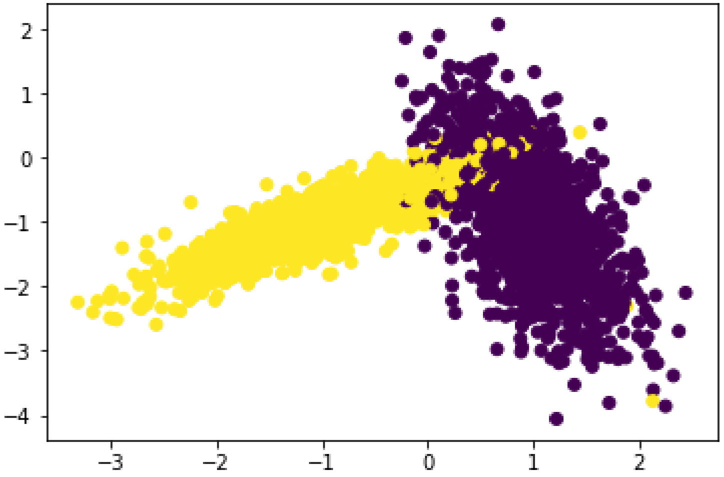
In [26]:

```
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)
```

In [27]:

```
%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()
```



In [28]:

```
import random
```

In [29]:

```
def random_dist(param_range):
    unique_values=random.sample(range(1, param_range), 10)
    unique_values.sort()
    return unique_values
```

In [31]:

```
def partition(xtrain,ytrain,folds):
    division = len(xtrain) / float(folds)
    xtrain=xtrain.tolist()
    ytrain=ytrain.tolist()
    group=([ xtrain[int(round(division * i)): int(round(division * (i + 1)))] for i in range(folds) ])
    label=([ ytrain[int(round(division * i)): int(round(division * (i + 1)))] for i in range(folds) ])
    return group,label
```

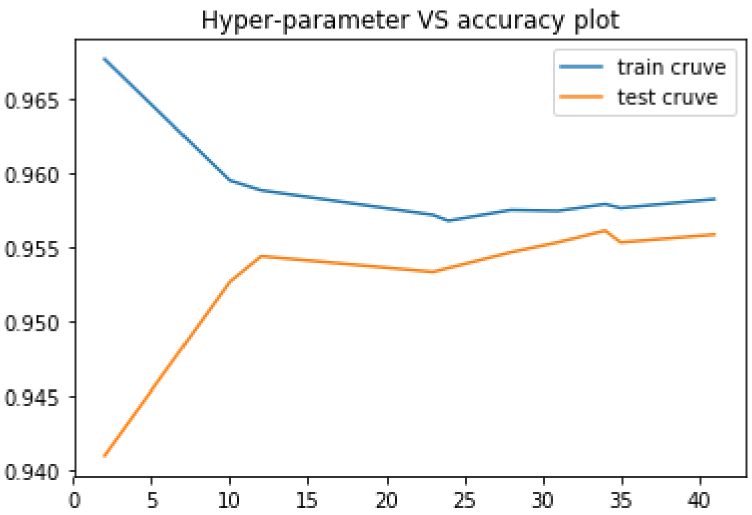
In [32]:

```
def RandomSerachCV(x_train, y_train, classifier, param_range, folds):
    train_scores = []
    test_scores  = []

    param_list=random_dist(param_range)
    print(param_list)
    params={'n_neighbors': param_list}



    group,label=partition(x_train,y_train,folds)

    for k in tqdm(params['n_neighbors']):
        trainscores_folds = []
        testscores_folds  = []

        for i in reversed(range(folds)):
            X_train = [group[iter] for iter in range(3) if iter != i]
            X_train = [item for sublist in X_train for item in sublist]
            Y_train = [label[iter] for iter in range(folds) if iter != i]
            Y_train = [j for sublist in Y_train for j in sublist]
            X_test  = group[i]
            Y_test  = label[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))

        train_scores.append(np.mean(np.array(trainscores_folds)))
        test_scores.append(np.mean(np.array(testscores_folds)))

    return train_scores,test_scores,params
```

In [41]:

```
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()

param_range=50
folds = 3

train_scores,cv_scores,params= RandomSerachCV(X_train,y_train,neigh,param_range,folds)


print(params)
print(train_scores)
print(cv_scores)


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

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

[2, 10, 12, 23, 24, 28, 31, 34, 35, 41]

100%|██████████| 10/10 [00:01<00:00,  5.78it/s]

{'n_neighbors': [2, 10, 12, 23, 24, 28, 31, 34, 35, 41]}
[0.9677333333333333, 0.9595333333333333, 0.9588666666666666, 0.9571999999999999, 0.9568, 0.9575333333333335, 0.957466
6666666668, 0.9579333333333334, 0.9576666666666668, 0.9582666666666667]
[0.9409333333333333, 0.9526666666666667, 0.9544, 0.9533333333333335, 0.9536000000000001, 0.9546666666666667, 0.955333
3333333334, 0.9561333333333333, 0.9553333333333334, 0.9558666666666668]
```

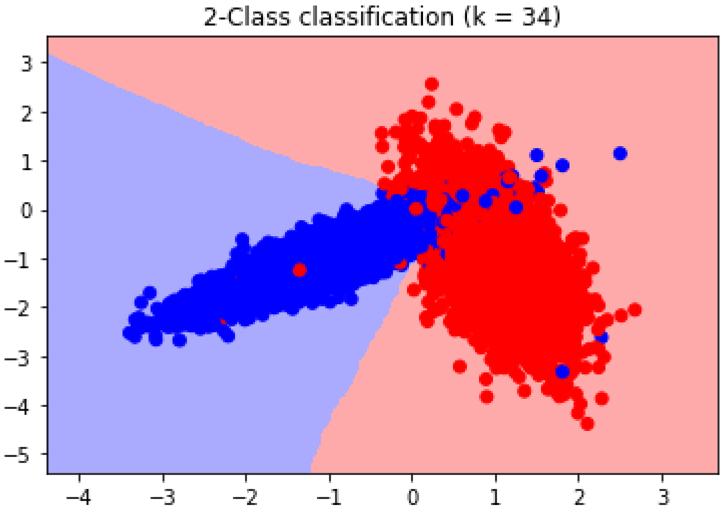


In [42]:

```
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 [43]:

```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 34)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```



In [44]:

```
!jupyter nbconvert --to html KNN_solve.ipynb
```

```
[NbConvertApp] Converting notebook KNN_solve.ipynb to html
[NbConvertApp] Writing 360538 bytes to KNN_solve.html
```