Exploratory Undersampling for Class-Imbalance Learning

Abstract - Undersampling is a popular method in dealing with class-imbalance problems, which uses only a subset of the majority class and thus is very efficient. The main deficiency is that many majority class examples are ignored. We propose two algorithms to overcome this deficiency. *EasyEnsemble* samples several subsets from the majority class, trains a learner using each of them, and combines the outputs of those learners. *BalanceCascade* trains the learners sequentially, where in each step, the majority class examples that are correctly classified by the current trained learners are removed from further consideration. Experimental results show that both methods have higher Area Under the ROC Curve, F-measure, and G-mean values than many existing class-imbalance learning methods. Moreover, they have approximately the same training time as that of undersampling when the same number of weak classifiers is used, which is significantly faster than other methods.

Index Terms - Class-imbalance learning, data mining, ensemble learning, machine learning, undersampling

I. INTRODUCTION

In many real-world problems, the data sets are typically imbalanced, i.e., some classes have much more instances than others. The level of imbalance (ratio of size of the majority class to minority class) can be as huge as 10^6. It is noteworthy that class imbalance is emerging as an important issue in designing classifiers.

Imbalance has a serious impact on the performance of classifiers. Learning algorithm that do not consider class imbalance tend to be overwhelmed by the majority class and ignore the minority class. For example, in a problem with imbalance level of 99, a learning algorithm that minimizes error rate could decide to classify all examples as the majority class in order to achieve a low error rate of 1%. However, all minority class examples will be wrong classified in this case. In problems where the imbalance level is huge, class imbalance must be carefully handled to build a good classifier.

Class imbalance is also closely related to cost-sensitive learning, another important issue in machine learning. Misclassifying a minority class instance is usually more serious than misclassifying a majority class one. For example, approving a fraudulent credit card application is more costly than declining a credible one. Breiman et al. pointed out that training set size, class priors, cost of errors in different classes, and placement of decision boundaries are all closely connected. In fact, many existing methods for dealing with class imbalance rely on connections among these four components. Sampling methods handle class imbalance by varying the minority and majority class sizes in the training set. Cost-sensitive learning deals with class imbalance by incurring different costs for the two classes and is considered as an important class of methods to handle class imbalance. More details about class-imbalance learning methods are presented in Section II.

In this paper, we examine only binary classification problems by ensembling classifiers built from multiple under sampled training sets. Undersampling is an efficient method for class-imbalance learning. This method uses a subset of the majority class to train the classifier. Since many majority class examples are ignored, the training set becomes more balanced and the training process becomes faster. However, the main drawback of undersampling is that potentially useful information contained in these ignored examples is neglected. The intuition of our proposed methods is then to wisely explore these ignored data while keeping the fast training speed of understanding.

We propose two way to use these data. One straightforward way is to sample several subsets independently from $\mathcal{N}$ (the majority class), use these subsets to train classifiers separately, and combine the trained classifiers. Another method is to use trained classifiers to guide the sampling process for subsequent classifiers. After we have trained n classifiers, examples correctly classified by them will be removed from $\mathcal{N}$. Experiments on 16 UCI data sets show that both methods have higher Area Under the receiver operating characteristics (ROC) Curve (AUC), F-measure, and G-mean values than many existing class-imbalance learning methods.

III. EasyEnsemble AND BalanceCascade

As was shown by Drummond and Holte, undersampling is an efficient strategy to deal with class imbalance. However, the drawback of undersampling is that it throws away many potentially useful data. In this section, we propose two strategies to explore the majority class examples ignored by undersampling: *EasyEnsemble* and *BalanceCascade*.

A. EasyEnsemble

Given the minority training set $\mathcal{P}$ and the majority training set $\mathcal{N}$, the undersampling method randomly samples a subset $\mathcal{N}'$ from $\mathcal{N}$, where $|\mathcal{N}'| < |\mathcal{N}|$. Usually, we choose $|\mathcal{N}'| = |\mathcal{P}|$ and therefore have $|\mathcal{N}'| \ll |\mathcal{N}|$ for highly imbalanced problems.

EasyEnsemble is probably the most straightforward way to further exploit the majority class examples ignored by undersampling, i.e., examples in $\mathcal{N} \cap \bar{\mathcal{N}}'$. In this method, we independently sample several subsets $\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_T$ from $\mathcal{N}$. For each subset $\mathcal{N}_i$ ($1 \le i \le T$), a classifier H_i is trained using $\mathcal{N}_i$ and all of $\mathcal{P}$. All generated classifiers are combined for the final decision. AdaBoost is used to train the classifier H_i. The pseudocode for *EasyEnsemble* is shown in Algorithm 1.

Algorithm 1 The *EasyEnsemble* algorithm

```
1: {Input: A set of minority class examples $\mathcal{P}$, a set of majority class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, the number of subsets $T$ to sample from $\mathcal{N}$, and $s_i$, the number of iterations to train an AdaBoost ensemble $H_i$}
2: $i \Leftarrow 0$
3: **repeat**
4: $i \Leftarrow i + 1$
5: Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = \mathcal{P}$.
6: Learn $H_i$ using $\mathcal{P}$ and $\mathcal{N}_i$. $H_i$ is an AdaBoost ensemble with $s_i$ weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is $\theta_i$, i.e.,
```

$$H_i(x) = sgn\Big(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i\Big) \quad (1)$$

```
7: **until** $i = T$
8: Output: An ensemble
```

$$H(x) = sgn\Big(\sum_{i=1}^{T}\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{T} \theta_i\Big) \quad (2)$$

The idea behind *EasyEnsemble* is simple. Similar to the Balanced Random Forests, EasyEnsemble generates T balanced subproblems. The output of the ith subproblem is AdaBoost classifier H_i, an ensemble with s_i weak classifiers $\{h_{i,j}\}$. An alternative view of $h_{i,j}$ is to treat it as a feature that is extracted by the ensemble learning method and can only take binary values. H_i, in this viewpoint, is simply a linear classifier built on these features. Features extracted from different subsets $\mathcal{N}_i$ thus contain information of different aspects of the original majority training set $\mathcal{N}$. Finally, instead of counting votes from $\{H_i\}_{i=1,...,T}$, we collect all the features $h_{i,j}(i = 1, 2, ..., T, j = 1, 2, ..., s_i)$ and form an ensemble classifier from them.

The output of *EasyEnsemble* is a single ensemble, but it looks like an "ensemble of ensembles". It is known that boosting mainly reduces bias, while bagging mainly reduces variance. Several works, combine different ensemble strategies to achieve stronger generalization. MultiBoosting, combines boosting with bagging/wagging by using boosted ensembles as base learners. Stochastic Gradient Boosting and Cocktail Ensemble also combine different ensemble strategies. It is evident that *EasyEnsemble* has benefited from the combination of boosting and a bagging-like strategy with balanced class distribution.

Both *EasyEnsemble* and Balanced Random Forests try to use balanced boostrap samples; however, the former uses the samples to generate boosted ensembles, while the latter uses the samples to train decision trees randomly. Costing also uses multiple samples of the original training set. Costing was initially proposed as a cost-sensitive learning method, while *EasyEnsemble* is proposed to deal with class imbalance directly. Moreover, the working style of *EasyEnsemble* is quite different from costing. For example, the costing method samples the examples with probability in proportion to their costs (rejection sampling). Since this is a probability-based sampling method, no positive example will definitely appear in all the samples (in fact, the probability of a positive example appearing in all the samples is small). While in *EasyEnsemble*, all the positive examples will definitely appear in all the samples. When the size of minority class is very small, it is important to utilize every minority class example.

B. BalanceCascade

EasyEnsemble is an unsupervised strategy to explore $\mathcal{N}$ since it uses independent random sampling with replacement. Our second algorithm, BalanceCascade, explores $\mathcal{N}$ in a supervised manner. The idea is as follows. After H_1 is trained, if an example $x_1 \in \mathcal{N}$ is correctly classified to be in the majority class by H_1, it is reasonable to conjecture that x_1 is somewhat redundant in $\mathcal{N}$, given that we already have H_1. Thus, we can remove some correctly classified majority class examples from $\mathcal{N}$. As in EasyEnsemble, we use AdaBoost in this method. The pseudocode of BalanceCascade is described in Algorithm 2.

```
Algorithm 2  The BalanceCascade algorithm
1: {Input: A set of minority class examples $\mathcal{P}$, a set of majority class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, the number of subsets $T$ to sample from $\mathcal{N}$, and $s_i$, the number of iterations to train an AdaBoost ensemble $H_i$}
2: $i \Leftarrow 0$, $f \Leftarrow \sqrt[T-1]{|\mathcal{P}|/|\mathcal{N}|}$, $f$ is the false positive rate (the error rate of misclassifying a majority class example to the minority class) that $H_i$ should achieve.
3: repeat
4:   $i \Leftarrow i + 1$
5:   Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$.
6:   Learn $H_i$ using $\mathcal{P}$ and $\mathcal{N}_i$. $\mathcal{H}_i$ is an AdaBoost ensemble with $s_i$ weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$.
     The ensemble's threshold is $\theta_i$ i.e.,
     $$H_i(x) = sgn\Big(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i\Big) \quad (3)$$
7:   Adjust $\theta_i$ such that $\mathcal{H}_i$'s false positive rate is $f$.
8:   Remove from $\mathcal{N}$ all examples that are correctly classified by $\mathcal{H}_i$.
9: until $i = T$
10: Output: A single ensemble
     $$H(x) = sgn\Big(\sum_{i=1}^{T}\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{T}\Big) \quad (4)$$
```

This method is called BalanceCascade since it is somewhat similar to the cascade classifier. The majority training set $\mathcal{N}$ is shrunk after every H_i is trained, and every node H_i is dealing with a balanced subproblem ($|\mathcal{N}_i| = |\mathcal{P}|$). However, the final classifier is different. A cascade classifier is the conjunction of all $\{H_i\}_{i=1,\ldots,T}$, i.e., $H(x)$ predicts positive if and only if all $H_i(x)$ ($i = 1, 2, \ldots, T$) predicts positive. Viola and Jones used the cascade classifier mainly to achieve fast testing speed. While in BalanceCascade, sequential dependence between classifiers is mainly exploited for reducing the redundant information in the majority class. This sampling strategy leads to a restricted sample space for the following undersampling process to explore as much useful information as possible.

BalanceCascade is similar to EasyEnsemble in their structures. The main difference between them is the lines 7 and 8 of Algorithm 2. Line removes the true majority class examples from $\mathcal{N}$, and line 7 specifies how many majority class examples can be removed. At the beginning of the Tth iteration, $\mathcal{N}$ has been shrunk $T-1$ times, and therefore, its current size is $|\mathcal{N}| \cdot f^{T-1} = |\mathcal{P}|$. Thus, after H_T is trained and $\mathcal{N}$ is shrunk again, the size of $\mathcal{N}$ is smaller than $|\mathcal{P}|$. We can stop the training process at this time.