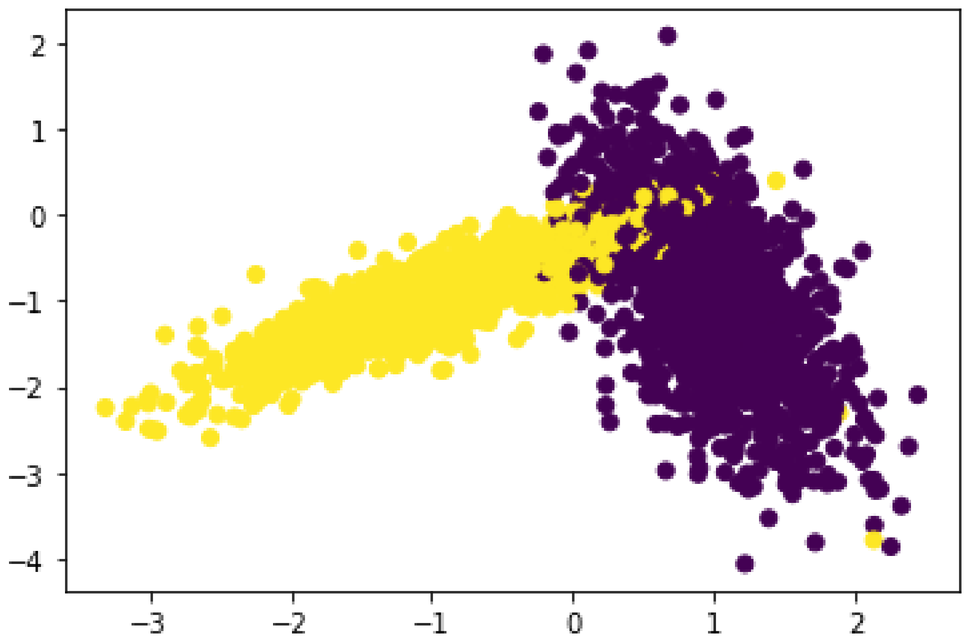
RandomSearchCV_assn4_solution

August 23, 2020

[1]:
```
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
```

[2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
colors = {0:'orange', 1:'blue'}
plt.scatter(X_test[:,0], X_test[:,1],c=y_test)
plt.show()
```



1 Implementing Custom RandomSearchCV

[11]:
```
def if_present(list1,list2):
    if(all(i in list1 for i in list2)):
        return True
    else:
        return False

#2.divide numbers ranging from  0 to len(X_train) into groups= folds
def randomly_divide_into_groups(x_train, folds):
    groups = []
    rs = r.sample(range(0,len(x_train)), int(len(x_train)/folds))
    total = [i for i in rs]
    groups.append(rs)
    for i in range(folds-1):
        rs = r.sample(range(0,len(x_train)), int(len(x_train)/folds))
        while (if_present(rs, total)):
            rs = r.sample(range(0,len(x_train)), int(len(x_train)/folds))
        else:
            groups.append(rs)
            total.append(rs)
    return groups
```

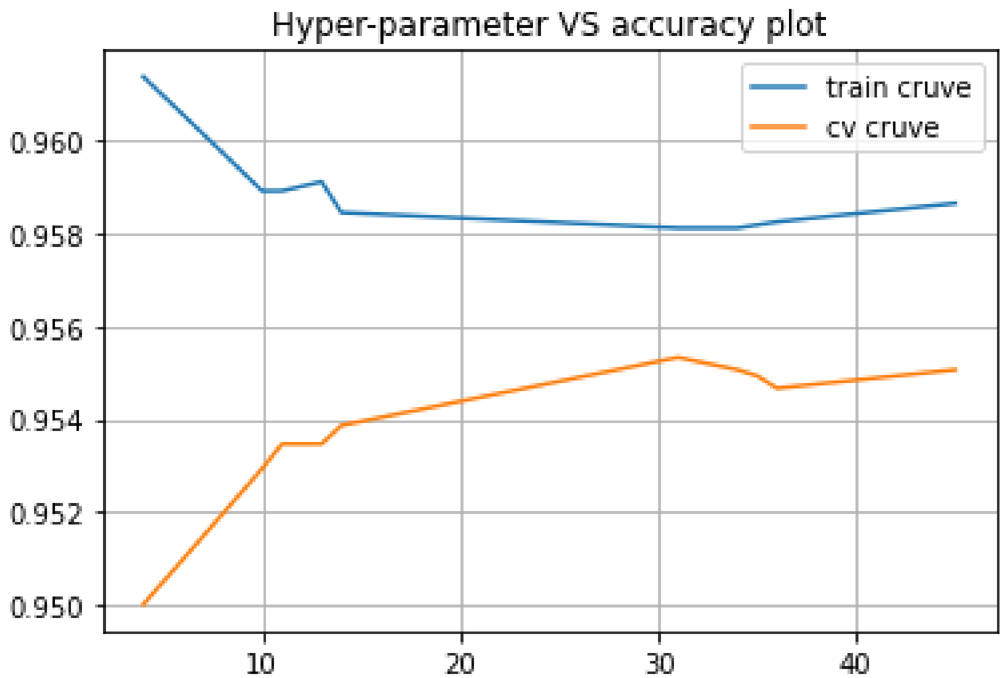
[12]:

```
    # x_train: its numpy array of shape, (n,d)
    # y_train: its numpy array of shape, (n,) or (n,1)
    # classifier: its typically KNeighborsClassifier()
    # param_range: its a tuple like (a,b) a < b
    # folds: an integer, represents number of folds we need to devide the data␣
↪and test our model
def RandomSearchCV(x_train,y_train,classifier, param_range, folds):
     train_score = []
     cv_scores = []
     #1.generate 10 unique values(uniform random distribution) in the given␣
↪range "param_range" and store them as "params"
     params = r.
↪sample(range(param_range['n_neighbors'][0],param_range['n_neighbors'][1]),10)
     params.sort()
     print('Hyperparameter: ', params)
     groups = randomly_divide_into_groups(x_train, folds)
     #3.for each hyperparameter that we generated in step 1:
     for h in params:
         trainscores_folds = []
         testscores_folds  = []
         for g in range(0,folds):
         #Extracting test indices
             test_indices = groups[g]
             train_indices = list(set(list(range(1, len(x_train)))) -␣
↪set(test_indices))
    # selecting the data points based on the train_indices and 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 = h
             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))
    # find the mean of train accuracies of above 3 steps and store in a list␣
↪"train_scores"
         train_score.append(np.mean(np.array(trainscores_folds)))
    # find the mean of test accuracies of above 3 steps and store in a list␣
↪"test_scores"
         cv_scores.append(np.mean(np.array(testscores_folds)))
     return train_score,cv_scores,params
```

[24]:

```
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random as r
import warnings
warnings.filterwarnings("ignore")

neigh = KNeighborsClassifier()

param_range = {'n_neighbors': (0,50)}

folds = 3
# 5. call function RandomSearchCV(x_train,y_train,classifier, param_range,␣
↪folds) and store the returned values into "train_score", and "cv_scores"
if (param_range['n_neighbors'][0] < param_range['n_neighbors'][1]):
    trainscores,cvscores,params = RandomSearchCV(X_train, y_train, neigh,␣
↪param_range, folds)

# 6. plot hyper-parameter vs accuracy plot as shown in reference notebook and␣
↪choose the best hyperparameter
plt.plot(params, trainscores, label='train cruve')
plt.plot(params, cvscores, label='cv cruve')
plt.title('Hyper-parameter VS accuracy plot')
plt.legend()
plt.grid()
plt.show()
```

```
Hyperparameter:  [4, 10, 11, 13, 14, 31, 34, 35, 36, 45]
```



[14]:
```
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()
```

[26]:

```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 31)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```