COEN 129

Final Project Report

Tracy Sun & Nicholas Fong

**Detecting Credit Card Fraud**

**Introduction**

The dataset contains two days of credit card transactions from European credit card holders back in September of 2013. The dataset contains 492 fraud out of the total of 284,807 transactions, and there are 28 confidential features pre-selected through PCA transformations with time and amount.

In this project, we will use different machine learning models: LDA, QDA, logistic regression, decision tree, and k-nearest neighbors algorithm to fit the dataset, so that we can understand purchasing habits to identify fraud and hope to conclude various other sociological patterns. Before we fit the dataset into different machine learning models, we normalized all the data points to get a more evenly distributed dataset. Though we do not need to perform feature selection for this project because the data already underwent a PCA transformation, we dropped the “time ” variable because it is just noise that doesn’t help us identify fraud and because we cannot draw any concrete conclusions due to the lack of information to all the feature variables or the information about the transactions.

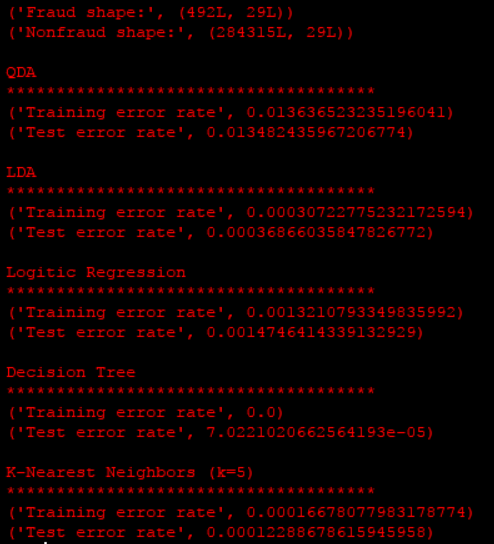
Linear and quadratic discriminant analysis are supervised learning models. The classifiers are easy to compute because they have close-formed solution. Logistic regression is a regression model that we used to predict categorical variables: fraud or nonfraud. Logistic regression measures the relationship between the categorical variables by estimating probabilities using a logistic function. The two new models that we tried are decision tree and k-nearest algorithm. Decision tree, also known as classification and regression tree, has the shape of a binary tree and uses a greedy algorithm on the training data to pick splits in the tree. K-nearest neighbors algorithm has no model other than storing the entire dataset, so there is no learning required. When KNN is used in classification, the output can be calculated as the class with the highest frequency from the K-most similar instances.

Due to the imbalance between low number of frauds relative to the number of total transactions, we predict complication of overfitting problem. Fitting the dataset into different models accurately without overfitting will be more difficult.

