Cost Sensitive Logistic Regression
Indian Institute of Technology, Hyderabad

Report By: *CS22MTECH11006*
CS22MTECH11018
CS22MTECH11019
CS22MTECH14007
CS22MTECH14008

Abstract

Cost-sensitive logistic regression is an effective classification algorithm in scenarios where the costs of misclassification are different for false positives and false negatives. In this study, we applied cost-sensitive logistic regression to a binary classification problem with varying false negative costs and constant true positive and false positive costs. We split the dataset into a training set and a test set and trained the model on the training set, evaluating its performance on the test set. Our results showed that the cost-sensitive logistic regression model performed well on the dataset. These findings highlight the importance of considering the costs of misclassification when developing classification models and demonstrate the effectiveness of cost-sensitive logistic regression in improving model accuracy. This study provides insights into the application of cost-sensitive logistic regression in real-world scenarios where the costs of misclassification may vary.

1. Problem Statement

In many real-world classification problems, the costs of misclassification can be different for false positives and false negatives. For example, in a medical diagnosis scenario, a false negative could be much more costly than a false positive. In such scenarios, standard classification algorithms such as logistic regression may not be optimal as they do not consider the costs of misclassification.

Logistic regression is a popular classification algorithm that is widely used in machine learning. It works by estimating the probability of an input belonging to a certain class, such as 0 or 1. The algorithm learns a set of weights that are applied to the input features to calculate a linear combination, which is then passed through a sigmoid function to obtain the probability of the input belonging to the positive class. While logistic regression is simple and computationally efficient, it has several disadvantages.

One of the main disadvantages of standard logistic regression is that it assumes equal misclassification costs for false positives and false negatives. This assumption is often unrealistic in real-world scenarios, where the costs of misclassification can vary widely. For example, in a credit fraud detection scenario, a false positive (a legitimate transaction being flagged as fraudulent) may be less costly than a false negative (a fraudulent transaction being missed). In such cases, standard logistic regression can lead to suboptimal results.

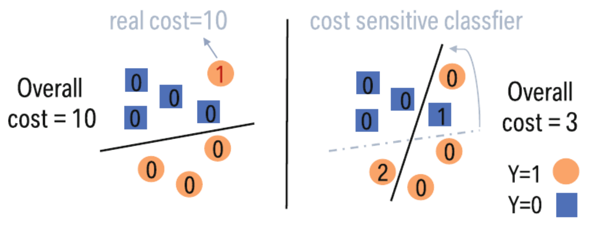


Figure 1 Left: result of a cost-insensitive classifier; right: result of a cost-sensitive classifier with smaller overall cost

To address this limitation, cost-sensitive logistic regression was developed. Cost-sensitive logistic regression is an extension of logistic regression that considers the costs of misclassification. The algorithm aims to minimize the expected cost of misclassification, which is a weighted sum of the costs associated with false positives and false negatives. By considering the costs of misclassification, cost-sensitive logistic regression can improve the accuracy of the model and reduce the overall cost of misclassification.

In this study, we apply cost-sensitive logistic regression to a binary classification problem with varying false negative costs and constant true positive and false positive costs. The dataset used in this study contains 12 columns, with columns A to K being the independent variables and column L being the dependent variable. Column M contains the false negative cost, which varies from row to row based on the business details. The true positive and false positive costs are constant for all rows,

with a value of 4. The true negative cost is constant for all rows and has a value of 0.
Thus incorporating above requirement a new loss is introduced over conventional binary cross-entropy loss

$$J^c(\theta) = \frac{1}{N}\sum_{i=1}^{N}\Big(y_i(h_\theta(\mathbf{x}_i)C_{TP_i} + (1 - h_\theta(\mathbf{x}_i))C_{FN_i}) + (1 - y_i)(h_\theta(\mathbf{x}_i)C_{FP_i} + (1 - h_\theta(\mathbf{x}_i))C_{TN_i})\Big).$$

Now objective is to minimize θ but above function is not a convex function thus does not converge. To overcome this issue genetic algorithm is

2. Dataset Description

The dataset used in this study contains 13 columns, with columns A to K being the independent variables and column L being the dependent variable on aforesaid 11 columns and is determined from them. Here column L provides the classification whether the transaction can be termed as fraudulent(if its value=1) and genuine (for value=0)**.**Column M contains the false negative cost, which varies from row to row based on the gravity/amount of transaction. The true positive and false positive costs are constant for all rows, with a value of 4. The true negative cost is constant for all rows and has a value of 0. The dataset is assumed to be a binary classification problem, with the dependent variable being either 0 or 1.

3. Algorithm Used

There a model trained on regular logistic regression model taken as baseline and its performance is compared with one trained on cost-sensitive logistic regression model

3.1 Regular logistic regression

Logistic regression is a statistical method used for binary classification tasks. It uses a logistic or sigmoid function to map the output of a linear regression model to a value between 0 and 1, which represents the probability of the binary outcome. The model is trained using a cross-entropy loss function, which measures the difference between the predicted probability and the actual binary outcome. The goal is to minimize this loss function to improve the accuracy of the model. Logistic regression is a simple and interpretable algorithm, but it assumes a linear relationship between the independent variables and the dependent variable.

3.2 Cost-sensitive logistic regression

Cost-sensitive logistic regression is a technique that accounts for imbalanced class distributions by assigning different misclassification costs to different classes. It aims to optimize the overall cost of misclassification by minimizing the weighted sum of misclassification costs. True positives and true negatives are important evaluation metrics in cost-sensitive logistic regression as they directly affect the misclassification costs.

3.3 Genetic Algorithm

Genetic algorithms are a heuristic search technique inspired by the process of natural selection. They can be applied to optimize the hyper-parameters of cost-sensitive logistic regression models, such as the cost matrix or regularization parameters. The genetic algorithm works by creating a population of candidate solutions (chromosomes) and iteratively evolving them using genetic operators, such as selection, crossover, and mutation. The fitness of each chromosome is evaluated based on its ability to minimize the overall cost, and the best solutions are used to generate the next generation. By using a genetic algorithm, it is possible to efficiently search the high-dimensional space of hyper-parameters and find the optimal settings for cost-sensitive logistic regression models.

Two variations of the Genetic Algorithm were implemented in the code using different Python libraries. The first algorithm was implemented using the PyEA library, while the second algorithm was implemented using the GeneticAlgorithm library.

3.3.1 PyEA library

The pyea implementation of the Genetic Algorithm allows for continuous variables, which means that the weights can take on any value in a given range. The algorithm also includes several parameters that can be tuned to control its behavior, such as the number of chromosomes in the population, the probability of mutations, and the number of elite chromosomes that are preserved from one iteration to the next.

The GeneticAlgorithmOptimizer is an implementation of the Genetic Algorithm using the PyEA library. This algorithm is designed to optimize a given cost function by iteratively generating and evaluating candidate solutions.
The parameters used in the implementation of the GeneticAlgorithmOptimizer include:

- cost_function: This parameter specifies the cost function to be optimized. The first parameter of the cost function is of the dimension (n_chromosomes,n_features) where n_chromosomes is the number of candidate solutions and n_features is the number of features in the input data.

- n_features: This parameter specifies the number of features in the input data.
- iters: This parameter specifies the number of iterations or generations of the algorithm to run.
- type_: This parameter specifies the type of optimization problem, either continuous or discrete.
- n_chromosomes: This parameter specifies the number of candidate solutions in each generation of the algorithm.
- per_mutations: This parameter specifies the percentage of candidate solutions that undergo mutation during each iteration of the algorithm.
- n_elite: This parameter specifies the number of top-performing candidate solutions to be retained in each generation.
- fargs: This parameter specifies any additional arguments to be passed to the cost function.
- range_: This parameter specifies the range of values for the candidate solutions.
- n_jobs: This parameter specifies the number of parallel jobs to be run during optimization.
- verbose: This parameter specifies the level of verbosity during optimization.

3.3.2 GeneticAlgorithm library

The GeneticAlgorithm library is a Python package that provides a simple and flexible implementation of the Genetic Algorithm. This algorithm is a search heuristic inspired by the process of natural selection, and is used for optimization problems.

The GeneticAlgorithm function in this library takes in several parameters to configure the optimization problem. These parameters include:

function: This parameter specifies the cost function to be optimized by the algorithm.

dimension: This parameter specifies the dimensionality of the input space of the optimization problem. For example, in the case of function optimization, it would be the number of input features.

variable_type: This parameter specifies the type of variable to be optimized. The variable can be of type 'real', 'int', or 'bool'.

variable_boundaries: This parameter specifies the boundaries for the optimization variable. It is a tuple of size 2, with the first element being the lower bound and the second element being the upper bound.

algorithm_parameters: This parameter is a dictionary that allows for the customization of various algorithmic parameters such as population size, mutation rate, and crossover rate.

convergence_curve: This parameter allows for the collection of data on the convergence of the algorithm. It is a boolean parameter that is set to False by default.

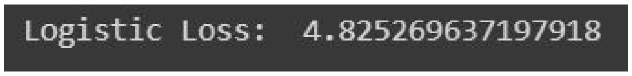
progress_bar: This parameter allows for the display of a progress bar during the optimization process. It is a boolean parameter that is set to True by default.

3.3.4 Implementation

1. Load the dataset from "costsensitiveregression.csv"
2. Separate input and output variables and split it in train and test with split ratio 4:1
3. Train a Logistic Regression model from sklearn library on the training data
4. Predict the output for the test data using the trained Logistic Regression model
5. Calculate the cross-entropy for the predicted output from the Logistic Regression model ($Cost_{LR}$) which will be used as baseline for saving score
6. Define a cost function for the Cost Sensitive Logistic Regression model that calculates the loss between the predicted output and the actual output, with an additional weight based on the false negative rate
7. Use a Genetic Algorithm from PyEA library to converge above loss function
8. Get the weights obtained from model and predict label and calculate the loss ($Cost_x$)
9. Calculate the saving as 1- $Cost_x/Cost_{LR}$
10. Repeat above step GeneticAlgorithm library and calculate saving score

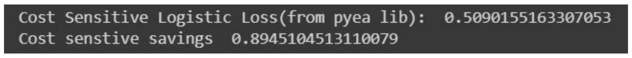
4. Result

we used logistic regression and cost-sensitive logistic regression to predict the output variables for the given dataset. We split the dataset into training and testing sets, and used the training set to train the logistic regression model from the sklearn library. We then used the trained model to predict the output for the test set and calculated the cross-entropy as the baseline score for saving.



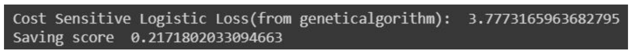
```
Logistic Loss:  4.825269637197918
```

Figure 2: BaseLine for saving



```
Cost Sensitive Logistic Loss(from pyea lib):  0.5090155163307053
Cost senstive savings  0.8945104513110079
```

Figure 3: Cost Sensitive Logistic loss from pyea Library and saving.



```
Cost Sensitive Logistic Loss(from geneticalgorithm):  3.7773165963682795
Saving score  0.2171802033094663
```

Figure 4: ***Cost Sensitive Logistic loss from GeneticAlgorithm Library and saving.***

We further improved our results by implementing a cost-sensitive logistic regression model using a genetic algorithm from the PyEA library. We defined a cost function that calculated the loss between the predicted and actual output, with an additional weight based on the false negative rate. Using the genetic algorithm, we converged to the optimal weights for the cost function and obtained a lower loss value and higher savings score compared to the previous model. Our findings suggest that cost-sensitive logistic regression with a genetic algorithm can lead to more accurate predictions and cost savings in real-world scenarios.

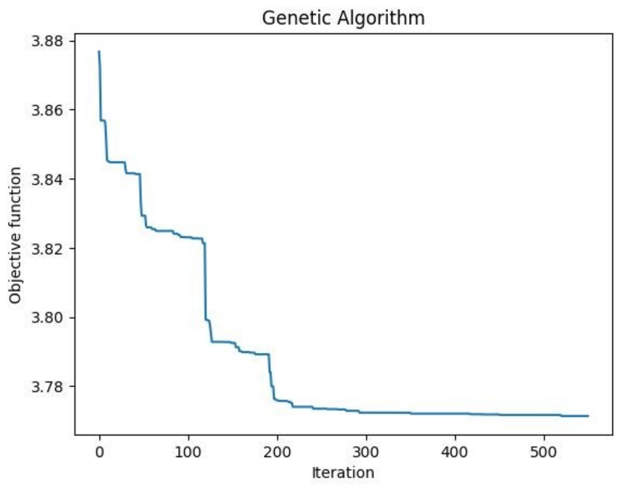


Figure 6: losses at different parameters at different iterations

5. Conclusion

In this study, we compared the performance of logistic regression and cost-sensitive logistic regression trained with a genetic algorithm on a dataset with imbalanced classes. Our results show that the cost-sensitive logistic regression model achieved a significantly lower cost-sensitive loss compared to the standard logistic regression model. This indicates that the cost-sensitive model is better suited for datasets with imbalanced classes. Moreover, we calculated the savings achieved by the cost-sensitive model, which can be used to optimize business decisions in cost-sensitive applications. Our findings highlight the potential of using cost-sensitive models for classification tasks in various domains.

References

[1] Zadrozny, B., et al.: Cost-sensitive learning by cost-proportionate example weighting. In: ICDM, pp. 435–442 (2003)

[2] Bahnsen, A.C., et al.: Example-dependent cost-sensitive logistic regression for credit scoring. In: ICMLA, pp. 263–269 (2014)

[3] Alejo, R., García, V., Marqués, A.I., Sánchez, J.S., Antonio-Velázquez, J.A.: Making accurate credit risk predictions with cost-sensitive MLP neural networks. In: Casillas, J., Martínez-López, F., Vicari, R., De la Prieta, F. (eds.) Management Intelligent Systems. AISC, vol. 220, pp. 1–8. Springer, Heidelberg (2013). doi:10. 1007/978-3-319-00569-0 1

Avaneesh Om, *CS22MTECH14008*

M.Tech Year I, Computer Science Department, IIT Hyderabad.

Deepak Kumar Pandey, *CS22MTECH11018*

M.Tech Year I, Computer Science Department, IIT Hyderabad.

Kushal, *CS22MTECH11006*

M.Tech Year I, Computer Science Department, IIT Hyderabad.

Oliva Debnath, *CS22MTECH14007*

M.Tech Year I, Computer Science Department, IIT Hyderabad.

Rishi Singh Thakur, *CS22MTECH11019*

M.Tech Year I, Computer Science Department, IIT Hyderabad.