In [24]:

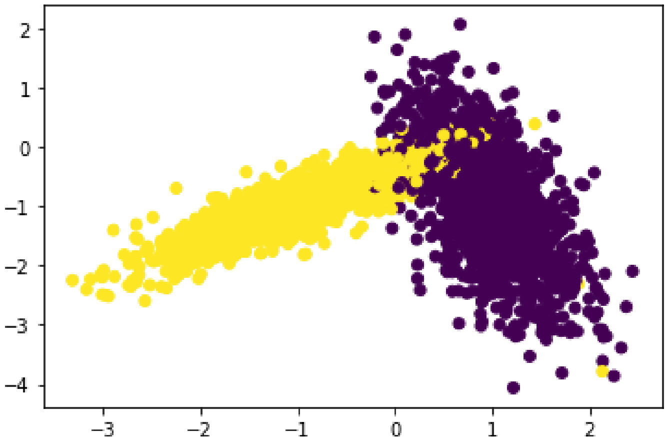
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
import math


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)
print(type(X_train))
# del X_train,X_test
```

```
<class 'numpy.ndarray'>
```

In [2]:

```
%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 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 100 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-v
alidation as follows

        # first we will keep group 1+group 2 i.e. 0-66 as train data and group 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 data 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 "t
est_scores"
    #4. return both "train_scores" and "test_scores"

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

In [100]:

```
#Building RandomSearch
def RandomSearchCV(x_train,y_train,classifier, param_range, folds):
    #10 Random hyperparameter value selection
    param = sorted(random.sample(range(param_range[0],param_range[1]),10))
    GroupPercentage = math.floor(len(x_train)/folds)

    groupDataIndices = []
    groupValues_x = []
    groupValues_y = []

    trainscores = []
    testscores  = []

    i = 0
    start = 0
    end = GroupPercentage

    #Createing indices for values to be stored in groups based on fold selection
    while i != folds:

        groupDataIndices.append(list(range(start,end)))
        i += 1
        start += GroupPercentage
        end += GroupPercentage

    #Storing the values in forms of groups based on fold selection
    for i in range(folds):
        groupValues_x.append(x_train[groupDataIndices[i]])
        groupValues_y.append(y_train[groupDataIndices[i]])


    #Performing K-Fold cross validations
```

```
    for k in tqdm(param):
        trainaccuracy_fold = []
        testaccuracy_fold = []

        for f in range(folds):
            x_train_cv = []
            y_train_cv = []
            x_test_cv = groupValues_x[f]
            y_test_cv = groupValues_y[f]

            for j in range(folds):
                if j != f:
                    x_train_cv = x_train_cv + list(groupValues_x[j])
                    y_train_cv = y_train_cv + list(groupValues_y[j])

            #Now train the model
            classifier.n_neighbors = k
            classifier.fit(x_train_cv,y_train_cv)

            y_predicted = classifier.predict(x_test_cv)
            testaccuracy_fold.append(accuracy_score(y_test_cv, y_predicted))

            y_predicted = classifier.predict(x_train_cv)
            trainaccuracy_fold.append(accuracy_score(y_train_cv, y_predicted))

        trainscores.append(np.mean(np.array(trainaccuracy_fold)))
        testscores.append(np.mean(np.array(testaccuracy_fold)))

    return param, trainscores, testscores
```

In [105]:

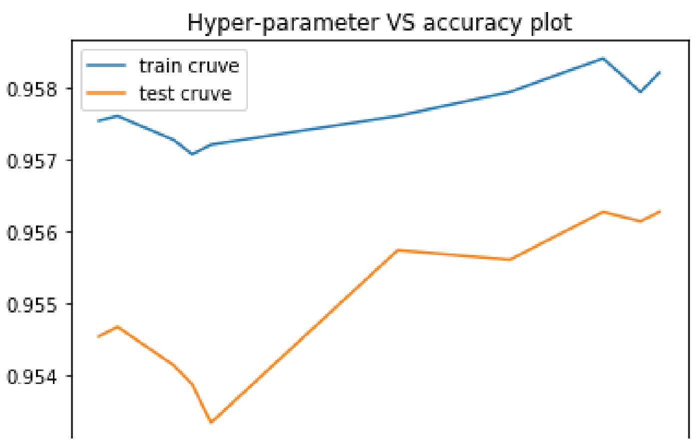
```
#Appying RandomSearch and plotting the Hyperparameter vs Accuracy plot
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings
warnings.filterwarnings("ignore")


neigh = KNeighborsClassifier()

hyperparameters,trainscores,testscores = RandomSearchCV(X_train,y_train,neigh, param_ran
ge=(1,50), folds=3)

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

```
100%|██████████████████████████████████████████████████|
10/10 [00:17<00:00,  1.75s/it]
```



In [103]:

```
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 [106]:

```
#Model trainig with best value of hypermeter choosen by Randomsearch
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)
```