Jorge Martín Lasaosa Follow
May 29, 2021 · 10 min read · ✦ · Listen

HANDS-ON TUTORIALS

Clustering on numerical and categorical features.

Using Gower Distance in Python.

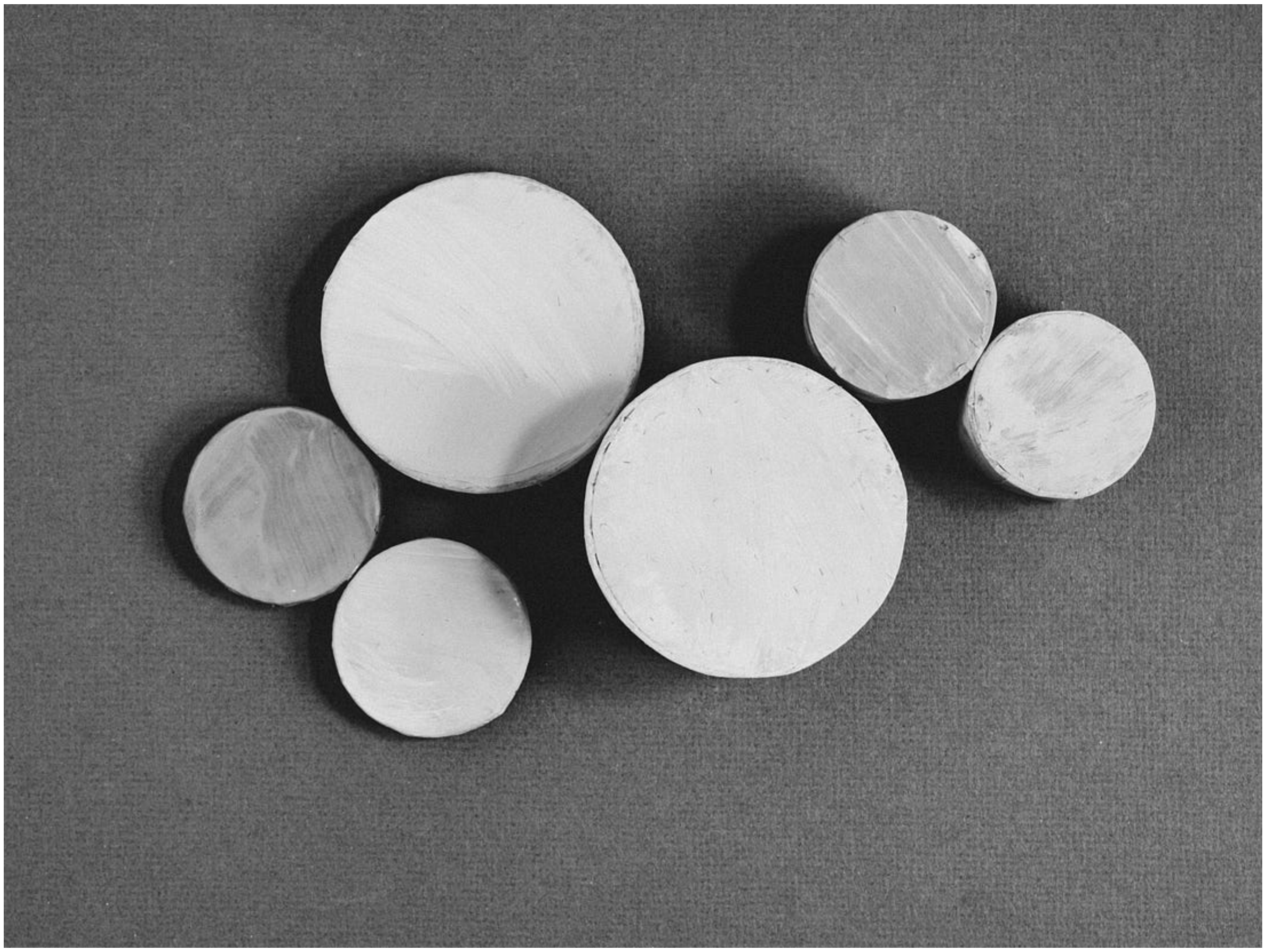


Photo by Munro Studio on Unsplash

Introduction

During the last year, I have been working on projects related to Customer Experience (CX). In these projects, Machine Learning (ML) and data analysis techniques are carried out on customer data to improve the company's knowledge of its customers. Recently, I have focused my efforts on finding different groups of customers that share certain characteristics to be able to perform specific actions on them.

