Learning Multidimensional Scaling (MDS) with Python

Authored by

Mohammed looti

October 27, 2025

Understanding Multidimensional Scaling (MDS)

In the realm of statistics and data analysis, **multidimensional scaling** (MDS) is a powerful technique designed to visualize the similarity or dissimilarity of observations within a dataset. It achieves this by representing complex relationships in a simplified, low-dimensional cartesian space, typically a 2-D plot, making it easier to identify patterns and clusters that might otherwise be hidden in high-dimensional data.

The primary goal of MDS is to arrange items in a space such that the distances between them in this lower-dimensional representation reflect their original dissimilarities as accurately as possible. This transformation allows researchers and analysts to gain intuitive insights into the structure of their data, observing which items are similar and which are different at a glance.

For those working with Python, performing multidimensional scaling is streamlined using the **MDS() function**, which is part of the **sklearn.manifold** submodule in the scikit-learn library. This function provides a robust and efficient way to apply MDS algorithms to your data.

Setting Up Your Python Environment and Data

To illustrate the practical application of MDS in Python, let's consider a practical example. We will begin by creating a pandas DataFrame that stores performance statistics for several basketball players. This DataFrame will serve as our high-dimensional dataset, where each player is an observation characterized by multiple attributes like points, assists, blocks, and rebounds.

The following code snippet demonstrates how to construct this sample DataFrame using the Python pandas library:

```
import pandas as pd

#create DataFrane
df = pd.DataFrame({'player': ,
'points': ,
'assists': ,
'blocks': ,
'rebounds': })

#set player column as index column
df = df.set_index('player')

#view Dataframe
print(df)
```

points assists blocks rebounds
player
A 4 3 7 4
B 4 2 3 5
C 6 2 6 5
D 7 5 7 6
E 8 4 5 5
F 14 8 8 8
G 16 7 8 10
H 19 6 4 4
I 25 8 2 3
J 25 10 2 2
K 28 11 1 2

Implementing Multidimensional Scaling in Python

With our DataFrame prepared, we can now proceed to apply multidimensional scaling. We will leverage the **MDS() function** from sklearn.manifold. This function takes our high-dimensional data and attempts to find a lower-dimensional embedding while preserving the original distances between data points as much as possible.

The following Python code demonstrates how to instantiate the **MDS()** object and then use its `fit_transform()` method to perform the scaling. The `random_state` parameter is included to ensure reproducibility of the results.

from sklearn.manifold import MDS

#perform multi-dimensional scaling
mds = MDS(random_state=0)
scaled_df = mds.fit_transform(df)

#view results of multi-dimensional scaling
print(scaled_df)

]

The output, `scaled_df`, is a NumPy array where each row corresponds to a player from our original DataFrame. Importantly, each player's complex statistical profile has been reduced to a simple (x, y) coordinate in a 2-D cartesian space. These coordinates are now ready for visualization, allowing us to interpret the underlying similarities between players.

Visualizing MDS Results for Better Insights

To truly understand the relationships revealed by MDS, visualizing the transformed coordinates is essential. We can easily plot these 2-D points using matplotlib.pyplot, a widely used plotting library in Python. The resulting scatterplot will graphically represent the similarity of players based on their original statistics.

The following Python code generates a scatterplot of the coordinates, adding labels for each player to facilitate identification and analysis:

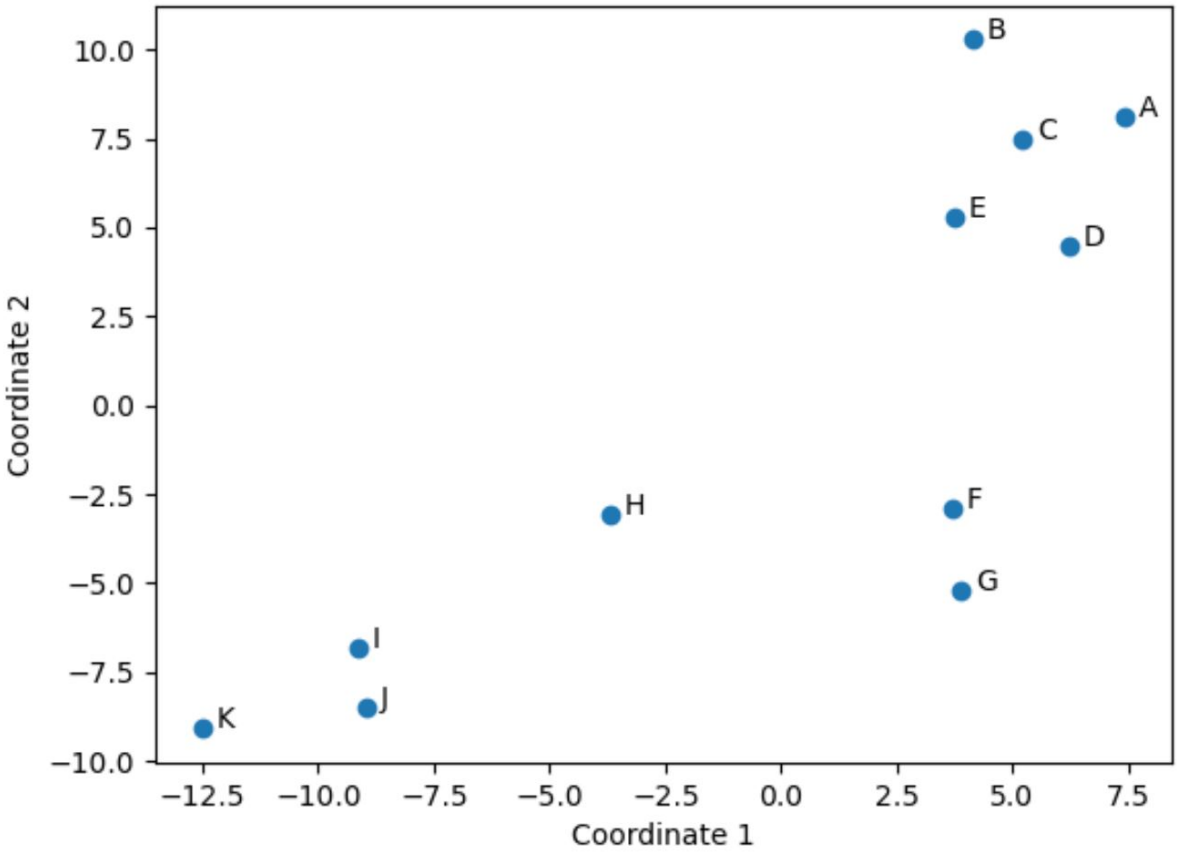
```
import matplotlib.pyplot as plt

#create scatterplot
plt.scatter(scaled_df, scaled_df)

#add axis labels
plt.xlabel('Coordinate 1')
plt.ylabel('Coordinate 2')

#add lables to each point
for i, txt in enumerate(df.index):
plt.annotate(txt, (scaled_df+.3, scaled_df))

#display scatterplot
plt.show()
```



This scatterplot provides a visual representation where proximity between points indicates similarity. Specifically, players from the original DataFrame who exhibit similar statistical values across the four original columns (points, assists, blocks, and rebounds) will appear close to each other in this 2-D plot. Conversely, players with vastly different statistics will be positioned further apart.

Deep Dive into MDS Interpretation: Player Similarities and Differences

The true power of multidimensional scaling lies in its ability to translate complex data relationships into an easily interpretable visual format. Let's examine specific examples from our basketball player dataset to understand how the scatterplot reflects underlying similarities and dissimilarities.

Consider players **F** and **G** in the plot. Visually, they are located in close proximity, suggesting a high degree of similarity in their playing styles based on the given statistics. To confirm this, let's review their original values from the DataFrame:

#select rows with index labels 'F' and 'G'
df.loc]

points assists blocks rebounds

player
F 14 8 8 8
G 16 7 8 10

As evident from the table, players F and G indeed have remarkably similar values across all four statistical categories. Their points (14 vs 16), assists (8 vs 7), blocks (8 vs 8), and rebounds (8 vs 10) are all very close. This statistical closeness is accurately represented by their minimal Euclidean distance in the 2-D MDS plot.

In stark contrast, let's examine players **B** and **K**. In the scatterplot, these two players are positioned at a significant distance from each other, indicating substantial differences in their performance profiles. Their original statistics paint a clear picture:

#select rows with index labels 'B' and 'K'
df.loc]

points assists blocks rebounds
player
B 4 2 3 5
K 28 11 1 2

The disparity between B and K is striking. Player K excels in points and assists, while player B has lower values in these categories but higher blocks and rebounds. This wide divergence in their statistical attributes directly translates to their distant placement in the 2-D plot, effectively visualizing their dissimilar player archetypes.

Ultimately, the 2-D plot generated by MDS serves as an intuitive and effective tool for quickly assessing the similarity between players based on all available variables in the DataFrame. It allows for immediate identification of clusters of similar observations and highlights outliers or unique entities within the dataset.

Conclusion and Further Exploration

This tutorial has demonstrated how to perform and interpret **multidimensional scaling** in Python using scikit-learn's **MDS() function**. By transforming high-dimensional data into an easily digestible 2-D visualization, MDS empowers analysts to uncover hidden patterns and relationships within their datasets. This technique is invaluable across various fields, from social sciences to bioinformatics, for exploratory data analysis and hypothesis generation.

For those interested in exploring more advanced data analysis and visualization techniques in

Python, consider delving into tutorials on topics such as Principal Component Analysis (PCA), t-SNE, or various clustering algorithms. These methods offer alternative ways to reduce dimensionality, identify groups, and gain deeper insights from complex data.