.535426	-0.092917	0.453923	23.745136
V11	284807.0	9.170318e-16	1.020713	-4.797473	-0.762494	-0.032757	0.739593	12.018913
V12	284807.0	-1.810658e-15	0.999201	-18.683715	-0.405571	0.140033	0.618238	7.848392
V13	284807.0	1.693438e-15	0.995274	-5.791881	-0.648539	-0.013568	0.662505	7.126883
V14	284807.0	1.479045e-15	0.958596	-19.214325	-0.425574	0.050601	0.493150	10.526766
V15	284807.0	3.482336e-15	0.915316	-4.498945	-0.582884	0.048072	0.648821	8.877742
V16	284807.0	1.392007e-15	0.876253	-14.129855	-0.468037	0.066413	0.523296	17.315112
V17	284807.0	-7.528491e-16	0.849337	-25.162799	-0.483748	-0.065676	0.399675	9.253526
V18	284807.0	4.328772e-16	0.838176	-9.498746	-0.498850	-0.003636	0.500807	5.041069
V19	284807.0	9.049732e-16	0.814041	-7.213527	-0.456299	0.003735	0.458949	5.591971
V20	284807.0	5.085503e-16	0.770925	-54.497720	-0.211721	-0.062481	0.133041	39.420904
V21	284807.0	1.537294e-16	0.734524	-34.830382	-0.228395	-0.029450	0.186377	27.202839
V22	284807.0	7.959909e-16	0.725702	-10.933144	-0.542350	0.006782	0.528554	10.503090
V23	284807.0	5.367590e-16	0.624460	-44.807735	-0.161846	-0.011193	0.147642	22.528412
V24	284807.0	4.458112e-15	0.605647	-2.836627	-0.354586	0.040976	0.439527	4.584549
V25	284807.0	1.453003e-15	0.521278	-10.295397	-0.317145	0.016594	0.350716	7.519589
V26	284807.0	1.699104e-15	0.482227	-2.604551	-0.326984	-0.052139	0.240952	3.517346
V27	284807.0	-3.660161e-16	0.403632	-22.565679	-0.070840	0.001342	0.091045	31.612198
V28	284807.0	-1.206049e-16	0.330083	-15.430084	-0.052960	0.011244	0.078280	33.847808
Amount	284807.0	8.834962e+01	250.120109	0.000000	5.600000	22.000000	77.165000	25691.160000
Class	284807.0	1.727486e-03	0.041527	0.000000	0.000000	0.000000	0.000000	1.000000

Fig 3 :mean,standarddeviation,minimum and maximum metrics of all attributes in data set

1.4 Preprocessing Of Data:

1.4.1KernalCenterer:Let K(x, z) be a kernel defined by phi(x)^T phi(z), where phi is a function mapping x to a Hilbert space. KernelCenteredcenters (i.e., normalize to have zero mean) the data without explicitly computing phi(x).Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using transform[10].

```
>>> from sklearn.preprocessing import KernelCenterer
>>> transformer = KernelCenterer().fit(df)
>>> new=transformer.transform(df)
>>> new=pd.DataFrame(new,columns=df.columns)
>>> new.to_csv('KernelCenterer.csv')
```

Fig 4 :Code for KernelCentererPreprocessing

Pandas-pd :pandas module used to read a comma-separated values (csv) file into DataFrame. Also supports optionally iterating or breaking of the file into chunks[14].

	count	mean	std	min	25%	50%	75%	max
V1	284807.0	-3.046228	9.214576	-918.477906	-3.176693	-1.125779	0.381193	2.694104
V2	284807.0	-3.046228	9.556079	-914.779851	-3.028263	-0.817559	0.219516	28.897519
V3	284807.0	-3.046228	9.016595	-931.254956	-3.065453	-1.164596	0.131320	4.329543
V4	284807.0	-3.046228	8.549739	-867.625183	-3.160939	-1.230064	-0.080145	18.684803
V5	284807.0	-3.046228	9.198659	-996.672673	-2.994367	-1.012695	0.017311	34.499014
V6	284807.0	-3.046228	8.386416	-809.627741	-2.826410	-1.250113	-0.536249	13.313904
V7	284807.0	-3.046228	8.160233	-762.339873	-3.052961	-1.134778	-0.124143	11.413864
V8	284807.0	-3.046228	8.780359	-910.276727	-2.754432	-0.839884	-0.221713	21.456221
V9	284807.0	-3.046228	8.695125	-886.801791	-2.979185	-1.105368	-0.137209	10.111188
V10	284807.0	-3.046228	8.755183	-894.934854	-2.738764	-1.055938	-0.320013	16.250464
V11	284807.0	-3.046228	8.637773	-876.075470	-2.903618	-1.180444	-0.142519	14.695999
V12	284807.0	-3.046228	8.645054	-892.118785	-3.022819	-1.016902	-0.057602	6.843604
V13	284807.0	-3.046228	8.629851	-875.802484	-2.916870	-1.134169	-0.143659	5.853985
V14	284807.0	-3.046228	8.598860	-889.725308	-2.923037	-1.045487	-0.164804	9.274047
V15	284807.0	-3.046228	8.629607	-874.051625	-2.891526	-1.083396	-0.098552	5.169244
V16	284807.0	-3.046228	8.626524	-865.614255	-2.891290	-0.973629	-0.071841	6.178555
V17	284807.0	-3.046228	8.614377	-890.103172	-2.746136	-1.032424	-0.305055	9.868076
V18	284807.0	-3.046228	8.589622	-884.897411	-2.793844	-1.015266	-0.239219	4.741740
V19	284807.0	-3.046228	8.663047	-877.427620	-2.802123	-0.978342	-0.191271	8.585230
V20	284807.0	-3.046228	8.347625	-937.427087	-2.738243	-0.920135	-0.337673	17.898794
V21	284807.0	-3.046228	8.532423	-904.549486	-2.710308	-0.872739	-0.351222	29.675627
V22	284807.0	-3.046228	8.657047	-877.217064	-2.727878	-0.995637	-0.308842	15.156065
V23	284807.0	-3.046228	8.672904	-884.510464	-2.662006	-0.847089	-0.268766	21.588013
V24	284807.0	-3.046228	8.598297	-878.344818	-2.754060	-0.971889	-0.179785	6.863644
V25	284807.0	-3.046228	8.621258	-878.374684	-2.700158	-0.906340	-0.284777	6.023979
V26	284807.0	-3.046228	8.595705	-879.513730	-2.713257	-0.900402	-0.238817	6.705869
V27	284807.0	-3.046228	8.578691	-851.317169	-2.699175	-0.843812	-0.226545	9.283093
V28	284807.0	-3.046228	8.584054	-898.359451	-2.670184	-0.819377	-0.240180	20.317105
Amount	284807.0	-3.046228	241.540643	-88.938251	-82.951673	-67.110420	-13.811605	24719.881014

1.4.2 Over-Sampling:In our data set one class of data is the underrepresented minority class in the data **sample**, **over sampling** techniques are used to duplicate these

1.4.2.1 Smote:There are a number of methods available to oversample a dataset used in a typical classification problem . The most common technique is known as SMOTE: Synthetic Minority Over-sampling results for a more balanced amount of positive results in training. **Over sampling** is used when the amount of data collected is

insufficient. Technique.To illustrate how this technique works consider some training data which has s samples, and f features in the feature space of the data. Note that these features, for simplicity, are continuous. As an example, consider a dataset of birds for classification. The feature space for the minority class for which we want to oversample could be beak length, wingspan, and weight (all continuous). To then oversample, take a sample from the dataset, and consider its k nearest neighbors (in feature space). To create a synthetic data point, take the vector between one of those k neighbors, and the current data point. Multiply this vector by a random number x which lies between 0, and 1. Add this to the current data point to create the new, synthetic data point[11][12][13].

[*]Classimblearn.over_sampling.SMOTE Parameters:
1.X :{array-like, sparse matrix}, shape (n_samples, n_features)
Matrix containing the data which has to be sampled**.**
2.y:array-like, shape (n_samples,) Corresponding label for each sample in X

```
>>> from imblearn.over_sampling import SMOTE
Using TensorFlow backend.
>>> X,Y=SMOTE().fit_resample(n, a)
```

Fig 6 : Code used for SMOTE

```
>>> OriginalData['Class'].value_counts()
0    284315
1       492
Name: Class, dtype: int64
```

FIg7 : Class attribute frequencies in original data

```
>>> OverSampledData['Class'].value_counts()
1    284315
0    284315
Name: Class, dtype: int64
```

Fig 8: Class attribute frequencies in Oversampled data

Series.value_counts(normalize=False, sort=True, ascending=False, bins=None, dropna=True): Returns objects containing counts of unique values.The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default[14].

	count	mean	std	min	25%	50%	75%	max
V1	568630.0	-5.193978	9.917008	-918.477906	-6.834686	-1.732740	0.055122	2.694104
V2	568630.0	-1.085131	10.578652	-914.779851	-2.330117	0.129974	2.658728	28.897519
V3	568630.0	-6.355250	10.066197	-931.254956	-7.794169	-3.074376	-0.725776	4.329543
V4	568630.0	-0.665809	9.564739	-867.625183	-2.379230	-0.046990	3.544364	18.684803
V5	568630.0	-4.473588	9.641199	-996.672673	-5.735064	-1.279249	0.108719	34.499014
V6	568630.0	-3.587831	8.483710	-809.627741	-3.667005	-1.300517	-0.475534	13.313904
V7	568630.0	-5.596637	9.090765	-762.339873	-7.456384	-2.265010	-0.544517	11.413864
V8	568630.0	-2.553465	9.873899	-910.276727	-2.904472	-0.493009	0.635023	21.456221
V9	568630.0	-4.183817	8.700796	-886.801791	-4.055580	-1.903222	-0.510095	10.111188
V10	568630.0	-5.715394	9.241667	-894.934854	-7.946436	-2.804391	-0.790634	16.250464
V11	568630.0	-1.001272	9.461833	-876.075470	-2.194878	-0.155398	2.903252	14.695999
V12	568630.0	-5.996868	9.234054	-892.118785	-7.990549	-3.355250	-0.703901	6.843604
V13	568630.0	-2.965904	8.779089	-875.802484	-2.766708	-0.765744	0.457307	5.853985
V14	568630.0	-6.385800	9.289283	-889.725308	-9.183039	-4.151272	-0.845257	9.274047
V15	568630.0	-2.929368	8.695810	-874.051625	-2.617140	-0.633628	0.414885	5.169244
V16	568630.0	-4.938625	9.018490	-865.614255	-6.253738	-2.174244	-0.347071	6.178555
V17	568630.0	-6.197562	9.941811	-890.103172	-9.303223	-2.731520	-0.555181	9.868076
V18	568630.0	-4.005625	8.788175	-884.897411	-4.671648	-1.353497	-0.203087	4.741740
V19	568630.0	-2.578524	8.817662	-877.427620	-2.469444	-0.655090	0.495291	8.585230
V20	568630.0	-2.705164	8.661475	-937.427087	-2.559352	-0.541667	0.465815	17.898794
V21	568630.0	-2.535374	9.074616	-904.549486	-2.524867	-0.508979	0.596238	29.675627
V22	568630.0	-2.902016	8.835592	-877.217064	-2.557971	-0.777927	0.316168	15.156065
V23	568630.0	-2.908858	8.980096	-884.510464	-2.508379	-0.559043	0.099038	21.588013
V24	568630.0	-2.957407	8.732113	-878.344818	-2.636533	-0.527559	0.266452	6.863644
V25	568630.0	-2.862330	8.818822	-878.374684	-2.621688	-0.527927	0.522143	6.023979
V26	568630.0	-2.870111	8.755025	-879.513730	-2.490666	-0.516112	0.324123	6.705869
V27	568630.0	-2.796744	8.683952	-851.317169	-2.533420	-0.455552	0.326479	9.283093
V28	568630.0	-2.854759	8.759925	-898.359451	-2.465178	-0.419513	0.209847	20.317105
Amount	568630.0	13.462587	242.420225	-88.938251	-85.517898	-70.072983	10.671237	24719.881014
Class	568630.0	0.500000	0.500000	0.000000	0.000000	0.500000	1.000000	1.000000

Fig 7: mean,standarddeviation,minimum and maximum metrics of all attributes in data

1.5 Building and Analysis Of Machine Learning Algorithms

1.5.1 Logistic Regression:

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression[15] (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1". In the logistic model, the log-odds (the logarithm of the odds) for the value labeled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). The corresponding probability of the value labeled "1" can vary between 0 (certainly the value "0") and 1 (certainly the value "1"), hence the labeling; the function that converts log-odds to probability is the logistic function, hence the name.[15].

Parameters:

Logistic Regression works with five different solvers. Each solver tries to find the parameter weights that minimize a cost function.Here are the five solvers:

1.5.1.1 newton-cg:A newton method. Newton methods use an exact Hessian matrix. It's slow for large datasets, because it computes the second derivatives.

