AI511 - Machine Learning
Project Report

Project Name: It's a Fraud!

Team Members:

Anick Bhattacharya (MT2022168)

Shubham Mondal (MT2022169)

Credit Card Fraud Detection

Credit card fraud is the act of using another person's credit card to make purchases or request cash advances without the cardholder's knowledge or consent. These criminals may obtain the card itself through physical theft, though increasingly fraudsters are leveraging digital means to steal the credit card number and accompanying personal information to make illicit transactions.

There is some overlap between identity theft and credit card theft. In fact, credit card theft is one of the most common forms of identity theft. In such cases, a fraudster uses an individual's personal information, which is often stolen as part of a cyberattack or data breach, to open a new account that the victim does not know about. This activity is considered both identity fraud and credit card fraud.

Problem Statement:

We are to build models for predicting whether an online transaction is fraudulent or not.

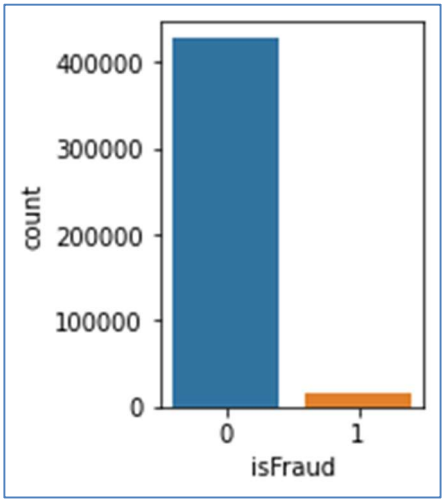
Input

train.csv - the training set (429525 datapoints 433 features and target, i.e. fraud or not)
test.csv - the test set (1147635 data points and 433 features)

We are to produce output as label
sample_submission.csv - a sample submission file in the correct format

EDA & DATA-PREPROCESSING

1. Handling the Imbalanced Data



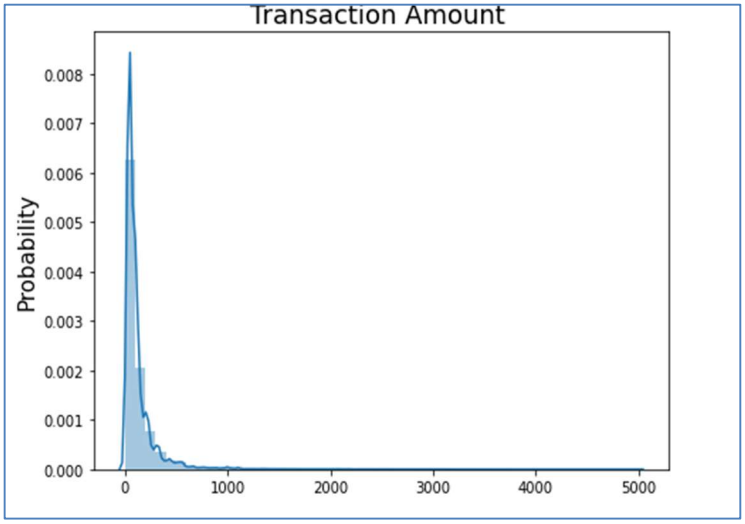
Number of Fraud transaction in the whole dataset are extremely low compared to Legit transactions. Balancing of the dataset was done using imblearn package.

```
22  from imblearn.under_sampling import RandomUnderSampler
23
24  rus=RandomUnderSampler(random_state=0)
25  X_sm,y_sm=rus.fit_resample(x,y)
26
```

We performed under sampling since training was very fast compared to oversampling and results were almost similar.

