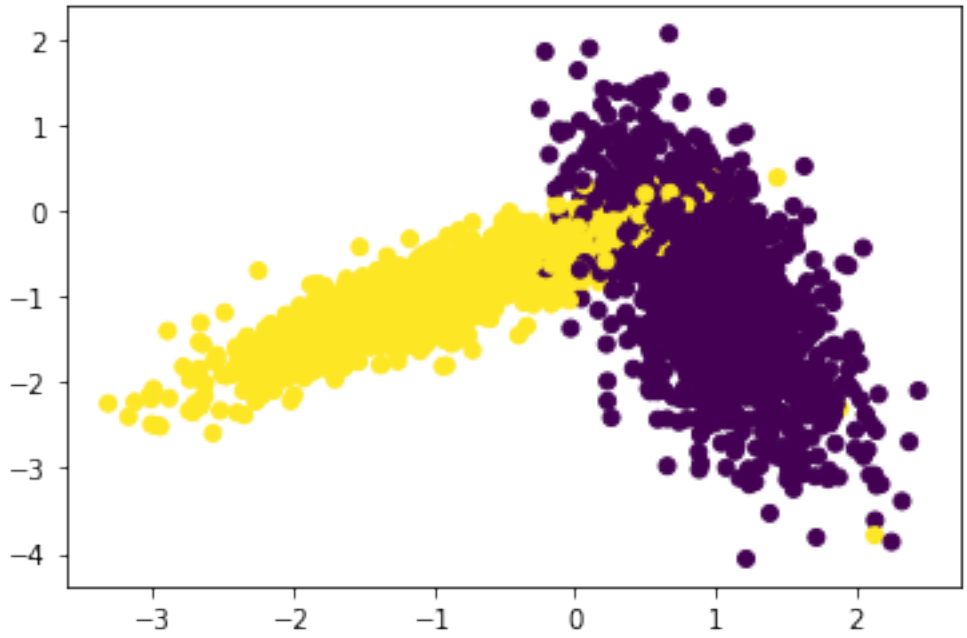
RandomSearchCV_with_KfoldCV_Assignment

January 24, 2020

[199]:
```
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

from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings



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
```

[200]:
```
%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 Implementing Custom RandomSearchCV

[201]:
```
def split_indices(data, num_parts):
    a=len(data)
    #Getting quotient and reminder
    q=a//num_parts
    r=a%num_parts
    parts=[]

    for i in range(num_parts):
        parts.append(q)
    for i in range(r):
        parts[i]+=1
    cum_sum=np.cumsum(parts)
    lst=[]

    for i in range(len(cum_sum)):
        if i ==0:

            lst.append((0,cum_sum[i]-1))
        else:
            lst.append((cum_sum[i-1],cum_sum[i]-1))
    return lst
```

[210]:
```
def RandomSearchCV(x_train,y_train,classifier, param_range, folds):
    train_scores = []
    test_scores  = []
    params=[]
    while len(params) != 10:
        val=int((np.random.randint(param_range[0],param_range[1],1)))
        if val not in params:
            params.append(val)
    params=sorted(params)
    print(params)

    lst_of_indices=split_indices(x_train,folds)

    for k in tqdm(params):
        trainscores_folds = []
        testscores_folds  = []

        for j in range(0, folds):
            # check this out: https://stackoverflow.com/a/9755548/4084039

            test_indices_range  = lst_of_indices[j]
            
↪test_indices=list(range(test_indices_range[0],test_indices_range[1]+1))
            indices=list(range(0,len(x_train)))
            train_indices= [i for i in indices if i not in test_indices]

            X_train = x_train[train_indices]
            Y_train = y_train[train_indices]
            X_test  = x_train[test_indices]
            Y_test  = y_train[test_indices]

            classifier.n_neighbors = k
            classifier.fit(X_train,Y_train)

            Y_predicted = classifier.predict(X_test)
            testscores_folds.append(accuracy_score(Y_test, Y_predicted))

            Y_predicted = classifier.predict(X_train)
            trainscores_folds.append(accuracy_score(Y_train, Y_predicted))

        train_scores.append(np.mean(np.array(trainscores_folds)))
        test_scores.append(np.mean(np.array(testscores_folds)))

    return train_scores, test_scores, params
```

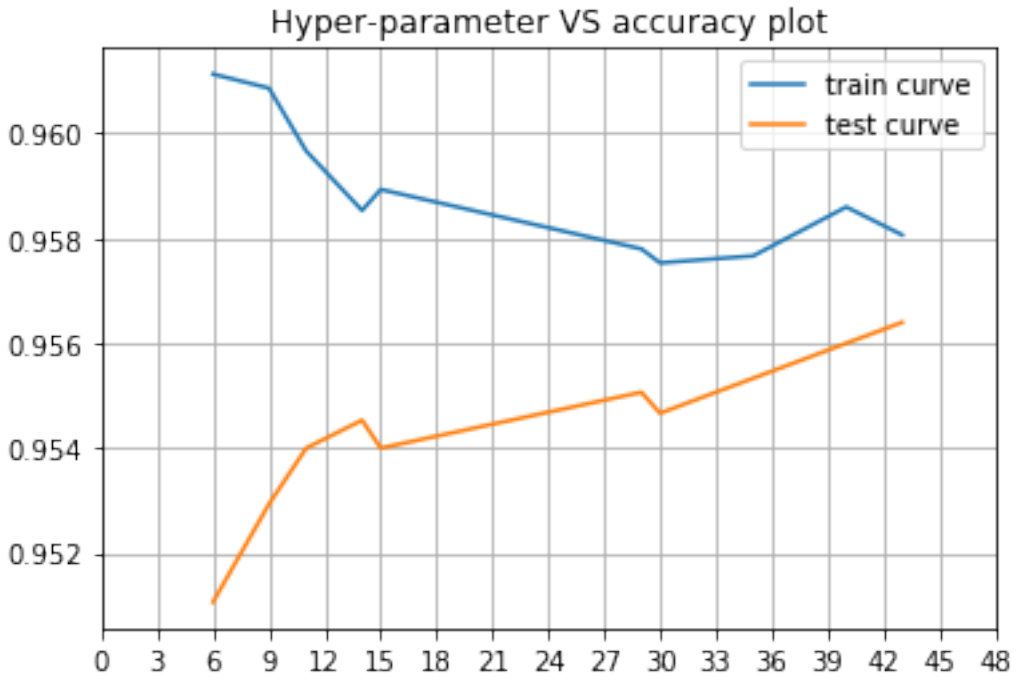
[237]:
```
neigh = KNeighborsClassifier()
params_range=(1,51)
```

```
folds=3

train_score , cv_score, num_neighbours= RandomSearchCV(X_train, y_train, neigh,␣
↪params_range, folds)
```

```
[6, 9, 11, 14, 15, 29, 30, 35, 40, 43]
```

```
100%|
    | 10/10 [00:12<00:00,  1.27s/it]
```

[238]:
```
plt.plot(num_neighbours,train_score, label='train curve')
plt.plot(num_neighbours,cv_score, label='test curve')
plt.title('Hyper-parameter VS accuracy plot')
plt.xticks(np.arange(0, 51, step=3))
plt.grid()
plt.legend()
plt.show()
```



As per the plot, best hyperparameter is k = 43

[215]:
```
# understanding this code line by line is not that importent
def plot_decision_boundary(X1, X2, y, clf):
        # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
```

```
    x_min, x_max = X1.min() - 1, X1.max() + 1
    y_min, y_max = X2.min() - 1, X2.max() + 1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max,␣
↪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()
```

[245]:
```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 43)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```