ISEN 689: Course Project

Imbalanced Classification

A Credit Card Fraud Detection Example

Dhananjay Patil

1.0 Abstract

Many of the classification problems, such as Fraud Detection, Product Categorization, Disease diagnosis, suffer from the problem of class imbalance, i.e. one class may have more number of instances than the others. If the original data is used to train the classifier, then this might lead to serious performance issues as the classifier might get overwhelmed by the majority class and ignore the minority class. In this report, we explore several techniques for improving the classification performance in the presence of data imbalance.

2.0 Introduction

Classification problems suffer from the issue of data imbalance when one of the class has overwhelmingly higher number of instances. This has a serious impact on the performance of the classifier. For example, if in a problem the majority class represents 99% of the samples, then even a null classifier would yield an accuracy of 0.99. That is, we can classify all the examples to majority class in order to achieve a low error rate of 1%. However, all the minority class instances will be misclassified. Misclassifying a minority class example is usually more costly in case of class imbalance problems.

There are few ways to deal with the data imbalance problem. One can collect more data. Though this might not be feasible in every case, it can be helpful in exposing different and perhaps, new information about the minority class. Also if one opts for natural or synthetic oversampling of the minority class, then having more data can have a considerable impact in the quality of oversampling obtained.

Resampling is another way to approach the problem as it helps obtain a balanced dataset by either under sampling the majority class or oversampling the minority class in natural or synthetic way. Based on the method of resampling used, the classifier performs better for both classes and is able to generalize the decision region in the areas with disproportionately high number of majority class examples.

3.0 Performance Measures

As discussed above, the error rate is not an appropriate measure when there is class imbalance. In this report, we use measures such as sensitivity/Recall, specificity to check the class wise performance. Also to make the threshold of classification less of a force and signify overall performance of the classifier, we use the measure of Area under ROC (Receiver Operating Characteristic).

For classification problems, Confusion matrices are used widely to get the overall picture.

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Figure: Confusion Matrix

Where the columns represent the predicted labels whereas the rows are actual labels.

TN - number of negative examples correctly classified
FP - number of negative examples incorrectly classified as positive
FN - number of positive examples incorrectly classified as negative
TP - number of positive examples correctly classified

As discussed above, Sensitivity/Recall is a good measure to check the performance on the minority class as it represents the fraction of positive class classified accurately.

$$\boldsymbol{Sensitivity} = \frac{\boldsymbol{TP}}{(\boldsymbol{TP} + \boldsymbol{FN})}$$

$$\boldsymbol{Specificity} = \frac{\boldsymbol{TN}}{(\boldsymbol{TN} + \boldsymbol{FP})}$$

The Receiver Operating Characteristic (ROC) curve is a standard technique for summarizing classifier performance over a range of tradeoffs between true positive and false positive error rates. On an ROC curve the X-axis represents false positive rate and the Y-axis represents the true positive rate. The ideal point on the ROC curve would be (0,100), i.e. all positive examples are classified correctly and no negative examples are misclassified as positive. Thus, Area under the ROC curve - AUC - is a useful measure for performance of a classifier as it is independent of decision criteria i.e. threshold for classification and prior probabilities.

4.0 The Dataset

A Credit Card Fraud Detection dataset was used to test various sampling techniques and check for the performance of different classification algorithms. This dataset was obtained from Kaggle. The datasets contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. Due to the sensitive nature of the information the original features have been transformed into 28 principal components. Apart from that the data contains three other features, namely, Amount – amount of transaction, Time – time elapsed between the transactions and the class of transaction, whether the transaction is genuine or fraudulent.

5.0 Resampling Methods and Performance using Logistic Regression

1.0 Performance on imbalanced dataset

We trained the Logistic Regression classifier (with regularization) using the original imbalanced dataset. And as expected, the performance on the minority class was poor as the classifier misclassified 36 out of 101 minority class instances.

Sensitivity/ Recall	Specificity	AUROC
0.643	0.999	0.9776

1.1 Random Under sampling of majority class

In this case, we randomly under sampled the majority class to use a subset of the majority class having size – 492 - equal to that of the minority class. One of the downsides to this approach is, if the size of the minority class instances is same, then the under sampling ignores huge number of majority class examples. This not only leads to poor performance in terms of majority class but also omits potential information that would have been useful in identifying wider decision regions for minority class.

As can be seen below, the classifier suffers from poor performance and misclassifies about 6985 out of 56861 majority class observations.

Sensitivity/ Recall	Specificity	AUROC
0.94	0.88	0.9689

1.2 Ensemble Methods

The main deficiency with the undersampling approach is that it ignores most of the majority class examples. Ensemble methods overcome this deficiency by doing what is called 'Informed Undersampling'. This is achieved by randomly undersampling the majority class multiple times either with or without replacement.

A. Balanced Bagging:

Balanced bagging is the simplest ensemble method. It makes use of bootstrap aggregation to derive a more informed and accurate classifier. It leverages bootstrap to obtain plenty of diverse observations and makes use of aggregation to reduce the variance of the resulting classifier.

The Balanced Bagging algorithm works as shown below:

1. For each iteration, draw a bootstrap sample from the minority class. Randomly draw the same number of cases, with replacement, from the majority class.

2. Apply your desired classification algorithm

3. Repeat the two steps above for the number of times desired. Aggregate the predictions of the ensemble and make the final prediction.

We used Balanced Bagging in concert with logistic regression. As can be seen below Balanced Bagging helps improve performance on the majority class as it reduced the number of misclassified instances for majority class by half, at the same time not affecting the performance for minority class.

Sensitivity/ Recall	Specificity	AUROC
0.94	0.93	0.9729

B. Easy Ensemble:

Easy Ensemble is a technique based on AdaBoost boosting, which combines several weak classifiers to generate a strong classifier. The idea is to use different subsets of data to train a weak classifier (weak classifiers are classifiers which perform only slightly better than a random classifier e.g. a decision stump). In short, AdaBoost assigns a weight to each training example that determines their probability of recurrence. If an example is misclassified, AdaBoost increases the weight on that particular observation so to make these examples a larger portion of next training set. Also a weight is assigned to a weak classifier on the basis of its accuracy.

In Easy Ensemble, we independently sample several subsets from majority class and train a classifier using one subset and all minority class instances. All generated classifier are combined for the final decision. By exploring features from each subset, this ensemble method lets us discover diverse information for the majority class. Boosting helps in reducing bias, by slow learning, while bagging helps in reducing variance.

Below results show vast improvement over the simple undersampling and Balanced Bagging as only 1153 out of 56861 majority class examples are misclassified.

Sensitivity/ Recall	Specificity	AUROC
0.91	0.98	0.9796

1.3 Oversampling Approaches

Before we do oversampling, it is important to understand that if we duplicate or synthesize new minority observations before cross validation then we will be training the classifier with one or more instances that are exactly the same in the validation set. So it is imperative that we first do the cross validation and then do the oversampling.

A. Minority over-sampling with replacement

Essentially, as the minority class is over-sampled by increasing amounts, the effect is to identify similar but more specific regions in the feature space as the decision region for the minority class. In our case, since the number of minority class samples is extremely low, this leads to overfitting as the learning algorithm does not cause the decision boundary to make inroads into the majority class region.

B. SMOTE – Synthetic Minority Oversampling technique

In this resampling technique, the minority class is over-sampled by creating “synthetic” examples rather than by over-sampling with replacement. Specifically, SMOTE takes each minority class sample and introduces "Synthetic examples" along the line segment joining any/all of the k minority class nearest neighbors. The number of neighbors are randomly chosen from the k nearest neighbors based on amount of oversampling. For e.g. if the amount of oversampling needed is 300%, three neighbors are chosen from kNN.

Steps in generating synthetic samples:

1. Take the difference between the feature vector/sample under consideration and its nearest neighbor.

2. Multiply the difference by a random number between 0 & 1.

3. Add the resulting vector to the sample under consideration.

We used different oversampling ratios to generate the synthetic samples. Since this method takes into account a larger share of majority class data, the performance is really good for the majority class. At the same time, the performance for minority class is similar to the one with under sampling approaches and improves as we increase the oversampling ratio.

Oversampling Ratio	Sensitivity	Specificity	AUROC
0.25	0.90	0.99	0.997
0.5	0.91	0.99	0.9885
1	0.94	0.97	0.9898

C. ADASYN – Adaptive Synthetic Sampling

The idea behind ADASYN is to use weighted distribution for different minority class examples according to their difficulty in learning, with more synthetic data generated for minority class examples that are harder to learn. This is the major difference between ADASYN and SMOTE.

Steps in generating synthetic samples:

1. Calculate the degree of class imbalance

2. Calculate the number of synthetic examples to be generated for minority class based on desired balance level

3. For each minority class examples find k nearest neighbors and calculate the fraction of neighbors belonging to majority class

4. Normalize the fractions obtained for all minority class examples.

5. Calculate the number of synthetic examples to be generated for each minority example based on Normalized fractions

6. Use the process used by SMOTE to generate new synthetic examples.

Similar to SMOTE, we trained the classifier using different oversampling ratios. The pattern was the same as for SMOTE, but the major difference was in the improvement of classification for minority class samples. This might be attributed to ADASYN's emphasis towards the difficult to learn minority class examples.

Oversampling Ratio	Sensitivity	Specificity	AUROC
0.25	0.94	0.97	0.9900
0.5	0.96	0.95	0.9907
1	0.97	0.90	0.9911

6.0 Conclusion

In this report, we discussed some of the resampling techniques that can be used to improve classification performance in the presence of class imbalance. We presented the performance of these resampling techniques and discussed about scenarios in which each of these method might or might not be applicable.

Though in our case, ADASYN with oversampling ratio of 0.5 has a better class wise performance than other methods discussed, use of any particular technique should be contingent upon the specific structure of a problem and the balance of class wise performance demanded by the application in addition to the choice of classifier.

By no means, this is an exhaustive list of the techniques available to obtain a balanced dataset and we plan to explore few of the other resampling approaches such as Condensed Nearest Neighbor, NearMiss and TomekLinks to name a few.

7.0 References

[1] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357,2002.

[2] X. Y. Liu, J. Wu and Z. H. Zhou, “Exploratory Undersampling for Class-Imbalance Learning,” in *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539-550, April 2009

[3] http://statistics.berkeley.edu/sites/default/files/tech-reports/666.pdf

[4] Survey of resampling techniques for improving classification performance in unbalanced datasets by Ajinkya More

[5] Imbalanced learn by scikit learn

[6] He, Haibo, Yang Bai, Edwardo A. Garcia, and Shutao Li. “ADASYN: Adaptive synthetic sampling approach for imbalanced learning,” In *IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence),* pp. 1322-1328, 2008.