EDA:

The training data has 307511 observations (each one a separate loan) and 122 features including the TARGET (the label we want to predict).

-For EDA first we examine the distribution of the Target Column : 0 -282686 and 1 - 24825,where 0 for the loan was repaid on time and 1 indicating the [4]

client had payment difficulties. There are far more loans that were repaid on time than loans that were not repaid.

Below are considerations to tackle this imbalance issue:

-We analyse (later) various imbalance techniques with respect to each model and see which technique works for which model.

-We use only the f1\_score and recall in order to determine the compare the different models and find the best working model. This is because if we classify a defaulter as non defaulter(i.e.1 as 0), it is so much worse than classifying a non defaulter as defaulter(i.e.0 as 1). i.e. we are more interested in having more true positives but at the same time have one of the better f1\_scores.

- Next we look at the Anomalies [4]

- DAYS\_EMPLOYED- The maximum value - 365243 is about 1000 years. So we replaced all anomalous values with nan.

- CODE\_GENDER- Replace the value XNA with nan since M and F are the only possible values

- DAYS\_LAST\_PHONE\_CHANGE-Replace 0 with nan since 0 is not a possible value for this column

- Next we look at the number of unique entries in each of the object (categorical) columns. The result is-

NAME CONTRACT TYPE -2

CODE GENDER -3

FLAG OWN CAR -2

FLAG OWN REALTY -2

NAME TYPE SUITE -7

NAME INCOME TYPE -8

NAME EDUCATION TYPE -5

NAME FAMILY STATUS -6

NAME HOUSING TYPE -6

OCCUPATION TYPE -18

WEEKDAY APPR PROCESS START -7

ORGANIZATION TYPE -58

FONDKAPREMONT MODE -4

HOUSETYPE MODE -3

WALLSMATERIAL MODE -7

EMERGENCYSTATE MODE -2

-Next we encode the Categorical Variables:

1) We have used Label Encoding for any categorical variables with only 2 categories

2) One-Hot Encoding for any categorical variables with more than 2 categories. In total - 3 columns were label encoded.

-Next we look at the number and percentage of missing values in each column There are 67 columns that have missing values.

We use mean mode approach to fix missing values issue:

every missing value numerical column with mean and every

missing value categorical column with mode. We create a new column for every such column that holds a 1 for every missing value in the existing column.

Below are the models implemented:

Logistic Regression

Random Forest

Decision Tree

K Nearest Neighbors

XGBoost

LightGBM

Performance issues:

There are two key factors that influence the performance of the models:

-The data is extremely unbalanced. Class 0 data is more than 10 times Class 1 in the training set:

({0: 226132, 1: 19876})

-The strategy applied to handle the missing values

BASE MODELS:

The base model of both Logistic and Random forest perform very poorly on the imbalanced data. The EDA, best missing value strategy and feature engineering are applied to the input dataset. Below are the results obtained (We are using f1\_score to compare the performance):

K-Nearest Neighbor model:

Train Confusion Matrix:

[[225409 723]

[ 18724 1152]]

Test Confusion Matrix:

[[56177 377]

[ 4890 59]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.614000 0.057959 0.106000 0.887909 0.920950

Test 0.135321 0.011922 0.021913 0.541189 0.914362

Random Forest model:

Train Confusion Matrix:

[[226129 3]

[ 3626 16250]]

Test Confusion Matrix:

[[56457 97]

[ 4916 33]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 1.000000 0.817569 0.900000 0.999822 0.985248

Test 0.253846 0.006668 0.012995 0.627320 0.918492

Decision Tree model:

Train Confusion Matrix:

[[226132 0]

[ 0 19876]]

Test Confusion Matrix:

[[51488 5066]

[ 4091 858]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 1.000000 1.000000 1.000000 1.000000 1.000000

Test 0.144835 0.173368 0.157822 0.541895 0.851113

Logistic Regression model:

Train Confusion Matrix:

[[226126 6]

[ 19875 1]]

Test Confusion Matrix:

[[56552 2]

[ 4949 0]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.143 0.00005 0.0 0.635367 0.919186

Test 0.000 0.00000 0.0 0.640536 0.919500

Observations:

-Both Logistic and Random Forest are unable to classify the minority class in the test and in the train dataset.

