**KAGGLE- CIS IEEE FRAUD DETECTION**

A PROJECT REPORT

**WEB MINING (CSE3024)**

*submitted by*

**ANISHA CHAZHOOR**

**(17BCE1053)**

**YASASHVINI**

**(17BCE1077)**

*in partial fulfillment for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

in

**COMPUTER SCIENCE AND ENGINEERING**



MAY, 2020

**ACKNOWLEDGEMENT**

We are highly grateful to Dr.Shridevi.S for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are also obliged to VIT University for giving us this opportunity. We will strive to use gained skills and knowledge in the best possible way. We would also like to thank our friends and family members for their constant support and encouragement without which this project would not have been possible.

**Anisha Chazhoor**

**17BCE1053**

**Yasashvini**

**17BCE1077**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapter** | **Title** | **Page** |
|  |  |  |
|  | Acknowledgement | 1 |
|  | Table of contents | 2 |
|  | Abstract | 3 |
|  |  | 4 |
| 1 | Introduction | 5 |
| 2 | Dataset Description |  |
| 3 | System Design |  |
|  | 3.1 Pre-processing | 7 |
|  | 3.2 Algorithm | 8 |
| 4 | Implementation and Result | 11 |
| 5 | Conclusion and future work | 13 |
|  |  |  |
|  | References | 14 |
|  |  |  |
|  |  |  |
|  |  |  |

**Abstract**

In this project, a technique for CIS IEEE fraud detection is developed. Credit card is quite useful for our day to day life. The goal is to detect least and accurate fraud detection. There are several methods for detecting credit card fraud like Group method of data handling, Bayesian learning and Neural network. Our method shall use Light GBM model where the tree is grown vertically to predict if a credit card is a fraud or not.

**Chapter 1 – 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. 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) in 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. IEEE-CIS works across a variety of AI and machine learning areas, including deep neural networks. The data comes from Vesta's real-world e-commerce transactions and contains a wide range of features from device type to product features. Our aim is to improve the efficacy of fraudulent transaction alerts for millions of people around the world, helping hundreds of thousands of businesses reduce their fraud loss and increase their revenue. There are various algorithms like Group method of data handling, Bayesian learning and Neural network with the help of which the accuracy can be improved.

**Chapter 2 – Dataset description**

The data is broken into two files identity and transaction, which are joined by TransactionID. Not all transactions have corresponding identity information.

Categorical Features – Transaction: ProductCD, card1 - card6, addr1, addr2, P\_emaildomain, R\_emaildomain, M1 - M9

Categorical Features – Identity: DeviceType, DeviceInfo, id\_12 - id\_38

The TransactionDT feature is a timedelta from a given reference datetime (not an actual timestamp).

Transaction table:

* 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.
* addr: address
* dist: 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.

Identity table:

