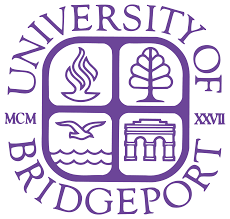
PROJECT REPORT

On

Credit Card Fraud Detection



By

Shreya Patel(1030301)

Year Spring 2019

**ACKNOWLEDGEMENT**

**I hereby take this opportunity to record my sincere thanks and heartily gratitude to Prof Miad Faezipour for her useful guidance and making available to me her intimate knowledge and experience for this project.**

**Abstract**

Credit Card Fraud Detection system is needed to handle the day by day increasing fraudulent. On a global scale, credit card processing fraud has hit [$32.320 trillion in total](https://www.nilsonreport.com/upload/content_promo/The_Nilson_Report_10-17-2016.pdf), with $21.84 billion lost in the US only. This data accounts for all sorts of transitions (online and in person), including transactions at POS, ATMs and those secured by PINs. These frauds often bear a recognizable pattern. With the help of machine learning, we can train your monitoring system to gather and analyze abnormal patterns and immediately switch to the credit fraud alert mode when things get awry.[3]

This system implements neural network algorithm machine learning model to implement supervised learning on real credit card transactions data set which learns by the training data. The neural network used in this project has 3 layers, the training data is input to the first layer and the third layer outputs the predicted output which is then compared to the actual output to measure the accuracy of the model.

This system uses TensorFlow which is an open source software library in python.

Table of Contents

[1 Introduction 2](#_Toc8174371)

[1.1 Problem Statement 2](#_Toc8174372)

[2 Background 3](#_Toc8174373)

[2.1 Classification 3](#_Toc8174374)

[2.2 Artificial Neural Networks 3](#_Toc8174375)

[2.3 Training a network 3](#_Toc8174376)

[2.3 Loss function 3](#_Toc8174377)

[3 Hardware and software 5](#_Toc8174378)

[3.1 Hardware 5](#_Toc8174379)

[3.2 Software 5](#_Toc8174380)

[3.2.1 Pycharm 5](#_Toc8174381)

[3.2.2 TensorFlow 5](#_Toc8174382)

[3.2.3 Numpy 5](#_Toc8174383)

[4 The data formatting process 6](#_Toc8174384)

[4.1 Formatting the data 6](#_Toc8174385)

[4.3 Spliting training and testing data 11](#_Toc8174386)

[4.4 Biased data reduction 12](#_Toc8174387)

[4.5 Formatting input matrix 12](#_Toc8174388)

[5 Methodology and Implementation 14](#_Toc8174389)

[5.1 Layers 14](#_Toc8174390)

[5.2 Loss function 14](#_Toc8174391)

[5.3 Other parameters 14](#_Toc8174392)

[6 Model accuracy and performance 16](#_Toc8174393)

[6.1 Accuracy results 16](#_Toc8174394)

[6.2 Network configuration 16](#_Toc8174395)

[6.3 Training data 17](#_Toc8174396)

[7 Conclusions 18](#_Toc8174397)

[7.1 Outlook 18](#_Toc8174398)

[8 References 19](#_Toc8174399)

# 1 Introduction

**Credit card fraud** is a wide-ranging term for [theft](https://en.wikipedia.org/wiki/Theft) and [fraud](https://en.wikipedia.org/wiki/Fraud) committed using or involving a [payment card](https://en.wikipedia.org/wiki/Payment_card), such as a [credit card](https://en.wikipedia.org/wiki/Credit_card) or [debit card](https://en.wikipedia.org/wiki/Debit_card), as a fraudulent source of funds in a transaction. The purpose may be to obtain goods without paying, or to obtain unauthorized funds from an account. Credit card fraud is also an adjunct to [identity theft](https://en.wikipedia.org/wiki/Identity_theft). Credit card fraud is a significant issue and has considerable cost for banks and car issuer companies. Thus, with this massive problem in transaction system, banks take credit card fraud very seriously, and have highly sophisticated security systems to monitor transaction and detect the frauds as quickly as possible once it is committed. A secured banking system requires high speed verification and authentication mechanisms. Fraud detection has become an important activity in order to decrease the impact of fraudulent transactions. There are various methods used for fraud detection each of them tries to increase the detection such as, Baysian algorithm, K-Nearest Neighbor, support vector machine etc. Fraud detection methods have been divided into two broad categories : Supervised and Unsupervised. Supervised learning is based on the learning data to predict new transaction as fraudulent or legitimate. While in unsupervised fraud detection, outliers or unusual transactions are identified as fraudulent transactions.[4]

## 

## 1.1 Problem Statement

The goal is to create a self learn model to predict fraud in a credit card transaction; the tasks involved are following:

* Install required software and libraries
* Download actual data which will be used to train and test our model
* Formatting data and dividing into groups
* Training the model using training data
* Testing the model with test data unknown to the model
* Measure accuracy of the resulting prediction

# 2 Background

## 

## 2.1 Classification

Classification is a method to extract information from data sets. This is done by dividing the data into categories based on some features. The idea is to derive a model which can perform the sorting process by training it on data objects where the category, or label, is known. The model should then be able to classify unlabeled data with sufficient accuracy. There are many

different models that are used for classification, e.g. neural networks.

## 

## 2.2 Artificial Neural Networks

Machine learning is a field in computer science aiming to imitate the human learning process. Artificial neural networks, or just neural networks, is a kind of machine learning technique where the structure of the human brain is the inspiration. The artificial neural network (ANN) is a network built of a number of interconnected neurons. The neurons are simple processing units that change their internal state, or activation, based on the current input and produces an output that depends on both the input and current activation.

The ANN is constructed by having a large number of these neurons working in parallel and connecting some neurons to others through weighted connections, creating a weighted and directed network of different layers. It is by adjusting these weighted connections and the internal activations of the neurons the ANN can be improved, or trained. Usually the network cycle through a set of training data sufficiently many times, until the weights have been adjusted enough to produce the desired output.

## 

## 2.3 Training a network

In order to adjust the weights correctly, we need a sufficiently large set of training data. There is no clear formula for how big this data set should be, but one aspect that is important to consider is variance between classes. If the disparity within a class is big, the number of training data objects should be larger. During the training process, for each *step*, one data object is run through the model and the weights are adjusted. When all training data has passed through the network once, one *epoch* is completed. The number of steps and epochs are important parts of the training process, as too few or too many can lead to under- or overfitting. A model is overfitted if it is too well adapted to the training data, but does not perform well in the general case. One possible cause of overfitting is if the classes in the training data are unbalanced. Underfitting occurs when the 10 chosen model does not fit well with the data and causes "overgeneralisation" by the model. One way to combat overfitting in neural networks is the use of so-called regularization.

## 2.3 Loss function”

An important part of the design of the network model, is choosing the loss function. The loss function represents the price paid for inaccuracy in predictions made by the neural network. By minimizing the loss function during the training process the error of the network also will be minimized. For classification problems, one of the most popular choices for loss function

is the softmax cross entropy functions, defined as:

*Hy*0(*y*) = −X*iy*0*ilog*(*yi*)*,”*

where y is the predicted probability function and y’ is the true

distribution.

# 

# 3 Hardware and software

## 3.1 Hardware

It is important to know the hardware that was used in the evaluation process. The training and evaluation of the neural network model has been done on a Windows 8 computer using a dual-core CPU at 3.4 GHz.

## 3.2 Software

### 3.2.1 Pycharm

**PyCharm** is an [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) used in [computer programming](https://en.wikipedia.org/wiki/Computer_programming), specifically for the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) language. It is developed by the Czech company [JetBrains](https://en.wikipedia.org/wiki/JetBrains" \o "JetBrains). It provides code analysis, a graphical debugger, an integrated unit tester, integration with [version control systems](https://en.wikipedia.org/wiki/Revision_control) (VCSes), and supports web development with [Django](https://en.wikipedia.org/wiki/Django_(web_framework)" \o "Django (web framework)). PyCharm is [cross-platform](https://en.wikipedia.org/wiki/Cross-platform), with [Windows](https://en.wikipedia.org/wiki/Windows), [macOS](https://en.wikipedia.org/wiki/MacOS" \o "MacOS) and [Linux](https://en.wikipedia.org/wiki/Linux) versions. The Community Edition is released under the [Apache License](https://en.wikipedia.org/wiki/Apache_License),[[7]](https://en.wikipedia.org/wiki/PyCharm" \l "cite_note-community-7) and there is also Professional Edition with extra features, released under a [proprietary license](https://en.wikipedia.org/wiki/Proprietary_software). I have written our project code in python using this IDE.[6]

### 3.2.2 TensorFlow

TensorFlow (TF) is an open source API developed by Google mainly for Machine Learning and Deep Learning, but it is also applicable for other numerical computations. Google uses TF in some of its commercial products, such as their speech recognition API and Gmail, but it is also used for research. The framework can be used both as backend, with C++, and frontend with Python. One of the advantages of using TF are the many built in functions. I are using the high-level API Estimators, which can be customized if needed. For this project, I have built our own estimator with a custom model. This model is utilizing the TF built in functions tf.*nn*, *tf.layers* and *tf.losses*, defining the structure and loss function of our neural network.[5]

### Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

* a powerful N-dimensional array object
* sophisticated (broadcasting) functions
* tools for integrating C/C++ and Fortran code
* useful linear algebra, Fourier transform, and random number capabilities

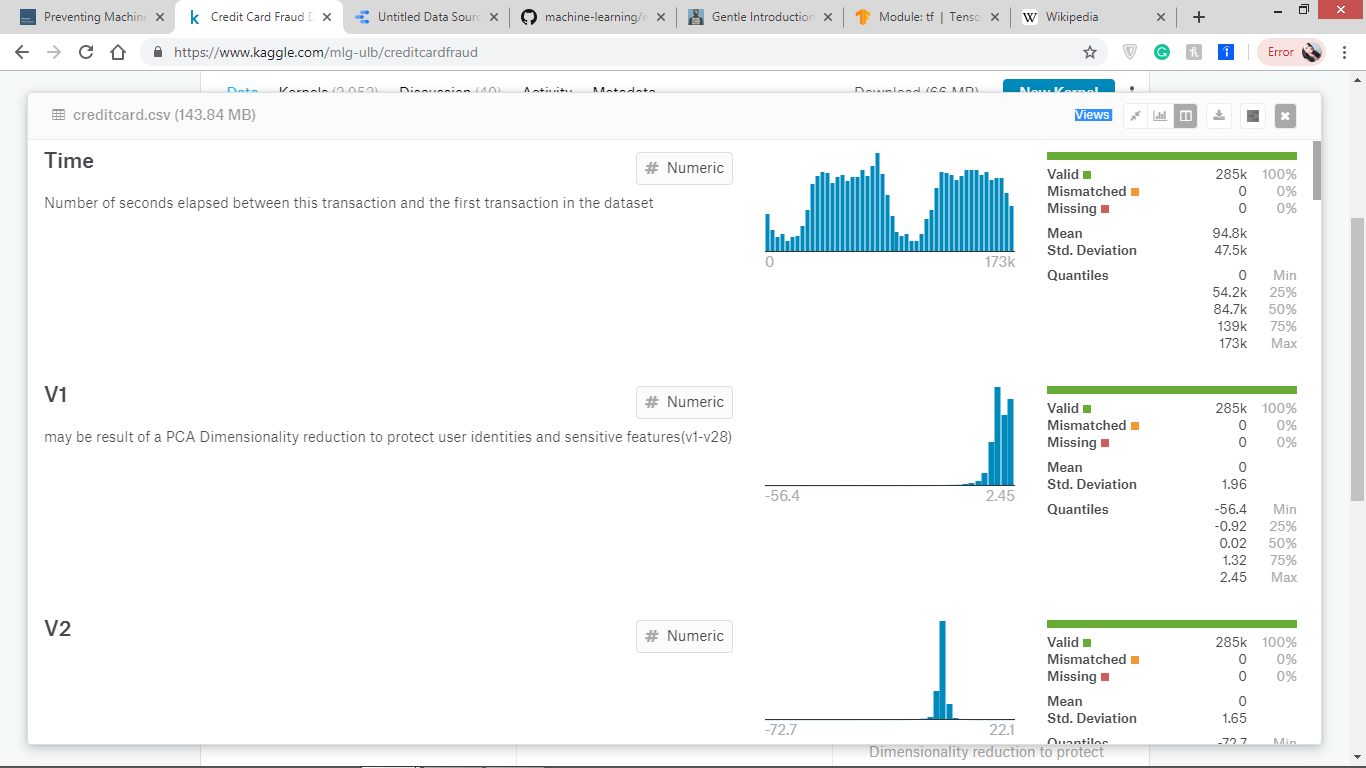
Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. In this system I will convert the data into numpy arrays before inputing it to the model.[8]

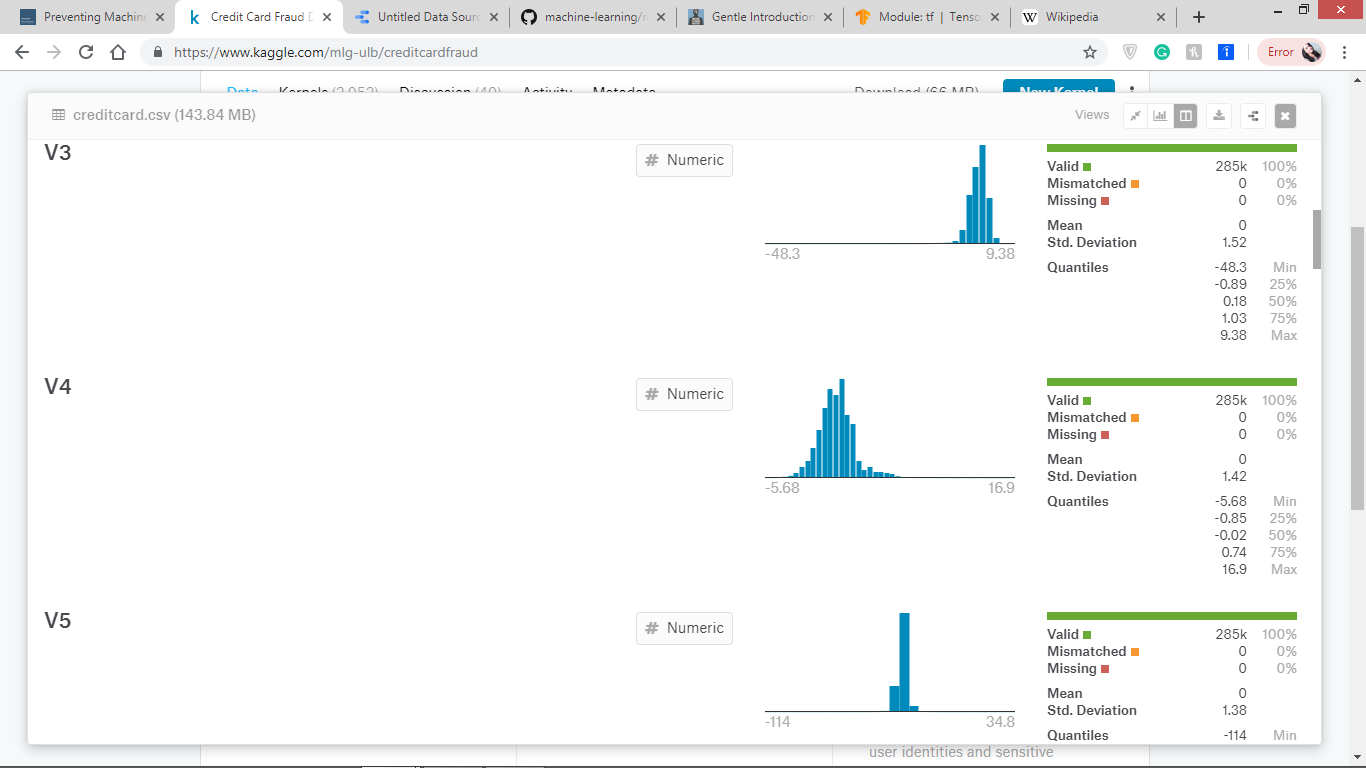
# 4 The data formatting process

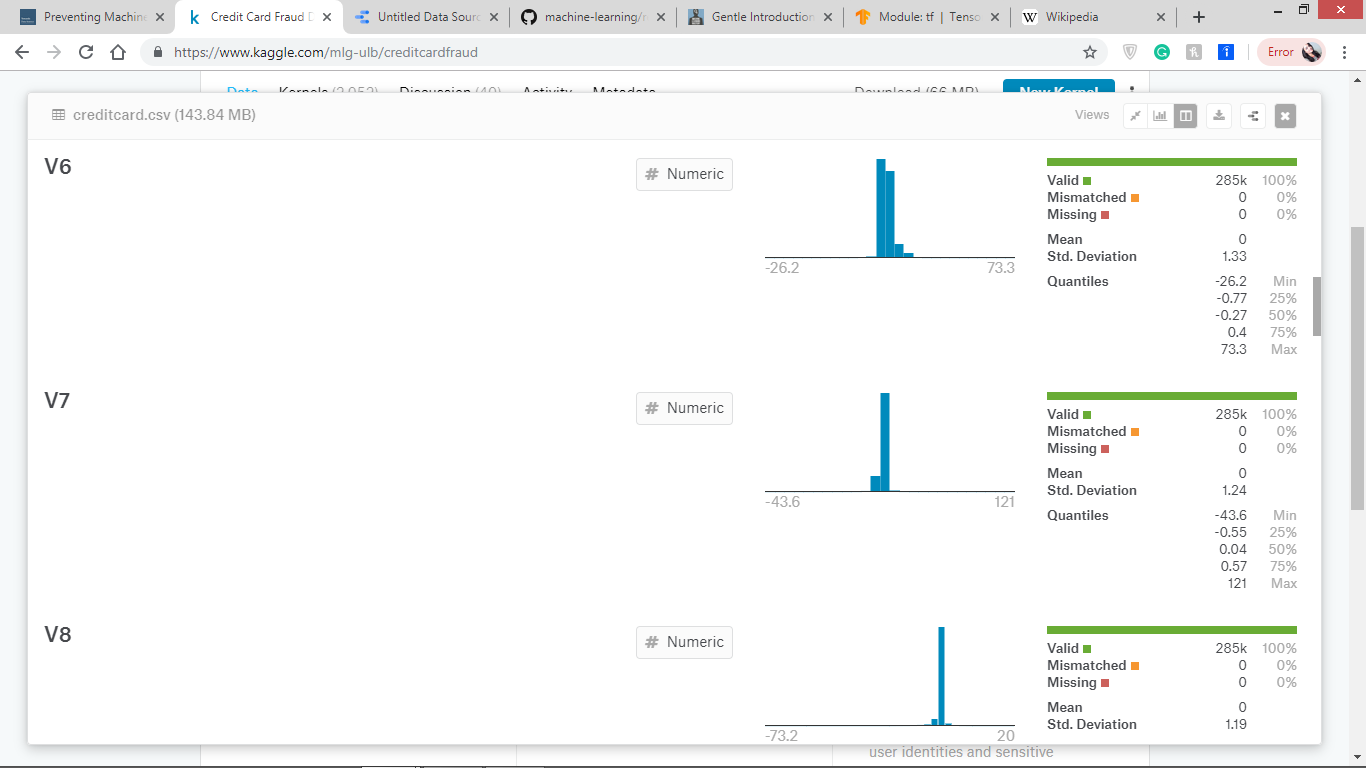
## 4.1 Formatting the data

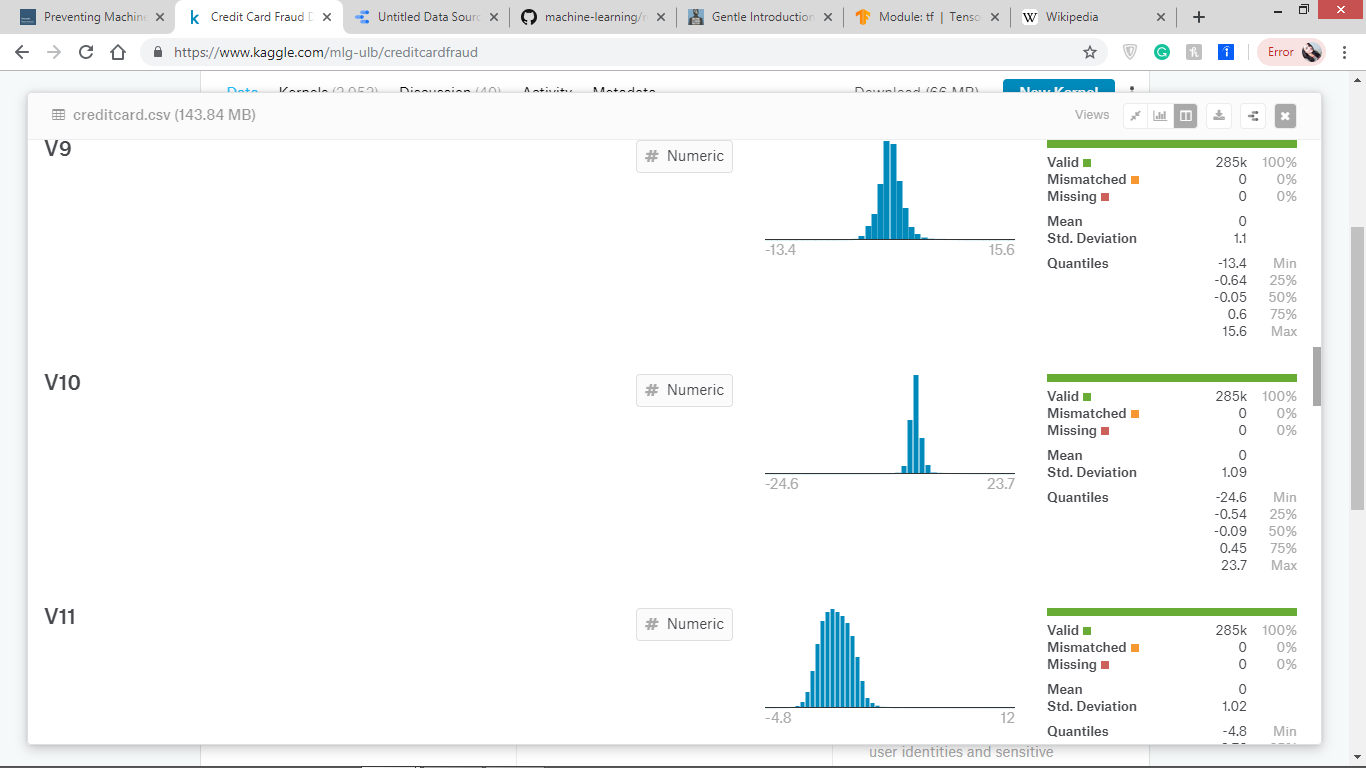
The datasets are downloaded from kaggle.com which contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

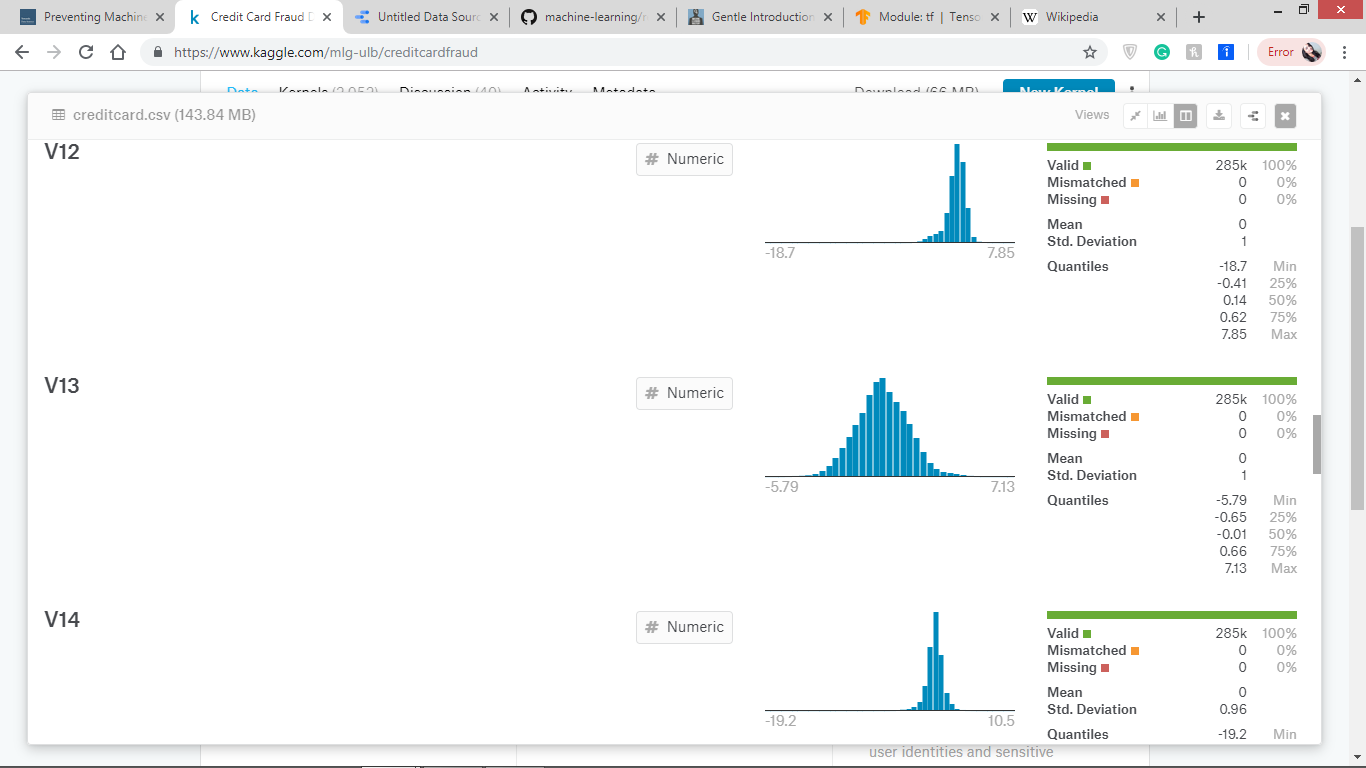
It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, they cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the 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 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.[3] Following is the data features graphs:

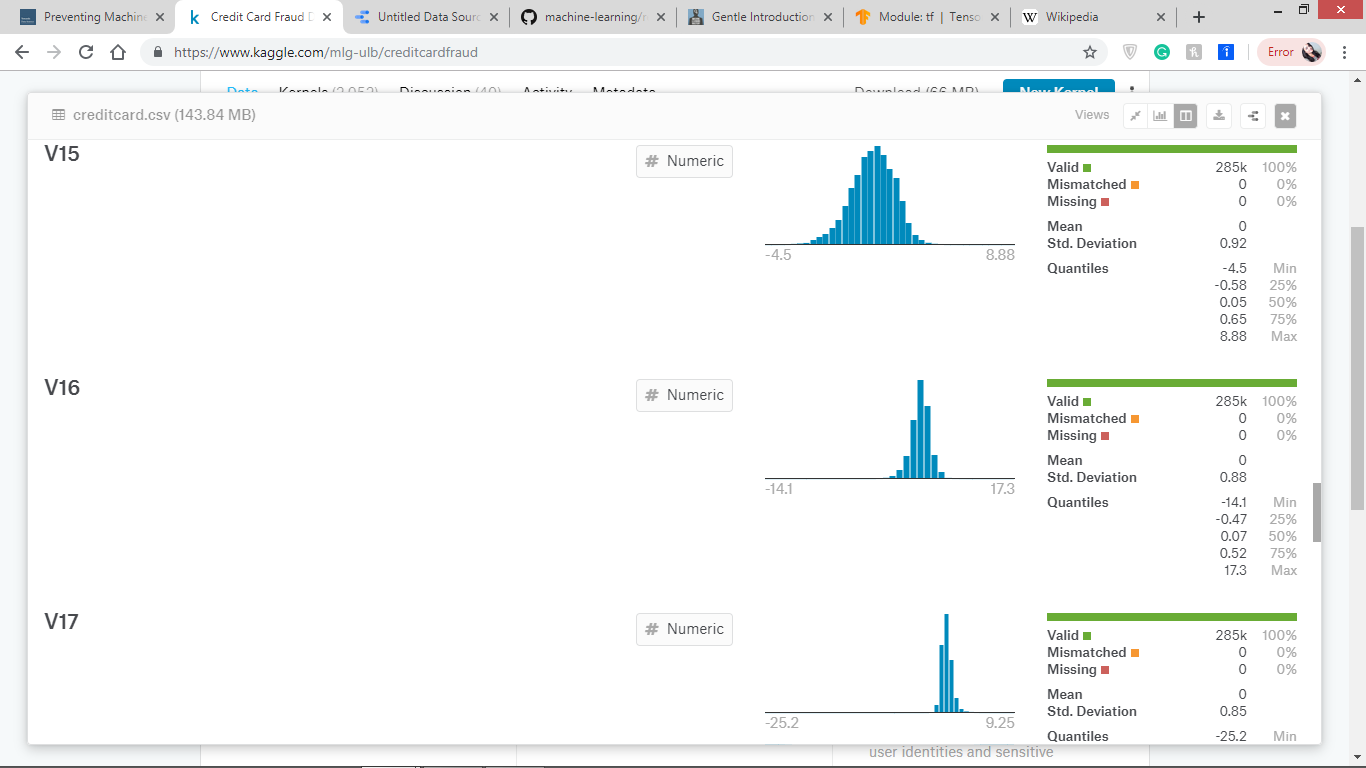


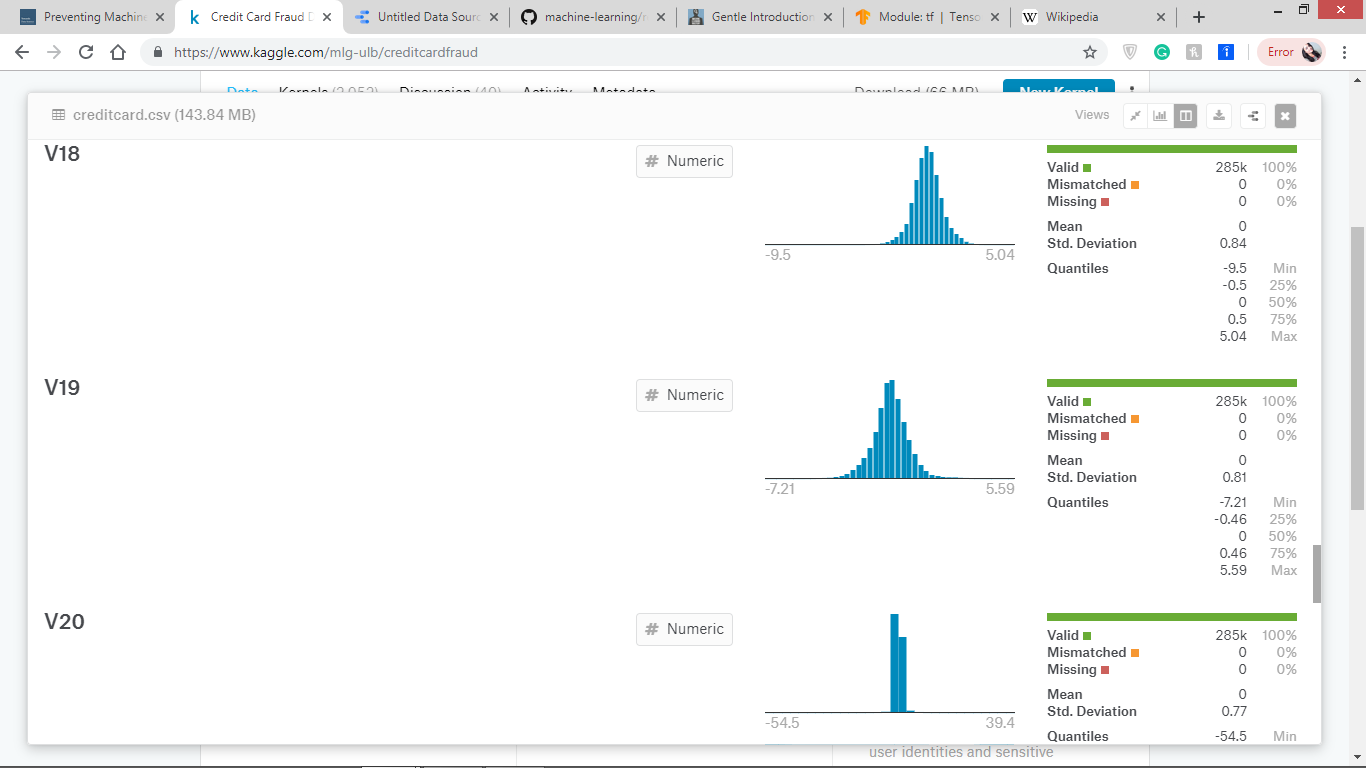


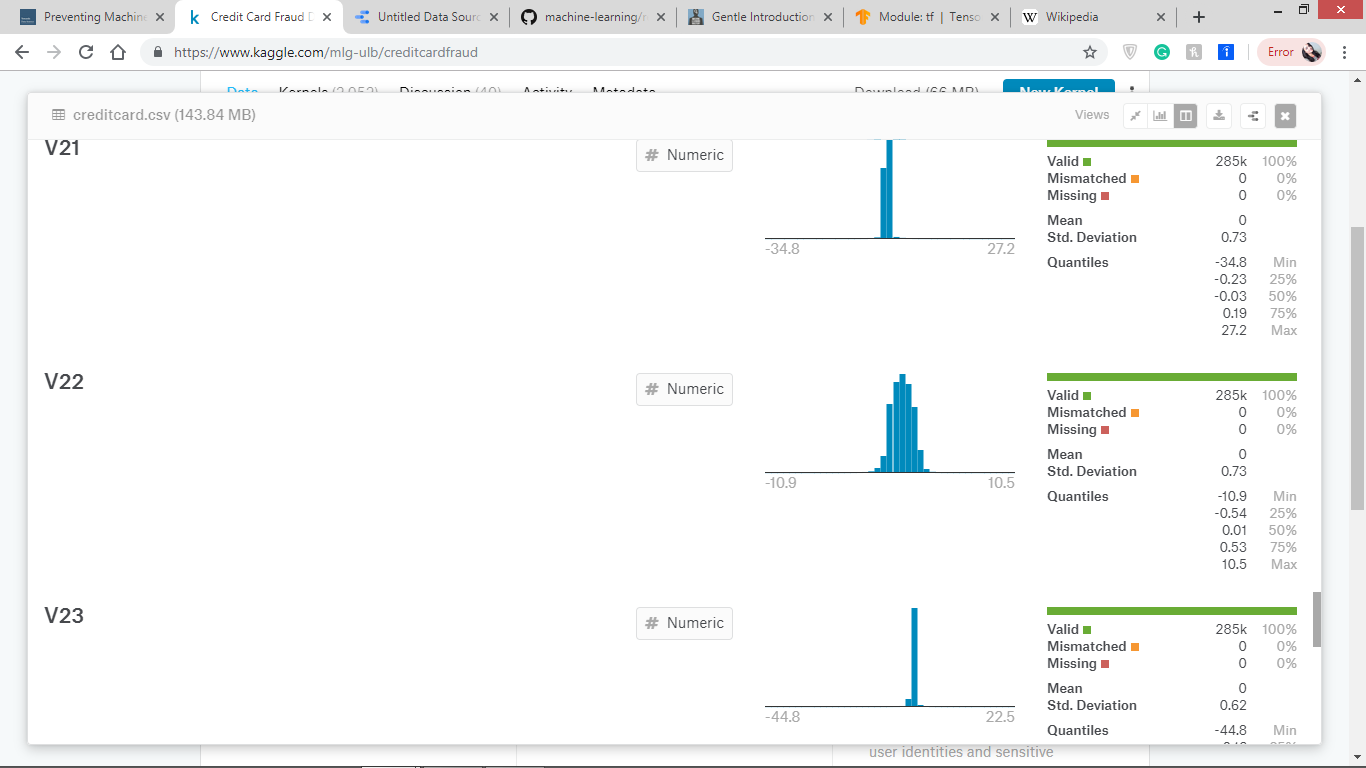


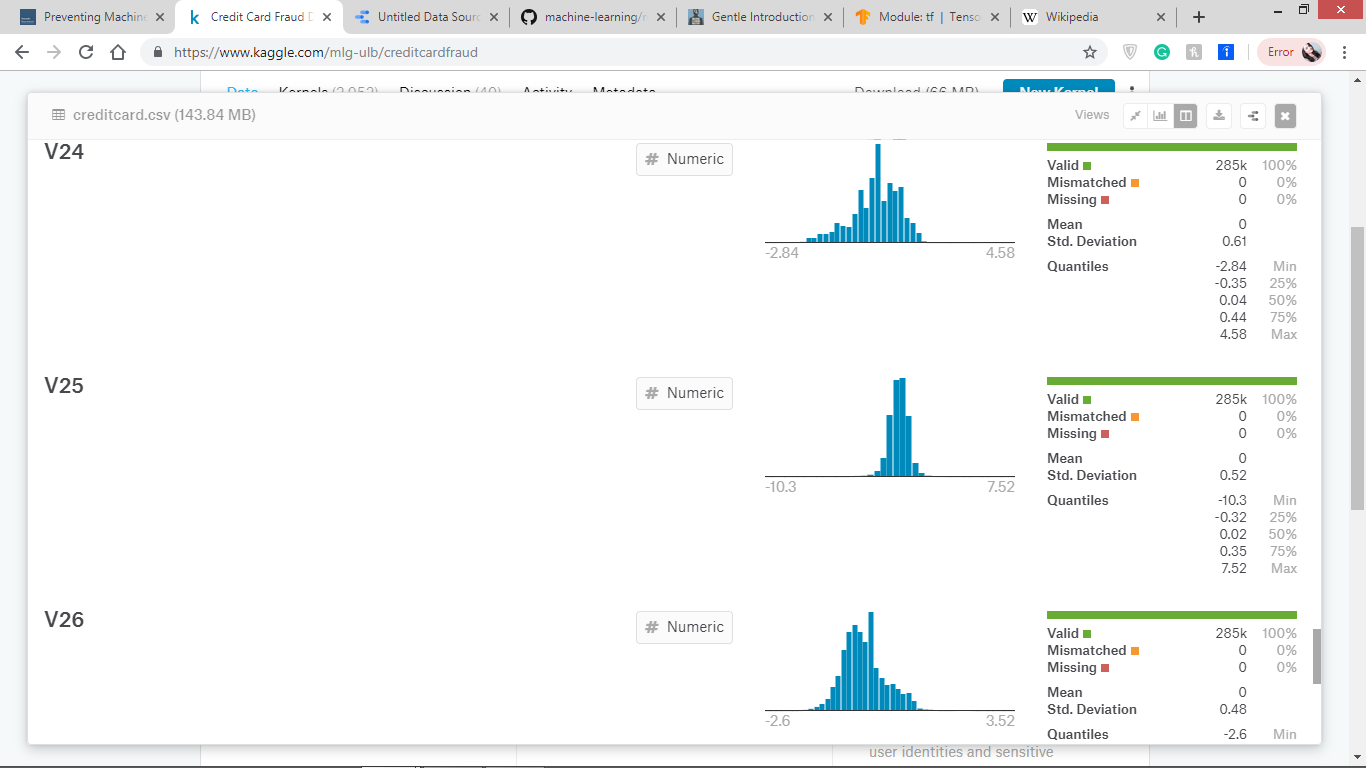


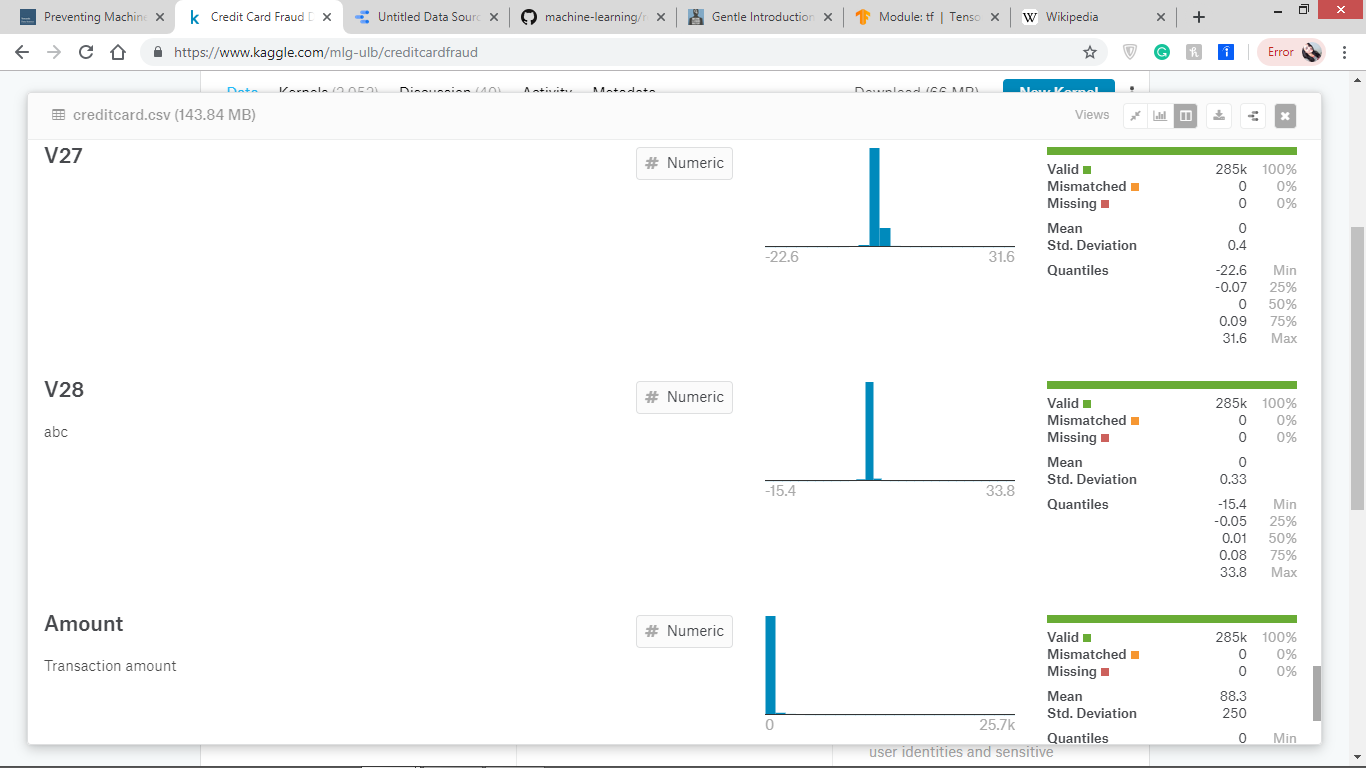












The data is imported to the system and then shuffled and converted into form of one hot encoding also, it is normalized to make it easier for our neural network functions to work on data which are in a specific form so as to increase the accuracy of our system.