-Decision Tree works the best among the baseline models with a recall rate of 0.04 and f1 score 0.07. This shows that the decision is not sensitive to unbalanced data. Also, it is possible that the samples in the minority class are good at representing the original population dataset. Hence, the decision tree is able to classify the test set with some accuracy. We can also see that it overfits the train data- all the classes are correctly classified in train. Conversely the random forest does not over the train data.

-The Decision tree is followed by the KNN model. The minority class in the test dataset is classified with some notable accuracy (f1\_score).

**IMPROVED MODELS:**

In order to improve the performance of the models, we first tuned the hyperparameters of each models on each missing value strategy. After finding the best missing value strategy. We proceed with handling the imbalance.

Subsection 2 )Handle the imbalance issue: \*sublevel1

When the samples between the two classes are not balanced then the model is more liable to learn about the majority class. This results in poorer classification of the minority class since there are not sufficient data available from the minority class. This behavior becomes more intense if the ratio of the majority to minority is very high – which is our case. There might be some models like decision tree that are exceptions to this scenario.

**Four approaches are applied to handle this issue:**

Oversampling**:**

In this method the data from the minority class is replicated so that a balance can be established between the two classes.

Undersampling:

In this method the data from the majority class is removed in order to balance the data.

Synthetic minority oversampling technique:

The over sampling of the minority dataset is done synthetically using an algorithm. Data from the minority class is not replicated. This prevent over fitting which is prevalent in oversampling.

Improve the cost function:

There are several approaches-[1] one which we are using is class weights. The class weights are added to the minority class so that the cost function accounts more for error in minority class.

To determine the best f1\_score for each noise we use the best grid model against the test data in step2.

**Noise levels for each imbalance technique:**

1. Sampling strategy- It is the ratio of the number of samples in minority class over the number of samples in the majority class. This parameter is used for OverSampling, UnderSampling and SMOTE.
2. Class weights- It specifies the weights to be given to the error from each class in the cost function. We set more weights on the minority class in order to make the model more sensitive to its errors on the minority class. This noise level is applicable for the cost function based approach.

**Experiment to find the best missing value strategy:**

**Approaches**

-Step1) Treat the anomalies.

-Step2) Perform the label encoding and one hot encoding on the categorical columns

-Step3) Find the columns containing missing values

-Step4) Apply the missing value strategy

- Step5) Split the data into train and test set

-Step6) Apply imbalance strategy on train set

-Step7) Perform a grid search for each model and find the best possible accuracy and the best hyper parameters.

-Step8) We use the hyper parameters found in Step7.

-Step8) Based on the results obtained we decide which method best works for which model.

**Below are the results and observations for each model:**

**DECISION TREE:**

**Oversampling: subsec level2**

F1\_score at different levels of noise (noise->f1\_score) :

(0.15, 0.145716),( 0.3,0.147995),(0.45, 0.142224),( 0.75, 0.144683), (0.9, 0.135496)

Best Sampling strategy--> 0.3

Y train after resampling Counter({0: 226132, 1: 67839})

Improved number of features-> 90

Best parameter on grid--> {'random\_state': 42, 'max\_features': 'auto', 'max\_depth': 50, 'criterion': 'gini'}

DECISION TREE model:

Train Confusion Matrix:

[[226130 2]

[ 3 67836]]

Test Confusion Matrix:

[[52080 4474]

[ 4196 753]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.99997 0.999956 0.999963 1.000000 0.999983

Test 0.14406 0.152152 0.147995 0.536525 0.859031

**SMOTE: sub section level2**

F1\_score at different levels of noise (noise->f1\_score) :

(0.15, 0.073491,(0.45, 0.083753),( 0.75, 0.083657), (0.9, 0.091882)

Sampling strategy--> 0.9

Y train after resampling Counter({0: 226132, 1: 203518})

Improved number of features-> 111

Best parameter on grid--> {'random\_state': 42, 'max\_features': 'auto', 'max\_depth': 20, 'criterion': 'gini'}

DECISION TREE model:

Train Confusion Matrix:

[[225182 950]

[ 14191 189327]]

Test Confusion Matrix:

