In [1]:

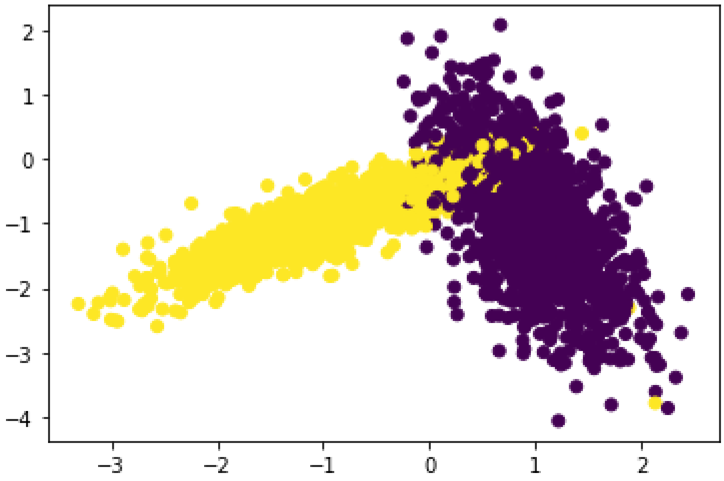
```
from tqdm import tqdm
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import euclidean_distances


x,y = make_classification(n_samples=10000, n_features=2, n_informative=2, n_redu
ndant= 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
```

In [2]:

```
%matplotlib inline
import matplotlib.pyplot as plt
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 our model


    #1.generate 10 unique values(uniform random distribution) in the given
range "param_range" and store them as "params"
    # ex: if param_range = (1, 50), we need to generate 10 random numbers
in range 1 to 50
    #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 1
00 into 3 groups
      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-validation as follows

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

        # second we will keep group 1+group 3 i.e. 0-33, 67-100 as train d
ata and group 2: 34-66 as test data, and find
          train and test accuracies

        # third we will keep group 2+group 3 i.e. 34-100 as train data and
group 1: 0-33 as test data, and find train and
          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_scores"
        # find the mean of test accuracies of above 3 steps and store in a
list "test_scores"
    #4. return both "train_scores" and "test_scores"

#5. call function RandomSearchCV(x_train,y_train,classifier, param_range,
folds) and store the returned values into "train_score", and "cv_scores"
#6. plot hyper-parameter vs accuracy plot as shown in reference notebook a
nd choose the best hyperparameter
#7. plot the decision boundaries for the model initialized with the best h
yperparameter, as shown in the last cell of reference notebook
```

In [3]:

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

In [4]:

```
def Random_search(x_train,y_train,classifier, param_range, folds):

    trainscores = []
    testscores  = []

    #generating 10 unique values in the given range "param_range" and store them
as "params"
    params = sorted(random.sample(range(param_range[0], param_range[1]), 10))

    x = len(x_train)
    #generate indices ranging from  0 to len(X_train)
    indices = list(range(x))
    #generating random shuffle of indices
    random.shuffle(indices)
    # divide len(X_train) by folds to find no. of elements in one groups
    n = x // folds

    #appending 'n' no.of indices in each groups
    group_indices = []
    for i in range(0,x, n):
        if x - (i + n) >= n:
            group_indices.append(indices[i : i+n])
        else:
            group_indices.append(indices[i:])
            break

    # for each unique value in params
    for k in tqdm(params):
        trainscores_folds = []
        testscores_folds  = []
        #for each gropus in group_indices
        for group in group_indices:
            train_indices = list(set(indices) - set(group))
            test_indices  = group

            # selecting the data points based on the train_indices and test_indi
ces
            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_train)
            #train accuracy
            trainscores_folds.append(accuracy_score(Y_train, Y_predicted))

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

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

    return params,trainscores,testscores
```

In [5]:

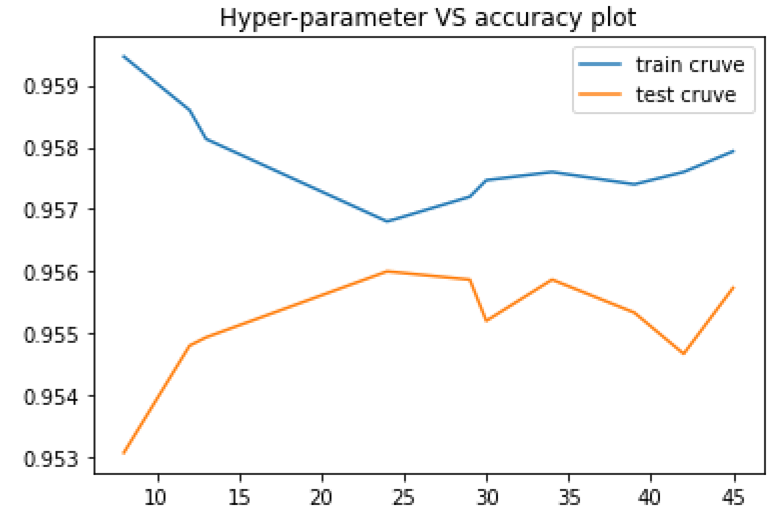
```
neigh = KNeighborsClassifier()
param_range =(1,50)
folds = 3

params,trainscores,testscores = Random_search(X_train, y_train, neigh, param_ran
ge,folds)
print(params)

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 [01:39<00:00,  9.91s/it]

[8, 12, 13, 24, 29, 30, 34, 39, 42, 45]
```



In [6]:

```
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()
```

In [8]:

```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 24)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```