In [1]:

```
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
```

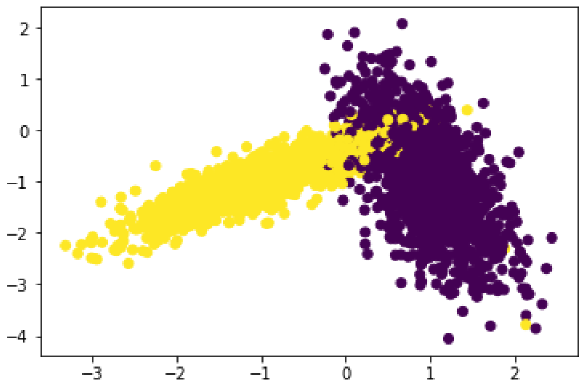
In [12]:

```
print(X_train[1])
```

```
[ 0.61696406 -0.00418956]
```

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:
```

```
    #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, 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: 3
4-66 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_scor
es"
        # 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 bes
t hyperparameter
#7. plot the decision boundaries for the model initialized with the best hyperparameter, as
shown in the last cell of reference notebook
```

In [3]:

```python
#Random numbers list
import random

def generateRandomNumbers(a,b):
    randomList = random.sample(range(a,b),10)
    return randomList

print(generateRandomNumbers(10,100))
```

```
[42, 12, 68, 26, 57, 84, 62, 64, 37, 87]
```

In [16]:

```python
d= [3, 7, 8, 11, 14, 16, 17, 18, 20, 21]
d1=d[2:6]
print(d1)
#s=(d)-(d1)
print(list(filter(lambda x: x not in d1, d)))
```

```
[8, 11, 14, 16]
[3, 7, 17, 18, 20, 21]
```

In [51]:

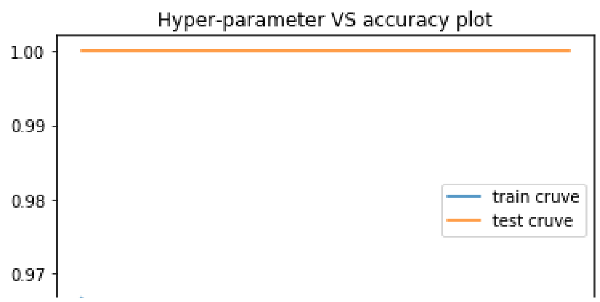
```python
from sklearn.metrics import accuracy_score


def select_indices_percentage(x_train,i,D):
    return random.sample(range(i, len(x_train)), int(D*len(x_train)))


def RandomSearchCV(x_train,y_train,classifier, param_range, folds):
    trainscores = []
    testscores  = []
    #Rand10Num = generateRandomNumbers(param_range[0],param_range[1])
```

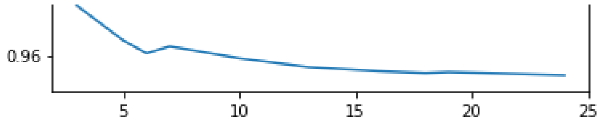
```
    for k in param_range: # tqdm(params['n_neighbors']):
        trainscores_folds = []
        testscores_folds  = []
        for j in range(0, folds):
            test_indices=[]
            n=len(x_train)-1
            #print(" Length:{} D:{} i:{}".format(n,D,i))
            test_indices  =[(j*n)/folds, ((j+1)*n)/folds]
            train_indices = list(set(list(range(0, n)))-set(test_indices))
            #select_indices_percentage(x_train,i,D)# x_train[i:D].tolist()
            test_indices=[int(x) for x in test_indices]

            # selecting the data points based on the train_indices and test_indices
            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_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))
        trainscores.append(np.mean(np.array(trainscores_folds)))
        testscores.append(np.mean(np.array(testscores_folds)))
    return trainscores,testscores
```

In [52]:

```
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 = {'n_neighbors':[3,5,7,9,11,13,15,17,19,21,23]}
param_range = (2,25)
folds = 4
Rand10Num=[]
Rand10Num = generateRandomNumbers(param_range[0],param_range[1])
Rand10Num.sort()
params = {'n_neighbors':Rand10Num}
print(params['n_neighbors'])
trainscores,testscores = RandomSearchCV(X_train, y_train, neigh, Rand10Num, folds)

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

```
[3, 5, 6, 7, 10, 13, 16, 18, 19, 24]
[0.9666611100738567, 0.9619936654841967, 0.9602600432080373, 0.9611935321259694,
0.9595932654095143, 0.9583930653721733, 0.9578596431333549, 0.9575929320139458,
0.9577262875736504, 0.9573262208945367] [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```





In [22]:

```
# 2 is best neighbors
```

In [53]:

```
# understanding this code line by line is not that importent
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 [54]:

```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 2)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```