Fraud Detection in Electricity and Gas Consumption

1st Thirunaavukkarasu Murugesan
dept. of Computer Science
Stevens Institute of Technology
New Jersey, USA
tmuruges@stevens.edu

2nd Sriram Ananthakrishna
dept. of Applied Mathematics (Data Science)
Stevens Institute of Technology
New Jersey, USA
sananth3@stevens.edu

3rd Kishankumar Ravikumar
dept. of Computer Science
Stevens Institute of Technology
New Jersey, USA
kravikum@stevens.edu

***Abstract*—The Tunisian Company of Electricity and Gas (STEG) is a public and a non-administrative company, it is responsible for delivering electricity and gas across Tunisia. The company suffered tremendous losses in the order of 200 million Tunisian Dinars due to fraudulent manipulations of meters by consumers. Using the client's billing history, the aim of the project is to detect and recognize clients involved in fraudulent activities.**

I. Introduction

Fraud detection is the process of recognizing unauthorized activities where money or property is obtained through false pretenses, such as phishing, stolen credit cards, and identity theft. Fraud detection relies on machine learning and data analytics to identify these malicious transactions.

Fraud detection leverages machine learning, statistical analysis, and behavior monitoring to identify the patterns and strategies used by criminals to commit fraud. When precursors to fraud are identified, the system can stop fraudulent activity before any damage occurs.

Here, we are using the Tunisian Company of Electricity and Gas data to detect fraudulent manipulations of meters by consumers through supervised machine learning techniques.

II. Related Work

This problem statement is available in zindi.africa as an ongoing competition. Hence, no particular work has been done in this particular problem statement or data. But, there are ample sources to tackle fraud detection.

Machine Learning (ML) techniques have emerged as alternative for the development of automatic Fraud Detection Systems. ML makes possible to take into account the wide range of ways in which fraud can be performed. The fraud detection problem corresponds to a supervised classification task aiming to determine whether a new transaction is legitimate or fraudulent one.

Different Supervised machine learning algorithms like Decision Trees, Naive Bayes Classification, Least Squares Regression, Logistic Regression, Neural Networks (Multilayer perceptrons) and Support Vector Machine (SVM) are used to detect fraudulent transactions in real-time datasets. However, the generalized model to be built has two challenges that make the process a specially tough task[1].

The first one is the unavoidable imbalance in the class distribution on the datasets. On the real world, the number of fraud transactions are much less than legal transactions, but the skewness on the distribution provokes a bias in the model prediction.

Lastly, since the large number of transactions and the amount of the information that must be processed every day, it is necessary to use training techniques with proved scalability.

To tackle class imbalance, there are many methods like downsampling the majority class, or upsampling the minority class, or using SMOTE, or balanced RF algorithms for improving the accuracy of the built model. To tackle the vastness of the data, it was recommended to use Apache Spark and SQL dataframe libraries to improve the scalability of tthe solution[2].

III. Data Cleaning and Pre-processing

A. Description of Dataset

The data provided by STEG is composed of two files. The first one is comprised of client data and the second one contains billing history since 2005.

Variable definitions for Client data:

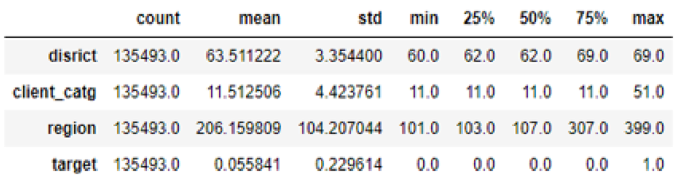
- Clientid: Unique id for client
- District: District where the client is
- Clientcatg: Category client belongs to
- Region: Area where the client is
- Creationdate: Date client joined
- Target: fraud:1 , not fraud: 0

Variable definitions for Invoice data:

- Clientid: Unique id for the client
- Invoicedate: Date of the invoice
- Tariftype: Type of tax
- Counternumber: a factor variable
- Countercoefficient: flag if excess consumption
- Consommationlevel1: Consumption level 1
- Consommationlevel2: Consumption level 2
- Consommationlevel3: Consumption level 3
- Consommationlevel4: Consumption level 4
- Oldindex: Old index
- Newindex: New index
- Monthsnumber: Month number
- Countertype: Type of counter

B. Data pre-processing

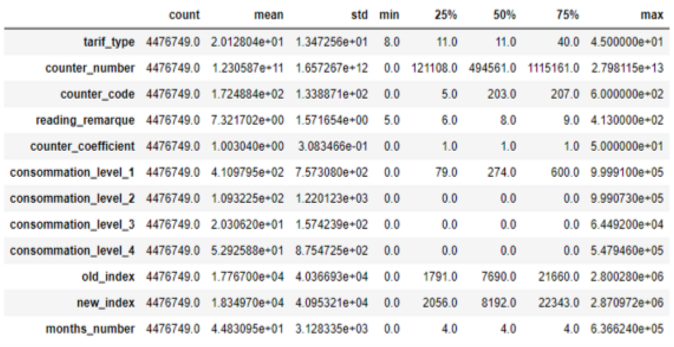
The first step towards pre-processing was to analyse the variables in both the files. Once the data was imported into



	count	mean	std	min	25%	50%	75%	max
disrict	135493.0	63.511222	3.354400	60.0	62.0	62.0	69.0	69.0
client_catg	135493.0	11.512506	4.423761	11.0	11.0	11.0	11.0	51.0
region	135493.0	206.159809	104.207044	101.0	103.0	107.0	307.0	399.0
target	135493.0	0.055841	0.229614	0.0	0.0	0.0	0.0	1.0

Fig. 1. Data Description of Client data

the python environment, the categorical variables were label-encoded. Once the process is finished, the data was checked for missing values. Surprisingly, there were no missing values or NaN values in the data. The data was then checked for outliers using the 2-SD method. These were the below steps that were taken to pre-process the data.



	count	mean	std	min	25%	50%	75%	max
tarif_type	4476749.0	2.012804e+01	1.347256e+01	8.0	11.0	11.0	40.0	4.500000e+01
counter_number	4476749.0	1.230587e+11	1.657267e+12	0.0	121108.0	494561.0	1115161.0	2.798115e+13
counter_code	4476749.0	1.724884e+02	1.338871e+02	0.0	5.0	203.0	207.0	6.000000e+02
reading_remarque	4476749.0	7.321702e+00	1.571654e+00	5.0	6.0	8.0	9.0	4.130000e+02
counter_coefficient	4476749.0	1.003040e+00	3.083466e-01	0.0	1.0	1.0	1.0	5.000000e+01
consommation_level_1	4476749.0	4.109795e+02	7.573080e+02	0.0	79.0	274.0	600.0	9.999100e+05
consommation_level_2	4476749.0	1.093225e+02	1.220123e+03	0.0	0.0	0.0	0.0	9.990730e+05
consommation_level_3	4476749.0	2.030620e+01	1.574239e+02	0.0	0.0	0.0	0.0	6.449200e+04
consommation_level_4	4476749.0	5.292588e+01	8.754725e+02	0.0	0.0	0.0	0.0	5.479460e+05
old_index	4476749.0	1.776700e+04	4.036693e+04	0.0	1791.0	7690.0	21660.0	2.800280e+06
new_index	4476749.0	1.834970e+04	4.095321e+04	0.0	2056.0	8192.0	22343.0	2.870972e+06
months_number	4476749.0	4.483095e+01	3.128335e+03	0.0	4.0	4.0	4.0	6.366240e+05

Fig. 2. Data Description of Invoice data

- Label encoding of factor variables
- converting date variable to datetime objects
- Checking for Missing values
- checking for outliers
- Creating new variables based on the business
- Joining the files to create a model dataset
- check unique value counts of each variable
- class balancing
- data normalization

Once the outliers were fixed, all the consommation levels aggregated with respect to the client id and the mean of the corresponding levels' were calculated. Once, data is ready, the files were merged based on client id to create a master dataset. This dataset will be used modeling purposes.

Further pre-processing steps include eliminating useless features. one of the methods to eliminate features include checking the feature's standard deviation, and if the sd of the feature is low, then, it means that the data is not varying much and doesnt contribute to the model.

Another way to eliminate features is to look for categorical features with high number of unique values. If the number of unique features is high, it means that the feature varies too much with respect to the dependent variable and no reliable prediction can be made on the output.

The last way to remove features is to look for correlation between variables. A heat map can be generated between

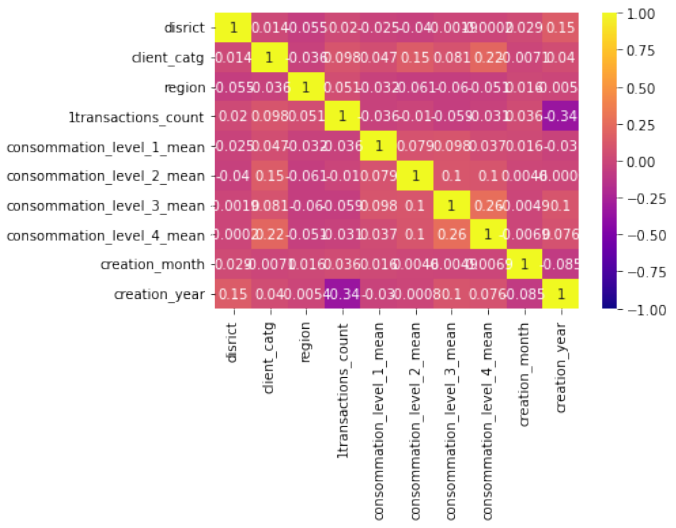


Fig. 3. Correlation Matrix

dependent variables to look for high correlation between variables and remove one of the variable.

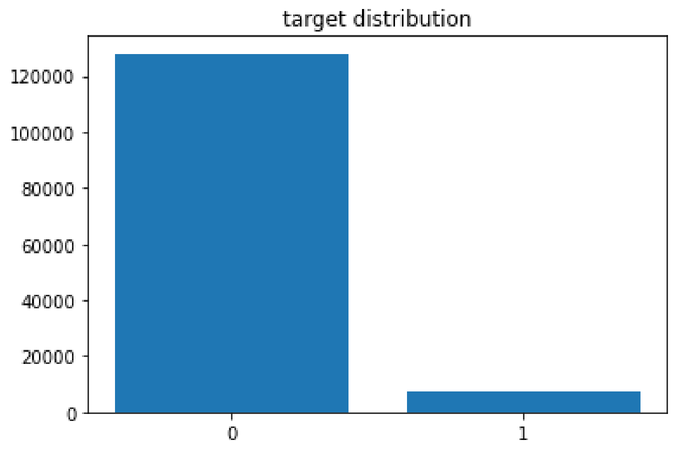


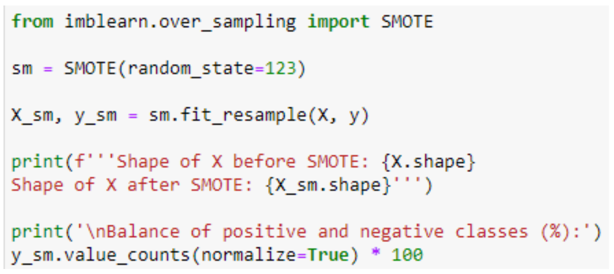
Fig. 4. Target label distribution

C. Handling Class Imbalance

One of the last major pre-processing step step which we carried out was class-balancing. This technique is carried out to reduce the bias in model. There are generally 3 methods of class balancing: up-sampling, down-sampling and Synthetically generated samples. Here, we are using the third method to balance the class and it is implemented using the SMOTE algorithm. The class of fraud and not fraud is initially at the ratio of 5:95 and it is balanced to 50:50, in order to reduce the bias.

- Choose a minority class input data point.
- Find its k nearest neighbors.
- Choose one of these neighbors and place a synthetic point anywhere on the line joining the point under consideration and its chosen neighbor.