**Source Code**

|  |
| --- |
| import math  import numpy as np  import csv  from sklearn.linear\_model import LogisticRegression  import sklearn.discriminant\_analysis  import sklearn.preprocessing  from sklearn import tree  from sklearn.neighbors import KNeighborsClassifier  file\_name = "creditcard.csv"  fraud = [] #class 1  nonfraud = [] #class 0  #read data file  with open(file\_name,'r') as f:  next(f)  for line in csv.reader(f):  if(int(line[-1]) == 0):  nonfraud.append(line[1:-1])  else:  fraud.append(line[1:-1])  f.close()  nonfraud = [[float(num) for num in row] for row in nonfraud]  fraud = [[float(num) for num in row]for row in fraud]  nonfraud = np.array(nonfraud)  fraud = np.array(fraud)  nonfraud = sklearn.preprocessing.normalize(nonfraud, axis=0)  fraud = sklearn.preprocessing.normalize(fraud, axis=0)  print ("Fraud shape:", np.shape(fraud)) # 492  print ("Nonfraud shape:", np.shape(nonfraud)) # 284315  #extract 80% training and 20% test data  num\_fraudtrain = 392  num\_fraudtest = 100  num\_nonfraudtrain = 227452  num\_nonfraudtest = 56863  np.random.shuffle(fraud) #randomize the data  np.random.shuffle(nonfraud)  fraud\_train = fraud[:num\_fraudtrain]  fraud\_test = fraud[num\_fraudtrain:]  nonfraud\_train = nonfraud[:num\_nonfraudtrain]  nonfraud\_test = nonfraud[num\_nonfraudtrain:]  #Process data for library analysis  training\_data = np.concatenate((fraud\_train, nonfraud\_train), axis=0)  training\_y = np.concatenate((np.ones(num\_fraudtrain), np.zeros(num\_nonfraudtrain)), axis=0)  test\_data = np.concatenate((fraud\_test, nonfraud\_test), axis=0)  test\_y = np.concatenate((np.ones(num\_fraudtest), np.zeros(num\_nonfraudtest)), axis=0)  #Test LDA and QDA again  qda = sklearn.discriminant\_analysis.QuadraticDiscriminantAnalysis()  qda.fit(training\_data, training\_y)  print("\nQDA\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Training error rate", 1-qda.score(training\_data,training\_y))  print("Test error rate", 1-qda.score(test\_data,test\_y))  lda = sklearn.discriminant\_analysis.LinearDiscriminantAnalysis()  lda.fit(training\_data, training\_y)  print("\nLDA\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Training error rate", 1-lda.score(training\_data,training\_y))  print("Test error rate", 1-lda.score(test\_data,test\_y))  #testing logistic regression  logistic = LogisticRegression()  logistic = logistic.fit(training\_data, training\_y)  print("\nLogitic Regression\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Training error rate", 1-logistic.score(training\_data,training\_y))  print("Test error rate", 1-logistic.score(test\_data,test\_y))  #testing decision trees  clf = tree.DecisionTreeClassifier()  clf.fit(training\_data, training\_y)  print("\nDecision Tree\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Training error rate", 1-clf.score(training\_data,training\_y))  print("Test error rate", 1-clf.score(test\_data,test\_y))  #testing k-nearest neighbors  knn = KNeighborsClassifier()  knn.fit(training\_data, training\_y)  print("\nK-Nearest Neighbors (k=5)\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Training error rate", 1-knn.score(training\_data,training\_y))  print("Test error rate", 1-knn.score(test\_data,test\_y)) |

**Output**



|  |  |
| --- | --- |
| **Classification Method** | **Test Error Rate** |
| Guess all as non-fraud | 0.172% |
| QDA | 1.348% |
| LDA | 0.037% |
| Logistic | 0.147% |
| Decision Tree | 0.007% |
| K-Nearest Neighbors | 0.012% |

**Conclusions**

After examining several different machine learning classification techniques, we have found that Decision Trees has the lowest test error rate. This means that Decision Trees has the best generalized model for predicting classifications of credit card transactions. If we use Decision Trees to analyze future credit card transactions, we can expect an error of about 0.0007%. This is 1 error for every 14241 transactions. For reference, if we blindly assumed that all credit card transactions are not fraudulent, we would have an error rate of 0.172% due to the high imbalance of the data.

One interesting result is that QDA had a much higher error rate than LDA. In fact, QDA is worse than blindly guessing that all transactions are not fraudulent. This is interesting because QDA is just a more generalized version of LDA with a quadratic decision boundary. The only thing different between LDA and QDA is that LDA assumes that all of the covariance matrices are the same. This seems to imply that both fraudulent and non-fraudulent credit card transactions have a very similar, if not the same covariance matrix. The higher error rate in QDA can be attributed to overfitting noise in the data since a linear decision boundary is better than a quadratic one.

With the highly accurate model that we have from Decision Trees, we hope to pass the model off to the kaggle competition so that they can not only our model to more accurately identify credit card fraud but also examine the model to understand the underlying nature and patterns of fraudulent credit card transactions. We can’t do this analysis ourselves because the original data is obscured for confidentiality reasons, but we hope that our project will advance our understanding of the world around us.

**References**

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

Data Source: <https://www.kaggle.com/dalpozz/creditcardfraud>

Data provided by the research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles)

Code developed in Python 2.7.11 using sklearn for the different classification models