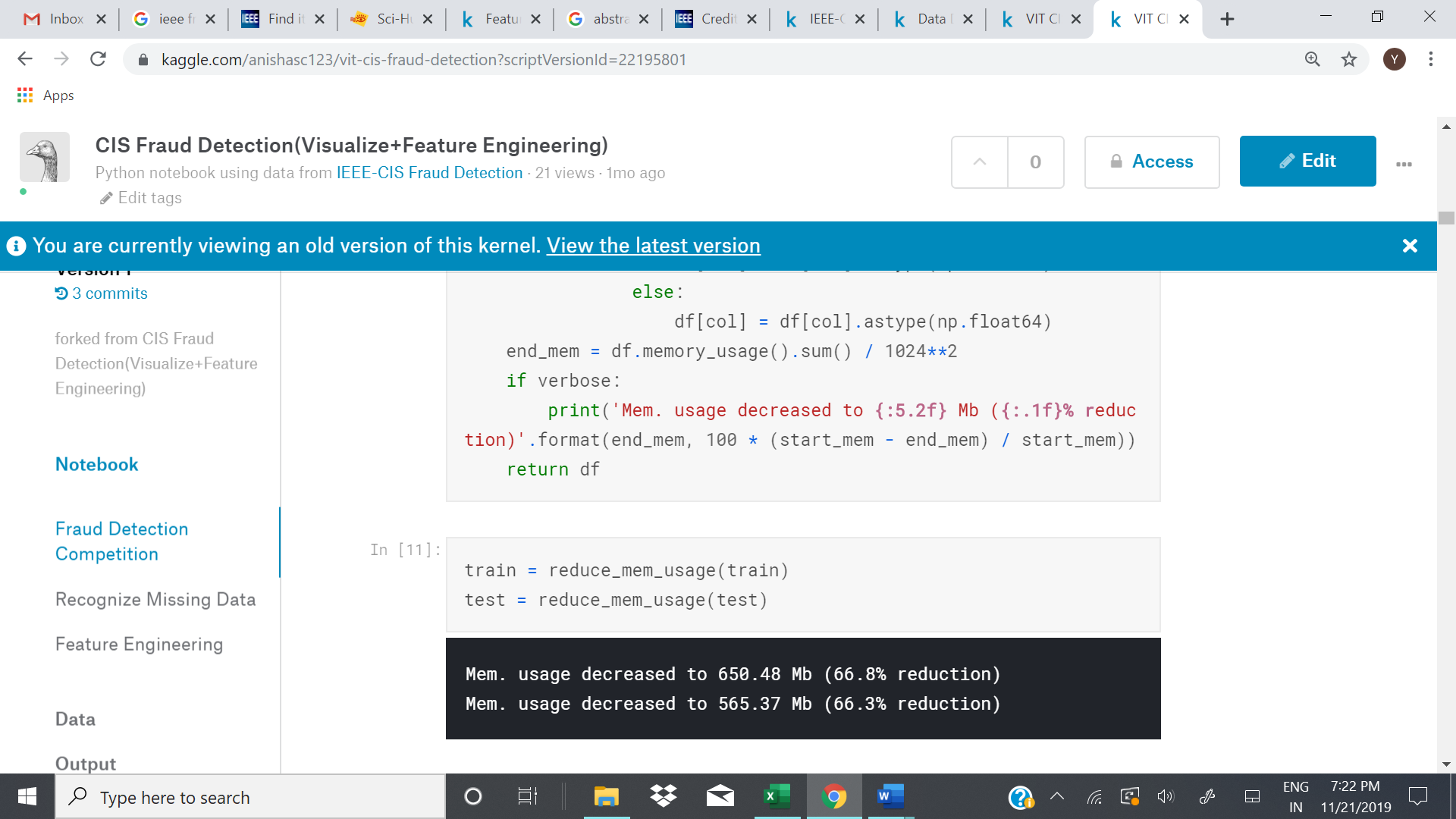
Variables in this table are identity information – network connection information (IP, ISP, Proxy, etc) and digital signature (UA/browser/os/version, etc) associated with transactions. They're collected by Vesta’s fraud protection system and digital security partners.

