Hierarchical Clustering and Dendrograms in Python

A Jupyter Notebook Example

Introduction

This presentation serves as a walk-through and detailed explanation of the provided Example Jupyter Notebook covering clustering and dendrograms in Python and Jupyter Notebook.
This Notebook accesses the provided dataset of cereal information and uses this information to create two dendrograms.
Here, we will step through the example code and explain each block's function.

Part 1 - Dendrograms

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.cluster.hierarchy
    import linkage, dendrogram
import sqlite3
```

These four lines import the required libraries. The *matplotlib* library is required to display the plot in the Jupyter Notebook environment. *numpy* is used for handling data. The *scipy* library builds on *numpy* and provides the functions for building and displaying the dendrogram. Lastly, *sqlite3* provides an interface to the database.

Dendrograms Cont.

```
db_filename = 'cereals.db'
conn = sqlite3.connect(
      db_filename)
c = conn.cursor()
```

These three lines establishes a connection to the sql database that contains the cereal information that will go into the dendrogram. In this case the database is named *cereals.db*. In other examples, we have used csv files instead of databases.

Dendrograms Cont.

```
c.execute("SELECT
      Protein, Fat, Carbohydrates
      FROM cereals")
results = sum(map(list,
      list(c.fetchall())), [])
X = np.matrix(results).
      reshape(-1,3)
```

These three lines are responsible for collecting data from the database and packaging it into an appropriate format for scipy. The first line executes the query and collects the protein, fat, and carbohydrate content for each cereal. The second line converts the resultant data from the tuple format returned by sqlite3 to a numpy matrix. The third line transposes the matrix into the correct format.

Dendrograms Cont.

```
links = linkage(X, 'centroid')
```

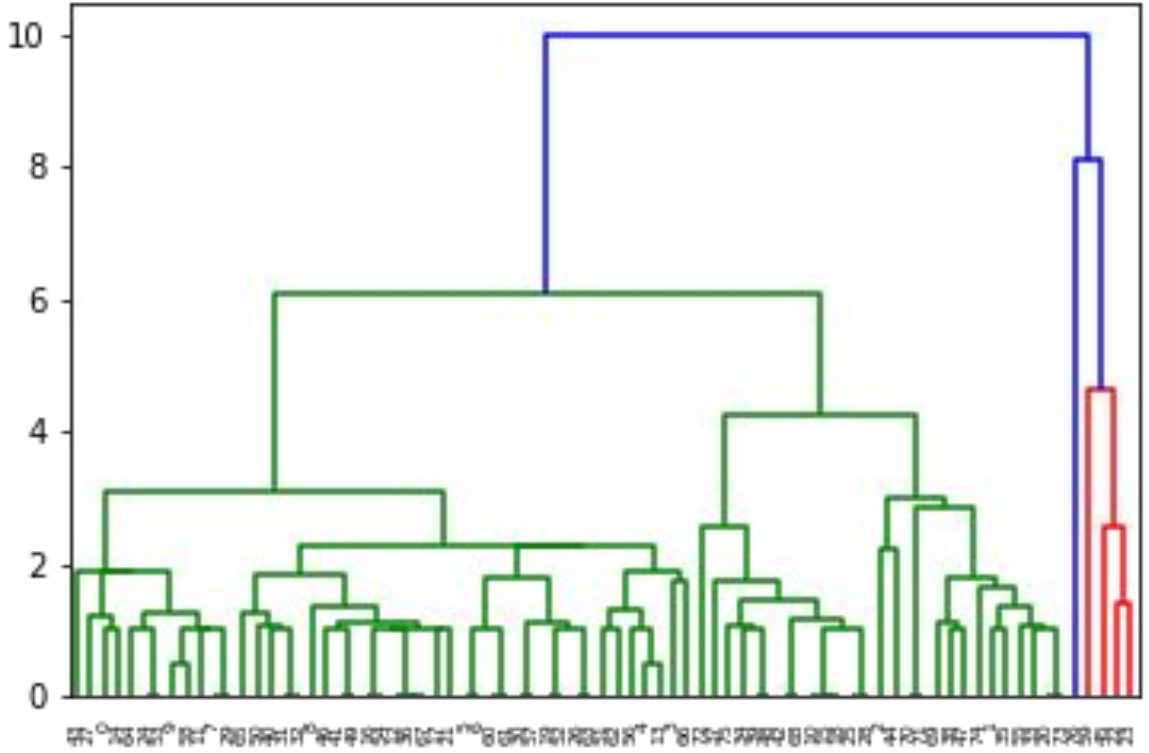
This single line is responsible for computing both the clustering that informs the dendrogram and the dendrogram's links. The function accepts the data, X, and a specifier for the distance metric to use when computing cluster distances. In this case, we are using 'centroid', which computes the centroid of each cluster and then the distance between them when merging clusters.

Dendrograms Cont.

```
dendrogram(links)
plt.show()
```

These final two lines construct the visualization of the dendrogram and then show the plot in the Notebook.

Resultant Plot



Part 2 - Normalizing Data

Previously, we have mentioned how important it is to normalize data when performing clustering. It is important to note that we did not normalize the data in the previous example. Next, we will normalize the same data and then re-draw the dendrogram.

Normalizing Cont.

```
from scipy.cluster.vw
      import whiten

Y = whiten(X)

links = linkage(Y, 'centroid')
```

Using Jupyter Notebook's ability to maintain the state of the kernel between cells, we can pick up directly where we left off in the previous cell. To normalize the data, we import the 'whiten' function from scipy, then apply it to the transposed data matrix. This normalizes the feature values to the range [0, 1]. Lastly, we recompute the dendrogram using the normalized feature values.

Normalizing Cont.

```
dendrogram(links)
plt.show()
```

Again, these final lines render and show the dendrogram previously computed by the linkage function.

Resultant Plot