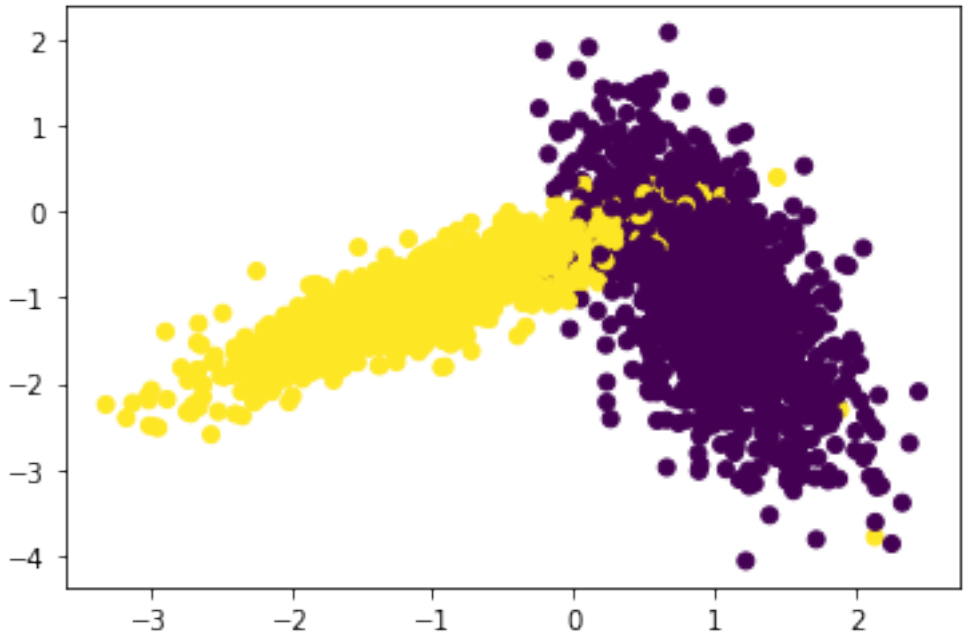
# Assignment_4_Instructions

February 16, 2022

[2]:
```
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
```

[3]:
```
%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()
```



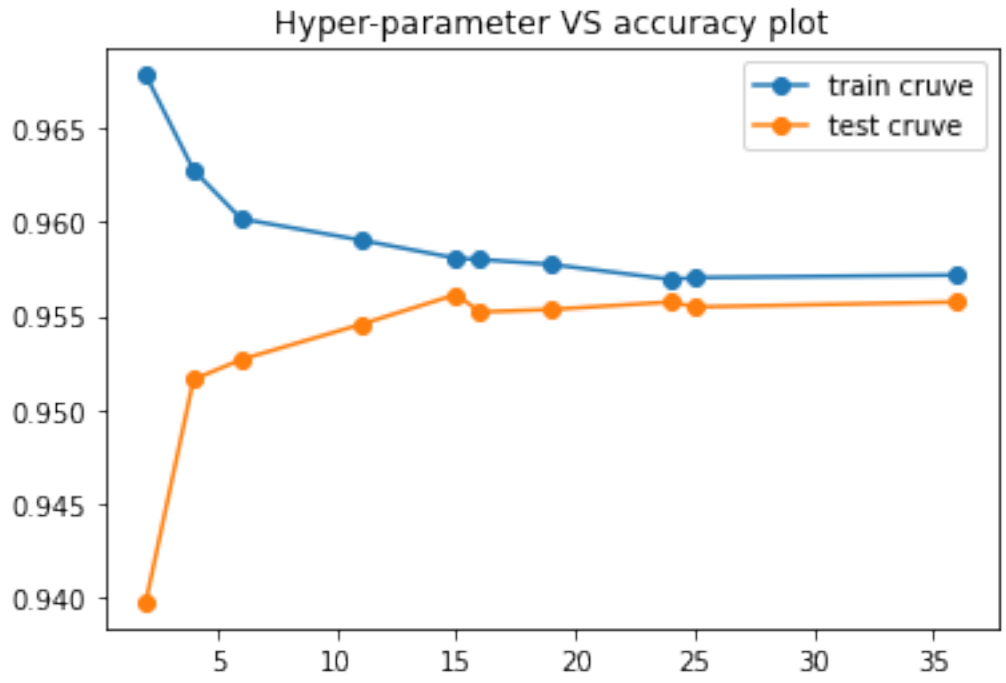

# 1 Implementing Custom RandomSearchCV

[18]:
```
from sklearn.metrics import accuracy_score
import math

def test_train_split_kfolds(fold,folds,x_train,y_train):
    train_size = len(y_train)
    part_size = math.ceil(train_size//folds)
    if fold == 0:
        return x_train[:part_size], y_train[:part_size], x_train[part_size:],␣
 ↪y_train[part_size:]
    elif fold == folds -1:
        return x_train[fold*part_size:], y_train[fold*part_size:], x_train[:
 ↪fold*part_size], y_train[:fold*part_size]
    else:
        test_data = x_train[fold*part_size:(fold+1)*part_size]
        test_label = y_train[fold*part_size:(fold+1)*part_size]
        train_data = np.append(x_train[:fold*part_size],␣
 ↪x_train[(fold+1)*part_size:],axis=0)

        train_label =np.append(y_train[:fold*part_size],␣
 ↪y_train[(fold+1)*part_size:],axis=0)
        return test_data, test_label, train_data, train_label
```

```
def RandomSearchCV(x_train,y_train,classifier, param_range, folds):
    k_list = random.sample(range(param_range[0], param_range[1]) , min(10 ,␣
