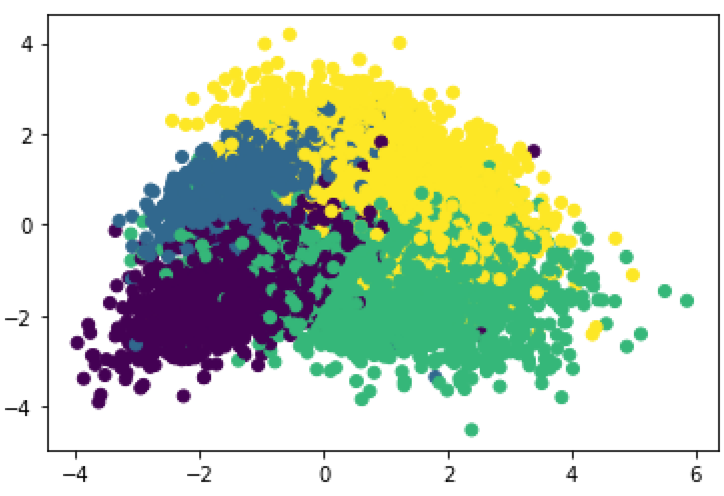
In [1]:

```python
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=50000, n_features=2,n_classes=4, n_informative=2, n_redundant= 0, n_cluster
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]:

```python
%matplotlib inline
import matplotlib.pyplot as plt
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 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
```

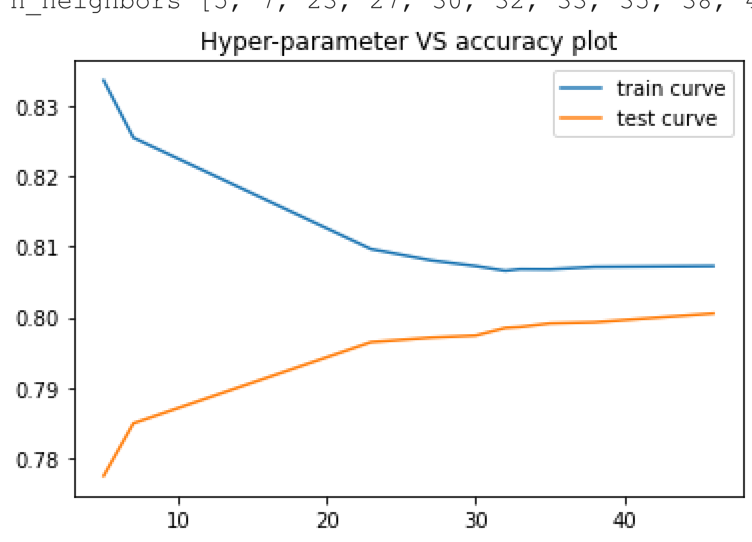
In [3]:

```python
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings
from tqdm import tqdm

def random_numbers(param_range, n):
    numbers = np.random.randint(param_range[0],param_range[1],n)
    values = sorted(numbers)
    return values

def RandomSearch(x_train, y_train, classifier, param_range, folds):
    trainscores = []
    testscores = []
    for K in param_range:
        fold_trainscores = []
        fold_testscores = []
        for fold in range(folds-1,-1,-1):

            X_train_value = np.split(x_train,folds)    # split the x_train into fold value
            Y_train_value = np.split(y_train,folds)    # split the y_train into fold value

            X_Test = X_train_value[fold]       # putting the fold(index) value into train as per intructions
            Y_Test = Y_train_value[fold]

            X_train_value.pop(fold)            # pop the test value from X_train_vaue to get only train data
            Y_train_value.pop(fold)

            X_train = np.concatenate(X_train_value)     # concatenate the train value
            Y_train = np.concatenate(Y_train_value)

            classifier.n_neighbors = K                     # getting the value og the K

            classifier.fit(X_train,Y_train)

            Y_predicted = classifier.predict(X_Test)

            fold_testscores.append(accuracy_score(Y_Test,Y_predicted))

            Y_predicted = classifier.predict(X_train)

            fold_trainscores.append(accuracy_score(Y_train, Y_predicted))

        trainscores.append(np.mean(np.array(fold_trainscores)))
        testscores.append(np.mean(np.array(fold_testscores)))
    return trainscores, testscores
```

In [4]:

```python
import warnings
warnings.filterwarnings("ignore")
neigh = KNeighborsClassifier()
folds = 3
param_range = (1,50)
n = 10
params = random_numbers(param_range,n)
trainscores , testscores = RandomSearch(X_train, y_train, neigh, params, folds)
print("n_neighbors",params)
plt.plot(params, trainscores,label='train curve')
plt.plot(params, testscores,label='test curve')
plt.title("Hyper-parameter VS accuracy plot")
plt.legend()
plt.show()
```

```
n_neighbors [5, 7, 23, 27, 30, 32, 33, 35, 38, 46]
```



In [5]:

```python
# 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 [7]:

```python
from matplotlib.colors import ListedColormap
neigh = KNeighborsClassifier(n_neighbors = 46)
neigh.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, neigh)
```