Automated Machine Learning for Imbalanced Binary Classification

Stefan Solarski[1]

[1]Ludwig Maximilian University of Munich

Abstract This project focuses on the problem of imbalanced classification which is a special case of regular classification. The datasets used contain a disproportionate distribution of classes, making it difficult to build a traditional machine learning model which will accurately predict the minority class. To address this issue we have developed an AutoML pipeline which leverages resampling techniques and special balanced algorithms which are adapted to work with skewed distributions. It consists of preprocessing, feature engineering, resampling, and hyperparameter tuning using Bayesian optimization. The model was evaluated on 10 datasets from obtained openml and outperforms a standard random forest for every dataset.

1 Introduction

Imbalanced datasets pose a big challenge to standard algorithms, since many of these algorithms rely on the assumption that the classes in the dataset are well balanced. This is a critical issue in many real-world scenarios such as fraud detection, medical diagnosis, and anomaly detection, where the minority class often represents the target class.

There are three main ways to address this issue: data-level methods, algorithm-level methods and hybrid methods (Krawczyk, 2016).

1.1 Data-level Methods

When working with imbalanced datasets we need a way to balance the classes if we want to use standard machine learning algorithms.

With respect to **undersampling**, the simplest method is random undersampling which just randomly samples datapoints from the majority class until the desired class balance is reached. An alternative to random undersampling is Tomek Links, a method that identifies samples from the majority and minority classes that are closest to each other and removes the majority class sample. The goal is to increase the margin between the classes by removing samples that are close to the boundary region. However, both of these methods lead to the removal of datapoints and thus possible information loss and underfitting.

Oversampling methods, on the other hand, balance the classes by generating synthetic samples from the underrepresented class to increase the sample size and help the algorithms learn to distinguish it. One of the most popular oversampling methods is Synthetic Minority Over-sampling Technique (SMOTE) which generates new samples by interpolating between the minority class samples thus creating new datapoints that are combinations of the existing ones. This is often effective in regions where the minority class is sparse. This method can lead to overfitting if the synthetic generation is not done correctly (Burnaev et al., 2015).

Hybrid resampling methods use a combination of the two previous techniques to minimize their weaknesses. A popular hybrid method is SMOTE-TL which combines the SMOTE oversampling with Tomek Links undersampling (Burnaev et al., 2015). By doing so, we can create a more balanced dataset that is less prone to overfitting and more representative of the underlying distribution.

Since all of these methods present individual pros and cons we ran experiments using all three ideas. These will be discussed in the following section.

1.2 Algorithm-level Methods

Algorithm-level methods are ways to modify standard algorithms such as random forest and XGBoost to adapt them for imbalanced classification. A popular solution is using class weighing. This involves assigning higher weights to the minority class during training to give it more importance in the decision-making process. This can help the algorithm not be totally biased towards the majority class. Another option is cost-sensitive learning. This means assigning different cost to different classification errors. For example, misclassifiying a minority class should have a higher cost than misclassifying a majority class. This incentivize the algorithm to focus on correctly classifying the minority class.

1.3 Hybrid-level Methods

There are also specific algorithms that are built with imbalanced datasets in mind. These algorithms are modifications of standard algorithms which enable resampling in-between iterations to address the class imbalance. Two popular hybrid algorithms are Balanced Random Forest and RUS Boost.

Balanced Random Forest improves perfrormance on imbalaced data by using two main techniques: random undersampling and bootstrap aggregating (Chen and Breiman, 2004) Combining these two, we apply random undersampling to each subset of the data during the bootstrap aggregation. This methods performs very well, especially when the class imbalance is severe.

Random undersampling Boosting (RUSBoost) combines random undersampling with AdaBoost as a base classifier. Again, the main idea is that each iteration starts with random undersampling after which the regular boosting iteration takes place. This algorithm achieves comperable results to the balanced random forest and is another good tool to have in our toolbelt.

2 Experiments

In this section, we will go through all the experiments conducted and explain both the findings and conclusions. The experiments were run on a laptop with a 4-core i7 CPU, 16GB of RAM and no support for CUDA GPU acceleration. The scoring was done with balanced accuracy.

2.1 Standard algorithms

The goal of the project is to build an AutoML system which outperforms a random forest using balanced accuracy as a metric. To begin the experiments we ran a random forest and a decision tree on all the datasets to get baseline measurements. All the results from the experiments are collected in Table 1. Immediately, out of the 10 datasets we could detect two separate groups of datasets, ones that the classifier easily scores over 90% on and others that lead to scores in the range of 60% to 70% with the random forest. Interestingly, the decision tree algorithm outperformed the random forest on 5 of the 10 datasets, specifically the more difficult ones to classify. On the other 5 datasets, both algorithms performed really well, with scores in the 90s. Next, we decided to add a boosting model and decided to try both XGBoost and CatBoost. These models outperformed the random forest in almost every dataset. However, they both still performed worse than the decision tree on 4 datasets. We did not continue using CatBoost because of its, approximately 8 times longer, fitting time compared to XGBoost, and only slight differences in results.

2.2 Resampling

Thereafter, we began to experiment with resampling methods. Specifically, we combined the XGBoost algorithm with the random undersampling, SMOTE and SMOTE Tomek algorithms. For different resampling techniques improved balanced accuracies for different datasets. Example, the datasets where the decision tree was dominating (the ones with low balanced accuracy) were most improved when running random undersampling. This was a ver surprising result since we expect random undersampling to lead to information loss and underfitting. SMOTE Tomek improved

Figure 1: Classifiers with default parameters on all datasets

	976	980	1002	1018	1019	1021	1040	1053	1461	41160
Decision Tree	0.948524	0.917299	0.614408	0.631854	0.967682	0.912711	0.952086	0.608031	0.699523	0.692272
Random Forest	0.962148	0.938651	0.547442	0.554433	0.988729	0.932494	0.960291	0.593875	0.695272	0.570881
XGBoost	0.979616	0.962724	0.577647	0.590694	0.991709	0.926929	0.981293	0.590267	0.722712	0.669650
XGBoost SMOTE	0.984384	0.974540	0.589217	0.603487	0.994645	0.939245	0.987767	0.606475	0.729861	0.669405
XGBoost SMOTETomek	0.984194	0.974540	0.596477	0.608257	0.994645	0.942010	0.987767	0.621131	0.740505	0.668629
XGBoost RandomUnderSampler	0.979647	0.978530	0.777127	0.793387	0.987943	0.954750	0.990335	0.655834	0.855513	0.818895
Random Forest SMOTE	0.977550	0.956805	0.597810	0.589607	0.990110	0.947861	0.985133	0.636282	0.720188	0.590460
XGBoost Weighted	0.988593	0.985105	0.529276	0.539355	0.994189	0.892660	0.990528	0.597424	0.846119	0.567952
Random Forest Weighted	0.964816	0.930662	0.539498	0.537498	0.983953	0.924280	0.976257	0.587361	0.658417	0.559600
RUSBoost	0.953546	0.947335	0.762507	0.784363	0.981539	0.861029	0.933950	0.578808	0.826289	0.708962
Balanced Random Forest	0.981584	0.980960	0.803348	0.814811	0.986684	0.955771	0.990380	0.680226	0.855337	0.765792
Balanced Random Forest KNNImputer	0.981584	0.980960	0.803348	0.814811	0.986684	0.955771	0.990380	0.679067	0.855337	0.756372
XGBoost RandomUnderSampler KNNImputer	0.979647	0.978530	0.777127	0.793387	0.987943	0.954750	0.990335	0.661616	0.855513	0.820606
Voting Classifier	0.980676	0.980808	0.792857	0.808168	0.988279	0.958519	0.990326	0.672176	0.862323	0.818703

the accuracy in datasets that were already performing well, with scores over the 90% mark. Thus, mixing XGBoost with random undersamping and XGBoost with SMOTE Tomek were the best algorithms for the job.

2.3 Weighing

We attempted to also weight the classes as explained in the previous chapter. Using class weighing on XGBoost further improved the results on the 5 datasets that were already showing good performance. However, this combination performed much worse on datasets that were already showing poor performance. We did not continue with this idea as it was not balanced enough to perform well on all datasets.

2.4 Hybrid Approach

We also tested hybrid approaches, namely Balanced Random Forest and RUSBoost. The Balanced Random Forest performed very well across all the datasets and seemed to be the most balanced algorithm so far. The RUSBoost was underwhelming, underperforming the Balanced Random Forest on every dataset.

2.5 Ensemble

Finally, we chose the Balanced Random Forest and XGBoost classifiers with Random Under Sampling as the best two algorithms and built an ensemble out of the two. The ensemble used a soft voting technique and performed similarly to the best results of each of the algorithms, even outperforming them in some of the tests.

3 The pipeline

The conclusions drawn from the previous experiments are expanded and further explored in the final AutoML pipeline. The pipeline is built using the imbalanced-learn Pipeline object enables the use of the aforementioned resampling techniques. The main components in the pipeline are: imputation, feature selection, resampling, and the Bayesian hyperparameter optimization over the previous 3 components and the chosen algorithm.

In the first step, we **impute** the missing values, either by using the sklearn's SimpleImputer or KNNImputer with 5 neighbours. Our experiments showed that their performance is rather similar, which can be seen in table 1.

Feature selection is based on sklearn VarianceThreshold which removes all the features which have very low variance. The parameter is tuned with BO.

Resampling uses imblearn resampling methods which implement exactly what we described in the sections above. This parameter is tuned in all algorithms except the balanced random forest.

The **classifier** is a choice between the Balanced Random Forest, XGBoost, the ensemble of the two, and decision trees as a baseline. If the pipeline is set on 'auto', whichever classifier performs the best with the default parametrs, will be further tuned using Bayes optimization.

And the **tuning** for all the elements above is done with sklearn-optimization using BayesSearchCV which does a Bayesian optimization over the defined search space. The pipeline has a stop time parameter which defines how long the Bayesian optimization should run for. We also attempted implementing multi-fidelity, both with the hpbanster and smac3 library, however both ran into compatibility issues, server instances breaking and missing some C++ redistributables.

4 Results

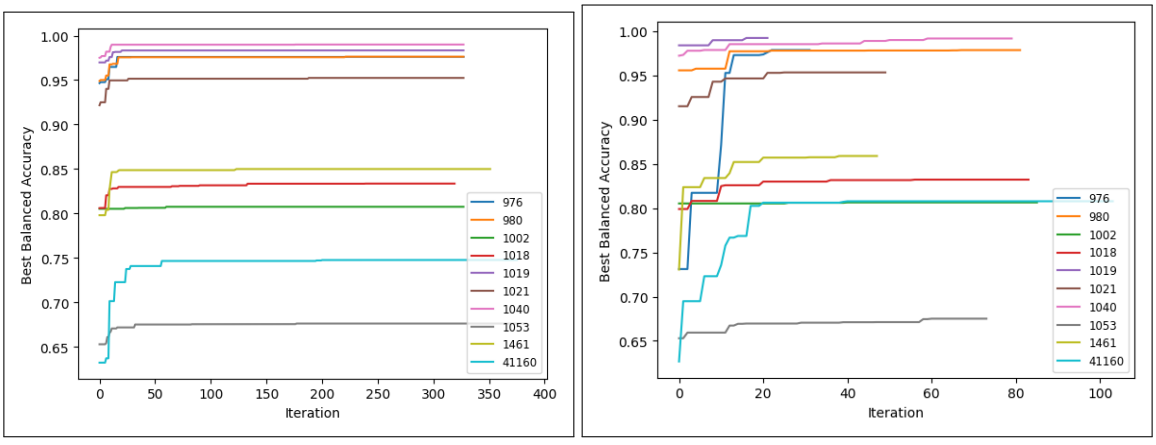
The evaluation was done with nested cross validation with 3-fold stratified cross validation for both the inner and outer evaluation. The AutoML pipeline optimizes the balanced accuracy for each one of the datasets by selecting the best classifiers, imputers, resamplers and hyperparameters for all of them. The final results are in the following table 2

Figure 2: Comparing AutoML system with baseline using balanced accuracy on all the datasets

	976	980	1002	1018	1019	1021	1040	1053	1461	41160
Decision Tree	0.948524	0.919832	0.606276	0.625605	0.968910	0.916949	0.953058	0.612182	0.701607	0.692096
Random Forest	0.961218	0.936109	0.545616	0.556767	0.989452	0.929235	0.963755	0.588576	0.694067	0.570503
AutoMLPipeline	0.984293	0.985962	0.805042	0.809523	0.994855	0.961816	0.991725	0.680712	0.853007	0.778498

The AutoML system outperforms both the decision tree and random forest on every dataset achieving the desired result. Below, we have two plots, the left one showing the tuning of the AutoML system using only Balanced Random Forest and the right one using the 'auto' setting which enables automatic selection of the best classifier within the pipeline.

Figure 3: Balanced Accuracy on each iteration of Balanced Random Forest for each dataset



The tuning for all datasets has been ran for 20 minutes. However, some datasets have gone through more iterations while others through less. This is explained by two factors: the size and complexity of the dataset and the complexity of the classifier that was tuned. Some of the datasets were fitted with the ensemble model, which takes approximately twice as long to tune.

References

Burnaev, E., Erofeev, P., and Papanov, A. (2015). Influence of resampling on accuracy of imbalanced classification. *Proc SPIE, Eighth International Conference on Machine Vision, 987525 (December 8, 2015)*, 9875:P. 5.

Chen, C. and Breiman, L. (2004). Using random forest to learn imbalanced data. *University of California, Berkeley*.

Krawczyk, B. (2016). Learning from imbalanced data: Open challenges and future directions. *Progress in Artificial Intelligence*, 5.