Report on Analysis of Credit Card Fraud Dataset

Ivo Mihov ID 9918179

MPhys student at

The University of Manchester

Description

The data chosen for this coursework was gathered from European credit card transactions gathered for 2 days [1]. The dataset was downloaded from the Kaggle website [1]. It was collected to improve existing fraud prevention mechanisms. The dataset consists of 30 attributes and 284 807 instances. Although this is a large amount of data, it is very imbalanced, since the frauds represent only 0.17% of all transactions. In the last column in the dataset, named ‘Class’, fraudulent transactions are labelled with ones and normal transactions – with zeros. Also, due to the sensitivity of the information (credit card details etc.), it is presented using the following attributes: the time from the first transaction in the dataset to the transaction in seconds, the amount transferred in an unknown currency (Euro is assumed), and 28 attributes named V1 to V28, possibly recovered through Principal Component Analysis (PCA) [1].

Methodology and Theory

First, it had to be confirmed that the attributes were indeed principal components and that they contribute to the variance of the data. It was found that one of the new principal components (found after doing PCA on the whole data, incl. attributes ‘Time’ and ‘Amount’) only accounted for 0.14% of the variance and all the other – for more than 1%. At that point the data was transformed to be in terms of the new principal components. Then a synthetic oversampling technique was used to increase the number of data points that classify as frauds. This way the ratio of fraudulent to legitimate transactions was enhanced and it was possible to analyse the data further.

PCA is a technique that exploits the eigen-decomposition of the correlation matrix [2]. Each eigenvector is a principal component (PC) of the distribution and each PC has a loading that is related to the variance for which it accounts (weight) by the formula:

where the summation is over all loadings [2]. In the case of PCA the loadings are the eigenvalues of the correlation matrix [2]. To get the matrix, the data must be standardised [2]. Standardisation is achieved by first subtracting the mean of each attribute from every observation in the data, yielding the centred data matrix **Y**, and then transforming the centred data using the D-matrix:

where **S** is the unbiased covariance matrix of the centred data (, where *n* is the number of data points in the sample), and **I** is the identity matrix, where is the number of attributes (Note the element-wise inverse square root!) [2]. Then the standardised data can be found by

where **Z** is the standardised data [2]. One can simply get the correlation matrix R by using

and solve its eigensystem to get the principal components [2]. Then the data can transformed to be in terms of the principal components, so it becomes orthogonalized [2].

After the data was orthogonalised, the minority class (the frauds) needed to be oversampled, since it represented only 0.17% of the data. The Synthetic Minority Oversampling TEchnique (SMOTE) was used, because instead of simply copying the existing data points, it created artificial new data points between instances of the minority class [3]. It did that in the following steps:

1. Check if the desired ratio after the oversampling is bigger than the existing one;
2. Choose N minority instances randomly (where N is the desired ratio multiplied by the number of instances in minority class, rounded to the nearest integer) and save them in array (N is always bigger than the number of minority data points);
3. Find the k (5 by default) nearest neighbours of each point in the array and save them in 2D array (shape (N, k));
4. Find difference between one of the points in and one of its nearest neighbours, chosen randomly. Append a new data point with attributes to the data matrix, where is a pseudo-random coefficient between 0 and 1, is the chosen existing point in array and is one of the *k* nearest neighbours of ;
5. Loop over all points in and repeat 4) for each of them;
6. Return the data matrix with the corresponding class labels [3].

Finally, logistic regression was used to classify the data. It is a simple linear model which fits an equation of the type

(1)

where [4]. Note that both and are vectors [4]. With frauds, we only have 2 classes, so

Therefore, the likelihood function is

where one usually takes the logarithm to make the optimisation simpler [4]. Additionally, regularisation can be added to the log-likelihood to prevent overfitting by keeping fitting parameters finite:

for the L2 regularisation case, where is the log-likelihood function and is a regularisation hyperparameter [5].

The maximisation is accomplished by finding the steepest descent using the gradient of , as well as the second derivative to find how the descent changes and decide on the size of the step in that direction [4]. The process is iterated until the algorithm has converged [4].

Due to the irregularity of the data, all three steps of the analysis had to be employed to analyse the dataset.

Results and Discussion

The raw data was examined by plotting the only two unhidden attributes – the amount of the transaction against the time since the first transaction in the dataset (Figure 1). The legitimate and fraudulent transactions were plotted separately to look for a correlation between the time of the transaction and its nature. By judging from the number and amounts of the transactions, and the length of 1 day in seconds (86,400 s), one may see that there is a bigger number of transactions during daytime. Also, a similar pattern can be seen with fraudulent transaction, but they might occur more often at night and less often in the morning. However, no conclusive evidence was found.

Картина, която съдържа текст, карта

Описанието е генерирано автоматично

Figure : Amount of transactions in Euros plotted versus time since first transaction in dataset in seconds. On the left plot only legitimate transactions are shown, and on the right - only fraudulent ones. (L stands for Legitimate, and F - for Fraudulent)

The results of the logistic regression classification were evaluated using several different types of scores (Table 1). The main one was a confusion matrix, which clearly shows how many instances of each class were classified right and wrong. To help with evaluation, accuracy, precision, F1 scores and Matthew’s Correlation Coefficients were also assigned to each test [6]. The accuracy is not a good measure of the algorithm here, because even if all transactions are classified legitimate, the accuracy would still be about 99.83%. Matthew’s correlation coefficient (MCC) is between -1 and 1 and is given by the expression

where TP stands for True Positives, TN – for True Negatives, FP – for False Positives and FN – for False Negatives [6].

*Table 1: Results from training with different parameters and amount of data. Note that is the sum of all instances in training data.*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class Weights (Legit:Fraud) and Other Parameters | Accuracy [6] | Precision [6] | MCC  (Matthew’s Correlation Coefficient) | F1 score  [6] | Confusion Matrix [6]   |  |  | | --- | --- | | True Negative | False Positive | | False Negative | True Positive | |
| (1:1) | 0.9993 | 0.8421 | 0.8055 | 0.8050 | |  |  | | --- | --- | | 47,373 | 12 | | 19 | 64 | |
| (1:2) | 0.9993 | 0.7753 | 0.8025 | 0.8023 | |  |  | | --- | --- | | 47,365 | 20 | | 14 | 69 | |
| (1:2.5) | 0.9993 | 0.7778 | 0.8096 | 0.8092 | |  |  | | --- | --- | | 47,365 | 20 | | 13 | 70 | |
| (1:3) | 0.9993 | 0.7553 | 0.8035 | 0.8023 | |  |  | | --- | --- | | 47,362 | 23 | | 12 | 71 | |
| (1:5) | 0.9991 | 0.71 | 0.7789 | 0.7760 | |  |  | | --- | --- | | 47,356 | 29 | | 12 | 71 | |
| (1:1), | 0.9994 | 0.8442 | 0.8128 | 0.8125 | |  |  | | --- | --- | | 47,373 | 12 | | 18 | 65 | |
| (1:1), | 0.9993 | 0.8333 | 0.8075 | 0.8075 | |  |  | | --- | --- | | 47,372 | 13 | | 18 | 65 | |
| (1:1),  Training data dataset | 0.9993 | 0.8054 | 0.7793 | 0.7792 | |  |  | | --- | --- | | 94,748 | 29 | | 39 | 120 | |
| (1:1),  Training data dataset | 0.9993 | 0.8110 | 0.8107 | 0.8110 | |  |  | | --- | --- | | 71,051 | 24 | | 24 | 103 | |
| (1:1),  Oversampling to 0.6% | 0.9993 | 0.8 | 0.8092 | 0.8095 | |  |  | | --- | --- | | 47,368 | 17 | | 15 | 68 | |
| (1:1),  Oversampling to 1.0% | 0.9993 | 0.7667 | 0.7980 | 0.7977 | |  |  | | --- | --- | | 47,364 | 21 | | 14 | 69 | |

\*Wherever it is not clarified, testing is done with a maximum limit of 100,000 iterations, minority class oversampled to 0.4%, regularisation parameter set to 0.05 and with training data being of whole dataset (every 6th transaction for testing).

As expected, a tendency towards a positive correlation between size of training data and scoring can be seen. Also, placing bigger weights on the minority class results in more TP’s, and more FP’s, which was expected. On the other hand, varying the regularisation parameter did not make any noticeable difference. Similarly to giving the fraud class a bigger weight, oversampling it to a bigger percentage results in more TP’s, but also more FP’s. Since it is necessary to catch a big percentage of the fraudsters, it is more sensible to weigh the minority class heavier. Thus, a class weight of (Majority weight : Minority weight) = (1:3) would be a good choice for the training algorithm.

Any attempts on visualising the data proved futile. The reasons are the large number of attributes and the hidden nature of 28 of them. One possibility for improvement to this analysis would be to use Bayesian Optimisation to vary the parameters of the training. Another option would be to get rid of the majority instance from pairs that are close to the decision boundary and belong to different classes (Tomek links) [7]. Also, one could try fitting non-linear models (e.g. neural network) to the data.

References

[1] "Credit Card Fraud Detection", *Kaggle.com*, 2019. [Online]. Available: https://www.kaggle.com/mlg-ulb/creditcardfraud. [Accessed: 24- Nov- 2019].

[2] Thomas House, “Lecture notes in Multivariate Statistics MATH48061”, The University of Manchester, 2019.

[3] N. Chawla, K. Bowyer, L. Hall and W. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique", *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002. Available: 10.1613/jair.953.

[4] C. Shalizi, *Advanced data analysis from an elementary point of view*. Cambridge: Cambridge University Press, 2013, pp. 232-238.

[5] S. Raschka, "Regularization in Logistic Regression: Better Fit and Better Generalization? - KDnuggets", *KDnuggets*, 2019. [Online]. Available: https://www.kdnuggets.com/2016/06/regularization-logistic-regression.html. [Accessed: 24- Nov- 2019].

[6] "API Reference — scikit-learn 0.21.3 documentation", *Scikit-learn.org*, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics. [Accessed: 24- Nov- 2019].

[7]"Handling imbalanced dataset in supervised learning using family of SMOTE algorithm.", *Datasciencecentral.com*, 2019. [Online]. Available: https://www.datasciencecentral.com/profiles/blogs/handling-imbalanced-data-sets-in-supervised-learning-using-family. [Accessed: 24- Nov- 2019].

Code

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import csv

import seaborn as sb

import pandas as pd

from imblearn.over\_sampling import SMOTE

from sklearn.linear\_model import LogisticRegression as LR

from sklearn.metrics import \*

#

# analysing 284 807 instances of credit card transactions in Europe in 2013

# dataset downloaded from https://www.kaggle.com/mlg-ulb/creditcardfraud

# by Ivo Mihov ( ID 9918179 )

# MPhys student at

# The University of Manchester

# ivo.mihov@protonmail.com

#

# step 1: extracting data

print('...........................\nstep 1: extracting data')

# first create a dictionary attr for our variables

attr = {}

file = open('creditcard.csv', 'r')

# counter i for the data extraction loop

i = 0

for line in file:

split = line.split(",")

# we get rid of symbols that make our labels look ugly and do not allow to cast data into float

split = map(lambda each:each.strip("\n"), split)

split = list(map(lambda each:each.strip('"'), split))

# in the first iteration we get the names of the attributes to fill the names in the dictionary

if i == 0:

i += 1

# initialise array of strings to save the attribute names

names = np.empty(len(split), dtype='U25')

for j in range(len(split)):

# save the labels in names array

names[j] = split[j]

# label the attribute in dictionary and assign it an empty array

attr[names[j]] = np.empty(284807, dtype='f8')

continue

# for all but first row the below code is executed

for k in range(len(names)):

# save data in attr dictionary

attr[names[k]][i-1] = split[k]

i += 1

file.close()

# fill a matrix with the data, so it's easier to work with it

data\_matr = np.empty((len(attr.keys()) - 1, 284807))

i = 0

for key in attr.keys():

if i == len(names) - 1:

class\_matr = attr[names[len(names) - 1]][:]

else:

data\_matr[i][:] = attr[key][:]

i += 1

# fill an array with class names, instead of numbers

class\_labels = np.empty(len(class\_matr), dtype=str)

class\_labels[class\_matr == 0] = 'LEGIT'

class\_labels[class\_matr == 1] = 'FRAUD'

'''

# make plots to visualise the data

matrix = np.empty((len(data\_matr)+1, len(data\_matr[0])), dtype='O')

matrix[:30] = data\_matr[:]

matrix[30] = class\_labels[:]

names[0] = 'Time (s)'

names[29] = 'Amount (Euros)'

df = pd.DataFrame(matrix.T, columns = names)

sb.relplot(x="Time (s)",

y="Amount (Euros)",

hue="Class",

style="Class",

col="Class",

data=df,

markers=['o', 'X'],

legend='full',

edgecolor=None,

s=3)

plt.savefig('time\_vs\_amount.png')

plt.show()

'''

# step 2: doing PCA to find variance and eventually exclude some attributes

print('...........................\nstep 2: doing PCA to find variance and eventually exclude some attributes')

# finding the covariance/correlation matrix

# will have to manually subtract the mean from each column so as not to construct a 280000 by 280000 matrix

# find average of each attribute and subtract from each row

avg = np.average(data\_matr, axis=1)

data\_matr -= avg[:, None]

S\_unb = np.true\_divide(1., (284807. - 1.), dtype='f8') \* np.dot(data\_matr, data\_matr.T)

D = np.matrix(np.diag(np.power(np.diag(S\_unb), -0.5)))

Z = np.matrix(data\_matr.T \* D)

R = np.matrix(1/float(284807 - 1) \* Z.T \* Z)

'''

# make a heatmap of the correlations

lbls = []

for i in range(np.shape(R)[0]):

lbls.append('PC' + str(i + 1))

# make a dataframe, so we have better labels

R\_df = pd.DataFrame(R, columns=lbls, index=lbls)

sb.set\_style(style='white')

fig, ax = plt.subplots(figsize=(11, 9))

cmap = sb.blend\_palette(['#86002a', '#d10b10', '#fffbe6', '#1f7ecc', '#002a86'], n\_colors=5, as\_cmap=True)

sb.heatmap(R\_df, mask=None, cmap=cmap, square=True, linewidths=0.5, cbar\_kws={"shrink": 0.5}, ax=ax)

plt.savefig('correlation\_heatmap.png')

plt.show()

'''

# finding principal components and weights

values, vectors = np.linalg.eig(R)

sum\_weights = np.sum(values)

weights = values / sum\_weights

indices\_pca = np.argsort(weights)[::-1]

values = values[indices\_pca]

vectors = vectors[indices\_pca]

weights = weights[indices\_pca]

print(weights)

# save the correlation matrix into a csv file

with open('Correlation.csv', 'w') as csv\_file:

writer = csv.writer(csv\_file)

writer.writerow(attr.keys())

writer.writerows(np.array(R))

print('...........................\nCorrelation matrix saved in current directory.\n')

# create a matrix from eigenvectors to orthogonalise data

unitary = vectors[0]

for i in range(len(vectors)):

if i != 0:

unitary = np.hstack((unitary, vectors[i]))

unitary = np.array(unitary).reshape((len(vectors), len(vectors)))

# orthogonalise all data (present in terms of principal components)

# this is vital for the Logistic Regression as it is a linear model, so orthogonality helps

pca\_data = np.dot(data\_matr.T, unitary)

# divide data into train and test set to validate training

# every 6th instance will be used for testing

indices\_test = np.arange(np.shape(data\_matr)[1])

indices\_test = indices\_test[indices\_test % 6 == 0]

indices\_train = np.arange(np.shape(data\_matr)[1])

indices\_train = indices\_train[indices\_train % 6 != 0]

pca\_train\_data = np.empty((len(indices\_train), 30), dtype='f8')

pca\_test\_data = np.empty((len(indices\_test), 30), dtype='f8')

pca\_train\_data[:] = pca\_data[indices\_train, :]

pca\_test\_data[:] = pca\_data[indices\_test, :]

pca\_train\_class = np.empty(len(indices\_train))

pca\_test\_class = np.empty(len(indices\_test))

pca\_train\_class[:] = class\_matr[indices\_train]

pca\_test\_class[:] = class\_matr[indices\_test]

'''

Using the SMOTE class to do synthetic oversampling of the minority class

works by focusing on the minority class only, and then creating new minority data points

by combining the features of the ones present, i.e. new ones lie on the line between two old instances.

This way all the information is kept and overfitting the minority class data is less of an issue.

'''

# step 3: oversampling fraud data points to balance dataset

print('...........................\nstep 3: oversampling fraud data points to balance dataset')

sm = SMOTE(sampling\_strategy=0.004) # sampling\_strategy is the ratio of minority class data points of all after oversampling

pca\_train\_data, pca\_train\_class = sm.fit\_resample(pca\_train\_data, pca\_train\_class)

# step 4: classification

print('...........................\nstep 4: classification')

print('Applying Logistic Regression Model...')

num\_frauds\_test = np.sum(pca\_test\_class == 1)

print('number of frauds in test dataset is', num\_frauds\_test)

lr = LR(class\_weight={0:1, 1:3},

solver='lbfgs',

multi\_class='ovr',

penalty='l2',

max\_iter=100000,

C=0.05).fit(pca\_train\_data, pca\_train\_class)

predictionsLogR = lr.predict(pca\_test\_data.reshape(-1, 30))

print('Accuracy:', accuracy\_score(pca\_test\_class, predictionsLogR))

print('Precision:', precision\_score(pca\_test\_class, predictionsLogR))

print('MCC:', matthews\_corrcoef(pca\_test\_class, predictionsLogR))

print('F1:', f1\_score(pca\_test\_class, predictionsLogR))

print('Confusion matrix on PCA data (LogR):\n', confusion\_matrix(pca\_test\_class, predictionsLogR))