- Repeat the steps until data is balanced.



```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=123)

X_sm, y_sm = sm.fit_resample(X, y)

print(f'''Shape of X before SMOTE: {X.shape}
Shape of X after SMOTE: {X_sm.shape}''')

print('\nBalance of positive and negative classes (%):')
y_sm.value_counts(normalize=True) * 100
```

Fig. 5. SMOTE Implementation

IV. ML Algorithms Implementations

Before moving to the model implementation, one common feature in all the model development was hyperparameter tuning.

Generally , we only have ambiguous idea of the ideal hyperparameters and the easiest way to restrict our search is to assess a wide range of values for each hyperparameter.

There are two ways to search our ideal hyperparameters, one being the grid search and other being random search.

The main distinction between random search and grid search is that with random search, not all of the values are examined, and the values that are tested are chosen at random. For example, if the distribution has 500 values and we provide n iter=50, random search will randomly select 50 of them to evaluate.

Induced randomness has to reduce the correlation, at the same time maintaining strength To enhance accuracy, thus the method is of searching in grid is preferred in all our models.

A. *Logistic Regression*

Logistic Regression is one of the simple yet effective supervised algorithms for classification. We used regularization parameter as one of the hyper parameter along with 'max_iter' and 'solver'.

'C': Regularization parameter. Higher the number, lesser the impact of regularization.

'max_iter': Maximum number of iterations taken for the solvers to converge.

'solver': Algorithm to use in the optimization problem. 'liblinear' was chosen as the best hyperparameter among other solvers such as 'sag' and 'saga'.

We have used the following parameter values tune our RF model and its confusion matrix is presented below.

'C': 2, 'max_iter': 150, 'solver': 'liblinear', 'random_state': 42

Metrics	Score
Accuracy	0.62
Recall score	0.62
Precision score	0.61
F1 Score	0.62

The accuracy from the model was lesser compared to the other models. The reason is that that the there is this

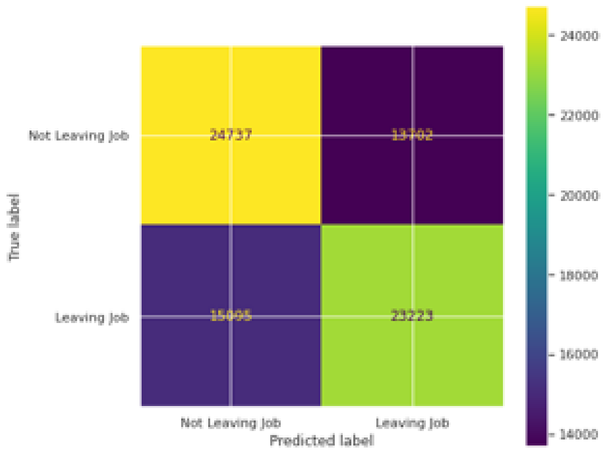


Fig. 6. Logistic Regression model

underlying assumption of linearity between dependent and independent variables. The relationship between the dependent and independent variables is weak in this case. Hence the data needs stronger model to capture this trend.

B. *K-Nearest Neighbours*

k-nearest neighbours works on an assumption that data points belonging to the same labels exist in proximity. KNN works by calculating the distances between a query and all of the instances in the data, picking the K closest examples to the query, and then voting for the most frequent label (in the case of classification).

We are choosing the following hyper parameters for k-NN

'n_neighbors': k nearest points to be specified 'p': the distance metric. '1' refers to the Manhattan distance and '2' refers to Euclidian distance.

By tuning the hyperparameters we got the following results for the hyperparameters for the best model result. The accuracy metrics are shown below 'n$_neighbors'$: 4,' p' : 1

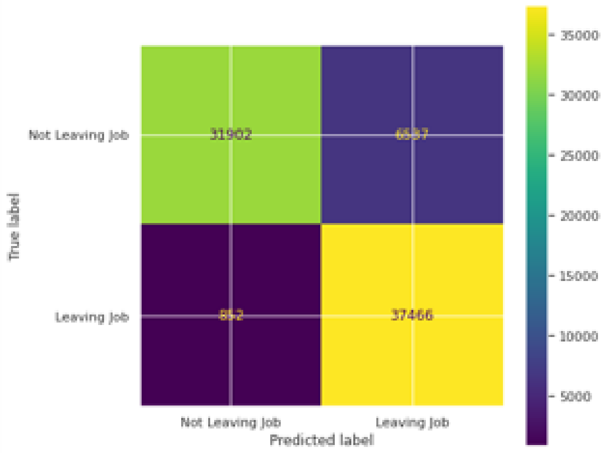


Fig. 7. K-Nearest Neighbours model

Metrics	Score
Accuracy	0.90
Recall score	0.91
Precision score	0.98
F1 Score	0.91

It of no surprise that K-NN performs well, because in oversampling we use K-NN technique to generate synthetic data points.So, even though this model performs well, K-NN can’t be taken as an optimal model among other models.

C. Decision Trees

A decision tree is a probability tree that enables to decide on the process. A decision tree is a tree-like model that acts as a decision support tool, visually displaying decisions and their potential outcomes, consequences, and costs. The branches can quickly be evaluated by calculating the entropy and compared to select the best courses of action.

There are two kinds of scoring to evaluate the impurity in a decision tree, ID3 and the Gini index algorithm. In this project, we use the Id3 algorithm.

The following hyperparameters were used to train the decision tree and obtain the best performing model.

’max_depth’: The maximum depth of the tree. ’min_samples_leaf’: The minimum number of samples required to be at a leaf node. ’max_features’: The number of features to consider when looking for the best split

The following hyperparameters were tuned and we got the below results for the best model. The results of this model are displayed below:

’max_depth’: 8,’max_features’: 5, ’min_samples_leaf’: 10

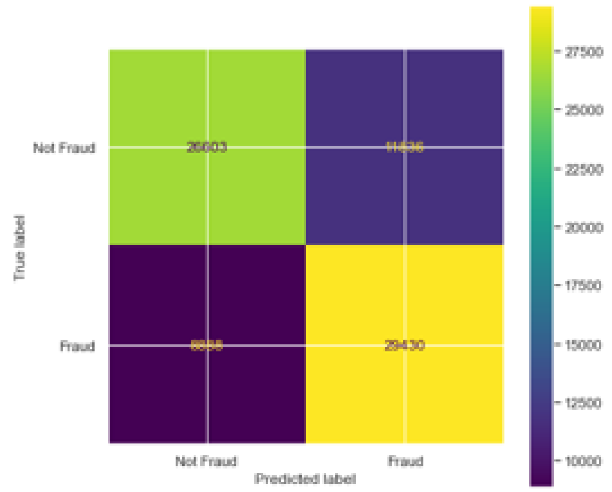


Fig. 8. Decision Tree model

Metrics	Score
Accuracy	0.73
Recall score	0.73
Precision score	0.77
F1 Score	0.74

D. Random Forest

Random Forest is a powerful ensemble algorithm we use in our classification. RF in general is used to join output of multiple Decision tress to give out one single output , i.e. RF Is a strong classifier which consist of many weak classifiers. Random forest has the advantage of faster prediction, can handle unbalanced data and is robust to outliers.

Random forest improves on bagging because it decorrelates the trees with the introduction of splitting on a random subset of features.

We have tried tuning the hyper parameters on parameters like ’max_depth, ’max_features’, ’min_samples_leaf’, ’n_estimators’.

max_depth: The maximum depth of each tree is specified by the max depth argument. When max depth is set to None, each tree will extend until every leaf is pure.

max_features: This is the size of the random subsets of features to consider when splitting a node.

n_estimators: The number of trees you wish to create before calculating maximum voting or prediction averages. A greater number of trees improves performance while increasing the run-time.

We have used the following parameter values tune our RF model and its confusion matrix is presented below.

’max_depth’: 7, ’max_features’: 4, ’min_samples$_{l}eaf'$: 10,$'n\_estimators'$: 50

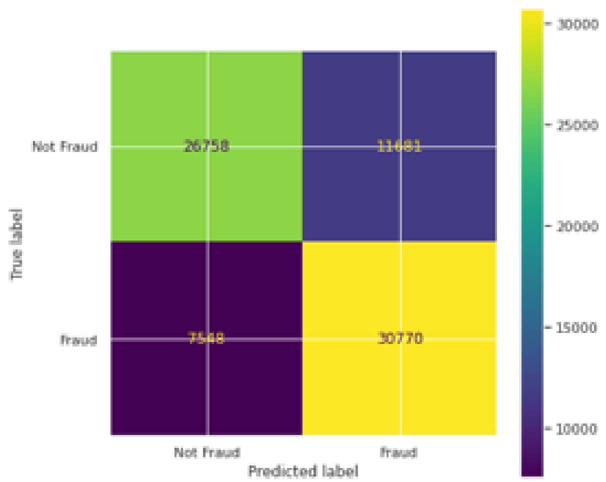


Fig. 9. Random Forest model

Metrics	Score
Accuracy	0.75
Recall score	0.75
Precision score	0.80
F1 Score	0.76

This took a lot of time, which is one of the disadvantage of this model.

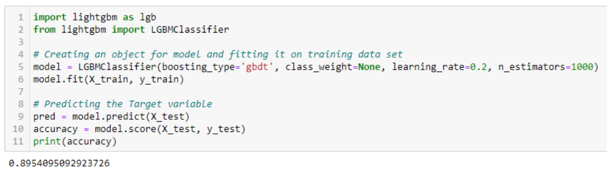
E. Light Gradient Boosted Machine

Since it is a classification problem, we can use supervised machine learning algorithms such as decision tree, Support

Vector machine, Neural Network and other related algorithms.

It is known that gradient boosted algorithms perform better than random forests for the following reasons.

Boosting is based on weak learners (high bias, low variance). In terms of decision trees, weak learners are shallow trees, sometimes even as small as decision stumps (trees with two leaves). Boosting reduces error mainly by reducing bias (and also to some extent variance, by aggregating the output from many models).



```
import lightgbm as lgb
from lightgbm import LGBMClassifier

# Creating an object for model and fitting it on training data set
model = LGBMClassifier(boosting_type='gbdt', class_weight=None, learning_rate=0.2, n_estimators=1000)
model.fit(X_train, y_train)

# Predicting the Target variable
pred = model.predict(X_test)
accuracy = model.score(X_test, y_test)
print(accuracy)
```

```
0.8954095092923726
```

Fig. 10. LGBM Implementation

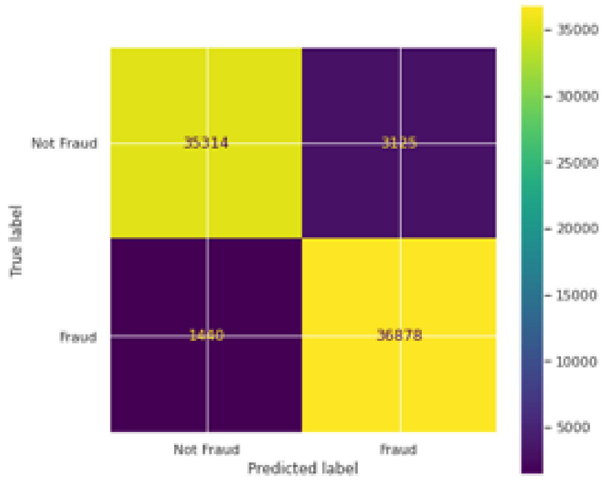


Fig. 11. LGBM confusion matrix

Light GBM has better accuracy than any other boosting algorithm. It uses a leaf-wise split strategy rather than a level-wise split approach to build significantly more complicated trees, which is the major element in achieving higher accuracy. It can, however, lead to over fitting, which can be avoided by increasing the max depth parameter[3].

Metrics	Score
Accuracy	0.94
Recall score	0.94
Precision score	0.96
F1 Score	0.94

The model is implemented using the following hyper parameters.

- class weight=None
- learning rate=0.2
- n estimators=1000

V. Comparison

While comparing the model, we can see that the Light GBM is way better than the other in terms of accuracy and other metrics. KNN has a precision value of 98 percent, and that is because we use SMOTE to create synthetic data points, which uses KNN as the base algorithm to develop synthetic data points. So, it is obvious to see a higher precision in the KNN algorithm when we use it to classify data points, but we can't take this as an optimal solution or a benchmark score.

