0122_oversampling_and_smote

April 12, 2019

1 Oversampling to correct for inbalanced data using naive sampling or SMOTE

Machine learning can have poor performance for minority classes. One method of improving performance is to balance out the number of examples between different classes. Here two methods are described:

1. Resampling from the minority classes to give the same number of examples as the majority class.
2. SMOTE (Synthetic Minority Over-sampling Technique): creating synthetic data based on creating new data points that are mid-way between two near neighbours in any particular class.

SMOTE uses imblearn See: https://imbalanced-learn.org

Install with: `pip install -U imbalanced-learn`, or `conda install -c conda-forge imbalanced-learn`

Reference

N. V. Chawla, K. W. Bowyer, L. O.Hall, W. P. Kegelmeyer, "SMOTE: synthetic minority oversampling technique," Journal of artificial intelligence research, 16, 321-357, 2002.

2 Create dummy data

First we will create some unbalanced dummy data: Classes 0, 1 and 2 will represent 1%, 5% and 94% of the data respectively.

```
In [1]: from sklearn.datasets import make_classification
        X, y = make_classification(n_samples=5000, n_features=2, n_informative=2,
                                   n_redundant=0, n_repeated=0, n_classes=3,
                                   n_clusters_per_class=1,
                                   weights=[0.01, 0.05, 0.94],
                                   class_sep=0.8, random_state=0)
```

Count instances of each class

```
In [2]: from collections import Counter
        print(sorted(Counter(y).items()))
```

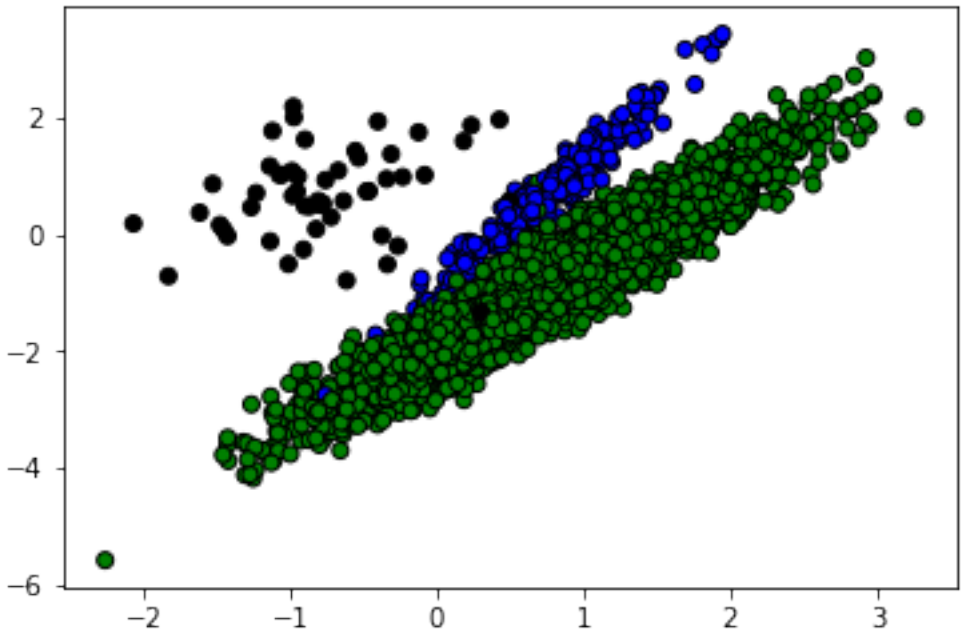
```
[(0, 64), (1, 262), (2, 4674)]
```

2.0.1 Define function to plot data

```
In [3]: import matplotlib.pyplot as plt

        def plot_classes(X,y):
            colours = ['k','b','g']
            point_colours = [colours[val] for val in y]
            X1 = X[:,0]
            X2 = X[:,1]
            plt.scatter(X1,X2,
                        facecolor = point_colours,
                        edgecolor = 'k')
            plt.show()
```

```
In [4]: plot_classes(X,y)
```



2.1 Oversample with naive sampling to match numbers in each class

With naive resampling we repeatedly randomly sample from the minority classes and add that the new sample to the existing data set, leading to multiple instances of the minority classes.This builds up the number of minority class samples.

```
In [5]: from imblearn.over_sampling import RandomOverSampler
        ros = RandomOverSampler(random_state=0)
        X_resampled, y_resampled = ros.fit_resample(X, y)
```

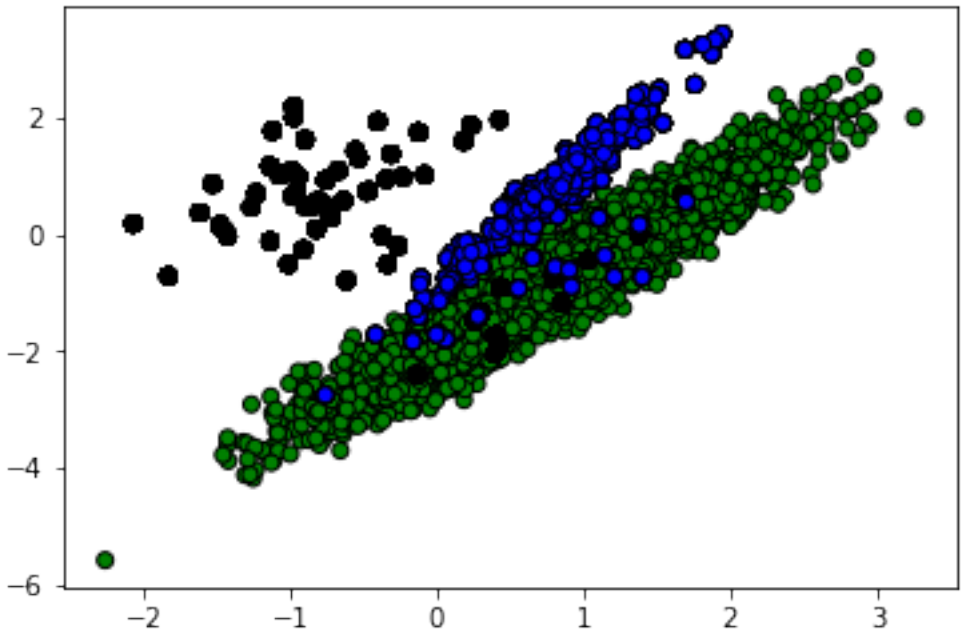
Count instances of each class in the augmented data

```
In [6]: from collections import Counter
        print(sorted(Counter(y_resampled).items()))
```

```
[(0, 4674), (1, 4674), (2, 4674)]
```

Plot augmented data (it looks the same as the original as points are overlaid).

```
In [7]: plot_classes(X_resampled,y_resampled)
```



3 SMOTE with continuous variables

SMOTE (synthetic minority oversampling technique) works by finding two near neighbours in a minority class, producing a new point midway between the two existing points and adding that new point in to the sample. The example shown is in two dimensions, but SMOTE will work across multiple dimensions (features). SMOTE therefore helps to 'fill in' the feature space occupied by minority classes.

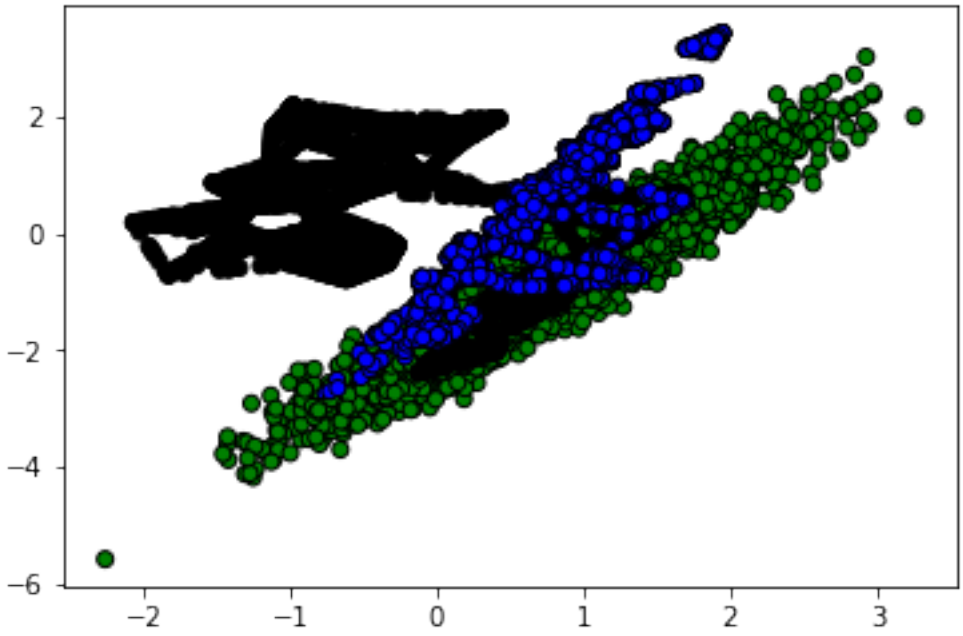
```
In [8]: from imblearn.over_sampling import SMOTE
        X_resampled, y_resampled = SMOTE().fit_resample(X, y)

        # Count instances of each class
        from collections import Counter
        print(sorted(Counter(y_resampled).items()))
```

```
[(0, 4674), (1, 4674), (2, 4674)]
```

Plot augmented data (note minority class data points now exist in new spaces).

```
In [9]: plot_classes(X_resampled,y_resampled)
```



4 SMOTE with mixed continuous and binary/categorical values

It is not possible to calculate a 'mid point' between two points of binary or categorical data. An extension to the SMOTE method allows for use of binary or categorical data by taking the most common occurring category of nearest neighbours to a minority class point.

```
In [10]: # create a synthetic data set with continuous and categorical features
         import numpy as np
         rng = np.random.RandomState(42)
         n_samples = 50
         X = np.empty((n_samples, 3), dtype=object)
         X[:, 0] = rng.choice(['A', 'B', 'C'], size=n_samples).astype(object)
         X[:, 1] = rng.randn(n_samples)
         X[:, 2] = rng.randint(3, size=n_samples)
         y = np.array([0] * 20 + [1] * 30)

In [11]: # Count instances of each class
         print(sorted(Counter(y).items()))

[(0, 20), (1, 30)]
```

```
In [12]: # Show last 10 values of X
         print (X[-10:])
```

```
[['A' 1.4689412854323924 2]
 ['C' -1.1238983345400366 0]
 ['C' 0.9500053955071801 2]
 ['A' 1.7265164685753638 1]
 ['A' 0.4578850770000152 0]
 ['C' -1.6842873783658814 0]
 ['B' 0.32684522397001387 0]
 ['A' -0.0811189541586873 2]
 ['B' 0.46779475326315173 1]
 ['B' 0.7361223506692577 0]]
```

Use SMOTENC to create new data points.

```
In [13]: from imblearn.over_sampling import SMOTENC
         smote_nc = SMOTENC(categorical_features=[0, 2], random_state=0)
         X_resampled, y_resampled = smote_nc.fit_resample(X, y)
```

```
In [14]: # Count instances of each class
         print(sorted(Counter(y_resampled).items()))
```

```
[(0, 30), (1, 30)]
```

```
In [15]: # Show last 10 values of X
         # (SMOTE data points are added to the end of the original data set)

         print (X_resampled[-10:])
```

```
[['C' -1.0600505672469849 1]
 ['C' -0.36965644259183145 1]
 ['A' 0.1453826708354494 2]
 ['C' -1.7442827953859052 2]
 ['C' -1.6278053447258838 2]
 ['A' 0.5246469549655818 2]
 ['B' -0.3657680728116921 2]
 ['A' 0.9344237230779993 2]
 ['B' 0.3710891618824609 2]
 ['B' 0.3327240726719727 2]]
```