CS6890: Fraud Analytics Using Predictive and Social Network Techniques, 2020

Assignment 4: Example dependent cost sensitive classification using deep neural net

SAI HARSHA KOTTAPALLI - CS17BTECH11036

Tanmay R - CS17BTECH11042

OBJECTIVE

The objective for this assignment is to identify fraudulent tax payers from the given dataset containing various details regarding their transactions.

Generally, most models only try to minimize the so-called zero-one loss where all misclassifications have equal cost and the correct classifications having cost zero.

But in the real world, this tends to not be the case, the cost of misclassifications might significantly vary based on the application and its instance. Particularly in our case, depending on the type of transactions done by a fraudulent person classifying him into genuine has different costs. For example, the cost incurred on classifying a person who has embezzled a few lakhs as genuine is not the same as someone who has embezzled a few hundreds.

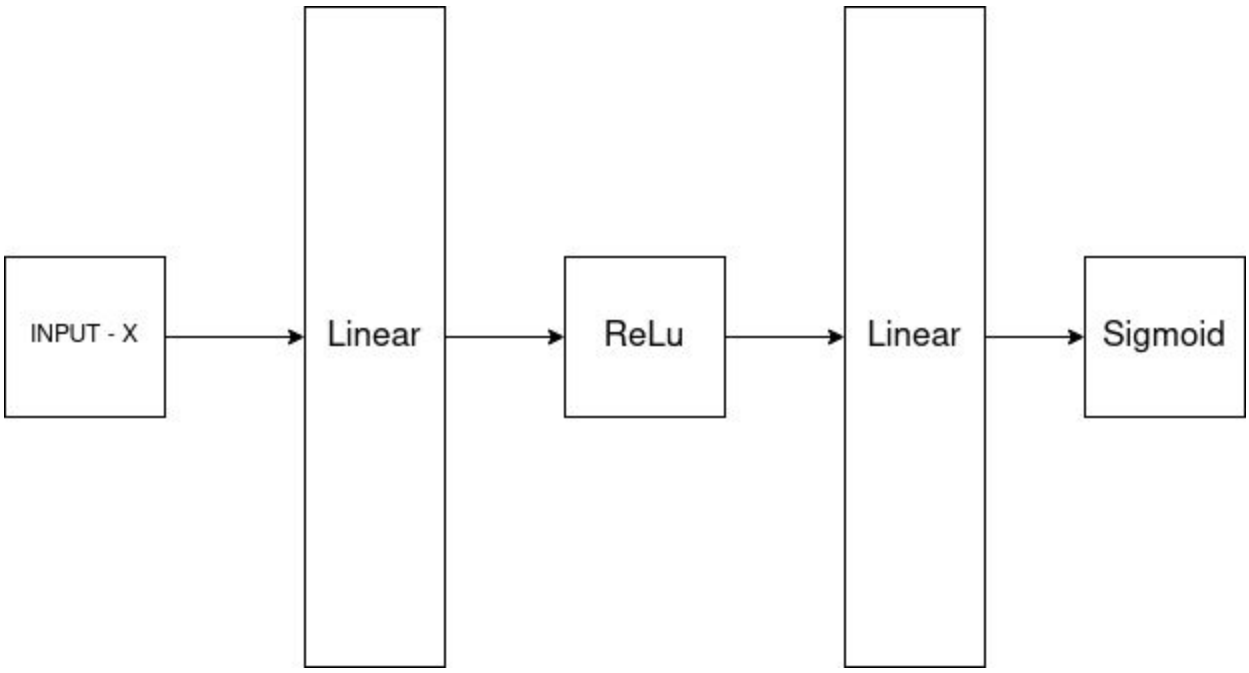
Neural networks are complex models, which try to mimic the way the human brain develops classification rules. A neural net consists of many different layers of neurons, with each layer receiving inputs from previous layers, and passing outputs to further layers until the terminal neurons output the final result. Essentially, at each layer, we apply linear transformation:

$$a_1 = W_1 * x + b_1$$

and then apply an activation function to each neuron. In our case, we will be using ReLu which will output the *input* directly if it is positive, otherwise, it will output zero.

For this assignment, we will use the Neural Network model and a cost sensitive loss function to predict if a person is fraudulent or genuine.

Here, the output is calculated using the following layers:



Here the first layer is of dimension 8 while the second layer is of dimension 1.

Finally using a threshold value of 0.5 we can classify the output based on the sigmoid output.

For the purpose of cost sensitive linear regression, we use the following cost function:

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

DATA SET SUMMARY

The dataset consists of 147636 people's transaction history comprising 11 features(independent variables) namely, NotCount, YesCount, Average tax per month(ATPM), purchase from fraudulent(PFD), purchase from genuine(PFG), Sales to fraudulent(SFD), sales to genuine(SFG), purse value in way bill(WP), sales value in way bills(WS), AH and AN.

It also provides us the label(Status) whether the person is fraudulent or genuine so that we can train the model accordingly and also the FNC associated with each person's classification, i.e., False Negative cost for misclassifying fraudulent person as genuine. Some of the cells are zero under FNC, this might be due to the fact that a person is associated with a very small business and there's not much impact in the real world by misclassifying him.

In the dataset, while training I tried to remove the data points which have extremely high FNC and the ones which have FNC as 0, but i didn't find any significant difference from taking random points , so i took a random split of 70% as training dataset and 30% for test dataset. In fact the accuracy dropped by a small amount, hence i decided to not remove the outliers.

Also since the data was already normalized but not exactly the method in which it was one, I decided to leave YesCount and NoCount as it is, so that the Neural Network can learn weights accordingly given that all of the features are independent.

EXPERIMENTATION RESULTS

In our case since it takes manpower to call the person identified as fraudulent to verify, we associate the cost of true positive and false positive as 150 each. There is no cost incurred for identifying genuine persons as genuine.

As for the false negative cost for misclassifying fraudulent people as genuine, as mentioned above is already provided in the dataset.

We use a seed at the beginning so that the results are reproducible from the code.

We then did do a grid search over learning rate from 1e-4 to 1e-8, with *Lecun initialization* for weights(the default in pytorch) and try out various other combinations for optimizations(Adam, SGD, etc) to find the optimal parameters and corresponding result:

Epochs	**TP**(given)	**TN**(given)	**FP**(given)	**FN**(given)	**LR**	**Optimizer**	**Accuracy**	**F1 Score**
2000	150	0	150	in dataset	5e-05	SGD	86.44%	76.14%

Where, TP = true positive cost, TN = true negative cot, FP = false positive cost, FN = false negative cost, LR = learning rate.

Here, F1 score is calculated as :

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

We also notice that as we increase epochs(on overfitting), the accuracy starts decreasing at some point after stabilizing.

When compared to logistic regression, though we have much more possibilities to explore by adding more layers and trying out different activation functions, but i have not found any significant improvement on this particular dataset. Though i have noticed that different hyperparameters and combinations give better accuracy than the one i have mentioned

above, the cost of it was very high(around 100 more than the cost obtained form above mentioned optimal parameters), this is a high cost as we can see that the TP and TN costs themselves are 150 each. It might also be explained by the fact that since the optimizers is SGD, it had attained a wrong local minima for cost.