In [59]:

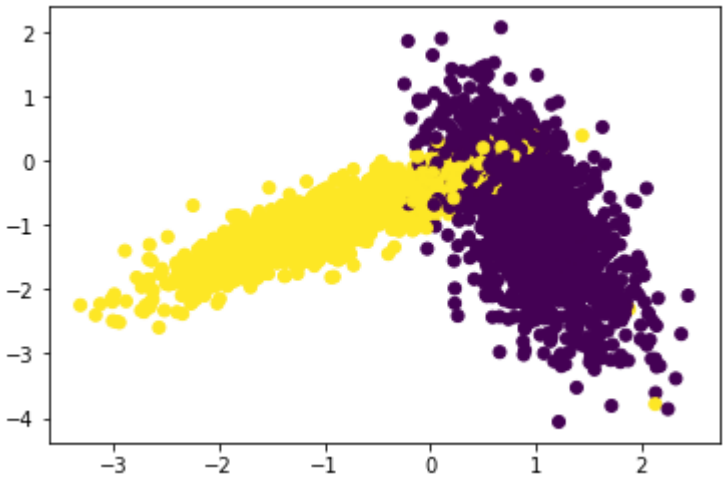
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances


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 [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 a
nd 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 ran
ge 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 int
o 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, and find train and
          test accuracies

        # second we will keep group 1+group 3 i.e. 0-33, 67-100 as train data an
d 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 and cho
ose the best hyperparameter
#7. plot the decision boundaries for the model initialized with the best hyperpa
rameter, as shown in the last cell of reference notebook
```

In [68]:

```
# it will take classifier and set of values for hyper prameter in dict type dict({hyper
parmeter: [list of values]})
# we are implementing this only for KNN, the hyper parameter should n_neighbors
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold


def randomcall(x_train,i,k):
    """
    This function returns the test indexes
    """
    n = len(x_train)
    li = []
    p = int(i*(n/k))
    q = int((n/k)*(i+1))
    li = [i for i in range(p,q)]
    return li



def RandomSearch(x_train,y_train,classifier, params, folds):
    trainscores = []
    testscores  = []
    for k in tqdm(params['n_neighbors']):  #Iterating the list of neighbours
        trainscores_folds = []
        testscores_folds  = []
        for j in range(0, folds):    #Iterating k-folds
            test_indices = randomcall(x_train,j,folds)  #Calling indexes of test datase
t
            train_indices  = list(set(list(range(0, len(x_train)))) - set(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)  #Predicting test
            testscores_folds.append(accuracy_score(Y_test, Y_predicted)) #All accuracy
 score of test  appending in list

            Y_predicted = classifier.predict(X_train) #Predicting train
            trainscores_folds.append(accuracy_score(Y_train, Y_predicted)) #All accurac
y scores of train appending in list
        trainscores.append(np.mean(np.array(trainscores_folds))) #Apply mean of trainsc
ore folds
        testscores.append(np.mean(np.array(testscores_folds))) #Apply mean of test scor
e filds
    return trainscores,testscores
```

In [70]:

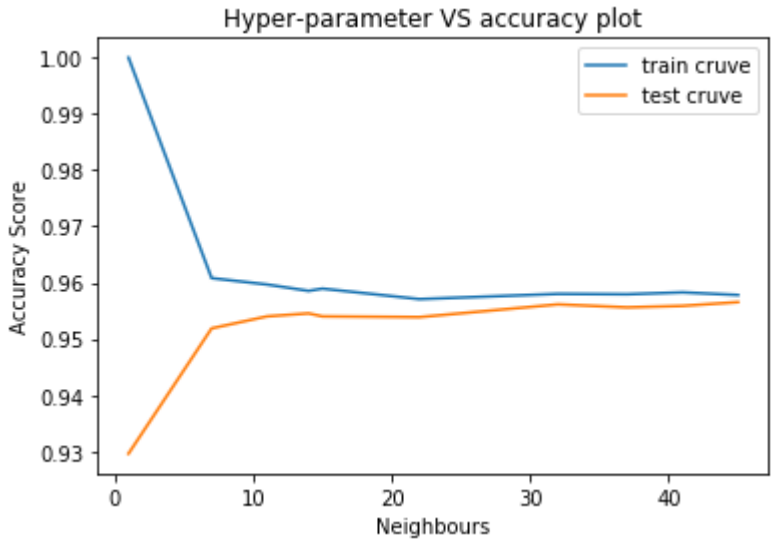
```
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings
warnings.filterwarnings("ignore")


neigh = KNeighborsClassifier()  #Using K Neighbors Classifier
lis = random.sample(range(1,50),10)
lis.sort()
params = {'n_neighbors': lis } #Randomly choosen unique numbers
folds = 3

trainscores,testscores = RandomSearch(X_train, y_train, neigh, params, folds)  #Calcula
te best train scores,test scores

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.xlabel('Neighbours')
plt.ylabel('Accuracy Score')
plt.show()
```

```
  0%|
| 0/10 [00:00<?, ?it/s]
 10%|
| 1/10 [00:00<00:08,  1.06it/s]
 20%|
| 2/10 [00:02<00:08,  1.01s/it]
 30%|
| 3/10 [00:03<00:07,  1.13s/it]
 40%|
| 4/10 [00:04<00:06,  1.14s/it]
 50%|
| 5/10 [00:05<00:05,  1.16s/it]
 60%|
| 6/10 [00:07<00:04,  1.21s/it]
 70%|
| 7/10 [00:08<00:04,  1.35s/it]
 80%|
| 8/10 [00:10<00:02,  1.37s/it]
 90%|
          | 9/10 [00:11<00:01,  1.41s/it]
100%|
           | 10/10 [00:13<00:00,  1.46s/it]
```



Observation : 1.Random Search CV chooses best Hyper parameter for any classification techniques 2.It select random combination of hyper parameters to train the model and score. 3.Compare to Grid Search CV, execution time is very fast. 4.Grid Search CV function is extremely costly exection both in computing time and power. 5.Random Search CV is very efficent for larger data. For Grid Search CV is very accurate for smaller data. 6.Based on train and test scores it chooses best hyper parameter.

In [54]:

```
# 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 [67]:

```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 32)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```