 ↪param_range[1] - param_range[0]))
    k_list = sorted(k_list)
    trainaccscore = []
    crossaccscore = []
    for k in k_list:
        trainscores_folds = []
        testscores_folds = []
        for fold in range(folds):

            X_test,Y_test,X_train,Y_train =␣
 ↪test_train_split_kfolds(fold,folds,x_train,y_train)
            #print(X_test.shape,Y_test.shape,X_train.shape,Y_train.shape)
            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))
        trainaccscore.append(np.mean(np.array(trainscores_folds)))
        crossaccscore.append(np.mean(np.array(testscores_folds)))
    return crossaccscore, trainaccscore, k_list
```

[25]:
```
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()

params_range = (1,50)
folds = 5

testscores, trainscores, k_list = RandomSearchCV(X_train, y_train, neigh,␣
 ↪params_range, folds)

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

```
plt.show()
```

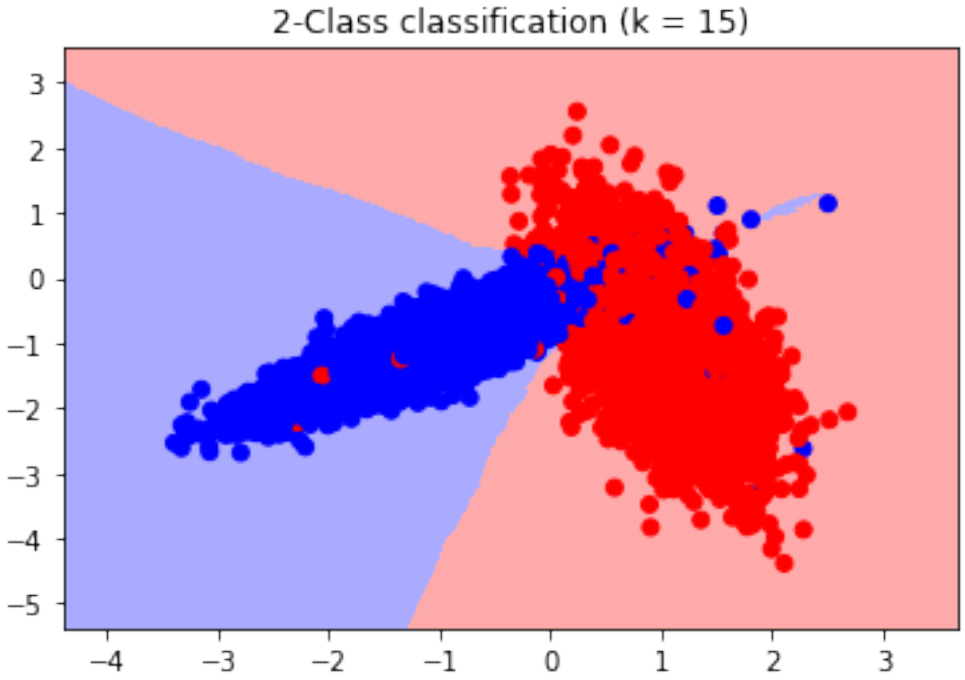



By ruing the above code 4-5 times it was clear that k= 15 seems to be working the best

[27]:
```
# Plot decision Boundary

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()
```

[28]:
```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 15)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```




[ ]: