Anomaly Detector

Distributed patterns to detect anomalies in time-series data

Team

in alphabetical order

Ankit Kumar Singh	U101114FCS045
Ashwin Pilgaonkar	U101114FCS052
Jeel Shah	U101114FCS172
Padegal Pranavi	U101114FCS224
Saumya Gupta	U101114FCS128
Surajnath Sidh	U101114FCS146

Objective

Problem Statement

Build an Anomaly detector to detect anomalies in time-series data

We are choosing credit card fraud detection for our project

Work Done

Team contribution

Work division

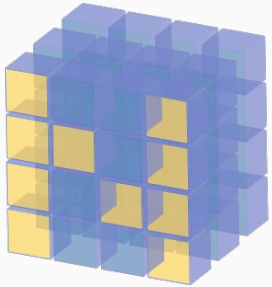
Suraj & Ashwin - Research, Building Neural Network with Tensorflow & Evaluation of Network
Jeel, Pranavi, Saumya & Ankit - Data exploration and Feature Engineering

During our meetings we made sure to explain our work to each other and helped wherever anyone had issues

Tools & Technologies

python™

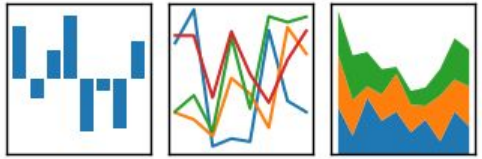
matplotlib



NumPy

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$




Trello


jupyter


spyder


TensorFlow

Experiments and Results

Finding Dataset

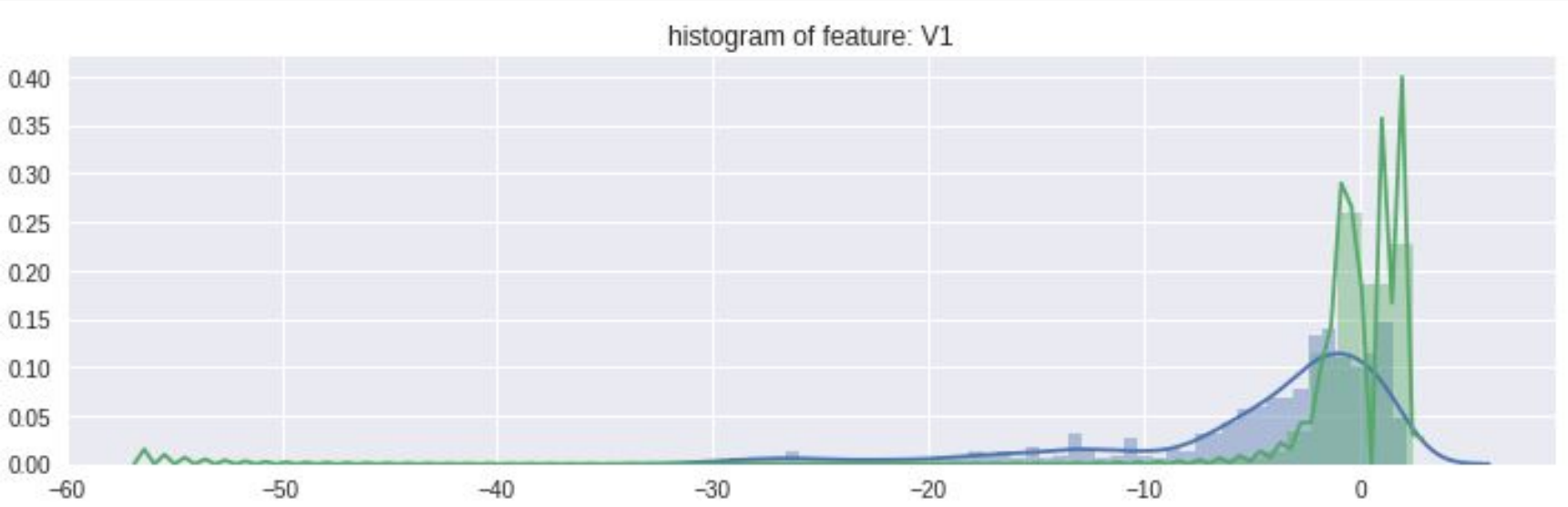
- We are focusing on Application of Anomaly Detection in Financial Domain(Fraud Detection).