[[54463 2091]

[ 4610 339]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.995007 0.930272 0.961551 0.990289 0.964760

Test 0.139506 0.068499 0.091882 0.589487 0.891046

**Under Sampling: \*subsec level2**

F1\_score at different levels of noise (noise->f1\_score) :

(0.15, 0.163427),(0.6, 0.194770),( 0.75, 0.193454), (0.9, 0.185412)

**Sampling strategy--> 0.6**

Y train after resampling Counter({0: 33126, 1: 19876})

Improved number of features-> 94

Best parameter on grid--> {'random\_state': 42, 'max\_features': 'auto', 'max\_depth': 20, 'criterion': 'gini'}

DECISION TREE model:

Train Confusion Matrix:

[[30749 2377]

[ 3446 16430]]

Test Confusion Matrix:

[[40102 16452]

[ 2640 2309]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.873611 0.826625 0.849469 0.963809 0.890136

Test 0.123074 0.466559 0.194770 0.583966 0.689576

**COST FUNCTION BASED APPROACH:**

Class weights--> {0: 0.1, 1: 0.9}

Improved number of features-> 95

Best parameter on grid--> {'random\_state': 42, 'max\_features': 'auto', 'max\_depth': 20, 'criterion': 'gini'}

DECISION TREE model:

Train Confusion Matrix:

[[187596 38536]

[ 1955 17921]]

Test Confusion Matrix:

[[44608 11946]

[ 2921 2028]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.317427 0.90164 0.469548 0.941764 0.835408

Test 0.145127 0.40978 0.214342 0.598636 0.758272

Observations:

-Oversampling-we observed that introduction of more noise i.e. replicating the minority class only leads to overfitting and the decision tree performs poorly. This is possibly the reason why the best result is found at such low sampling strategy (0.3).

-Interestingly the best f1 score obtained at 0.3 in oversampling is even worse than the BASE model. This is because the model trains on same samples of the minority class multiple times due to replication and eventually trains specific to the samples themselves and not on the class features.

-One of the better results are with undersampling and worst with SMOTE. Similar to RF it is unable to learn and benefit from the new synthetic samples. Although it benefits from the undersampling strategy where the balance is set at 0.6 and a recal of 0.46 and f1\_score 0.19 is achieved. The results at undersampling and oversampling are 10 times better than that at Smote.

- It works the best with the cost function based approach with f1\_score more than 0.2 and recall more than 0.4.

**Random Forest:subsec level1**

**OVERSAMPLING**:subsec level2

F1\_score at different levels of noise (noise->f1\_score) :

(0.15-> 0.039771), (0.45, 0.063275), (0.75, 0.067977), ( 0.9, 0.069861)

Best Sampling strategy--> 0.9

Y train after resampling Counter({0: 226132, 1: 203518})

Improved number of features-> 85

Best parameter on grid--> {'n\_estimators': 17, 'max\_features': 'auto', 'max\_depth': 90, 'bootstrap': True}

Random Forest model:

Train Confusion Matrix:

[[226131 1]

[ 0 203518]]

Test Confusion Matrix:

[[56226 328]

[ 4758 191]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.999995 1.000000 0.999998 1.000000 0.999998

Test 0.368015 0.038594 0.069861 0.682486 0.917305

**SMOTE: subsec l2**

F1\_score at different levels of noise (noise->f1\_score) :

(0.15-> 0.201405), (0.45, 0.021696), (0.75, 0.017765), ( 0.9, 0.015032)

Y train before resampling Counter({0: 226132, 1: 19876})

Sampling strategy--> 0.15

Y train after resampling Counter({0: 226132, 1: 33919})

Improved number of features-> 105

Best parameter on grid--> {'n\_estimators': 13, 'max\_features': 'auto', 'max\_depth': 90, 'bootstrap': True}

Random Forest model:

Train Confusion Matrix:

[[226129 3]

[ 1513 32406]]

Test Confusion Matrix:

[[56395 159]

[ 4859 90]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.999907 0.955394 0.977144 0.999970 0.99417

Test 0.361446 0.018185 0.034629 0.651394 0.91841

**UNDERSAMPLING**: subsectionlevel2

F1\_score at different levels of noise (noise->f1\_score) :

(0.15-> 0.034629), (0.6, 0.255757), (0.75, 0.246021), ( 0.9, 0.239339)

Sampling strategy--> 0.6

Y train after resampling Counter({0: 33126, 1: 19876})

Improved number of features-> 96

Best parameter on grid--> {'n\_estimators': 17, 'max\_features': 'auto', 'max\_depth': 90, 'bootstrap': True}

Random Forest model:

Train Confusion Matrix:

[[33099 27]

[ 148 19728]]

Test Confusion Matrix:

[[46656 9898]

[ 2772 2177]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.998633 0.992554 0.995584 0.999957 0.996698

Test 0.180290 0.439887 0.255757 0.700518 0.793994

Sampling strategy--> 0.9

Y train after resampling Counter({0: 22084, 1: 19876})

Improved number of features-> 95

Best parameter on grid--> {'n\_estimators': 17, 'max\_features': 'auto', 'max\_depth': 90, 'bootstrap': True}

Random Forest model:

Train Confusion Matrix:

[[22050 34]

[ 68 19808]]

Test Confusion Matrix:

[[39331 17223]

[ 1935 3014]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.998286 0.996579 0.997432 0.999962 0.997569

Test 0.148935 0.609012 0.239339 0.705175 0.688503

COST FUNCTION BASED APPROACH:

Class weights--> {0: 0.6, 1: 0.4}

Improved number of features-> 95

Best parameter on grid--> {'n\_estimators': 5, 'max\_features': 'auto', 'max\_depth': 90, 'bootstrap': True}

Random Forest model:

Train Confusion Matrix:

[[225899 233]

[ 3616 16260]]

Test Confusion Matrix:

[[55684 870]

[ 4652 297]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.985873 0.818072 0.894168 0.993990 0.984354

Test 0.254499 0.060012 0.097122 0.607989 0.910216

Observation:

-Oversampling:

The performance of Random Forest increases with more samples from the minority class. It has much less tendency to overfit when compared to the decision tree. This is due to the bootstrapping which is done as part of random forest that involves random resampling of data with replacement.

-We also observe that the number of trees (estimators) are very low. This is possibly due to the data being imbalanced it tends to converge to a decision tree. i.e. there are not enough samples from the minority class in the trees inside the random forest.

- The highest performance with oversampling is less than the f1\_score without oversampling.

-Smote:

The performance with Smote is the lowest, the random forest is unable to learn the new samples added since they are different from the actual dataset.

-Undersampling:

RF performs the best with undersampling with f1\_score reaching peak at 0.26 at 0.6 sampling strategy and recall of 0.6 at 0.9 sampling strategy. This shows how sensitive RF is with imbalanced data. With balanced data its f1\_score is 10 times that before resampling

* Cost function based approach is not that helpful with RF.

**Logistic Regression:**

**OVER SAMPLING:**

F1\_score at different levels of noise (noise->f1\_score):

(0.15, 0.101225),( 0.45, 0.291804),(0.6, 0.293157),(0.75, 0.283418),(0.9, 0.266181)

Recall score at different levels of noise (noise->recall):

(0.15, 0.057587 ),( 0.45, 0.358254),(0.6, 0.478683),(0.75,0.577288),(0.9,0.642756)

Sampling strategy--> 0.9

Y train after resampling Counter({0: 226132, 1: 203518})

Improved number of features-> 86

Best parameter on grid--> {'penalty': 'l1', 'C': 0.5}

Logistic Regression model:

Train Confusion Matrix:

[[163385 62747]

[ 72562 130956]]

Test Confusion Matrix:

[[40783 15771]

[ 1768 3181]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.676066 0.643462 0.659361 0.748222 0.685072

Test 0.167845 0.642756 0.266181 0.748511 0.714827

Sampling strategy--> 0.6

Y train after resampling Counter({0: 226132, 1: 135679})

Improved number of features-> 87

Best parameter on grid--> {'penalty': 'l1', 'C': 0.2}

Logistic Regression model:

Train Confusion Matrix:

[[190761 35371]

[ 71021 64658]]

Test Confusion Matrix:

[[47710 8844]

[ 2580 2369]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.646393 0.476551 0.548628 0.748798 0.705946

Test 0.211273 0.478683 0.293157 0.748457 0.814253

**SMOTE:**subsec level2

F1\_score at different levels of noise (noise->f1\_score):

(0.15, 0.115375),( 0.45, 0.295989),(0.6, 0.293079),(0.75, 0.283418),(0.9, 0.279562)

Sampling strategy--> 0.45

Y train after resampling Counter({0: 226132, 1: 101759})

Improved number of features-> 112

Best parameter on grid--> {'penalty': 'l1', 'C': 0.8}

Logistic Regression model:

Train Confusion Matrix:

[[201569 24563]

[ 64445 37314]]

Test Confusion Matrix:

[[50415 6139]

[ 3023 1926]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.603035 0.36669 0.456061 0.764148 0.728544

Test 0.238810 0.38917 0.295989 0.747931 0.851032

**UNDERSAMPLING: \*subsec level2**

F1\_score at different levels of noise (noise->f1\_score) :

(0.15-> 0.039771), (0.45, 0.063275), (0.75, 0.067977), ( 0.9, 0.069861)

Sampling strategy--> 0.6

Y train after resampling Counter({0: 33126, 1: 19876})

Improved number of features-> 96

Best parameter on grid--> {'penalty': 'l1', 'C': 0.8}

Logistic Regression model:

Train Confusion Matrix:

[[27915 5211]

[10488 9388]]

Test Confusion Matrix:

[[47812 8742]

[ 2595 2354]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.643058 0.472328 0.544627 0.746108 0.703804

Test 0.212149 0.475652 0.293425 0.748014 0.815668

COST FUNCTION BASED APPROACH:

Class weights--> {0: 0.1, 1: 0.9}

Improved number of features-> 96

Best parameter on grid--> {'penalty': 'l1', 'C': 0.8}

Logistic Regression model:

Train Confusion Matrix:

[[173117 53015]

[ 8101 11775]]

Test Confusion Matrix:

[[43219 13335]

[ 2002 2947]]

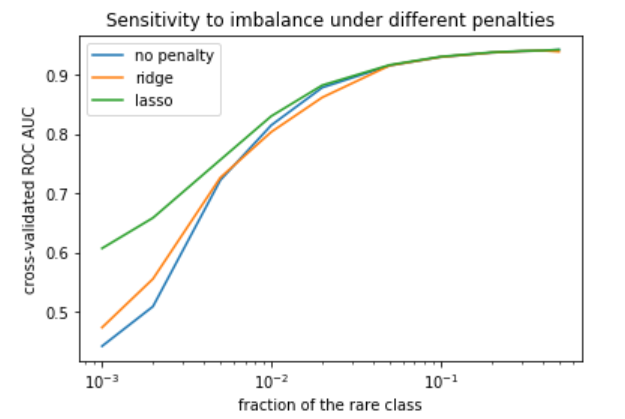
PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 0.181741 0.592423 0.278152 0.747352 0.751569

Test 0.180997 0.595474 0.277613 0.748337 0.750630

Observation:

* All techniques of imbalance improve the performance logistic regression with almost the same intensity. The best is with SMOTE with recall score shooting as high as 0.6 at sampling strategy 0.9 and highest f1\_score of approx 0.3 at sampling strategy 0.6. The statistical methods like Logistic regression tend to sharply underestimate the probability of rare events [2].
* The L1 regularization form works best with imbalanced data fig 1 [3]



**Fig1**

-The cost function based approach is by far the best for logistic with optimal results for both recall and f1\_score.

K Nearest Neighbors: \*Subsectionlevel1

**OVERSAMPLING:\*Subsection level2**

F1\_score at different levels of noise (noise->f1\_score):

(0.15, 0.107061),(0.45, 0.107061),(0.6, 0.107061),(0.75, 0.107061),(0.9, 0.107061)

Y train before resampling Counter({0: 226132, 1: 19876})

Sampling strategy--> For all sampling strategies:

Y train after resampling Counter({0: 226132, 1: 33919})

Improved number of features-> 95

Best parameter on grid--> {'n\_neighbors': 1, 'algorithm': 'auto'}

K NEAREST Neighbor: model:

Train Confusion Matrix:

[[226132 0]

[ 0 33919]]

Test Confusion Matrix:

[[52026 4528]

[ 4413 536]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

Train 1.000000 1.000000 1.000000 1.00000 1.000000

Test 0.105845 0.108305 0.107061 0.51412 0.854625

**SMOTE**: subsectionlevel2

F1\_score at different levels of noise (noise->f1\_score):

(0.15, 0.113533),(0.45, 0.127832),(0.6, 0.129434),(0.75,0.134434),(0.9, 0.134869)

Sampling strategy--> 0.9

Y train after resampling Counter({0: 226132, 1: 203518})

Improved number of features-> 115

Best parameter on grid--> {'n\_neighbors': 1, 'algorithm': 'auto'}

K NEAREST Neighbor: model:

Train Confusion Matrix:

[[226132 0]

[ 0 203518]]

Test Confusion Matrix:

[[46927 9627]

[ 3895 1054]]

PRECISION RECALL F1\_SCORE ROC\_AUC\_SCORE ACCURACY

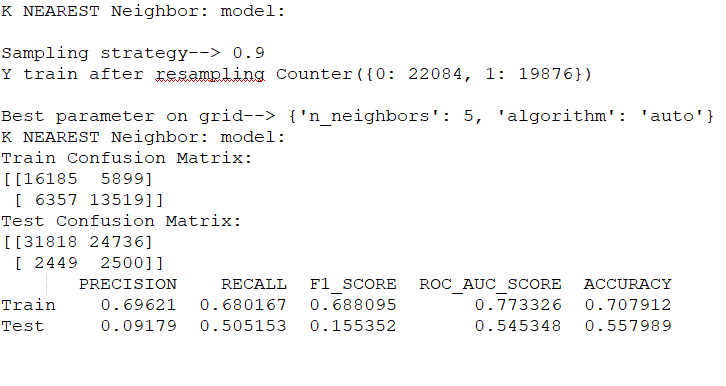
Train 1.00000 1.000000 1.000000 1.000000 1.000000

Test 0.09868 0.212972 0.134869 0.521373 0.780141

Undersampling:

F1\_score at different levels of noise (noise->f1\_score):

(0.15, 0. 0.125849),(0.45, 0.149095),(0.9, 0.155352)



Observation:

-Interestingly in the oversampling technique the KNN does not show any change in performance whatsoever. The accuracies at different noise levels are exactly the same. This is because replicating same dataset does not give any new additional information especially since the grid gives the 1 nearest neighbors as the winner. The class to which the new data belong- there is already one instance of that present before sampling so oversampling has absolutely no effect.

* The SMOTE technique also uses KNN to synthetically add new data. Hence, since it is not a replication of the existing data, the new data gives useful information to the model. We can observe (from the f1\_scores above) that with more synthetic data the KNN model is able to predict more. Hence, best sampling strategy is high at 0.9
* The undersampling technique works best for KNN at sampling strategy 0.9 with the best f1\_score at 0.155352. More importantly since the model can learn the minority class as much as the majority class at sampling strategy 0.9, the recall is very high i.e. the true positives are highest at 0.5. We observe an increase in the performance as the balance is increased in the dataset.
* There are no class weights based approach for KNN.

**CONCLUSION:**

**-Oversampling stands mediocre with RF and Decision trees and the lowest for KNN.**

**-Smote deteriorates or has not effect on the performance of Decision trees and Random forest.**

**-Undersampling boosts the performance the most for all the four methods when compared to oversampling and smote.**

**- Amongst all models Logistic performs the best with cost function based resampling at 1:9 class weights for majority:minority class. Highest f1\_score--.28,recall=0.6 and ROC\_AUC\_Score=0.75**

**REFERENCES:**

[1] https://datascience.stackexchange.com/questions/9440/how-to-customise-cost-function-in-scikit-learns-model

[2] https://gking.harvard.edu/files/abs/0s-abs.shtml

[3] https://datascience.stackexchange.com/questions/3699/what-cost-function-and-penalty-are-suitable-for-imbalanced-datasets/5031

[4] <https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction>

[5] https://www.kaggle.com/c/home-credit-default-risk