In [3]:
```
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, rando
m_state=60)
X_train, X_test, y_train, y_test = train_test_split(x,y,stratify=y,random_state=42)
```

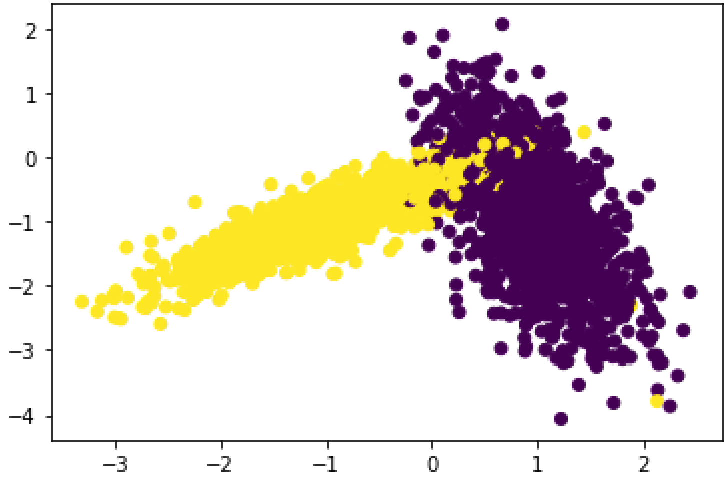
In [4]:
```
length_x_train=len(X_train)
length_x_train
```

Out[4]: 7500

In [5]:
```
length_y_train=len(y_train)
length_y_train
```

Out[5]: 7500

In [6]:
```
%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

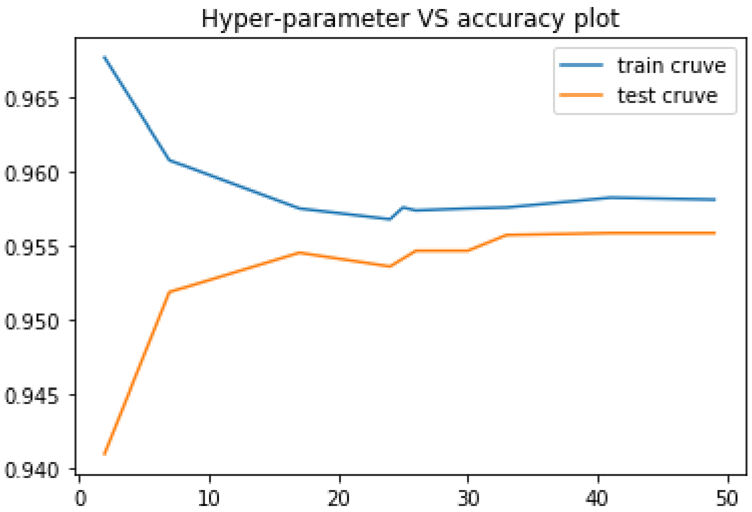
```
In [7]: from sklearn.metrics import accuracy_score #importing for accuracy
        import numpy as np #we importing numpy for scientific calculation purpose

        # we writing this function for breaking the train data into cross validation and next train data.
        def train_test_data_splitter(no_of_datapoints,folds):
            list_of_test_data = []
            spliting_form_of_train_data = []

            for i in range(folds+1):
                list_of_test_data.append(i*(no_of_datapoints/folds))

            for j in range(folds):
                spliting_form_of_train_data.append((int(list_of_test_data[j]),int(list_of_test_data[j+1])))
            return spliting_form_of_train_data

        def RandomSearchCV(x_train,y_train,classifier, params, folds):
            no_of_datapoints = len(x_train)
            index_provider   = train_test_data_splitter(no_of_datapoints,folds)
            trainscores = []
            testscores  = []

            for k in tqdm(params):
                trainscores_folds=[]          #initialize two list for store accuracy value
                testscores_folds =[]

                for j in range(folds):
                    test_indices=index_provider[j] #provide the index of tuple which store in list
                    test_indices=list(test_indices) #we converting in list for finding the range



                    X_test = x_train[test_indices[0]:test_indices[1]]
                    Y_test = y_train[test_indices[0]:test_indices[1]]

                    # here we finding X_train and Y_train part
                    X_train=x_train[list(set(list(range(0,len(x_train))))-set(list(range(test_indices[0],test_indices[1]))))]
                    Y_train=y_train[list(set(list(range(0,len(y_train))))-set(list(range(test_indices[0],test_indices[1]))))]
```

```python
            classifier.n_neighbors = k

            #making the model
            classifier.fit(X_train,Y_train)

            # find testscore predicted value
            Y_predicted = classifier.predict(X_test)

            # # find prediction of x_train data
            testscores_folds.append(accuracy_score(Y_test, Y_predicted))

            # find prediction of x_train data
            Y_predicted = classifier.predict(X_train)

            # find accuracy and append the value
            trainscores_folds.append(accuracy_score(Y_train,Y_predicted))

          # here we first finding the mean and then append the value in trainscore and testscore
        trainscores.append(np.mean(np.array(trainscores_folds)))
        testscores.append(np.mean(np.array(testscores_folds)))

    return trainscores,testscores # function return values
```

In [11]:

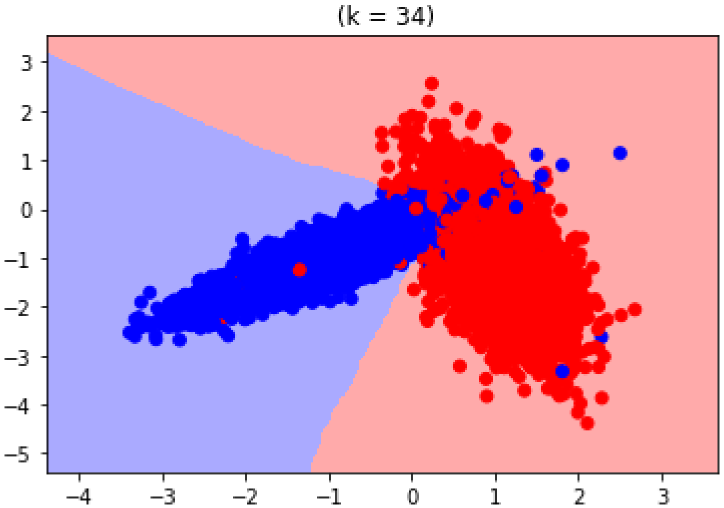
```python
from sklearn.metrics import accuracy_score #for accuracy
from sklearn.neighbors import KNeighborsClassifier #importing classifier
import matplotlib.pyplot as plt #for ploting the graph
import random # for randomly select the numbers
import warnings

warnings.filterwarnings('ignore') #ignore the warnings messages

neigh=KNeighborsClassifier()
#generate 10 value randomly between 1 to 50
params=random.sample(range(1,50), 10)

# sorting of generated value
params.sort()

#initialize the number of folds
folds=3

# calling the function
trainscores,testscores=RandomSearchCV(X_train,y_train,neigh,params,folds)

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

```
100%|██████████| 10/10 [00:11<00:00,  1.11s/i
t]
```



In [14]:
```python
####sir this code take from your reference book####


def plot_decision_boundary(X1, X2, y, clf):
        # this code for coloring the graph
    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(" (k = %i)" % (clf.n_neighbors))
    plt.show()
```

In [15]:
```
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 [ ]: