parth.pandey13103447@gmail.com_6

February 10, 2020

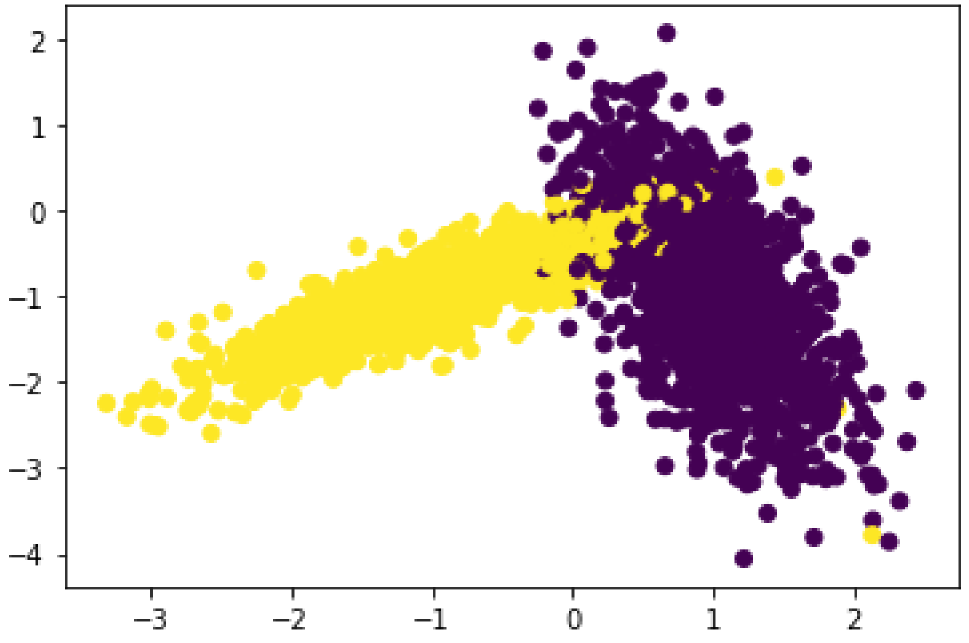
1 Assignment

1.1 Creating Dataset

[3]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances


x,y = make_classification(n_samples=10000, n_features=2, n_informative=2,␣
 ↪n_redundant= 0, n_clusters_per_class=1, random_state=60)
X_train, X_test, y_train, y_test =␣
 ↪train_test_split(x,y,stratify=y,random_state=42)

# del X_train,X_test
```

1.2 Plotting Data

[4]:
```
%matplotlib inline
import matplotlib.pyplot as plt
colors = {0:'red', 1:'blue'}
plt.scatter(X_test[:,0], X_test[:,1],c=y_test)
plt.show()
```



1.3 Implementing Custom RandomSearchCV

[5]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

1.3.1 Implementing the RandomSeacrchCV function

[6]:
```
def RandomSearchCV(x_train, y_train, classifier, param_range, folds):
    train_Scores = []
    test_Scores = []
    params = np.sort(np.random.choice(np.arange(3,35,2),10,replace=False))
    n = len(x_train)
    group= []
    z = n//folds
    for i in range(folds):
        group.append(n//folds)
    if n % folds != 0:
        group[-1] = group[-1] + (n - sum(group))
    else:
        group[-1] += 1
    group = np.cumsum(group)
    test_indices = set()
    diff = True
    for k in tqdm(params):
        test_scores_fold = []
        train_scores_fold = []
# Now choosing the data set
# To attain proper randomization added temp as the set of all indices and
# removing the used test indices.
        temp = set(np.arange(n))
        for fold in range(folds):
            if fold == 0:
                test_indices = np.random.choice(np.arange(n),z,replace=False)
            else:
                temp.difference_update(test_indices)
                test_indices = np.random.choice(list(temp),z,replace=True)
            train_indices = list(set(list(range(0,n))) - set(test_indices))
            x_tr = x_train[train_indices]
            x_ts = x_train[test_indices]
            y_tr = y_train[train_indices]
            y_ts = y_train[test_indices]
            classifier.n_neighbors = k
            classifier.fit(x_tr,y_tr)
            y_pr = classifier.predict(x_ts)
            test_scores_fold.append(accuracy_score(y_ts,y_pr))
            y_pr = classifier.predict(x_tr)
            train_scores_fold.append(accuracy_score(y_tr,y_pr))
        train_Scores.append(np.mean(np.array(train_scores_fold)))
        test_Scores.append(np.mean(np.array(test_scores_fold)))
    return train_Scores,test_Scores,params
```

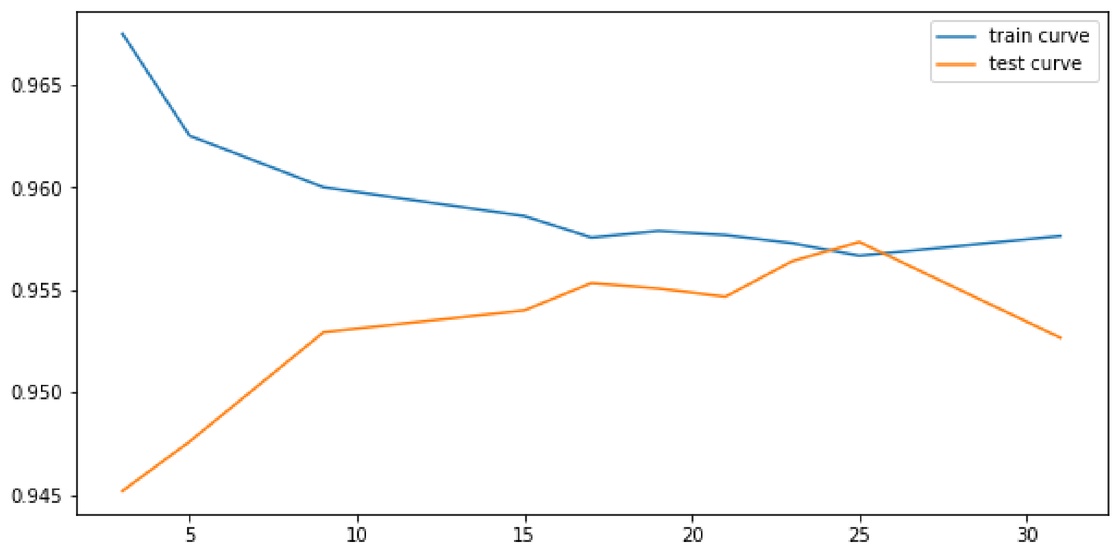
1.3.2 Calling the Function

[7]:
```
param_range = (1,50)
train_scores , test_scores , params = RandomSearchCV(X_train,y_train,␣
 ↪KNeighborsClassifier(), param_range, 3 )
```

```
100%|          | 10/10 [00:08<00:00,  1.24it/s]
```

1.3.3 Plotting the trainscores with the test scores for best k

[8]:
```
plt.figure(figsize=(10,5))
plt.plot(params,train_scores,label='train curve')
plt.plot(params,test_scores,label='test curve')
plt.legend()
plt.show()
```



[9]:
```
print(params)
```

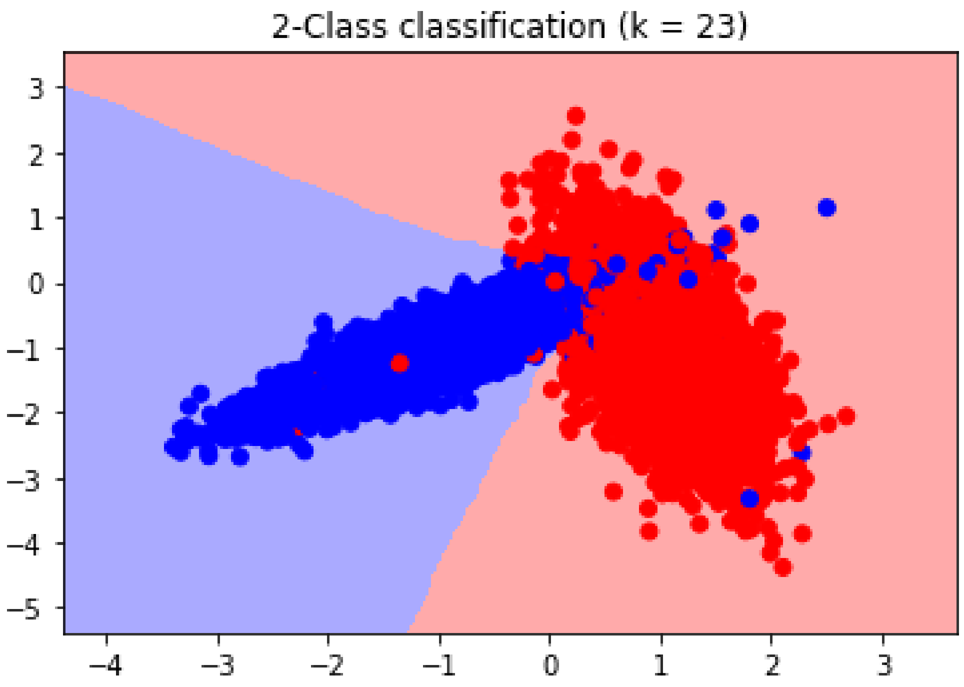
```
[ 3  5  9 15 17 19 21 23 25 31]
```

1.3.4 Implementing the Decision Boundary function

[10]:
```
def plot_decision_boundary(x1,x2,y,clf):
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

    xmin, xmax = x1.min()-1, x1.max()+1
    ymin, ymax = x2.min()-1, x2.max()+1

    xx,yy = np.meshgrid(np.arange(xmin,xmax,0.02),np.arange(ymin,ymax,0.02))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
    # Plot also the training points
    plt.scatter(x1, x2, c=y, cmap=cmap_bold)

    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("2-Class classification (k = %i)" % (clf.n_neighbors))
    plt.show()
```

1.3.5 Calling the Function

[11]:
```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 23)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```



[ ]:
```
```