handling learnability imbalance in multiclass-classification

Theodor Peifer
email: thp7219@thi.de
Technische Hochschule Ingolstadt

***Abstract*—Neural Networks have proven themselves to be powerful classification tools by solving problems in a range of domains with high accuracy. Yet this accuracy is never evenly distributed across all classes, which means that the true-positive rates of each class separately are different. This can happen even in balanced datasets since some classes are more difficult to learn by the model than others (this phenomenon is further referred to as *learnability-imbalance*). A common way to address this problem is to give a weight to the error function for each class to penalize losses of certain classes higher or lower. This research will address the determination of such weights to counteract the learnability-imbalance in balanced datasets using previously calculated evaluation scores. Therefore the goal is to find methods to lower the variance of the true positive rates of each class.**

I. Introduction

A frequent problem in classification appears when working with datasets that have an unequal amount of samples per class and are therefore called a *class imbalanced* datasets. Since there will be some classes, that have less elements for the model to learn from, their features will be harder to extract what finally will result in a lower true positive rate, i.e. a per-class accuracy [1]. Thus, the consequence of having different class sizes can be described as having a *learnability imbalance* in the dataset since some classes are more difficult to learn that others.

But such learnability differences can appear also in balanced datasets for a variety of reasons, e.g. when the quality of the data of a class is lower than the rest of the data. A second reason, that this research will be focused on, is that when some classes are similiar, the model can confuse their samples with each other more easily what will often result in a lower accuracy of those classes.

Even though this issue is an inevitable product of every normal classification, in most cases the learnability difference of the classes is either low or not from great interest. But there can be more extreme cases where the model needs to produce fair and unbiased results using a dataset that has an obvious learnability imbalance. An example is *name-ethnicity classification*, where a model predicts the ethnicity of a name only by its letters [5]. Nationalities that speak the same language and therfore have similiar names (e.g. *british* and *american*) result in a lower accuracy (see figure 1) which therefore can lead to wrong or unfair interpretations when the model is used for social sciences experiments.

Another dataset that is a good showcase for the learnability imbalance is the CIFAR-10 [2] dataset (32×32 RGB images of ten different classes) because it also contains similiar classes such as *dog* and *cat*. When looking at the true positive rates produced by a convolutional neural network [3], trained on that dataset, it can be seen that there are some classes that got misclassified more often (in this case bird, cat, deer and dog) which hints to a more difficult learnability. The CIFAR-10 and the name-nationality dataset will be used for experiments in order to find methods for minimizing such variances in the evaluation scores.

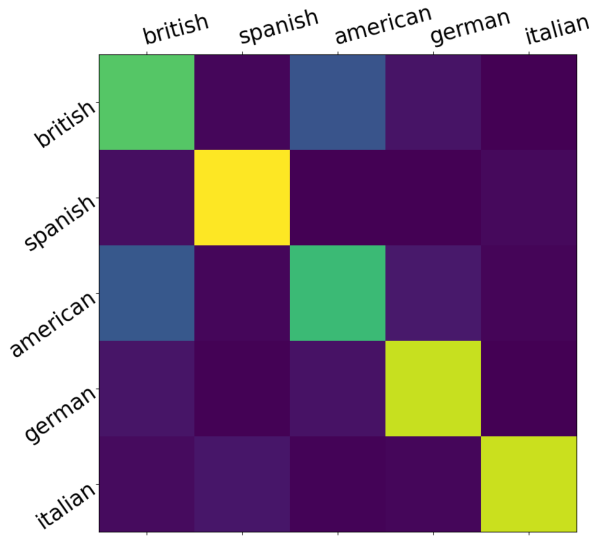

Fig. 1. confusion matrix representing the true positive distribution of the name-nationality dataset produced by a recurrent neural network

To figure out such methods one should first go back to the class imbalance problem because there are already solutions existing: It is common to weight the error function [1] according to the size of each class, i.e. the number of samples it contains. Therefore, for every class there is a weight, which is greater the fewer elements it contains and that gets multiplied with the loss produced by its samples. That will cause, that samples from a smaller class will produce a higher error and, since the aim of a neural network is to minimize the loss [4], will have a higher impact of the learning process. Another way to try to compensate for the different class sizes is to use more augmentation [5] on the samples of smaller classes. Augmentation describes the random transformation

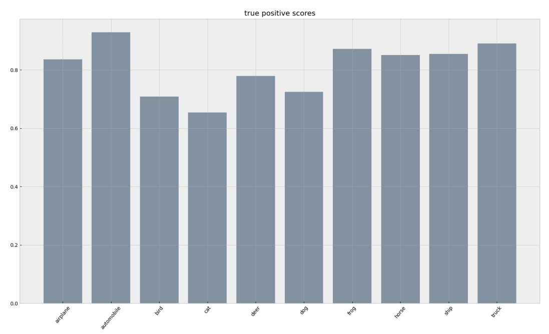

Fig. 2. true positive scores of a model trained on the CIFAR-10 dataset

of the samples, in order to synthetically generate more input data for the model. For example, in image classification it is common to randomly flip, mirror or to add white noise to the images [6]. This can help the model generalizing better. But since both methods rely on the different proportions of the classes in the dataset, the question arises how to apply them on datasets that, even though they don't suffer from class imbalance, still show a big learnability imbalance. When working with such datasets the learnability differences of the individual classes are mostly only identifiable after the model has been trained and evaluated normally. A confusion matrix or the calculation of the true positive rates can then reveal what classes where learned the best and which should have received a higher weight. These rates can be used to determine which loss weights or how much data augmentation should be used for each class.

Algorithm 1 creating loss weights for a balanced dataset

C $\hat{=}$ *amount of classes*

train(weights) *describes the initialization and training process of a classification model in which* $weights_i$ *will be multiplied to every loss generated by a sample of class* i.

evaluate() *creates the set* s *with* $|s| = C$ *and* $s_i \in [0;1]$ *which contains the true positive scores of all classes of the test dataset*

W(s) *is a function that creates a set of loss-weights* w *with* $|w| = C$ *and* $w_i \in [0;1]$ *using a set of true positive scores s:* $W : s \rightarrow w$ *;* $\{s_i, ..., s_C\} \rightarrow \{w_i, ..., w_C\}$

process:

```
1: $train(weights = \{w_1 = 1, w_2 = 1, ..., w_c = 1\})$
2: $s = evaluate()$
3: $w = W(s)$
4: $train(weights = w)$
5: $s_{new} = evaluate()$
6: $compare(s, s_{new})$
```

II. State of the Art

TODO first section here

III. Approach

TODO second section here

IV. Third Section

TODO third section here

References

[1] Name Name, Name Name, Name Name (2006). Title, 24(1), 29-33.