17.10.3

Combination Sampling With SMOTEENN

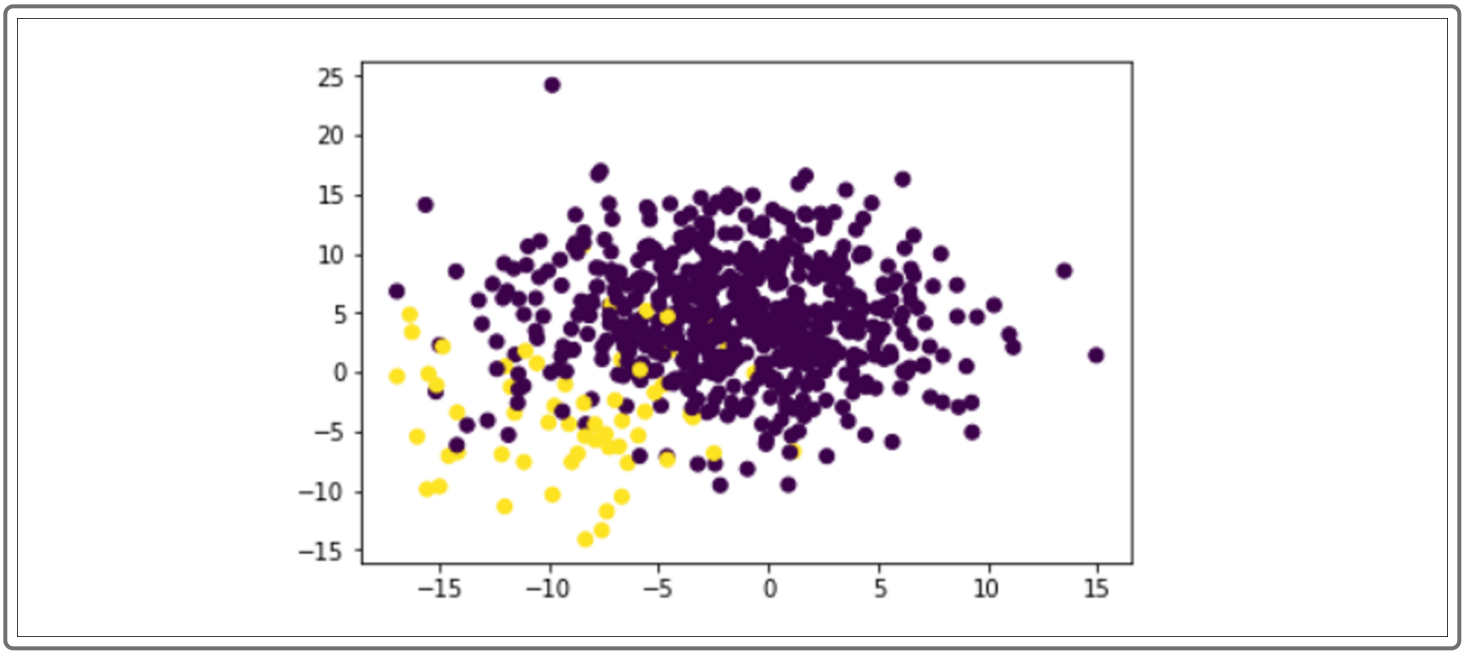
You discuss the results of oversampling and undersampling with Jill. When you point out to her that the improvements seem to be modest, she explains that incremental improvements are usually more realistic than drastic ones. Jill also tells you that such small improvements, in tandem with other tweaks, can add up to make a significant difference. For now, however, she suggests learning about SMOTEENN, an approach to resampling that combines aspects of both oversampling and undersampling.

As previously discussed, a downside of oversampling with SMOTE is its reliance on the immediate neighbors of a data point. Because the algorithm doesn't see the overall distribution of data, the new data points it creates can be heavily influenced by outliers. This can lead to noisy data. With downsampling, the downsides are that it involves loss of data and is not an option when the dataset is small. One way to deal with these challenges is to use a sampling strategy that is a combination of oversampling and undersampling.

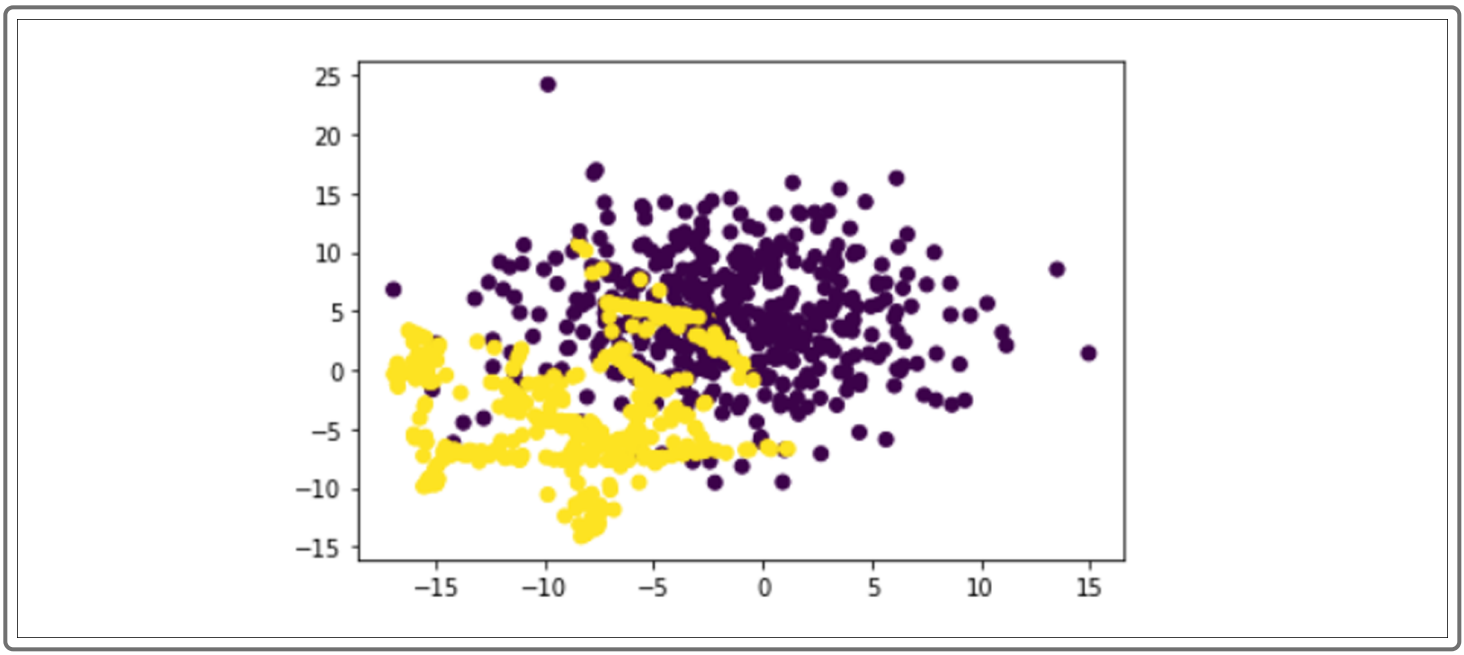
SMOTEENN combines the SMOTE and Edited Nearest Neighbors (ENN) algorithms. SMOTEENN is a two-step process:

1. Oversample the minority class with SMOTE.
2. Clean the resulting data with an undersampling strategy. If the two nearest neighbors of a data point belong to two different classes, that data point is dropped.

