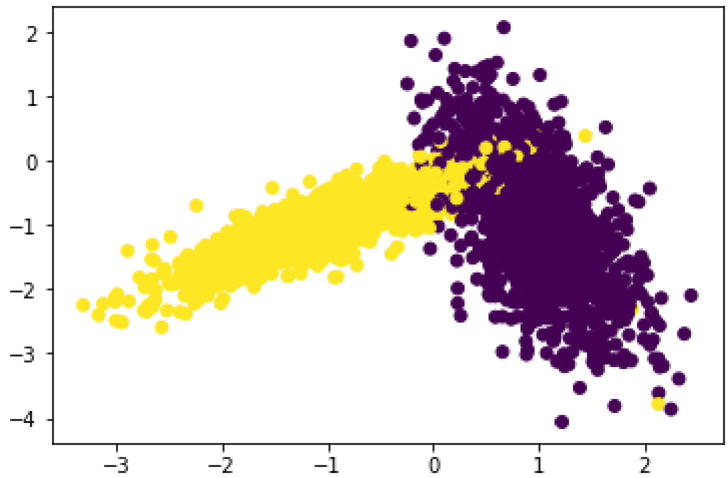
In [0]:

```
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, r
X_train, X_test, y_train, y_test = train_test_split(x,y,stratify=y,random_state=42)

# del X_train,X_test
```

In [0]:

```
%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()
```



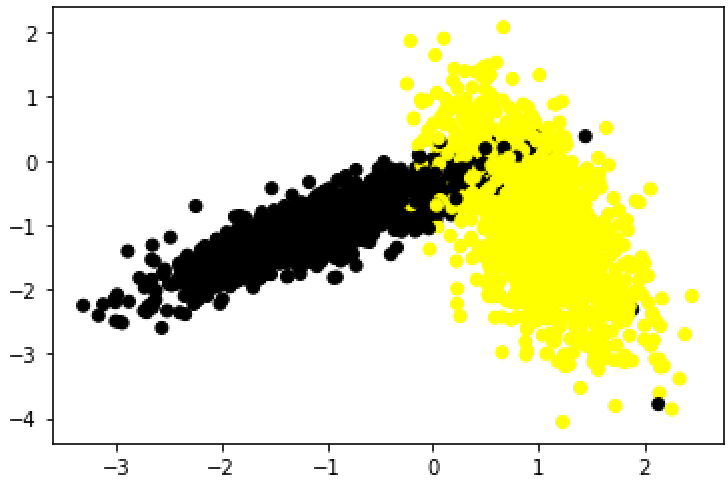
Implementing Custom RandomSearchCV

```
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) a
nd store the returned values into "train_score", and "cv_scores"
#6. plot hyper-parameter vs accuracy plot as shown in reference notebook and choos
e the best hyperparameter
#7. plot the decision boundaries for the model initialized with the best hyperpara
meter, as shown in the last cell of reference notebook
```

In [8]:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import euclidean_distances
import random
import scipy as sp
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline
x,y=make_classification(n_samples=10000,n_features=2,n_informative=2,n_redundant=0,n_cluste
X_train,X_test,y_train,y_test=train_test_split(x,y,stratify=y,random_state=42)
#print(y_test.shape)
colors=['yellow','black']
plt.scatter(X_test[:,0],X_test[:,1],c=y_test,cmap=ListedColormap(colors))
plt.show()

def groups(li,n):
    return [li[i::n] for i in range(n)]
```

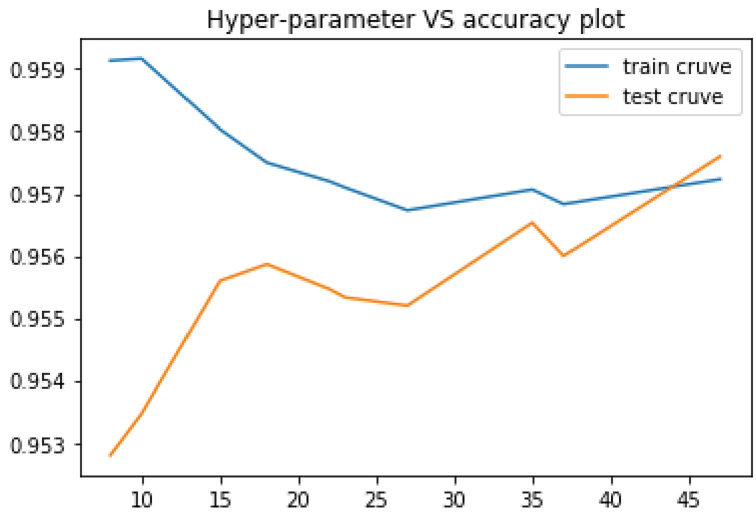


In [9]:

```
def randomsearchcv(x_train,y_train,classifier,params,folds):
    trainscores = []
    testscores  = []
    lol=[]



    full_index=groups(list(range(len(x_train))),folds)
    flattened  = [val for sublist in full_index for val in sublist]
    for i in full_index:
        lol.append(list(set(flattened)-set(i)))

    for k in params:
        trainscores_folds = []
        testscores_folds = []
        for i in lol:
            X_train = x_train[i]
            Y_train = y_train[i]
            X_test = x_train[list(set(flattened)-set(i))]
            Y_test = y_train[list(set(flattened)-set(i))]
            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)))
    #print(trainscores,testscores)
    return trainscores,testscores
```

In [19]:

```
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=50
params=random.sample(range(1,50),10)
params=sorted(params)
folds = 5

trainscores,testscores = randomsearchcv(X_train, y_train, neigh, params,folds)


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()
```

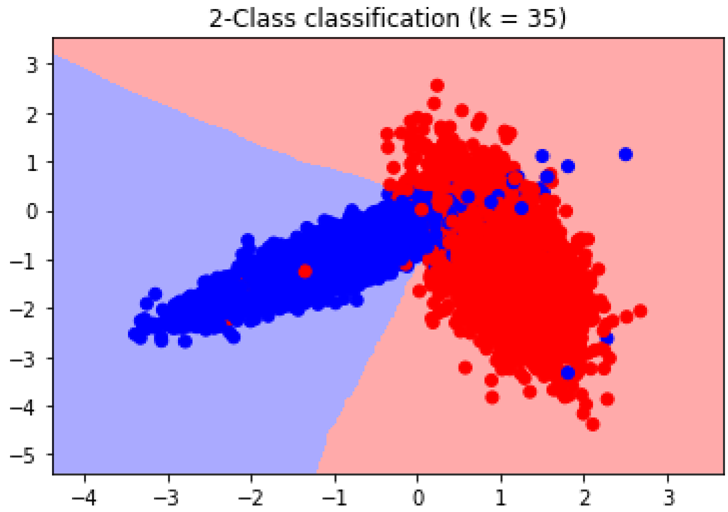


In [11]:

```
# 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 [22]:

```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors =35)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```



In [ ]: