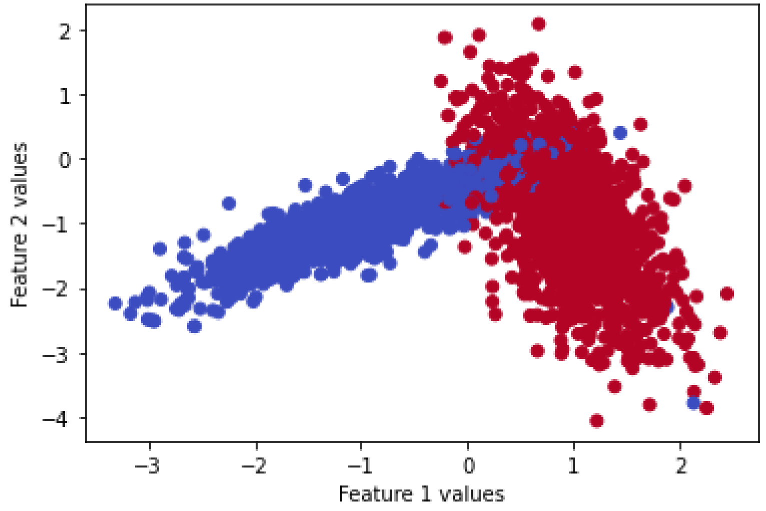
In [2]:
```
# necessary libraries:
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances

# create a binary classification and split into train and test datasets
x,y = make_classification(n_samples=10000, n_features=2, n_informative=2, n_redundant= 0, n_clusters_per_class=1, rar
X_train, X_test, y_train, y_test = train_test_split(x,y,stratify=y,random_state=42)

# del X_train,X_test
```

In [2]:
```
# printing the size of training and testing datasets:
print(len(X_train))
print(len(y_train))
print(len(X_test))
print(len(y_test))
```

```
7500
7500
2500
2500
```

In [3]:
```
# Scatter plot between feature 1 and 2 of test data along with its respective classes.
%matplotlib inline
import matplotlib.pyplot as plt
plt.scatter(X_test[:,0], X_test[:,1],c=y_test,cmap = "coolwarm_r")
plt.xlabel("Feature 1 values")
plt.ylabel("Feature 2 values")
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 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
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 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 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
```

Implementing custom randomsearchCV:

In [4]:
```
from sklearn.metrics import accuracy_score # necessary library

# A function to generate 10 random hyperparameter ->K values(Neighbors)
def generate_10_random_kvalues(params):
    params = list(range(params[0],params[1]))
    n_neighbors = random.sample(params,10)
    return n_neighbors

# A function to randomly search the optimal hyperparameters using training data
def RandomSearch(x_train,y_train,classifier,param_range,folds):
    trainscores = []              # store final training accuracies into this list
    cvscores = []                 # store final validation accuracies into this list

    # store nearest neighbors by calling the function()
    n_neighbors = generate_10_random_kvalues(param_range)

    for k in tqdm(n_neighbors):   # Start the search with the random "n_neighbors"

        trainscores_folds = []    # Train accuracy of each fold is stored in this list
        cvscores_folds  = []      # Cv accuracy of each fold is stored in this list

        # create a list of indices of the training data size & split into "K" equal folds
        indices = [i for i in range(len(x_train))]
        indices_toarray = np.asarray(indices)
        k_folds = np.array_split(indices_toarray,folds)

        for fold in k_folds:      # for each fold in k_fold run the loop

            # seperate the cv and train indices for each fold
            cv_indices = fold.tolist()
            train_indices = list(set(list(range(1, len(x_train)))) - set(cv_indices))

            # create train and cv group with their respective indices:
            X_train = x_train[train_indices]
            Y_train = y_train[train_indices]
            X_cv = x_train[cv_indices]
            Y_cv = y_train[cv_indices]

            # Use the classifier to fit the training data:
            classifier.n_neighbors = k
            classifier.fit(X_train,Y_train)

            # predict for cross validated group/set
            Y_predicted = classifier.predict(X_cv)
            cvscores_folds.append(accuracy_score(Y_cv, Y_predicted))# fetch accuracies

            # predict for training data set
            Y_predicted = classifier.predict(X_train)
            trainscores_folds.append(accuracy_score(Y_train, Y_predicted))# fetch accuracies

        # compute the mean of train and cv accuracies of each fold & store it in a list
        trainscores.append(np.mean(np.array(trainscores_folds)))
        cvscores.append(np.mean(np.array(cvscores_folds)))

    return trainscores,cvscores,n_neighbors # return the desired scores
```

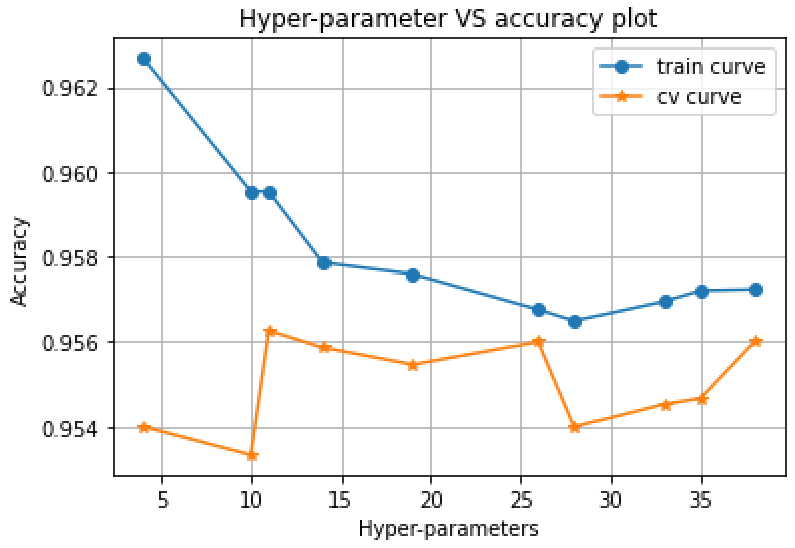
In [6]:

```
# necessary libraries:
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings
warnings.filterwarnings("ignore")

# A custom function To reduce jaggedness of the final plot
# Sort the K values and then plot
def forclean_plot(scores,n_neighbors):
    zippedlist = list(zip(n_neighbors,scores))
    sorted_list = sorted(zippedlist,key=lambda x:x[0])
    list1 = [];list2 = []
    for i in sorted_list:
        list1.append(i[0])
        list2.append(i[1])
    return list2,list1

# its KNN classifier
neigh = KNeighborsClassifier()

# Required lower limit and upperlimit of "k"
a = int(input("The lower limit value for k :"))
b = int(input("The upper limit value for k :"))

param_range = (a,b) # Initial parameter range

folds = int(input("The Number of folds are: "))
#folds = 10(Thumb rule)

# call the randomsearch()
trainscores,cvscores,n_neighbors = RandomSearch(X_train, y_train, neigh, param_range,folds)

trainscores,n_neighbors = forclean_plot(trainscores,n_neighbors)
cvscores,n_neighbors = forclean_plot(cvscores,n_neighbors)

# PLOT A HYPER-PARAMETER VS ACCURACY:
```

```
plt.grid()
plt.plot(n_neighbors,trainscores, label='train curve',marker='o')
plt.plot(n_neighbors,cvscores, label='cv curve',marker='*')
plt.title('Hyper-parameter VS accuracy plot')
plt.xlabel("Hyper-parameters")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

```
The lower limit value for k :2
The upper limit value for k :45
The Number of folds are: 10
```

```
100%|██████████████████████████████████████████████████████████████████| 10/10 [00:33<00:00,  3.31s/i
t]
```



In [8]:
```
# Lets see the accuracy values for the above plot:
print("The randomsearch K values :",n_neighbors)
print("\nThe training accuracies for different K values:",trainscores)
print("\nThe validation accuracies for different K values:",cvscores)
```

```
The randomsearch K values : [4, 10, 11, 14, 19, 26, 28, 33, 35, 38]

The training accuracies for different K values: [0.9626764985759209, 0.9595353363735644, 0.9595353385686769, 0.9578
610493735698, 0.9575943409997641, 0.9567646038096178, 0.9564979108016001, 0.956957224938674, 0.9572091075221021, 0.
9572387481272946]
```

```
The validation accuracies for different K values: [0.954, 0.9533333333333334, 0.9562666666666665, 0.9558666666666666
5, 0.9554666666666666, 0.9559999999999998, 0.954, 0.9545333333333332, 0.9546666666666667, 0.9559999999999998]
```

Optimal K = 26 since this parameter has high and close accuracy difference of (0.9567-0.9559 = 0.0008) for cv and train curve.

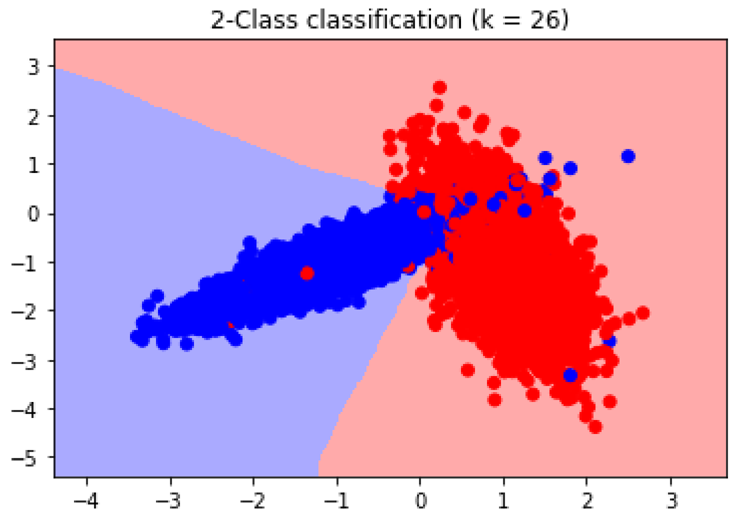
In [10]:

```python
# understanding this code line by line is not that important
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()
```

Decision boundary for optimal hyperparameter:

In [11]:

```python
# Lets plot the decision boundary for our training data with best hyperparameter:
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 26)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```


2-Class classification (k = 26)
3
2
1
0
−1
−2
−3
−4
−5
−4
−3
−2
−1
0
1
2
3