Statistical Computing Proposal

Mark H. White II — markhwhiteii@gmail.com

Introduction

Many applications of classification problems in machine learning involve class imbalance—a situation where the class of interest (the "minority" or "positive" class) makes up a very small percentage of the total data (i.e., 5% or less of cases are in this class). Algorithms that try to maximize the overall accuracy of prediction bias in favor of the majority (or "negative") class. For example, if 1% of cases are classified as "positive," then an algorithm can ensure 99% accuracy by simply predicting *all* cases as "negative." This obviously misses the whole point of training a model in the first place; we are training the model explicitly because we want to be able to predict this class or event with a low base rate. Class imbalance problems arise in important applications: Will a contact lead to a donation to our campaign? Does this person have cancer? Is this credit card transation fraudulent?

There are a number of ways to address class imbalance in two-class classification problems [1-3]. When people propose these methods or review a number of them, they generally use real-world data sets to see how their methods perform [3-5]. While this adds the benefit of ecological validity, it does not allow us to see systematic relationships between characteristics of the data and performance. These reviews can yield conclusions like, "Method X outperformed Method Y when looking at the mean of their F1 scores across 40 different datasets." I want to take a Monte Carlo approach, systematically varying aspects about the data across 1,000 simulations; this will allow me to make conclusions like, "The AUROC for Method X is positively correlated with sample size, while there is no correlation for Method Y."

Proposed Methods

I describe the proposed methods in two parts. First, I will describe how I will generate the imbalanced data, varying important characteristics of the data; second, I will describe which methods and models I will test on these data.

Data Characteristics

I will generate 1,000 data sets and test each of the models on every data set. I will vary certain characterists of the data. These characteristics will be drawn randomly from a Gaussian distribution, so that I can perform regression analyses to see the relationship between accuracy and characteristics of the data. The data will vary by:

1. The number of cases. The mean of this distribution will be 5,005,000 with a standard deviation of 1,665,000. This is set so that greater than 99% of the draws will lie between 10,000 and 10,000,000 cases.
2. The number of predictors. The mean of this distribution will be 45 with a standard deviation of 10. This is set so that greater than 99% of the draws will lie between 15 and 75 predictors.
3. The proportion of predictors that are noise. The mean of this distribution will be .20 with a standard deviation of .033. This is set so that greater than 99% of the proportion of noise predictors lie between .10 and .30.
4. The proportional size of the positive class. The mean of this distribution will be .03 with a standard deviation of .006 so that greater than 99% of the proportional size of the positive class will lie between .012 and .048.

I believe that these characteristics allow me to mimic the characteristics of big datasets without being too computationally expensive. Compared to another review looking at ways to handle class imbalance [3], these simulated datasets will be much larger and have much more class imbalance.

Methods and Models

Previous reviews [1-3] suggest a number of ways to handle class imbalance. Approaches might focus on processing the data before training (e.g., sampling techniques), adjusting or employing specific algorithms during training (e.g., adjusting a cost matrix), or processing predictions after training (e.g., lowering the probability threshold to deem a case "positive"). Some of the approaches might depend on the specific applied situation in which an analyst finds herself (for example, the cost of calling someone who does not donate to a campaign can be worked into the weighting of a loss function or the adjustment of a cost matrix). Therefore, I will focus on general approaches.

Galar and colleagues [3] find that ensemble methods, paired with data pre-processing, performed well for class imbalance. Unfortunately, some of the algorithms they employed are not widely-available and easily-implementable in R packages or standard Python libraries yet. I focus on using algorithms that are in R packages. I will vary two factors, making for a fully-crossed experimental design.

Factor 1: Sampling Technique

Sampling techniques generally involve (a) undersampling, or dropping cases from the majority class to achieve balance, (b) oversampling, or bootstrapping cases from the minority class to achieve balance, or (c) creating synthetic positive class cases. The approaches I will use, and their associated R package and function, are:

1. Random undersampling, `unbalanced::ubUnder()`.
2. Random oversampling, `unbalanced::ubOver()`.
3. SMOTE (synthetic minority oversampling technique), `unbalanced::ubSMOTE()`. There are many flavors of the SMOTE algorithm available in the `smotefamily` package; if it will not be too computationally intensive, I would like to investigate other variants such as Borderline SMOTE, `smotefamily::BLSMOTE()`.
4. No sampling, a control condition.

Factor 2: Algorithm

I will focus on tree-based methods, primarily using ensemble methods. Two popular ensemble methods are boosting and bagging. Boosting is an approach whereby a model will train a number of times in serial; each iteration, the model focuses more and more on incorrectly classified cases (or residuals). Bagging is an approach whereby a number of decision trees are trained on different bootstrap samples of a dataset, and a random forest extends this further by only including a random subset of predictor variables. The approaches I will use, and their associated R package and function, are:

1. AdaBoost, `fastAdaboost::adaboost()`.
2. Extreme Gradient Boosting, `xgboost::xgb,train()`.
3. Random Forest, `randomForest::randomForest()`.
4. C5.0, a single decision tree as a control condition, `C50::C5.0()`.

Proposed Analyses

As my dependent variables, I will look at: F1 scores, precision, recall, the area under an ROC curve, and overall accuracy as a comparison to these. This means that, for each of the 1,000 datasets, I will run 16 models and save 5 scores from each model. To save on computing time, accuracy will be judged by training the data on 70% of the sample and testing it on the other 30% (instead of using k-fold cross-validation). For each of these models, I will also save the four varying characteristics of the dataset.

From this "results" data frame, I will be able to look at (a) which model performed the best overall, and (b) relationships between sample characteristics and model performance.

References

1. Branco, P., Torgo, L., & Ribeiro, R. (2015). A survey of predictive modelling under imbalanced distributions. Retrieved from https://arxiv.org/abs/1505.01658.
2. Longadge, R., Dongre, S., & Malik, L. (2013). Class imbalance problem in data mining: Review. Retrieved from https://arxiv.org/abs/1305.1707.
3. Galar, M., Fernández, A., Barrenechea, E., Bustince, H., & Herrera, F. (2012). A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches. *IEE Transactions on Systems, Man, and Cybernatics—Part C: Applications and Reviews, 42*(4), 463-484.
4. Weiss, G. M., McCarthy, K., & Zabar, B. (2007). Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs? *DMIN, 7*, 35-41.
5. Chawla, N., Bowyer, K., Hall, L., & Kegelmeyer, W. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research, 16*, 321-357.