Importing the necessary libraries

In [39]:
```
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 random import seed
```

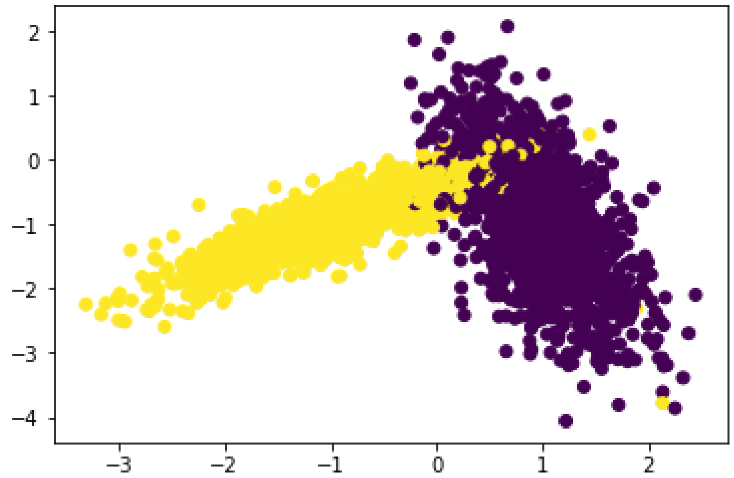
creating the artifical data

In [40]:
```
x,y = make_classification(n_samples=10000, n_features=2, n_informative=
2, n_redundant= 0, n_clusters_per_class=1, random_state=60)
```

Dividing the data into train and test

In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(x,y,stratify=y,rand
om_state=42)
```

In [42]:
```
%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()
```



Implementation of randomsearchCV function

In [45]:
```
def RandomSearchCV(x_train,y_train,classifier, param_range, folds):
    trainscores = []
    testscores  = []
    global params
    params=[]
    seed(7)
    params= random.sample(range(param_range[0],param_range[1]), 10) # c
reating random set of unique numbers
    params.sort() # sorting parameters
    print(params)
    length = int(len(x_train)/folds)
    foldsX = []
    foldsY = []
    for i in range(folds-1):
        foldsX += [x_train[i*length:(i+1)*length]]
        foldsY += [y_train[i*length:(i+1)*length]]        #block of sta
tement used for creating folds
    foldsX += [x_train[(folds-1)*length:len(x_train)]]
    foldsY += [y_train[(folds-1)*length:len(y_train)]]
    for k in tqdm(params):
```

```
        trainscores_folds = []
        testscores_folds  = []
        for j in range(0, folds):
            X_test  = foldsX[j]
            Y_test  = foldsY[j]
            G=np.delete(foldsX,j,0)      #removing cross validation dat
a
            V=np.delete(foldsY,j,0)
            X_train = np.concatenate(G)
            Y_train =  np.concatenate(V) # data used for training

            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
)) # calculating accuracy score for each fold

            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))) # calcul
ating the accuracy score for each k
    return trainscores,testscores
```

In [46]:

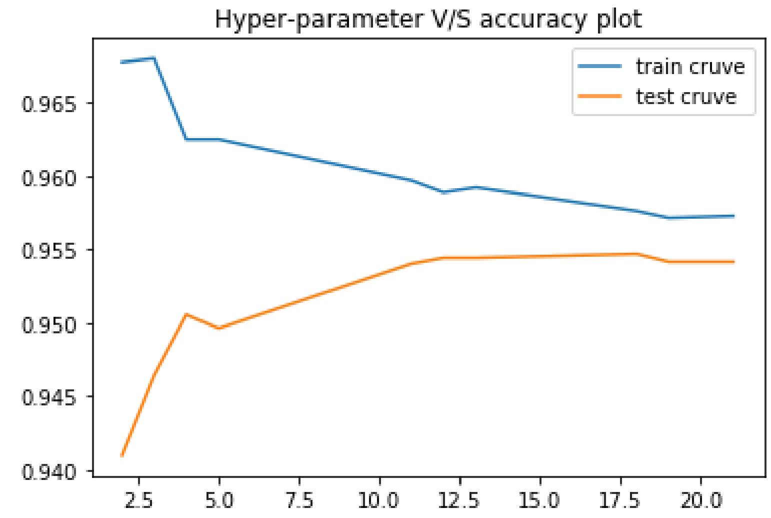
```
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings
warnings.filterwarnings("ignore")

neigh = KNeighborsClassifier()
param_range=(1,30)
folds = 3
trainscores,testscores = RandomSearchCV(X_train,y_train,neigh, param_ra
nge, folds) # calling the function
plt.plot(params,trainscores, label='train cruve')
```

```
plt.plot(params,testscores, label='test cruve')
plt.title('Hyper-parameter V/S accuracy plot')
plt.legend()
plt.show()
print(params)
```

```
[2, 3, 4, 5, 11, 12, 13, 18, 19, 21]
```

```
100%|██████████| 10/10 [00:01<00:00,  7.05it/s]
```




```
[2, 3, 4, 5, 11, 12, 13, 18, 19, 21]
```

- from the plot we can see that k=18 is the best hyper parameter

In [47]:
```
def plot_decision_boundary(X1, X2, y, clf):
    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)
    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()
```

Drawing decision boundry for optimal k

In [48]:

```
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 18)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```