AMATH 582: Final Project

Lee Burke
Taylor Cunnington

March 14th, 2017

Abstract

Sec. I Introduction and Overview

Machine learning algorithms have trouble with unbalanced data. Algorithms are typically designed to minimize the error rate or something very similar. This inherently favors the majority class.

Our dataset is a great example of imbalanced data. It is real credit card data and we are attempting to identify fraudulent transactions. Our dataset has only 492 of 284807 data points that are positive. This means that the positive class represents a mere 0.17% of the total. In this light it's actually extremely easy to build a classifier with 99.83% accuracy–just always predict negative!

Accuracy, and therefore error, are not the metrics we care most about in this case. We want to be able to identify nearly all the fraudulent transactions. What we care about most is reducing false negatives (FNs). Unless the data is extremely well separated in, which ours is not, if we optimize for low numbers of FNs we will receive lower accuracy. This is because if the class distributions are overlapping then decreasing FPs by will increase false positives (FPs) by a much larger number. In light of this consideration it's immediately clear that we need a better way to measure the success of our algorithm besides accuracy.

Because our dataset necessarily includes sensitive information the features have been obscured by an SVD decomposition. All features except some arbitrary time and transactions amount have been anonymized in this way, so we will not be performing any feature analysis in this work. We will instead concentrate on methods to work with imbalanced data. We focus specifically only how to evaluate the performance of a classifier in this scenario because, as covered more thoroughly below, common performance metrics do not work well. We will investigate performance metrics by applying methods known to help with imbalanced and comparing this to classification without those methods. We will use linear discriminant analysis as our classification scheme throughout.

Sec. II Theoretical Background

Sec. II.1 Classification

The LDA algorithm builds normal probability distributions for each class with mean and covariance parameters $(\vec{\mu}_0, \sigma_0)\,(\vec{\mu}_0, \sigma_0)\,and\,(\vec{\mu}_1, \sigma_1)\,(\vec{\mu}_1, \sigma_1)$ These are referred to as the prior distributions. Classifying subsequent observations is then potentailly just a matter of comparing their probability of being in class 0 or 1 and applying a threshold. MATLAB actually attempts to minimize the cost function:

$$\hat{y} = \arg \min_{y=1,\ldots,K} \sum_{k=1}^{K} \hat{P}(k|x) C(y|k) \tag{1}$$

where $\hat{y}$ is the predicted probability. K is the number of classes. $\hat{P}$ is the posterior probability of class k for a given observation x, and $C(y|k)$ is the cost of misclassifying k as y. $\hat{P}$ is determined from Bayes rule:

$$\hat{P}(k|x) = \frac{P(x|k)P(k)}{P(x)} \tag{2}$$

Here the posterior probability that x is in class k is what LDA estmiates from the training data. $P(k)$ and $P(x)$ are readily calcuated the from the data. Note that for the case of an extremely small minority class such as ours $P(n)$ will be extremely small (where $k = n$); this means that $\hat{P}(n|x)$ will be small as well. This is a source of bias for LDA toward the majority class.

There are some known methods for improving performance when classifying imbalanced data. The four common solution types are given here:

1. Under-sampling – Removing data from the larger class to balance the class sizes. May lead to a poorer choice of decision line due to losing data at the border of the classes.
2. Oversampling – Adding extra observations on top of existing minority class observations to balance out the class sizes. May lead to overfitting with some classification models.
3. Synthetic data generation – Generating artificial data from your existing data to balance classes. Generated data generally stays within the n-dimensional volume that minimally encloses the existing data.
4. Cost functions– Classification algorithms use cost functions (decision functions) to define their decision boundaries. With imbalanced data you would set the misclassification of the minority class to be much more costly, to encourage the algorithm to classify them correctly more often than the majority class.

The first three attempt to reduce the imbalance by reducing the number of majority data point or increasing the number of minorty data points. These methods deal with the data only, not the classifier. The final option does alter the classifier however, to try to make it classify the minority class more reliably.

Sec. II.2 Performance evaluation

It's not just accuracy that serves as a poor metric in this task. Precision and Recall are two common measures. Plotting the former on the y-axis and the latter on the x-axis gives you Precision-Recall (PR) curves, which are often used to evaluate algorithms. Recall is the fraction of positive entries that you correctly identified as positive: tp/(tp+fn). Precision is the fraction of entries that were actually positive out of all the ones you guessed were positive: tp/(tp+fp).

In our case we want to optimize for recall over precision. That is, when we are given an input that is actually positive we want to correctly predict as much as possible that it is positive, even if that means we end up with false positives. If we get false positives that lowers our precision, but since false negatives are more costly than false positives here we can accept potentially many more false positives.

For the reason above precision is not a good metric for this task. For the functionality we want to optimize we will likely see precision increase over a simple majority class classifier, but we can't really be sure of the exact relationship. A better metric would be one that is guaranteed to decrease monotonically as we approach our ideal of perfect recall, regardless of what happens to precision.

A metric that does work well is the Area Under the Precision-Recall Curve (AUPRC)...

Sec. III Algorithm Implementation and Development

As stated above our data was prepared and features provided as U columns of the SVD decomposition, so we did not alter or clean the data. We created a "full" data matrix X from the 28 columns of U and the amount of the transaction. We used standard Z-score of the X matrix, Z, for all implementations of the LDA classifier.

Then we split the data for cross validation... or did we do k-fold validation??

Using LDA as our classification algorithm we compared the synthetic data generation and alternate cost function approaches to the default algorithm. For synthetic data generation we chose to use ADASYN (Adaptive Synthetic Sampling Approach for Imbalanced Learning) a generation algorithm similar to the well known SMOTE (Synthetic Minority Over-Sampling Technique) algorithm, which is ADASYN's precursor. The goal of both is to improve the class balance by increasing the number of minority class members. SMOTE places synthetic data between existing data points randomly (linear interpolation), with no preference shown to any specific points. ADASYN does the same thing but places more synthetic data points close to the boundary between classes because those are the original data

points that are more difficult to learn. (Does this favor decision trees or SVMs or something? I imagine that all this would do for an LDA is to move the mean of the minority gaussian close to the boundary... SMOTE might be better here...)

We also tried changing the cost function of our LDA model to discourage FNs. Since LDA works by creating probability distributions the cost comes into play only when making predictions. Each observation that it is trying to predict has a calculated posterior probability. It tries to minimize the "classification cost"–this is a decision cost function. Unfortunately if an observation has an extremely small posterior probability then even with a vary large cost for miscalculating that observation it may not change the classification cost by much, meaning there will be little change to the resulting prediction.

Sec. IV Computational Results

GIVE BASELINE LDA RESULTS

Altering the prediction cost function does not make a difference in our case. Weighting false negatives as very costly makes very little difference, even when the cost of a false negative is 10^8 as costly as a false positive. This seems odd, since we only expect class p to be about 500 times as likely class n ($P(p) \approx 500 \times P(n)$). There must be something else going on... PERFORMANCE MATRICS

Synthetic data generation did improve recall. As expected this came at the cost of precision and accuracy. PERFORMANCE METRICS

Sec. V Summary and Conclusions

Appendix A MATLAB functions used and brief implementation explanation

-

Appendix B MATLAB codes