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

```

In [ ]:
```

```

two class classification dataset

In [2]:
```
x,y = make_classification(n_samples=10000, n_features=2, n_informative=2, n_r
X_train, X_test, y_train, y_test = train_test_split(x,y,stratify=y,random_sta


```

In [3]:
```
print("size of xtrain and ytrain: ",len(X_train),len(y_train))
print("size of xtest and ytest : ",len(X_test),len(y_test))
```

```
size of xtrain and ytrain:  7500 7500
size of xtest and ytest :  2500 2500
```

In [52]:
```
X_train
```

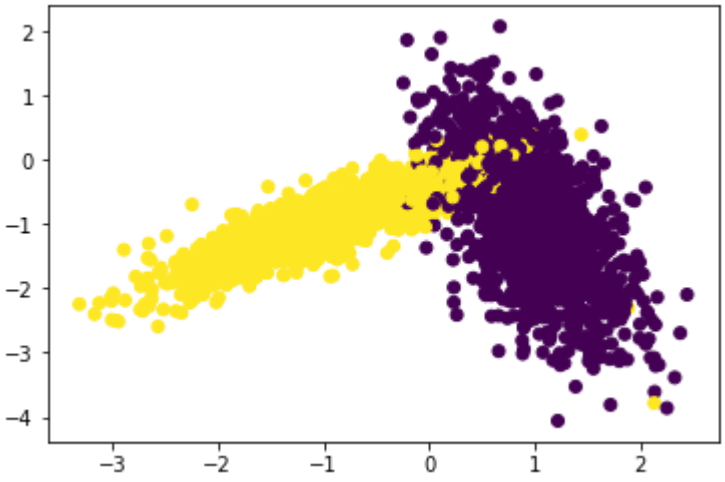
Out[52]:
```
array([[ 0.45267141, -1.42381257],
       [ 0.61696406, -0.00418956],
       [-1.80708012, -1.34499648],
       ...,
       [ 0.63107723, -0.4743162 ],
       [-0.47320722, -0.6387028 ],
       [ 1.07909424, -1.67541279]])
```

In [33]:
```
y_train
```

Out[33]:
```
array([0, 0, 1, ..., 0, 1, 0])
```

Distribution of 2 classes against the y label

In [4]:
```
%matplotlib inline
import matplotlib.pyplot as plt
colors = {0:'red', 1:'blue'}
plt.scatter(X_test[:,0], X_test[:,1],c=y_test)
# plt.title("distribution of 2 classes")
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 model


    #1.generate 10 unique values(uniform random distribution) in the give
n range "param_range" 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 d
o cross-validation as follows

        # first we will keep group 1+group 2 i.e. 0-66 as train data and
group 3: 67-100 as test 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 an
d group 1: 0-33 as test 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_scores"
        # 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 the 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 shown in the last cell of reference notebook
```

In [49]:

```
from random import random
def randm():
    nlist=[]
    for i in range(1,50):
        nlist.append(i)
    n=np.random.choice(nlist, 10, replace=False)
    n.sort()
    return n
n=randm()
print(n)
```

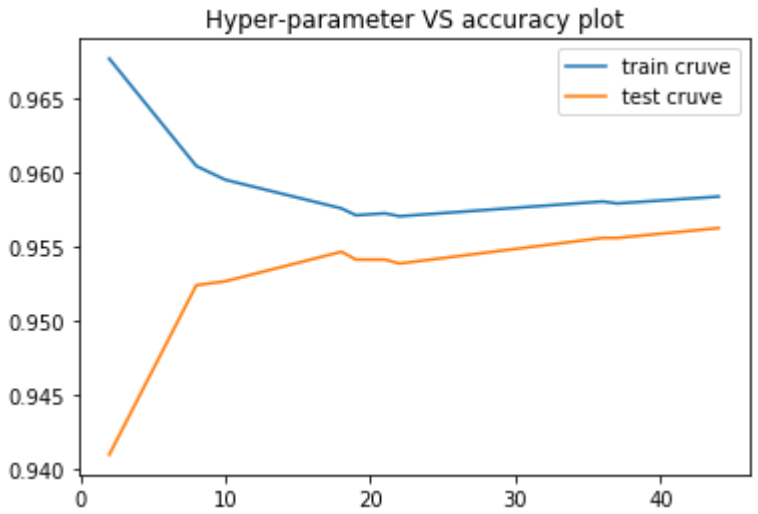
```
[ 2  8 10 18 19 21 22 36 37 44]
```

In [50]:

```
# referance=#https://github.com/arnisudheendra/test/blob/master/RANDOM%20SEAR
# it will take classifier and set of values for hyper prameter in dict type d
# we are implementing this only for KNN, the hyper parameter should n_neighbo
from sklearn.metrics import accuracy_score
def Randomsearchcv(x_train,y_train,classifier, param_range, folds):
    train_scores = []
    test_scores  = []
    params = {'n_neighbors':n}
    for k in tqdm(params['n_neighbors']):
        trainscoresf = []
        testscoresf = []
        for i in range(0, folds):
            values=len(x_train)/folds
            values1=int(values)
            test_indices=list(set(list(range((values1*i), (values1*(i+1))))))
            train_indices = list(set(list(range(0, len(x_train)))) - set(test
            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)
            testscoresf.append(accuracy_score(Y_test, Y_predicted))
            Y_predicted = classifier.predict(X_train)
            trainscoresf.append(accuracy_score(Y_train, Y_predicted))
        train_scores.append(np.mean(np.array(trainscoresf)))
        test_scores.append(np.mean(np.array(testscoresf)))
    return train_scores,test_scores,params
```

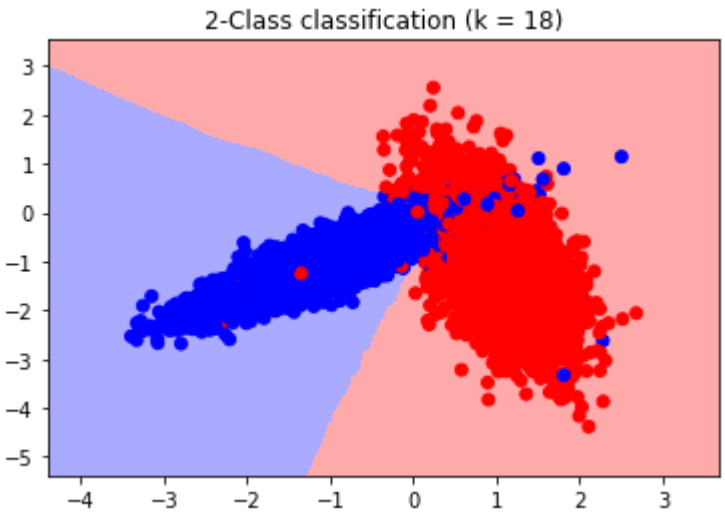
In [51]:
```
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings
warnings.filterwarnings("ignore")


neigh = KNeighborsClassifier()

param_range = {'n_neighbors':n}
print("Random Values = ", param_range)
folds = 3

trainscores,testscores,params = Randomsearchcv(X_train, y_train, neigh, param


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

```
 10%|████                                      | 1/10 [00:00<00:01,  8.06it/s]

Random Values =  {'n_neighbors': array([ 2,  8, 10, 18, 19, 21, 22, 36, 37, 4
4])}

100%|████████████████████████████████████████| 10/10 [00:03<00:00,  3.22it/s]
```



In [30]:
```
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_ma
    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 [46]:
```
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)
```



In [ ]:
```
1
```