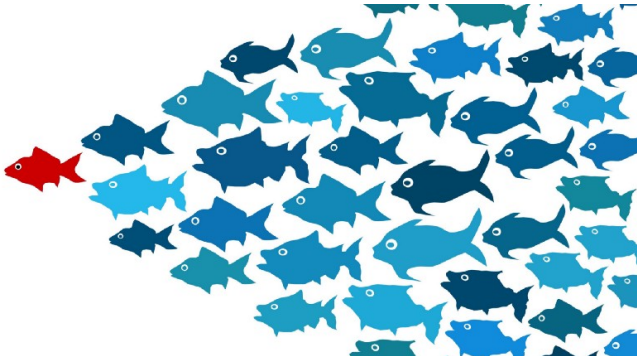
Dealing with Imbalanced Classes in Machine Learning

Devin Soni

Feb 2, 2018 · 5 min read



Introduction

Most real-world classification problems display some level of class imbalance, which is when each class does not make up an equal portion of your data-set. It is important to properly adjust your metrics and methods to adjust for your goals. If this is not done, you may end up optimizing for a meaningless metric in the context of your use case.

For example, suppose you have two classes—A and B. Class A is 90% of your data-set and class B is the other 10%, but you are most interested in identifying instances of class B. You can reach an accuracy of 90% by simply predicting class A every time, but this provides a useless classifier for your intended use case. Instead, a properly calibrated method may achieve a lower accuracy, but would have a substantially higher true positive rate (or recall), which is really the metric you should have been optimizing for. These scenarios often occur in the context of detection, such as for abusive content online, or disease markers in medical data.

I will now discuss several techniques that can be used to mitigate class imbalance. Some techniques are applicable to most classification problems while others may be more suited to specific levels of imbalance. For this article, I will discuss these in terms of binary classification, but for the most part, the same will hold for multi-class classification. I will also assume **the goal is to identify the minority class**, as otherwise, these techniques are not really necessary.

Metrics

Generally, this problem deals with the trade-off between **recall** (percent of truly positive instances that were classified as such) and **precision** (percent of positive classifications that are truly positive). In

situations where we want to detect instances of a minority class, we are usually concerned more so with recall than precision, as in the context of detection, it is usually more costly to miss a positive instance than to falsely label a negative instance. For example, if we are trying to detect abusive content, it is trivial to have a manual reviewer find that content is actually not abusive, but it is much harder to identify abusive content that was never even flagged as such. Thus, when comparing approaches to imbalanced classification problems, consider using metrics beyond accuracy such as recall, precision, and AUROC. It may be that switching the metric you optimize for during parameter selection or model selection is enough to provide desirable performance detecting the minority class.

Cost-sensitive Learning

In regular learning, we treat all misclassifications equally, which causes issues in imbalanced classification problems, as there is no extra reward for identifying the minority class over the majority class. Cost-sensitive learning changes this, and uses a function **C(p, t)** (usually represented as a matrix) that specifies the cost of misclassifying an instance of class **t** as class **p**. This allows us to penalize misclassifications of the minority class more heavily than we do with misclassifications of the majority class, in hopes that this increases the true positive rate. A common scheme for this is to have the cost equal to the inverse of the proportion of the data-set that the class makes up. This increases the penalization as the class size decreases.

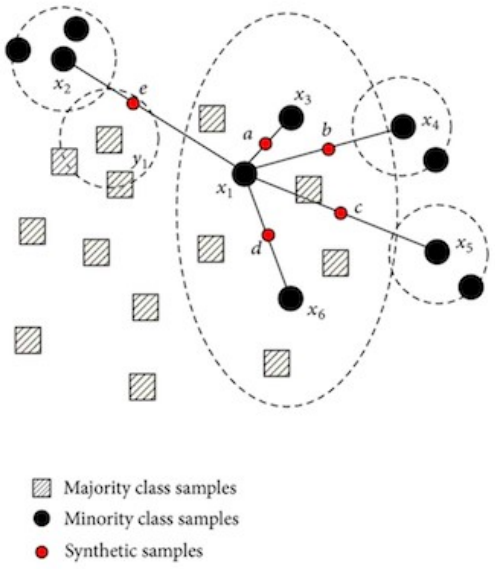
	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	C_{TP_i}	C_{FP_i}
Predicted Negative $c_i = 0$	C_{FN_i}	C_{TN_i}