1.5.1.2 lbfgs:Stands for Limited-memory Broyden–Fletcher–Goldfarb–Shanno. It approximates the second derivative matrix updates with gradient evaluations. It stores only the last few updates, so it saves memory. It isn't super fast with large data sets[16].
1.5.1.3 liblinear:Uses a coordinate descent algorithm. Coordinate descent is based on minimizing a multivariate function by solving univariate optimization problems in a loop. In other words, it moves toward the minimum in one direction at a time.. It performs pretty well with high dimensionality. It does have a number of drawbacks. It can get stuck,is unable to run in parallel, and can only solve multi-class logistic regression with one-vs.-rest[17].
1.5.1.4 sag :Stochastic Average Gradient descent. A variation of gradient descent and incremental aggregated gradient approaches that uses a random sample of previous gradient values. Fast for big datasets[18]**.**
1.5.1.5 saga:Extension of sag that also allows for L1 regularization. Should generally train faster than sag.The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver[19].

1.5.2 Building Classification Model

sklearn.linear_model.LogisticRegression class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted[20].

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
data=pd.read_csv('OverSampledData')
x=data.drop(['Class'],axis=1)
y=data['Class']
classifier1=LogisticRegression(random_state=0,solver='newton-cg').fit(x,y)
classifier2=LogisticRegression(random_state=0,solver='lbfgs').fit(x,y)
classifier3=LogisticRegression(random_state=0,solver='liblinear').fit(x,y)
classifier4=LogisticRegression(random_state=0,solver='sag').fit(x,y)
classifier5=LogisticRegression(random_state=0,solver='saga').fit(x,y)
```

Fig 8: Code used for building 5 classifiers of logistic regression

1.5.3 Results and Analysis

Here we using original data set for testing and checking performance metrics
1.5.3.1 Accuracy classification score:This function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in class.
1.5.3.2 the precision:The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.
1.5.3.3 the recall:The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.
1.5.3.4 confusion matrix:By definition a confusion matrix Cis such that Ci,jis equal to the number of observations known to be in group iand predicted to be in group jThus in binary classification, the count of true negatives is C0,0, false negatives is C1,0, true positives is C1,1and false positives is C

```
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,recall_score
test_data=pd.read_csv('OriginalData')
test_data=OriginalData
X=test_data
Y=test_data['Class']
predicted=classifier.predict(X)
print('accuracy_score is',accuracy_score(Y,predicted))
print('precision is',precision_score(Y,predicted))
print('recall is',recall_score(Y,predicted))
print(confusion_matrix(Y,predicted))
```

Fig 9 :Code used for metrics on test data

	lbfgs	Lib- linear	newton-cg	sag	saga
accuracy	0.9802	0.9816	0.9812	0.9973	0.9963
recall	0.9085	0.9065	0.9166	0.8252	0.8170
precision	0.0740	0.0793	0.0616	0.3755	0.2955

class	0 lb	1
0	278723	5592
1	45	447

class	0ll	1
0	279143	5172
1	46	446

class	0nc	1
0	279035	5280
1	46	446

Table 2,3,4: confusion matrics over lbfgs,lib-linear, newton-cg classifiers

class	0	1
0	283640	675
1	86	406

class	0	1
0	283366	949
1	90	402

Table 5,6 : confusion matrix of sag and saga classifiers

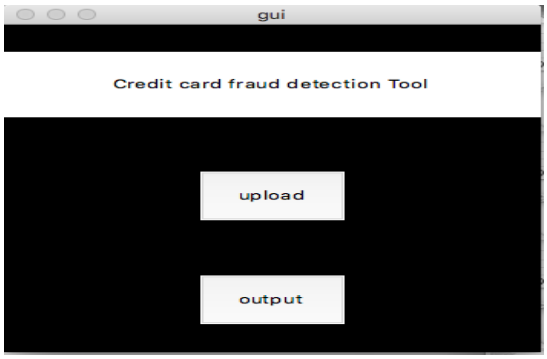

Fig 10 : graphical user interface for fraud classifier

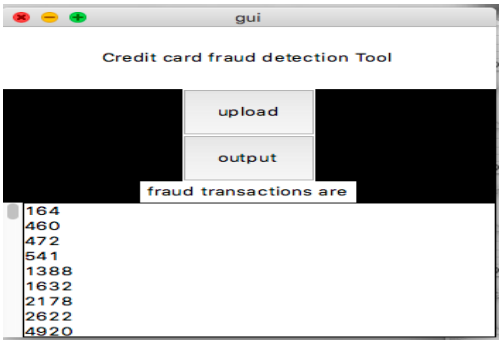

Fig 11:fraud transactions list

II. Conclusion

Before implementing this project there were many flaws in classification models.Here we covered all those and made them out of it .After that we achieved a highly resistant classification model over this highly imbalanced data.

References

[1].dataset has been collected and analysed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection.
[2].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.
[3].DalPozzolo, Andrea; Caelen, Olivier; Le Borgne, Yann-Ael; Waterschoot, Serge; Bontempi, Gianluca. Learned lessons in credit card fraud detection from a practitioner perspective, Expert systems with applications,41,10,4915-4928,2014, Pergamon.
[4].DalPozzolo, Andrea; Boracchi, Giacomo; Caelen, Olivier; Alippi, Cesare; Bontempi, Gianluca. Credit card fraud detection: a realistic modeling and a novel learning strategy, IEEE transactions on neural networks and learning systems,29,8,3784-3797,2018,IEEE.
[5].DalPozzolo, Andrea Adaptive Machine learning for credit card fraud detection ULB MLG PhD thesis (supervised by G. Bontempi).
[6].Carcillo, Fabrizio; Dal Pozzolo, Andrea; Le Borgne, Yann-Aël; Caelen, Olivier; Mazzer, Yannis; Bontempi, Gianluca. Scarff: a scalable framework for streaming credit card fraud detection with Spark, Information fusion,41, 182-194,2018,Elsevier.
[7].Carcillo, Fabrizio; Le Borgne, Yann-Aël; Caelen, Olivier; Bontempi, Gianluca. Streaming active learning strategies for real-life credit card fraud detection: assessment and visualization, International Journal of Data Science and Analytics, 5,4,285-300,2018,Springer International Publishing.
[8].BertrandLebichot, Yann-Aël Le Borgne, Liyun He, Frederic Oblé, Gianluca Bontempi Deep-Learning Domain Adaptation Techniques for Credit Cards Fraud Detection, INNS DDL 2019: Recent Advances in Big Data and Deep Learning, pp 78-88, 2019.
[9].FabrizioCarcillo, Yann-Aël Le Borgne, Olivier Caelen, Frederic Oblé, Gianluca Bontempi Combining Unsupervised and Supervised Learning in Credit Card Fraud Detection Information Sciences, 2019.
[10].https://scikitlearn.org/stable/modules/generated/sklearn.preprocessing.KernelCenterer.html#sklearn.preprocessing.KernelCenterer.
[11].https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume16/chawla02ahtml/chawla2002..html
[12].Chawla, Nitesh V.; Herrera, Francisco; Garcia, Salvador; Fernandez, Alberto (2018-04-20). "SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary". Journal of Artificial Intelligence Research. 61: 863–905. doi:10.1613/jair.1.11192. ISSN 1076-9757.
[13].(1, 2) N. V. Chawla, K. W. Bowyer, L. O.Hall, W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," Journal of artificial intelligence research, 16, 321-357, 2002.
[14].https://pandas.pydata.org/pandas-docs/stable/reference/index.html.
[15].https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression.
[16].L-BFGS-B – Software for Large-scale Bound-constrained OptimizationCiyou Zhu, Richard Byrd, Jorge Nocedal and Jose Luis Morales. http://users.iems.northwestern.edu/~nocedal/lbfgsb.html
[17].LIBLINEAR – A Library for Large Linear Classification https://www.csie.ntu.edu.tw/~cjlin/liblinear/
[18].SAG – Mark Schmidt, Nicolas Le Roux, and Francis Bach Minimizing Finite Sums with the Stochastic Average Gradient https://hal.inria.fr/hal-00860051/document
[19].SAGA – Defazio, A., Bach F. & Lacoste-Julien S. (2014).SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives https://arxiv.org/abs/1407.0202
[20].Hsiang-Fu Yu, Fang-Lan Huang, Chih-Jen Lin (2011). Dual coordinate descent methods for logistic regression and maximum entropy models. Machine Learning 85(1-2):41-75.https://www.csie.ntu.edu.tw/~cjlin/papers/maxent_du.