2. Outlier Removal

We performed outlier removal on Transaction amount where Transaction amount was greater than 800(~98%) and lesser than 10(0.01%).




```
Transaction Amounts Quantiles:
0.010       9.242188
0.025      14.500000
0.100      25.953125
0.250      43.312500
0.500      68.750000
0.750     125.000000
0.900     275.250000
0.975     648.262500
0.990    1104.000000
Name: TransactionAmt, dtype: float64
```

3. Handling missing Values

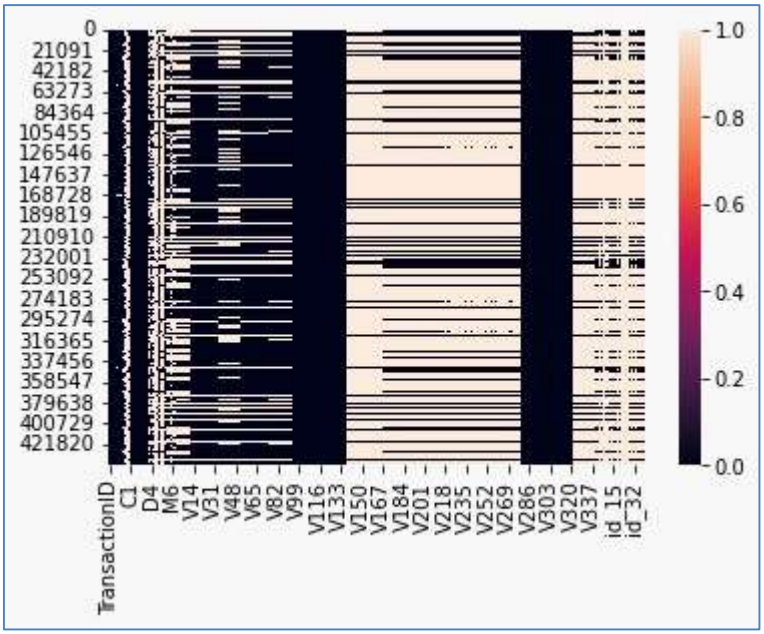
There were many columns with as high missing values as 99%. Columns with missing values more than 85 %, were dropped directly and the rest. After some pre-processing and imputing we further dropped columns which had more than 70% missing values. And the rest were imputed as below:

a) Handling missing values from categorical columns

Categorical columns were imputed using simple imputer and were simply imputed with "NA" as that would prevent any bias on any specific value and also extremely fast to implement.

b) Handling missing values from numerical columns

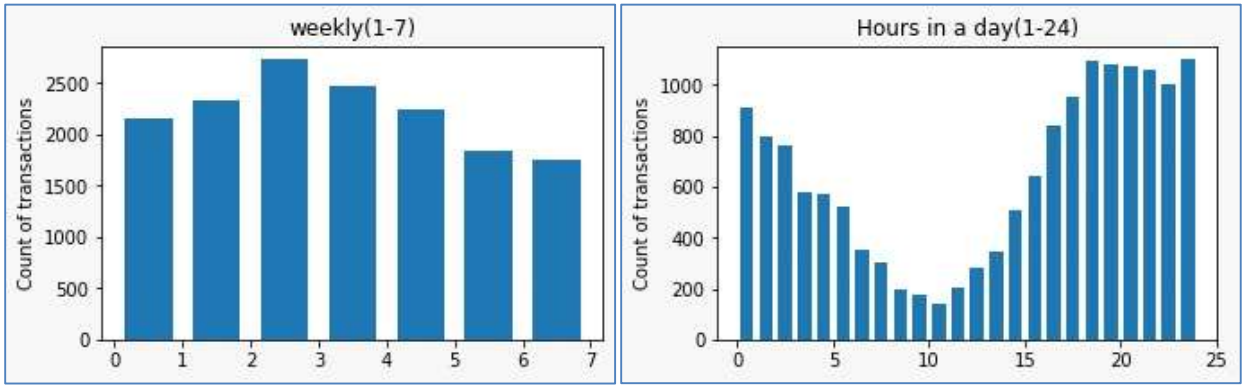
Missing Values from numerical columns were handled by imputing with "min – 1" as other strategies like median were adding bias to the model and thus degrading the model accuracy.



