Credit Card Fraud Detection

HARSH BANSAL

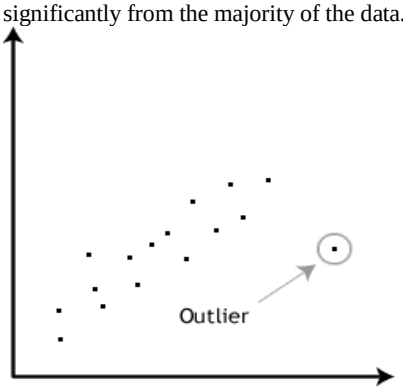
MACHINE LEARNING INTERN

AI TECHNOLOGY AND SYSYTEMS

bansalh944@gmail.com

www.ai-techsystems.com

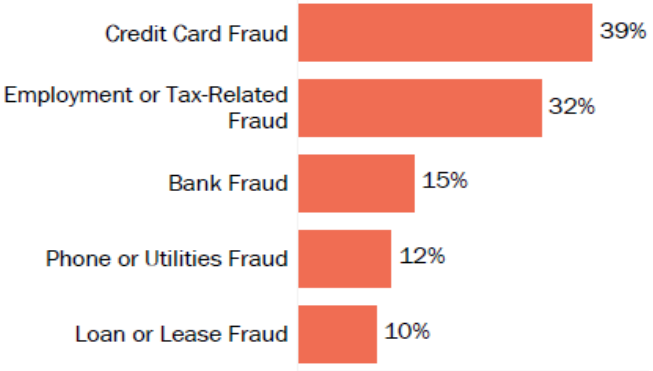
***Abstract*— In data mining, anomaly detection is referred to the identification of items or events that do not conform to an expected pattern or to other items present in a dataset. Typically, these anomalous items have the potential of getting translated into some kind of problems such as structural defects, errors or frauds. Using machine learning for anomaly detection helps in enhancing the speed of detection. Anomaly detection can be a key for solving intrusions, as while detecting anomalies, perturbations of normal behavior indicate a presence of intended or unintended induced attacks, defects, faults, and so on. Implementing machine learning algorithms will provide companies with a simple yet effective approach for detecting and classifying these anomalies. Machine learning algorithms have the ability to learn from data and make predictions based on that data. Machine learning for anomaly detection includes techniques that provide a promising alternative for detection and classification of anomalies based on an initially large set of features. Anomaly detection (or outlier detection) is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data.**



Keywords—Credit Card, Neural Network, Anomaly Detection

I. INTRODUCTION

Credit card fraud is a wide-ranging term for theft and fraud committed using or involving a payment card, such as a credit card or 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. According to the United States Federal Trade Commission, while the rate of identity theft had been holding steady during the mid-2000s, it increased by 21 percent in 2008. However, credit card fraud, that crime which most people associate with ID theft, decreased as a percentage of all ID theft complaints for the sixth year in a row. Although incidences of credit card fraud are limited to about 0.1% of all card transactions, they have resulted in huge financial losses as the fraudulent transactions have been large value transactions. In 1999, out of 12 billion transactions made annually, approximately 10 million—or one out of every 1200 transactions—turned out to be fraudulent. Also, 0.04% (4 out of every 10,000) of all monthly active accounts were fraudulent. Even with tremendous volume and value increase in credit card transactions since then, these proportions have stayed the same or have decreased due to sophisticated fraud detection and prevention systems. Today's fraud detection systems are designed to prevent one-twelfth of one percent of all transactions processed which still translates into billions of dollars in losses.



II. SOFTWARE REQUIREMENTS AND SPECIFICATION

A. *PRODUCT PERSPECTIVE*

The purpose of this project work is to provide a proper supervised and unsupervised methodology and efficient algorithm for the purpose of same. So, we aim at developing an algorithm which will prove to be more and more fruitful for this purpose and will be of immense applicability in detecting credit card frauds. The employment of newer concepts and algorithms will prove to be a much better headway in field.

B. *USER CHARACTERSRICS*

The users who are developing the whole application or are trying to make an evaluation of proper working of newest ideas and the algorithms implemented need to be proficient in the following areas:

- Python programming with libraries Pandas and NumPy
- Visualization libraries like matplotlib and seaborn
- Machine Learning with libraries sklearn, Isolation Forest, One Class SVM, Local Outlier Factor.
- Deep Learning with libraries with Keras, Minisom

C. *SPECIFIC REQUIREMENTS*

The whole project will be developed using the Jupyter Notebook which uses the Python language for the purpose of development of project. So, the specific requirement for this project work are:

- Anaconda including Jupyter
- Python 3
- Window 7/8.1/10 or Linux or Mac

III. PROJECT PLAN

The major aspect of this project to develop a best suited algorithm to find the outliers or frauds in case of credit cards. We will implement

several machine learning and deep learning algorithms and compare them and choose the best algorithm.

We will implement algorithms like:

- Neural Network
- Isolation Forest
- OneClassSVM
- Self-Organizing Maps
- Local Outlier Factor
- DBSCAN

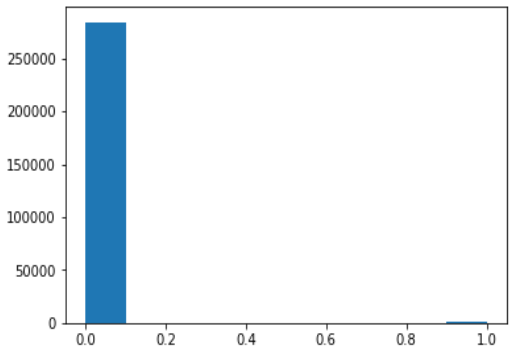
For this purpose, we have been provided a dataset. The datasets contain transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the 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, we 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-dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.
For Dataset click here
Now we will study each algorithm in detail.

A. *Artificial Neural Network*

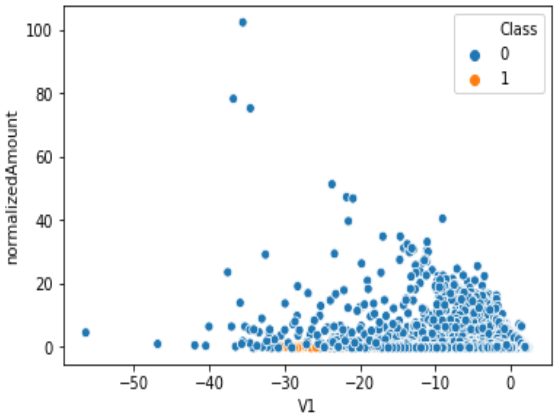
ANN are concept of deep learning which are implement using keras (in this case). ANN are composed of neurons. First layer or Input layer is the input neuron which consist the transaction and amount of each customer. The hidden layer consists of weights, bias and activation function. We can add as much hidden layer for tuning the performance. In this case we are using 3 layers. The output layer is the final layer where we get the classified output. The output either be 1 or 0 where 1 indicate fraud case and 0 indicate normal.

- Libraries Used for Data Preprocessing: Pandas, NumPy
- Libraries Used for Visualization: Matplotlib, Seaborn
- Libraries Used for Neural Network: Keras
- Libraries Used for Model Evaluation: Confusion Matrix, Classification Report
- Activation Function used in hidden layers: Rectifier
- Activation Function used in output layer: Sigmoid

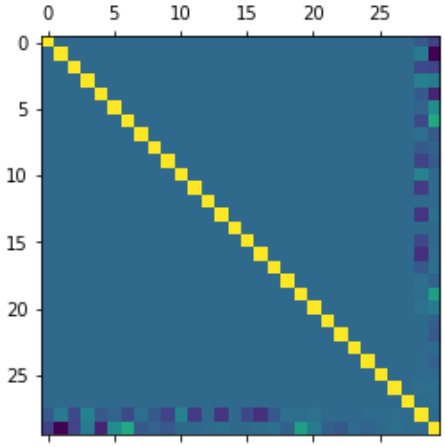
1) *Plots:*



X=Classes Y=No. of Classes



X=Transaction (can be any case) Y=Amount



Correlation Matrix

2) *Code Analysis:*

- No. of lines of code: **80**
- Source Code Memory: **108 kb**
- Time taken: **6:16:46 (6 minute 16 seconds 46 nanoseconds) (May vary in different system)**
- Model Summary:

Layer	(type)	Output Shape	Param #
dense_1	(Dense)	(None, 16)	480
dense_2	(Dense)	(None, 24)	408
dropout_1	(Dropout)	(None, 24)	0
dense_3	(Dense)	(None, 20)	500
dense_4	(Dense)	(None, 24)	504
dense_5	(Dense)	(None, 1)	25

Total params: 1,917
Trainable params: 1,917
Non-trainable params: 0

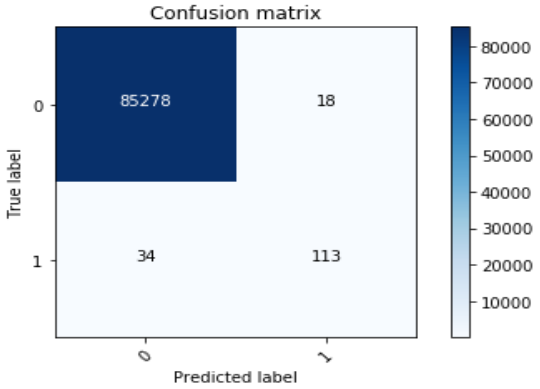
- Model Compilation:

```
Epoch 1/5
199364/199364 [==============================] - 51s 258us/step - loss:
0.0106 - acc: 0.9980
Epoch 2/5
199364/199364 [==============================] - 48s 243us/step - loss:
0.0040 - acc: 0.9993
Epoch 3/5
199364/199364 [==============================] - 58s 289us/step - loss:
0.0038 - acc: 0.9993
Epoch 4/5
199364/199364 [==============================] - 54s 273us/step - loss:
0.0034 - acc: 0.9994
Epoch 5/5
199364/199364 [==============================] - 46s 233us/step - loss:
0.0032 - acc: 0.9994
```

*3) **Model Evaluation:***

Libraries Used: **Confusion Matrix and Classification Report**

Confusion Matrix Results:



True Positive=85278
True Negative=113
False Positive=18
False Negative=34

Classification Report:

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	85312
1.0	0.77	0.86	0.81	131
micro avg	1.00	1.00	1.00	85443
macro avg	0.88	0.93	0.91	85443
weighted avg	1.00	1.00	1.00	85443

Final accuracy achieved: **99% (0.9994)**

*4) **Source Code:***

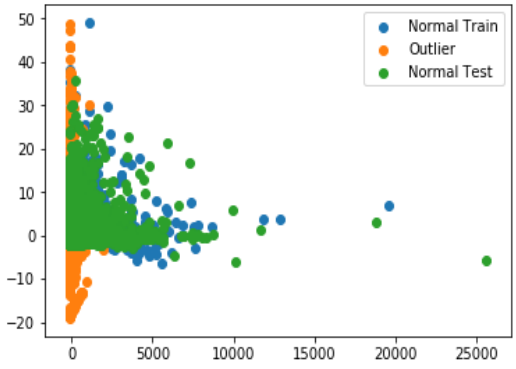
For ipynb file, click here

Place it in your home directory of anaconda and run the file using Jupyter Notebook

B. *Anomaly Detection Algorithms*

Anomaly detection is a technique used to identify unusual patterns that do not conform to expected behavior, called outliers. It has many applications in business, from intrusion detection (identifying strange patterns in network traffic that could signal a hack) to system health monitoring (spotting a malignant tumor in an MRI scan), and from fraud detection in credit card transactions to fault detection in operating environments.

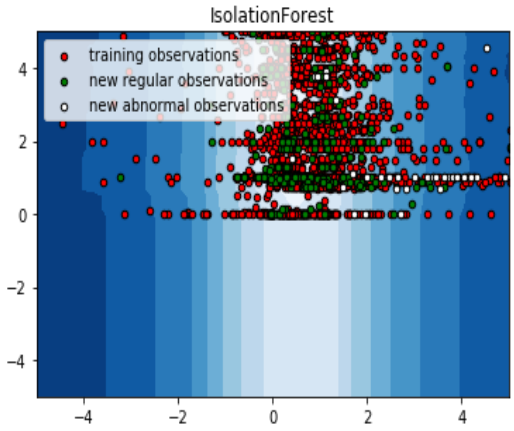
For Anomaly Detection algorithms, we will divide the dataset into 3 parts- Normal outliers, Normal Train, Normal Test



*a) **Isolation Forest***: Isolation forest detects anomalies by randomly partitioning the domain space. Yeah, you're heard me right- It works similar to Decision trees algorithm, where we start with a root node and keep on partitioning the space. In Isolation forest we partition randomly, unlike Decision trees where the partition is based on gain. Partitions are created by randomly selecting a feature and then randomly creating a split value between the maximum and the minimum value of the feature. We keep on creating the partitions until we isolate all the points (in most cases we also set a limit on number of partitions/heights of the tree).

- Libraries Used for Data Preprocessing: **Pandas, NumPy**
- Libraries Used for Visualization: **Matplotlib, Seaborn**
- Libraries Used for Implementing Algorithm: **sklearn.ensemble**
- Hyperparameters Used And their Values: **behaviour='old', bootstrap=False, contamination='legacy',**
- **max_features=1.0, max_samples=100, n_estim ators =100, n_jobs=None, random_state=None, verbose=0**

1) Plots after training model:



*2) **Code Analysis:***

- No. of lines of code: **55**
- Source Code Memory: **70 kb**
- Time taken: **1:10:06 (1 minute 10 seconds 6 nanoseconds) (May vary in different system)**

*3) **Model Accuracy:***

Accuracy test: **0.8554597215175602 (85%)**
Accuracy outliners: **0.9044715447154471 (90%)**

*4) **Source Code:***

For ipynb file, click here

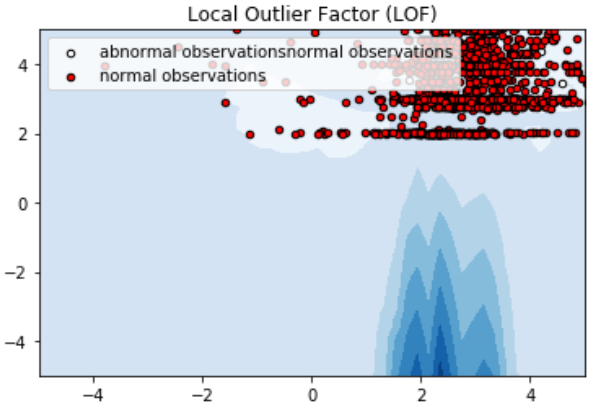
Place it in your home directory of anaconda and run the file using Jupyter Notebook

b) Local Outlier Factor: The Local Outlier Factor (LOF) algorithm is an unsupervised anomaly detection method which computes the local density deviation of a given data point with respect to its neighbors. It considers as outliers the samples that have a substantially lower density than their neighbors. This example shows how to use LOF for outlier detection which is the default use case of this estimator in scikit-learn. Note that when LOF is used for outlier detection it has no predict, decision function and score samples methods. The number of neighbors considered (parameter n_neighbors) is typically set 1) greater than the minimum number of samples a cluster has to contain, so that other samples can be local outliers relative to this cluster, and 2) smaller than the maximum number of close by samples that can potentially be local outliers. In practice, such information are generally not available, and taking n_neighbors=20 appears to work well in general.

- Libraries Used for Data Preprocessing: **Pandas, NumPy**
- Libraries Used for Visualization: **Matplotlib, Seaborn**
- Libraries Used for Implementing Algorithm: **sklearn.neighb ors**

- Hyperparameters Used And their Values**: algorithm='au to ', contamination='legacy', leaf_size=30, metric='minkow ski', metric_params=None, n_jobs=None, n_neighbors= 20, novelty=False, p=2**

*1)**Plot after Training the model:***



*2) **Code Analysis**:*

- No. of lines of code: **50**
- Source Code Memory: **70 kb**
- Time taken: **35:10:06 (35 minute 10 seconds 6 nanoseconds) (May vary in different system)**

*3) **Model Accuracy:***

Number of error cases:**5669**

Accuracy: **0.8998545127406777(90% approx.)**

*4) **Source Code**:*

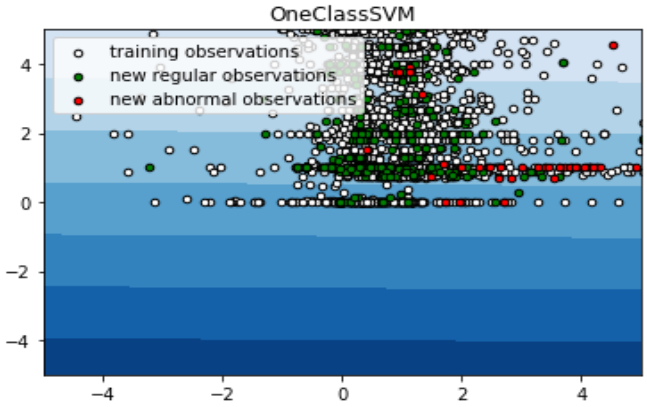
For ipynb file, click here

Place it in your home directory of anaconda and run the file using Jupyter Notebook

c) OneClassSVM: A One-Class Support Vector Machine is an unsup ervised learning algorithm that is trained only on the 'normal' data, in our case the negative examples. It considers as outliers the samples t hat have a substantially lower density than their neighbors. In one-cla ss learning we train the model only on the positive class data-set and t ake judgments from it on the universe [A union ~A] spontaneously. It 's a hot research topic and there are multiple tools available, like One -class SVM and Isolation Forest, to achieve this task. One-class learni ng can prove to be vital in scenarios where data consisting of ~A sam ples can take up any distribution and it isn't possible to learn a pattern for ~A class.

