Cost Sensitive Logistic Regression

Anirudh Raghav
CH19BTECH11033

Anushka Khare
CH19BTECH11029

Tanmay Goyal
AI20BTECH11021

Tanay Yadav
AI20BTECH11026

Vedika Verma
CS19BTECH11057

Abstract

Logistic Regression is one of the most popular Machine Learning methods that unlike the name suggests, is a classification method. Given the data, it predicts the probability of a sample belonging to class 1. Logistic Regression uses a Binary cross Entropy Loss function that provides an equal cost of misclassification. However, we shall now look at Cost Sensitive Logistic Regression, which shall provide unequal costs of misclassification, i.e the model shall wish to minimize the error of prediction while also taking care of the cost of misclassification. A simple example of why misclassification costs are necessary is: an RTPCR test returning negative for a Covid-positive person is much more dangerous than an RTPCR test returning positive for a Covid-negative person. In this work, we shall try to predict the dependent variables using Cost Sensitive Logistic Regression.

1. Problem Statement

Logistic Regression is one of the most popular Machine Learning methods that unlike the name suggests, is a classification method. Given the data, it predicts the probability of a sample belonging to class 1. Logistic Regression uses a Binary cross Entropy Loss function that provides an equal cost of misclassification. The loss function for Logistic Regression is as follows:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}) + (1-y)\log(1-\hat{y})$$

where $\hat{y}$ is the probability of the sample being classified to class 1 and is given by:

$$\hat{y} = \sigma(w_1x_1 + w_2x_2 + \dots w_nx_n + b)$$

where $\sigma(.)$ is the sigmoid function, which converts $w_1x_1 + w_2x_2 + \dots w_nx_n + b)$ into a number between 0 and 1 and is given by:

$$\sigma : \mathbb{R} \to [0,1]\ ;\ \sigma(x) = \frac{1}{1+e^{-x}}$$

Now, we shall now look at Cost Sensitive Logistic Regression, which shall provide unequal costs of misclassification, i.e the model shall wish to minimize the error of prediction while also taking care of the cost of misclassification. The loss function shall now be modified as:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N} y[C_{TP}\hat{y} + C_{FN}(1-\hat{y})] + (1-y)[C_{FP}\hat{y} + C_{TN}(1-\hat{y})]$$

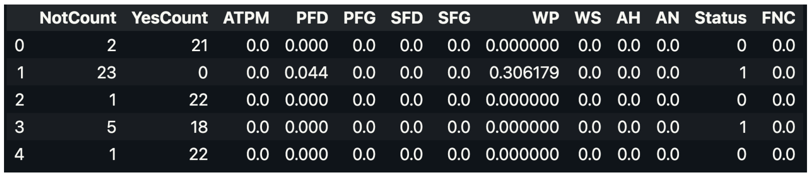
Here, we can see the error is now being weighted by the costs of misclassification, and thus, the model is forced to optimize such that the costs of misclassification are taken into account.

2. Dataset

The dataset consists of 13 columns and 147635 rows. The first 11 columns, namely ['NotCount', 'YesCount', 'ATPM', 'PFD', 'PFG', 'SFD', 'SFG', 'WP', 'WS', 'AH', 'AN'] are the independent variables, while the column 'Status' is the dependent variable. The 13th column, 'FNC' is the false negative cost for each row.

Apart from this, the True Positive and False positive costs are 4 for all rows, and the True Negative cost is 0 for all rows.

A snapshot of the dataset is given below:



	NotCount	YesCount	ATPM	PFD	PFG	SFD	SFG	WP	WS	AH	AN	Status	FNC
0	2	21	0.0	0.000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0	0.0
1	23	0	0.0	0.044	0.0	0.0	0.0	0.306179	0.0	0.0	0.0	1	0.0
2	1	22	0.0	0.000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0	0.0
3	5	18	0.0	0.000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	1	0.0
4	1	22	0.0	0.000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0	0.0

Figure 1. A snapshot of the dataset

3. Algorithm and Methodology

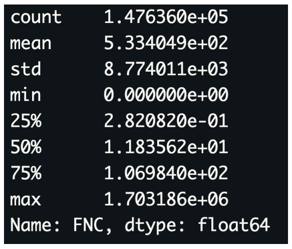
We shall implement logistic regression using a one-layer Neural Network (no hidden layers) followed by a sigmoid activation to convert it into a probability.

1. We shall break the dataset into X, which is all the independent features, Y, which is the column 'Status', the

dependent variable, and FNC, the False Negative cost for each row.

2. We shall then split the data into training and testing data in the ratio of 80:20 by picking random indices. In this way, we split X, Y, and FNC into training and testing components.

3. We check the statistics of the False Negative Costs and find that it becomes very high for certain rows, as compared to the other costs which are of the order of 10^0. This results in the model only predicting $\hat{y} = 1$ since that takes the False Negative Cost out of the Equation. Thus, we normalize the False Negative Cost to be constrained between 0 and 5. We also assume that the testing samples will arrive one-by-one; hence, we cannot normalize the False Negative cost for the testing samples using the maximum and minimum of the testing samples. Thus, we assume the maximum and minimum of the training samples is used in normalizing the False Negative Cost of the testing samples. Finally, we assume the False Negative cost is known to us and can be fed to the model, and hence, can be used as a predictive feature since the False Negative Cost should also have a say during testing.



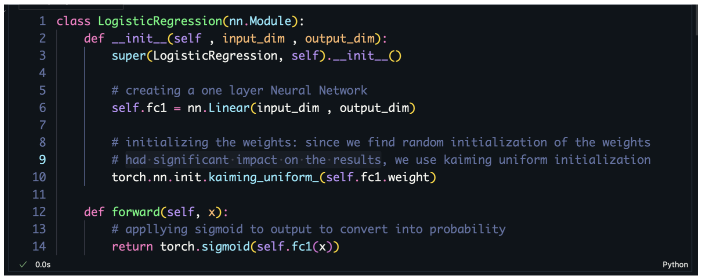
```
count    1.476360e+05
mean     5.334049e+02
std      8.774011e+03
min      0.000000e+00
25%      2.820820e-01
50%      1.183562e+01
75%      1.069840e+02
max      1.703186e+06
Name: FNC, dtype: float64
```

Figure 2. A snapshot of the statistics of the False Negative Costs Columns

4. We define the Logistic Regression Model as a one-layer Neural network with no Hidden Layer and a sigmoid activation function at the end. We observed that random initialization of the weights led to a significant change in our results. Thus, we decided to initialize the weights using Kaiming Initialization. We use an Adam optimizer with a learning rate of 0.005.

5. We train two different models: one with the cost-sensitive loss function and one with the normal Binary Cross Entropy Loss.

6. Finally, we also find the number of misclassifications made by both models and compare the two.



```python
class LogisticRegression(nn.Module):
    def __init__(self , input_dim , output_dim):
        super(LogisticRegression, self).__init__()

        # creating a one layer Neural Network
        self.fc1 = nn.Linear(input_dim , output_dim)

        # initializing the weights: since we find random initialization of the weights
        # had significant impact on the results, we use kaiming uniform initialization
        torch.nn.init.kaiming_uniform_(self.fc1.weight)

    def forward(self, x):
        # appllying sigmoid to output to convert into probability
        return torch.sigmoid(self.fc1(x))
```

Figure 3. Defining the Class for the Logistic Regression Model

4. Results

1. The training and testing curve for the Cost-Sensitive Model is given below:

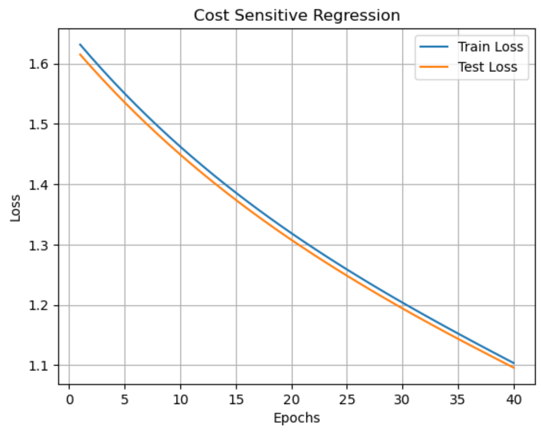


Figure 4. Training and Testing Loss for the Cost-Sensitive Model

2. The training and testing curve for the Normal Model is given below:

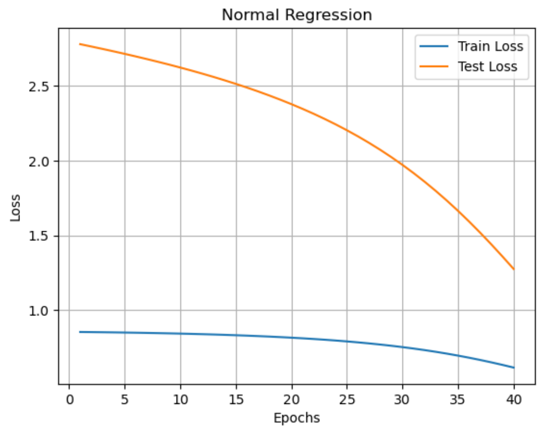


Figure 5. Training and Testing loss for the Normal Model

3. The accuracy of the Cost-Sensitive Model turns out to be 86.64% while the accuracy for the Normal Model turns out to be 35.56%.

4. We compare the number of misclassifications for both models:

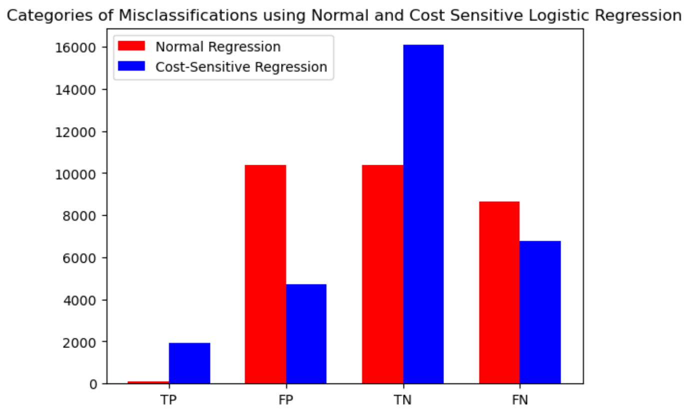


Figure 6. Training and Testing loss for the Normal Model

5. We find that the number of False Negative Predictions immediately goes down since they have a high cost associated with them. On the other hand, the number of True Negatives goes up since they do not have a cost associated with them. It can also be seen using the fact that if the true label $y = 0$, then if the prediction of $\hat{y}$ goes down, then the prediction of $\hat{y}$ needs to go up. We also find that the Normal model almost never correctly predicts the class $y = 1$. Thus, we can say that the cost-sensitive loss function has brought about a change in the way the model Learns.