Sample cost function matrix

Sampling

A simple way to fix imbalanced data-sets is simply to balance them, either by **oversampling** instances of the minority class or **undersampling** instances of the majority class. This simply allows us to create a balanced data-set that, in theory, should not lead to classifiers biased toward one class or the other. However, in practice, these simple sampling approaches have flaws. Oversampling the minority can lead to model overfitting, since it will introduce duplicate instances, drawing from a pool of instances that is already small. Similarly, undersampling the majority can end up leaving out important instances that provide important differences between the two classes.

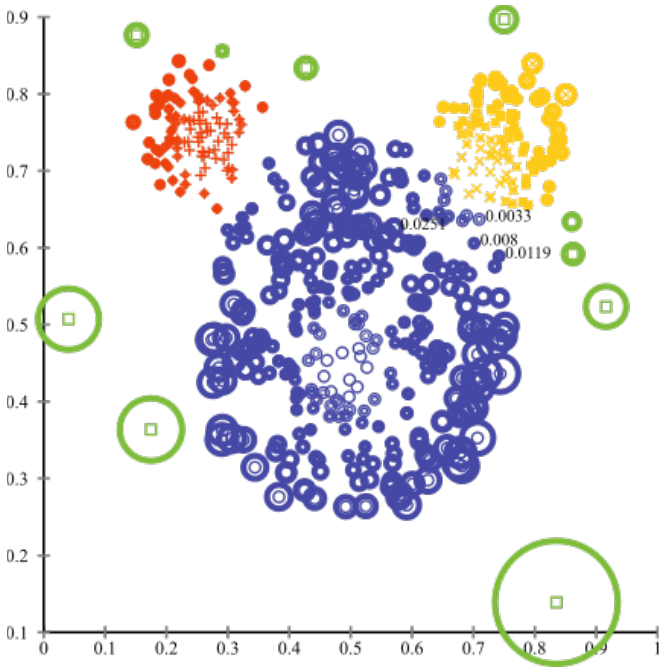
There also exist more powerful sampling methods that go beyond simple oversampling or undersampling. The most well known example of this is **SMOTE**, which actually creates new instances of the minority class by forming convex combinations of neighboring instances. As the graphic below shows, it effectively draws lines between minority points in the feature space, and samples along these lines. This allows us to balance our data-set without as much overfitting, as we create new synthetic examples rather than using duplicates. This however does not prevent all overfitting, as these are still created from existing data points.



Visualization of SMOTE

Anomaly Detection

In more extreme cases, it may be better to think of classification under the context of anomaly detection. In anomaly detection, we assume that there is a “normal” distribution(s) of data-points, and anything that sufficiently deviates from that distribution(s) is an anomaly. When we reframe our classification problem into an anomaly detection problem, we treat the majority class as the “normal” distribution of points, and the minority as anomalies. There are many algorithms for anomaly detection such as clustering methods, One-class SVMs, and Isolation Forests.



Visualization of clustering method for anomaly detection

Conclusion

Hopefully some combination of these methods will allow you to create a better classifier. Like I said before, some of these techniques lend themselves better to different degrees of imbalance. For example, simple sampling techniques may allow you to overcome slight imbalance, whereas anomaly detection methods may be required for extreme imbalances. Ultimately, there is no one-size-fits-all method for this problem, and you just have to try out each method and see how they apply to your specific use case and metrics for success.

. . .

Make sure you give this post **50 claps** and my blog a **follow** if you enjoyed this post and want to see more.