Author : Taipa Gibon Huchu (Data Scientist)

Business orientated loss function in fraud detection

The precision, recall, F1-score may not really work, The article will describe the traditional metrics for binary classification, why they don't work, the new cost function, and the implementation of the new loss function in Logistic Regression with sklearn and DNN with Tensorflow.

Introduction

Fraud detection plays a more and more important role in many industries: insurance, banking, retail, eCommerce, and even more. The ability to capture more fraudulent behavior and save more money for the business is how we evaluate the performance of a fraud detection model. However, to translate it into a data science problem, especially into a supervised machine learning problem, people also choose the wrong metrics when building the models.

In the traditional binary classification problems, we try to minimize the loss function such as Log-Loss or maximize metrics like F1-score, accuracy, or AUC, etc. It is totally correct for data science, but not right for real-world business problems.

First, accuracy doesn't work because of the imbalanced datasets. In fraud detection problems, usually, we have 99% more non-fraudulent data and less than 1% fraud data. So if we predict all data points as a normal class, we will at least get 99% correct. This is called accuracy paradox.

Second, F1, AUC, and log-loss also don't work. Why? Before going deeper about the reason, let's recap the definition of these metrics. Feel free to skip this section if you are already familiar with them.

Traditional Metrics Recap

When the model makes the decisions for binary classification problems, it will predict whether it's positive or negative, in other words, fraud or non-fraud, based on the testing independent features.

Given the fact, the model can predict correctly. If it predicts positive given it's positive, we call it **True Positive(TP)**. If it predicts negative given it's negative, we call is **True**

Negative(TN). However, the model will make mistakes: **type I error and type II error**.

Type I error, also called False Positive(FP), is the number of data wrongly identified as positive given it's negative.

Type II error, also called False Negative(FN), is the number of data wrongly identified as negative given it's positive.

There are also other situations: the model predicts positive correctly out of total true positives, which is called **Recall or Sensitivity or TPR(True Positive Rate)**, and the model predicts negative correctly out of total true negative, which is called **Specificity or TNR (True Negative Rate),** the model predicts positive correctly out of total items identified as positive, which is **Precision.**

The F1-score is a combination of precision and recall-2*Precision*Recall/(Precision + Recall), while the accuracy is total items classified correctly (TP+TN) / (TN+TP + FP+FN).

AUC, is when we randomly pick a positive case and a negative case, the probability that the positive case outranks the negative one according to the classifier is given by the AUC. Mathematically, it equals the area under the ROC curve. Notice that AUC is independent of the threshold set for classification because it only considers the rank of each prediction and not its absolute value.

New Cost Function

Now let's go back to the question: why they don't work well for fraud detection. Because they consider the costs for every mistake the same. In the real world, it's totally not true. For example, customer A makes a transaction and the system predicts that it's a fraud order, then cancel the order. The cost for the company is the customer service fee when customer A contacts the company about the transaction. Customer B makes a fraud transaction the system approves it. Then the cost is chargeback for the bank and the order value.

Given that, we will have different costs for different predicted outputs.

For the True Positive, the model predicts it's a fraud given it is really a fraud, we cancel the order correctly, the company will have no costs or just administration fees.

For the True Negative, the model predicts it's a good order given it is really good order, we will approve it and no costs occur. ($0)

For the False Positive, the model predicts it's a fraud given it is good order, we will need to pay for the customer service fee(and potential the cost of losing a customer).

For the False Negative, the model predicts it's a good order given it is a fraud, we approve the order wrongly and need to pay for the chargebacks while losing the amount of the order.(**example-dependent**)

So to evaluate the performance of the model in terms of costs, we need to calculate the total costs from all outputs using the following formula:

We can make a ratio (cost_ratio) from the costs by dividing it with the max_cost, which is the costs from when we predict everything as negative(approve all orders!). Now, our goal is to minimize the cost_ratio.

Modify the Loss Function in Logistic Regression

Remember that the standard loss function for Logistic Regression is log-loss, which punishes false negatives and false positives equally. In our new cost function, we need to punish false negative and false positives differently.

Modify the Loss Function in Keras TensorFlow for DNN models

In Keras, a loss function **must accept only 2 arguments**: y_true and y_pred, which are the target tensor and model output tensor, correspondingly. But what if we want our loss/metric to depend on other tensors other than these two? We can use function closure, where we create a function which returns a function of

Reference:

Advanced Keras — Constructing Complex Custom Losses and Metrics
A simple trick for constructing multi-argument custom losses and metrics in Keras
towardsdatascience.com

Choosing the Right Metric for Evaluating Machine Learning Models — Part 2
Second part of the series focussing on classification metricsmedium.com