4. Feature Engineering

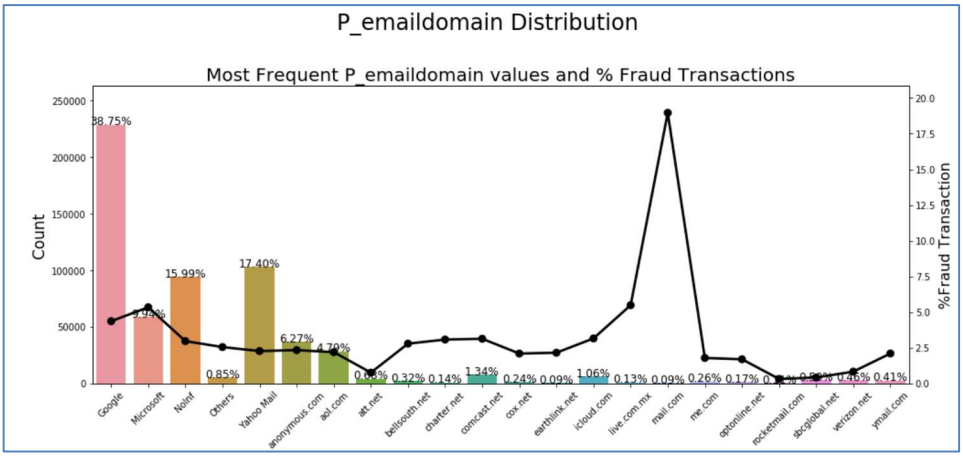
a) From Transaction Delta(Date-Time)

We tried to give meaning to the transaction delta by grouping them in to time slots(weekly, by day, and by hours).We were able to infer from the distribution that most frauds were occurring during peak usage of the cards(evening- to late night)



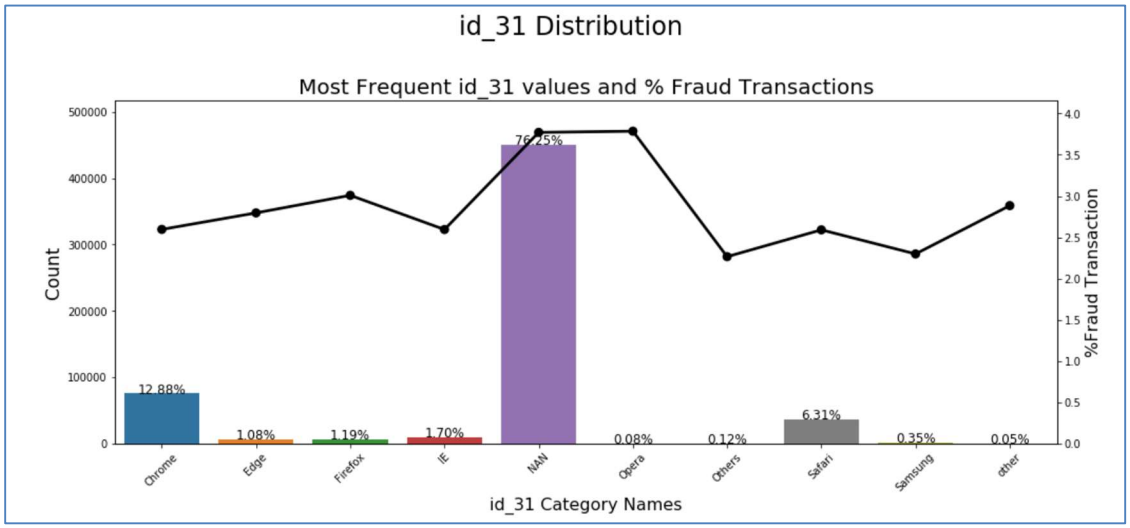
b) Categorizing P_emaildomain and R_emaildomain

We were able to categorize different P_emaildomain under a different umberellas.
Example: ['hotmail.com','outlook.com','msn.com', …] can be categorized under “Microsoft”.



c) Categorizing and binning id columns

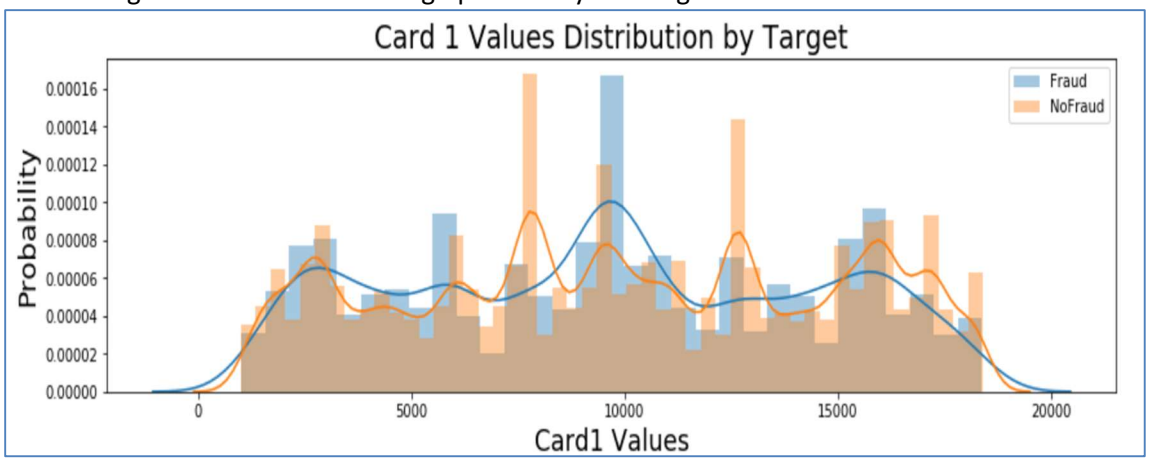
id_30 and id_31 were device specific(os-version, browser/device-model) id columns and we were to put under some bins to decrease the unnecessary variance in the data.



d) Finding Deviation from usual transaction amount by card1

We wanted to get the deviation of transaction amount of each card from their usual(mean) transaction amount as

sudden large transactions have a high probability of being fraudulent.



e) Standard Feature Engineering on Transaction Amount

Standard feature engineering on the transaction amount was done and the overall deviation from the mean and std deviation was added as a new feature to the dataset.

f) Principal Component Analysis on V columns

We applied PCA on the V columns and reduced the V feature to 30 principal components (15, 30,45, 50 were tried) on both train and test.

CLASSIFICATION MODELS

Classification Models Tried

Here is a list of all the classification models which we tried to get the best possible accuracy. Theyhave been ranked according to the accuracy of the results which they produced.

1. XGBoost – Used GPU- Fast
2. Random Forest Classifier – Slow
3. SVM – Slow
4. KNN – Training is fast but Prediction is slow
5. Logistic Regression CV – Fast
6. Ada Boost - Fast
7. Decision Tree – Slow
8. Neural Network using Keras - Slow

Model Explanations

1. XGBoost

```
rus=RandomUnderSampler(random_state=0)
X_sm,y_sm=rus.fit_resample(x,y)

import xgboost as xgb

my_model = xgb.XGBClassifier()

params={"learning_rate": [0.1,0.15,0.2,0.25,0.3,0.35,0.5,0.7],
        "max_depth": [3,2,5,6,8,10,12,15,20,30,50],
        "min_child_weight": [1,3,5,7],
        "gamma": [0.0,0.1,0.2,0.3],
        "colsample_bytree": [0.3,0.5,0.7,0.4]
       }

clf=RandomizedSearchCV(my_model,param_distributions=params,n_iter=5,scoring='f1_micro',n_jobs=-1,cv=5,verbose=3)

clf.fit(X_sm,y_sm)

print(clf.best_params_)

my_model = xgb.XGBClassifier(min_child_weight=3,max_depth=20,learning_rate=0.1,gamma=0.3,colsample_bytree=0.4)

my_model.fit(X_sm,y_sm)

ypred=my_model.predict(test)

pd.DataFrame(ypred).to_csv('xg_pred.csv')

=======
```

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient**,** flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of

examples.It implements machine learning algorithms under the Gradient Boosting framework.

XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

Hyperparameters Tuned:

Learning Rate: The **learning_rate** parameter can be set to control the weighting of new trees added to the model.

Max Depth:

- Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 indicates no limit on depth. Beware that XGBoost aggressively consumes memory when training a deep tree. exact tree method requires non-zero value.
- range: [0,∞]

Minimum Child Weight:

- Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression task, this simply corresponds to minimum number of instances needed to be in each node. The larger min_child_weight is, the more conservative the algorithm will be.
- range: [0,∞]

Gamma:

- Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be.
- range: [0,∞]

Colsample by Tree:

- This is a family of parameters for subsampling of columns.
- All colsample_by parameters have a range of (0, 1], the default value of 1, and specify the fraction of columns to be subsampled.
- Colsample_by_tree is the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.

Kaggle Final Score: 87.4

2. Random Forest

```
from imblearn.under_sampling import RandomUnderSampler
rus=RandomUnderSampler(random_state=0)
X_sm,y_sm=rus.fit_resample(x,y)
del x
del y
gc.collect()
params={"n_estimators": [100,200,300],
        "min_samples_split" : [2],
        "max_features" : ["sqrt","log",30],
        "n_jobs" : [-1],
        "max_samples" : [0.1,0.5,1.0],
        'verbose' : [1]
        }
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

grid_search = GridSearchCV(estimator = RandomForestClassifier(),param_grid= params ,  n_jobs = 1, cv =5, scoring='f1_micro')
grid_result = grid_search.fit(X_sm,y_sm.values.ravel())
print(grid_result.best_params_)

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
rfc = RandomForestClassifier(min_samples_split=2,max_f Loading... qrt',n_estimators=200, verbose = 1, n_jobs=-1,max_samples=1.0)
np.mean(cross_val_score(rfc,X_sm,y_sm,scoring="f1_micro", cv =5))
rfc.fit(X_sm, y_sm)
del X_sm
del y_sm
gc.collect()
test=pd.read_csv("test_mod.csv")
ypred = rfc.predict(test)
pd.DataFrame(ypred).to_csv("random_pred5.csv")
```

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

Hyperparameters Tuned:

n_estimators: The number of trees in the forest.

Final used value : 200

min_samples_split: The minimum number of samples required to split an internal node:

- If int, then consider min_samples_split as the minimum number.
- If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

Final used value : 2

max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or

until all leaves contain less than min_samples_split samples.

Final used value : None

max_features : The number of features to consider when looking for the best split:

Final used value : sqrt

Kaggle Final Score: 86.5

3: SVM

```
params={
        "kernel": ['rbf'],
        "gamma": [0.01],
        "C": [2,3,5]
       }

clf=RandomizedSearchCV(sv,param_distributions=params,scoring='f1_micro',n_jobs=-1,cv=2)
clf.fit(X_train,Y_train)
print(clf.best_params_)

print(clf.best_score_)

sv = svm.SVC(kernel="rbf",gamma=1,C=0.1)
sv.fit(X_train,Y_train)

f1_score(Y_train,sv.predict(X_train))
f1_score(Y_test,sv.predict(X_test))

sv = svm.SVC(kernel="rbf",gamma=0.1,C=1)
sv.fit(X_train,Y_train)
f1_score(Y_train,sv.predict(X_train))
f1_score(Y_test,sv.predict(X_test))

sv = svm.SVC(kernel="rbf",gamma=1,C=100)
sv.fit(X_train,Y_train)
f1_score(Y_train,sv.predict(X_train))
f1_score(Y_test,sv.predict(X_test))

sv = svm.SVC(kernel="rbf",gamma=1,C=10)
sv.fit(X_train,Y_train)
f1_score(Y_train,sv.predict(X_train))
f1_score(Y_test,sv.predict(X_test))

sv = svm.SVC(kernel="rbf",gamma=0.01,C=3)
sv.fit(X_train,Y_train)
f1_score(Y_train,sv.predict(X_train))

f1_score(Y_test,sv.predict(X_test))
ypred=sv.predict(test)
pd.DataFrame(ypred).to_csv('svm_pred.csv')
```

Hyperparameters tuned:

Kernel : RBF (Radial Basis Function)

The radial kernel is $K(x_i,x_j)=e_{\gamma \cdot x_i x_j} K(xi,xj)=e\gamma \cdot xi'xj$.

This kernel is the most used and most successful kernel, due to the flexibility of separating observations with this method. Additionally to the cost parameter CC, the hyperparameter γγ has to be tuned. In the paper A practical guide to Support Vector Classifier they recommend to search CC in the range $[2^{-5};2^{15}]$[2−5;215] and γγ in the range $[2^{-15};2^{3}]$[2−15;23].

Gamma : As the gamma decreases, the regions separating different classes get more generalized. Very large gamma values result in too specific class regions (overfitting).

C : C parameter adds a penalty for each misclassified data point. If c is small, the penalty for misclassified points is low so a decision boundary with a large margin is chosen at the expense of a greater number of misclassifications.

4: KNN

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
#grid_search
params={"weights": ["uniform", "distance"],
        "n_neighbors": [4, 6, 8, 10],
         "algorithm ":['auto']
        }
grid_search = RandomizedSearchCV(KNeighborsClassifier(), params, scoring = 'f1_micro', n_jobs = -1)
grid_result = grid_search.fit(X_sm,y_sm)
#cross validation
knn = KNeighborsClassifier(weights = "distance", n_neighbors = 6, algorithm = 'auto')
np.mean(cross_val_score(knn,X_sm,y_sm,scoring="f1_micro", cv =3))
# fitting the model and getting prediction
ypred = knn.fit(X_sm, y_sm)
ypred = knn.predict(test)
pd.DataFrame(ypred).to_csv("knn_ht_pred.csv")
```

Hyperparameters tuned:

n_neighbors: Number of neighbors to use by default for kneighbors queries.

Final used value : 6

Weights: Weight function used in prediction.

Final used value : distance

Algorithm : Algorithm used to compute the nearest neighbors:

Final used value : auto

Kaggle Final Score: 81.2

5. Adaboost:

```
params={"n_estimators": [100,200, 300],
        "learning_rate": [0.05, 0.2, 1.0],
        "algorithm": ["SAMME","SAMME.R"],
        }
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score

grid_search = RandomizedSearchCV(AdaBoostClassi ravel: Any ns, n_jobs = -1, scoring='f1_micro', verbose =1)
grid_result = grid_search.fit(X_sm,y_sm.values.ravel())
print(grid_result.best_params_)

