Oversampling

Jill congratulates you on the great work you've done so far. Well done! Before setting you free to tackle your machine learning assignment, however, she would like you to become familiar with a family of resampling techniques designed to deal with class imbalance.

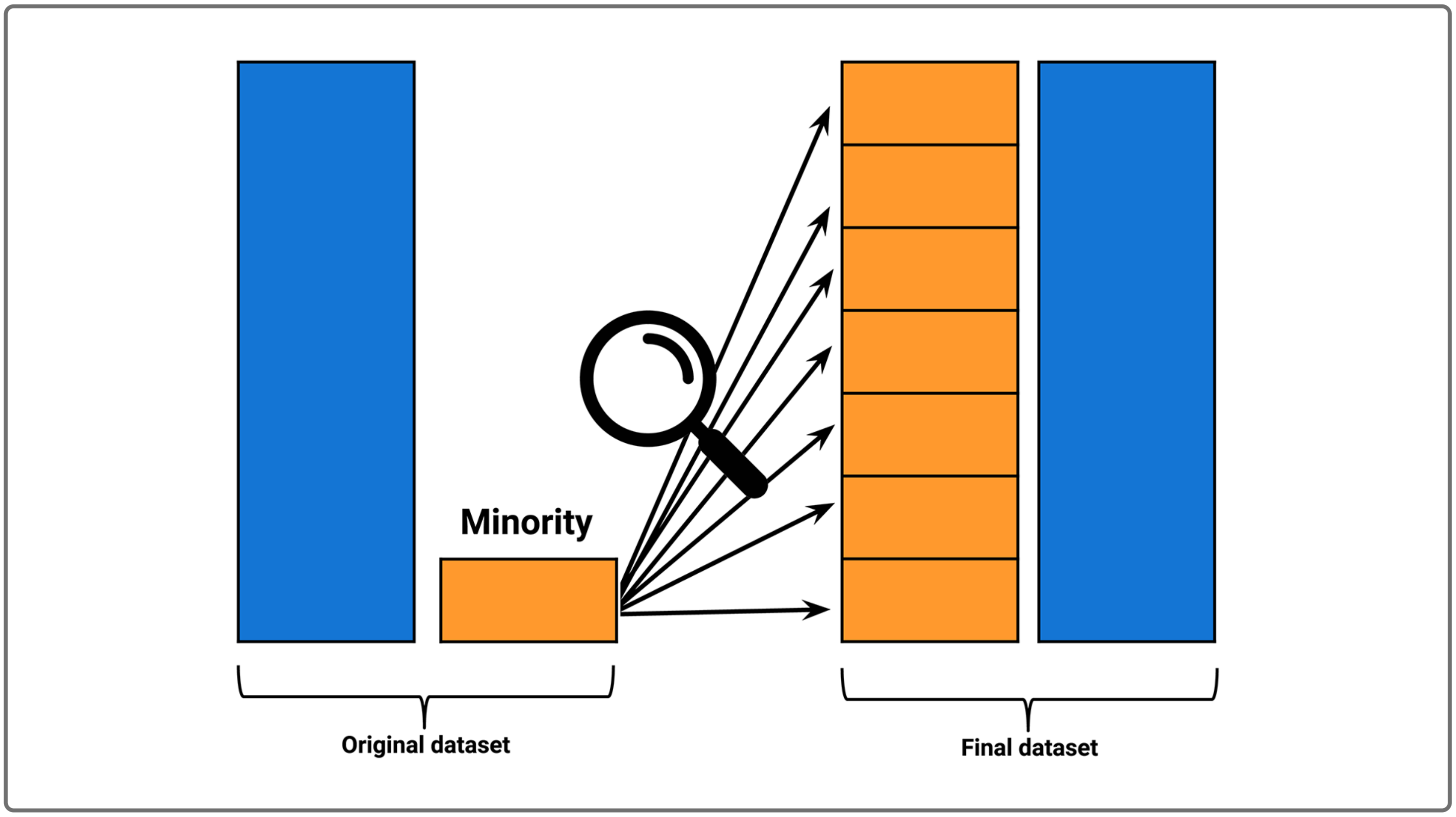
Class imbalance is a common problem in classification. It occurs when one class is much larger than the other class. For example, if you work for a credit card company and want to detect fraudulent transactions, you will deal with many more non-fraudulent transactions than fraudulent ones. In this case, the non-fraudulent class is much larger than the fraudulent class.

Under Jill's tutelage, you will learn about three techniques to address class imbalance: oversampling, undersampling, and a combination approach of oversampling and undersampling. But for now, you'll start with oversampling.

Class imbalance refers to a situation in which the existing classes in a dataset aren't equally represented. Earlier we discussed a fraud detection scenario in which a large number of credit card transactions are legitimate, and only a small number are fraudulent. For example, let's say that out of 100,000 transactions, 50 are fraudulent and the rest are legitimate. The pronounced imbalance between the two classes (fraudulent and non-fraudulent) can cause machine learning models to be biased toward the majority class. In such a case, the model will be much better at predicting non-fraudulent transactions than fraudulent ones. This is a problem if the goal is to detect fraudulent transactions!

In such a case, even a model that blindly classifies every transaction as non-fraudulent will achieve a very high degree of accuracy. As we saw previously, one strategy to deal with class imbalance is to use appropriate metrics to evaluate a model's performance, such as precision and recall.

Another strategy is to use **oversampling**. The idea is simple and intuitive: If one class has too few instances in the training set, we choose more instances from that class for training until it's larger.



We'll discuss two oversampling techniques: random oversampling and synthetic minority oversampling technique.

Random Oversampling

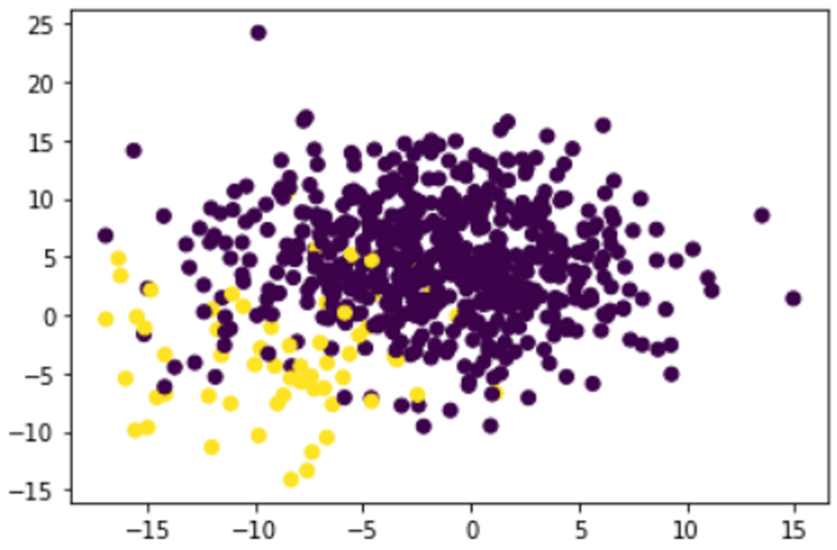
In **random oversampling,** instances of the minority class are randomly selected and added to the training set until the majority and minority classes are balanced. The Python implementation is simple.

To get started, download the necessary files.

[Download 17-10-1-oversampling.zip (https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_17/17-10-1-oversampling.zip)](https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_17/17-10-1-oversampling.zip)

Open the downloaded Jupyter Notebook. After importing dependencies, an unbalanced dataset with two classes is artificially created and plotted, as shown in the following code blocks and resulting chart.

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from collections import Counter


X, y = make_blobs(n_samples=[600, 60], random_state=1, cluster_std=5)
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()
```



The visualization confirms the imbalance—the purple class visibly outnumbers the yellow class.

Next, the dataset is split into training and testing sets.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
Counter(y_train)
```

Python's `Counter` module confirms the imbalance in the training set. There are 451 samples from the purple class and 44 samples from the yellow class.

```
Counter({0: 451, 1: 44})
```

Next, we randomly oversample the minority class with the `imblearn` library.

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=1)
```

```
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)
```

Let's break down what's happening here:

- An instance of `RandomOverSampler` is instantiated as `ros`.
- The training data (`X_train` and `y_train`) is resampled using the `fit_resample()` method.
- The results are called `X_resampled` and `y_resampled`.

Counting the classes of the resampled target verifies that the minority class has been enlarged.

```
Counter(y_resampled)


Counter({0: 451, 1: 451})
```

With a resampled dataset, we can now carry out the familiar pattern of training a model, making predictions, and evaluating the model's performance. For this example, we'll use a `LogisticRegression` model. The following code instantiates and trains the model.

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs', random_state=1)
model.fit(X_resampled, y_resampled)
```

The model creates predictions. We then generate a `confusion_matrix` with the results.

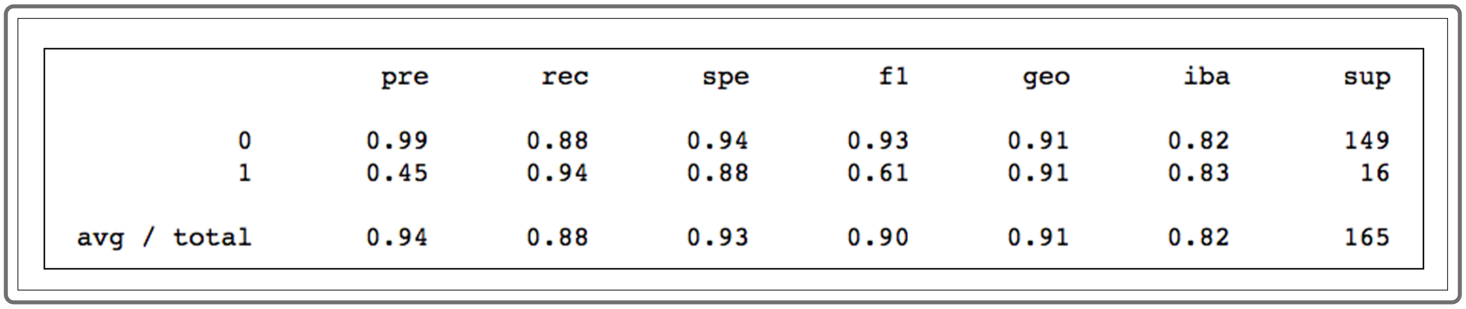
```
from sklearn.metrics import confusion_matrix
y_pred = model.predict(X_test)
confusion_matrix(y_test, y_pred)
```

To assess the accuracy score of the model, we'll use the `balanced_accuracy_score` module.

```
from sklearn.metrics import balanced_accuracy_score
balanced_accuracy_score(y_test, y_pred)
```

The accuracy score is high at around 90%, but this number can be misleading, especially in an unbalanced dataset. Let's examine the classification report to assess the results further. We'll use the `classification_report_imbalanced` to do so.

```
from imblearn.metrics import classification_report_imbalanced
print(classification_report_imbalanced(y_test, y_pred))
```

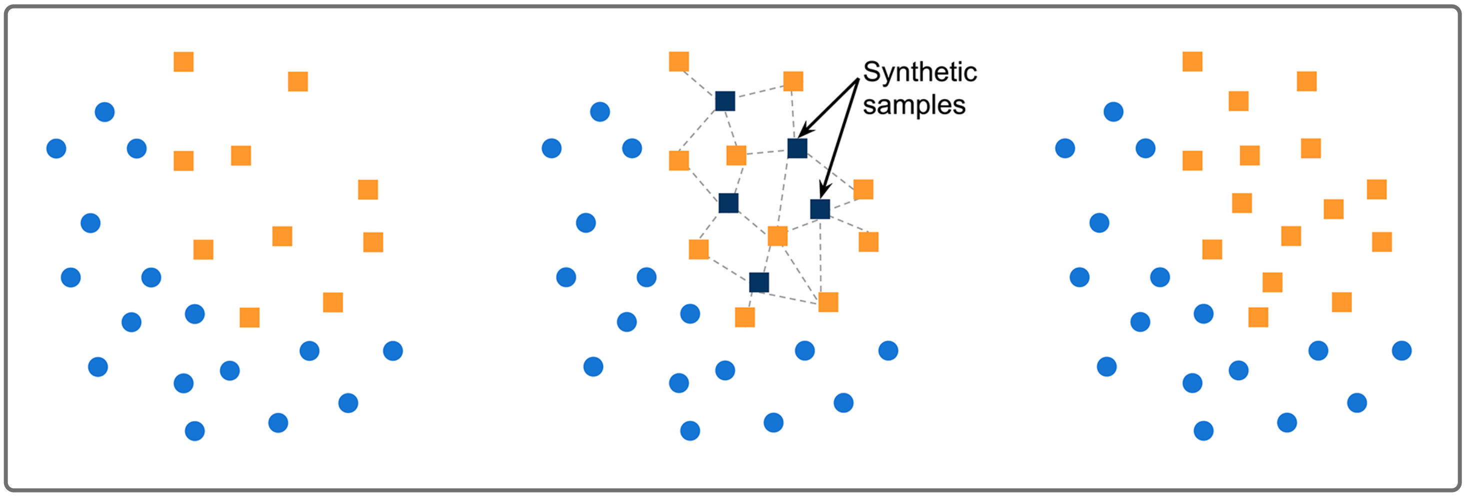


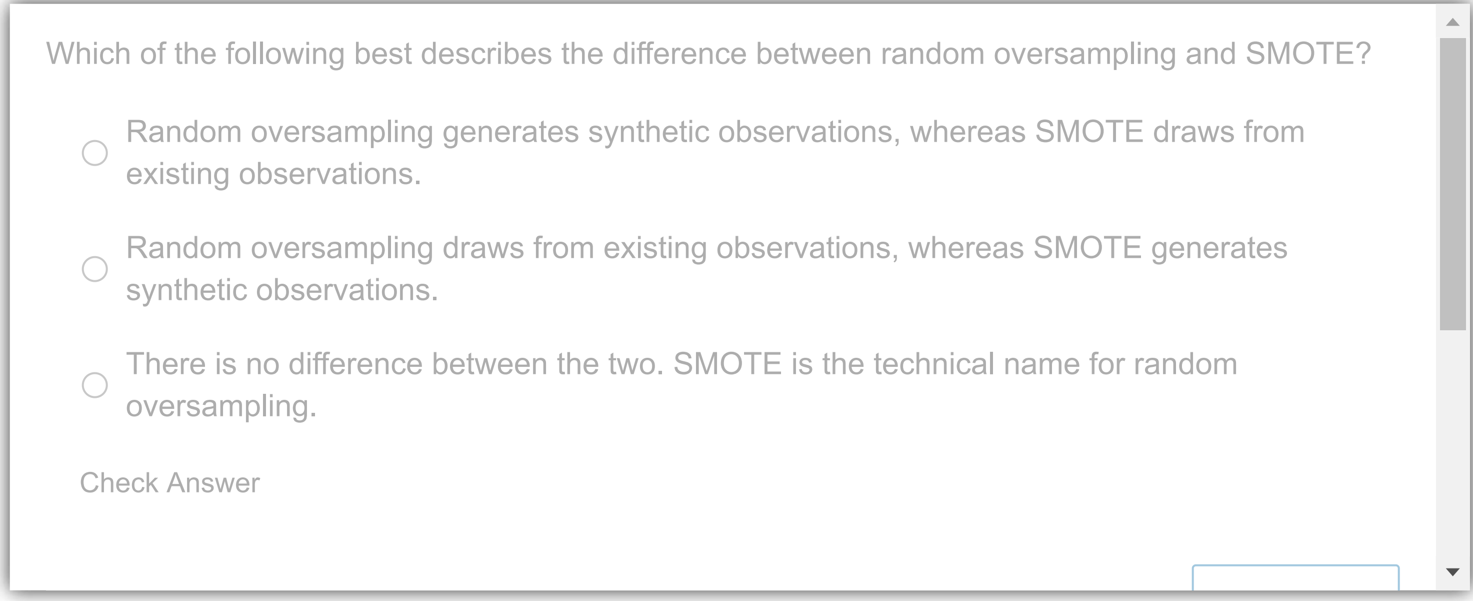
```
                   pre       rec       spe        f1       geo       iba       sup

          0       0.99      0.88      0.94      0.93      0.91      0.82       149
          1       0.45      0.94      0.88      0.61      0.91      0.83        16

avg / total       0.94      0.88      0.93      0.90      0.91      0.82       165
```

While precision ("pre" column) and recall ("rec" column) are high for the majority class, precision is low for the minority class.

Synthetic Minority Oversampling Technique

The **synthetic minority oversampling technique (SMOTE)** is another oversampling approach to deal with unbalanced datasets. In SMOTE, like random oversampling, the size of the minority is increased. The key difference between the two lies in how the minority class is increased in size. As we have seen, in random oversampling, instances from the minority class are randomly selected and added to the minority class. In SMOTE, by contrast, new instances are interpolated. That is, for an instance from the minority class, a number of its closest neighbors is chosen. Based on the values of these neighbors, new values are created.





Which of the following best describes the difference between random oversampling and SMOTE?

- ◯ Random oversampling generates synthetic observations, whereas SMOTE draws from existing observations.
- ◯ Random oversampling draws from existing observations, whereas SMOTE generates synthetic observations.
- ◯ There is no difference between the two. SMOTE is the technical name for random oversampling.

Check Answer

Let's look at SMOTE in action. Note that the following code is contained in the same Jupyter Notebook as the random oversampling example, and that we are using the same training data (`X_train` and `y_train`). We use the `SMOTE` module from the `imblearn` library to oversample the minority class.

```
from imblearn.over_sampling import SMOTE
X_resampled, y_resampled = SMOTE(random_state=1,
sampling_strategy='auto').fit_resample(
   X_train, y_train)
```

The following actions are taking place:

- The `sampling_strategy` argument specifies how the dataset is resampled. By default, it increases the minority class size to equal the majority class's size.

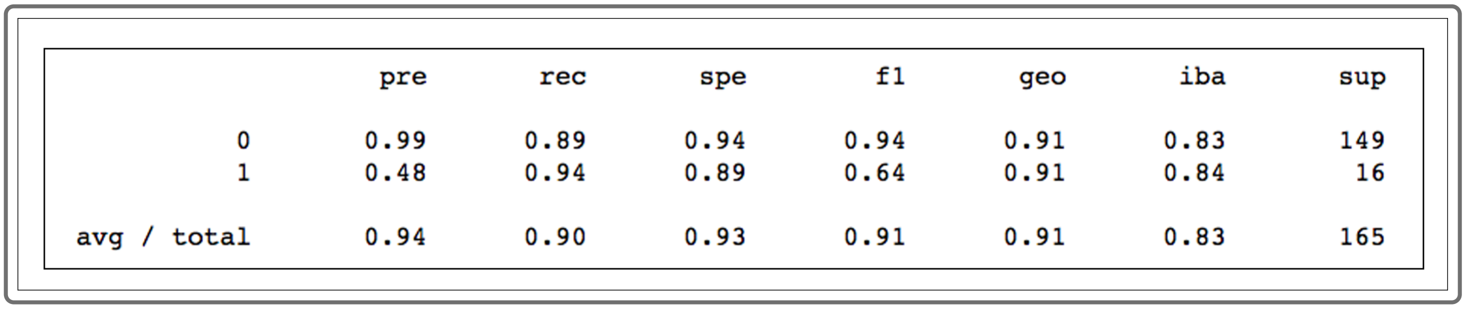
- Again, the `fit_resample()` method is used on the training data to train the SMOTE model and to oversample in a single step.

Counting the number of instances by class verifies that they are now equal in size.

```
Counter(y_resampled)

Counter({0:451, 1:451})
```

We'll again train a `LogisticRegression` model, predict, then assess the accuracy and generate a `confusion_matrix`, as shown in the following code blocks.

```
model = LogisticRegression(solver='lbfgs', random_state=1)
model.fit(X_resampled, y_resampled)

y_pred = model.predict(X_test)
balanced_accuracy_score(y_test, y_pred)

confusion_matrix(y_test, y_pred)

print(classification_report_imbalanced(y_test, y_pred))
```



```
                   pre       rec       spe        f1       geo       iba       sup

          0       0.99      0.89      0.94      0.94      0.91      0.83       149
          1       0.48      0.94      0.89      0.64      0.91      0.84        16

avg / total       0.94      0.90      0.93      0.91      0.91      0.83       165
```

The metrics of the minority class (precision, recall, and F1 score) are slightly improved over those of random oversampling.

It's important to note that although SMOTE reduces the risk of oversampling, it does not always outperform random oversampling. Another deficiency of SMOTE is its vulnerability to outliers. We said earlier that a minority class instance is selected, and new values are generated based on its distance from its neighbors. If the neighbors are extreme outliers, the new values will reflect this. Finally, keep in mind that sampling techniques cannot overcome the deficiencies of the original dataset!