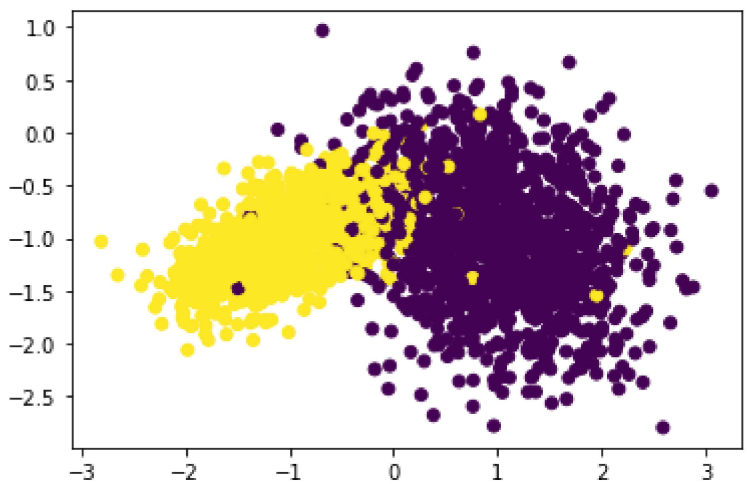
In [1]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances


x,y = make_classification(n_samples=11000, 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)
```

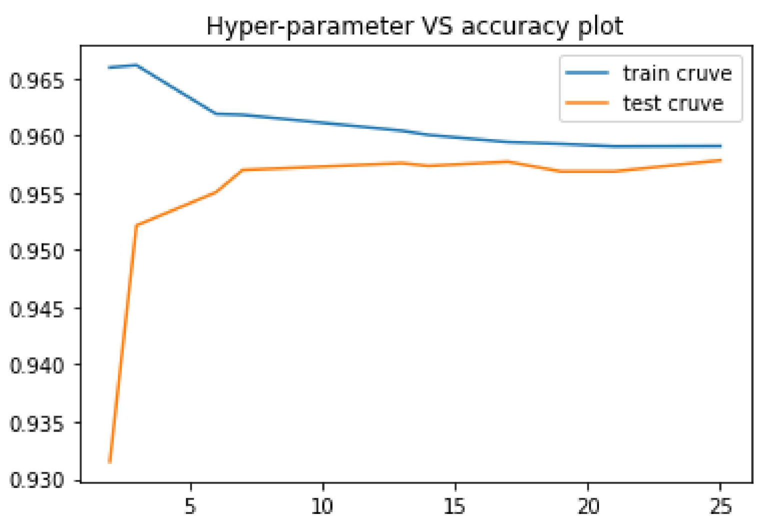
In [2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
colors = {0:'red', 1:'blue'}
plt.scatter(X_test[:,0], X_test[:,1],c=y_test)
plt.show()
```



In [16]:
```
from sklearn.metrics import accuracy_score

def RandomCVSearch(x_train,y_train,classifier, param, folds):
    trainscores = []
    testscores  = []

    param_list = []
    param_list = random.sample( range(param[0], param[1]), 10)

    param_list.sort()
    print(param_list)

    params = { 'n_neighbors' : param_list }
    for k in tqdm(params['n_neighbors']):
        trainscores_folds = []
        testscores_folds  = []
        for j in range(0, folds):
          values = len(x_train)/folds
          limit = int(values)
          test_indices = list( set( list( range( limit*j, limit*(j+1) ) ) ) )
          train_indices  = list(set(list(range(1, len(x_train)))) - set(test_i
ndices))

          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,param_list
```

In [17]:
```
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 = (1,30)
folds = 10

train_score,cv_score,params = RandomCVSearch(X_train, y_train, neigh, param_ra
nge, folds)

plt.plot(params,train_score, label='train cruve')
plt.plot(params,cv_score, label='test cruve')
plt.title('Hyper-parameter VS accuracy plot')
plt.legend()
plt.show()
```

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

[2, 3, 6, 7, 13, 14, 17, 19, 21, 25]

100%|██████████| 10/10 [00:22<00:00,  2.26s/it]
```



In [18]:

```
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 [19]:

```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 8)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```