**Chapter 3 – System Design**

**3.1 Preprocessing:**

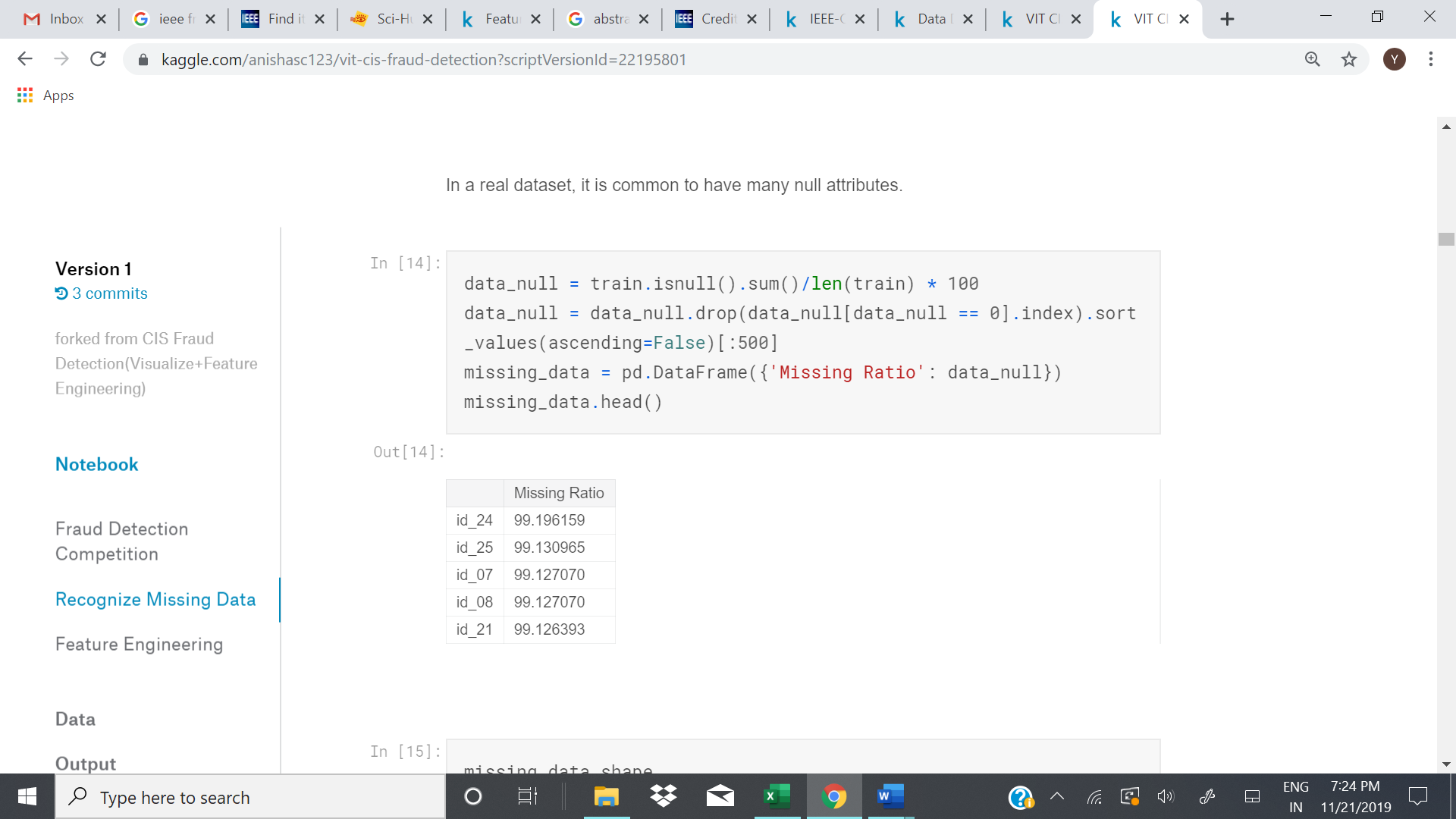
3.1.1 Reduce memory

The memory used by the dataset is very high. Initially, the dataset required 973.77Mb. This was due to unnecessary memory usage of the high memory data-types. This was replaced with lower order datatypes keeping the data loss in consideration. As one can observe, the memory usage reduced more than 65%.



