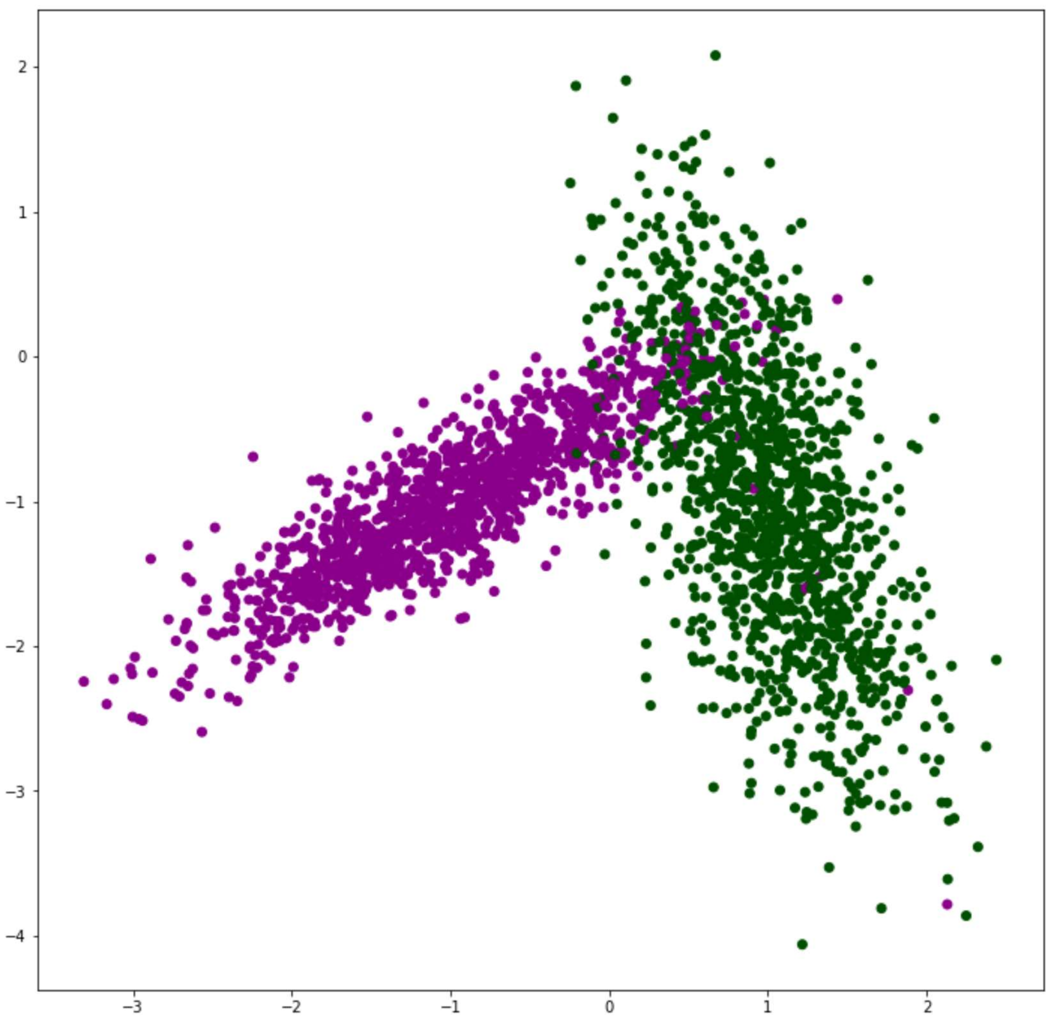
In [3]:

```
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
from sklearn.metrics import accuracy_score
from matplotlib.colors import ListedColormap


x,y = make_classification(n_samples=10000, n_features=2, n_informative=2, n_redundant= 0, n_clusters_per_class=1, random_state=60)
X_train, X_test, y_train, y_test = train_test_split(x,y,stratify=y,random_state=42)

print (len(X_train))
print (len(y_train))
```

```
7500
7500
```

In [4]:

```
%matplotlib inline
import matplotlib.pyplot as plt
colors =['#004F00','#8B008B']
plt.figure(figsize = (12,12))
plt.scatter(X_test[:,0], X_test[:,1],c=y_test, cmap  = ListedColormap(colors) )
plt.show()
```



Implementing Custom RandomSearchCV

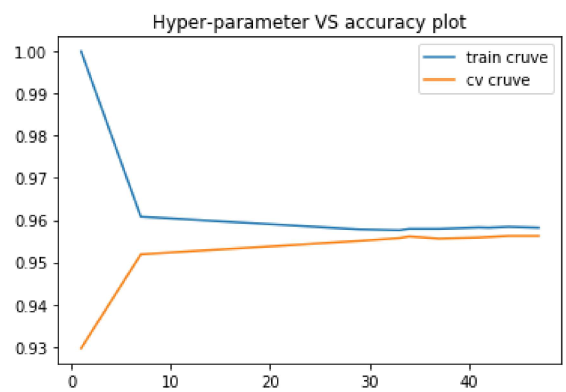
In [6]:

```
def RandomSearchCV(x_train,y_train,classifier, param_range, folds):
    trainscores = []
    cvscores  = []

    params = random.sample(range(param_range[0],param_range[1]), 10)  ## Generating 10 unique values(unif
orm random distribution) in the given range "param_range" and store them as "params"
    params = np.sort(params) ## Sorting the generated value for better graph
    size = int(len(x_train)/folds) ## size of the buckets

    for k in tqdm(params):
        trainscores_folds = []
        cvscores_folds  = []

        for j in range(0, folds):

            ## for each fold defining train and test indices
            train_indices = list(set(list(range(0, len(x_train)))) - set(range(j*size,(j+1)*size)))
            cv_indices  = list(set(range(j*size,(j+1)*size)))

            ## selecting the training data points based on the train_indices and cv_indices
            X_train = x_train[train_indices]
            X_cv  = x_train[cv_indices]
            Y_train = y_train[train_indices]
            Y_cv  = y_train[cv_indices]


            classifier.n_neighbors = k ## initialising KNN classifier neighbours
            classifier.fit(X_train,Y_train) ## Fiting the training data

            Y_cv_predicted = classifier.predict(X_cv) ## Predicting classes of CV dataset
            cvscores_folds.append(accuracy_score(Y_cv, Y_cv_predicted))

            Y_train_predicted = classifier.predict(X_train)
            trainscores_folds.append(accuracy_score(Y_train, Y_train_predicted))
        trainscores.append(np.mean(np.array(trainscores_folds)))
        cvscores.append(np.mean(np.array(cvscores_folds)))
    return params,trainscores,cvscores
```

In [8]:

```
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 = (1,50)
folds = 3

params,trainscores,cvscores = RandomSearchCV(X_train,y_train,neigh, param_range, folds)


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 [00:09<00:00,  1.10it/s]
```



In [11]:

```
score_diff = []

for i in range(0,len(params)):
    diff = trainscores[i] - cvscores[i]
    score_diff.append(diff)

best_param_dict = dict(zip(params, score_diff))
best_param = min(best_param_dict, key=best_param_dict.get)
```

In [13]:

```
best_param_dict
```

Out[13]:

```
{1: 0.07039999999999991,
 7: 0.008933333333333349,
 29: 0.0027333333333333654,
 33: 0.001866666666666572,
 34: 0.0018000000000001348,
 37: 0.0023333333333334094,
 41: 0.0024000000000000687,
 42: 0.0022000000000000908,
 44: 0.0021333333333333204,
 47: 0.0019333333333333425}
```

In [14]:

```
best_param
```

Out[14]:

```
34
```

In [15]:

```
def plot_decision_boundary(X1, X2, y, clf):
    cmap_light = ListedColormap(['#B4EEB4', '#EEAEEE'])
    cmap_bold = ListedColormap(['#004F00','#8B008B'])

    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(figsize = (12,12))
    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 [16]:

```
neigh = KNeighborsClassifier(n_neighbors = best_param)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```