Experiment - 7

Objective:
Learn implementation of k-nearest neighbor in python

1. Importing the modules

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

2. Creating Dataset

Scikit-learn has a lot of tools for creating synthetic datasets, which are great for testing machine learning algorithms. I'm going to utilize the make blobs method.

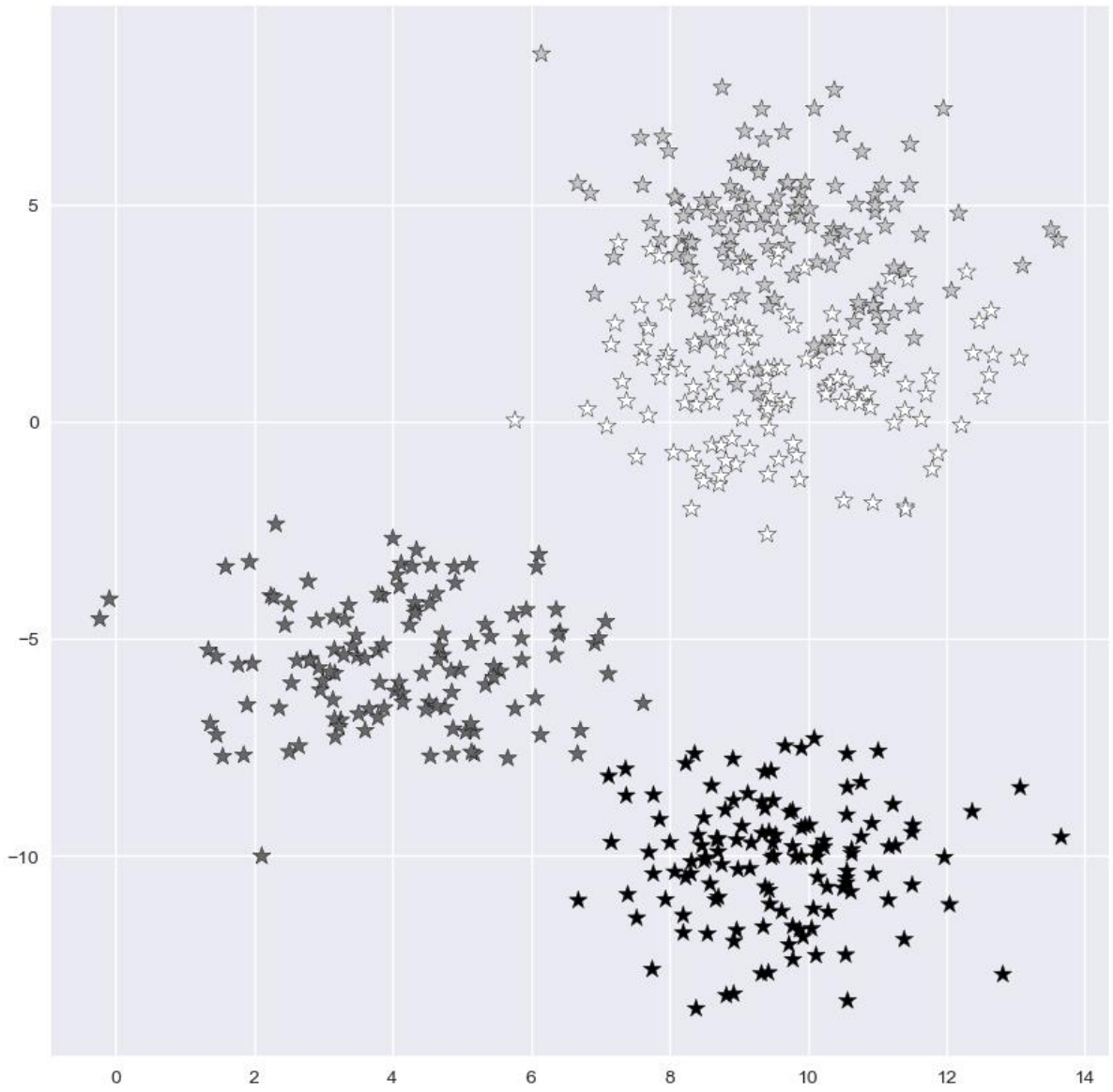
```
X, y = make_blobs(n_samples = 500, n_features = 2, centers =
4,cluster_std = 1.5, random_state = 4)
```

This code generates a dataset of 500 samples separated into four classes with a total of two characteristics. Using associated parameters, you may quickly change the number of samples, characteristics, and classes. We may also change the distribution of each cluster (or class).

3. Visualize the Dataset

```
plt.style.use('seaborn')
plt.figure(figsize = (10,10))
plt.scatter(X[:,0], X[:,1], c=y, marker=
'*',s=100,edgecolors='black')
plt.show()
```

Output:



4. Splitting Data into Training and Testing Datasets

It is critical to partition a dataset into train and test sets for every supervised machine learning method. We first train the model and then put it to the test on various portions of the dataset. If we don't separate the data, we're simply testing the model with data it already knows. Using the train_test_split method, we can simply separate the tests.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state = 0)
```

With the train size and test size options, we may determine how much of the original data is utilized for train and test sets, respectively. The default separation is 75% for the train set and 25% for the test set.

5. KNN Classifier Implementation

After that, we'll build a kNN classifier object. I develop two classifiers with k values of 1 and 5 to demonstrate the relevance of the k value. The models are

then trained using a train set. The k value is chosen using the n_neighbors argument. It does not need to be explicitly specified because the default value is 5.

```
knn5 = KNeighborsClassifier(n_neighbors = 5)
knn1 = KNeighborsClassifier(n_neighbors=1)
```

6. Predictions for the KNN Classifiers

Then, in the test set, we forecast the target values and compare them to the actual values.

```
knn5.fit(X_train, y_train)
knn1.fit(X_train, y_train)
y_pred_5 = knn5.predict(X_test)
y_pred_1 = knn1.predict(X_test)
```

7. Predict Accuracy for both k values

```
from sklearn.metrics import accuracy_score
print("Accuracy with k=5", accuracy_score(y_test,
y_pred_5)*100)
print("Accuracy with k=1", accuracy_score(y_test,
y_pred_1)*100)
```

Output:

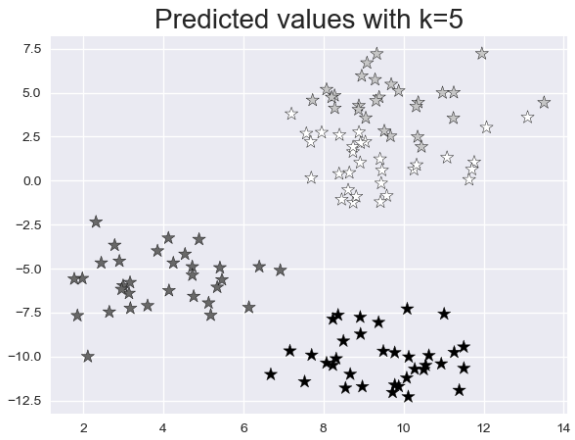
```
Accuracy with k=5 93.60000000000001
Accuracy with k=1 90.4
```

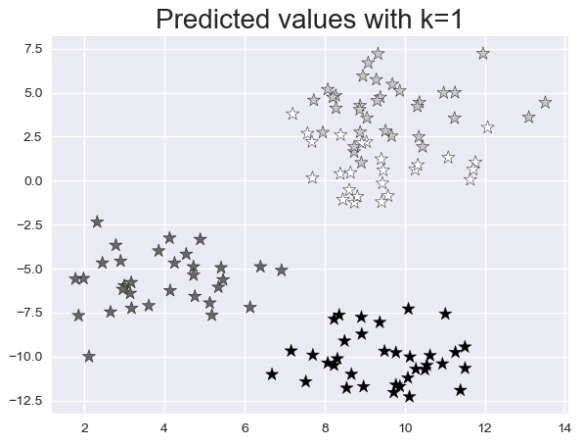
8. Visualize Predictions

Let's view the test set and predicted values with k=5 and k=1 to see the influence of k values.

```
plt.figure(figsize = (15,5))
plt.subplot(1,2,1)
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_5, marker= '*',
s=100,edgecolors='black')
plt.title("Predicted values with k=5", fontsize=20)
plt.subplot(1,2,2)
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_1, marker= '*',
s=100,edgecolors='black')
plt.title("Predicted values with k=1", fontsize=20)
plt.show()
```

Output:





Conclusion

Hopefully, you now have a better understanding of the KNN algorithm. We've looked at a variety of ideas for how KNN saves the complete dataset in order to generate predictions. KNN is one of several lazy learning algorithms that don't use a learning model to make predictions. By averaging the similarity between an incoming observation and the data already available, KNN creates predictions on the fly (just in time).