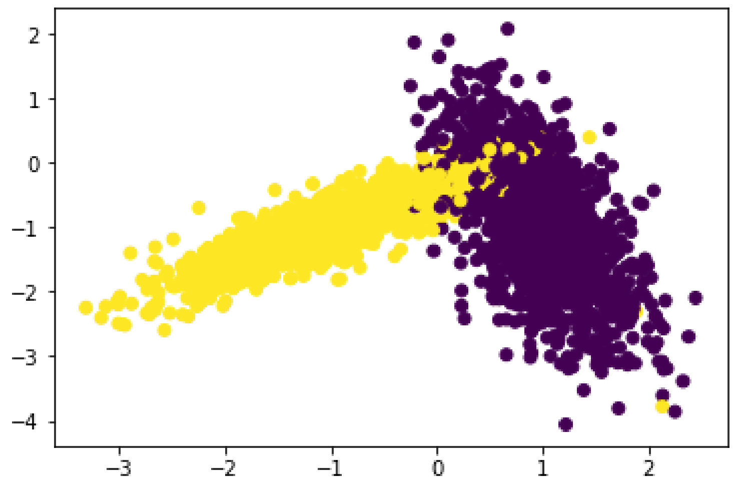
In [55]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore")



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,rand
om_state=42)

# del X_train,X_test
```

In [56]:
```
colors = {0:'yellow', 1:'black'}
plt.scatter(X_test[:,0], X_test[:,1],c=y_test)
plt.show()
```

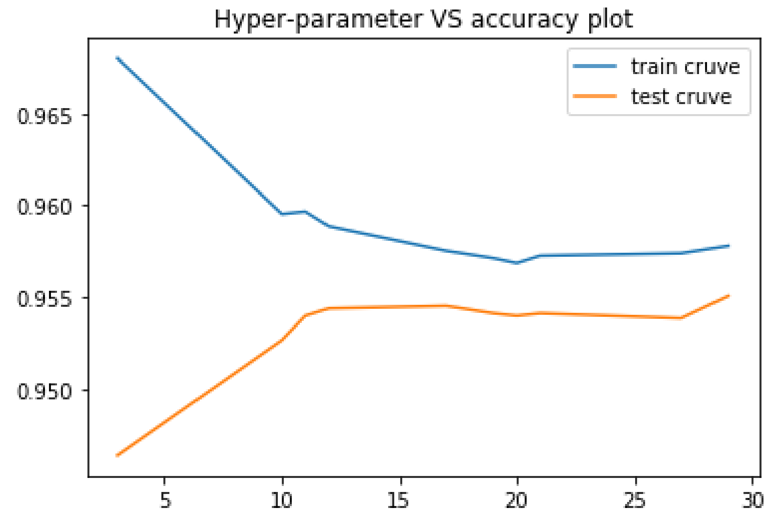


In [59]:
```
import random
neigh = KNeighborsClassifier()
l=list(range(2,30))
nl=list(random.sample(l,k=10))
nl.sort()
params=nl
#print(params)

params = {'n_neighbors':params}
folds =int(input())


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


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

```
  0%|
| 0/10 [00:00<?, ?it/s]

 10%|████████
| 1/10 [00:01<00:11,  1.30s/it]

 20%|████████████████▌
| 2/10 [00:02<00:10,  1.32s/it]

 30%|█████████████████████████
| 3/10 [00:04<00:09,  1.37s/it]

 40%|█████████████████████████████████
| 4/10 [00:05<00:08,  1.46s/it]

 50%|█████████████████████████████████████████▌
| 5/10 [00:07<00:07,  1.46s/it]

 60%|██████████████████████████████████████████████████
| 6/10 [00:08<00:05,  1.48s/it]

 70%|██████████████████████████████████████████████████████████
| 7/10 [00:10<00:04,  1.48s/it]

 80%|██████████████████████████████████████████████████████████████████
▍                | 8/10 [00:11<00:02,  1.47s/it]

 90%|██████████████████████████████████████████████████████████████████
█████████        | 9/10 [00:13<00:01,  1.46s/it]
```

```
100%|██████████| 10/10 [00:14<00:00,  1.46s/it]
```



Implementing Custom RandomSearchCV

```
def RandomSearchCV(x_train,y_train,classifier, param_range, fold
s):
    # 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 d
evide 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 rando
m numbers in range 1 to 50
    #2.devide numbers ranging from  0 to len(X_train) into group
s= folds
    # ex: folds=3, and len(x_train)=100, we can devide numbers f
rom 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 y
ou 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: 34-66 as test data, and find
          train and test accuracies

        # third we will keep group 2+group 3 i.e. 34-100 as trai
n 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 procedu
re

        # 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"
```

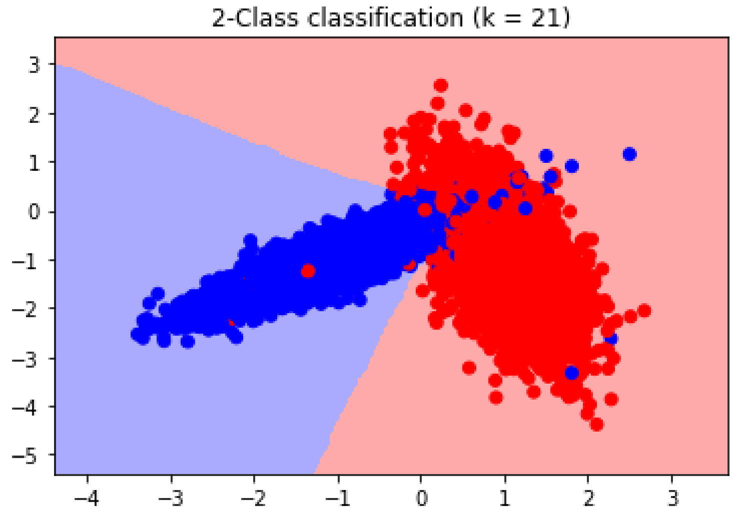
```
#5. call function RandomSearchCV(x_train,y_train,classifier, par
am_range, folds) and store the returned values into "train_scor
e", and "cv_scores"
#6. plot hyper-parameter vs accuracy plot as shown in reference
notebook and choose the best hyperparameter
#7. plot the decision boundaries for the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

In [43]:

```
def RandomSearchCV(x_train,y_train,classifier, params, folds):
    trainscores = []
    testscores  = []
    for k in tqdm(params['n_neighbors']):
        trainscores_folds = []
        testscores_folds  = []
        kf=KFold(n_splits=folds)
        for train_index,test_index in kf.split(X=x_train):
            X_train = x_train[train_index]
            Y_train = y_train[train_index]
            X_test  = x_train[test_index]
            Y_test  = y_train[test_index]

            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_predicte
d))
        trainscores.append(np.mean(np.array(trainscores_folds)))
        testscores.append(np.mean(np.array(testscores_folds)))
    return trainscores,testscores
```

In [61]:
```
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 [63]:
```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 21)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```



In [27]:
```
#import random
#l=list(range(2,40))
#nl=random.sample(l,k=10)
#nl.sort()
```