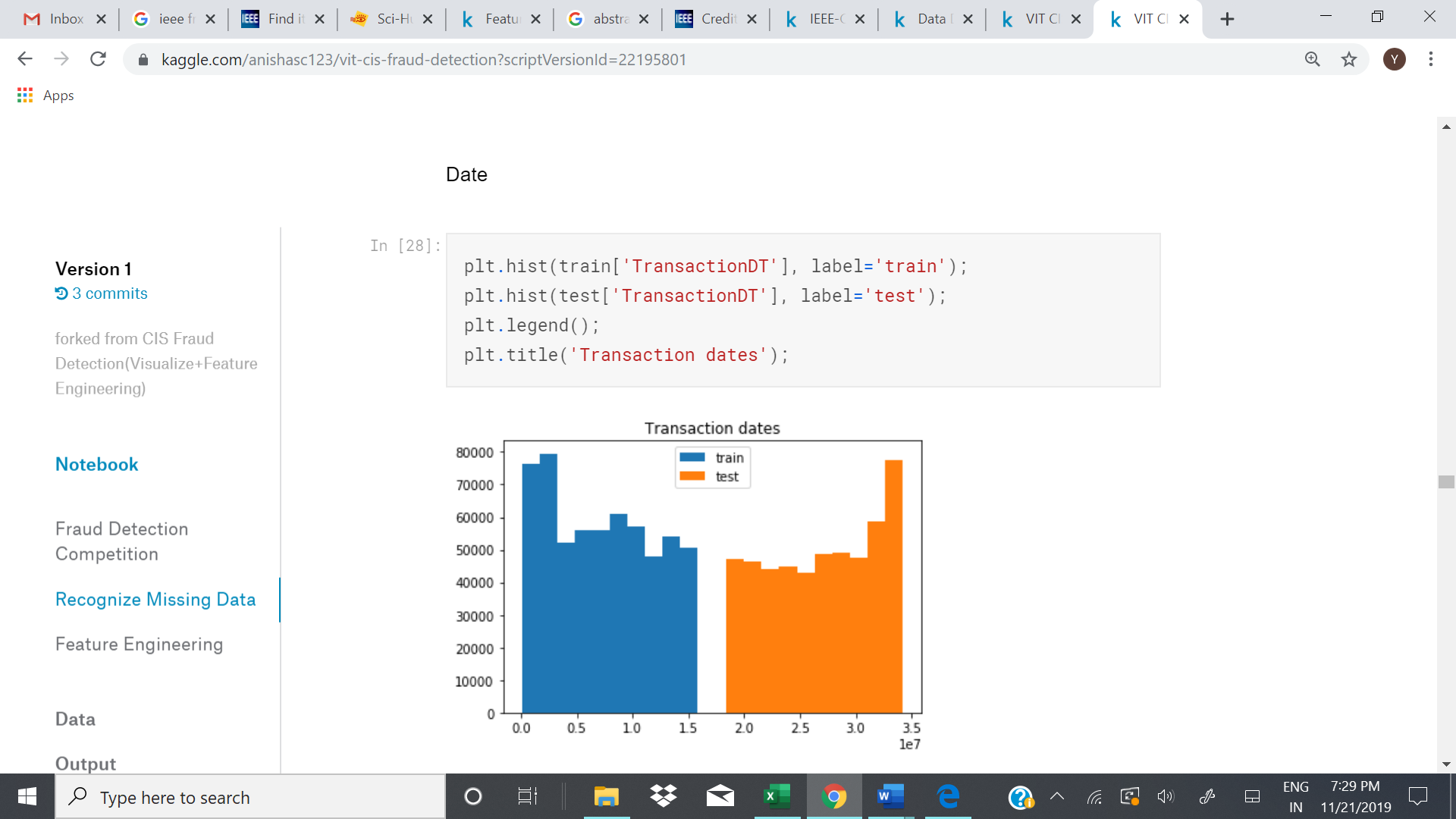
3.1.2 Recognize missing data

This dataset has a lot of missing data. Here we will list out the missing data ratio in descending order. Later we will drop the columns with more than 90% NULL values and 90% repeated values.



3.1.3 Date

Next we will plot the transaction date for train and test data to check for overlapping values. We can conclude that they are completely disjoint and independent of each other.



3.1.4 Extract email domain from email-ID

Replace NULL email IDs with “Email ID not provided” and the extract the email domain from the email ID. We can observe that there are several domain providers too, so we will extract the domain providers and add it as a new column.

3.1.5 Modify Device name

There are several redundant device names spelt in different ways. Sometimes abbreviations and layman terms are used causing inconsistency in the data. We will replace the device names to more uniform names.

**3.2 Algorithm**

We will be using Light GBM algorithm, a relatively new algorithm which has found widespread use in modern machine learning problems on Kaggle. Light GBM is fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms. Also, it is surprisingly very fast, hence the word ‘Light’. There several advantages of Light GBM such as faster training speed and higher efficiency, lower memory usage, better accuracy than any other boosting algorithm and compatibility with Large Datasets. Moreover, parallel learning is supported.

The algorithm is as follows:

1. Create a pipeline for numerical and categorical data separately. Then we will take the union of the two pipelines into a full pipeline.
2. Next we will declare certain no\_of\_folds for KFOLDs. Here we have taken it as 2
3. Split the dataset according to the parameters given. Set verbose eval as 200 so that the accuracy is printed after every 200 iterations. Set maximum iterations at 1800 and maximum initial stopping rounds as 100.
4. Run the classifier Light GBM using the given parameters on the given pipeline.

Parameters used:

1. num\_iterations: number of boosting iterations to be performed ; default=100; type=int
2. num\_leaves : number of leaves in one tree ; default = 31 ; type =int
3. device : default= cpu ; options = gpu,cpu. Device on which we want to train our model. Choose GPU for faster training.
4. max\_depth: Specify the max depth to which tree will grow. This parameter is used to deal with overfitting.
5. min\_data\_in\_leaf: Min number of data in one leaf.
6. feature\_fraction: default=1 ; specifies the fraction of features to be taken for each iteration
7. bagging\_fraction: default=1 ; specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.
8. min\_gain\_to\_split: default=.1 ; min gain to perform splitting
9. max\_bin : max number of bins to bucket the feature values.
10. min\_data\_in\_bin : min number of data in one bin
11. num\_threads: default=OpenMP\_default, type=int ;Number of threads for Light GBM.
12. label : type=string ; specify the label column
13. categorical\_feature : type=string ; specify the categorical features we want to use for training our model
14. num\_class: default=1 ; type=int ; used only for multi-class classification

**Chapter 4: Implementation and results**

The Light GBM has been implemented according to the algorithm given in the previous chapter. Initially preprocessing is also performed. The light gbm parameters used are as follows:

lgb\_params = {

'objective':'binary',

'boosting\_type':'gbdt',

'metric':'auc',

'n\_jobs':-1,

'learning\_rate':0.005,

'num\_leaves': 2\*\*8,

'max\_depth':-1,

'tree\_learner':'serial',

'colsample\_bytree': 0.7,

'subsample\_freq':1,

'subsample':0.7,

'n\_estimators':1800,

'max\_bin':255,

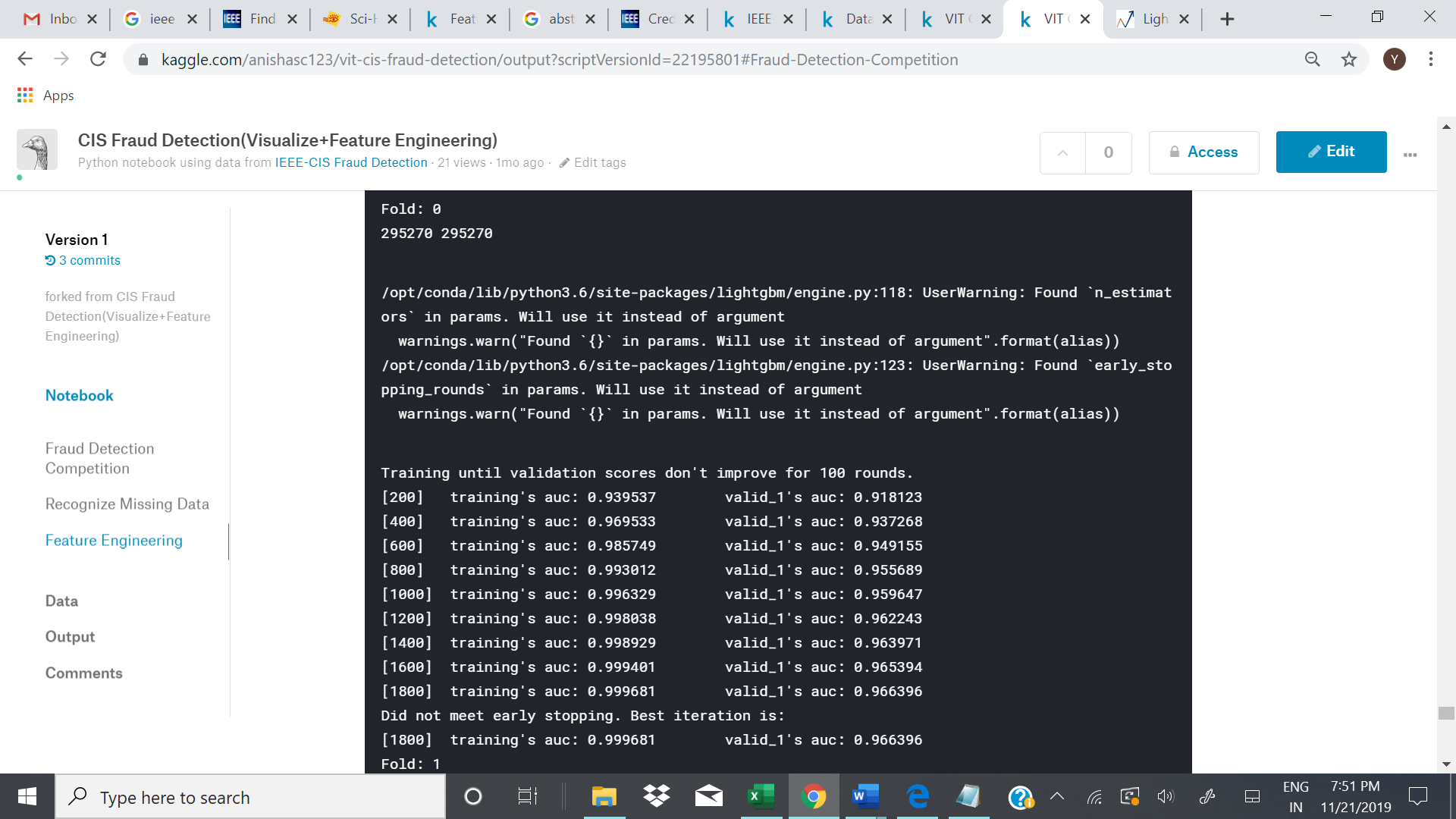
'verbose':-1,

'seed': SEED,

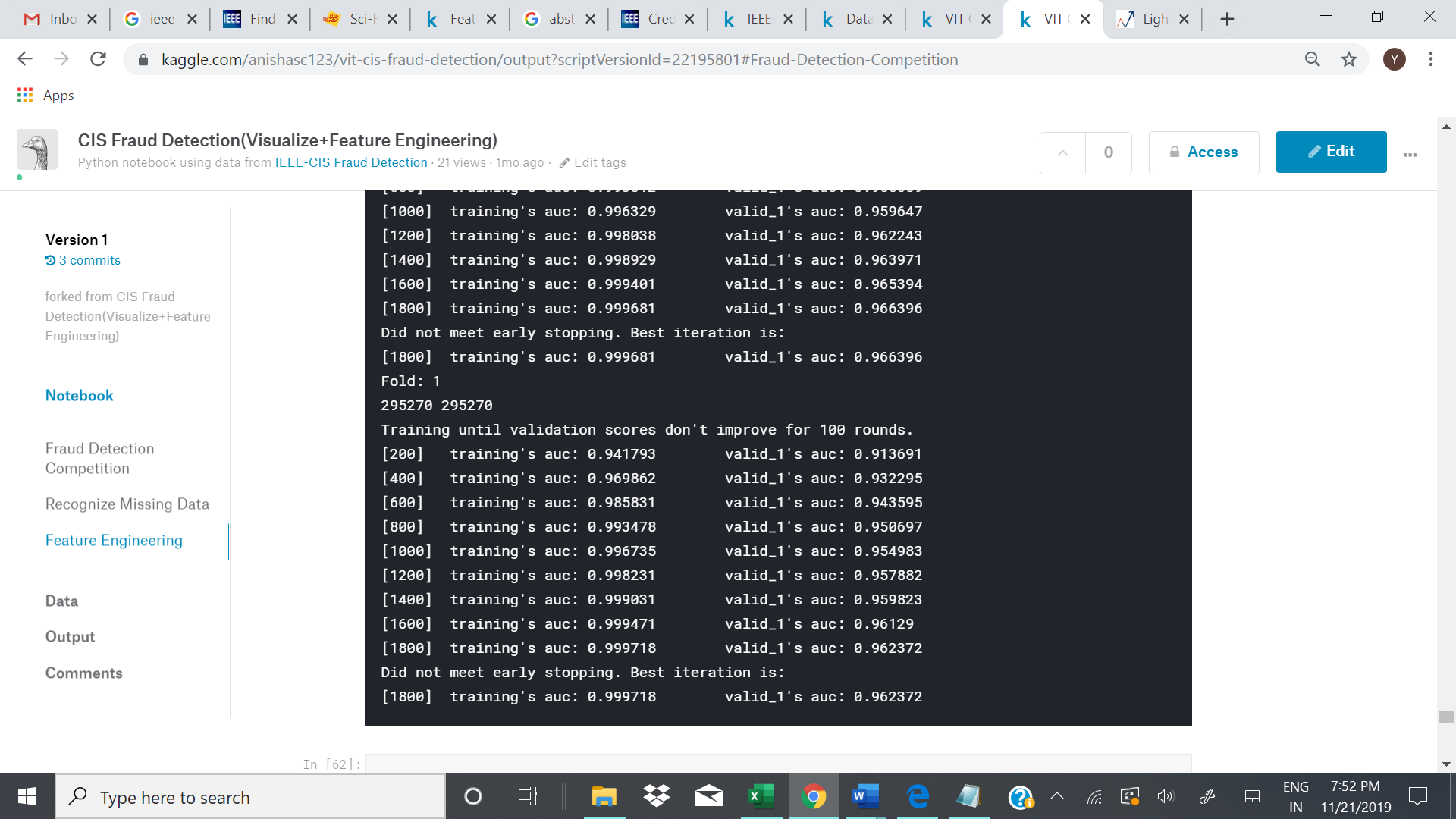
'early\_stopping\_rounds':100,

}

The results for Fold=0



Results for fold=1



The predicted values are stored in a csv file. The 1st five records are:

|  |  |  |
| --- | --- | --- |
| 1 | TransactionID | isFraud |
| 2 | 3663549 | 0.0012161125867563931 |
| 3 | 3663550 | 0.0033129234212224577 |
| 4 | 3663551 | 0.0018886075821488138 |
| 5 | 3663552 | 0.0011563402598689342 |

**Chapter 5: Conclusion and future work**

Our algorithm performed with a public score of **94.49%** and a private score of **92.19%**.

Inferences drawn:

Owing to such imbalance in data, an algorithm that does not do any feature analysis and predicts all the transactions as non-frauds will also achieve an accuracy of 99.828%. Therefore, accuracy is not a correct measure of efficiency in our case. We need some other standard of correctness while classifying transactions as fraud or non-fraud.

The ‘Time’ feature does not indicate the actual time of the transaction and is more of a list of the data in chronological order. So we assume that the ‘Time’ feature has little or no significance in classifying a fraud transaction. Therefore, we eliminate this column from further analysis.

Future work:

LightGBM can significantly outperform XGBoost and SGB in terms of computational speed and memory consumption. For the future work, we will study the optimal selection of a and b in Gradient-based One-Side Sampling and continue improving the performance of Exclusive Feature Bundling to deal with large number of features no matter they are sparse or not.

**References**

[1] LightGBM: A Highly Efficient Gradient Boosting Decision Tree.

[2] Lokanayaki, K., and A. Malathi. "Data preprocessing for liver dataset using SMOTE." *International Journal of Advanced Research in Computer Science and Software Engineering*

[3] Lin, Weiwei, et al. "An ensemble random forest algorithm for insurance big data analysis." Ieee Access 5 (2017): 16568-16575.