Experiment the two classifiers using the iris dataset. Use 20% of the data for testing. Evaluate the performance. Decide the value of k based on experimentation.

1)NAÏVE BAYES ALGORITHM USING IRIS DATASET:

In [52]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import math

le = LabelEncoder()

df = pd.read_csv('../iris.csv')

X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

df
```

Out[52]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [53]:

```python
class NaiveBayesClassifier():
    def calc_prior(self, features, target):
        self.prior = (features.groupby(target).apply(lambda x: len(x)) / self.rows).to_numpy
        return self.prior

    def calc_statistics(self, features, target):
        self.mean = features.groupby(target).apply(np.mean).to_numpy()
        self.var = features.groupby(target).apply(np.var).to_numpy()

        return self.mean, self.var

    def gaussian_density(self, class_idx, x):
        mean = self.mean[class_idx]
        var = self.var[class_idx]
        numerator = np.exp((-1/2)*((x-mean)**2) / (2 * var))
#         numerator = np.exp(-((x-mean)**2 / (2 * var)))
        denominator = np.sqrt(2 * np.pi * var)
        prob = numerator / denominator
        return prob

    def calc_posterior(self, x):
        posteriors = []

        # calculate posterior probability for each class
        for i in range(self.count):
            prior = np.log(self.prior[i]) ## use the log to make it more numerically stable
            conditional = np.sum(np.log(self.gaussian_density(i, x)))
            posterior = prior + conditional
            posteriors.append(posterior)
        # return class with highest posterior probability
        return self.classes[np.argmax(posteriors)]
```

```python
    def fit(self, features, target):
        self.classes = np.unique(target)
        self.count = len(self.classes)
        self.feature_nums = features.shape[1]
        self.rows = features.shape[0]

        self.calc_statistics(features, target)
        self.calc_prior(features, target)

    def predict(self, features):
        preds = [self.calc_posterior(f) for f in features.to_numpy()]
        return preds

    def accuracy(self, y_test, y_pred):
        accuracy = np.sum(y_test == y_pred) / len(y_test) * 100
        return accuracy
```

In [54]:

```python
model = NaiveBayesClassifier()

model.fit(X_train, y_train)
predictions = model.predict(X_test)

print('Accuracy = ', model.accuracy(y_test, predictions))
```

```
Accuracy =  90.0
```

In [55]:
```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

y_true = y_test
y_pred = predictions
print('\nConfusion Matrix: \n', confusion_matrix(y_true, y_pred))

# classification report for precision, recall f1-score and accuracy
matrix = classification_report(y_true,y_pred)
print('\nClassification report : \n',matrix)
```

```
Confusion Matrix:
 [[ 9  0  0]
 [ 0  8  1]
 [ 0  2 10]]

Classification report :
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       0.80      0.89      0.84         9
           2       0.91      0.83      0.87        12

    accuracy                           0.90        30
   macro avg       0.90      0.91      0.90        30
weighted avg       0.90      0.90      0.90        30
```

2) KNN ALGORITHM USING IRIS DATASET:

In [94]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from scipy.stats import mode
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt


# K Nearest Neighbors Classification
class K_Nearest_Neighbors_Classifier() :
    def __init__( self, K ) :
        self.K = K

    # Function to store training set
    def fit( self, X_train, Y_train ) :
        self.X_train = X_train
        self.Y_train = Y_train
        # no_of_training_examples, no_of_features
        self.m, self.n = X_train.shape
```

```python
    # Function for prediction
    def predict( self, X_test ) :
        self.X_test = X_test
        # no_of_test_examples, no_of_features
        self.m_test, self.n = X_test.shape
        # initialize Y_predict
        Y_predict = np.zeros( self.m_test )
        for i in range( self.m_test ) :
            x = self.X_test[i]
            # find the K nearest neighbors from current test example
            neighbors = np.zeros( self.K )
            neighbors = self.find_neighbors( x )
            # most frequent class in K neighbors
            Y_predict[i] = mode( neighbors )[0][0]
        return Y_predict
```

```python
    # Function to find the K nearest neighbors to current test example
    def find_neighbors( self, x ) :
        # calculate all the euclidean distances between current
        # test example x and training set X_train
        euclidean_distances = np.zeros( self.m )
        for i in range( self.m ) :
            d = self.euclidean( x, self.X_train[i] )
            euclidean_distances[i] = d
        # sort Y_train according to euclidean_distance_array and
        # store into Y_train_sorted
        inds = euclidean_distances.argsort()
        Y_train_sorted = self.Y_train[inds]
        return Y_train_sorted[:self.K]

    # Function to calculate euclidean distance
    def euclidean( self, x, x_train ) :
        return np.sqrt( np.sum( np.square( x - x_train ) ) )

# Driver code


# Importing dataset
df = pd.read_csv( "../iris.csv" )
X = df.iloc[:,:-1].values
Y = df.iloc[:,-1:].values

Y = le.fit_transform(Y)

# Splitting dataset into train and test set
X_train, X_test, Y_train, Y_test = train_test_split(
X, Y, test_size = 0.2)

neighbors = np.arange(1, 9)
test_accuracy = np.empty(len(neighbors))
train_accuracy = np.empty(len(neighbors))
```

```python
for i, k in enumerate(neighbors):
    model = K_Nearest_Neighbors_Classifier(k)
    model.fit( X_train, Y_train )

    # Prediction on test set
    Y_pred = model.predict( X_test )
    Y_pred1 = model.predict( X_train )

    # measure performance
    correctly_classified = 0
    correctly_classified1 = 0
    count = 0

    for count in range( np.size( Y_pred ) ) :
        if Y_test[count] == Y_pred[count] :
            correctly_classified = correctly_classified + 1
        count = count + 1
    test_accuracy[i] = (correctly_classified / count ) * 100

    count = 0
    for count in range( np.size( Y_pred1 ) ) :
        if Y_train[count] == Y_pred1[count] :
            correctly_classified1 = correctly_classified1 + 1
        count = count + 1

    train_accuracy[i] = (correctly_classified1 / count ) * 100
    print( "\nAccuracy on test set by our model with K =", k, ": ", test_accuracy[i])
    print( "Accuracy on train set by our model with K =", k, ": ", train_accuracy[i])
```

```
Accuracy on test set by our model with K = 1 :  100.0
Accuracy on train set by our model with K = 1 :  100.0

Accuracy on test set by our model with K = 2 :  93.33333333333333
Accuracy on train set by our model with K = 2 :  97.5

Accuracy on test set by our model with K = 3 :  100.0
Accuracy on train set by our model with K = 3 :  95.0

Accuracy on test set by our model with K = 4 :  96.66666666666667
Accuracy on train set by our model with K = 4 :  95.83333333333334

Accuracy on test set by our model with K = 5 :  100.0
Accuracy on train set by our model with K = 5 :  96.66666666666667

Accuracy on test set by our model with K = 6 :  100.0
Accuracy on train set by our model with K = 6 :  96.66666666666667

Accuracy on test set by our model with K = 7 :  96.66666666666667
Accuracy on train set by our model with K = 7 :  97.5

Accuracy on test set by our model with K = 8 :  100.0
Accuracy on train set by our model with K = 8 :  95.83333333333334
```

In [95]:

```
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```