University of Michigan - Dearborn

Project report on

Fraud Detection on Credit Card Transactions using Transactional and User Data

Prof David Daniszewski

Group Members

Sanjyot Thete (48844546)
Vishnu Priya Velamuri (77120506)
Vinayaka Manjunatha Malya (02338153)

Introduction

Imagine standing at the check-out counter at the grocery store with a long line behind you and the cashier not-so-quietly announces that your card has been declined. At this moment, you probably aren't thinking about the data science that determined your fate.

Embarrassed, and certain you have the funds to cover everything needed for an epic nacho party for 50 of your closest friends, you try your card again. Same result. As you step aside and allow the cashier to tend to the next customer, you receive a text message from your bank. "Press 1 if you really tried to spend $500 on cheddar cheese."

While perhaps cumbersome (and often embarrassing) at the moment, this fraud prevention system is actually saving consumers millions of dollars per year. Researchers from the IEEE Computational Intelligence Society (IEEE-CIS) want to improve this figure, while also improving the customer experience. With higher accuracy fraud detection, you can get on with your chips without the hassle.

Objective

Our main objective is to predict the probability that an online transaction is a fraud or not by training benchmarked machine learning models on a challenging large scale dataset. We also want to analyse the data for obvious trends and patterns to improve the accuracy of our prediction.

Data Description

The data is obtained from a Kaggle competition hosted by Vesta. The data comprises two files, identity, and transaction, which can, in turn, be joined on TransactionID. But, not all transactions have corresponding identity information. The identity file has 144233 rows and 41 columns and the transaction file has 590540 rows and 394 columns. The real meaning of each feature is not known and it is not revealed by the host since the nature of data is confidential.

a) Identity table

Variables in this table are identity information – network connection information (IP, ISP, Proxy, etc) and digital signature (UA/browser/os/version, etc) associated with transactions.

They're collected by Vesta's fraud protection system and digital security partners.

The field names are masked and a pairwise dictionary was not provided for privacy protection and contract agreement.

- Numeric features: id_01 - id_11

Categorical Features:

- DeviceType
- DeviceInfo
- id_12 - id_38

b) Transaction table

The transaction table basically has information related to the transaction such as purchaser, recipient email id domains, credit card information, amount and so on. These features are also masked and do not contain the actual values.

- TransactionDT: timedelta from a given reference datetime (not an actual timestamp)
- TransactionAMT: transaction payment amount in USD
- ProductCD: product code, the product for each transaction
- card1 - card6: payment card information, such as card type, card category, issue bank, country, etc.
- addr1, addr2: address
- dist1, dist2: distance
- (P_) and (R_) emaildomain: purchaser and recipient email domain
- C1-C14: counting, such as how many addresses are found to be associated with the payment card, etc. The actual meaning is masked.
- D1-D15: timedelta, such as days between previous transaction, etc.
- M1-M9: match, such as names on card and address, etc.

- Vxxx: Vesta engineered rich features, including ranking, counting, and other entity relations.

Categorical Features:

- ProductCD
- card1 - card6
- addr1, addr2
- P_emaildomain, R_emaildomain
- M1 - M9

Approaches (which were considered)

a) Preprocessing approaches

- Sample cases such that all fraud cases are covered and the class imbalance is lessened (20k fraud, 80k legit)
- Remove high null ratio features
- For each product, remove a set of features with high null ratio (these features can be different across products)

b) Modeling approaches (which were considered)

- 5 different models for 5 products in the data
- Xgboost, LightGBM algorithms since they natively handle null values

Exploratory Data Analysis

On initial exploration of the dataset, we found a high portion of data as null values. In order to estimate how null values are distributed and their proportions, we explored the data further concentrating on null values. The following are the experiments and their outcomes.

a) Null percentage in features

The below plot shows 3 distinct blocks of features grouped together by their null value percentage. One block has the least nulls, one around 50% nulls and the last block with over 80% nulls.

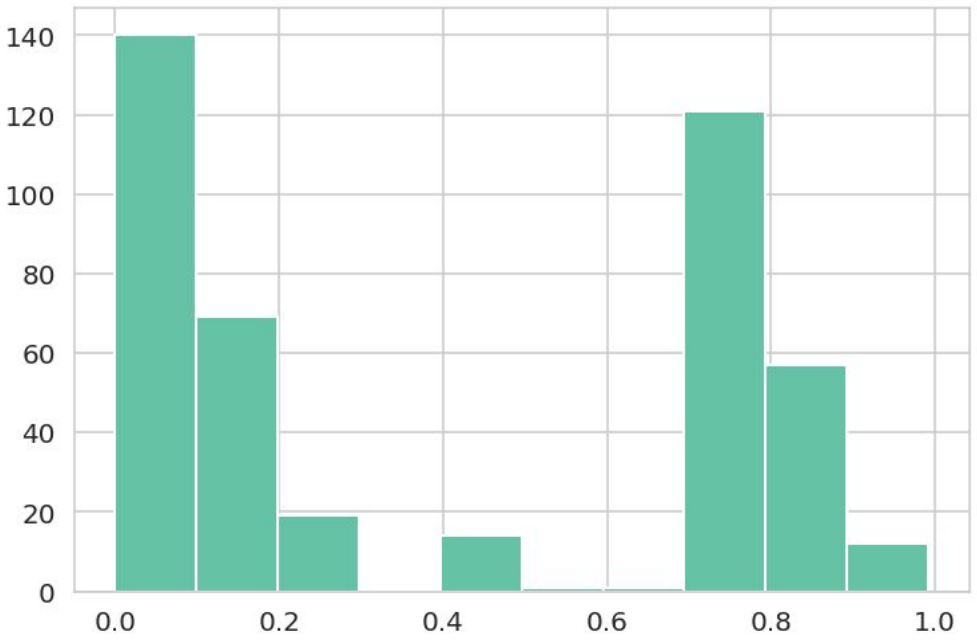


Figure 1: Histogram of null value percentage in features. X-axis is the null value percentage while Y-axis shows the number of features falling in that bucket.

b) Null Value blocks

To further explore the behaviour of null values and their distribution across the features, we try to see the null value distribution across data in more detail. This experiment gave us an insight that sets of features are tightly coupled with each other as apparent in the commonality of their null values (blocks). If we look at Figure 2, we can clearly see this relation. A block from around feature 95 to feature 140 is mostly not null while a block from 140 to 160 is mostly null, on the same rows.

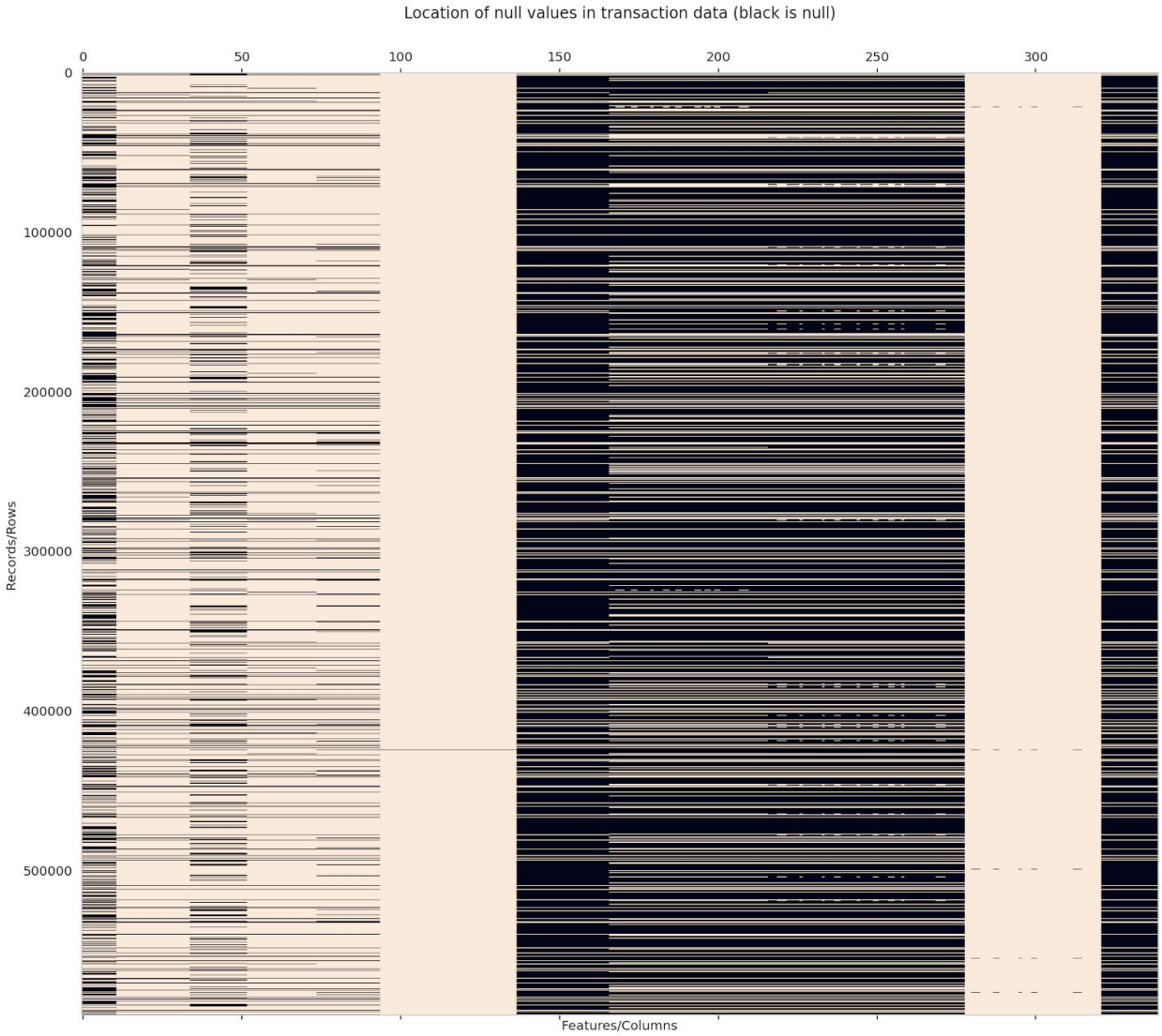


Figure 2: Null value visualization over the entire dataset. Vertical axis contains the actual rows in data scaled to fit in the image. Horizontal axis contains the numerous features present in data. Black color represents null values.

c) Fraud percentage across products

From the problem description, we know that the column ‘ProductCD’ contains the specific product for each transaction. These products are tagged as ‘C’, ‘H’, ‘R’, ‘S’ and ‘W’. To find out if the products are mostly independent of each other, we check the average fraud percentage and average transaction amount split by fraud or non-fraud, for each product. This showed us (Figure 3) that there was a significant variation in these quantities across the products. The highest fraud percentage is for product C at 12% while the least is for product W at 2%.

