Imbalanced classification

Part 4: Cost-sensitive learning

By: Noureddin Sadawi, PhD

University of London

Cost-sensitive algorithms

- Use the costs as a penalty for misclassification when the algorithms are trained.
- Given that most ML algorithms are trained to minimise error, cost for misclassification is added to the error or used to weigh the error during the training process.
- This approach can be used for iteratively trained algorithms, such as logistic regression and artificial neural networks.

The Python library scikit-learn provides examples of these cost-sensitive extensions via the **class_weight** argument.

Class weights

- Usually classes and the errors for each class are considered to have the same weighting (e.g. 1.0).
- Weightings can be modified based on the importance of each class.
- The weighting is applied to the loss so that smaller weight values result in a smaller error value, and in turn, less update to the model coefficients.
- A larger weight value results in a larger error calculation, and in turn, more update to the model coefficients.
- **Small weight: less importance, less update to the model coefficients.**
- **Large weight: more importance, more update to the model coefficients.**

Cost-sensitive decision tree

- **Cost-sensitive decision trees gained a great deal of attention.**
- The split points of the tree are chosen to best separate examples into two groups with minimum mixing.
- When both groups are dominated by examples from one class, the criterion used to select a split point will see good separation, when in fact, the examples from the minority class are being ignored.
- This can be overcome by modifying the criterion used to evaluate split points to take the importance of each class into account.

Cost-sensitive decision tree

- In a DT we are looking for a clean separation of examples into groups where a group of examples of all 0 or all 1 class is the **purest**, and a 50–50 mixture of both classes is the **least pure.**
- The calculation of a purity measure involves calculating the probability of an example of a given class being misclassified by a split.
- Calculating these probabilities involves summing the number of examples in each class within each group.
- The splitting criterion can be updated to take the purity of the split into account as well as the importance of each class (**using weights**).

Cost-sensitive logistic regression

- LR is an effective model for binary classification tasks, although by default, it is not effective for imbalanced classification (but it can be modified).
- The coefficients of the LR algorithm are fit using an optimisation algorithm that minimises the negative log likelihood (loss) for the model on the training dataset.
- This involves the repeated use of the model to make predictions followed by an adaptation of the coefficients in a direction that reduces the loss of the model.
- The calculation of the loss for a given set of coefficients can be modified to take the class balance into account.

Cost-sensitive logistic regression

- By default, the errors for each class may be considered to have the same weighting, say 1.0 (these weightings can be adjusted based on the importance of each class).
- The weighting is applied to the loss so that smaller weight values result in a smaller error value, and in turn, less update to the model coefficients.
- A larger weight value results in a larger error calculation, and in turn, more update to the model coefficients.
- The weightings are sometimes referred to as importance weightings.
- The challenge of weighted LR is the choice of the weighting to use for each class.

How to assign class weights

- Domain expertise, gained by speaking to domain experts.
- Tuning, determined by a hyperparameter search such as a grid search.
- Heuristic, specified using a general best practice.
- A common practice is to use the inverse of the class distribution present in the training dataset.
  - Example: 100 instances in class 0 and only 1 instance in class 1.
  - 1:100 = 0.5:50.
  - Class 0 can have a weight value of 0.5.
  - Class 1 can have a weight value of 50.

How to assign class weights

When a class i is assigned a weight value, that value should be proportional to the cost of misclassifying an instance of class i

- A common practice is to use this method:
  Weight of class i = Total No of Samples / (No of Classes * No of instances in class i).
- For example: 1000 instance, 993 in class 0, and 7 in class 1
- The weighting for class 0 = 1000 / (2 * 993) = 0.5.
- The weighting for class 1 = 1000 / (2 * 7) = 71.