- Libraries Used for Data Preprocessing: **Pandas, NumPy**
- Libraries Used for Visualization: **Matplotlib, Seaborn**
- Libraries Used for Implementing Algorithm: **sklearn.SVC**

*1) **Plot after training model:***



*2) **Code Analysis:***

- No. of lines of code: **55**
- Source Code Memory: **70 kb**
- Time taken: **30:15:26 (30 minute 15 seconds 26 nanoseconds) (May vary in different system)**

*3) **Model Accuracy:***

Accuracy test: **0.04693753875641243**

Accuracy outliners: **0.9126016260162602**

*4) **Source Code:***

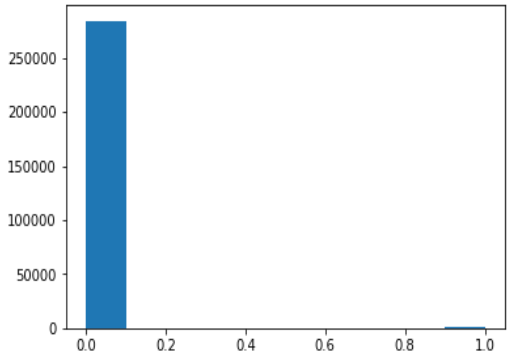
For ipynb file, click here

Place it in your home directory of anaconda and run the file using Jupyter Notebook

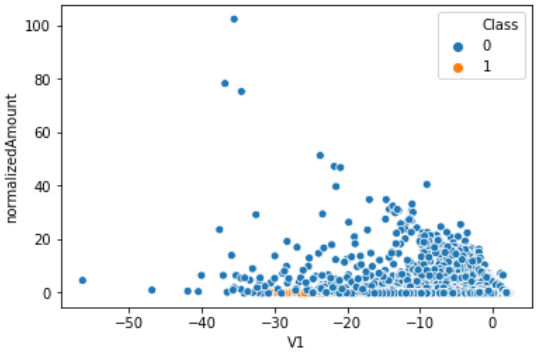
C. DBSCAN: Density-Based Spatial Clustering of Applications with Noise. Another way to find the outliers is DBSCAN which uses clustering. In this method, we calculate the distance between points (the Euclidean distance or some other distance) and look for points which are far away from others. A natural way to group together hosts that are behaving similarly is to use a clustering algorithm. We use DBSCAN, a popular density-based clustering algorithm, for this purpose. DBSCAN works by greedily agglomerating points that are close to each other. Clusters with few points in them are considered outliers. We use a simplified form of DBSCAN to detect outliers on time series. We consider each host to be a point in d-dimensions, where d is the number of elements in the time series. Any point can agglomerate, and any point that is not in the largest cluster will be considered an outlier.

- Libraries Used for Data Preprocessing: **Pandas, NumPy**
- Libraries Used for Visualization: **Matplotlib, Seaborn**
- Libraries Used for Implementing Algorithm: **DBSCAN**
- Hyper Parameters: **algorithm='auto', eps=0.5, leaf_size=3 0, metric='euclidean', metric_params=None, min_sampl es=10, n_jobs= None, p=None**

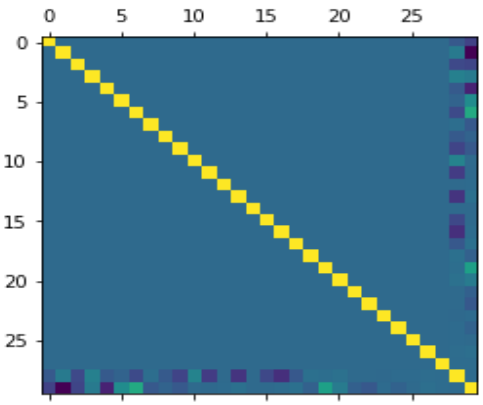
*1) **Plots:***



X=Classes Y=No. of Classes



X=Transaction (can be any case) Y=Amount



Correlation Matrix

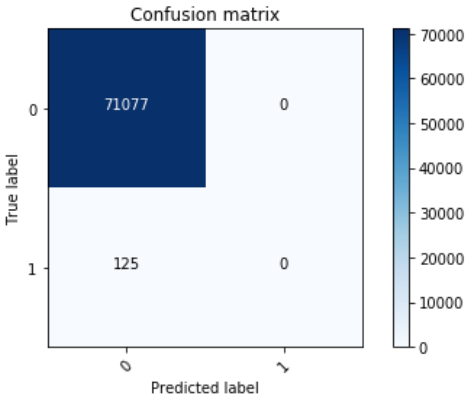
*2) **Code Analysis:***

- No. of lines of code: **60**
- Source Code Memory: **28 kb**
- Time taken: **1:16:46 (6 minute 16 seconds 46 nanoseconds) (May vary in different system)**

*3) **Model Evaluation:***

Libraries Used: **Confusion Matrix and Classification Report**

Confusion Matrix Results:



True Positive=71077
True Negative=0
False Positive=125
False Negative=0

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	71077
1	0.00	0.00	0.00	125
micro avg	1.00	1.00	1.00	71202
macro avg	0.50	0.50	0.50	71202
weighted avg	1.00	1.00	1.00	71202

Final accuracy achieved:99%

*4) **Source Code:***

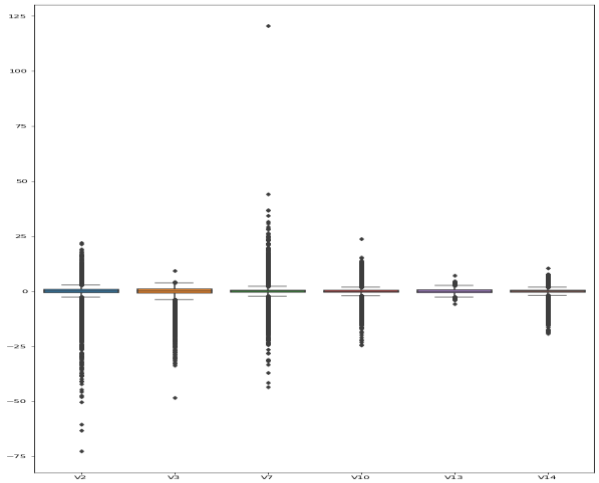
For ipynb file, click here

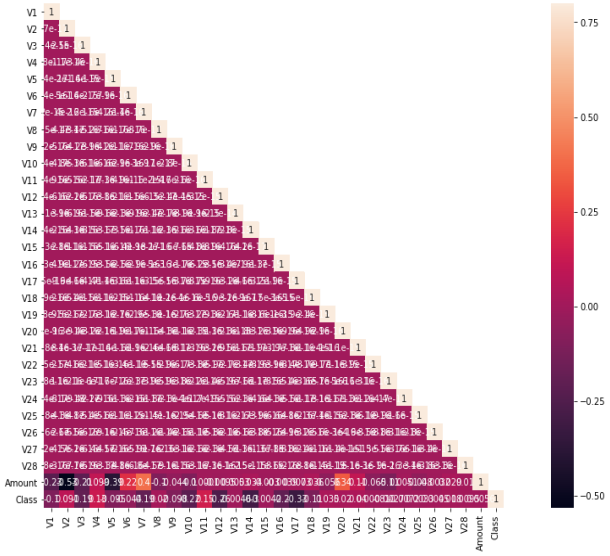
Place it in your home directory of anaconda and run the file using Jupyter Notebook

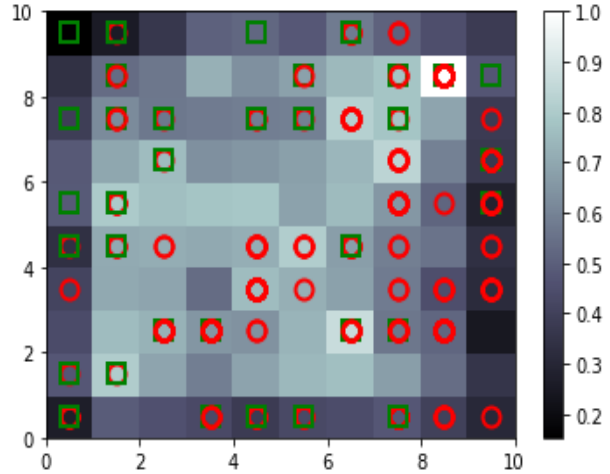
D. Self-Organizing Maps: A **self-organizing map** (**SOM**) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a **map**, and is therefore a method to do dimensionality reduction. Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning (such as backpropagation with gradient descent), and in the sense that they use a neighborhood function to preserve the topological properties of the input space. Each data point in the data set recognizes themselves by competing for representation. SOM mapping steps starts from initializing the weight vectors.

- Libraries Used for Data Preprocessing: **Pandas, NumPy**
- Libraries Used for Visualization: **Matplotlib, Seaborn**
- Libraries Used for Implementing Algorithm: **MINISOM**

*1) **Plots:***







*2) **Code Analysis:***

- No. of lines of code: **60**
- Source Code Memory: **28 kb**

- Time taken: **41:20:40 (41 minute 20 seconds 4 nanoseconds) (May vary in different system)**

*3) **Source Code:***

For ipynb file, click here

Place it in your home directory of anaconda and run the file using Jupyter Notebook

V. CONCLUSION

Several algorithms have been implemented on same data set to detect the frauds in case of credit cards. All the algorithms have been analyzed and compared on basis of accuracy they are giving on same data. We implemented different type of algorithms which include neural network from deep learning, anomaly detection algorithms like isolation forest, OneClassSVM, Local Outlier Factor, supervised algorithm like DBSCAN and unsupervised algorithm like Self-Organizing Maps. This was done to attain the best approach for the purpose. Upon analyzing we conclude that 3-Layer Neural Network have been the best algorithms for the purpose of credit card fraud detection as it provides best accuracy with best precision and recall on both cases. It was also decent in case of time complexity and space complexity but not the best compared to isolation forest and DBSCAN. But accuracy is best of all. It gives 99% accuracy which close to perfect. DBSCAN algorithm is also providing the 99% accuracy but its precision and recall are zero for fraud case which is not acceptable. In future this type of algorithm can be used in different cases. For better performance we can change the layers properties for better results. This also proves the importance of deep learning in field of artificial intelligence.

ACKNOWLEDGMENT

The dataset has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. More details on current and past projects on related topics are available on https://www.researchgate.net/project/Fraud-detection-5 and the page of the DefeatFraud project Please cite the following works: 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 Dal Pozzolo, Andrea; Caelen, Olivier; Le Borgne, Yann-Ael; Waterschoot, Serge; Bontempi, Gianluca. Learned lessons in credit card fraud detection from a practitioner perspective, Expert systems with applications,41,10,4915-4928,2014, Pergamon Dal Pozzolo, Andrea; Boracchi, Giacomo; Caelen, Olivier; Alippi, Cesare; Bontempi, Gianluca. Credit card fraud detection: a realistic modeling and a novel learning strategy, IEEE transactions on neural networks and learning systems,29,8,3784-3797,2018,IEEE Dal Pozzolo, Andrea Adaptive Machine learning for credit card fraud detection ULB MLG PhD thesis (supervised by G. Bontempi) Carcillo, Fabrizio; Dal Pozzolo, Andrea; Le Borgne, Yann-Aël; Caelen, Olivier; Mazzer, Yannis; Bontempi, Gianluca. Scarff: a scalable framework for streaming credit card fraud detection with Spark, Information fusion,41, 182-194,2018,Elsevier Carcillo, Fabrizio; Le Borgne, Yann-Aël; Caelen, Olivier; Bontempi, Gianluca. Streaming active learning strategies for real-life credit card fraud detection: assessment and visualization, International Journal of Data Science and Analytics, 5,4,285-300,2018,Springer International Publishing Bertrand Lebichot, Yann-Aël

REFERENCES

[1] http://mlg.ulb.ac.be

[2] https://www.researchgate.net/project/Fraud-detection-5

[3] https://www.datadoghq.com/blog/outlier-detection-algorithms-at-datadog/#dbscan

[4] https://towardsdatascience.com/self-organizing-maps-ff5853a118d4

[5] https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e

[6] https://towardsdatascience.com/local-outlier-factor-for-anomaly-detection-cc0c770d2ebe

[7] https://carldawson.net/outlier-detection-svm/

https://www.kaggle.com/mlg-ulb/creditcardfraud