cleanbalance.png 800x550

dirtyblance.png 800x550

But when you start looking at real, uncleaned data one of the first things you notice is that it’s a lot noisier and imbalanced. Scatterplots of real data often look more like this:

scatterplot of imbalanced classes

The primary problem is that these classes are imbalanced: the red points are greatly outnumbered by the blue.

Research on imbalanced classes often considers imbalanced to mean a minority class of 10% to 20%. In reality, datasets can get far more imbalanced than this. —Here are some examples:

1. About 2% of credit card accounts are defrauded per year1. (Most fraud detection domains are heavily imbalanced.)
2. Medical screening for a condition is usually performed on a large population of people without the condition, to detect a small minority with it (e.g., HIV prevalence in the USA is ~0.4%).
3. Disk drive failures are approximately ~1% per year.
4. The conversion rates of online ads has been estimated to lie between 10-3 to 10-6.
5. Factory production defect rates typically run about 0.1%.

Many of these domains are imbalanced because they are what I call needle in a haystack problems, where machine learning classifiers are used to sort through huge populations of negative (uninteresting) cases to find the small number of positive (interesting, alarm-worthy) cases.

When you encounter such problems, you’re bound to have difficulties solving them with standard algorithms. Conventional algorithms are often biased towards the majority class because their loss functions attempt to optimize quantities such as error rate, not taking the data distribution into consideration2. In the worst case, minority examples are treated as outliers of the majority class and ignored. The learning algorithm simply generates a trivial classifier that classifies every example as the majority class.

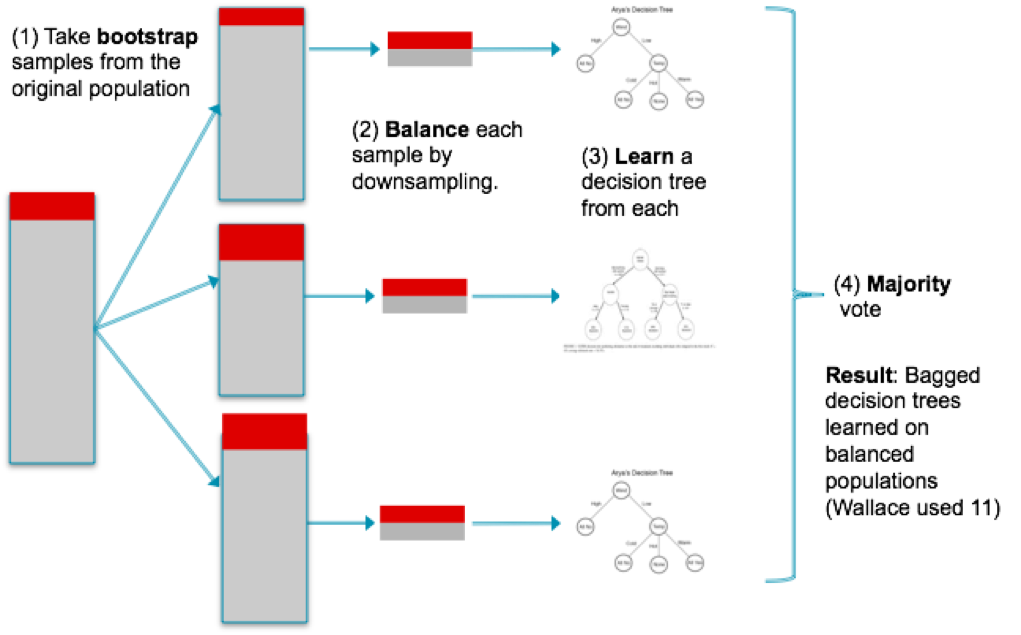
This might seem like pathological behavior but it really isn’t. Indeed, if your goal is to maximize simple accuracy (or, equivalently, minimize error rate), this is a perfectly acceptable solution. But if we assume that the rare class examples are much more important to classify, then we have to be more careful and more sophisticated about attacking the problem.

<https://www.svds.com/learning-imbalanced-classes/>

1. Don’t use accuracy (or error rate) to evaluate your classifier! There are two significant problems with it. Accuracy applies a naive 0.50 threshold to decide between classes, and this is usually wrong when the classes are imbalanced. Second, classification accuracy is based on a simple count of the errors, and you should know more than this. You should know which classes are being confused and where (top end of scores, bottom end, throughout?)  
     
   You should be visualizing classifier performance using a ROC curve, a precision-recall curve, a lift curve, or a profit (gain) curve.
2. Don’t get hard classifications (labels) from your classifier (via score3 or predict). Instead, get probability estimates via proba or predict\_proba.
3. When you get probability estimates, don’t blindly use a 0.50 decision threshold to separate classes. Look at performance curves and decide for yourself what threshold to use (see next section for more on this). Many errors were made in early papers because researchers naively used 0.5 as a cut-off.
4. No matter what you do for training, always test on the natural (stratified) distribution your classifier is going to operate upon. See sklearn.cross\_validation.StratifiedKFold.
5. You can get by without probability estimates, but if you need them, use calibration (see sklearn.calibration.CalibratedClassifierCV)
6. The two-dimensional graphs in the first bullet above are always more informative than a single number, but if you need a single-number metric, one of these is preferable to accuracy:
7. The Area Under the ROC curve (AUC) is a good general statistic. It is equal to the probability that a random positive example will be ranked above a random negative example.
8. The F1 Score is the harmonic mean of precision and recall. It is commonly used in text processing when an aggregate measure is sought.
9. Cohen’s Kappa is an evaluation statistic that takes into account how much agreement would be expected by chance.

Some data scientists (naively) think that oversampling is superior because it results in more data, whereas undersampling throws away data. But keep in mind that replicating data is not without consequence—since it results in duplicate data, it makes variables appear to have lower variance than they do. The positive consequence is that it duplicates the number of errors: if a classifier makes a false negative error on the original minority data set, and that data set is replicated five times, the classifier will make six errors on the new set. Conversely, undersampling can make the independent variables look like they have a higher variance than they do.

So a final step is to use [bagging](https://en.wikipedia.org/wiki/Bootstrap_aggregating) to combine these classifiers. The entire process looks like this:

This technique has not been implemented in Scikit-learn, though a file called blagging.py(balanced bagging) is available that implements a BlaggingClassifier, which balances bootstrapped samples prior to aggregation.