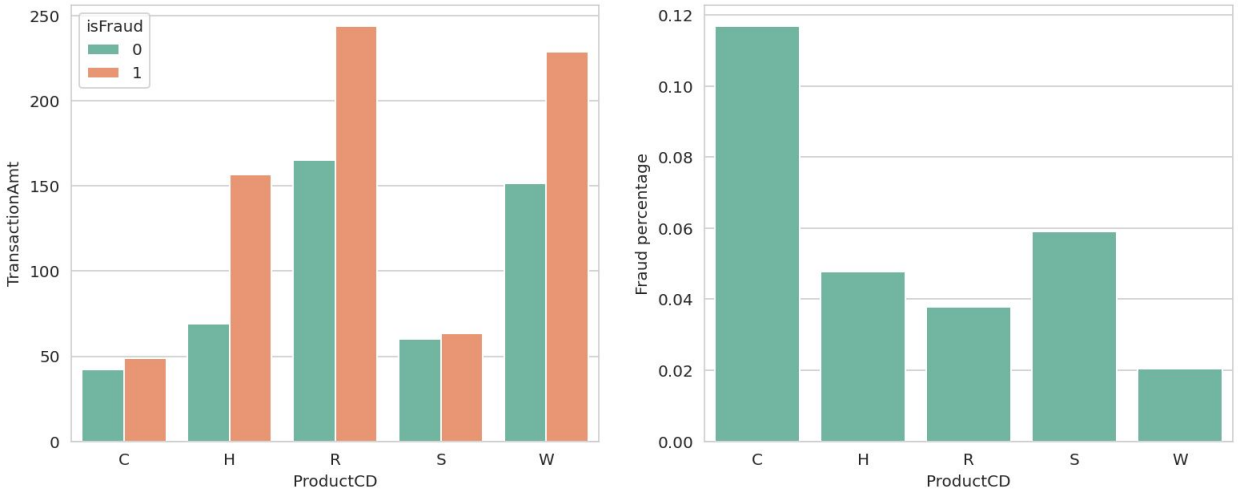
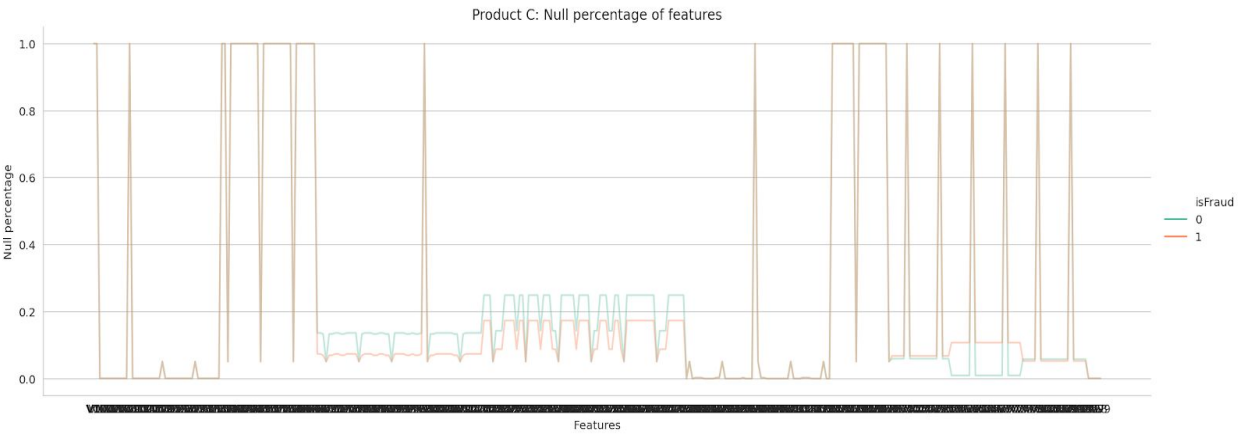