adb = AdaBoostClassifier(n_estimators = 300, learning_rate = 0.2, algorithm= 'SAMME.R')
np.mean(cross_val_score(adb,X_sm,y_sm,scoring="f1_micro", cv =2))
adb.fit(X_sm, y_sm);
del X_sm
del y_sm
gc.collect()
test=pd.read_csv("test_mod.csv")
ypred = adb.predict(test)
pd.DataFrame(ypred).to_csv('ada_pred2.csv')
```

AdaBoost, short for Adaptive Boosting, is a statistical classification meta-algorithm formulated by Yoav Freund and Robert Schapire in 1995, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. Usually, AdaBoost is presented for binary classification, although it can be generalized to multiple classes or bounded intervals on the real line.

Hyperparameters Tuned :

n_estimators: The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early. Values must be in the range [1, inf).
Final Used Value: 300

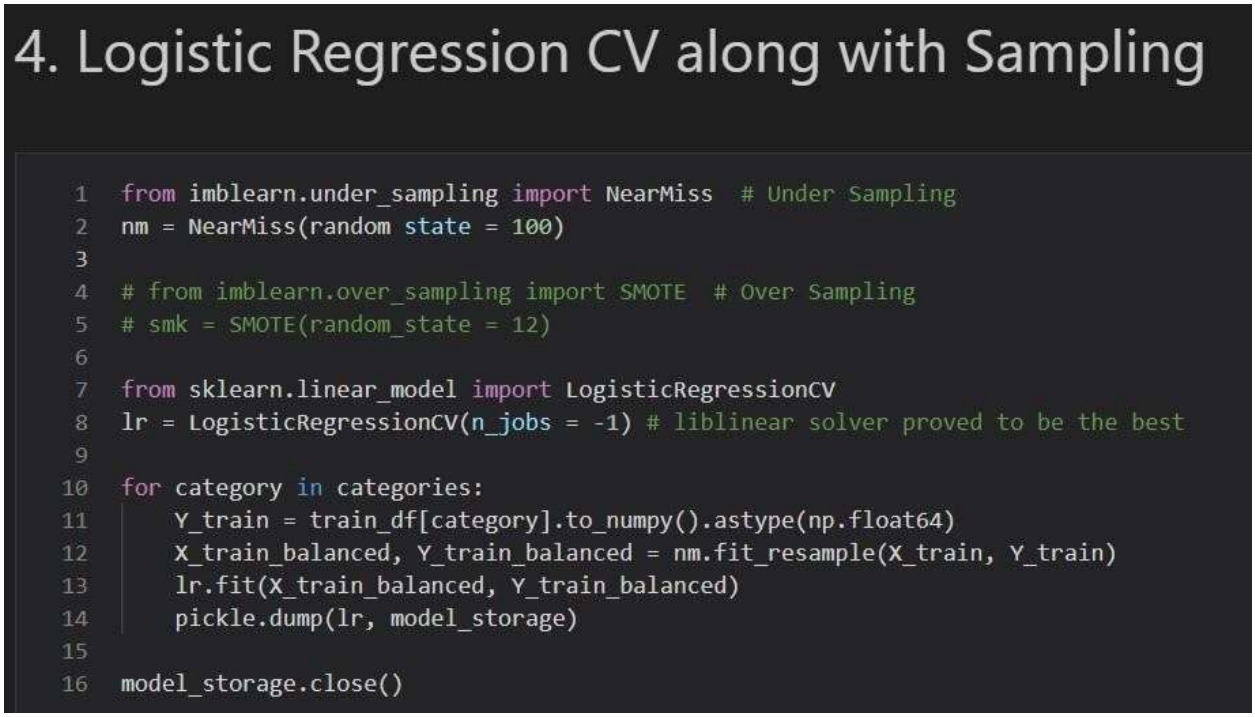
learning_rate: Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the learning_rate and n_estimators parameters. Values must be in the range (0.0, inf).
Final used value : 0.2

Algorithm: If 'SAMME.R' then use the SAMME.R real boosting algorithm. estimator must support calculation of class probabilities. If 'SAMME' then use the SAMME discrete boosting algorithm. The SAMME.R algorithm typically converges faster than SAMME, achieving a lower test error with fewer boosting iterations.
Final used value : SAMME.R

Kaggle Final Score: 78.4

6. Logistic Regression CV (along with trials of under sampling & over sampling)

Handling the imbalanced classes in the training data set using **under sampling**



- imblearn module was used for performing under sampling.
- Near - Miss Under Sampling: This algorithm looks at the class distribution and randomly eliminates samples from the larger class.

Hyperparameters of Logistic Regression CV model:

- solver = 'liblinear'

  This represents the algorithm for optimization. For larger datasets, 'sag' and 'saga' solvers are better.

- Tol = 0.0001

  Tolerance for stopping criteria.

- max_iter = 150

  Maximum number of iterations of the optimization algorithm.

- cv = cross-validation generator

  The default cross-validation generator used is Stratified K-Folds. If an integer is provided, then it is the number of folds used. The best hyperparameter is selected by the cross- validator StratifiedKFold, but it can be changed using the cv parameter.

- Stratified K Fold:

  The number of instances of each class for each experiment in the train and test data are takenin a methodical way.The folds are selected so that the mean response value is approximately equal in

all the folds. In the case of a binary classification, each fold contains roughly the same proportions of the two types of class labels i.e. 1/0 or YES/NO.

7. Decision Tree:

