Implementation of Cost Sensitive Logistic Regression

Rahul Vigneswaran K
CS23MTECH02002

SarveshKumar Purohit
AI22MTECH14006

Rithik Agarwal
CS22MTECH11004

Vinayak Nambiar
AI22MTECH13005

Roshin Roy
AI22MTECH13006

Parth Nitesh Thakkar
CS22MTECH14005

Code can be found here here.

1. Problem Statement

Class imbalance is one the most challenging problem in modern day in the field of machine learning. most of the classification models are biased due to an overwhelming number of majority classes.

The regular machine learning classifiers treat the miss classifications like False negatives and False positives equally i.e. they associate same cost in both of the cases, which might not be helpful in some cases like fraud detection.In the case of fraud detection generally, False negatives are much more costly than False positives.

Due to these reasons a new method for logistic regression is proposed which is known as "**Cost sensitive logistic regression**".which takes different costs into account like False negative costs(FNC), False positive costs(FPS),True postive costs(TPC) and True negative costs(TNC) and integrates these costs into a loss function, which will be defined in the Algorithm section below.

2. Description of the dataset

The dataset at hand consists of one file - 'costsensitiveregression.csv'.it has a total of 11 different features and 1 column that specifies the label 1 or 0, 1 represents fraud and 0 represents not fraud. There is one last column associated with each entry that is the false negative cost, which is defined for each instance, This cost indicates the penalty when the model predicts the output Not fraud but the original label was fraud, other costs such as False positive costs,true negative costs,true positive costs are constant for all the data rows and are mentioned below.

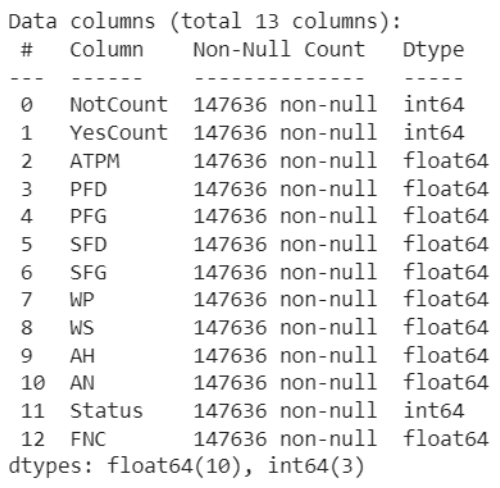
- False positive costs = 4
- True positive costs = 4
- True negative costs = 0

After analyzing the data we came to know that there is a significant imbalance in the number of data rows for fraud and non-fraud classes, the exact numbers are given below.

- Fraud cases = 44082
- Non Fraud cases = 103554

we can observe that the Non fraud cases are more than 2 times in number and thus it is very difficult for a normal logistic regression model to perform well, as it assumes the I.I.D property of the data-set.

We have taken a 80:20 split during our implementation and here is an overall summary of the data-set.



```
Data columns (total 13 columns):
 #   Column    Non-Null Count   Dtype  
---  ------    --------------   -----  
 0   NotCount  147636 non-null  int64  
 1   YesCount  147636 non-null  int64  
 2   ATPM      147636 non-null  float64
 3   PFD       147636 non-null  float64
 4   PFG       147636 non-null  float64
 5   SFD       147636 non-null  float64
 6   SFG       147636 non-null  float64
 7   WP        147636 non-null  float64
 8   WS        147636 non-null  float64
 9   AH        147636 non-null  float64
 10  AN        147636 non-null  float64
 11  Status    147636 non-null  int64  
 12  FNC       147636 non-null  float64
dtypes: float64(10), int64(3)
```

Figure 1. Summary of the dataset

3. Algorithm Used

3.1. Cost Sensitive Loss function

$$L(\theta) = 1/N \sum_{i}^{N} (y_i(h_\theta(X_i)C_{tp_i} + (1 - h_\theta(X_i))C_{fp_i}) + (1 - y_i)(h_\theta(x_i))C_{fp_i} + (1 - h_\theta(x_i))C_{tn_i}) \quad (1)$$

Note: that the above function is not a convex function

3.2. Approaches

here we have used **3 approaches** to implement the model

1)Vanilla Logistic regression with BCE(Binary cross entropy loss) and Adam optimizer

2)Cost sensitive logistic regression with loss function equation 1 and Adam optimizer

3)Genetic algorithm based Cost sensitive logistic regression

Iwe have produced results for all of the three approaches and compared all of them in terms of savings score.

Logistic regression function

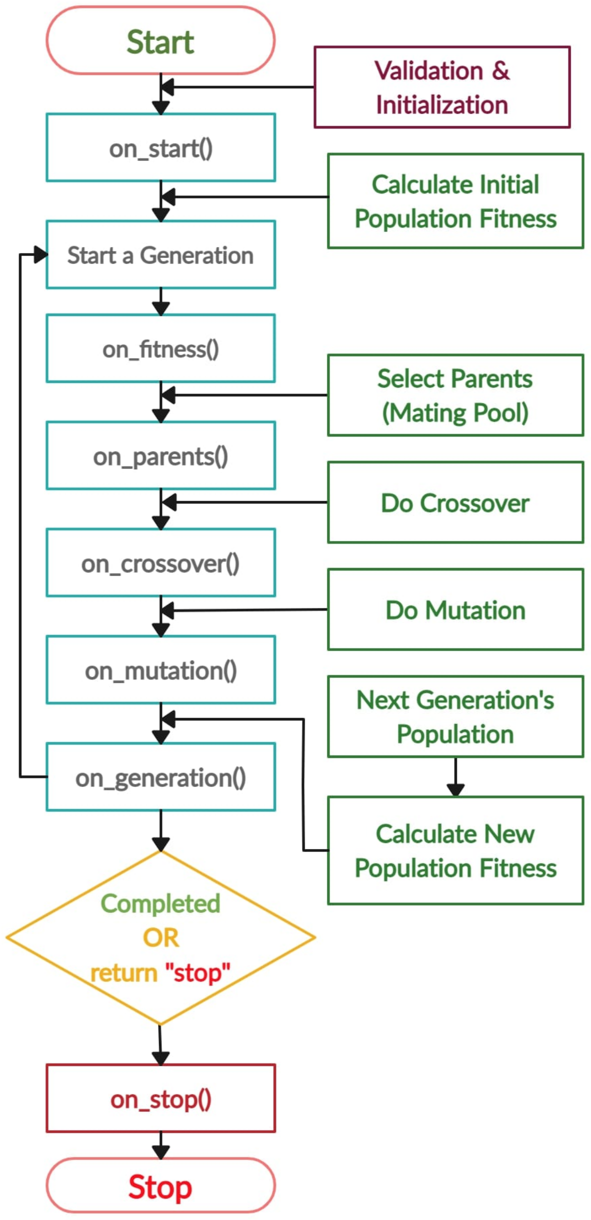
$$z = w^T * x + b$$
$$F(z) = \frac{1}{(1 + e^{-z})} \quad (2)$$

The algorithms for **approach 1 and 2** will be pretty much the same. only difference is that in approach 1 the loss function used is BCE and in second approach eq 1 is used as the loss function.

Algorithm 1 The Adam Algorithm for approach 1 and 2

```
$lr = 0.01$        //declare the learning rate
$\beta_1 = 0.9$        //decay rate for momentum
$\beta_2 = 0.99$       //decay rate for gradient history
$\epsilon = 10^{-8}$     //declaring epsilon to avoid divide by zero error
$m_0 = 0$          // declare initial moment vector
$v_0 = 0$          // declare initial grad history vector
$i = 0$            //initialize step
While $\theta$ not converged do
    i += 1
    $g_i = \nabla L(\theta_i)$ //calculate gradients
    $m_i = \beta_1 * m_{i-1} + (1 - \beta_1) * g_i$ //update momentum
    $v_i = \beta_2 * v_{i-1} + (1 - \beta_2) * (g_i)^2$ //update gradient history
    $\theta_i = \theta_{i-1} - \frac{lr * m_i}{\sqrt{v_i} + \epsilon}$ //update the parameters
end while
return $\theta$
```

The genetic algorithm which is used in the approach 3 is explained using a flow chart displayed below. genetic algorithms are used when the loss functions which you are trying to minimize is not a convex function and we may get stuck with a local optima.



Start
Validation & Initialization
on_start()
Calculate Initial Population Fitness
Start a Generation
on_fitness()
Select Parents (Mating Pool)
on_parents()
Do Crossover
on_crossover()
Do Mutation
on_mutation()
Next Generation's Population
on_generation()
Calculate New Population Fitness
Completed OR return "stop"
on_stop()
Stop

Figure 2. Genetic algorithm for approach 3

Above figure 2 summarizes the genetic algorithm approach which will not assume that our loss function is convex and it will try to optimize parameters accordingly.

4. Results

4.1. Results of Approach 1(Vanilla LR with adam optimizer)

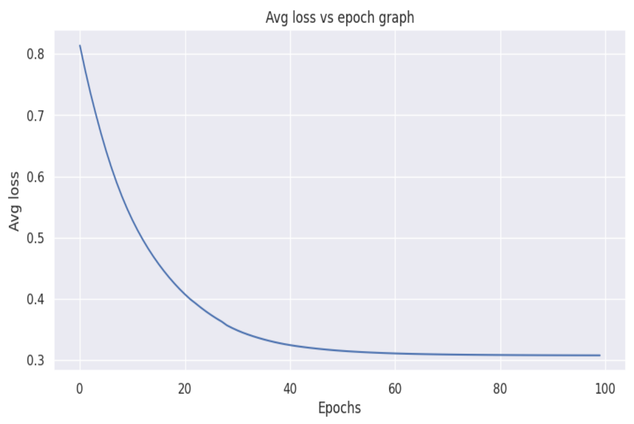


Figure 3. Avg loss per epoch

We can observe that the BCE loss is starting to converge on the value ≈ 0.3075.
The **avg cost** obtained with this approach while testing is = **26.54**

4.2. Results of Approach 2(Cost sensitive LR with adam optimizer)

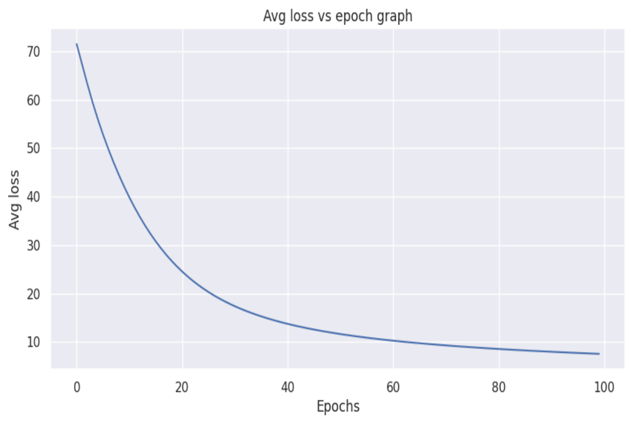


Figure 4. Avg loss per epoch

We can observe that the Cost sensitive loss is starting to converge on the value ≈ 7.554
The **avg cost** obtained with this approach while testing is = **2.67**

We can observe that results of approach 2 are much better than approach 1 as we we have taken costs into account.

4.3. Results of Approach 3(Genetic algorithm based cost sensitive LR)

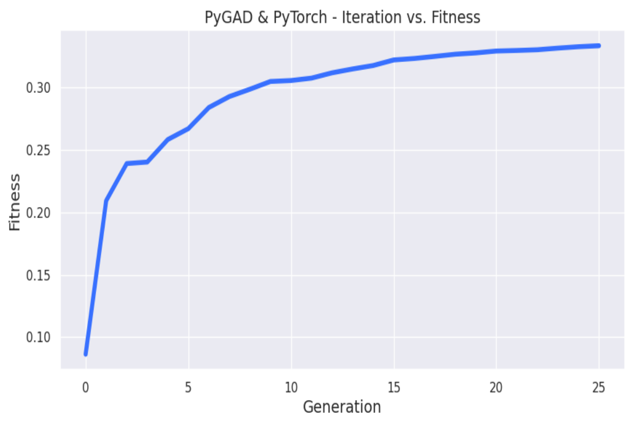


Figure 5. Fitness per generation

Note : Fitness = $\frac{1}{loss+\epsilon}$*(So it should increase with the increase in generation in the case of genetic algorithms)*

We can observe that the fitness is starting to converge on the value ≈ 0.3388, therefore the loss must be around $\approx$ 2.958
The **avg cost** obtained with this approach while testing is = **2.8**

Thus we can conclude that Approach 3 and approach 2 are performing pretty well and lastly the approach 1 is not performing well(in terms of costs), as we are not considering costs into account while training.

4.4. Final comparison based on Savings score

Savings score is a metric which will calculate the savings in terms of cost when using a cost sensitive algorithm as supposed to normal algorithm.

cost of vanilla logistic regression = CLR
cost of cost sensitive logistic regression = CSLR

$$Savings\ cost\ = \frac{CLR - CSLR}{CLR}$$

Total Costs and avg costs of all the approaches and their savings costs are summarized in the table below.

Summary of testing			
Approach	**Total cost(Lower is better)**	**Avg cost(Lower is better)**	**Savings cost**
Vanilla LR	783682	26.54	N/A
Cost-sensitive LR with Adam optimizer	78976	2.67	0.8992
Genetic algorithm based cost sensitive LR	82730	2.80	0.8944

We can observe that both our approach 2 and 3 are feasible for the current problem of cost sensitive logistic regression.

References

[1] The library used for the genetic algorithm was Pygad https://pygad.readthedocs.io/en/latest/

[2] Figure 2 working of a genetic algorithm. https://pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html#life-cycle-of-pygad

[3] Cost matters: A New Example-Dependent Cos Sensitive Logistic Regression Mode https://www.researchgate.net/publication/316369719_Cost_Matters_A_New_Example-Dependent_Cost-Sensitive_Logistic_Regression_Model