Statistical Descriptions

Mathematics of Deep Learning - Classification Project

December 20, 2019

The aim of this document is to detail the mathematical and statistical descriptions of the methods and algorithms used to deal with the Credit dataset. The interpretations of the graphical representations and the models will be kept in the RMarkdown file.

We recall that we had at our disposal two datasets (train and test) of 5380 and 1345 customers each, respectively. For each of the datasets, the data is labeled, which means that we are in a supervised learning case because we know if the bank issued the credit to every customer ($Y = y_1 = 0$) or not ($Y = y_2 = 1$). In total, we have $p = 19$ explanatory variables plus the target variable Y. Mathematically, this comes up to learn a function f on $\{(x_i, Y_i)\}_{1 \leq i \leq n}$ where $x_i = \begin{pmatrix} x_i^1 \\ . \\ . \\ x_i^p \end{pmatrix}$ is an individual.

1 SMOTE: Solving the class-imbalance of the target variable

One of the main steps that was required for the modeling preparation was solving the class-imbalance of the data. Indeed, after the cleaning process, we had of 393 (7.30%) observations that had $Y = 1$ and 4987 (92.7%) with $Y = 0$, which is understandable because a bank issues a credit to fewer customers. The issue in those cases is that many standard classifiers such as Decision Trees are constantly using majority rules when making predictions, and so they will hardly ever predict the minority class $Y = 1$ for any new observation from the test set. This leads to biased and inaccurate models.

To solve this problem we used the function SMOTE, implemented in the R package $DMwR$. [al02] The general idea of this method is both to artificially generate new examples of the minority class using the nearest neighbors of these cases, as well as to under-sample the majority class. Indeed, the literature suggests that simple over-sampling is not always useful enough because the classification rule is unmodified. [Rok10] The Synthetic Minority Over-sampling TEchnique SMOTE seems to address this problem. For the parameters we chose:

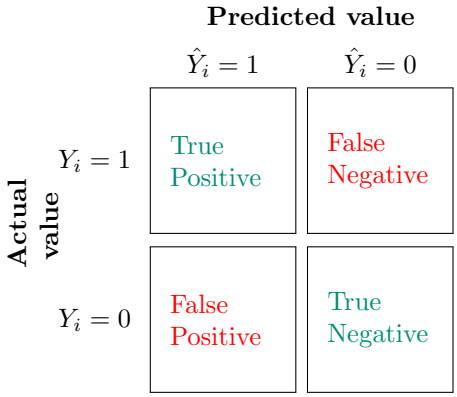
- *perc.over (controls over-sampling) = 100*: this will double the number of extra cases from the minority class that are generated, based on $k = 5$ (value by default) nearest neighbours
- *perc.under (controls under-sampling) = 200*: this will select the double of cases from the majority class than the number of created cases from the minority class.

It is very important to check that SMOTE kept the original balance of the classes when under-sampling. The sampling it uses is not stratified so it could happen that the resulting 786 individuals are not representative of the initial population of 4987. In this case when predicting for new individuals of the test set these would have modalities that were never seen before, during the learning phase.

Hence at the end we have, 786 (50.0%) observations that had $Y = 1$ and 786 (50.0%) with $Y = 0$. This is a perfectly balanced dataset.

2 Model evaluation

Here we present the techniques used in order to assess the performance of our models. To summarize the type of errors that we make when predicting, we use a confusion matrix:



A False Negative is also called the type II error, and a False Positive is also called the type I error. The important usual metrics are:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Precision answers the question: 'When predicting $\hat{Y}_i = 1$, how many times are we correct', whereas recall answers: 'How many times do we predict correctly among the cases where we actually have $Y_i = 1$'. They are usually combined into a single metric:

$$f1\_score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \in [0, 1]$$

A perfect classification would yield an $f1\_score$ of 1, and if we predict everything wrongly an $f1\_score$ of 0.

3 Models

3.1 Logit

Y only takes to values so it is a Bernoulli random variable. The principal idea of Logit model is to use a logistic function (sigmoid shape) to describe the probability of success. Hence, in our case, the probability for the bank not to issue a credit to a customer i is:

$$\mathbb{P}(Y_i = 1|x_i) = \frac{1}{1 + \exp(-x_i\beta)} := f(-x_i\beta)$$

It is well defined because the function is always increasing and its asymptotic behavior is: $\lim_{t\to+\infty} f(t) = 1$ and $\lim_{t\to-\infty} f(t) = 0$ so that $\mathbb{P}(Y_i = 1|x_i) \in [0, 1]$. Of course, we also have:

$$\mathbb{P}(Y_i = 0|x_i) = 1 - \mathbb{P}(Y_i = 1|x_i)$$

The vector of coefficients β is estimated by Maximum Likelihood Method. In other words, it means that every $\beta_0, \beta_1, ..., \beta_p$ is selected such that the likelihood of predicting a high $\mathbb{P}(Y_i = 1|x_i)$ and a low $\mathbb{P}(Y_i = 0|x_i)$ is high. Once we have these probabilities, we still need to choose a threshold s above which we will classify as not giving the credit and vice versa. To do this, we use a confusion matrix such as the one presented above. If we increase the threshold, then we will classify as 1 less often, so the number of False Negatives will increase, which decreases the recall. Conversely, decreasing s increases the number of False Positives, which decreases the precision. Hence, it's all about finding the right balance. In practice we choose s so that $f1\_score(s)$ is maximised.

3.2 Tree based models: XGBoost

XGBoost is a decision-tree based algorithm that was launched in 2014 by the machine learning community *dmlc*. Since then, it has proved to be very efficient when dealing with structured (i.e. tabular) where the dataset is not so large. It has implemented many improvements that make it much more efficient than a simple classification tree. Here we present each of these improvements:

3.2.1 Growing a simple tree

First of all, let us present the Classification Tree algorithm. We will be in this case where the target variable is $Y \in \{0, 1\}$.

Algorithm 1 Building a single tree

Input

- matrix X of size $(n \times p)$ containing the data ;
- matrix $W = diag(w_1, ..., w_n)$ containing the weights for each individual ;

Iteration

for $j = 1, \dots, p$ *(for every variable)* **do**

  for $s = x^j_{(1)}, \dots, x^j_{(n)}$ *(for every possible threshold)* **do**

    - Two half-planes given by splitting the variable x^j at s:

$$R_1(j,s) = \{x_i = (x_i^1, ..., x_i^p)' \in \mathbb{R}^p \text{ s.t. } x_i^j \leq s\}$$

$$R_2(j,s) = \{x_i = (x_i^1, ..., x_i^p)' \in \mathbb{R}^p \text{ s.t. } x_i^j > s\}$$

    - Frequencies of each each class $Y = 0, 1$ of the elements of each region R_1, R_2:

$$\hat{p}_{11}(j,s) = \frac{1}{\bar{w}_1} \sum_{i \in R_1} w_i \mathbb{1}_{\{Y_i = 0\}} \quad \text{and} \quad \hat{p}_{12}(j,s) = \frac{1}{\bar{w}_1} \sum_{i \in R_1} w_i \mathbb{1}_{\{Y_i = 1\}}$$

$$\hat{p}_{21}(j,s) = \frac{1}{\bar{w}_1} \sum_{i \in R_2} w_i \mathbb{1}_{\{Y_i = 0\}} \quad \text{and} \quad \hat{p}_{22}(j,s) = \frac{1}{\bar{w}_1} \sum_{i \in R_2} w_i \mathbb{1}_{\{Y_i = 1\}}$$

    - Two Gini indexes representing the impurity of each region:

$$G_1(j,s) = \hat{p}_{11}(1 - \hat{p}_{11}) + \hat{p}_{12}(1 - \hat{p}_{12})$$

$$G_2(j,s) = \hat{p}_{21}(1 - \hat{p}_{21}) + \hat{p}_{22}(1 - \hat{p}_{22})$$

  end

end

Once we have the Gini coefficients for every variable and every threshold we choose the optimal j^* and s^* to split the node by solving:

$$(j^*, s^*) = \underset{(j,s)}{argmin}\ \bar{w}_1 G_1(j,s) + \bar{w}_2 G_2(j,s)$$

This gives two optimal regions $R_1(j^*, s^*)$ and $R_2(j^*, s^*)$, and so we can compute the labels associated to each region by using a majority condition (we take the predicted label to be the most frequent one):

$$\hat{\lambda}_k = argmax\ (\hat{p}_{k1}(j^*, s^*),\ \hat{p}_{k2}(j^*, s^*)), \quad k = 1, 2$$

$$\hat{Y}_k(j^*, s^*) = y_{\hat{\lambda}_k}(j^*, s^*), \quad k = 1, 2$$

For instance, if $\hat{Y}_1$('Years at Business', 3) $= y_1 = 0$ then it means that for this node, the variable 'Years at Business' along with the threshold 3 is the couple (j^*, s^*) that yields the minimal weighted sum of Gini indexes of the two regions, and that among the customers that have spent 3 or less years at their business (i.e. region 1), the majority class is 0.

Output The iteration presented above is what is done to build a node solely. After that it's just a matter of repeating the operation dividing the data by two at each step.

In practice, it seems that overgrowing a tree and then pruning it yields best accuracies than directly growing a smaller tree. But how large should the final tree be? A too large tree could have some risk of

overfitting, whereas if it is too small it might fail in capturing the important structure of the data. We usually try to use a penalized criterion:

$$\Lambda_\alpha(T) = \sum_{k=1}^{M_T} w_k G_k + \alpha \frac{M_T}{n},$$

where T is a subtree, M_T is the number of regions of T, G_k are Gini indexes and α is a positive parameter controlling the strenght of the second term. The first term represents the goodness of the fit of T, and the second one is a penalty punishing trees with too many leaves. The approach consists in starting with a large tree T_0, and then successively computing $\Lambda_\alpha(T)$ for every subtree, and choosing α such that $\Lambda(T)$ is minimized.

3.2.2 Boosting: we build the trees sequentially

Let us first talk about **bagging**. Bagging means Bootstrap AGGregatING, which is simply generating many subsets of the training set, learning on them, and then aggregating the results. This is the case of Random Forests, for example, which can be seen as the first evolution of the simple CART. We present the algorithm below:

Algorithm 2 Bagging principle

Input $\{(x_i, Y_i)\}_{1 \leq i \leq n}$ the training set, B the number of bags, m an integer
Iteration
for $b = 1, \dots, B$ **do**

- Sample m variables among the p: yields a new training set D_b
- Learn a classification tree T_b on D_b

end
Output Predict by averaging: $\frac{1}{B} \sum_{b=1}^{B} T_b$

Bagging is useful to prevent the model from being too exposed to the instability of a single tree. It improves accuracy because in average, when combining predictions from several trees, the best nodes appear more frequently.

The idea behind the boosting technique goes a step further. Instead of growing all the trees at the same time (using parallel processing), we sequentially grow one tree at a time, by improving its precedent version. As before, we can choose the number of predictors: that here we will denote B. There are different types of boosting techniques, the most basic one being AdaBoost. We won't present the details of AdaBoost here, but the idea is that at one iteration, the observations that were incorrectly classified will now carry more weight, meaning that they will have more importance in the following tree. XGBoost is in itself a type of boosting. The core of it is Gradient Boosting (which uses gradient Descent), but it is done with hardware enhancements (e.g. Distributed Computing, Out-of-Core Computing, Cache optimization,...) that make it faster, and especially well-suited when we have large amounts of data.

Usually gradient descent is explained when we talk about minimizing the expected loss $\mathbb{E}\left(L(y, F(x))\right)$, but in fact it much more general and used at different stages of XGBoost. Here we present it's simple theoretical version to minimize a differentiable multi-variable function:

Algorithm 3 Sequential Stochastic Gradient Descent

Input : Function $f(x),\ x \in \mathbb{R}^p$; Multiplicative coefficients $(\gamma_k)_{k\in\mathbb{N}}$
Initialization : Pick a guess for the local minimum $\theta_0 \in \mathbb{R}^p$.
Iterate :

$$\forall k \in \mathbb{N},\ \theta_{k+1} = \theta_k - \gamma_{k+1}\nabla f(\theta_k)$$

Output : $\lim_{k\to+\infty} \theta_k$

Note that when f is convex, gradient descent can converge to a global minima.

Algorithm 4 Boosting

Input $\{(x_i, Y_i)\}_{1\leq i\leq n}$ the training set

Iteration
for $t = 1, \ldots, B$ **do**

- Fit the model and get a predictor: $g_b(x)$
- Fit this model to the residuals: $\epsilon_b(x) = Y - g(x)$
- Update the predictor: $g_{b+1}(x) = g_b(x) + \epsilon_b$

end
Output Final predictor g_B

This idea of combining multiple weak learners into a single strong learner can lead to overfitting, and XGBoost is also very appreciated because it can handle regularization.

DAGNEAUX Estée
FORTÓ Guillem

References

[al02] Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research* 16 (2002), 321â357. DOI: https://arxiv.org/pdf/1106.1813.pdf.

[Rok10] Lara Lusa Rok Blagus. "Class prediction for high-dimensional class-imbalanced data". In: *BMC Bioinformatics* 1471-2105.11 (2010). DOI: https://bmcbioinformatics.biomedcentral.com/track/pdf/10.1186/1471-2105-11-523.