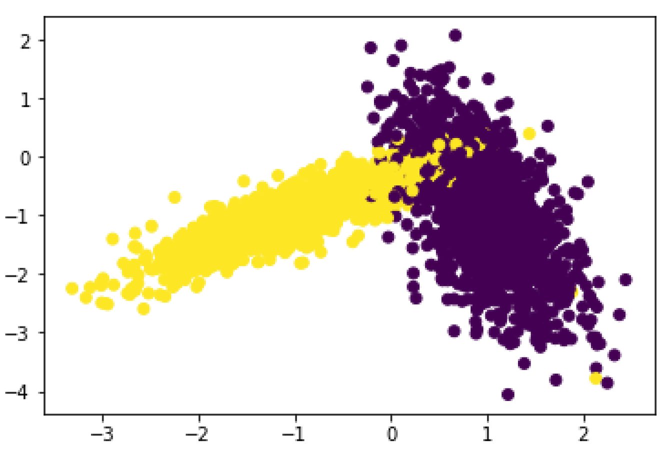
In [66]:
```python
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances


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)

# del X_train,X_test
```

In [15]:
```python
%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()
```



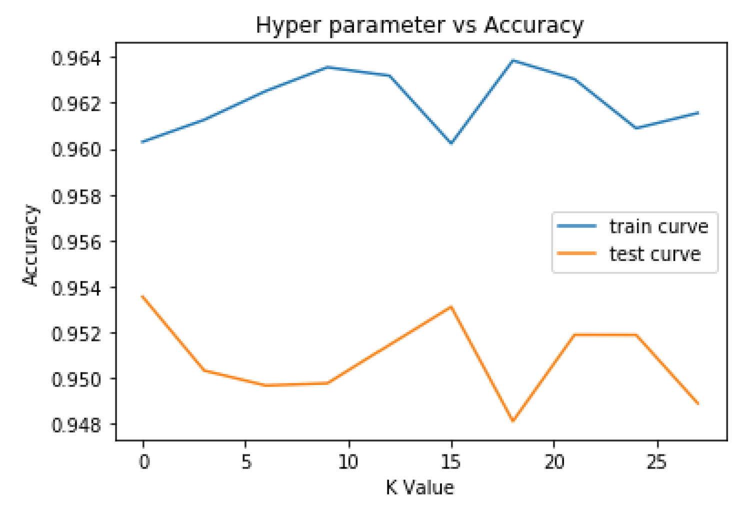
Implementing Custom RandomSearchCV

```
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 data and test our m
odel


    #1.generate 10 unique values(uniform random distribution) in the given range "param_rang
e" and store them as "params"
    # ex: if param_range = (1, 50), we need to generate 10 random numbers in range 1 to 50
    #2.devide numbers ranging from  0 to len(X_train) into groups= folds
    # ex: folds=3, and len(x_train)=100, we can devide numbers from 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 you will do cross-validation a
s follows

        # first we will keep group 1+group 2 i.e. 0-66 as train data and group 3: 67-100 as t
est 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 train data and group 1: 0-33 as t
est data, and find train and
          test accuracies
        # based on the 'folds' value we will do the same procedure

        # find the mean of train accuracies of above 3 steps and store in a list "train_score
s"
        # 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, param_range, folds) and store th
e returned values into "train_score", 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 s
hown in the last cell of reference notebook
```

In [23]:
```python
#Majority of the code has been refered from the GridSearchCV function it seems to be perfect.

from sklearn.metrics import accuracy_score
def randomly_select_60_percent_indices_in_range_from_1_to_len(x_train):
    return random.sample(range(0, len(x_train)), int(0.6*len(x_train)))

# opt 2 and 3 are performed below:
def RandomSearchCV(x_train,y_train,classifier, param_range, folds):
    train_score = []
    test_score = []
    for k in tqdm(param_range):
        trainscores_folds = []
        testscores_folds = []
        for j in range(0, folds):
            train_indices = randomly_select_60_percent_indices_in_range_from_1_to_len(x_train)
            test_indices = list(set(list(range(1, len(x_train)))) - set(train_indices))

            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_neighbor = 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_score.append(np.mean(np.array(trainscores_folds)))
        test_score.append(np.mean(np.array(testscores_folds)))
#4. Returning value of train_score, test_score
    return train_score, test_score
```

In [82]:
```python
from sklearn.neighbors import KNeighborsClassifier
import random
import warnings
warnings.filterwarnings('ignore')

neigh = KNeighborsClassifier()

#1. Generating unique and uniform random variable for a given range.
parm_range = np.arange(0,30,3)
param = np.sort(parm_range)
print(param)
folds = 3

#5. Calling the RandomSearchCV function
train_score,cv_score = RandomSearchCV(X_train,y_train,neigh, param_range, folds)

#6. Plotting the graph Hyper parameter vs Accuracy
plt.plot(param,trainscores, label='train curve')
plt.plot(param,testscores, label='test curve')
plt.title('Hyper parameter vs Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('K Value')
plt.legend()
plt.show()
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
[ 0  3  6  9 12 15 18 21 24 27]

 10%|█         | 1/10 [00:00<00:01,  4.90it/s]

 20%|██        | 2/10 [00:00<00:01,  5.34it/s]

 30%|███       | 3/10 [00:00<00:01,  5.48it/s]

 40%|████      | 4/10 [00:00<00:01,  5.65it/s]

 50%|█████     | 5/10 [00:00<00:00,  5.98it/s]

 60%|██████    | 6/10 [00:00<00:00,  6.06it/s]

 70%|███████   | 7/10 [00:01<00:00,  5.99it/s]

 80%|████████  | 8/10 [00:01<00:00,  5.98it/s]

 90%|█████████ | 9/10 [00:01<00:00,  5.84it/s]

100%|██████████| 10/10 [00:01<00:00,  5.94it/s]
```



In [83]:
```python
#Used the same set of codes to perform this operation.

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)
    # 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 [84]:
```python
#Selected K-NN : 15 since looking the above plot, it seems 15 represents optimal K value.
#7. plotting the decision boundary graph

from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 15)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```