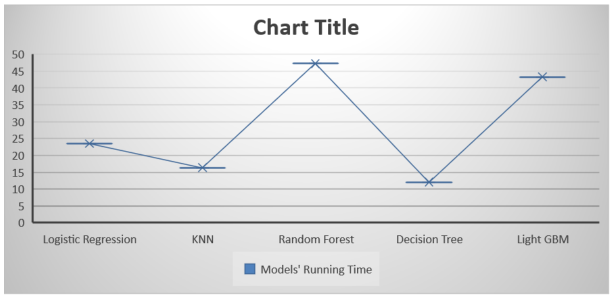


Fig. 12. Time Analysis

Random forest is another complex model that we used in our classification. Still, it took too much time to run for the given hyperparameters, and we could not attain the benchmark accuracy from this model. One possible reason for this could be model complexity and diverse data points.

Once again, considering the range of hyperparameters, Light GBM running time is better than all the other models. As mentioned earlier, Light GBM, with all its speed, took nearly 43 minutes to run through the high range of n_estimaters and a lower range learning rate.

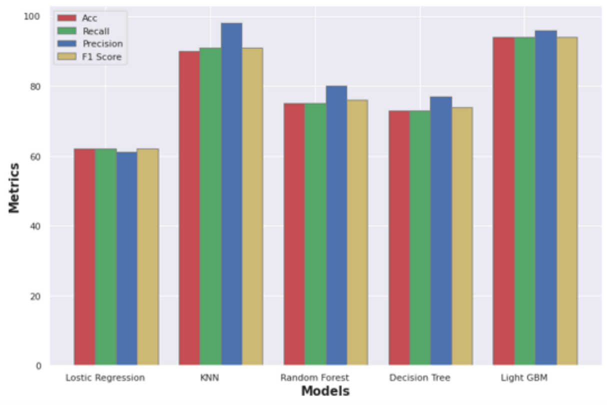


Fig. 13. Model Comparison

With all these mentioned, we are taking the Light GBM as our final model, and we were also able to reach an accuracy of 94 percent, which is a benchmark score, considering the imbalance of target labels. This is because even without a machine learning model, if this fraudulent data is manually given non-fraud labels to all records, it still reaches an accuracy of 94 percent, considering the counts of the target labels. So, after generating synthetic data points and running a machine learning model on meaningful data, we arrived at this solution through our model.

VI. Conclusion

In this paper ,using light GBM an accuracy of 94 percent and a precision score of 96 percent which is very close to the actual imbalance in the target variable. So, our project was able to make a real impact in detecting fraudulent billing in this case. In this project we were able to effectually create synthetic data points in the minority class which could also help us in detecting frauds. Extraction of data features for detecting aberrant power theft behavior and gas usage was shown to be effective in the model-by-model analysis. It is feasible to use the SMOTE balancing method when the data is similar to the Tunisian Electricity and gas consumption board data set. The methodological basis analysis has the potential to be a valuable tool for online monitoring and data analysis of large-scale power usage. In future, we can carry forward this model to similar organization like PSEG – Electricity and Gas utility company in and around New York and New Jersey. Finally, our team is submitting this as an online submission in the ongoing competition in Zindi.africa.

References

[1] Dornadula, Vaishnavi Nath; Geetha, S (2019). Credit Card Fraud Detection using Machine Learning Algorithms. Procedia Computer Science, 165(), 631–641. doi:10.1016/j.procs.2020.01.057

[2] Melo-Acosta, German E.; Duitama-Munoz, Freddy; Arias-Londono, Julian D. (2017). [IEEE 2017 IEEE Colombian Conference on Communications and Computing (COLCOM) - Cartagena, Colombia (2017.8.16-2017.8.18)] 2017 IEEE Colombian Conference on Communications and Computing (COLCOM) - Fraud detection in big data using supervised and semi-supervised learning techniques. , (), 1–6. doi:10.1109/ColComCon.2017.8088206

[3] An Optimized LightGBM Model for Fraud Detection, K Huang 2020 J. Phys.: Conf. Ser. 1651 012111