- We are using anonymized and preprocessed dataset used by *Dal Pozzolo, Andrea, and Gianluca Bontempi. "Adaptive machine learning for credit card fraud detection." (2015).*

Feature Engineering

Plotting Feature Histograms

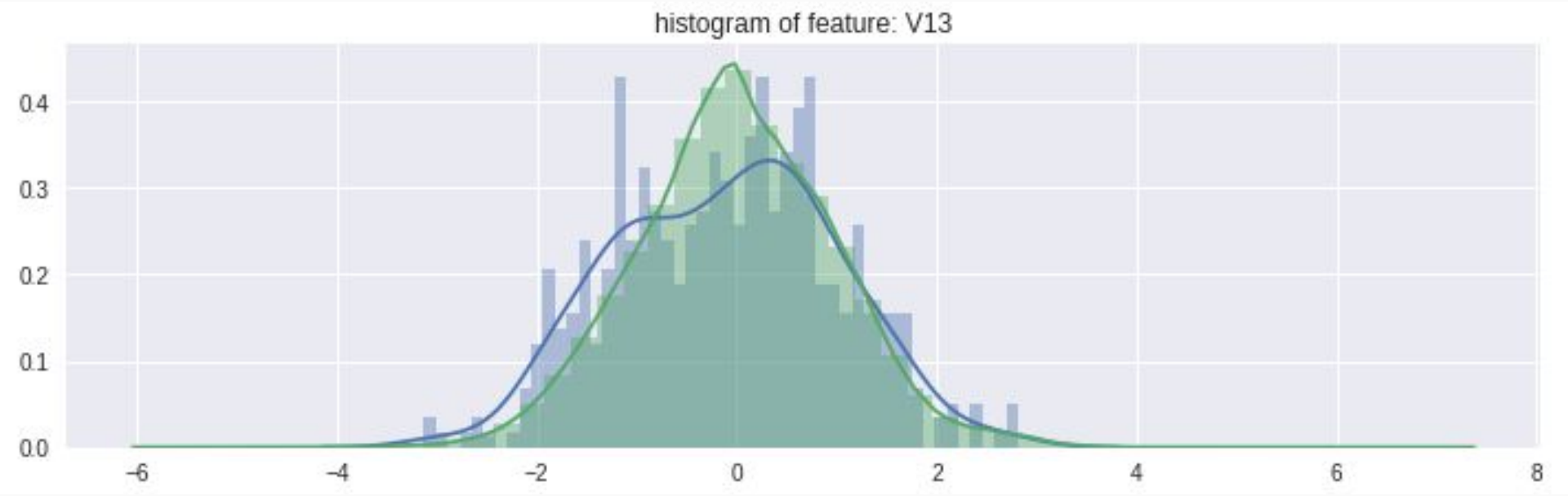
Draw a histogram of each feature



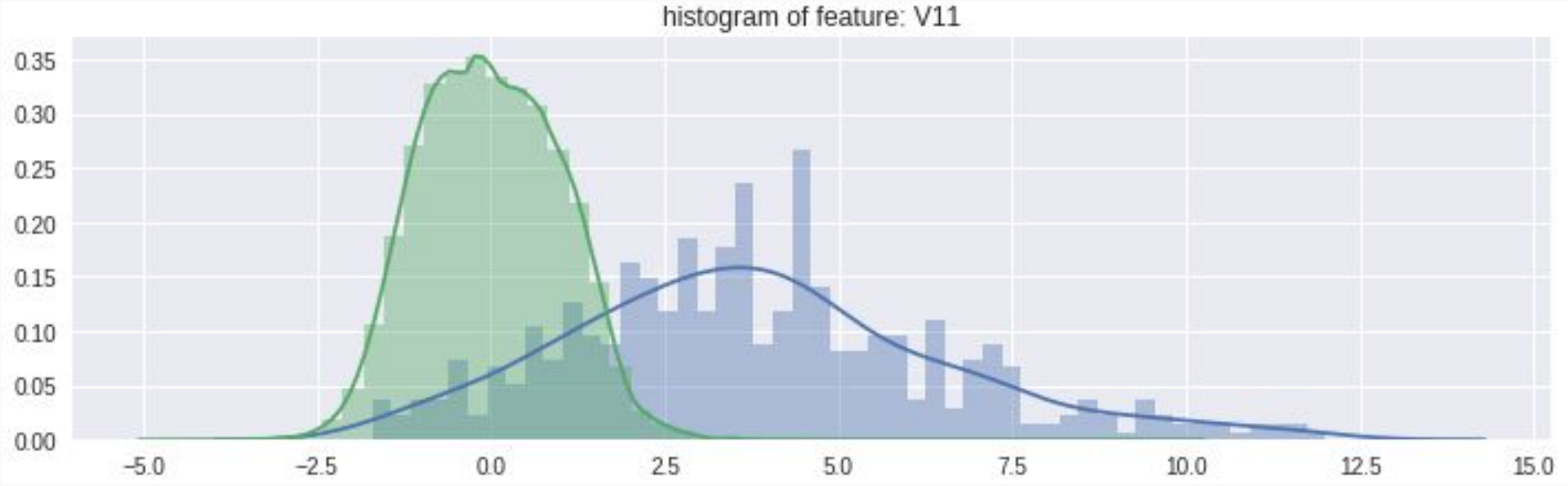
Drop less important features

- Drop all the features that have similar distributions between fraud and non-fraud transactions
- We compared the distributions using *np.percentile()* If the distributions were too similar, We dropped the feature
- Based on the plots, new features are created to identify values where fraudulent transaction are more clear to see.

Less important feature



More important feature



Account for unbalanced classes in data

- Due to the imbalance in the data, ratio will act as an equal weighting system for the 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.

Prepare training and testing data

Training Set

- Add 80% of the fraudulent transactions in training data
- Add 80% of the normal transactions in training data

Testing Set (Validation Set)

- Add rest of 20% of the fraudulent and normal transactions to test data

Shuffle training and testing data

- Shuffle the data frames to ensure that the training is done in a random order and neural network does not learn about inherent order of data.

```
X_train = shuffle(X_train)
X_test = shuffle(X_test)
```

Neural Network

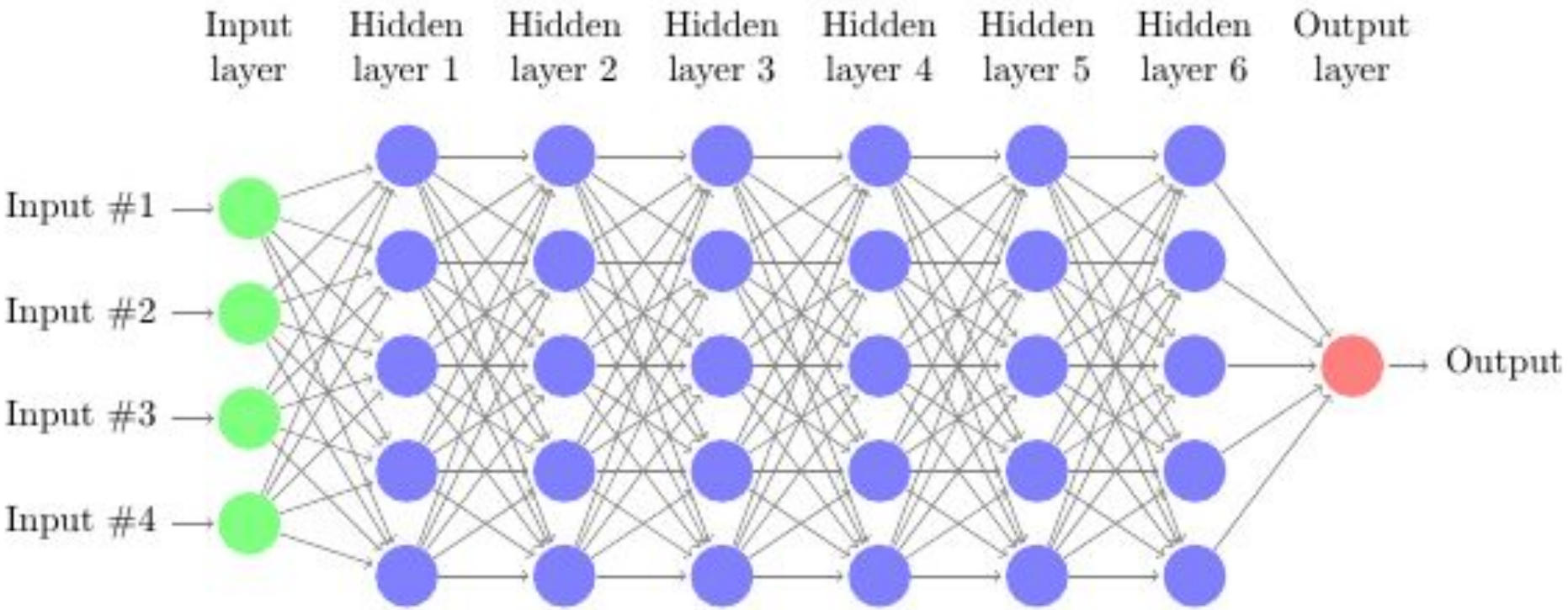
Neural Network

We are using multilayer structure, with

- 1 Input and 1 Output layer
- 4 Hidden layer

Cost function: Cross Entropy

Optimizer: Adam Optimizer (an optimizer from Gradient Descent family)

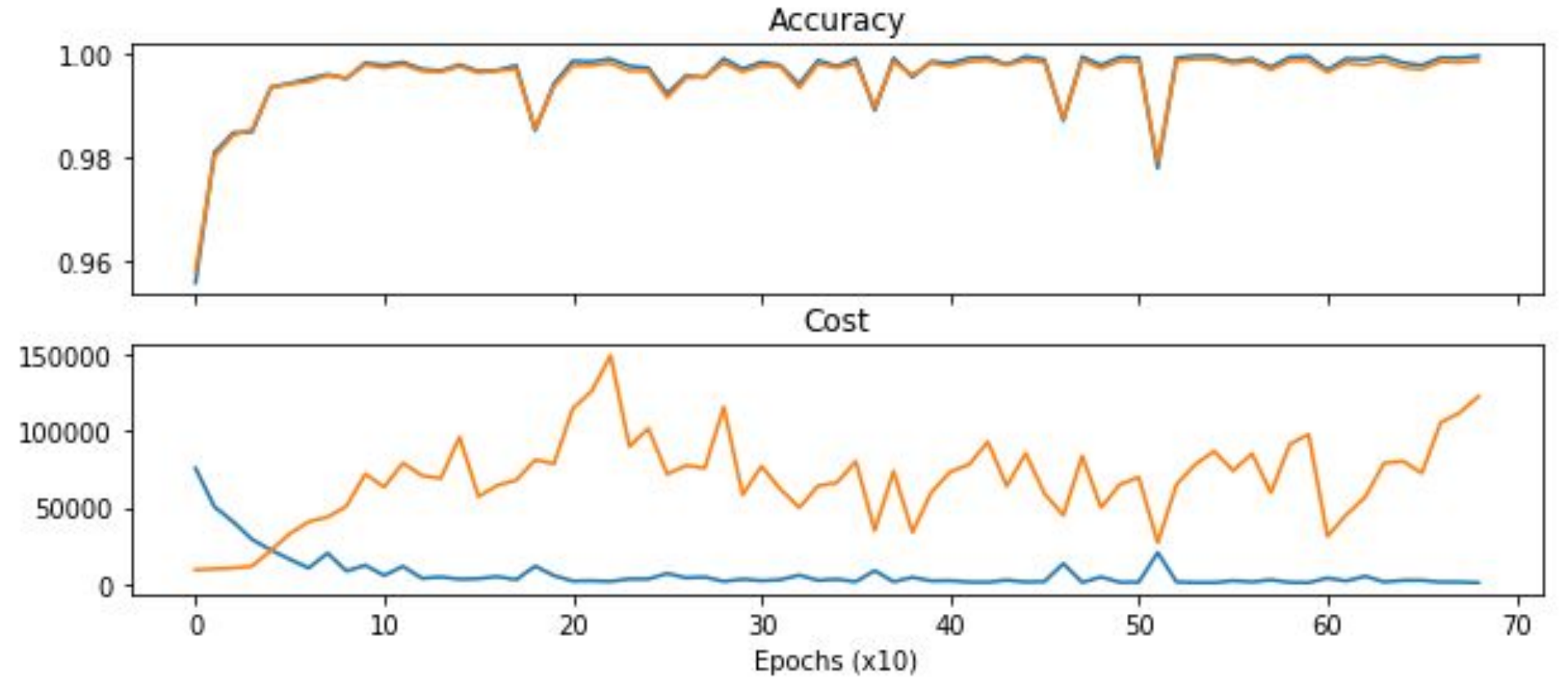

Input layer
Hidden layer 1
Hidden layer 2
Hidden layer 3
Hidden layer 4
Hidden layer 5
Hidden layer 6
Output layer
Input #1
Input #2
Input #3
Input #4
Output

Training

```
Epoch: 550 Acc = 0.99843 Cost = 2309.86304 Valid_Acc = 0.99803 Valid_Cost =  73952.83594
Epoch: 560 Acc = 0.99899 Cost = 1477.31189 Valid_Acc = 0.99846 Valid_Cost =  85214.71875
Epoch: 570 Acc = 0.99731 Cost = 2832.08936 Valid_Acc = 0.99680 Valid_Cost =  59580.73047
Epoch: 580 Acc = 0.99927 Cost = 1183.93884 Valid_Acc = 0.99856 Valid_Cost =  91285.68750
Epoch: 590 Acc = 0.99940 Cost = 1013.15448 Valid_Acc = 0.99856 Valid_Cost =  97696.12500
Epoch: 600 Acc = 0.99681 Cost = 4024.29736 Valid_Acc = 0.99635 Valid_Cost =  31208.68945
Epoch: 610 Acc = 0.99896 Cost = 2060.65454 Valid_Acc = 0.99814 Valid_Cost =  45282.00000
Epoch: 620 Acc = 0.99885 Cost = 5213.19141 Valid_Acc = 0.99775 Valid_Cost =  56822.42969
Epoch: 630 Acc = 0.99938 Cost = 1453.96887 Valid_Acc = 0.99853 Valid_Cost =  78973.88281
Epoch: 640 Acc = 0.99817 Cost = 2497.89038 Valid_Acc = 0.99723 Valid_Cost =  80219.64062
Epoch: 650 Acc = 0.99770 Cost = 2533.55322 Valid_Acc = 0.99691 Valid_Cost =  72468.42969
Epoch: 660 Acc = 0.99915 Cost = 1413.71790 Valid_Acc = 0.99838 Valid_Cost =  105465.48438
Epoch: 670 Acc = 0.99909 Cost = 1355.28833 Valid_Acc = 0.99824 Valid_Cost =  111732.75781
Epoch: 680 Acc = 0.99945 Cost = 991.83801 Valid_Acc = 0.99856 Valid_Cost =  122508.90625

Optimization Finished!
```

Plot of accuracy and cost summaries

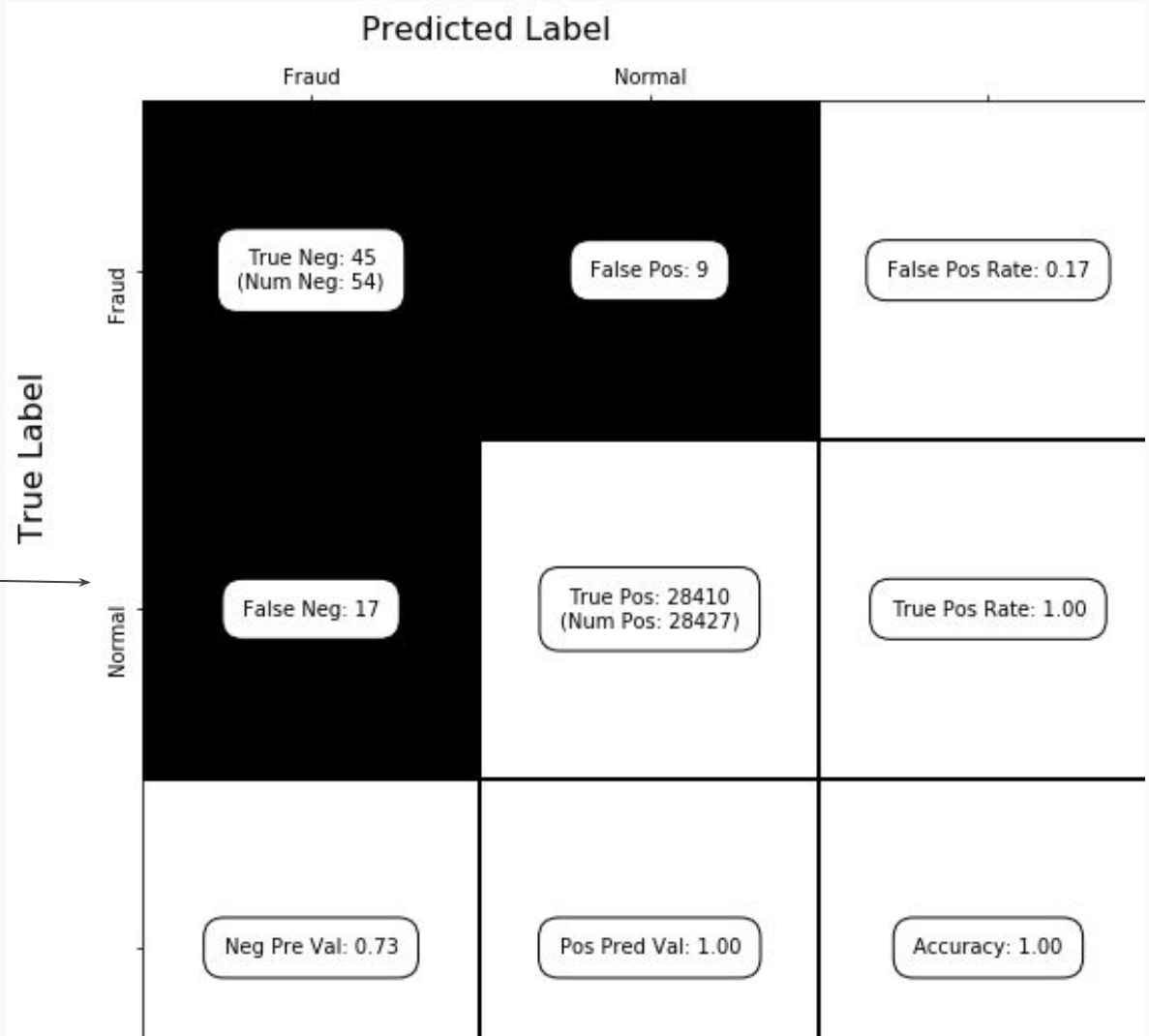


Evaluation

Accuracy on validation(test) set = 0.998911

F1 Score - see chart

```
Accuracy = 0.998911
```



References

References

- Chandola, Varun, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." ACM computing surveys (CSUR) 41.3 (2009)
- Dal Pozzolo, Andrea, and Gianluca Bontempi. "Adaptive machine learning for credit card fraud detection." (2015) (*Ph.D Thesis*)
- Diederik P. Kingma, Jimmy Ba "Adam: A Method for Stochastic Optimization" arXiv:1412.6980 [cs.LG] (2017)

References

Matplotlib: Python plotting — Matplotlib 2.1.0 documentation. (n.d.). Retrieved from https://matplotlib.org/

NumPy — NumPy. (n.d.). Retrieved from http://www.numpy.org/

Python Data Analysis Library — pandas: Python Data Analysis Library. (n.d.). Retrieved from http://pandas.pydata.org/

TensorFlow. (n.d.). Retrieved from https://www.tensorflow.org/

Thanks