“  
# Shuffle and randomize data  
shuffled\_data = credit\_card\_data.sample(frac=1)  
#print(shuffled\_data)  
# Change Class column into Class\_0 ([1 0] for legit data) and Class\_1 ([0 1] for fraudulent data)  
one\_hot\_data = pd.get\_dummies(shuffled\_data, columns=['Class'])  
#print(one\_hot\_data)  
# Change all values into numbers between 0 and 1  
normalized\_data = (one\_hot\_data - one\_hot\_data.min()) / (one\_hot\_data.max() - one\_hot\_data.min())

“

## 4.3 Spliting training and testing data

Formatted data is then converted into numpy arrays and then split into training and testing data. As the data is already random we take the 80% of data for training purpose and 20% to test our model. The feature values i.e v1,v2…v28 is the input and the resulting class that it belongs to (class 0/class1) is the output. Class 0 means the transaction is valid and class 1 means fraudulent.

ar\_X, ar\_y = np.asarray(df\_X.values, dtype='float32'), np.asarray(df\_y.values, dtype='float32')  
# Allocate first 80% of data into training data and remaining 20% into testing data  
train\_size = int(0.8 \* len(ar\_X))  
(raw\_X\_train, raw\_y\_train) = (ar\_X[:train\_size], ar\_y[:train\_size])  
(raw\_X\_test, raw\_y\_test) = (ar\_X[train\_size:], ar\_y[train\_size:])

## 4.4 Biased data reduction

As the data set is taken from actual transactions the number of fraudulent transactions are very less compared to the valid transactions. If the system is trained with more valid transactions than fraud then its accuracy to detect fraud will be low so the whole project would not have the accuracy needed. To remove this biasing a technique called logit weighting is applied. In this technique we apply some weight to the fraudulent transaction data so that while training the system performs more attention to the fraud transactions data. This weighting is applied to the training data which is later to be used to train the system.

“# Gets a percent of fraud vs legit transactions (0.0017% of transactions are fraudulent)  
count\_legit, count\_fraud = np.unique(credit\_card\_data['Class'], return\_counts=True)[1]

fraud\_ratio = float(count\_fraud / (count\_legit + count\_fraud))  
print('Percent of fraudulent transactions: ', fraud\_ratio)  
  
# Applies a logit weighting of 578 (1/0.0017) to fraudulent transactions to cause model to pay more attention to them  
weighting = 1 / fraud\_ratio+20  
print(weighting)  
  
raw\_y\_train[:, 1] = raw\_y\_train[:, 1] \* weighting

“

## 4.5 Formatting input matrix”

Setup dimensions for input of the neural network of the model. Also, creating node and weight default matrices.

input\_dimensions = ar\_X.shape[1]  
# 2 cells for the output  
output\_dimensions = ar\_y.shape[1]  
# 100 cells for the 1st layer  
num\_layer\_1\_cells = 100  
# 150 cells for the second layer  
num\_layer\_2\_cells = 150  
  
# We will use these as inputs to the model when it comes time to train it (assign values at run time)  
X\_train\_node = tf.placeholder(tf.float32, [None, input\_dimensions], name='X\_train')  
y\_train\_node = tf.placeholder(tf.float32, [None, output\_dimensions], name='y\_train')  
  
# We will use these as inputs to the model once it comes time to test it  
X\_test\_node = tf.constant(raw\_X\_test, name='X\_test')  
y\_test\_node = tf.constant(raw\_y\_test, name='y\_test')  
  
# First layer takes in input and passes output to 2nd layer  
weight\_1\_node = tf.Variable(tf.zeros([input\_dimensions, num\_layer\_1\_cells]), name='weight\_1')  
biases\_1\_node = tf.Variable(tf.zeros([num\_layer\_1\_cells]), name='biases\_1')  
  
# Second layer takes in input from 1st layer and passes output to 3rd layer  
weight\_2\_node = tf.Variable(tf.zeros([num\_layer\_1\_cells, num\_layer\_2\_cells]), name='weight\_2')  
biases\_2\_node = tf.Variable(tf.zeros([num\_layer\_2\_cells]), name='biases\_2')  
  
# Third layer takes in input from 2nd layer and outputs [1 0] or [0 1] depending on fraud vs legit  
weight\_3\_node = tf.Variable(tf.zeros([num\_layer\_2\_cells, output\_dimensions]), name='weight\_3')  
biases\_3\_node = tf.Variable(tf.zeros([output\_dimensions]), name='biases\_3')”

# 

# 5 Methodology and Implementation”

## 5.1 Layers

Our model has 3 layers:

The first layer uses sigmoid function: f(x)=1/(1+e-x)

Sigmoid function outputs in the range (0, 1), it makes it ideal for binary classification problems just as this problem where we need to find the probability of the data belonging to a particular class. The sigmoid function is differentiable at every point and its derivative comes out to be: f’(x)=f(x)\*(1-f(x)) . Since the expression involves the sigmoid function, its value can be reused to make the backward propagation faster.

The second layer is the hidden layer which uses the same sigmoid function but with the dropout function so as to increase the accuracy of our model and to prevent overfitting. The flatten layer reshapes the data and makes it ready to use in the prediction layer.

The third layer is the one which provides the predicted output which uses softmax function to convert the score matrix to probabilities to state too which class a particular training sample belongs.

def network(input\_tensor):  
 # Sigmoid fits modified data well  
 layer1 = tf.nn.sigmoid(tf.matmul(input\_tensor, weight\_1\_node) + biases\_1\_node)  
 # Dropout prevents model from becoming lazy and over confident  
 layer2 = tf.nn.dropout(tf.nn.sigmoid(tf.matmul(layer1, weight\_2\_node) + biases\_2\_node), 0.85)  
 # Softmax works very well with one hot encoding which is how results are outputted  
 layer3 = tf.nn.softmax(tf.matmul(layer2, weight\_3\_node) + biases\_3\_node)  
 return layer3”

## 5.2 Loss function

This model uses the softmax cross entropy function in the *tf.losses* module. “The parameters available for the loss function include weights, scope, reduction and smoothing. Weights can be used to bias the network toward a certain class, e.g. to combat overfitting. The optimization method is also a parameter for the network.

# Cross entropy loss function measures differences between actual output and predicted output  
cross\_entropy = tf.losses.softmax\_cross\_entropy(y\_train\_node, y\_train\_prediction)”

## 5.3 Other parameters

The batch size of the network is the number of data points passed through the network in each propagation. After one batch is processed, the weights are updated. The learning rate is a measure of how much the model is inclined to abandon the belief that a certain feature is most predominant, and instead choose another feature. Both too high and too low learning rates can make the training process longer. Parameters for the training process, such as the number of steps for the training, are also available for user modification.”

The loss function used in our neural network is softmax cross entropy. We are not using any of the modification parameters available in TensorFlow, but will discuss possible improvements regarding the loss function in the discussion section of the report. We settled for a learning rate of 0.0045, which was sufficient in our case. For the optimization method, we used Adam optimizer.

