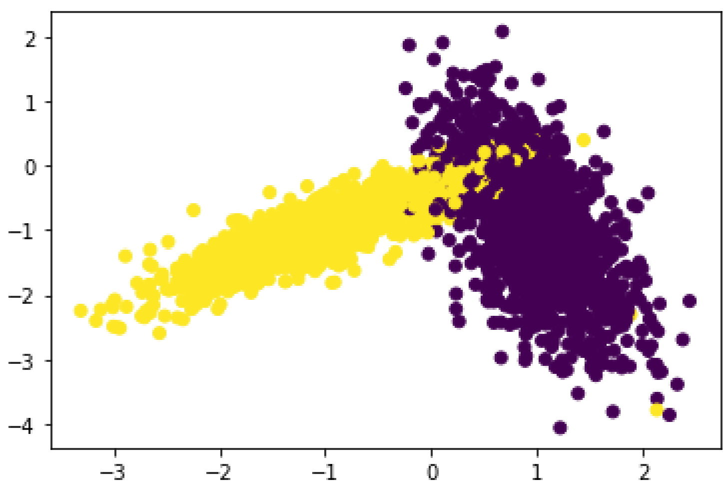
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
import random
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.neighbors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

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

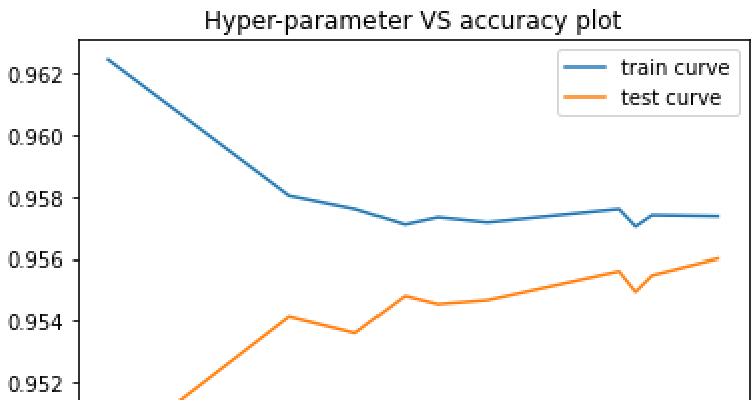
```
%matplotlib inline
import matplotlib.pyplot as plt
colors = {0:'red', 1:'blue'}
plt.scatter(X_test[:,0], X_test[:,1],c=y_test)
plt.show()
```



Implementing Custom RandomSearchCV

```
def RandomSearchCV(x_train,y_train, classifier,param_range ,folds):
    trainscores = []
    testscores = []

    # Generating 10 unique values from given parameter range
    global params
    params = random.sample(range(param_range[0], param_range[1]), 10)
    params=sorted(params)
```

```
    x_train_split = []
    y_train_split = []
    # diving X_train into groups as per no of folds
    for i in range(0, len(x_train), int(len(x_train)/folds)):
        x_train_split.append(x_train[i:i+int(len(x_train)/folds)])
        y_train_split.append(y_train[i:i+int(len(y_train)/folds)])

  #for each hyperparameter(K) that we generated , dividing dataset into Train and Cross Valid

    for k in params:
        trainscores_folds = []
        testscores_folds  = []

        for group in range(len(x_train_split)):
            x_train_group =  np.concatenate(x_train_split[0:group] + x_train_split[group+1:])
            x_cv_group = x_train_split[group]
            y_train_group = np.concatenate(y_train_split[0:group] + y_train_split[group+1:])
            y_cv_group = y_train_split[group]

            #KNN Classifier
            classifier.n_neighbors = k
            classifier.fit(x_train_group, y_train_group) # applying KNN Classifier on each gr

            # Calculating the accuracy of train group based on predicted label and actual lab
            Y_predicted = classifier.predict(x_train_group)
            trainscores_folds.append(accuracy_score(y_train_group, Y_predicted))

            # Calculating the accuracyog CV group based on predicted label and actual label
            Y_predicted = classifier.predict(x_cv_group)
            testscores_folds.append(accuracy_score(y_cv_group, Y_predicted))

        #  mean of train and test accuracies
        trainscores.append(np.mean(np.array(trainscores_folds)))
        testscores.append(np.mean(np.array(testscores_folds)))

    return trainscores, testscores

param_range=(1,50)
classifier = KNeighborsClassifier()
train_score,cv_scores=RandomSearchCV(X_train,y_train, classifier,param_range, folds = 5)

# Plotting hyper-parameter(k) vs accuracy
plt.plot(params,train_score, label='train curve')
plt.plot(params,cv_scores, label='test curve')
plt.title('Hyper-parameter VS accuracy plot')
plt.legend()
plt.show()
```

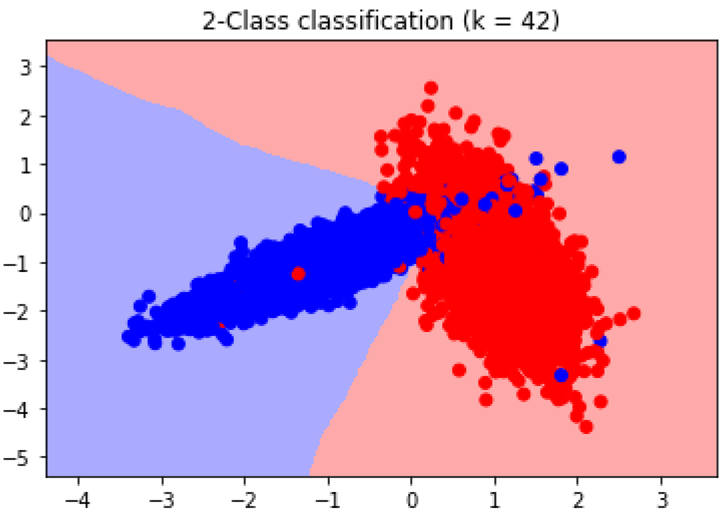


Inference:

From this plot it can be seen that for K=42 the accuracy is highest hence choosing K=42

```
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 = 42)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```


2-Class classification (k = 42)
3
2
1
0
−1
−2
−3
−4
−5
−4
−3
−2
−1
0
1
2
3