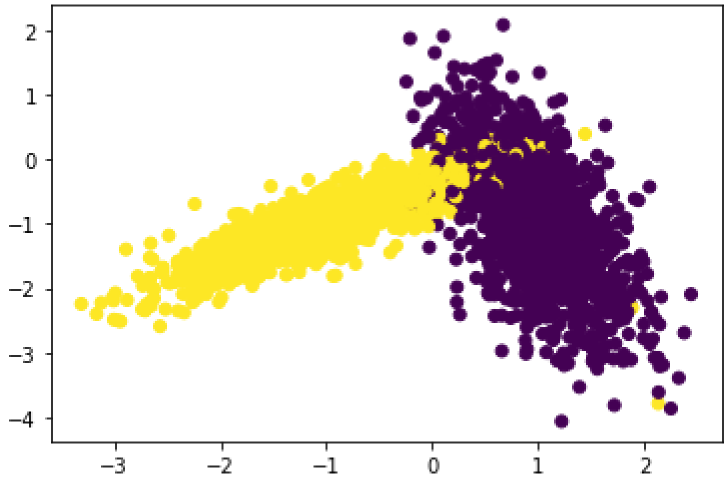
In [1]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
import random
from sklearn.metrics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


x,y = make_classification(n_samples=10000, n_features=2, n_informative=2, n_
X_train, X_test, y_train, y_test = train_test_split(x,y,stratify=y,random_st

# del X_test, y_test
### only X_train and y_train important
```

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

In [4]:
```
def create_fold(X_train, y_train, folds):
    total_x_train = []
    total_y_train = []
    size_of_each_fold = len(X_train)//folds

    for i in range(1,folds+1):
        train_1 = X_train[size_of_each_fold*(i-1): size_of_each_fold*(i)]

        test_1 = y_train[size_of_each_fold*(i-1): size_of_each_fold*(i)]
        total_x_train.append(train_1)
        total_y_train.append(test_1)
    return list(total_x_train), list(total_y_train)
```

In [6]:
```
## check the folds
check_x, check_y = X_train[:4], y_train[:4]
folds = 4
```

```
total_x_train , total_y_train = create_fold(check_x, check_y, folds)

for k in range(folds):
    train_x_cv = total_x_train[k]
    train_x = np.vstack(total_x_train[:k] + total_x_train[k+1:])
    print(train_x,"&",train_x_cv)
```

```
[[ 0.61696406 -0.00418956]
 [-0.60025705 -0.72979921]
 [-0.67612274 -0.48412042]] & [[ 0.45267141 -1.42381257]]
[[ 0.45267141 -1.42381257]
 [-0.60025705 -0.72979921]
 [-0.67612274 -0.48412042]] & [[ 0.61696406 -0.00418956]]
[[ 0.45267141 -1.42381257]
 [ 0.61696406 -0.00418956]
 [-0.67612274 -0.48412042]] & [[-0.60025705 -0.72979921]]
[[ 0.45267141 -1.42381257]
 [ 0.61696406 -0.00418956]
 [-0.60025705 -0.72979921]] & [[-0.67612274 -0.48412042]]
```

In [7]:
```
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 da


    ##### STEP-1 ---> Generate 10 unique value randomly in 'param_range'

    param_ls = sorted(random.sample(range(1, param_range), 10))
    param_ls # these are the hypyerparameter K in K-NN

    ##### STEP-2 ---> Divide the Training Data in "folds" number of time

    total_x_train , total_y_train = create_fold(X_train, y_train, folds)


    #### SET-3 Do k-fold CV
    train_score = []
    test_score = []
    for nn in tqdm(param_ls):




        for k in range(folds):
            train_fold = []
            test_fold = []
            train_x_cv = total_x_train[k]
            train_x=  np.vstack(total_x_train[:k] + total_x_train[k+1:])
            train_y_cv = total_y_train[k]
            train_y=  np.hstack(total_y_train[:k] + total_y_train[k+1:])

            ### Setting up the nearest neighbors
            classifier.n_neighbors = nn
            classifier.fit(train_x, train_y)

            ### Train accuracies
            y_pred_train = classifier.predict(train_x)
            train_fold.append(accuracy_score(train_y,y_pred_train))

            ### Test accuracies
```

```
            y_pred = classifier.predict(train_x_cv)
            test_fold.append(accuracy_score(train_y_cv, y_pred))


            ### calculating the mean of accuracies
        train_score.append(np.mean(np.array(train_fold)))
        test_score.append(np.mean(np.array(test_fold)))
    return (train_score, test_score, param_ls)
```

In [8]:

```
knn = KNeighborsClassifier()
folds = 3 # this is like k in k-fold CV
param_range = 50
trainscores, testscores, n_neighbors = RandomSearchCV(X_train, y_train,knn,
```

```
100%|██████████| 10/10 [00:03<00:00,  2.99it/s]
```

In [9]:

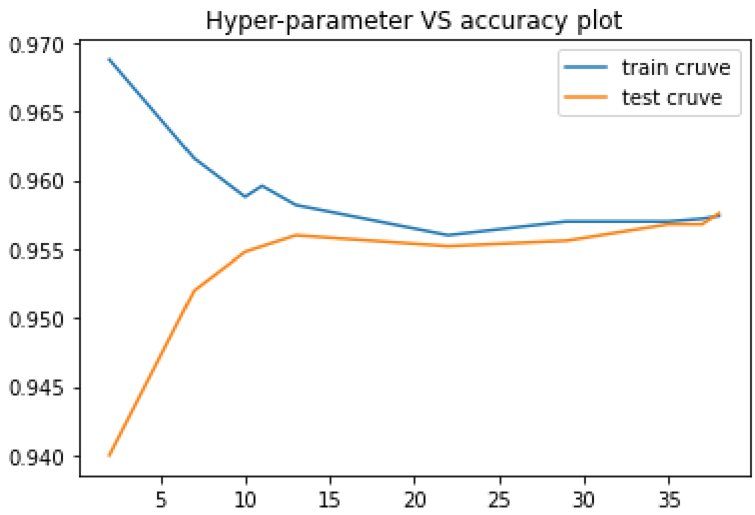
```
print(trainscores,"\n",testscores,"\n",n_neighbors)
```

```
[0.9688, 0.9616, 0.9588, 0.9596, 0.9582, 0.956, 0.957, 0.957, 0.9572, 0.9574]
 [0.94, 0.952, 0.9548, 0.9552, 0.956, 0.9552, 0.9556, 0.9568, 0.9568, 0.9576]
 [2, 7, 10, 11, 13, 22, 29, 35, 37, 38]
```

In [10]:

```
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

plt.plot(n_neighbors,trainscores, label='train cruve')
plt.plot(n_neighbors,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_r
```

```python
    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 [12]:
```python
from matplotlib.colors import ListedColormap
for i in n_neighbors[-2:]:
    neigh = KNeighborsClassifier(n_neighbors = i)
    neigh.fit(X_train, y_train)
    plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```