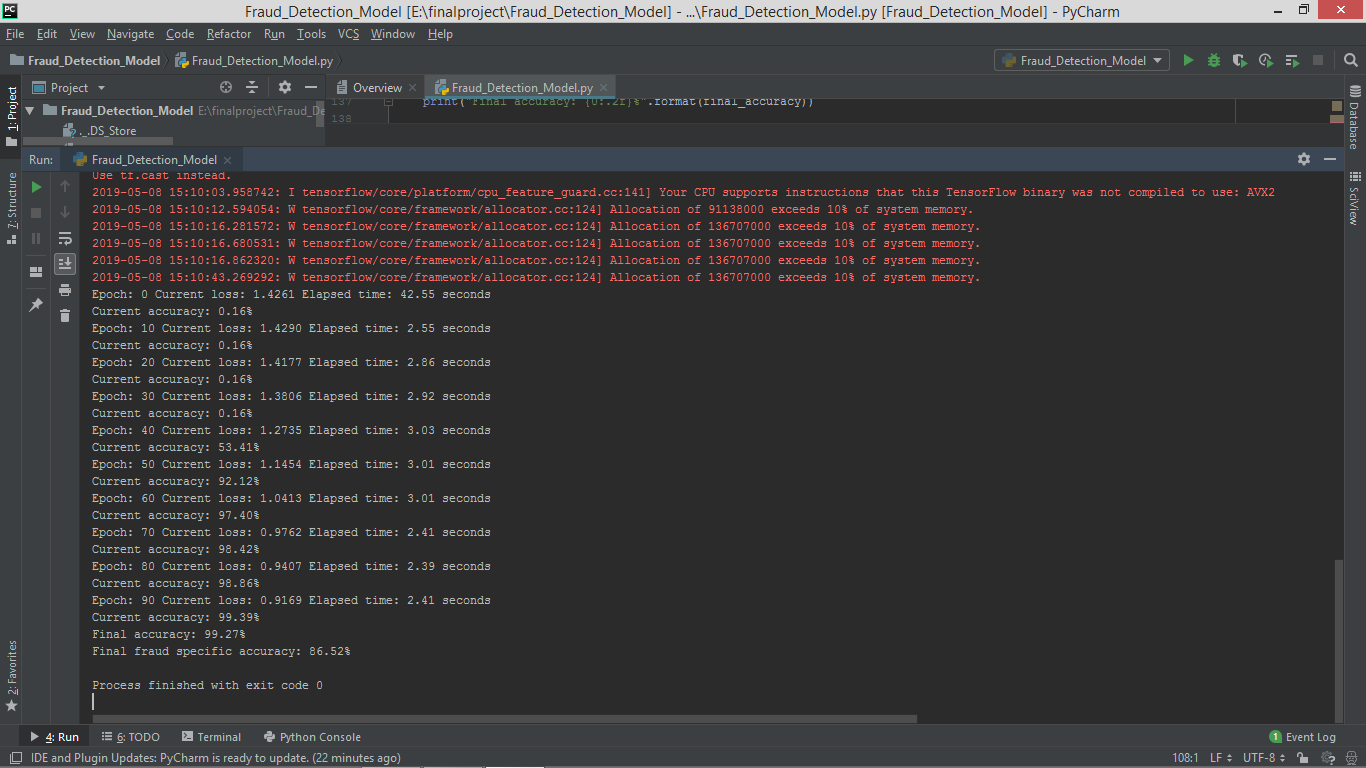
The number of steps in the training process was a parameter that we investigated when we evaluated the accuracy of the network. The results are presented in the network performance section of the report.

with tf.Session() as session:  
 tf.global\_variables\_initializer().run()  
 for epoch in range(num\_epochs):  
  
 start\_time = time.time()  
  
 \_, cross\_entropy\_score = session.run([optimizer, cross\_entropy],  
 feed\_dict={X\_train\_node: raw\_X\_train, y\_train\_node: raw\_y\_train})  
  
 if epoch % 10 == 0:  
 timer = time.time() - start\_time  
  
 print('Epoch: {}'.format(epoch), 'Current loss: {0:.4f}'.format(cross\_entropy\_score),  
 'Elapsed time: {0:.2f} seconds'.format(timer))  
  
 final\_y\_test = y\_test\_node.eval()  
 final\_y\_test\_prediction = y\_test\_prediction.eval()  
 final\_accuracy = calculate\_accuracy(final\_y\_test, final\_y\_test\_prediction)  
 print("Current accuracy: {0:.2f}%".format(final\_accuracy))  
  
 final\_y\_test = y\_test\_node.eval()  
 final\_y\_test\_prediction = y\_test\_prediction.eval()  
 final\_accuracy = calculate\_accuracy(final\_y\_test, final\_y\_test\_prediction)  
 print("Final accuracy: {0:.2f}%".format(final\_accuracy))  
  
final\_fraud\_y\_test = final\_y\_test[final\_y\_test[:, 1] == 1]  
final\_fraud\_y\_test\_prediction = final\_y\_test\_prediction[final\_y\_test[:, 1] == 1]  
final\_fraud\_accuracy = calculate\_accuracy(final\_fraud\_y\_test, final\_fraud\_y\_test\_prediction)  
print('Final fraud specific accuracy: {0:.2f}%'.format(final\_fraud\_accuracy))

# 6 Model accuracy and performance

## 6.1 Accuracy results

When the project first started I had hoped to reach an accuracy of more than 90 percent for fraud detection of this model. However, within the scope of this project, I did manage to meet these expectations for either of our chosen paths. As this models accuracy is near to 100% if taken as whole which is pretty good.



Regarding the accuracy of the number of fraudulent transactions in this model I believe that the low accuracy, around 87 % as sufficient, can be partly attributed to the small size of the dataset. Since the labeling had to be done manually we couldn’t produce enough data, or data of assured quality, to properly train the neural network model. The tests we did does however show promise for the setup.

I do not believe that we have reached the best possible accuracy for a neural network.

def calculate\_accuracy(actual, predicted):  
 actual = np.argmax(actual, 1)  
 predicted = np.argmax(predicted, 1)  
 return (100 \* np.sum(np.equal(predicted, actual)) / predicted.shape[0])

“

## 6.2 Network configuration

Due to the time limitations of this project, I am not able to thoroughly investigate the ideal parameters of our network. As almost every parameter affects how the network trains and, following that, how good it becomes at classifying its input it is something that logically should have an effect on the result. I did look into some of the parameters of the training process, but the structure of the network as well as the parameters for the layers and the loss function could use more work. As noted in the result section, I experienced overfitting with our network. One way to counter this could be to bias the network toward the less represented classes, by changing how the weight parameters are adjusted by the loss function.

## 6.3 Training data

A likely cause of the problem with overfitting was the dataset used for training. It had an unbalanced class distribution and to get better training data, a few different things could be done. Balancing the bias can be done by removing non-fraud transactions. Getting more training data would allow for some preprocessing methods.

# 7 Conclusions”

I would recommend further investigations into the use of neural networks for the Credit card fraud detection project, as I believe that it has great potentials in many aspects. The specific work I did in this project, implementing a NN for identification of fraud, show promise with regards to computational performance. Concerning the accuracy, the network did not perform as well as I would have hoped. However, I believe that with some improvements of the training data set and network parameters I could improve the accuracy sufficiently.

## 7.1 Outlook

The future development I would recommend includes

• Improving the training data by several means: simulate for better

balance, using preprocessing, get more data for the training and testing of fraud detection.

• Customizing the loss function if well balanced training can not be achieved.

• Tuning and customizing the parameters of the network, regarding structure, layers, filters etc.

# “

# 8 References

1. <https://www.tensorflow.org/api_docs/python/tf>
2. <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>
3. <https://www.kaggle.com/mlg-ulb/creditcardfraud>
4. <https://www.fool.com/the-ascent/credit-cards/blog/how-do-credit-card-companies-spot-fraud/>
5. [https://www.wikipedia.org](https://www.wikipedia.org/)/Tensorflow
6. <https://en.wikipedia.org/wiki/PyCharm>
7. <https://towardsdatascience.com/preventing-machine-learning-bias-d01adfe9f1fa>
8. <https://www.numpy.org/>