Figure 3: Transaction amount and Fraud percentage across each product

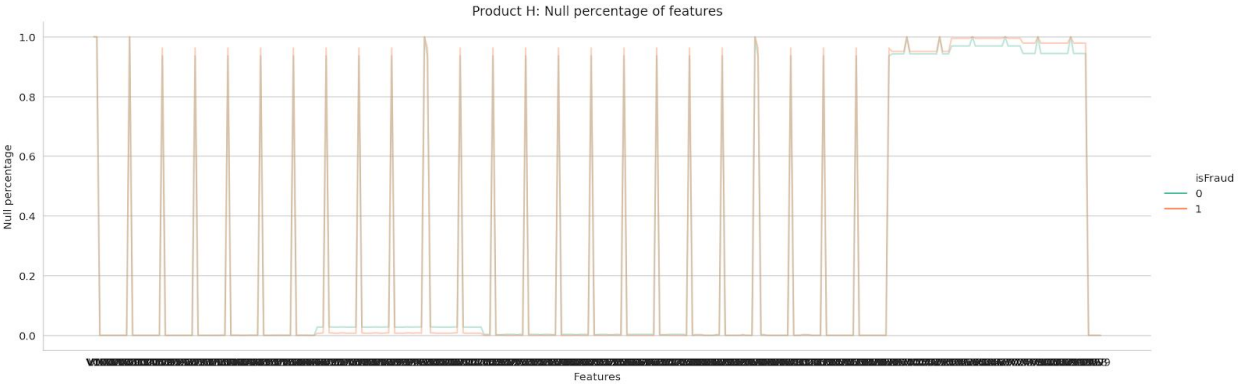
d) Null percentage across product split by fraud status

In order to understand the independence of each product from others, we decided to plot the null percentage for each feature, for each product.

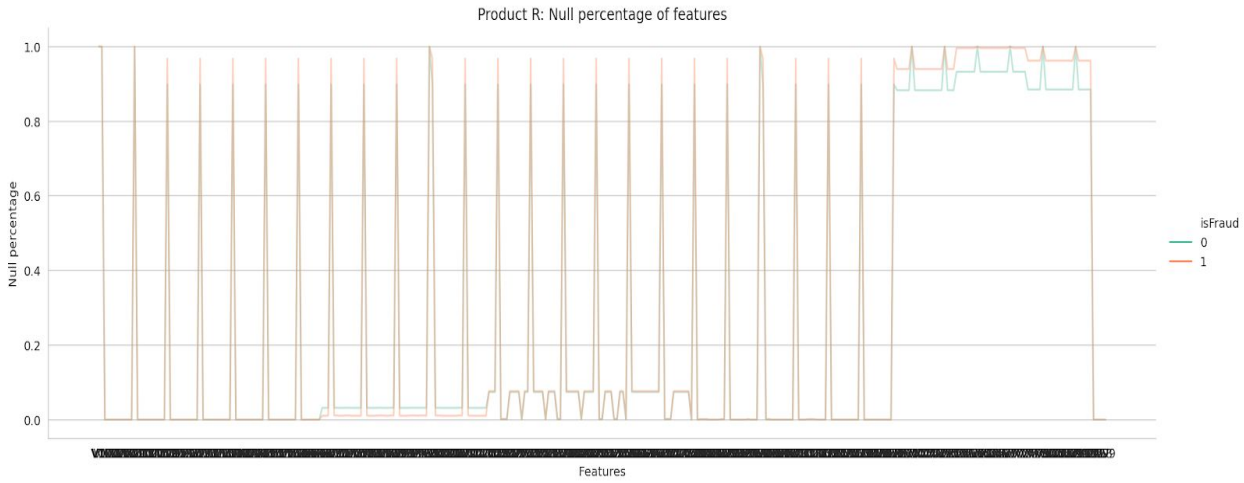
Product C



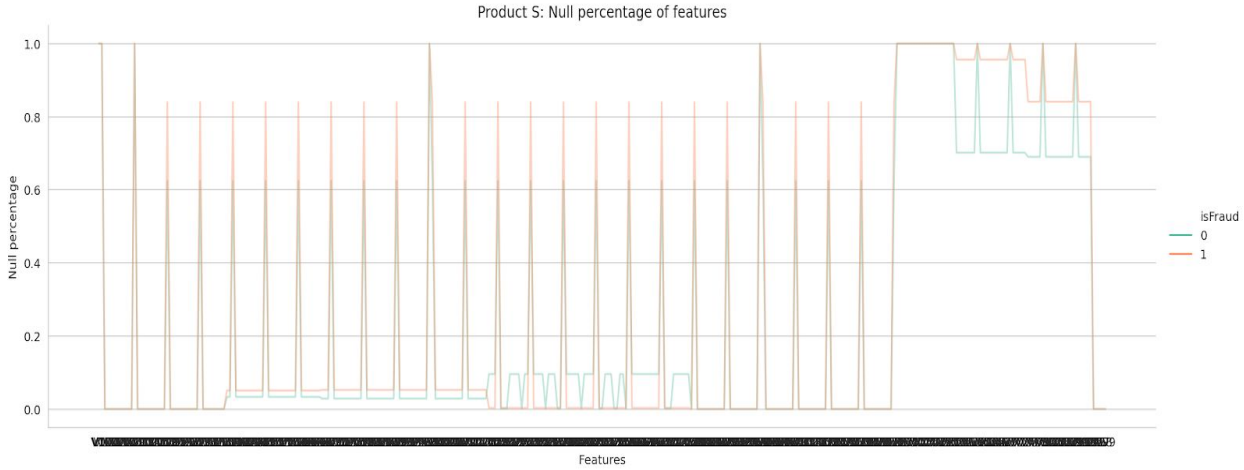
Product H



Product R



Product S



Product W

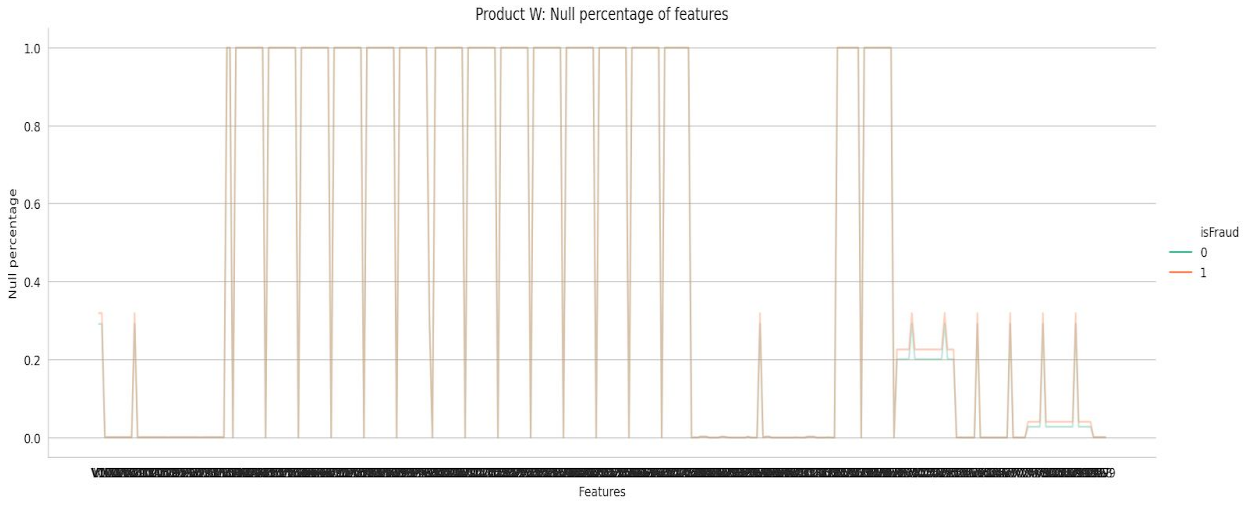


Figure 4: Null percentage of features across products

We observe two significant things:

1. The null percentage varies across fraud status, thus there is some information in the null presence that can be learnt. Hence we need to choose a modeling approach that can handle null values by design.
2. The null percentage in features varies quite a lot. One set of features showing nulls in one product does not have similar behavior for a different product.

Clearly observing a product independence, we decided to not use one machine learning approach to model this case, but instead use 5 different ML models to learn the trends in frauds for each product, independently.

We decide to use the XGBoost algorithm which implements a forest of boosted decision trees to learn patterns.

Final approach

- Split the data by Products since they show quite independent distributions in null values, suggesting that each product is independent from others to a high degree.
- Use XGBoost to train a binary classifier for each product and Grid Search for hyperparameters
- Combine the product-specific trained classifier into a single function for training and predicting
- Model performance measurement by Accuracy, F1 score, precision, recall, and AUROC

Preprocessing (Label encoding)

There are several categorical features in both the tables as described in the data description section. To feed data into a machine learning algorithm that handles null values well, like XGBoost or LGBM, we needed to convert categorical features to int, float or bool.
For this process, we chose Scikit-learn's Label Encoder to convert categorical features to numeric values.

Training

File summary

We are given a transaction file and a user identity file. We merge these two datasets while conserving all the rows in the transaction file, by using a left join. This combined dataset has 590,540 rows and 434 columns. Out of the 434 columns, we use 430 as features, rest of the 4 were either Id columns or the fraud label. We split the data into 67% training and 33% testing sets.

Infrastructure

Since this data was very large, we faced memory issues while doing operations on it. We then resorted to Statistics & Computation Service (SCS) of University of Michigan to help with the compute and memory requirements. We used 'ssh' to access the servers, set up conda and re-created our development environment on the SCS server. We operated JupyterLab on the server while forwarding the notebook interface over 'ssh' to our own laptops. From this point onwards, all the work was done as if the development was happening locally.

XGBoost

XGBoost is an optimized distributed gradient boosting library. Gradient boosting approach is a powerful way to learn patterns. A forest of trees is grown like Random Forest, but every observation which is difficult to classify is given an increased weight for the subsequent trees. This results in a forest which is able to focus on hard to learn patterns, unlike Random Forest, which is agnostic of the observation level difficulty in classification.

We start with mostly default arguments to the XGBoost classifier and improve upon them in the grid search. The parameters we used are:

a. **learning_rate=0.1**
   Learning rate used for boosting
b. **n_estimators=1000**
   Number of trees in the forest
c. **max_depth=5**
   Maximum depth of trees, to limit overfitting
d. **min_child_weight=1**

Minimum sum of instance weights needed in leaf nodes. This parameter discourages splitting if the improvement in information gain is not significant, hence acting as a kind of regularization.

e. **gamma=0**
   Minimum loss reduction required to make a further partition on a leaf. This parameter discourages splitting if the improvement in information gain is not significant.
f. **subsample=0.8**
   Percentage of rows to sample from original data for constructing each tree
g. **colsample_bytree=0.8**
   Percentage of column to sample from original feature set for constructing each tree
h. **objective= 'binary:logistic'**
   With this objective, we choose the log loss function as our loss function
i. **nthread=xgb_threads**
   Number of threads to make available to XGBoost for training an instance. This was iterated over and settled on 6, based on the SCS machine compute power.
j. **scale_pos_weight=1**
   Balancing of positive and negative weights on observation. We kept this at default.
k. **seed=42**
   Our random seed for this project
l. **verbosity=1**
   To print training logs on the console

Hyperparameter grid search

We tune two main parameters in our model,

a. learning_rate
   This parameter has one of the most significant impacts on the training behavior. We iterate this value over 0.001, 0.01 and 0.1 for 4 of the 5 products.
b. n_estimators
   Number of trees determines the behavior of the forest significantly. Hence we iterate over the values 500, 1000, 2000 and 5000 for 4 of the 5 products.

For the product ‘W’, since the number of records were very high (439, 670), we reduced the parameter space for this product to complete training in a reasonable time.

We set the scoring criteria for our Grid Search as the area under the ROC curve of the models. For each combination of the above two parameters, we perform a 5-fold cross-validation and get a mean AUROC for the models. This score is obtained for each parameter combination in grid search for each product. Best hyperparameters are then found out for each product, independently of each other. We give each Grid Search process a total of 4 CPU threads. Each of these threads spin out 6 threads for XGBoost training, thus totalling a 24 threaded model training and grid search. Total training process took approximately 7 hours.

Results

Accuracy

We obtain an accuracy of 98.12% on our test set (held out data). We understand that since there is a very large class imbalance (only 3.5% of labels are fraud), if we predicted everything as non-fraud, we would still get an accuracy of 96.5%. Hence, we look at metrics which perform better even in the cases of class imbalance

Precision/Recall and F1-score on Fraud class

We achieve following values of these metrics,

- Precision: 0.946
- Recall: 0.49
- F1-score: 0.646
- AUROC: 0.934

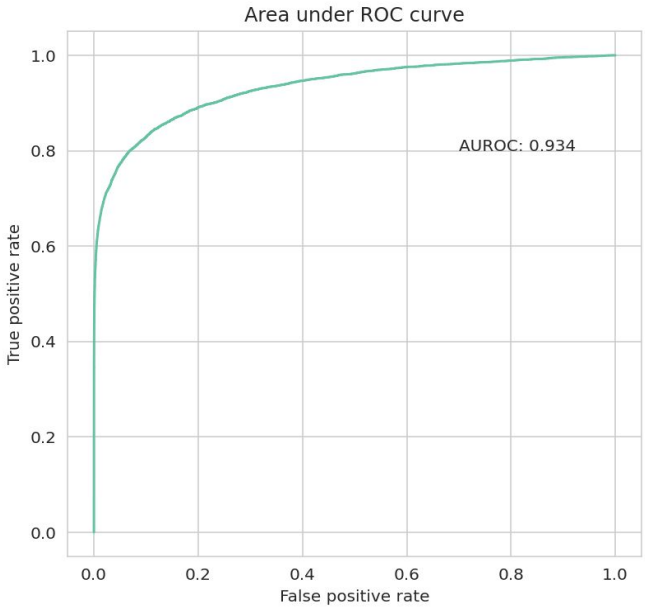
We observe and conclude a couple of things from these metrics

1. We observe that Precision is high for our models. This implies that out of our predictions, 94.6% percent of cases turn out to be fraudulent transactions.
2. Recall is not so great. This implies that we are not able to catch some frauds and our model is labelling 50% frauds as false negatives.

3. F1-score is depicting a balanced view of the Precision-Recall trade-off
4. AUROC curve is a good value. This implies our model has learnt the class discrimination pretty well

AUROC curve

We plot the ROC curve and calculate the area under it.



We observe that much of the area in this unit square is covered by the ROC curve of our model.

Conclusion

We attempt to solve a machine learning problem of fraud detection in a credit card company. We perform exploratory data analysis on a very large dataset and gain significant insights which help direct our modelling approach. Data was preprocessed and a ML classifier using XGBoost algorithm was trained on a multi-core machine and the output was analyzed for model performance. We achieved a precision of 94.6% and a recall of 49% on the fraud class in the test set.

Contributions

All 3 team members were involved in each stage of problem solving. We defined owners for sub-problems. Vinayak was involved in literature survey and problem formulation. Vishnu was responsible for the exploratory data analysis and deriving insights from them. Sanjyot was responsible for ML workflow, training and results.

References

i) http://www.aaai.org/Papers/Workshops/1997/WS-97-07/WS97-07-015.pdf

ii) https://link.springer.com/chapter/10.1007/978-3-642-04003-0_10

iii) https://ieeexplore.ieee.org/abstract/document/6784638

iv) https://ieeexplore.ieee.org/abstract/document/618940

v) https://www.kaggle.com/c/ieee-fraud-detection/data

vii) https://www.kaggle.com/c/ieee-fraud-detection/discussion/101203