Credit Card Fraud Detection – Learning unbalanced data

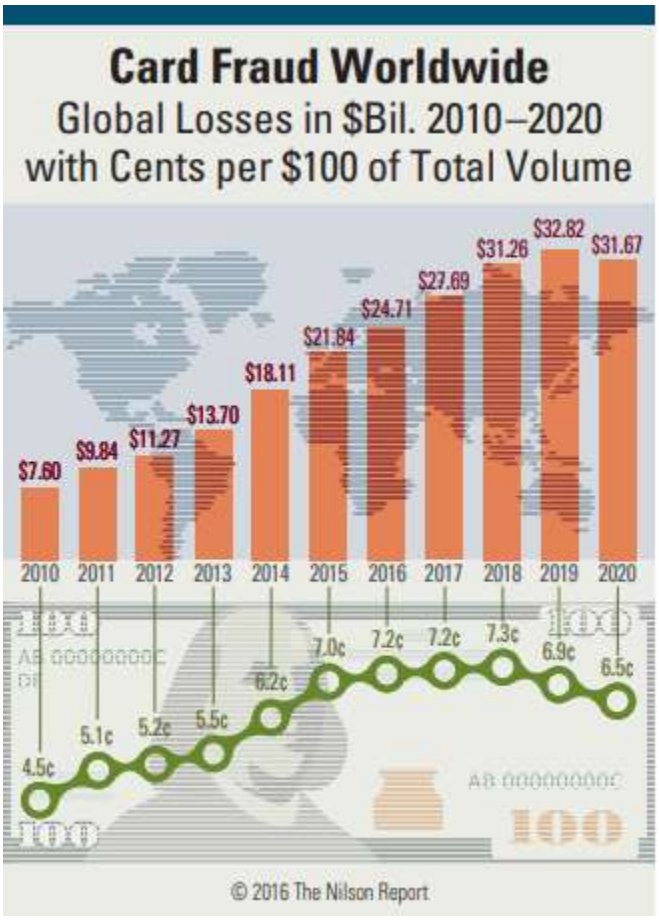
By

Pinaki Bhagat | Sydney Correa
CSCIE82 Fall 2018
Prof. Peter Henstock

TABLE OF CONTENTS

ABSTRACT

Credit card fraud detection and timely prevention is much more relevant in this digital economy. Annual global fraud losses reached $21.8 billion in 2015. About every 12 cents per $100 were stolen in the US during the same year.



Using machine learning to detect financial fraud dates back to the early 1990s and has advanced over the years. Generally speaking, credit card frauds detection approaches can be classified in two main families, which correspond to the two families of machine learning algorithms: supervised and unsupervised ones. In the former family, past transactions are labeled as legal or illegal, for instance, based on expert judgement or customer's claims; the algorithms then learn over these data, to create a model that is applied to new instances appearing in the system. On the other hand, unsupervised techniques are based on the automatic detection of patterns that are considered "normal" for a given user, for then detecting transactions that are not coherent with such patterns, as illegal users are expected to depart from the owner's behaviors.

Both families have their own advantages and disadvantages, and in general supervised approaches are more effective in detecting illegal transactions, although they require a large initial training set (which in some cases may not be available).

There can be many approaches to tackle this problem including traditional machine learning techniques such as logistic regression, k-means, random forest. There are also advanced deep learning techniques such as CNN and autoencoding that can be used to improve the accuracy of prediction.

Our intent would be to try these different approaches on the same dataset and compare their performance against each other. As this problem represents a typical class imbalance issue so, we will try employing various balancing techniques and evaluate the relative performance of these algorithms.

Traditional methods

a) KNN
b) Ensemble learning
    a. Random Forest
    b. ExtraTrees
    c. AdaBoost
    d. GradientBoost
    e. XGBoost

Deep Learning

a) Autoencoders

Class Imbalance

a) SMOTE

Dataset

The datasets 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.

Due to confidentiality of data, original features are masked and the dataset contains only numerical input variables which are the result of a PCA transformation. Features “Time” and “Amount” are not transformed and indicates the seconds elapsed between each transaction and the first transaction in the dataset and transaction amount respectively.

Enterprise Software

Recently quite a few enterprise software companies have come up which packages the open source machine learning libraries to create a product suitable for enterprise ready machine learning. One such company is H2O.ai. We would like to try this software package to figure out the best algorithm the software recommends given the dataset. This will help establish credibility of the tool as well as our familiarity with enterprise softwares.

We also have tried to perform this complete analysis in Amazon AWS to test out the benefits of using cloud platform for machine learning.

TRADITIONAL TECHNIQUES

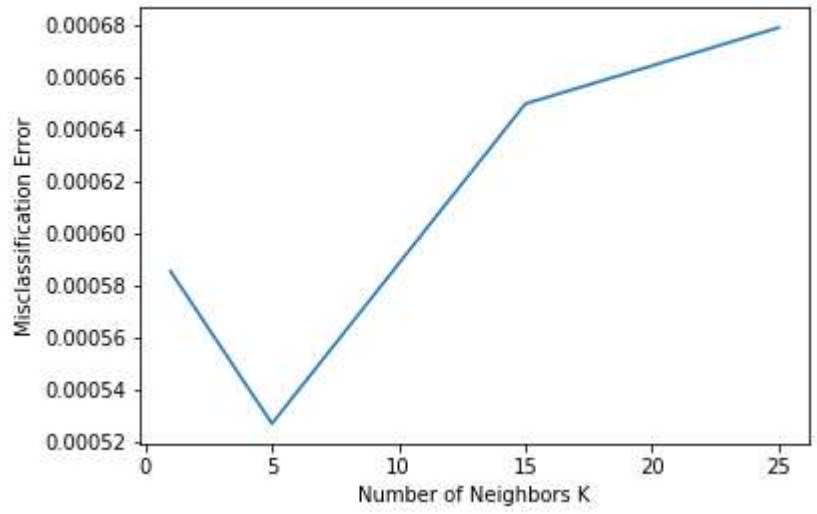
In this section we will discuss the results for each of the traditional methods employed to predict the fraudulent transaction.

The dataset presents unique problem of class imbalance. All traditional and advanced algorithms will predict with a very high accuracy using a confusion matrix. However in this case we would recommend to measure accuracy using the Area Under the Precision-Recall Curve (AUPRC).

K-NEAREST NEIGHBOURS

Dataset was split between Train and Test sets. Hyper parameter tuning for k was performed with a range of [1,5,15,25] and value of k=5 was chosen as the best parameter for executing KNN.

```
The optimal number of neighbors is 5
```



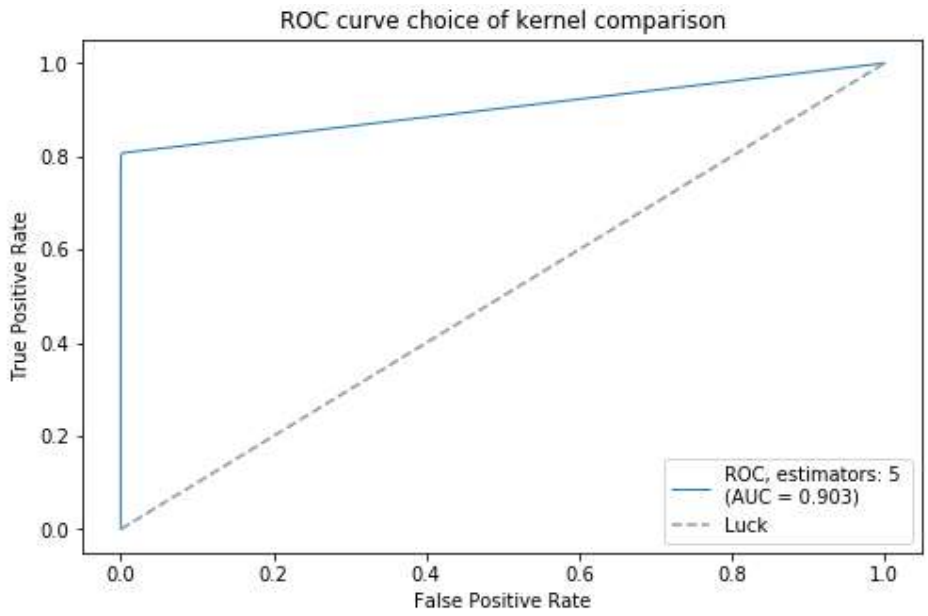
The trained KNN classifier was then used to predict on the test dataset to obtain accuracy and other key metrics.

```
0.999420666409185
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     56863
          1       0.90      0.75      0.82        99

avg / total       1.00      1.00      1.00     56962
```

The recall and f1-score are pretty high and in general the KNN classifier worked pretty well. This is also evident from the high Area under curve.



ENSEMBLE LEARNING

Dataset was split into train and test and following algorithms were used as learners on the train data:

a. Random Forest
b. Extra Trees
c. AdaBoost
d. GradientBoost

Parameters used for these learner algorithms are as below:

N_estimators = 16
Max_depth = 10
N_jobs = -1
Learning rate = 0.75
min_samples_leaf = 2

Individual predictions of the algorithms based on test data are below:

	AdaBoost	ExtraTrees	GradientBoost	RandomForest
0	0.278610	0.000171	0.000346	0.000128
1	0.042268	0.008257	0.000353	0.000136
2	0.280778	0.000221	0.000360	0.000144
3	0.291032	0.000237	0.000376	0.000304
4	0.276816	0.000298	0.000363	0.000098

A model correlation was performed to figure out which modeling techniques are highly correlated and doesn't require to be fed into the final aggregator algorithm. Correlation results are below:

	AdaBoost	ExtraTrees	GradientBoost	RandomForest
AdaBoost	1.000000	0.123530	0.107517	0.124163
ExtraTrees	0.123530	1.000000	0.792294	0.963315
GradientBoost	0.107517	0.792294	1.000000	0.823572
RandomForest	0.124163	0.963315	0.823572	1.000000

ExtraTrees alogorithm was dropped from the stacking model as it was highly correlated with GradientBoost and RandomForest.

XGBoost was used as the final learner accuracy and other key metrics are below:

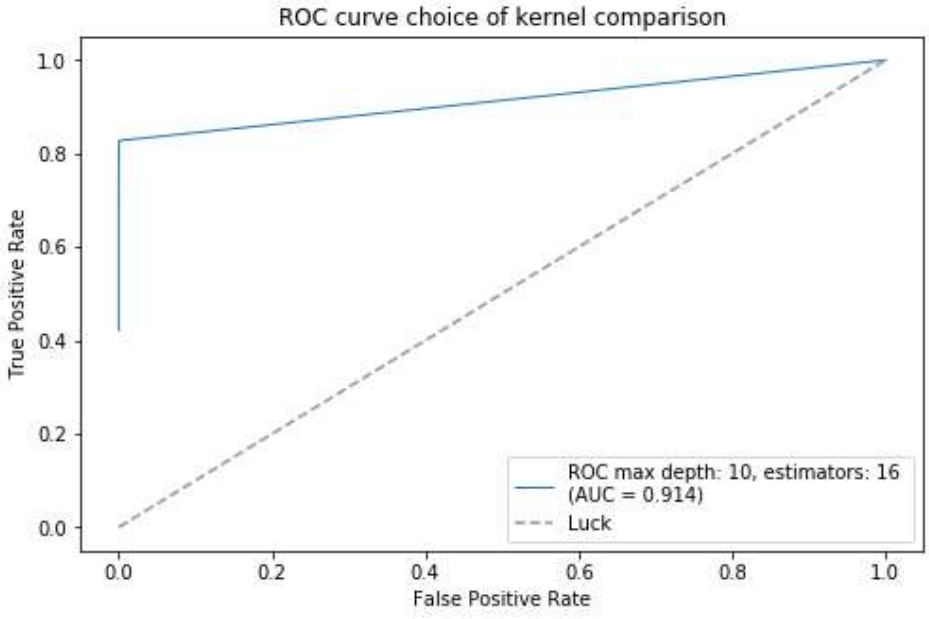
```
Accuracy score:  99.95%

Classification report:
             precision    recall  f1-score   support

   Output 0       1.00      1.00      1.00    113726
   Output 1       0.93      0.76      0.83       197

avg / total       1.00      1.00      1.00    113923
```

The f1-score as well as recall came out a bit better than individual KNN model. This is also evident from the higher Area under curve.



DEEP LEARNING

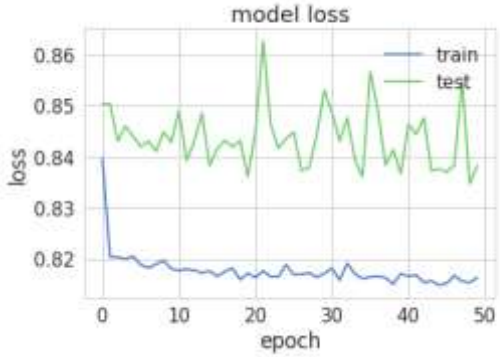
Deep learning tries to look at the problem of credit card fraud and anomaly detection by training the network on non-fraudulent transactions, resulting in a low MSE. This causes fraudulent transactions to be detected by flagging them based on higher than normal MSE values.

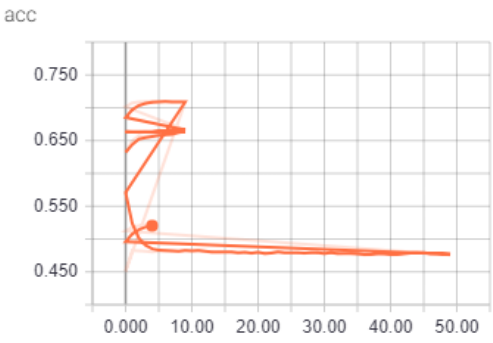
AUTOENCODER

Data is split into train and test sets, after which the train set would filter for only Class= 0 (non-Fraudulent transactions). Autoencoder algorithm will be used to reduce the reconstruction error.

Reconstruction error for Autoencoder: $L(x,x') = ||x- x'||^2$

The Autoencoder is setup using 4 fully connected layers with 14,7,7 and 29 neurons respectively. First two are encoders and last 2 are decoders. Activation filters used as “tanh”, “relu”.





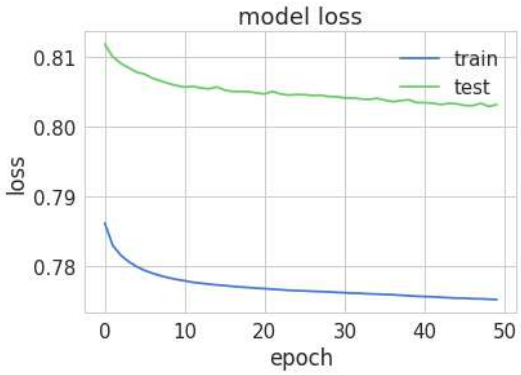
Model is trained with ephocs = 50 with a sample batch size = 32 using Adam optimizer with learning rate = 0.01

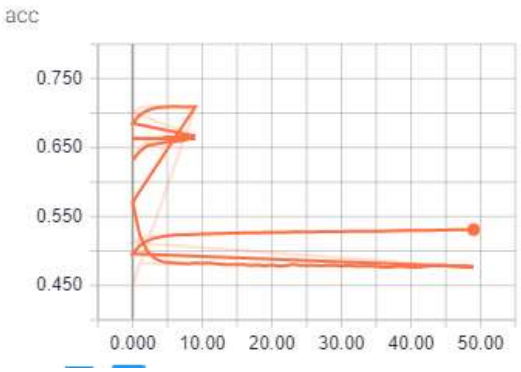
- Train loss ranged from 0.84 to 0.8163
- Validation loss ranged from 0.85 to 0.832
- Accuracy improved from 0.459 to 0.4761

Model tuning -

Model was tuned to increase sample batch size = 16 and higher learning rate =0.0001

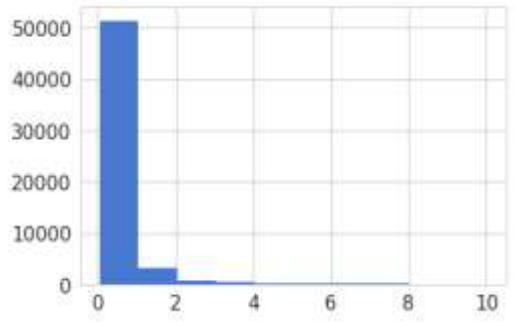
- Train loss dropped from 0.8163 to 0.7752
- Validation loss dropped from 0.832 to 0.8032
- Accuracy improved from 0.4761 to 0.5310

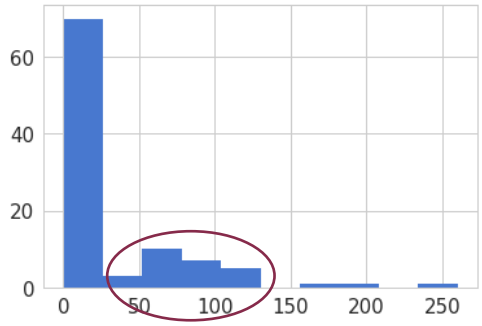




Evaluation –

Looking at the error plots without fraudulent transactions and with fraudulent transactions clearly indicates higher error rates (shown in fig below on right) in the later, implying anomaly detection, in our case being fraudulent transactions.

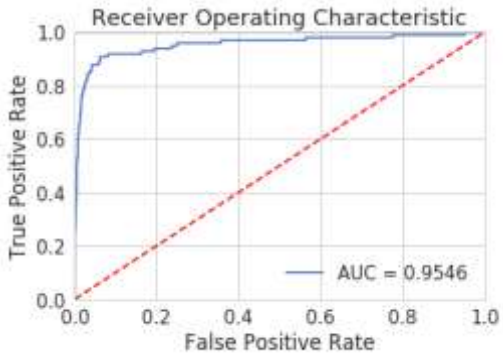


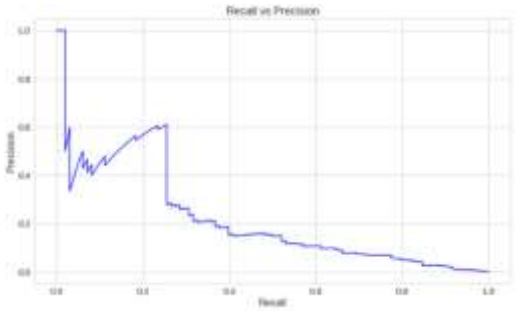


Performance Metrics

ROC curve: AUC = 0.9546 indicates model has good precision rate

Recall vs Precision: indicates both high precision would suggest low false positive rates and high recall would indicate low false positive rates.



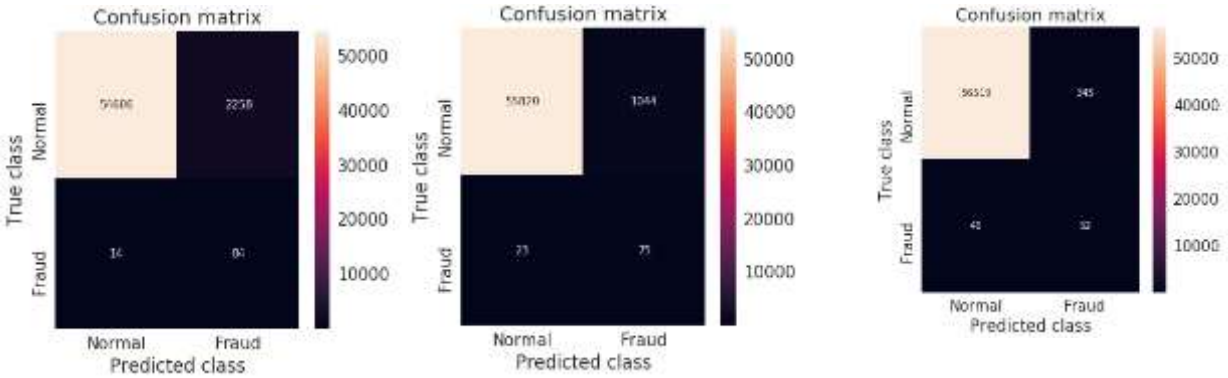


Prediction

We played around with different threshold values for prediction of fraud vs non fraud transactions. We found this to be a subjective threshold value which can vary the values of the confusion matrix

Error: y_pred = [1 if e > threshold else 0 for e in error_df.reconstruction_error.values]

Below are 3 examples of varying confusion matrices for 3 different threshold values (2, 5, 10)



H2O - AUTOML

Recently quite a few enterprise software companies have come up which packages the open source machine learning libraries to create a product suitable for enterprise ready machine learning. One such company is H2O.ai. We tried this software package to figure out the best algorithm the software recommends given the dataset

H2O.AI

We explored using the popular H2O.ai platform, specifically the AutoML package. The package runs the dataset through various ML models including ensembles to train and test. The output is then shown on a leaderboard scoring different model performances.

The package is available to run as a service on Amazon AWS Market place with a lot of bells and whistles but we went with pip installing H2O on our AWS instance and then running it on a standard jupyter notebook.

The AutoML options include the following packages – Random Forest (DRF), an Extremely Randomized Forest (XRT), three pre-specified XGBoost GBM (Gradient Boosting Machine) models, five pre-specified H2O GBMs, a near-default Deep Neural Net, a random grid of XGBoost GBMs, a random grid of H2O GBMs, a random grid of Deep Neural Nets (source: H2O.ai documentation)

Training using AutoML

The first step is to initiate the h2o instance including the AutoML package, which internally runs on a JVM on Ubuntu.

Running the AutoML package through jupyter is quite intuitive. Plugging in the data set is quite effortless. The package automatically parses the dataset. Preparing the credit card dataset required dropping unwanted columns, creating the Class variable as a factor and splitting into train and test sets.

The AutoML package requires 2 inputs which are max # of models and max run time. In our case we tuned with max # models = 5, 10, 15

To solve for the class imbalance we set the balance_classes and max_after_balance_size

Default parameters: run time = 1 hour and CV = 5 fold

```
aml = H2OAutoML(max_models=10,
                seed=1234,
                balance_classes = True,
                max_after_balance_size=0.8)
```

aml.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

Leader board with max_models=5 scores with no oversampling shows XGBoost as the leader:

model_id	auc	logloss	mean_per_class_error	rmse	mse
XGBoost_1_AutoML_20181219_234301	0.984103	0.00250716	0.101853	0.0200578	0.000402316
GLM_grid_1_AutoML_20181219_234301_model_1	0.973893	0.00411683	0.100894	0.0265999	0.000707553
XGBoost_2_AutoML_20181219_234301	0.971338	0.00324639	0.10298	0.0203073	0.000412386
DRF_1_AutoML_20181219_234301	0.956017	0.00393489	0.0894407	0.0207101	0.000428908
XRT_1_AutoML_20181219_234301	0.952861	0.00359792	0.10298	0.0206141	0.000424941
StackedEnsemble_BestOfFamily_AutoML_20181219_234301	0.94077	0.00311164	0.0950754	0.0207072	0.000428788
StackedEnsemble_AllModels_AutoML_20181219_234301	0.93763	0.00312438	0.09733	0.020708	0.000428822

Leader board with max_models = 5 scores with oversampling shows XGBoost as the leader:

model_id	auc	logloss	mean_per_class_error	rmse	mse
XGBoost_1_AutoML_20181220_015035	0.981113	0.00249092	0.080723	0.0198301	0.000393234
GLM_grid_1_AutoML_20181220_015035_model_1	0.978821	0.00398823	0.0999025	0.0261906	0.000685946
XGBoost_2_AutoML_20181220_015035	0.964032	0.00359919	0.0953707	0.0204597	0.000418601
DRF_1_AutoML_20181220_015035	0.952295	0.0108255	0.0924557	0.035489	0.00125947
StackedEnsemble_BestOfFamily_AutoML_20181220_015035	0.947073	0.00314415	0.101216	0.0208256	0.000433706
StackedEnsemble_AllModels_AutoML_20181220_015035	0.946946	0.00316611	0.102682	0.0209337	0.000438221
XRT_1_AutoML_20181220_015035	0.946124	0.0114528	0.096847	0.0356493	0.00127087

Leader board with max_models= 10 scores with oversampling shows 7 models

and XGBoost as the leader:

model_id	auc	logloss	mean_per_class_error	rmse	mse
XGBoost_1_AutoML_20181220_005816	0.981113	0.00249092	0.080723	0.0198301	0.000393234
GLM_grid_1_AutoML_20181220_005816_model_1	0.978821	0.00398823	0.0999025	0.0261906	0.000685946
XGBoost_2_AutoML_20181220_005816	0.972787	0.00289851	0.0895256	0.0205521	0.000422387
DRF_1_AutoML_20181220_005816	0.952372	0.0108481	0.0939194	0.0355181	0.00126154
XRT_1_AutoML_20181220_005816	0.947599	0.0113933	0.0968495	0.0359485	0.00129229
StackedEnsemble_BestOfFamily_AutoML_20181220_005816	0.947481	0.00314425	0.101216	0.0208257	0.000433709
StackedEnsemble_AllModels_AutoML_20181220_005816	0.946547	0.00316667	0.102682	0.0209589	0.000439276

Model performance metric of the XGBoost model show high AUC and pr_AUC

```
ModelMetricsBinomial: xgboost
** Reported on train data. **

MSE: 0.00018621979612176758
RMSE: 0.013646237434610596
LogLoss: 0.0009447178048844039
Mean Per-Class Error: 0.004578774041536837
AUC: 0.99980426010319
pr_auc: 0.94372678763981
Gini: 0.99960852020638
```

Confusion Matrix:

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.3811877965927124:

	0	1	Error	Rate
0	199240.0	4.0	0.0	(4.0/199244.0)
1	35.0	306.0	0.1026	(35.0/341.0)
Total	199275.0	310.0	0.0002	(39.0/199585.0)

Conclusion

H20. Currently does not have an inbuilt model for XGBoost and recommends calling it from Python using an API. We decided to abandon H2O at this point, as we have already implemented a version of XGBoost as part of our ensemble approach. Benefits of H2O that we felt are:

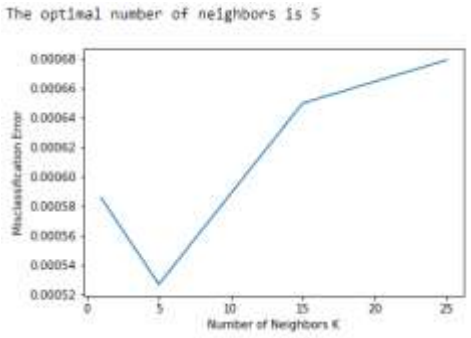
a. Easily available in AWS marketplace
b. Easy to install
c. Comes with a predefined algorithms and makes data preparation easy
d. Does only have support for basic algorithms. Anything advanced has to be done manually through Python APIs.

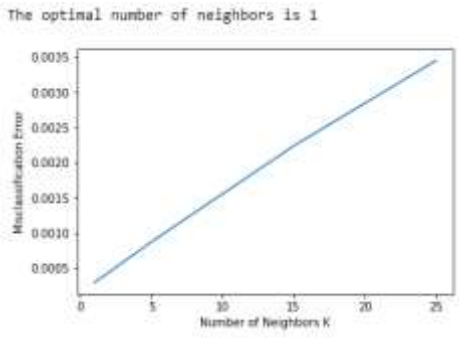
CLASS IMBALANCE USING SMOTE, UPSAMPLING AND DOWNSAMPLING

To address the class imbalance issue with the credit card dataset (total fraudulent transactions are 0.172%) we tried under sampling and oversampling the data using various approaches, including H2O and sklearn imblearn (SMOTE and SMOTEENN) and undersampling (RandomUndersampler).

However SMOTE was either not required or did not give the desired results. In the case of H2O, we were able to use the H2O inbuilt features for class imbalance. However both approaches (unbalanced and balanced) gave similar performance metrics.

In the case of KNN using oversampled data resulted in a degradation in model performance, and resulted in a KNN value of N= 1 (vs. N=5 for unbalanced data). We are assuming that synthetically inflating the number of fraudulent cases did not help with choosing the right number of neighbors.





In the case of Autoencoders, trying to balance the dataset would beat the fundamental idea of detecting anomalies by training the network on non-fraudulent class data and then using high reconstruction error to flag transactions.

LESSONS LEARNT

CHALLENGES & MITIGATION

Here are some of the challenges we ran into broadly classified into execution and technical categories

Execution

1. We found early on that the size of the dataset would require running on a faster machine, this forced us to tune all our models on the AWS cluster. We used a p2.xlarge cluster with 4 cpus which resulted in significant CPU utilization

2. We tried spending considerable amount of time trying to balance the class data, however the final models, we ended up with having better performance metrics, came from unbalanced data.

3. The lack of datasets for this use case (there are just 3-4 out in Kaggle) during our research lead us to narrowing down to the creditcard.csv which are based on the PCA outputs of the original dataset. This added a bit of uncertainty to how the models would perform.

Technical challenges

1. We ran into a firewall issue when trying to bring up our AWS cluster when at our office premises. This put some limitations in terms of how and when we could train our models

2. Splitting data into test, train and validate having just one dataset, required us to split the data into train (80%) and test(20%) and then split train into train (80% of 0.8) and validate (20% of 0.8)

3. We found a couple of depreciated python functions, which took up a good amount of time troubleshooting. For eg, we used the ensemble functions from our lectures, however we found that the kf.split button to not work and had to be changed to iloc.

FUTURE ENHANCEMENT

Both conventional as well as Deep Learning methods were performed to detect the fraudulent transactions. All these methods can only detect anomalies based on features present in the data. While data mining algorithms are able to detect hidden patterns in data, they usually lack the capacity of synthesizing metrics describing the global structure created by the interactions between the different features. It will fail to detect specific rings or network of fraudsters that exhibit certain types of behavior which can only be identified through network graph analysis.

NETWORK GRAPH ANALYSIS

Graph Embedding can be used to effectively increase the accuracy of individual credit card analysis tool. This can be done by instantiating nodes as individual customers and merchants with node properties about their financial histories. Edges would represent financial transactions between these entities with node properties like timestamp of transaction and amount paid. This graph can then be embedded in lower dimensional space. Principal component analysis and spectral embedding by eigenvalue decomposition can be used as effective embedding techniques for this case. Feature vectors can then be built from the flattened nodes. These target vectors (or matrix for a multi-class target regime) of known labels can be through a nueral network to train the model.

DEEP FEATURE SYNTHESIS

We can leverage automated feature engineering approach to dramatically reduce false positives in fraud detection. Deep Feature algorithm can be used to automatically derive behavioral features based on historical data of the credit card associated with a transaction. Additional features (behavioral patterns) are generated from the data and can then be fed through a Random Forest classifier to predict the fraudulent transactions.

PARENCLITIC NETWORK ANALYSIS + ANN

Complex networks and data mining can be integrated as complementary tools in order to extract, synthesize, and create new representations of a data source, with the aim of, for instance, discover new hidden patterns in a complex structure. The appropriate integration of complex network metrics can result in improved classification rates with respect to classical data mining algorithms and, reciprocally, there are many situations in which data mining can be used to solve important issues in complex network theory and applications. Features extracted from a network based representation of data, leveraging on a recently proposed parenclitic approach, can play an important role: while not effective in themselves, such features can improve the score obtained by a standard ANN classification model.

PRODUCTION IMPLEMENTATION

Implementing the final model in a production like environment has paramount importance when it comes to organizations. These models can be deployed in production and made to predict real time in order to prevent fraud from happening rather than predicting it after the fact.

REFERENCES

[1] http://news.mit.edu/2018/machine-learning-financial-credit-card-fraud-0920
[2] https://docs.featuretools.com/index.html
[3] https://uu.diva-portal.org/smash/get/diva2:1150344/FULLTEXT01.pdf
[4] https://medium.com/@curiousily/credit-card-fraud-detection-using-autoencoders-in-keras-tensorflow-for-hackers-part-vii-20e0c85301bd
[5] https://www.datascience.com/blog/fraud-detection-with-tensorflow\
[6] https://www.kaggle.com/arjunjoshua/predicting-fraud-in-financial-payment-services
[7] https://github.com/stellargraph/stellargraph
[8] https://github.com/georgymh/ml-fraud-detection
[9] https://datascience.stackexchange.com/questions/24081/what-are-graph-embedding
[10] https://www.experoinc.com/post/node-classification-by-graph-convolutional-network
[11] https://cambridge-intelligence.com/detect-credit-card-fraud-with-network-visualization/
[12] https://datascience.stackexchange.com/questions/24081/what-are-graph-embedding
[13] https://www.experoinc.com/post/node-classification-by-graph-convolutional-network
[14] https://www.experoinc.com/online-seminar/fraud-detection-and-optimization-using-graph-ml
[15] https://blog.securedtouch.com/fraud-losses-and-false-positives-numbers
[16] https://nilsonreport.com/upload/content_promo/The_Nilson_Report_10-17-2016.pdf
[17] https://www.experoinc.com/post/fraud-detection-using-deep-learning-on-graph-embeddings-and-topology-metrics
[18] https://www.experoinc.com/online-seminar/fraud-detection-and-optimization-using-graph-ml
[19] http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science.html
[20] https://www.youtube.com/watch?v=AeNufTq1W5I
[21] https://youtu.be/QkQLIDFlkyc