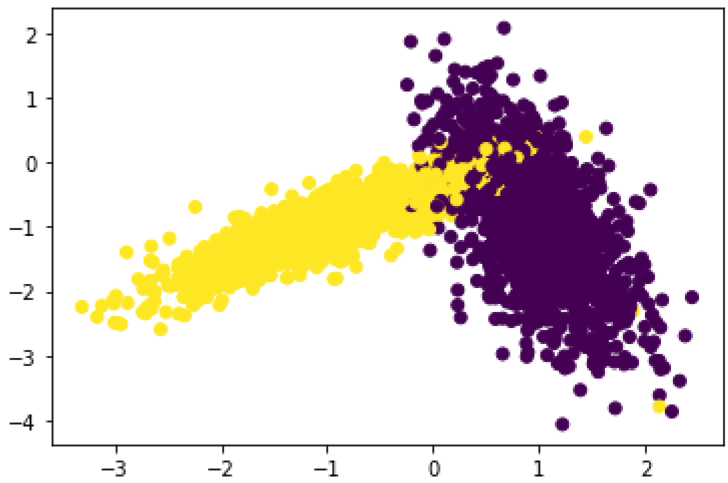
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
```

```
x,y = make_classification(n_samples=10000, n_features=2, n_informative=2, n_redundant= 0,
X_train, X_test, y_train, y_test = train_test_split(x,y,stratify=y,random_state=42)
```

```
%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()
```



Implementing Custom RandomSearchCV

```
def RandomSearchCV(x_train,y_train,classifier, param_range, folds):
    # 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


    #1.generate 10 unique values(uniform random distribution) in the given range "param_
    # ex: if param_range = (1, 50), we need to generate 10 random numbers in range 1 to
    #2.devide numbers ranging from  0 to len(X_train) into groups= folds
    # ex: folds=3, and len(x_train)=100, we can devide numbers from 0 to 100 into 3 grou
```

```
      group 1: 0-33, group 2:34-66, group 3: 67-100
    #3.for each hyperparameter that we generated in step 1:
        # and using the above groups we have created in step 2 you will do cross-validat

        # first we will keep group 1+group 2 i.e. 0-66 as train data and group 3: 67-100
          test accuracies

        # second we will keep group 1+group 3 i.e. 0-33, 67-100 as train data and group
          train and test accuracies

        # third we will keep group 2+group 3 i.e. 34-100 as train data and group 1: 0-33
          test accuracies
        # based on the 'folds' value we will do the same procedure

        # find the mean of train accuracies of above 3 steps and store in a list "train_
        # find the mean of test accuracies of above 3 steps and store in a list "test_sc
    #4. return both "train_scores" and "test_scores"

# 5. call function RandomSearchCV(x_train,y_train,classifier, param_range, folds) and st
# 6. plot hyper-parameter vs accuracy plot as shown in reference notebook and choose the
# 7. plot the decision boundaries for the model initialized with the best hyperparameter
```

```
from sklearn.metrics import accuracy_score

def RandomSearchCV(x_train,y_train,classifier, param_range, folds):
  params = sorted([int(i) for i in random.sample(range(*(param_range)),10)]) #generating 1
  trainscores = []
  testscores  = []

  for k in tqdm(params):
      trainscores_folds = []
      testscores_folds  = []
      #to get the test and train indices we are applying the logic as,
        # j = for each folds
        # n = len(x_train)
        # k = total no of folds
        #test_indices = range(j*(n/k), (j+1)(n/k))
        #train_indices = range(0,n) - (test_indices)
      for j in range(folds):
        test_indices  = range(j * int(len(x_train)/folds), (j+1) * int(len(x_train)/folds)
        train_indices = list(set(range(0, 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]
```

```
            # defining the classifier
            classifier.n_neighbors = k
            classifier.fit(X_train,Y_train)

            #appending the test and train values into trainscores_folds and testscores_folds

            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))

          trainscores.append(np.mean(np.array(trainscores_folds))) #extracting mean of all the
          testscores.append(np.mean(np.array(testscores_folds)))   #extracting mean of all the

      #returning the train and test scores, as well as params value which we will be using it
      return trainscores,testscores,params
```

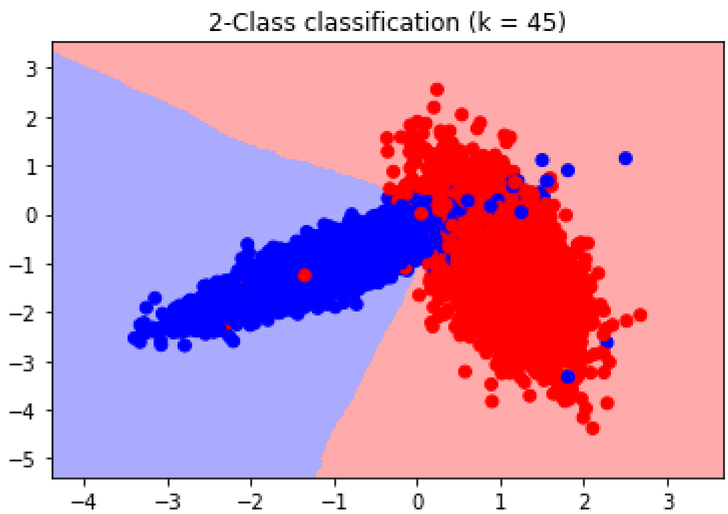
```
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings
warnings.filterwarnings("ignore")


neigh = KNeighborsClassifier()

param = (1,50)
folds = 3

trainscores,testscores,params = RandomSearchCV(X_train, y_train, neigh, param, folds)

plt.plot(params,trainscores, label='train cruve')
plt.plot(params,testscores, label='test cruve')
plt.title('Hyper-parameter VS accuracy plot')
plt.legend()
plt.show()
```

```
100%|██████████| 10/10 [00:06<00:00,  1.48it/s]
```

```
#So after running multiple instances of the above code, I could say that the best k would
```

test cruve

```
# understanding this code line by line is not that important
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()


from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 45)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```



Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.