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 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)
```

```
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.utils._metadata_requests._MetadataRequester:
 |
 |  get_metadata_routing(self)
 |      Get metadata routing of this object.
 |
 |      Please check :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      Returns
 |      -------
 |      routing : MetadataRequest
 |          A :class:`~sklearn.utils.metadata_routing.MetadataRequest` encapsulating
 |          routing information.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from sklearn.utils._metadata_requests._MetadataRequester:
 |
 |  __init_subclass__(**kwargs) from abc.ABCMeta
 |      Set the ``set_{method}_request`` methods.
 |
 |      This uses PEP-487 [1]_ to set the ``set_{method}_request`` methods. It
 |      looks for the information available in the set default values which are
 |      set using ``__metadata_request__*`` class attributes, or inferred
 |      from method signatures.
 |
 |      The ``__metadata_request__*`` class attributes are used when a method
 |      does not explicitly accept a metadata through its arguments or if the
 |      developer would like to specify a request value for those metadata
 |      which are different from the default ``None``.
 |
 |      References
 |      ----------
 |      .. [1] https://www.python.org/dev/peps/pep-0487
```

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

centers = [[2, 4], [6, 6], [1, 9]]
n_classes = len(centers)
data, labels = make_blobs(n_samples=150,
                          centers=np.array(centers),
                          random_state=1)
# do a 80-20 split of the data
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 matplotlib.pyplot as plt
import numpy as np

# Generate synthetic data
centers = [[2, 4], [6, 6], [1, 9]]
data, labels = make_blobs(n_samples=150,
                          centers=np.array(centers),
                          random_state=1)

# Perform the 80-20 split
train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels,
    train_size=0.8,
    test_size=0.2,
    random_state=42
)
# perform a KNN analysis of the simulated data
```

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

# Create and fit a KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)  # Using 5 neighbors
knn.fit(train_data, train_labels)

# Predict on the test data
test_data_predicted = knn.predict(test_data)

# output accuracy score

# Calculate and output accuracy
accuracy = accuracy_score(test_labels, test_data_predicted)
print(f"Test Accuracy Score: {accuracy:.2f}")

# plot your different results

# Function to plot decision boundaries
def plot_decision_boundaries(X, y, model, title):
    h = .02  # Step size in the mesh
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.8, cmap='viridis')
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', s=50, cmap='viridis')
    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

# Plot decision boundaries and data points for training data
plot_decision_boundaries(train_data, train_labels, knn, 'KNN Decision Boundaries (Training Data)')

# Plot decision boundaries and data points for test data
plot_decision_boundaries(test_data, test_labels, knn, 'KNN Decision Boundaries (Test Data)')
```

```
Test Accuracy Score: 1.00
```