1 INTRODUCTION

1.1 Overview

Automation and artificial intelligence (AI) are transforming businesses and will contribute to economic growth via contributions to productivity [1-10]. They will also help address challenges in areas of healthcare, technology & other areas. At the same time, these technologies will transform the nature of work and the workplace itself. In this code pattern, we will focus on building state of the art systems for churning out predictions which can be used in different scenarios. We will try to predict fraudulent transactions which we know can reduce monetary loss and risk mitigation. The same approach can be used for predicting customer churn, demand and supply forecast and others. Building predictive models require time, effort and good knowledge of algorithms to create effective systems which can predict the outcome accurately. With that being said, IBM has introduced Auto AI which will automate all the tasks involved in building predictive models for different requirements. We will get to see how Auto AI can churn out great models quickly which will save time and effort and aid in faster decision making process [11-20].

When the reader has completed this code pattern, they will understand how to:

Quickly set up the services on cloud for model building. Ingest the data and initiate the Auto AI process. Build different models using Auto AI and evaluate the performance.

Choose the best model and complete the deployment. Generate predictions using the deployed model by making ReST calls.

Compare the process of using Auto AI and building the model manually.

1.2 Purpose

The Credit Card Fraud Detection Problem includes modeling past credit card transactions with the knowledge of the ones that turned out to be fraud. This model is then used to identify whether a new transaction is fraudulent or not.

2 LITERATURE SURVEY

2.1 Existing problem

What is a fraud? There are many formal definitions but essentially a fraud is a crime of deceiving and scamming people in their financial transactions. Frauds have always existed throughout human history but in this age of digital technology, the strategy, extent and magnitude of financial frauds is becoming wide-ranging — from credit cards transactions to health benefits to insurance claims. Fraudsters are also getting super creative. Who has never received an email from a Nigerian royal widow that she is looking for trusted someone to hand over large sums of her inheritance?

No wonder why is fraud a big deal? Estimated loses in business organizations soar up to 4–5% of their revenues due to fraudulent transactions. A 5% fraud may not sound a lot, but in monetary terms, it is non-trivial and outweighs the costs of not doing anything by a large margin. A PwC survey found that 50% of 7,200 companies they surveyed had been victims of fraud of some kind. A very recent study by FICO found that 4 out of 5 banks in their survey have experienced an increase in fraud activities and this is expected to rise in the future.

Although many organizations took measures to counter frauds, it could never be eradicated. The goal is really to minimize its impacts. The benefits of this screening must be weighed against the costs, such as, investment in fraud detection technology and potentially losing customers due to “false positive” alarms.

So the purpose of this article is to highlight some tools, techniques and best practices in the field of fraud detection. Towards the end, I will provide a python implementation using publicly available dataset as well as IBM Auto AI Machine Learning services to predict the risk of fraud.

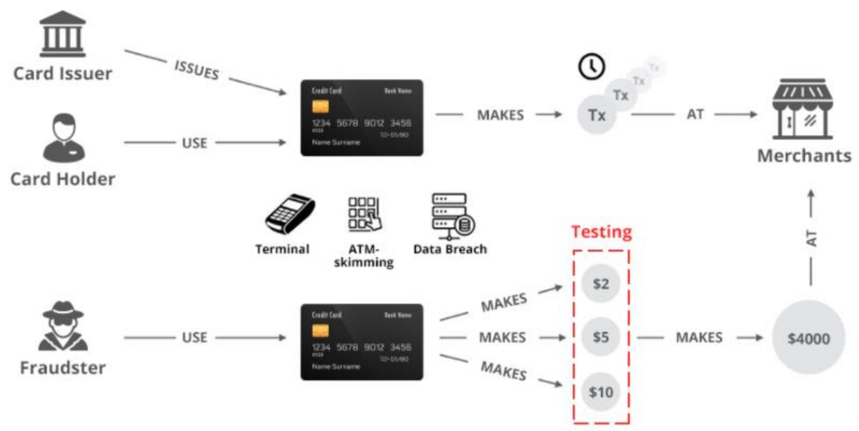
2.2 Proposed solution

In the past (i.e. before machine learning became the trend) the standard practice was to use the so-called “Rule-based approach”. Since every rule has an exception, this technique was able to only partially mitigate the problem. With ever-increasing online transactions and production of a large volume of customer data, machine learning has been increasingly seen as an effective tool to detect and counter frauds. However, there is no specific tool, the silver bullet that works for all kinds of fraud detection problems in every single industry. The nature of the problem is different in every case and every industry. Therefore every solution is carefully tailored within the domain of each industry.

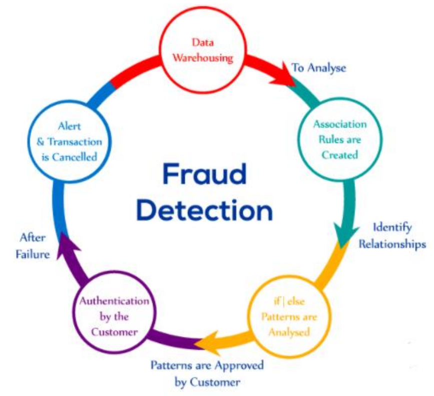
In machine learning, parlance fraud detection is generally treated as a supervised classification problem, where observations are classified as “fraud” or “non-fraud” based on the features in those observations. It is also an interesting problem in ML research due to imbalanced data — i.e. there are a very few cases of frauds in an extremely large amount of transactions. How to deal with imbalance classes is itself a subject of another discussion. Frauds are also isolated using several outlier detection techniques. Outlier detection tools have their own way of tackling the problem, such as time series analysis, cluster analysis, real-time monitoring of transactions etc.

3 THEORITICAL ANALYSIS

3.1 Block diagram



3.2 Hardware / Software designing


Fraud Detection
Data Warehousing
To Analyse
Association Rules are Created
Identify Relationships
if | else Patterns are Analysed
Patterns are Approved by Customer
Authentication by the Customer
After Failure
Alert & Transaction is Cancelled

Statistical techniques: average, quantiles, probability distribution, association rules

Supervised ML algorithms: logistic regression, neural net, time-series analysis

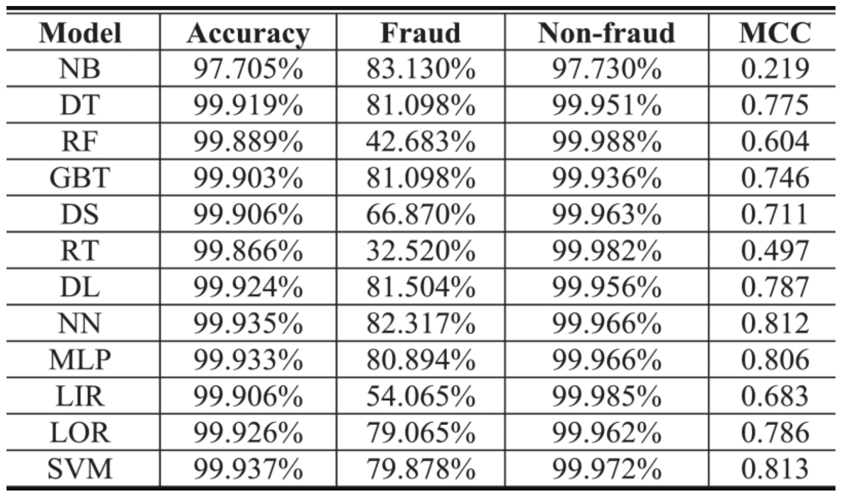
Unsupervised ML algorithms: Cluster analysis, Bayesian network, Peer group analysis, break point analysis, Benford’s law (law of anomalous numbers).

4 EXPERIMENTAL INVESTIGATIONS

A publicly available data set is downloaded from the link mentioned in the referencces. It contains a total of 284,807 transactions made in September 2013 by European cardholders. The data set contains 492 fraud transactions, which is highly imbalanced. Due to the confidentiality issue, a total of 28 principal components based on transformation are provided. Only the time and the amount data are not transformed, and are provided as such.

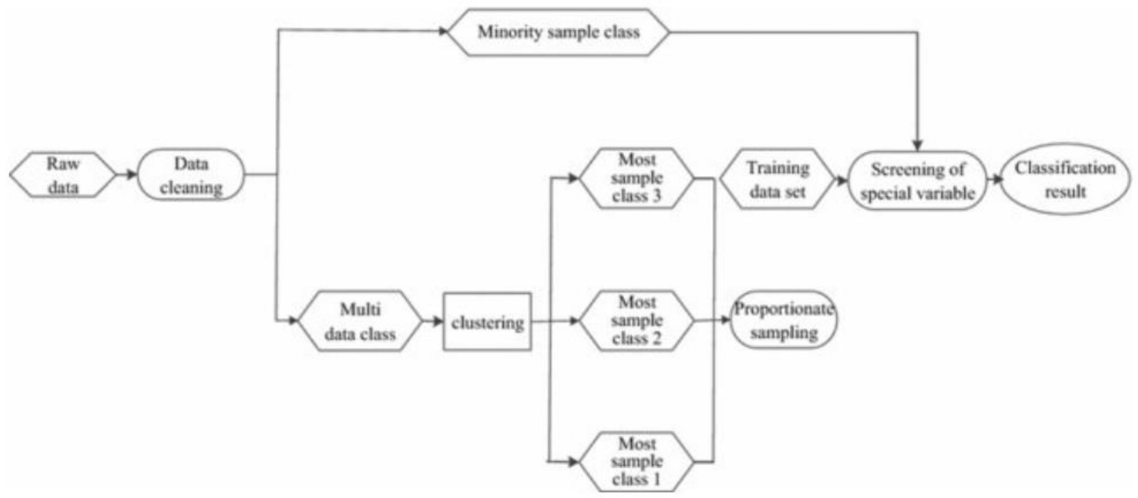
The results from various models are shown in Table below. It can be seen that the accuracy rates are high, generally around 99%. This however is not the real outcome, as the rate of fraud detection varies from 32.5% for RT up to 83% for NB. The rate of non-fraud detection is similar to the accuracy rates, i.e., the non-fraud results dominate the accuracy rates. SVM produces the highest MCC score of 0.813, while the lowest is from NB with an MCC score of 0.219.

Table-1.



Model	Accuracy	Fraud	Non-fraud	MCC
NB	97.705%	83.130%	97.730%	0.219
DT	99.919%	81.098%	99.951%	0.775
RF	99.889%	42.683%	99.988%	0.604
GBT	99.903%	81.098%	99.936%	0.746
DS	99.906%	66.870%	99.963%	0.711
RT	99.866%	32.520%	99.982%	0.497
DL	99.924%	81.504%	99.956%	0.787
NN	99.935%	82.317%	99.966%	0.812
MLP	99.933%	80.894%	99.966%	0.806
LIR	99.906%	54.065%	99.985%	0.683
LOR	99.926%	79.065%	99.962%	0.786
SVM	99.937%	79.878%	99.972%	0.813

5 FLOWCHART



Flowchart-1. Fraud prediction model

6 RESULT

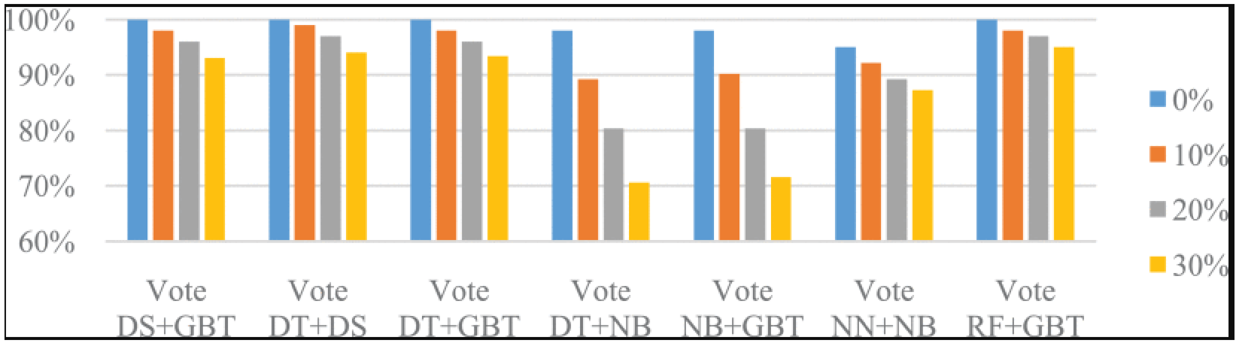
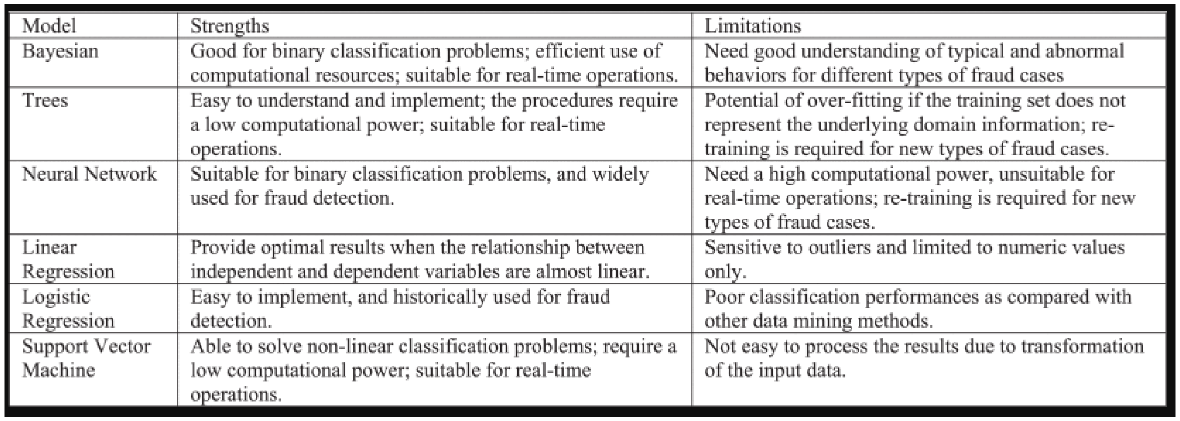


Table-2. Results

Fraud detection data set provided by IBM Watson is used in the experiment. A total of 287,224 transactions are recorded, with 102 of them classified as fraud cases. The data consist of a time series of transactions. To comply with customer confidentiality requirements, no personal identifying information is used. The features used in the experiment are given in the table above. A total of 11 features are used. The codes used are based on the standard ISO 8583, while the last two codes are based on ISO 4217. As PAN is a 16-digit credit card VOLUME 6, 2018 14281 K. Credit Card Fraud Detection Using AdaBoost and Majority Voting Table. Features in credit card data. Results of various individual models. number, a running sequence of numbers is used to mask the real numbers, in order to protect the personal information of customers. The results from various individual models are shown in results table. All accuracy rates are above 99%, with the exception of SVM at 95.5%. The non-fraud detection rates of NB, DT, and LIR are at 100%, while the rest are close to perfect, with the exception of SVM. The best MCC rates are from NB, DT, RF, and DS, at 0.990. The fraud detection rates vary from 7.4% for LIR up to 100% for RF, GBT, DS, NN, MLP, and LOR. Similar to the benchmark experiment, AdaBoost has been used with all individual models. The results are shown in Table 8. The accuracy and non-fraud detection rates are similar to those without AdaBoost. AdaBoost helps improve the fraud detection rates, with a noticeable difference for NB, DT, RT, which produce a perfect accuracy rate. The most results of AdaBoost and the results of majority voting significant improvement is achieved by LIR, i.e., from 7.4% to 94.1% accuracy. This clearly indicates the usefulness of AdaBoost in improvement the performance of individual classifiers. The best MCC score of 1 are achieved by NB and RF. The majority voting method is then applied to the same models used in the benchmark experiment. The results are shown in Table 9. The accuracy and non-fraud detection rates are perfect, or near perfect. DS+GBT, DT+DS, DT+GBT, and RF+GBT achieve a perfect fraud detection rate. The MCC scores are close to or at 1. The results of majority voting are better than those of individual models. To further evaluate the robustness of the machine learning algorithms, all real-world data samples are corrupted noise, at 10%, 20% and 30%. Noise is added to all data features. It can be seen that with the addition of noise, the fraud detection rate and MCC rates deteriorate, as expected.

7 ADVANTAGES & DISADVANTAGES



Model	Strengths	Limitations
Bayesian	Good for binary classification problems; efficient use of computational resources; suitable for real-time operations.	Need good understanding of typical and abnormal behaviors for different types of fraud cases
Trees	Easy to understand and implement; the procedures require a low computational power; suitable for real-time operations.	Potential of over-fitting if the training set does not represent the underlying domain information; re-training is required for new types of fraud cases.
Neural Network	Suitable for binary classification problems, and widely used for fraud detection.	Need a high computational power, unsuitable for real-time operations; re-training is required for new types of fraud cases.
Linear Regression	Provide optimal results when the relationship between independent and dependent variables are almost linear.	Sensitive to outliers and limited to numeric values only.
Logistic Regression	Easy to implement, and historically used for fraud detection.	Poor classification performances as compared with other data mining methods.
Support Vector Machine	Able to solve non-linear classification problems; require a low computational power; suitable for real-time operations.	Not easy to process the results due to transformation of the input data.

8 CONCLUSION

A study on credit card fraud detection using machine learning algorithms has been presented in this paper. A number of standard models which include NB, SVM, and DL have been used in the empirical evaluation. A publicly available credit card data set has been used for evaluation using individual (standard) models and hybrid models using AdaBoost and majority voting combination methods. The MCC metric has been adopted as a performance measure, as it takes into account the true and false positive and negative predicted outcomes. The best MCC score is 0.823, achieved using majority voting. A real credit card data set from a financial institution has also been used for evaluation. The same individual and hybrid models have been employed. A perfect MCC score of 1 has been achieved using AdaBoost and majority voting methods. To further evaluate the hybrid models, noise from 10% to 30% has been added into the data samples. The majority voting method has yielded the best MCC score of 0.942 for 30% noise added to the data set. This shows that the majority voting method offers robust performance in the presence of noise.

9 FUTURE SCOPE

`

For future work, the methods studied in this paper will be extended to online learning models. In addition, other online learning models will be investigated. The use of online learning will enable rapid detection of fraud cases, potentially in real-time. This in turn will help detect and prevent fraudulent transactions before they take place, which will reduce the number of losses incurred every day in the financial sector.

10 BIBILOGRAPHY

[1] Y. Sahin, S. Bulkan, and E. Duman, ''A cost-sensitive decision tree approach for fraud detection,'' Expert Syst. Appl., vol. 40, no. 15, pp. 5916–5923, 2013.

[2] A. O. Adewumi and A. A. Akinyelu, ''A survey of machine-learning and nature-inspired based credit card fraud detection techniques,'' Int. J. Syst. Assurance Eng. Manage., vol. 8, no. 2, pp. 937–953, 2017.

[3] A. Srivastava, A. Kundu, S. Sural, and A. Majumdar, ''Credit card fraud detection using hidden Markov model,'' IEEE Trans. Depend. Sec. Comput., vol. 5, no. 1, pp. 37–48, Jan. 2008.

[4] The Nilson Report. (Oct. 2016). [Online]. Available: https://www.nilson report.com/upload/content_promo/The_Nilson_Report_10-17-2016.pdf

[5] J. T. Quah and M. Sriganesh, ‘‘Real-time credit card fraud detection using computational intelligence,'' Expert Syst. Appl., vol. 35, no. 4, pp. 1721–1732, 2008.

[6] S. Bhattacharyya, S. Jha, K. Tharakunnel, and J. C. Westland, ‘‘Data mining for credit card fraud: A comparative study,'' Decision Support Syst., vol. 50, no. 3, pp. 602–613, 2011.

[7] N. S. Halvaiee and M. K. Akbari, ‘‘A novel model for credit card fraud detection using artificial immune systems,'' Appl. Soft Comput., vol. 24, pp. 40–49, Nov. 2014.

[8] S. Panigrahi, A. Kundu, S. Sural, and A. K. Majumdar, ‘‘Credit card fraud detection: A fusion approach using Dempster–Shafer theory and Bayesian learning,'' Inf. Fusion, vol. 10, no. 4, pp. 354–363, 2009.

[9] N. Mahmoudi and E. Duman, ‘‘Detecting credit card fraud by modified fisher discriminant analysis,'' Expert Syst. Appl., vol. 42, no. 5, pp. 2510–2516, 2015.

[10] D. Sánchez, M. A. Vila, L. Cerda, and J. M. Serrano, ‘‘Association rules applied to credit card fraud detection,'' Expert Syst. Appl., vol. 36, no. 2, pp. 3630–3640, 2009.

[11] E. Duman and M. H. Ozcelik, ‘‘Detecting credit card fraud by genetic algorithm and scatter search,'' Expert Syst. Appl., vol. 38, no. 10, pp. 13057–13063, 2011.

[12] P. Ravisankar, V. Ravi, G. R. Rao, and I. Bose, ‘‘Detection of financial statement fraud and feature selection using data mining techniques,'' Decision Support Syst., vol. 50, no. 2, pp. 491–500, 2011. [13] E. Kirkos, C. Spathis, and Y. Manolopoulos, ‘‘Data mining techniques for the detection of fraudulent financial statements,'' Expert Syst. Appl., vol. 32, no. 4, pp. 995–1003, 2007.

[14] F. H. Glancy and S. B. Yadav, ‘‘A computational model for financial reporting fraud detection,'' Decision Support Syst., vol. 50, no. 3, pp. 595–601, 2011.

15] D. Olszewski, ‘‘Fraud detection using self-organizing map visualizing the user profiles,’’ Knowl.-Based Syst., vol. 70, pp. 324–334, Nov. 2014.

[16] E. Rahimikia, S. Mohammadi, T. Rahmani, and M. Ghazanfari, ‘‘Detecting corporate tax evasion using a hybrid intelligent system: A case study of Iran,’’ Int. J. Account. Inf. Syst., vol. 25, pp. 1–17, May 2017.

[17] I. T. Christou, M. Bakopoulos, T. Dimitriou, E. Amolochitis, S. Tsekeridou, and C. Dimitriadis, ‘‘Detecting fraud in online games of chance and lotteries,’’ Expert Syst. Appl., vol. 38, no. 10, pp. 13158–13169, 2011.

[18] C.-F. Tsai, ‘‘Combining cluster analysis with classifier ensembles to predict financial distress,’’ Inf. Fusion, vol. 16, pp. 46–58, Mar. 2014.

[19] F. H. Chen, D. J. Chi, and J. Y. Zhu, ‘‘Application of random forest, rough set theory, decision tree and neural network to detect financial statement fraud—Taking corporate governance into consideration,’’ in Proc. Int. Conf. Intell. Comput. 2014, pp. 221–234.

[20] Y. Li, C. Yan, W. Liu, and M. Li, ‘‘A principle component analysis-based random forest with the potential nearest neighbor method for automobile insurance fraud identification,’’ Appl. Soft Comput., to be published, doi: 10.1016/j.asoc.2017.07.027.

[21] S. Subudhi and S. Panigrahi, ‘‘Use of optimized Fuzzy C-Means clustering and supervised classifiers for automobile insurance fraud detection,’’ J. King Saud Univ.-Comput. Inf. Sci., to be published, doi: 10.1016/j.jksuci.2017.09.010.

[22] M. Seera, C. P. Lim, K. S. Tan, and W. S. Liew, ‘‘Classification of transcranial Doppler signals using individual and ensemble recurrent neural networks,’’ Neurocomputing, vol. 249, pp. 337–344, Aug. 2017.

[23] E. Duman, A. Buyukkaya, and I. Elikucuk, ‘‘A novel and successful credit card fraud detection system implemented in a Turkish bank,’’ in Proc. IEEE 13th Int. Conf. Data Mining Workshops (ICDMW), Dec. 2013, pp. 162–171.

APPENDIX

A. Source code

```
# coding: utf-8

# # Predicting Credit Card Fraud

# The goal for this analysis is to predict credit card fraud in the transactional data. I will be using
tensorflow to build the predictive model, and t-SNE to visualize the dataset in two dimensions at the end
of this analysis. If you would like to learn more about the data, visit:
https://www.kaggle.com/dalpozz/creditcardfraud.
#
# The sections of this analysis include: Exploring the Data, Building the Neural Network, and Visualizing
the Data with t-SNE.

# In[143]:

import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.cross_validation import train_test_split
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix, f1_score, classification_report
import seaborn as sns
import matplotlib.gridspec as gridspec
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
from show_confusion_matrix import show_confusion_matrix


# In[2]:

df = pd.read_csv("creditcard.csv")


# ## Exploring the Data

# In[3]:

df.head()


# The data is mostly transformed from its original form, for confidentiality reasons.
# In[4]:

df.describe()


# In[5]:

df.isnull().sum()


# No missing values, that makes things a little easier.
#
# Let's see how time compares across fradulent and normal transactions.

# In[6]:

print ("Fraud")
print (df.Time[df.Class == 1].describe())
print ()
print ("Normal")
print (df.Time[df.Class == 0].describe())


# In[7]:

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(10,4))

bins = 50

ax1.hist(df.Time[df.Class == 1], bins = bins)
ax1.set_title('Fraud')

ax2.hist(df.Time[df.Class == 0], bins = bins)
ax2.set_title('Normal')

plt.xlabel('Time (in Seconds)')
plt.ylabel('Number of Transactions')
plt.show()


# The 'Time' feature looks pretty similar across both types of transactions. You could argue that
```

```
fraudulent transactions are more uniformly distributed, while normal transactions have a cyclical
distribution. This could make it easier to detect a fraudulent transaction during at an 'off-peak' time.

# Now let's see if the transaction amount differs between the two types.

# In[8]:

print ("Fraud")
print (df.Amount[df.Class == 1].describe())
print ()
print ("Normal")
print (df.Amount[df.Class == 0].describe())


# In[9]:

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(10,4))

bins = 30

ax1.hist(df.Amount[df.Class == 1], bins = bins)
ax1.set_title('Fraud')

ax2.hist(df.Amount[df.Class == 0], bins = bins)
ax2.set_title('Normal')

plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.yscale('log')
plt.show()


# In[10]:

df['Amount_max_fraud'] = 1
df.loc[df.Amount <= 2125.87, 'Amount_max_fraud'] = 0


# Most transactions are small amounts, less than \$100. Fraudulent transactions have a maximum value far
less than normal transactions, \$2,125.87 vs \$25,691.16.
#
# Let's compare Time with Amount and see if we can learn anything new.
# In[11]:

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(10,6))

ax1.scatter(df.Time[df.Class == 1], df.Amount[df.Class == 1])
ax1.set_title('Fraud')

ax2.scatter(df.Time[df.Class == 0], df.Amount[df.Class == 0])
ax2.set_title('Normal')

plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show()


# Nothing too useful here.
#
# Next, let's take a look at the anonymized features.

# In[12]:

# Select only the anonymized features.
v_features = df.ix[:,1:29].columns


# In[13]:

v_features


# In[14]:

plt.figure(figsize=(12,28*4))
gs = gridspec.GridSpec(28, 1)
for i, cn in enumerate(df[v_features]):
    ax = plt.subplot(gs[i])
    plt.hist(df[cn][df.Class == 1], bins=50, alpha = 0.4)
    plt.hist(df[cn][df.Class == 0], bins=50, alpha = 0.3)
    plt.yscale('log')
    ax.set_xlabel('')
    ax.set_title('histogram of feature: ' + str(cn))
```

```
plt.show()


# In[15]:

# Drop all of the features that have very similar distributions between the two types of transactions.
df = df.drop(['V28','V27','V26','V25','V24','V23','V22','V20','V15','V13','V8'], axis =1)


# In[16]:

# Based on the plots above, these features are created to identify values where fraudulent transaction are
more common.
df['V1_'] = df.V1.map(lambda x: 1 if x < -3 else 0)
df['V2_'] = df.V2.map(lambda x: 1 if x > 2.5 else 0)
df['V3_'] = df.V3.map(lambda x: 1 if x < -4 else 0)
df['V4_'] = df.V4.map(lambda x: 1 if x > 2.5 else 0)
df['V5_'] = df.V5.map(lambda x: 1 if x < -4.5 else 0)
df['V6_'] = df.V6.map(lambda x: 1 if x < -2.5 else 0)
df['V7_'] = df.V7.map(lambda x: 1 if x < -3 else 0)
df['V9_'] = df.V9.map(lambda x: 1 if x < -2 else 0)
df['V10_'] = df.V10.map(lambda x: 1 if x < -2.5 else 0)
df['V11_'] = df.V11.map(lambda x: 1 if x > 2 else 0)
df['V12_'] = df.V12.map(lambda x: 1 if x < -2 else 0)
df['V14_'] = df.V14.map(lambda x: 1 if x < -2.5 else 0)
df['V16_'] = df.V16.map(lambda x: 1 if x < -2 else 0)
df['V17_'] = df.V17.map(lambda x: 1 if x < -2 else 0)
df['V18_'] = df.V18.map(lambda x: 1 if x < -2 else 0)
df['V19_'] = df.V19.map(lambda x: 1 if x > 1.5 else 0)
df['V21_'] = df.V21.map(lambda x: 1 if x > 0.6 else 0)


# In[17]:

# Create a new feature for normal (non-fraudulent) transactions.
df.loc[df.Class == 0, 'Normal'] = 1
df.loc[df.Class == 1, 'Normal'] = 0


# In[18]:

# Rename 'Class' to 'Fraud'.
```

```
df = df.rename(columns={'Class': 'Fraud'})


# In[19]:

# 492 fraudulent transactions, 284,315 normal transactions.
# 0.172% of transactions were fraud.
print(df.Normal.value_counts())
print()
print(df.Fraud.value_counts())


# In[20]:

pd.set_option("display.max_columns",101)
df.head()


# In[21]:

# Create dataframes of only Fraud and Normal transactions.
Fraud = df[df.Fraud == 1]
Normal = df[df.Normal == 1]


# In[124]:

# Set X_train equal to 80% of the fraudulent transactions.
X_train = Fraud.sample(frac=0.8)
count_Frauds = len(X_train)

# Add 80% of the normal transactions to X_train.
X_train = pd.concat([X_train, Normal.sample(frac = 0.8)], axis = 0)

# X_test contains all the transaction not in X_train.
X_test = df.loc[~df.index.isin(X_train.index)]


# In[125]:

# Shuffle the dataframes so that the training is done in a random order.
X_train = shuffle(X_train)
```

```
X_test = shuffle(X_test)


# In[126]:

# Add our target features to y_train and y_test.
y_train = X_train.Fraud
y_train = pd.concat([y_train, X_train.Normal], axis=1)

y_test = X_test.Fraud
y_test = pd.concat([y_test, X_test.Normal], axis=1)


# In[127]:

# Drop target features from X_train and X_test.
X_train = X_train.drop(['Fraud','Normal'], axis = 1)
X_test = X_test.drop(['Fraud','Normal'], axis = 1)


# In[128]:

# Check to ensure all of the training/testing dataframes are of the correct length
print(len(X_train))
print(len(y_train))
print(len(X_test))
print(len(y_test))


# In[129]:

'''
Due to the imbalance in the data, ratio will act as an equal weighting system for our model.
By dividing the number of transactions by those that are fraudulent, ratio will equal the value that when
multiplied
by the number of fraudulent transactions will equal the number of normal transaction.
Simply put: # of fraud * ratio = # of normal
'''
ratio = len(X_train)/count_Frauds

y_train.Fraud *= ratio
y_test.Fraud *= ratio
```

```
# In[130]:

# Names of all of the features in X_train.
features = X_train.columns.values

# Transform each feature in features so that it has a mean of 0 and standard deviation of 1;
# this helps with training the neural network.
for feature in features:
    mean, std = df[feature].mean(), df[feature].std()
    X_train.loc[:, feature] = (X_train[feature] - mean) / std
    X_test.loc[:, feature] = (X_test[feature] - mean) / std


# ## Train the Neural Net

# In[219]:

# Split the testing data into validation and testing sets
split = int(len(y_test)/2)

inputX = X_train.as_matrix()
inputY = y_train.as_matrix()
inputX_valid = X_test.as_matrix()[:split]
inputY_valid = y_test.as_matrix()[:split]
inputX_test = X_test.as_matrix()[split:]
inputY_test = y_test.as_matrix()[split:]


# In[212]:

# Number of input nodes.
input_nodes = 37

# Multiplier maintains a fixed ratio of nodes between each layer.
mulitplier = 1.5

# Number of nodes in each hidden layer
hidden_nodes1 = 18
hidden_nodes2 = round(hidden_nodes1 * mulitplier)
hidden_nodes3 = round(hidden_nodes2 * mulitplier)
```

```python
# Percent of nodes to keep during dropout.
pkeep = tf.placeholder(tf.float32)


# In[213]:

# input
x = tf.placeholder(tf.float32, [None, input_nodes])

# layer 1
W1 = tf.Variable(tf.truncated_normal([input_nodes, hidden_nodes1], stddev = 0.15))
b1 = tf.Variable(tf.zeros([hidden_nodes1]))
y1 = tf.nn.sigmoid(tf.matmul(x, W1) + b1)

# layer 2
W2 = tf.Variable(tf.truncated_normal([hidden_nodes1, hidden_nodes2], stddev = 0.15))
b2 = tf.Variable(tf.zeros([hidden_nodes2]))
y2 = tf.nn.sigmoid(tf.matmul(y1, W2) + b2)

# layer 3
W3 = tf.Variable(tf.truncated_normal([hidden_nodes2, hidden_nodes3], stddev = 0.15))
b3 = tf.Variable(tf.zeros([hidden_nodes3]))
y3 = tf.nn.sigmoid(tf.matmul(y2, W3) + b3)
y3 = tf.nn.dropout(y3, pkeep)

# layer 4
W4 = tf.Variable(tf.truncated_normal([hidden_nodes3, 2], stddev = 0.15))
b4 = tf.Variable(tf.zeros([2]))
y4 = tf.nn.softmax(tf.matmul(y3, W4) + b4)

# output
y = y4
y_ = tf.placeholder(tf.float32, [None, 2])


# In[214]:

# Parameters
training_epochs = 2000
training_dropout = 0.9
display_step = 10
```

```
n_samples = y_train.shape[0]
batch_size = 2048
learning_rate = 0.005


# In[215]:

# Cost function: Cross Entropy
cost = -tf.reduce_sum(y_ * tf.log(y))

# We will optimize our model via AdamOptimizer
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

# Correct prediction if the most likely value (Fraud or Normal) from softmax equals the target value.
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))


# In[216]:

accuracy_summary = [] # Record accuracy values for plot
cost_summary = [] # Record cost values for plot
valid_accuracy_summary = []
valid_cost_summary = []
stop_early = 0 # To keep track of the number of epochs before early stopping

# Save the best weights so that they can be used to make the final predictions
checkpoint = "/Users/Dave/Desktop/Programming/Personal Projects/CreditCardFraud_Kaggle/best_model.ckpt"
saver = tf.train.Saver(max_to_keep=1)

# Initialize variables and tensorflow session
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for epoch in range(training_epochs):
        for batch in range(int(n_samples/batch_size)):
            batch_x = inputX[batch*batch_size : (1+batch)*batch_size]
            batch_y = inputY[batch*batch_size : (1+batch)*batch_size]

            sess.run([optimizer], feed_dict={x: batch_x,
                                             y_: batch_y,
                                             pkeep: training_dropout})
        # Display logs after every 10 epochs
        if (epoch) % display_step == 0:
            train_accuracy, newCost = sess.run([accuracy, cost], feed_dict={x: inputX,
                                                                            y_: inputY,
                                                                            pkeep: training_dropout})

            valid_accuracy, valid_newCost = sess.run([accuracy, cost], feed_dict={x: inputX_valid,
                                                                                  y_: inputY_valid,
                                                                                  pkeep: 1})

            print ("Epoch:", epoch,
                   "Acc =", "{:.5f}".format(train_accuracy),
                   "Cost =", "{:.5f}".format(newCost),
                   "Valid_Acc =", "{:.5f}".format(valid_accuracy),
                   "Valid_Cost = ", "{:.5f}".format(valid_newCost))

            # Save the weights if these conditions are met.
            if epoch > 0 and valid_accuracy > max(valid_accuracy_summary) and valid_accuracy > 0.999:
                saver.save(sess, checkpoint)

            # Record the results of the model
            accuracy_summary.append(train_accuracy)
            cost_summary.append(newCost)
            valid_accuracy_summary.append(valid_accuracy)
            valid_cost_summary.append(valid_newCost)

            # If the model does not improve after 15 logs, stop the training.
            if valid_accuracy < max(valid_accuracy_summary) and epoch > 100:
                stop_early += 1
                if stop_early == 15:
                    break
            else:
                stop_early = 0

    print()
    print("Optimization Finished!")
    print()

with tf.Session() as sess:
    # Load the best weights and show its results
    saver.restore(sess, checkpoint)
    training_accuracy = sess.run(accuracy, feed_dict={x: inputX, y_: inputY, pkeep: training_dropout})
    validation_accuracy = sess.run(accuracy, feed_dict={x: inputX_valid, y_: inputY_valid, pkeep: 1})

    print("Results using the best Valid_Acc:")
    print()
    print("Training Accuracy =", training_accuracy)
    print("Validation Accuracy =", validation_accuracy)


# In[217]:

# Plot the accuracy and cost summaries
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(10,4))

ax1.plot(accuracy_summary) # blue
ax1.plot(valid_accuracy_summary) # green
ax1.set_title('Accuracy')

ax2.plot(cost_summary)
ax2.plot(valid_cost_summary)
ax2.set_title('Cost')

plt.xlabel('Epochs (x10)')
plt.show()


# In[218]:

# Find the predicted values, then use them to build a confusion matrix
predicted = tf.argmax(y, 1)
with tf.Session() as sess:
    # Load the best weights
    saver.restore(sess, checkpoint)
    testing_predictions, testing_accuracy = sess.run([predicted, accuracy],
                                                     feed_dict={x: inputX_test, y_:inputY_test, pkeep: 1})

    print("F1-Score =", f1_score(inputY_test[:,1], testing_predictions))
    print("Testing Accuracy =", testing_accuracy)
    print()
    c = confusion_matrix(inputY_test[:,1], testing_predictions)
    show_confusion_matrix(c, ['Fraud', 'Normal'])
```

```
# Although the neural network can detect most of the fraudulent transactions (79.59%), there are still some
that got away. About 0.10% of normal transactions were classified as fraudulent, which can unfortunately
add up very quickly given the large number of credit card transactions that occur each minute/hour/day.
Nonetheless, this models performs reasonably well and I expect that if we had more data, and if the
features were not pre-transformed, we could have created new features, and built a more useful neural
network.

# ## Visualizing the Data with t-SNE

# First we are going to use t-SNE with the original data, then with the data we used for training our
neural network. I expect/hope that the second scatter plot will show a clearer contrast between the normal
and the fraudulent transactions. If this is the case, its signals that the work done during the feature
engineering stage of the analysis was beneficial to helping the neural network understand the data.

# In[5]:

# Reload the original dataset
tsne_data = pd.read_csv("creditcard.csv")


# In[6]:

# Set df2 equal to all of the fraulent and 10,000 normal transactions.
df2 = tsne_data[tsne_data.Class == 1]
df2 = pd.concat([df2, tsne_data[tsne_data.Class == 0].sample(n = 10000)], axis = 0)


# In[7]:

# Scale features to improve the training ability of TSNE.
standard_scaler = StandardScaler()
df2_std = standard_scaler.fit_transform(df2)

# Set y equal to the target values.
y = df2.ix[:,-1].values


# In[52]:

tsne = TSNE(n_components=2, random_state=0)
x_test_2d = tsne.fit_transform(df2_std)
```

```
# In[56]:

# Build the scatter plot with the two types of transactions.
color_map = {0:'red', 1:'blue'}
plt.figure()
for idx, cl in enumerate(np.unique(y)):
    plt.scatter(x = x_test_2d[y==cl,0],
                y = x_test_2d[y==cl,1],
                c = color_map[idx],
                label = cl)
plt.xlabel('X in t-SNE')
plt.ylabel('Y in t-SNE')
plt.legend(loc='upper left')
plt.title('t-SNE visualization of test data')
plt.show()


# The are two main groupings of fraudulent transactions, while the remaineder are mixed within the rest of
the data.
#
# Note: I have only used 10,000 of the 284,315 normal transactions for this visualization. I would have
liked to of used more, but my laptop crashes if many more than 10,000 transactions are included. With only
3.15% of the data being used, there should be some accuracy to this plot, but I am confident that the
layout would look different if all of the transactions were included.

# In[59]:

# Set df_used to the fraudulent transactions' dataset.
df_used = Fraud

# Add 10,000 normal transactions to df_used.
df_used = pd.concat([df_used, Normal.sample(n = 10000)], axis = 0)


# In[60]:

# Scale features to improve the training ability of TSNE.
df_used_std = standard_scaler.fit_transform(df_used)

# Set y_used equal to the target values.
```

```
y_used = df_used.ix[:,-1].values


# In[61]:

x_test_2d_used = tsne.fit_transform(df_used_std)


# In[62]:

color_map = {1:'red', 0:'blue'}
plt.figure()
for idx, cl in enumerate(np.unique(y_used)):
    plt.scatter(x=x_test_2d_used[y_used==cl,0],
                y=x_test_2d_used[y_used==cl,1],
                c=color_map[idx],
                label=cl)
plt.xlabel('X in t-SNE')
plt.ylabel('Y in t-SNE')
plt.legend(loc='upper left')
plt.title('t-SNE visualization of test data')
plt.show()


# It appears that the work we did in the feature engineering stage of this analysis has been for the best.
We can see that the fraudulent transactions are all part of a group of points. This suggests that it is
easier for a model to identify the fraudulent transactions in the testing data, and to learn about the
traits of the fraudulent transactions in the training data.

# In[ ]:
```