A case study using iris dataset for KNN algorithm

```
# import modules for this project
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# load iris dataset
iris = datasets.load_iris()
data, labels = iris.data, iris.target

# training testing split
res = train_test_split(data, labels,
                       train_size=0.8,
                       test_size=0.2,
                       random_state=12)
train_data, test_data, train_labels, test_labels = res

# Create and fit a nearest-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier
# classifier "out of the box", no parameters
knn = KNeighborsClassifier()
knn.fit(train_data, train_labels)

# print some interested metrics
print("Predictions from the classifier:")
learn_data_predicted = knn.predict(train_data)
print(learn_data_predicted)
print("Target values:")
print(train_labels)
print(accuracy_score(learn_data_predicted, train_labels))

# re-do KNN using some specific parameters.
knn2 = KNeighborsClassifier(algorithm='auto',
                            leaf_size=30,
                            metric='minkowski',
                            p=2,         # p=2 is equivalent to euclidian distance
                            metric_params=None,
                            n_jobs=1,
                            n_neighbors=5,
                            weights='uniform')

knn.fit(train_data, train_labels)
test_data_predicted = knn.predict(test_data)
accuracy_score(test_data_predicted, test_labels)
```

```
Predictions from the classifier:
[0 2 0 1 2 2 2 0 2 0 1 0 0 0 1 2 2 1 0 2 0 1 2 1 0 2 1 1 0 0]
Target values:
[0 2 0 1 2 2 2 0 2 0 1 0 0 0 1 2 2 1 0 1 0 1 2 1 0 2 1 1 0 0]
0.9666666666666667
Predictions from the classifier:
[0 1 2 0 2 0 1 1 0 1 1 0 0 0 0 0 0 0 2 0 2 1 1 1 0 2 1 1 2 0 2 0 2 1 2 2 1
 1 1 2 2 0 2 2 0 1 0 2 2 0 1 1 0 0 1 1 1 1 2 1 2 0 0 1 1 2 0 2 1 0 2 2 1 2
 2 0 0 2 1 1 2 0 1 1 0 1 1 2 2 1 0 2 0 2 0 0 1 2 2 1 2 2 0 1 1 0 2 2 2 1 2
 2 2 0 0 1 0 2 2 1]
Target values:
[0 1 2 0 2 0 1 1 0 1 1 0 0 0 0 0 0 0 2 0 2 1 1 1 0 2 1 1 2 0 2 0 2 2 2 2 1
 1 1 1 2 0 2 2 0 1 0 2 2 0 1 1 0 0 1 1 1 1 2 1 2 0 0 1 1 1 0 2 1 0 2 2 1 2
 2 0 0 2 1 1 2 0 1 1 0 1 1 2 2 1 0 2 0 2 0 0 1 2 2 1 2 2 0 1 1 0 2 2 2 1 2
 2 2 0 0 1 0 2 2 1]
0.975
0.9666666666666667
```

Use this command to help with choice of paramters in the `KNeighborsClassifier` function.

```
help(KNeighborsClassifier)
```

```
Help on class KNeighborsClassifier in module sklearn.neighbors._classification:

class KNeighborsClassifier(sklearn.neighbors._base.KNeighborsMixin, sklearn.base.ClassifierMixin, sklearn.neighbors._base.NeighborsBa
 |  KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=
 |
 |  Classifier implementing the k-nearest neighbors vote.
 |
```

```
|  Read more in the :ref:`User Guide <classification>`.
|
|  Parameters
|  ----------
|  n_neighbors : int, default=5
|      Number of neighbors to use by default for :meth:`kneighbors` queries.
|
|  weights : {'uniform', 'distance'} or callable, default='uniform'
|      Weight function used in prediction.  Possible values:
|
|      - 'uniform' : uniform weights.  All points in each neighborhood
|        are weighted equally.
|      - 'distance' : weight points by the inverse of their distance.
|        in this case, closer neighbors of a query point will have a
|        greater influence than neighbors which are further away.
|      - [callable] : a user-defined function which accepts an
|        array of distances, and returns an array of the same shape
|        containing the weights.
|
|  algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'
|      Algorithm used to compute the nearest neighbors:
|
|      - 'ball_tree' will use :class:`BallTree`
|      - 'kd_tree' will use :class:`KDTree`
|      - 'brute' will use a brute-force search.
|      - 'auto' will attempt to decide the most appropriate algorithm
|        based on the values passed to :meth:`fit` method.
|
|      Note: fitting on sparse input will override the setting of
|      this parameter, using brute force.
|
|  leaf_size : int, default=30
|      Leaf size passed to BallTree or KDTree.  This can affect the
|      speed of the construction and query, as well as the memory
|      required to store the tree.  The optimal value depends on the
|      nature of the problem.
|
|  p : int, default=2
|      Power parameter for the Minkowski metric. When p = 1, this is
|      equivalent to using manhattan_distance (l1), and euclidean_distance
|      (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.
|
|  metric : str or callable, default='minkowski'
|      The distance metric to use for the tree.  The default metric is
|      minkowski, and with p=2 is equivalent to the standard Euclidean
|      metric. For a list of available metrics, see the documentation of
|      :class:`~sklearn.metrics.DistanceMetric`.
|      If metric is "precomputed", X is assumed to be a distance matrix and
|      must be square during fit. X may be a :term:`sparse graph`,
```

Use the following code to generate an artificial dataset which contain three classes. Conduct a similar KNN analysis to the dataset and report your accuracy.

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns  # Import seaborn for the heatmap

# New centers
centers = [[1, 5], [8, 8], [3, 2]]
n_classes = len(centers)

# Create the simulated data
data, labels = make_blobs(n_samples=300,
                          centers=np.array(centers),
                          random_state=1)

# do a 80-20 split of the data
train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, test_size=0.2, random_state=42)

# perform a KNN analysis of the simulated data
```

```
accuracy_scores = []
k_values = range(1, 7)

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(train_data, train_labels)
    predicted_labels = knn.predict(test_data)
    accuracy_scores.append(accuracy_score(test_labels, predicted_labels))

# For the heatmap, reshape the accuracy scores into a 2D array.
accuracy_matrix = np.array(accuracy_scores).reshape(1, len(k_values))

# Plot a heatmap of accuracy scores for different k values
plt.figure(figsize=(10, 6))
sns.heatmap(accuracy_matrix, annot=True, cmap='Blues', cbar=True, xticklabels=k_values, yticklabels=["Accuracy"])
plt.title('HEATMAP for KNN Classifier Accuracy with Diffrent K Values')
plt.xlabel('Number of Neighbors - k')
plt.ylabel('Accuracy')
plt.show()
```