```
23
24  rus=RandomUnderSampler(random_state=0)
25  X_sm,y_sm=rus.fit_resample(x,y)
26
27  params={"max_depth": [2,3,5,20,50],
28          "min_samples_leaf": [5,10,20,50,100],
29          "criterion": ["gini","entropy"]
30          }
31
32  from sklearn import tree
33  dt = tree.DecisionTreeClassifier()
34  clf=GridSearchCV(estimator=dt,param_grid=params,scoring='f1_micro')
35
36  clf.fit(X_sm,y_sm)
37
38  print(clf.best_estimator_)
39
40  print(clf.best_score_)
41
42  dt=tree.DecisionTreeClassifier(criterion='entropy', max_depth=50, min_samples_leaf=50)
43
44  dt.fit(X_sm,y_sm)
45
46  ypred=dt.predict(test)
47
48  pd.DataFrame(ypred).to_csv('dt_pred.csv')
```

Hyperparameters

- Max_Depth:

  - The maximum depth of the tree. If this is not specified in the Decision Tree, the nodes will be expanded until all leaf nodes are pure or until all leaf nodes contain less than min_samples_split.

- Default = None
- Input options → integer

- Min_Sample_Leaf:

  - The minimum samples required to be at a leaf node. Therefore, a split can only happen if it leaves at least the min_samples_leaf in both of the resulting nodes.
  - Default = 1
  - Input options → integer or float (if float, then min_samples_leaf is fraction)

- Criterion:
  How to measure the quality of a split in a decision tree. You can input “gini” for Gini Impurity or “entropy” for information gain.

8. Neural Network using Keras

```
epochs = 100
model = Sequential()
model.add(Conv1D(32, 2, activation='relu', input_shape = X_train[0].shape))
model.add(BatchNormalization())
model.add(MaxPool1D(2))
model.add(Dropout(0.2))

model.add(Conv1D(64, 2, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool1D(2))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=Adam(lr=0.0001), loss = 'binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verbose=1)
```

Layers:

1. Dense layer with 64 neurons and ReLU activation function
2. Batch Normalisation
3. Dropout to control overfitting
4. Dense layer with 64 neurons and ReLU activation function
5. Dropout to control overfitting
6. Sigmoid layer (Output Layer)

RESULT SUMMARY

PYTHON LIBRARIES

Numpy
Pandas
Sklearn
Imblearn
xgboost

Documentations

RESOURCES

- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeClassifier.html

Articles

- https://www.geeksforgeeks.org/ml-voting-classifier-using-sklearn/
- https://www.geeksforgeeks.org/random-forest-regression-in-python/
- https://en.wikipedia.org/wiki/Tf%E2%80%93idf
- https://towardsdatascience.com/besides-word-embedding-why-you-need-to-know-character-embedding-6096a34a3b10
- https://towardsdatascience.com/stemming-vs-lemmatization-2daddabcb221#:~:text=Stemming%20and%20Lemmatization%20both%20generate,words%20which%20makes%20it%20faster.

Video lectures:

- **https://www.youtube.com/watch?v=4Im0CT43QxY&t=488s**
- **https://www.youtube.com/watch?v=9HomdnM12o4&t=691s**
- **https://www.youtube.com/watch?v=HdlDYng8g9s**
-