towards
data science

Dealing with Imbalanced Data

A guide to effectively handling imbalanced datasets in Python

Tara

Feb 3, 2019 · 5 min read ★

Photo by Ales Nesetril on Unsplash

Imbalanced classes are a common problem in machine learning classification where there are a disproportionate ratio of observations in each class. Class imbalance can be found in many different areas including medical diagnosis, spam filtering, and fraud detection.

In this guide, we'll look at five possible ways to handle an imbalanced class problem.

Important Note: This guide will focus solely on addressing imbalanced classes and will not addressing other important machine learning steps including, but not limited to, feature selection or hyperparameter tuning.

Data

We will use the Credit Card Fraud Detection Dataset available on Kaggle. The dataset is high imbalanced, with only 0.17% of transactions being classified as fraudulent. The full notebook can be found here.

Our objective will be to correctly classify the minority class of fraudulent transactions.

The Problem with Imbalanced Classes

Most machine learning algorithms work best when the number of samples in each class are about equal. This is because most algorithms are designed to maximize accuracy and reduce error.

The Problem with Accuracy

Here we can use the DummyClassifier to always predict “not fraud” just to show how misleading accuracy can be.

We got an accuracy score of 99.8% — And without even training a model! Let’s compare this to logistic regression, an actual trained classifier.

Maybe not surprisingly, our accuracy score decreased as compared to the dummy classifier above. This tells us that either we did something wrong in our logistic regression model, or that accuracy might not be

our best option for measuring performance.

Let's take a look at some popular methods for dealing with class imbalance.

1. Change the performance metric

As we saw above, accuracy is not the best metric to use when evaluating imbalanced datasets as it can be very misleading. Metrics that can provide better insight include:

- **Confusion Matrix:** a table showing correct predictions and types of incorrect predictions.
- **Precision:** the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.
- **Recall:** the number of true positives divided by the number of positive values in the test data. Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.
- **F1: Score:** the weighted average of precision and recall.

Let's see what happens when we apply these F1 and recall scores to our logistic regression from above.

These scores don't look quite so impressive. Let's see what other methods we might try to improve our new metrics.

2. Change the algorithm

While in every machine learning problem, it's a good rule of thumb to try a variety of algorithms, it can be especially beneficial with imbalanced datasets. Decision trees frequently perform well on imbalanced data. They work by learning a hierarchy of if/else questions and this can force both classes to be addressed.

While our accuracy score is slightly lower, both F1 and recall have increased as compared to logistic regression! It appears that for this specific problem, random forest may be a better choice of model.

3. Resampling Techniques — Oversample minority class

Our next method begins our resampling techniques.

Oversampling can be defined as adding more copies of the minority class. Oversampling can be a good choice when you don't have a ton of data to work with.

We will use the resampling module from Scikit-Learn to randomly replicate samples from the minority class.

Important Note

Always split into test and train sets BEFORE trying oversampling

techniques! Oversampling before splitting the data can allow the exact same observations to be present in both the test and train sets. This can allow our model to simply memorize specific data points and cause overfitting and poor generalization to the test data.

After resampling we have an equal ratio of data points for each class! Let's try our logistic regression again with the balanced training data.

Our recall score increased, but F1 is much lower than with either our baseline logistic regression or random forest from above. Let's see if undersampling might perform better here.

4. Resampling techniques — Undersample majority class

Undersampling can be defined as removing some observations of the majority class. Undersampling can be a good choice when you have a ton of data -think millions of rows. But a drawback is that we are removing information that may be valuable. This could lead to underfitting and poor generalization to the test set.

We will again use the resampling module from Scikit-Learn to randomly remove samples from the majority class.

Again, we have an equal ratio of fraud to not fraud data points, but in this case a much smaller quantity of data to train the model on. Let's again apply our logistic regression.

Undersampling underperformed oversampling in this case. Let's try one more method for handling imbalanced data.

5. Generate synthetic samples

A technique similar to upsampling is to create synthetic samples. Here we will use imblearn's SMOTE or Synthetic Minority Oversampling Technique. SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we can use for training our model.

Again, it's important to generate the new samples only in the training set to ensure our model generalizes well to unseen data.

After generating our synthetic data points, let's see how our logistic regression performs.

Our F1 score is increased and recall is similar to the upsampled model above and for our data here outperforms undersampling.

Conclusion

We explored 5 different methods for dealing with imbalanced datasets:

1. Change the performance metric
2. Change the algorithm
3. Oversample minority class

4. Undersample majority class
5. Generate synthetic samples

It appears for this particular dataset random forest and SMOTE are among the best of the options we tried here.

These are just some of the many possible methods to try when dealing with imbalanced datasets, and not an exhaustive list. Some others methods to consider are collecting more data or choosing different resampling ratios — you don't have to have exactly a 1:1 ratio!

You should always try several approaches and then decide which is best for your problem.

Machine Learning | Python | Data Science | Towards Data Science | Class Imbalance