As you may have already guessed, the project was carried out by performing **clustering.** For those unfamiliar with this concept, clustering is the task of dividing a set of objects or observations (e.g., customers) into different groups (called clusters) based on their features or properties (e.g., gender, age, purchasing trends). The division should be done in such a way that the observations are as similar as possible to each other within the same cluster. In addition, each cluster should be as far away from the others as possible. [1]

One of the main challenges was to find a way to perform clustering algorithms on data that had both categorical and numerical variables. In the real world (and especially in CX) a lot of information is stored in categorical variables. Although there is a huge amount of information on the web about clustering with numerical variables, it is difficult to find information about mixed data types.

This is a complex task and there is a lot of controversy about whether it is appropriate to use this mix of data types in conjunction with clustering algorithms. However, I decided to take the plunge and do my best. For the remainder of this blog, I will share my personal experience and what I have learned.

In the next sections, we will see what the Gower distance is, with which clustering algorithms it is convenient to use, and an example of its use in Python.

Disclaimer: I consider myself a data science newbie, so this post is not about creating a single and magical guide that everyone should use, but about sharing the knowledge I have gained. My main interest nowadays is to keep learning, so I am open to criticism and corrections. Feel free to share your thoughts in the comments section!

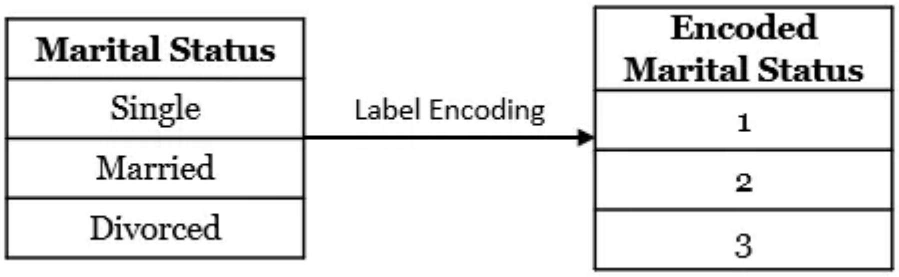
Gower Distance

Now that we understand the meaning of clustering, I would like to highlight the following sentence mentioned above.

“*The division should be done in such a way that the observations are* ***as similar as possible*** *to each other within the same cluster*”.

How can we define **similarity** between different customers? In the case of having only numerical features, the solution seems intuitive, since we can all understand that a 55-year-old customer is more similar to a 45-year-old than to a 25-year-old. There are many ways to measure these distances, although this information is beyond the scope of this post.

But, what if we not only have information about their age but also about their *marital status* (e.g. single, married, divorced…)? Such a categorical feature could be transformed into a numerical feature by using techniques such as imputation, label encoding, one-hot encoding… However, these transformations can lead the clustering algorithms to misunderstand these features and create meaningless clusters. For example, if we were to use the label encoding technique on the *marital status* feature, we would obtain the following encoded feature:



Label Encoding technique applied to Marital Status feature

The problem with this transformation is that the clustering algorithm might understand that a *Single* value is more similar to *Married* (*Married*[2]–*Single*[1]=1) than to *Divorced* (*Divorced*[3]–*Single*[1]=2). As shown, transforming the features may not be the best approach.

And here is where Gower distance (measuring similarity or dissimilarity) comes into play. **Gower Similarity (GS)** was first defined by J. C. Gower in 1971 [2]. To calculate the similarity between observations *i* and *j* (e.g., two customers), *GS* is computed as the average of partial similarities (*ps*) across the *m* features of the observation.

$$GS_{ij} = \frac{1}{m}\sum_{f=1}^{m} ps_{ij}^{(f)}$$

Similarity between observations i and j. Having each observation m different features, either numerical, categorical or mixed.

Partial similarities calculation depends on the type of the feature being compared.

- For a **numerical feature**, the partial similarity between two individuals *i* and *j* is one minus the subtraction between their values in the specific feature (in absolute value) divided by the total range of the feature.

$$ps_{ij}^{(f)} = 1 - \frac{|x_{if} - x_{jf}|}{R_f}$$

Similarity between observation i and j in feature f when f is numerical.

$$R_f = \max f - \min f$$

Range of a feature f

- For a **categorical** feature, the partial similarity between two individuals is one only when both observations have exactly the same value for this feature. Zero otherwise.

Partial similarities always range from 0 to 1. So, when we compute the average of the partial similarities to calculate the GS we always have a result that varies from zero to one. Zero means that the observations are as different as possible, and one means that they are completely equal.

Thanks to these findings we can measure the degree of similarity between two observations when there is a mixture of categorical and numerical variables. However, before going into detail, we must be cautious and take into account certain aspects that may compromise the use of this distance in conjunction with clustering algorithms.

Mathematical properties of Gower Distance

There are two questions on Cross-Validated that I highly recommend reading:

- Gower's Dissimilarity Index
- Hierarchical clustering with mixed type data — what distance/similarity to use?

Both define **Gower Similarity** (GS) **as non-Euclidean and non-metric. Gower Dissimilarity** (GD = 1 — GS) has the same limitations as GS, so it **is also non-Euclidean and non-metric**. Nevertheless, Gower Dissimilarity defined as √GD is actually a Euclidean distance (therefore metric, automatically) when no specially processed ordinal variables are used (if you are interested in this you should take a look at how Podani extended Gower to ordinal characters)

This is important because if we use GS or GD, we are using a distance that is not obeying the Euclidean geometry. Thus, methods based on Euclidean distance must not be used, as some clustering methods:

- K-means
- Ward's, centroid, median methods of hierarchical clustering
- …

Gower Distance in Programming Languages

Related work in R or Python

Now, can we use this measure in R or Python to perform clustering? Regarding R, I have found a series of very useful posts that teach you how to use this distance measure through a function called *daisy:*

- Clustering on mixed type data: A proposed approach using R.
- Clustering categorical and numerical datatype using Gower Distance.
- Hierarchical Clustering on Categorical Data in R (only with categorical features).

However, I haven't found a specific guide to implement it in Python. That's why I decided to write this blog and try to bring something new to the community. Forgive me if there is currently a specific blog that I missed.

Gower Distance in Python

First of all, it is important to say that for the moment we cannot natively include this distance measure in the clustering algorithms offered by scikit-learn. This is an open issue on scikit-learn's GitHub since 2015. However, since 2017 a group of community members led by Marcelo Beckmann have been working on the implementation of the Gower distance. Hopefully, it will soon be available for use within the library.

During this process, another developer called Michael Yan apparently used Marcelo Beckmann's code to create a non scikit-learn package called *gower* that can already be used, without waiting for the costly and necessary validation processes of the scikit-learn community. **Note that this implementation uses Gower Dissimilarity (GD).**

Hands-on Implementation

Data

When I learn about new algorithms or methods, I really like to see the results in very small datasets where I can focus on the details. So for the implementation, we are going to use a small synthetic dataset containing made-up information about customers of a grocery shop.

```
import pandas as pd

# Creating a dictionary with the data
dictionary = {"age": [22, 25, 30, 38, 42, 47, 55, 62, 61, 90],
              "gender": ["M", "M", "F", "F", "F", "M", "M", "M", "M", "M"],
              "civil_status": ["SINGLE", "SINGLE", "SINGLE", "MARRIED", "MARRIED", "SINGLE", "M
              "salary": [18000, 23000, 27000, 32000, 34000, 20000, 40000, 42000, 25000, 70000],
              "has_children": [False, False, False, True, True, False, False, False, False, Tru
              "purchaser_type": ["LOW_PURCHASER", "LOW_PURCHASER", "LOW_PURCHASER", "HEAVY_PURC

# Creating a Pandas DataFrame from the dictionary
dataframe = pd.DataFrame.from_dict(dictionary)
```

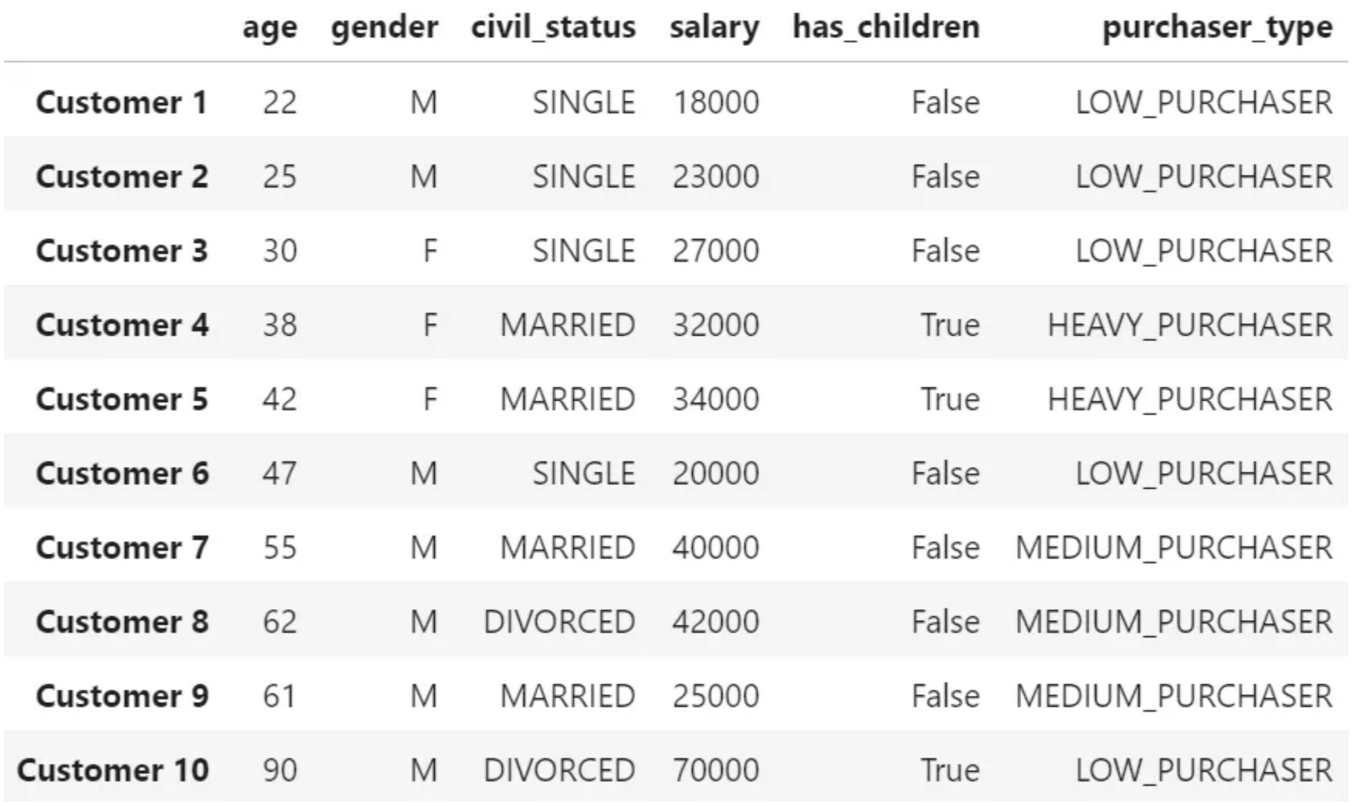
GowerImplementation.py hosted with ❤ by GitHub view raw

Python code for creating the Pandas DataFrame

The data created have 10 customers and 6 features:

- **Age:** Numerical
- **Gender:** Categorical
- **Civil Status:** Categorical
- **Salary:** Numerical
- **Does the client have children?:** Binary
- **Purchaser Type:** Categorical

All of the information can be seen below:



	age	gender	civil_status	salary	has_children	purchaser_type
Customer 1	22	M	SINGLE	18000	False	LOW_PURCHASER
Customer 2	25	M	SINGLE	23000	False	LOW_PURCHASER
Customer 3	30	F	SINGLE	27000	False	LOW_PURCHASER
Customer 4	38	F	MARRIED	32000	True	HEAVY_PURCHASER
Customer 5	42	F	MARRIED	34000	True	HEAVY_PURCHASER
Customer 6	47	M	SINGLE	20000	False	LOW_PURCHASER
Customer 7	55	M	MARRIED	40000	False	MEDIUM_PURCHASER
Customer 8	62	M	DIVORCED	42000	False	MEDIUM_PURCHASER
Customer 9	61	M	MARRIED	25000	False	MEDIUM_PURCHASER
Customer 10	90	M	DIVORCED	70000	True	LOW_PURCHASER

Customers synthetic data

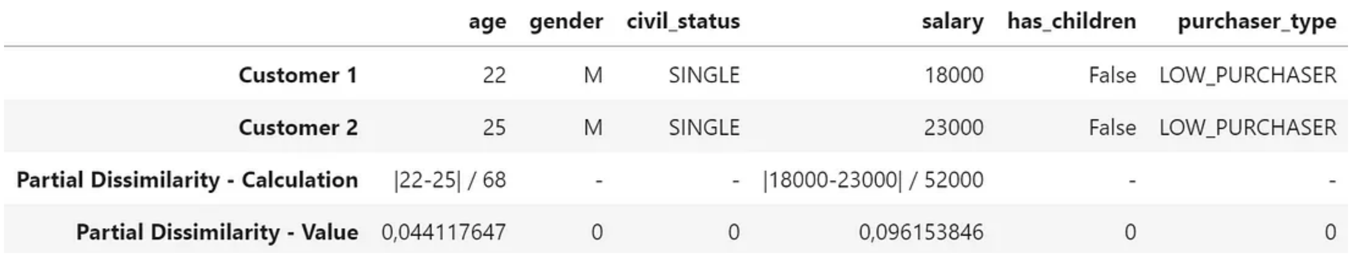
Gower Distances

Now, it is time to use the *gower package* mentioned before to calculate all of the distances between the different customers. Let's do the first one manually, and **remember that this package is calculating the Gower Dissimilarity (DS).** So the way to calculate it changes a bit.

- For a **numerical** feature, the partial dissimilarity between two customers *i* and *j* is the subtraction between their values in the specific feature(in absolute value) divided by the total range of the feature. The range of *salary* is 52000 (70000–18000) while the range of *age* is 68 (90–22). Note the **importance of not having outliers** in these features. An extremely large or small erroneous value would directly affect the range, and therefore the differences in that feature would become much less important than they are.

- For a **categorical** feature, the partial dissimilarity between two customers is one when both customers have a different value for this feature. Zero otherwise.

Following this procedure, we then calculate all partial dissimilarities for the first two customers.



	age	gender	civil_status	salary	has_children	purchaser_type
Customer 1	22	M	SINGLE	18000	False	LOW_PURCHASER
Customer 2	25	M	SINGLE	23000	False	LOW_PURCHASER
Partial Dissimilarity - Calculation	|22-25| / 68	-	-	|18000-23000| / 52000	-	-
Partial Dissimilarity - Value	0,044117647	0	0	0,096153846	0	0

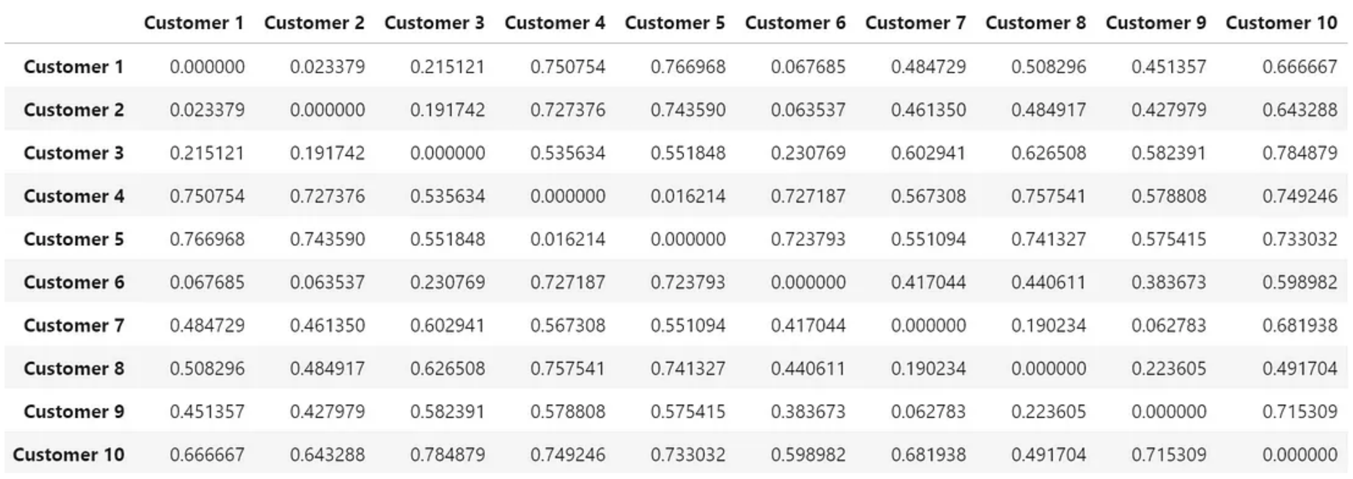
Calculating the partial dissimilarities for the first two customers

The Gower Dissimilarity between both customers is the average of partial dissimilarities along the different features: (0.044118 + 0 + 0 + 0.096154 + 0 + 0) / 6 =0.023379. As the value is close to zero, we can say that both customers are very similar.

Let's use *gower package* to calculate all of the dissimilarities between the customers. Then, store the results in a matrix:

```python
import gower

distance_matrix = gower.gower_matrix(dataframe)
```

1_DistanceMatrix.py hosted with ❤ by GitHub view raw



	Customer 1	Customer 2	Customer 3	Customer 4	Customer 5	Customer 6	Customer 7	Customer 8	Customer 9	Customer 10
Customer 1	0.000000	0.023379	0.215121	0.750754	0.766968	0.067685	0.484729	0.508296	0.451357	0.666667
Customer 2	0.023379	0.000000	0.191742	0.727376	0.743590	0.063537	0.461350	0.484917	0.427979	0.643288
Customer 3	0.215121	0.191742	0.000000	0.535634	0.551848	0.230769	0.602941	0.626508	0.582391	0.784879
Customer 4	0.750754	0.727376	0.535634	0.000000	0.016214	0.727187	0.567308	0.757541	0.578808	0.749246
Customer 5	0.766968	0.743590	0.551848	0.016214	0.000000	0.723793	0.551094	0.741327	0.575415	0.733032
Customer 6	0.067685	0.063537	0.230769	0.727187	0.723793	0.000000	0.417044	0.440611	0.383673	0.598982
Customer 7	0.484729	0.461350	0.602941	0.567308	0.551094	0.417044	0.000000	0.190234	0.062783	0.681938
Customer 8	0.508296	0.484917	0.626508	0.757541	0.741327	0.440611	0.190234	0.000000	0.223605	0.491704
Customer 9	0.451357	0.427979	0.582391	0.578808	0.575415	0.383673	0.062783	0.223605	0.000000	0.715309
Customer 10	0.666667	0.643288	0.784879	0.749246	0.733032	0.598982	0.681938	0.491704	0.715309	0.000000

Distance matrix

We can interpret the matrix as follows. In the first column, we see the dissimilarity of the first customer with all the others. This customer is similar to the second, third and sixth customer, due to the low GD.

Perform clustering on the distance matrix

The matrix we have just seen can be used in almost any scikit-learn clustering algorithm. However, we must remember the limitations that the Gower distance has due to the fact that it is neither Euclidean nor metric.

To use Gower in a scikit-learn clustering algorithm, we must look in the documentation of the selected method for the option to pass the distance matrix directly. Although the name of the parameter can change depending on the algorithm, we should almost always put the value *precomputed,* so I recommend going to the documentation of the algorithm and look for this word.

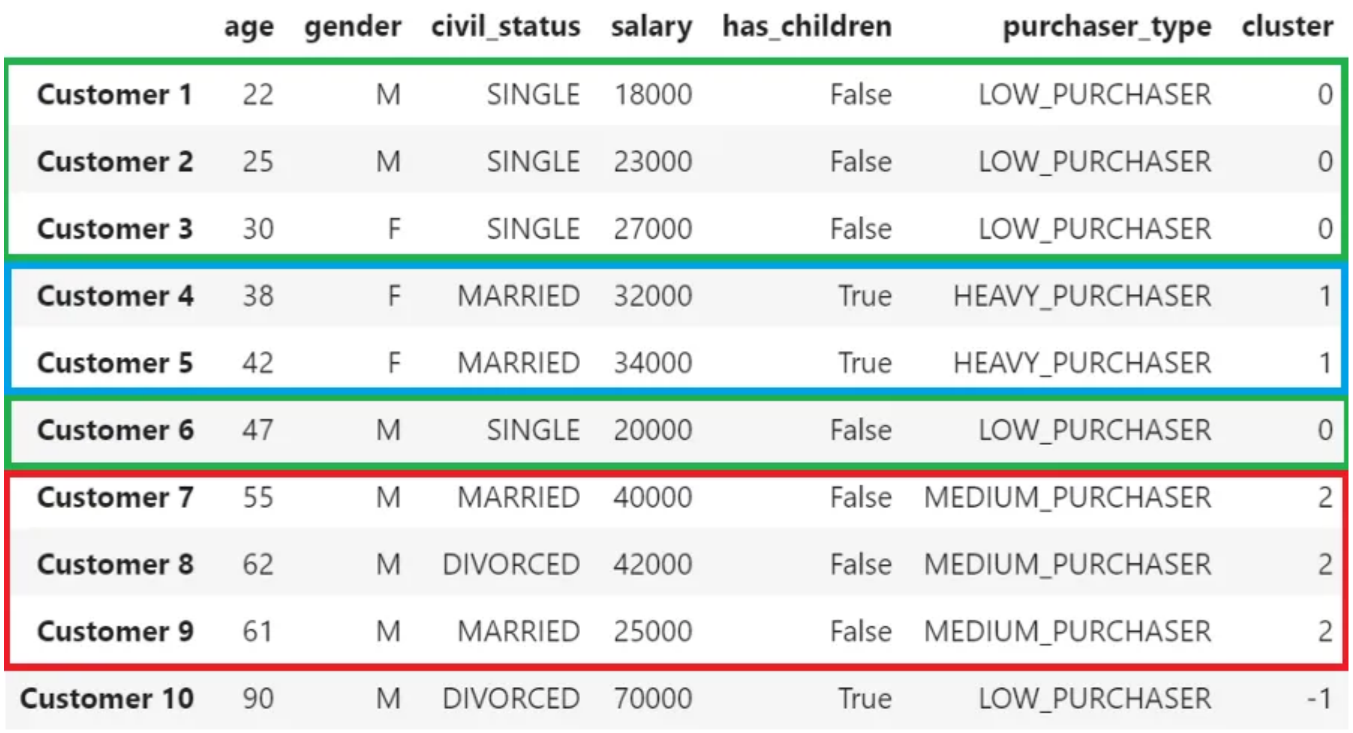
In this post, we will use the **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) algorithm. Understanding the algorithm is beyond the scope of this post, so we won't go into details. I leave here the link to the theory behind the algorithm and a gif that visually explains its basic functioning.

Here we have the code where we define the clustering algorithm and configure it so that the metric to be used is "*precomputed*". When we fit the algorithm, instead of introducing the dataset with our data, we will introduce the matrix of distances that we have calculated. In addition, we add the results of the cluster to the original data to be able to interpret the results.

```python
from sklearn.cluster import DBSCAN

# Configuring the parameters of the clustering algorithm
dbscan_cluster = DBSCAN(eps=0.3,
                        min_samples=2,
                        metric="precomputed")

# Fitting the clustering algorithm
dbscan_cluster.fit(distance_matrix)

# Adding the results to a new column in the dataframe
dataframe["cluster"] = dbscan_cluster.labels
```

DBSCAN_1.py hosted with ❤ by GitHub view raw

The final results can be seen below:



	age	gender	civil_status	salary	has_children	purchaser_type	cluster
Customer 1	22	M	SINGLE	18000	False	LOW_PURCHASER	0
Customer 2	25	M	SINGLE	23000	False	LOW_PURCHASER	0
Customer 3	30	F	SINGLE	27000	False	LOW_PURCHASER	0
Customer 4	38	F	MARRIED	32000	True	HEAVY_PURCHASER	1
Customer 5	42	F	MARRIED	34000	True	HEAVY_PURCHASER	1
Customer 6	47	M	SINGLE	20000	False	LOW_PURCHASER	0
Customer 7	55	M	MARRIED	40000	False	MEDIUM_PURCHASER	2
Customer 8	62	M	DIVORCED	42000	False	MEDIUM_PURCHASER	2
Customer 9	61	M	MARRIED	25000	False	MEDIUM_PURCHASER	2
Customer 10	90	M	DIVORCED	70000	True	LOW_PURCHASER	-1

If we analyze the different clusters we have:

- **Cluster 0 (Green):** Customers with a salary between 18,000 and 27,000, who do not have children and purchase infrequently.
- **Cluster 1 (Blue):** Mothers in their 40s who earn around 33,000 and purchase a

lot.

- **Cluster 2 (Red):** Men in their 60s, moderate purchasers with no children
- **Cluster -1:** This is not a cluster itself, but the customer that the algorithm has identified as an outlier. It cannot be identified in any cluster.

Mixed Data Types Python Cluster Analysis Editors Pick Hands On Tutorials

ers are divided. Thus, we could carry out specific actions on them, such as personalized advertising campaigns, offers aimed at specific groups…It is true that this example is very small and set up for having a successful clustering, real projects are much more complex and time-consur ficant results.

458 10

Conclusion

This post proposes a methodology to perform clustering with the Gower distance in Python. It also exposes the limitations of the distance measure itself so that it can be used properly. Finally, the small example confirms that clustering developed in this way makes sense and could provide us with a lot of information.

I hope you find the methodology useful and that you found the post easy to read. And above all, I am happy to receive any kind of feedback. So feel free to share your thoughts!

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Get this newsletter

References

[1] Wikipedia Contributors, Cluster analysis (2021), https://en.wikipedia.org/wiki/Cluster_analysis

[2] J. C. Gower, A General Coefficient of Similarity and Some of Its Properties (1971), Biometrics

About Help Terms Privacy

Get the Medium app

Download on the App Store GET IT ON Google Play