DK911b-Machine_Learning-Lab2

October 9, 2019

1 Scikit-Lab 2

- scikit-learn is the leading machine learning software in Python
- scikit-learn is a project started in Paris, Inria and Telecom Paris
- scilkit-learn is easy to use and extend

1.0.1 ZHANG Xin

2 Task 1:

2.0.1 - Implement a majority class classifier: a classifier that predicts the class label that is most frequent in the dataset.

- Classifiers in scikit-learn has two main methods:
  - Build a model: fit(self, X, Y)
  - Make a prediction: predict(self, X)
- Template for implementing classifier is given:

```
In [37]: import numpy as np

         class NewClassifier:
             def __init__(self):
                 self.num = 0
             def fit(self, X, Y):
                 if isinstance(Y, np.ndarray) and len(Y.shape) > 1 and Y.shape[1] > 1:
                     raise NotImplementedError('Majority class classifier not supported')

                 counts = np.bincount(Y)
                 self.num = np.argmax(counts)

                 return self

             def predict(self, X):
                 Y = []
                 for i in range(0,X.shape[0]):
                     Y.append(self.num)
```

```
            return Y
```

Test if the classifier works

```
In [47]: clf = NewClassifier()
         clf.fit(np.empty(shape=(3,3)),np.array([0,0,1]))
         clf.predict(np.empty(shape=(4,4)))
```

```
Out[47]: [0, 0, 0, 0]
```

3 Task 2:

3.0.1 - Implement k-fold cross validation

```
In [60]: from sklearn.metrics import accuracy_score
         from sklearn.utils import shuffle
         from scipy.sparse import coo_matrix

         def cross_validation(clf, dataset, n_folds):
             #data initialization
             X = dataset.data
             y = dataset.target
             X_sparse = coo_matrix(X)
             data_X, X_sparse, data_y = shuffle(X, X_sparse, y)
             k = n_folds
             training_list = []
             class_list = []
             spilt_size = int(data_X.shape[0] / k)

             #Spilt the data and classes into N parts in two lists
             for i in range(0, k):
                 training_list.append(data_X[i * spilt_size:(i + 1) * spilt_size])
                 class_list.append(data_y[i * spilt_size:(i + 1) * spilt_size])

             #begin the validation with the split from former step
             sum_accuracy = 0
             for i in range(0, k):
                 temp_training = []
                 temp_class = []
                 temp_test = []
                 temp_class_test = []
                 for j in range(0, k):
                     if (j != i):
                         temp_training.extend(training_list[j])
                         temp_class.extend(class_list[j])
                     else:
```

```
                temp_test.extend(training_list[j])
                temp_class_test.extend(class_list[j])
        temp_training = np.concatenate(temp_training, axis= 0)
        temp_training = np.reshape(temp_training, (spilt_size * (k -1),data_X.shape[1
        temp_test = np.concatenate(temp_test, axis = 0)
        temp_test = np.reshape(temp_test, (spilt_size,data_X.shape[1]))
        temp_class = np.array(temp_class)
        temp_class_test = np.array(temp_class_test)
        clf.fit(temp_training, temp_class)
        temp_predicted = clf.predict(temp_test)
        sum_accuracy += accuracy_score(temp_class_test, temp_predicted)
    score = sum_accuracy/k
    return score
```

The code can be way shorter if `sklearn.model_selection.train_test_split(*arrays, **options)` is used. But it's better to implement from basic python and numpy commands.

Test if it works properly

```
In [48]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn import datasets
         knn = KNeighborsClassifier()
         iris = datasets.load_iris()

         cross_validation(knn,iris,7)
```

```
Out[48]: 0.9251700680272109
```

4 Task 3:

4.0.1 Use the majority class classifier to evaluate one dataset, and explain the evaluation results:

- https://scikit-learn.org/stable/datasets/index.html

```
In [99]: clf = NewClassifier()
         cross_validation(clf,iris,5)
```

```
Out[99]: 0.26
```

To analyse the result, let us firstly check the number of each classes in the dataset.

```
In [82]: y = iris.target
         counts_1 = np.bincount(y)
         print(counts_1)
```

```
[50 50 50]
```

So we can see that in the iris dataset, the number of the three classes (0,1,2) are all 50. So, In each step of the cross-validation, we suppose that we take a samples from class A, b from class B, c from class C for the test. So we have $a + b + c = \frac{150}{k}$, let us suppose $a > b > c$, so c will be the majority class in the training as less c is taken into test set. And $c < \frac{50}{k}$, so we get

$$accurary = \frac{c \times k}{150} < \frac{50}{150} < \frac{1}{3}$$

So for the iris dataset with the majority class classifier, the accuracy cannot be larger than $\frac{1}{3}$, or for each dataset, the accuracy cannot be larger than $\frac{1}{No.classes}$

5 Task 4: *OPTIONAL*

5.0.1 - Implement another classifier with higher performance than the majority class classifier, evaluate it and comment the results

So, as we have analysed earlier, if we just want a better performance form the `iris` dataset, we can just take the minority class instead.

```
In [131]: class BetterClassifierForIris:
              def __init__(self):
                  self.num = 0
              def fit(self, X, Y):
                  if isinstance(Y, np.ndarray) and len(Y.shape) > 1 and Y.shape[1] > 1:
                      raise NotImplementedError('Majority class classifier not supported')

                  counts = np.bincount(Y)
                  self.num = np.argmin(counts)

                  return self

              def predict(self, X):
                  Y = []
                  for i in range(0,X.shape[0]):
                      Y.append(self.num)

                  return Y

          clf = BetterClassifierForIris()
          cross_validation(clf,iris,10)

Out[131]: 0.48
```

For a more general case, we can just perform a linear regression with X and y and keep the `int` value by rounding.

```
In [217]: class BetterClassifier:
              def __init__(self, alpha=0.03, n_iter=1500):
```

```python
        self.alpha = alpha
        self.n_iter = n_iter
        self.params = []
        self.coef_ = None
        self.intercept_ = None
        self.X = []
        self.y = []
        self.n_samples = 0
        self.n_features = 0

    def fit(self, X, y):
        self.n_samples = len(y)
        self.n_features = np.size(X, 1)
        self.params = np.zeros((self.n_features + 1, 1))
        self.X = np.hstack((np.ones(
            (self.n_samples, 1)), (X - np.mean(X, 0)) / np.std(X, 0)))
        self.y = y[:, np.newaxis]
        for i in range(self.n_iter):
            self.params = self.params - (self.alpha/self.n_samples) * \
            self.X.T @ (self.X @ self.params - self.y)

        self.intercept_ = self.params[0]
        self.coef_ = self.params[1:]

        return self

    def predict(self, X):
        n_samples = np.size(X, 0)
        y = np.hstack((np.ones((n_samples, 1)), (X-np.mean(X, 0)) \
                            / np.std(X, 0))) @ self.params
        y = [int(i) for i in y.T[0]]
        return y
```

In [219]:
```python
clf = BetterClassifier()
cross_validation(clf,iris,10)
```

Out[219]: 0.6533333333333333

We can see that the performance is better with a simple linear regression.