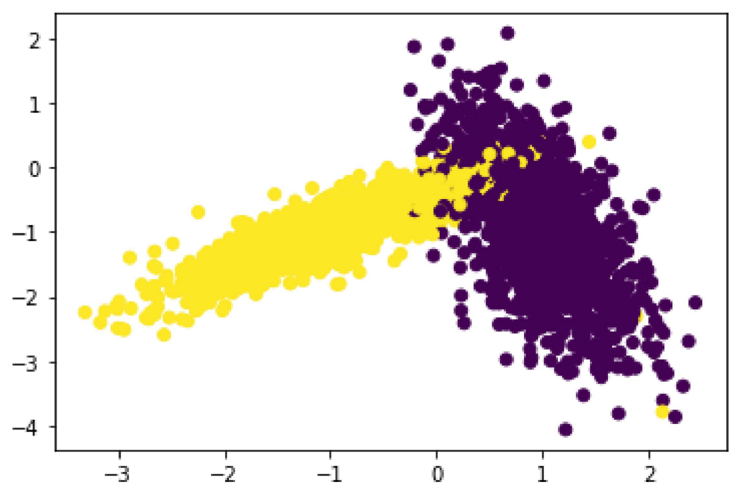
In [1]:

```
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


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

# del X_train,X_test
```

In [2]:

```
%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 th
e data and test our model


    #1.generate 10 unique values(uniform random distribution) in the giv
en range "param_range" and store them as "params"
```

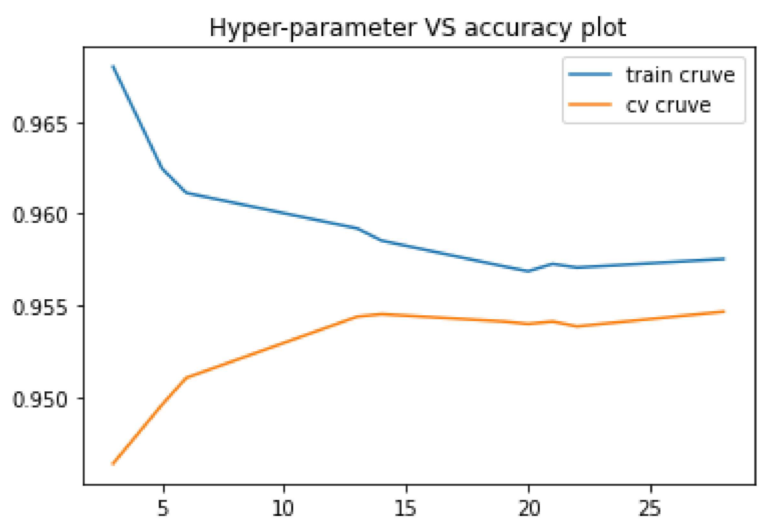
```
    # ex: if param_range = (1, 50), we need to generate 10 random number
s in range 1 to 50
    #2.devide numbers ranging from  0 to len(X_train) into groups= folds
    # ex: folds=3, and len(x_train)=100, we can devide numbers from 0 to
100 into 3 groups
      group 1: 0-33, group 2:34-66, group 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 a
nd 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 i
n 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_rang
e, folds) and store the returned values into "train_score", and "cv_scor
es"
#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 [9]:

```
from sklearn.metrics import accuracy_score

#function to implement random search cv, we pass the below parameters:
# x_train,y_train = randomly generated train data
# classifier = knn
# param_range = tuple (a,b) a<b; for selecting the k randomly
# folds = number of folds/ n
def random_search(x_train,y_train,classifier, param_range, folds):
    # lists of final train and cv scores for random values of k
    train_scores = []
    cv_scores  = []
    # dividing the train data/total folds gives the size of each fold
    split = len(x_train)//folds
    param_list = list(np.arange(*param_range))
    # uniformly distributed values of k is stored in params
    params = sorted(random.sample(param_list,10))

    for k in tqdm(params):
        # lists of train and cv scores for different set of folds
        trainscores_folds = []
        cvscores_folds  = []

        for j in range(0, folds):
            #each loop generates different set of train and cv data
            start = j* split #start point of a fold
            cv_indices  = np.arange(start, start+split)# end point =start+siz
            #subtracting cv_indices from the total indices of x_train to get
            train_indices = list(set(np.arange(0,len(x_train)))-set(cv_indice
            #using the above indices to get the data from the x_train, y_trai
            X_train = x_train[train_indices]
            Y_train = y_train[train_indices]
            X_cv = x_train[cv_indices]
            Y_cv  = y_train[cv_indices]
            #knn classifier for each k in params and for different set of dat
            classifier.n_neighbors = k
            classifier.fit(X_train,Y_train)
            #knn predicting y labels on cv data for different sets of data(fo
            Y_predicted = classifier.predict(X_cv)
            #appending the scores
            cvscores_folds.append(accuracy_score(Y_cv, Y_predicted))
            #knn predicting y labels on train data for different sets of data
            Y_predicted = classifier.predict(X_train)
            #appending the scores
            trainscores_folds.append(accuracy_score(Y_train, Y_predicted))
        # we take the average of the above appended scores, so for each k: we
        train_scores.append(np.mean(np.array(trainscores_folds)))
        cv_scores.append(np.mean(np.array(cvscores_folds)))
    return train_scores,cv_scores,params
```

In [10]:
```python
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings
warnings.filterwarnings("ignore")


knn_fn = KNeighborsClassifier()

param_range = (1,30)
folds = 3

trainscores,cvscores,params = random_search(X_train, y_train, knn_fn, param_r


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

```
100%|██████████████████████████████████████████████████████████████| 10/10 [00:15<00:00,  1.59s/it]
```

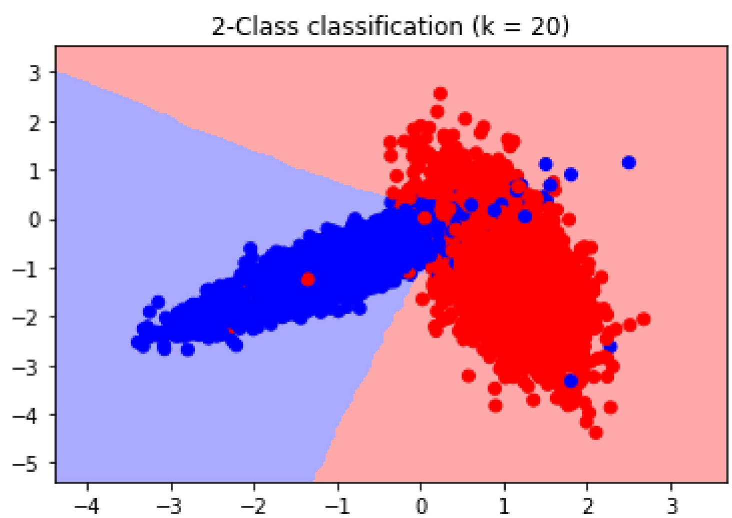


In [11]:

```
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 [12]:

```
from matplotlib.colors import ListedColormap
knn_cla = KNeighborsClassifier(n_neighbors = 20)
knn_cla.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, knn_cla)
```



In [ ]: