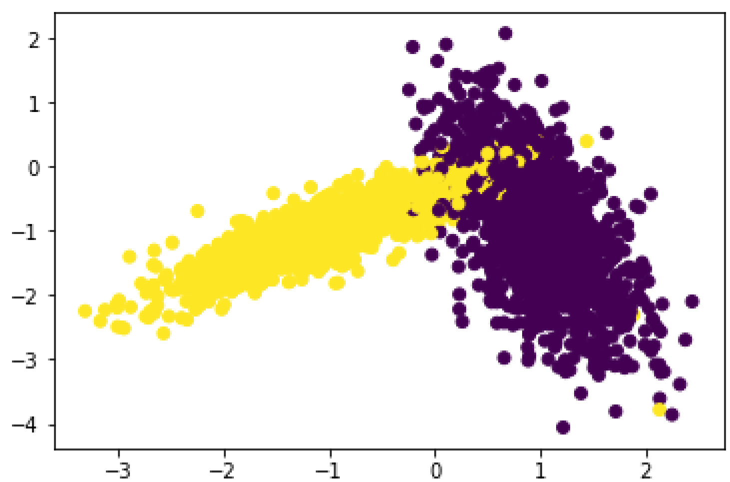
In [96]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances


x,y = make_classification(n_samples=10000, n_features=2, n_informative=2, n_redundant= 0, n_clusters_per_class=1,
X_train, X_test, y_train, y_test = train_test_split(x,y,stratify=y,random_state=42)

# del X_train,X_test
```

In [97]:
```
%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-validation 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, a
nd 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, a
nd 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 and choose the best hyperparame
```

ter

#7. plot the decision boundaries for the model initialized with the best hyperparameter, as shown in the last cell of reference notebook

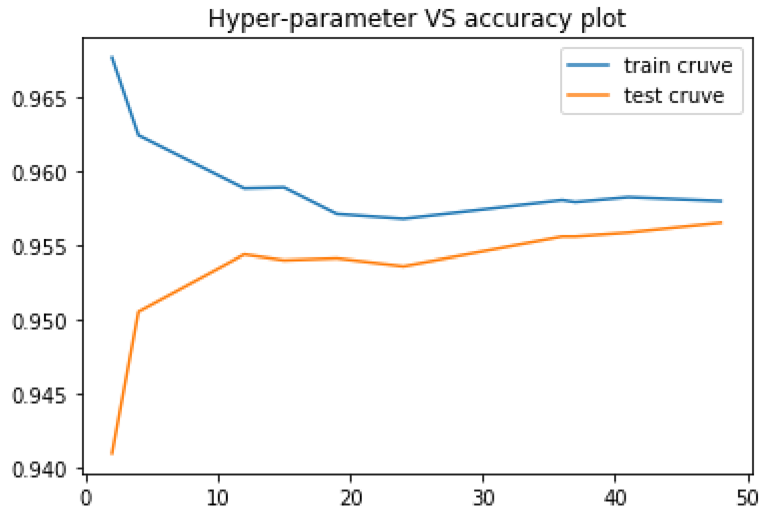
In [98]:

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

    while len(params)!=10:
        params.add(int(random.uniform(param_range[0],param_range[1])))
    params=list(params)  #Generating 10 random values of K from the range configurable range(1,50)
    params.sort() #Sorting the values of k in asc order for grapgh to be plotted neatly


    #https://stackoverflow.com/questions/58756377/how-does-the-numpy-function-array-split-work-mathematically

    #Calculating the size of sections by dividing the data set(x_train) by value of folds
    quotient, remainder = divmod(len(x_train), folds)
    section_sizes = (remainder * [quotient+1] +(folds- remainder) * [quotient])


    for k in params:
        trainscores_folds = []
        cvscores_folds  = []
        u=0
        t=0
        for i in range(0,folds):
            t+=section_sizes[i]
            cv_indices=list(x for x in range(u,t)) #calculating the CV indices
            train_indices  = list(set(list(range(0, len(x_train)))) - set(cv_indices)) #calculating the train inc
            u+=section_sizes[i]

            temp_tr_x=x_train[train_indices]
            temp_tr_y=y_train[train_indices]
            temp_cv_x=x_train[cv_indices]
            temp_cv_y=y_train[cv_indices]

            classifier.n_neighbors = k
            classifier.fit(temp_tr_x,temp_tr_y)


            Y_predicted = classifier.predict(temp_cv_x) #CV prediction
            cvscores_folds.append(accuracy_score(temp_cv_y, Y_predicted))
```

```
            Y_predicted = classifier.predict(temp_tr_x) #Train prediction
            trainscores_folds.append(accuracy_score(temp_tr_y, Y_predicted))

        trainscores.append(np.mean(np.array(trainscores_folds)))
        cvscores.append(np.mean(np.array(cvscores_folds)))
    return trainscores,cvscores,params
```

In [111]:
```
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings
warnings.filterwarnings("ignore")
import numpy as np


classifier = KNeighborsClassifier()
folds=int(input("Enter the number of folds you want: "))

param_range=(1,50)



trainscores,testscores,params=RandomSearchCV(X_train,y_train,classifier, param_range, folds)

#print(trainscores)
#print(testscores)
#print(len(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()
```

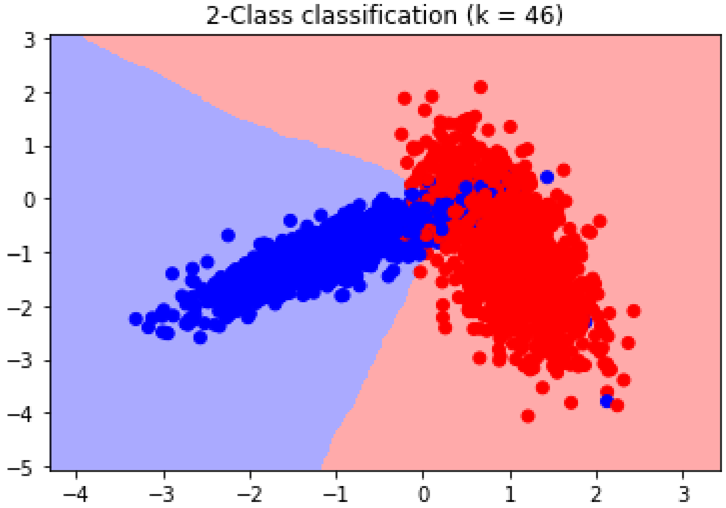
```
Enter the number of folds you want: 3
```



From the above graph we observe that the test accuracy and train accuracy are closest at K=46. Thus taking the value of K=46 for X_test

In [101]:
```
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 [110]:
```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 46)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_test[:, 0], X_test[:, 1], y_test, neigh)
```



In [ ]: