In [42]:

```
#generates a random n class classification problem-https://scikit-learn.org/stable/module
s/generated/sklearn.datasets.make_classification.html
from sklearn.datasets import make_classification

#to split data into train and test
from sklearn.model_selection import train_test_split

#to standardise the data-mean=0 and std dev=1-https://scikit-learn.org/stable/modules/gen
erated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
import seaborn as sns

#to find the eucliadean distance between d dimensional vectors-https://scikit-learn.org/s
table/modules/generated/sklearn.metrics.pairwise.euclidean_distances.html
from sklearn.metrics.pairwise import euclidean_distances

#to measure the accuracy
from sklearn.metrics import accuracy_score
```

In [43]:

```
#x is the nd-array storing the samples(n) and the features(d)
#y stores the list of class labels for the n samples
#generates a random n class classification problem
#https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.h
tml
x,y=make_classification(n_samples=1000,n_features=2,n_informative=2,n_redundant=0,n_clas
ses=2,n_clusters_per_class=2)
```

In [44]:

```
print(x)
print("\n")
print(len(x))
print("\n")
print(y)
print("\n")
print(len(y))
```

```
[[ 0.72164587  0.40105383]
 [-1.39966883  1.33507737]
 [ 0.21154823 -0.88118737]
 ...
 [-1.40884507 -1.23603973]
 [ 1.18861099  0.0932105 ]
 [ 1.16962599  0.38385689]]


1000


[1 0 0 0 1 1 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 1 1 0 1 1 0 1 0 0 0 0 1 1 1 1 1
 0 1 1 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 1 0 0 1 0 1 0 1 0 1
 1 1 0 1 1 1 0 0 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 0 1 1
 1 0 1 0 1 0 0 1 1 1 0 1 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1
 0 1 0 1 0 1 1 1 0 0 1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 0
 1 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 1 0 0 1 1
 1 0 1 0 0 0 0 1 1 1 0 1 1 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1
 1 0 1 0 0 1 1 1 0 1 1 0 1 0 1 1 0 0 0 0 0 1 1 1 1 1 0 1 0 0 1 0 0 0 0 1 0
 0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 1 1 0 0 1 0 0 1 1 0 0 1 0 1 1 1 0 1
 0 0 0 0 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 0 1 0 0 0 0 0
 1 0 0 1 0 0 0 1 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 1 1 0
 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 0 1 0 1 1 0 1 1 0 0 0 1 1 0 1 0 1 0 1 1 1
 0 0 1 0 1 1 0 0 1 0 1 0 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0
 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 1
 0 0 0 0 0 1 0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 1 1 0 0 0 0 1 1 1 0 0 1 0 0
 0 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 1 1
 1 1 0 1 0 0 0 1 0 1 0 1 1 1 1 1 0 1 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 1
 0 0 1 0 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 0 0 0 0 1 1 1 1 1 0 0
 0 0 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 1 0 0 0 0 1 1 0 1 1 1 0 0 0 1 1 1 0 1 1
 0 0 1 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 1 1 1 1 1 0 1 0 0 0 1 0
 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 1
 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 1 0 1 1 0 1 1 0 0 0 1 1 0
 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 1 0 1 0 1 1 1
 1 0 1 0 1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1 0 1 1 1 1 1 0 0 0 1 0 0 1 0 1
 1 1 0 0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0
 1 0 0 1 0 0 1 0 0 1 0 1 1 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 1 1 0 0 1
 1]
```

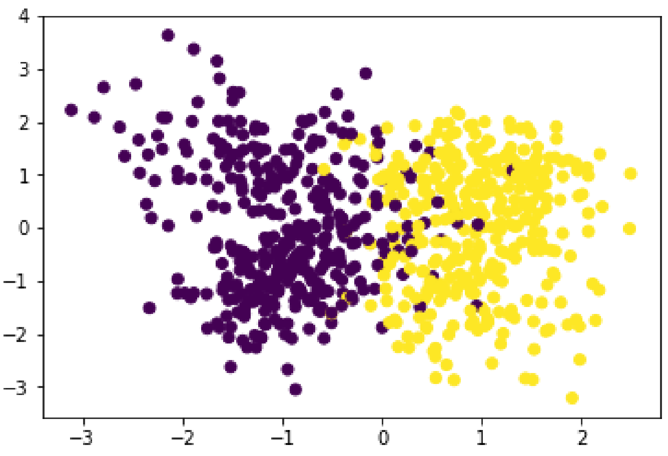
```
1000
```

In [45]:

```python
#creating training and testing data from our dataframe
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [46]:

```python
#https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.scatter.html
import matplotlib.pyplot as plt
plt.scatter(x_train[:,0],x_train[:,1],c=y_train)
#1st parameter-x coord
#2nd parameter-y coord
#3rd parameter-classes labels
plt.show()
```



IMPLEMENTING CUSTOM RANDOM SEARCH CROSS VALIDATION

In [47]:

```python
# 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_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 do cross-validation as fo
```

```
llows

# 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 and 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 stor
e the returned values into "train_score", and "cv_scores"
#6. plot hyper-parameter vs accuracy plot as shown in reference notebook and choose the b
est hyperparameter
#7. plot the decision boundaries for the model initialized with the best hyperparameter,
as shown in the last cell of reference notebook
```

In [48]:

```python
import random

def return_10_random(start,end): #returns a list of 10 randomly generated numbers in the
given range
    pop=[i for i in range(start,end+1)]
    l=random.sample(pop,10) #samples 10 unqiue points randomly from the range
    return l
```

In [51]:

```python
def RandomSearchCV(x_train,y_train,classifier, param_range, folds):
    training_accuracy=[] #stores the training accuracy of various k
    cross_val_accuracy=[] #stores the cv accuracy of various k

    params=return_10_random(param_range[0],param_range[1]) #gives 10 randomly chosen num
bers from the range
    params.sort() #sorting the parameters in ascending  order
    #in grid search we took the whole parameters already given to us

    for k in tqdm(params):
        classifier.set_params(n_neighbors=k) #the model has been described for a particu
lar k

        x_folds=np.array_split(x_train,folds) #splits the training input into equal samp
les of size folds
        y_folds=np.array_split(y_train,folds) #spits the training output into equal samp
les of size folds


        training_accuracy_fold=[] #stores the training accuracy for the various fold case
s of a k
        cross_val_accuracy_fold=[] #stores the  cv accuracy for the various fold cases of
a k

        for i in range(folds): #for each k we repeat this process folds number of time
            x_cv=x_folds[i] #array of input of a particular fold-the ith fold is the cv
dataset
            y_cv=y_folds[i] #array of output of a particular fold-the ith fold is the cv
dataset

            l_input=[array for array in x_folds] #list containing all input folds
            l_output=[array for array in y_folds] #list containing all output folds
            l_input.pop(i) #removing the cv data input-pop is used to remove by index-rem
oving the ith fold by index
            l_output.pop(i) #removing the cv data output-pop is used to remove by index-r
emoving the ith fold by index

            x_train_final=np.concatenate([ele for ele in l_input]) #this contains all in
put except the cv input
            y_train_final=np.concatenate([ele for ele in l_output]) #this contains all o
utput except the cv output

            classifier.fit(x_train_final,y_train_final) #training the model on the traini
ng data

            #calculation of training accuracy
            y_predict=classifier.predict(x_train_final)
            train_acc=accuracy_score(y_train_final,y_predict,normalize=True)
            training_accuracy_fold.append(train_acc)

            #calculation of cv accuracy
            y_predict=classifier.predict(x_cv)
            cv_acc=accuracy_score(y_cv,y_predict,normalize=True)
            cross_val_accuracy_fold.append(cv_acc)

        #now for a particular k and for all folds of it training and cv accuracy has been
calculated so we take their mean
        #to get the final train and cv accuracy score for that k
        training_accuracy.append(np.mean(training_accuracy_fold))
        cross_val_accuracy.append(np.mean(cross_val_accuracy_fold))


    return params,training_accuracy,cross_val_accuracy
```

In [52]:

```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings #to ignore warnings if any arise
warnings.filterwarnings("ignore")


#we just describe the model with no parameters-because the function inputs the classifier
separately and the parameters sep
knn_model = KNeighborsClassifier()

#we send parameter as a tuple in random search cv (a,b) st a<b
#the function generates random 10 k from this range
#in gridsearch cv we used to send the complete range of k's/parameters we wantes
param_range = (1,30)

#cross-validation folds
folds = 3

params,train_accuracy_scores, cross_validation_accuracy_scores= RandomSearchCV(x_train,
y_train, knn_model, param_range, folds)
```

```
100%|████████████████████████████████████████████████████████████████████████████████|
10/10 [00:00<00:00, 30.51it/s]
```

In [53]:

```
print("the 10 randomly chosen parameters (k's) from the given range are: \n")
print(params)
```

```
the 10 randomly chosen parameters (k's) from the given range are:

[8, 9, 13, 16, 17, 19, 20, 23, 26, 27]
```

In [54]:

```
print("the training accuracy scores of the 10 randomly chosen parameters from the range g
iven: \n")
print(train_accuracy_scores)
print("\n")
print(len(train_accuracy_scores))
```

```
the training accuracy scores of the 10 randomly chosen parameters from the range given:

[0.9478530050576994, 0.9449963698523128, 0.9421427980626959, 0.9407121828981752, 0.942142
7980626959, 0.9399984070237997, 0.9435688181035617, 0.942139734646926, 0.9392815677336545
, 0.9399953436080298]


10
```

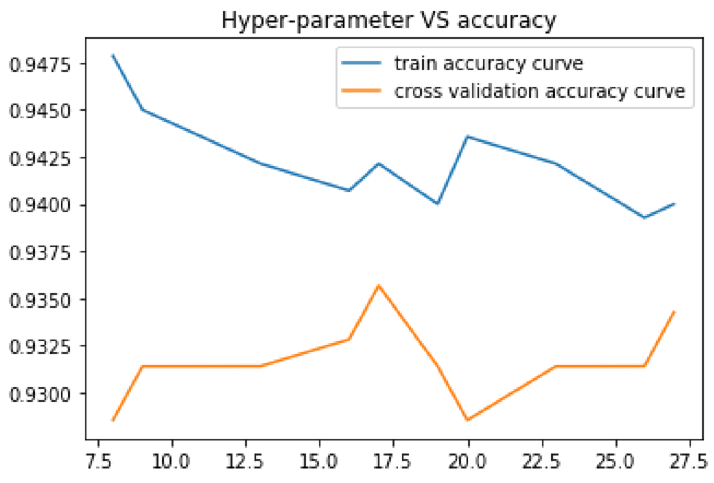
In [55]:

```
print("the cross validation accuracy scores of the 10 randomly chosen parameters from the
range given: \n")
print(cross_validation_accuracy_scores)
print("\n")
print(len(cross_validation_accuracy_scores))
```

```
the cross validation accuracy scores of the 10 randomly chosen parameters from the range
given:

[0.9285487203942141, 0.9313977232432169, 0.931403836983236, 0.9328283384077376, 0.9356834
549967598, 0.9313977232432169, 0.9285426066541946, 0.9313977232432168, 0.931403836983236,
0.9342650673122775]


10
```

In [56]:

```
plt.plot(params,train_accuracy_scores, label='train accuracy curve')
plt.plot(params,cross_validation_accuracy_scores, label='cross validation accuracy curve'
)
plt.title('Hyper-parameter VS accuracy ')
plt.legend()
plt.show()
```



VVV IMPORTANT- we take that hyperparameter where the difference between the cv accuracy and the train accuracy is the minimum

In [57]:

```
diff=np.array(train_accuracy_scores)-np.array(cross_validation_accuracy_scores)
least_diff=min(diff)
diff=list(diff)
idx=diff.index(least_diff) #idx of the parameter where the diff is the least
print(diff)
print("\n")
print(len(diff))
print("\n")
print(idx)
```

```
[0.019304284663485327, 0.013598646609095932, 0.010738961079459819, 0.007883844490437553,
0.006459343065936096, 0.008600683780582763, 0.015026211449367l1, 0.010742011403709228, 0.
007877730750418421, 0.005730276295752291]


10


9
```

In [58]:

```
parameter_optimised=params[idx] #finding the value of the best k
print("the final paramter after optimisation: ",parameter_optimised)
```

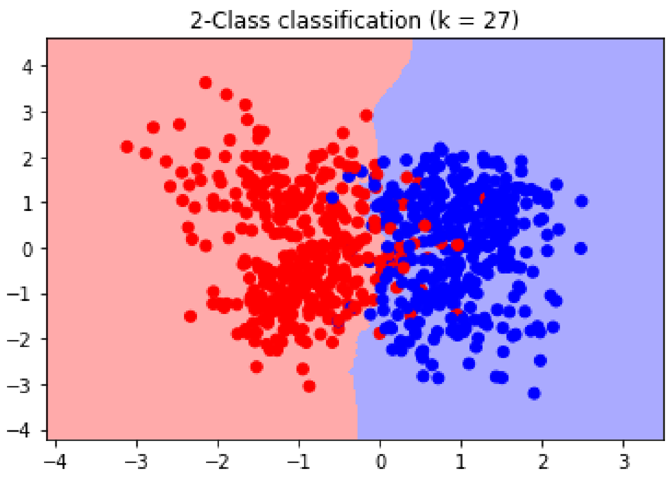
```
the final paramter after optimisation:  27
```

In [59]:

```
# 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_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 [60]:

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



Observe a smooth decision boundary which is a feature of a model with the best possible hyperparameter-here K in KNN

we have completed implementing the randomsearch cross validation function successfully