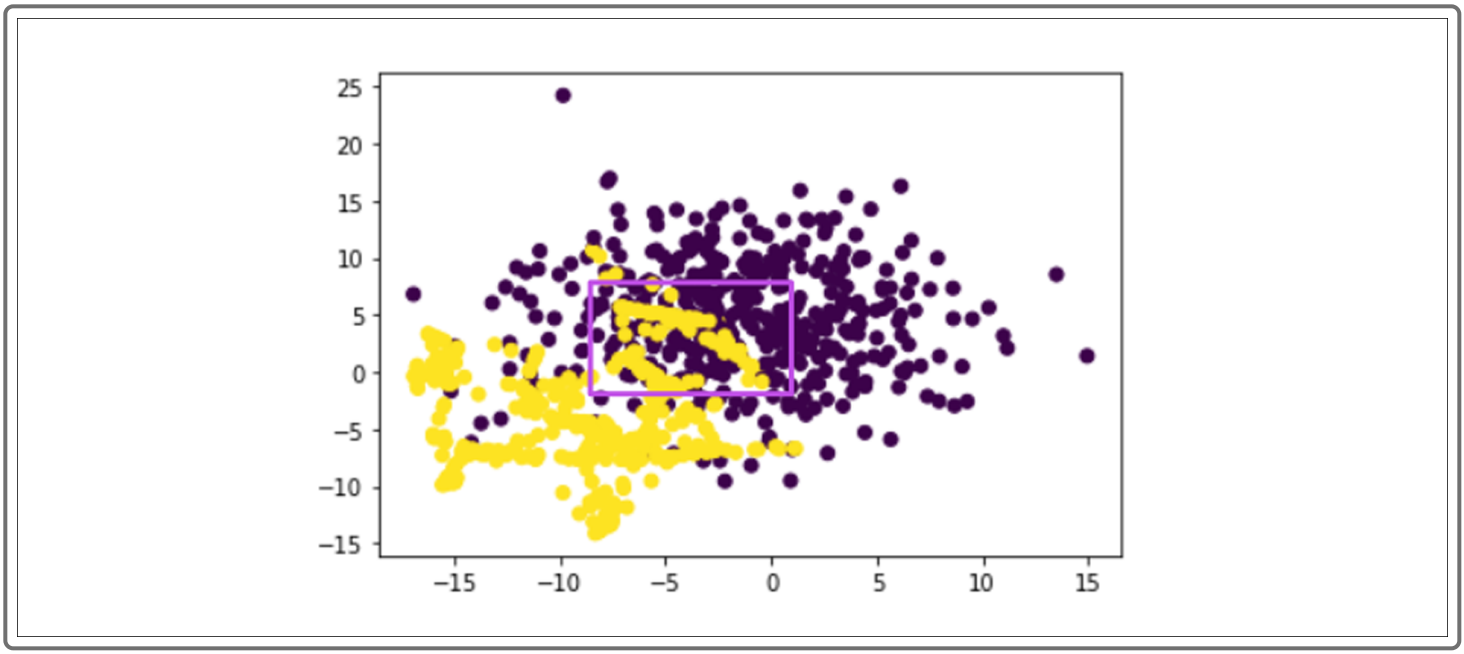
The series of images below help illustrate the SMOTEENN technique. The first image represents a synthetically generated dataset (using the `make_blobs` module) and shows two classes: purple as the majority class and yellow as the minority class.



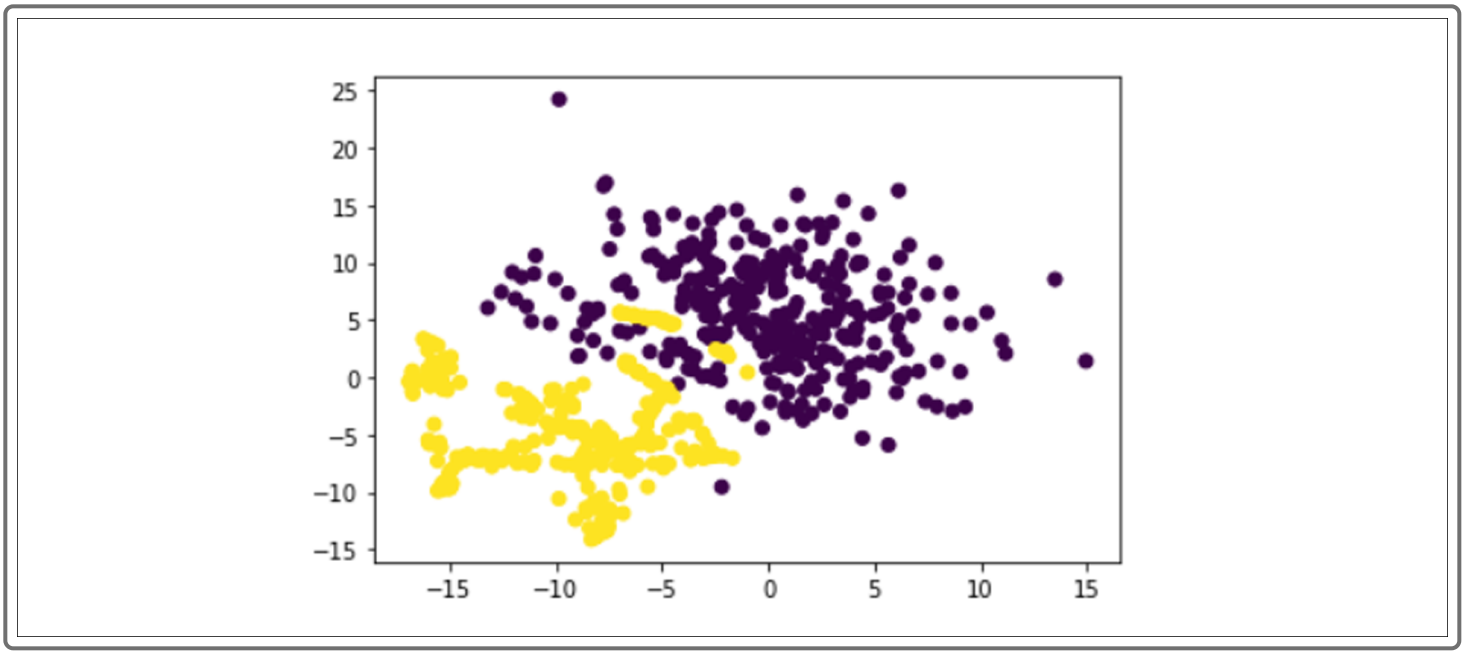
In the following image, the minority class is oversampled with SMOTE.



Note that the two classes significantly overlap, as the box indicates below. This overlap makes classification difficult.



In the next image, SMOTEENN is applied, instead of SMOTE. As with SMOTE, the minority class is oversampled; however, an undersampling step is added, removing some of each class's outliers from the dataset. The result is that the two classes are separated more cleanly.



Let's apply SMOTEENN to the credit card default dataset and compare its results.

Download 17-10-3-combination_sampling.zip (https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_17/17-10-3-combination_sampling.zip)

The code is much the same as before. The only difference is in using the SMOTEENN module.

```
import pandas as pd
from path import Path
from collections import Counter

data = Path('../Resources/cc_default.csv')
df = pd.read_csv(data)
df.head()
```

	ID	ln_balance_limit	sex	education	marriage	age	default_next_month
0	1	9.903488	1	2	0	24	1
1	2	11.695247	1	2	1	26	1
2	3	11.407565	1	2	1	34	0
3	4	10.819778	1	2	0	37	0
4	5	10.819778	0	2	0	57	0

Again, the `ID` and `default_next_month` columns are filtered to create the features dataset. The `default_next_month` column is defined as the target dataset.

```
x_cols = [i for i in df.columns if i not in ('ID', 'default_next_month')]
X = df[x_cols]
y = df['default_next_month']
```

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)
```

Next, we import the `SMOTEENN` module and create an instance of `SMOTEENN`, which resamples the dataset.

```
from imblearn.combine import SMOTEENN
smote_enn = SMOTEENN(random_state=0)
X_resampled, y_resampled = smote_enn.fit_resample(X, y)
```

Again, we use a `LogisticRegression` model to generate predictions.

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

As before, the evaluation metrics are generated. First, the `confusion_matrix` is generated.

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

Next, the `balanced_accuracy_score` is generated.

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

Finally, we print the classification report.

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

	pre	rec	spe	f1	geo	iba	sup
0	0.83	0.64	0.56	0.72	0.60	0.36	5832
1	0.31	0.56	0.64	0.40	0.60	0.35	1668
avg / total	0.72	0.62	0.57	0.65	0.60	0.36	7500

Resampling with SMOTEENN did not work miracles, but some of the metrics show an improvement over undersampling.

Which of the following best describes the difference between SMOTE and SMOTEENN?

- ◯ SMOTE generates synthetic observations, whereas SMOTEENN does not.
- ◯ SMOTE involves only oversampling, whereas SMOTEENN involves only undersampling.
- ◯ SMOTE does not involve undersampling.

Check Answer

Finish ▶