Clustering on Mixed Data Types in Python

Ryan Kemmer Follow
Jan 24 · 5 min read

Image by the Author

During my first ever data science internship, I was given a seemingly simple task to find clusters within a dataset. Given my basic knowledge of clustering algorithms like K-Means, DBSCAN, and GMM I thought that I could easily get this task done. However, as I took a closer look into the dataset, I realized the data contained a mixture of **categorical** and **continuous** data, and many common methods of clustering I knew would not easily work.

> **Categorical data** consists of multiple discrete categories that commonly do not have any clear order or relationship to each-other. This data might look like *“Android”* or *“iOS”*.
>
> **Continuous data** consists of real numbers that can take any value. This data might look like *“3.14159”* or *“43"*.

Many datasets contain a mixture of categorical and continuous data. However, it is not straightforward how to cluster datasets with mixed data types. So how do we cluster on data that has both categorical and continuous features? Lets take a look at two simple ways to approach this problem using Python.

Dataset

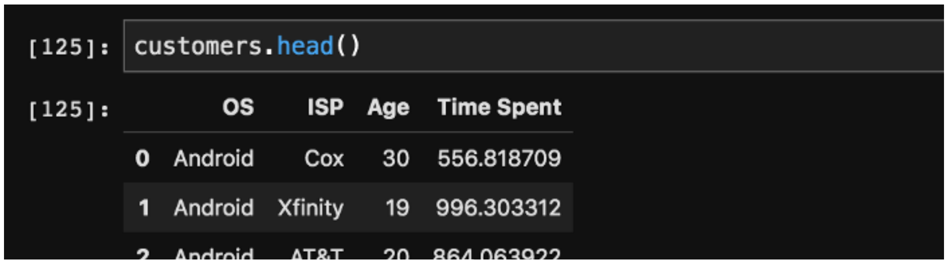
In this post, I am going to cluster a small dataset I created that has a mixture of categorical and continuous features. My fake data represents customer data that might be used to understand customers of an E-commerce website/app. Our fake dataset will have 4 features:

- **OS** — operating system of our fake customer (*Categorical*)
- **ISP** — internet service provider of our fake customer (*Categorical*)
- **Age** — customer age (*Continuous*)
- **Time Spent** — time that our fake users spent on our website (*Continuous*)

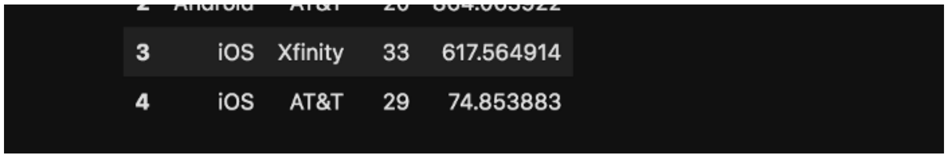
Here is the code used to generate our fake data.

```
#create dataset
import numpy as np
import pandas as pd

operating_systems = ["Android","iOS"]
isp_names = ["Cox","HughesNet","Xfinity","AT&T"]

data = []
for i in range(100):
    row = []
    row.append(np.random.choice(operating_systems)) #OS
    row.append(np.random.choice(isp_names)) #ISP
    row.append(np.random.poisson(lam=25)) #Age
    row.append(np.random.uniform(low=0.5, high=1000)) #Time Spent
    data.append(row)

customers = pd.DataFrame(data, columns = ['OS', 'ISP','Age','Time
Spent'])
```

Here is what our fake dataset looks like. Now lets get our hands dirty and do some clustering!



```
[125]: customers.head()

[125]:        OS      ISP  Age  Time Spent
       0  Android      Cox   30  556.818709
       1  Android  Xfinity   19  996.303312
       2  Android     AT&T   20  864.063922
```



2	Android	AT&T	[illegible]	[illegible]
3	iOS	Xfinity	33	617.564914
4	iOS	AT&T	29	74.853883

Method 1: K-Prototypes

The first clustering method we will try is called K-Prototypes. This algorithm is essentially a cross between the **K-means** algorithm and the **K-modes** algorithm.

To refresh our memory, **K-means** clusters data using euclidean distance. Meanwhile, **K-modes** clusters categorical data based off the number of matching categories between data points. A mixture of both of these: the K-prototype algorithm, is just what we need to cluster our fake customers!

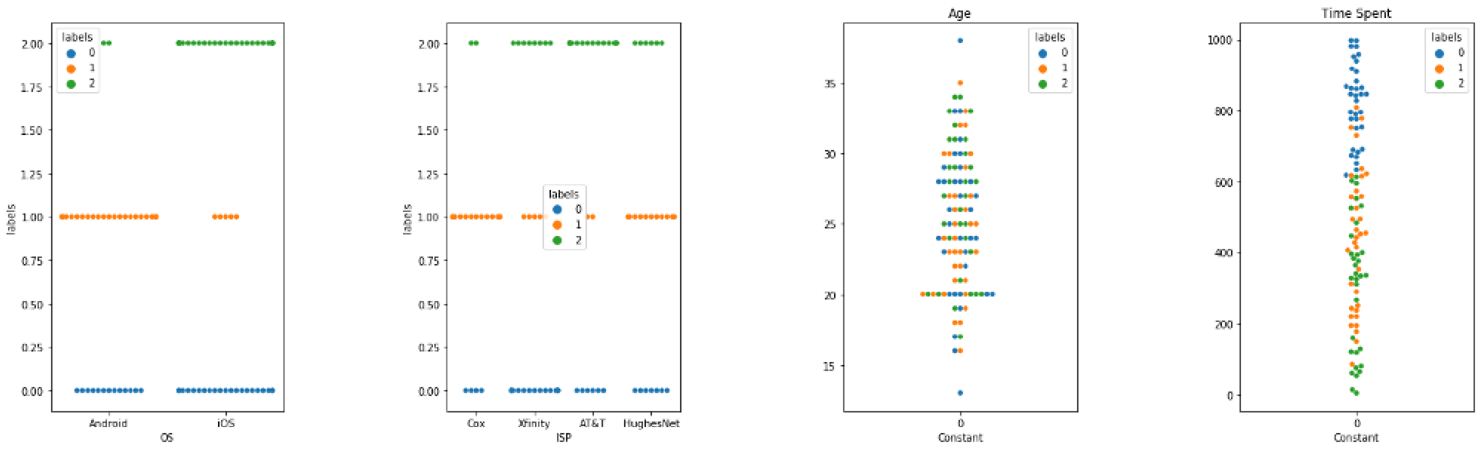
First, lets normalize the continuous features in our data to ensure that one feature is not interpreted as being more important than the other.

```
from sklearn import preprocessing

customers_norm = customers.copy()
scaler = preprocessing.MinMaxScaler()
customers_norm[['Age','Time Spent']] =
scaler.fit_transform(customers_norm[['Age','Time Spent']])
```

Now, lets use the K-prototypes algorithm to cluster our data. My implementation of the algorithm is very simple, with 3 clusters and the Cao initialization. When training the model, we specify which columns in the data are categorical (columns 0 and 1). More information on implementing and fine tuning this algorithm can be found here: https://github.com/nicodv/kmodes.

```
from kmodes.kprototypes import KPrototypes

kproto = KPrototypes(n_clusters=3, init='Cao')

clusters = kproto.fit_predict(customers_norm, categorical=[0, 1])

#join data with labels
labels = pd.DataFrame(clusters)
labeledCustomers = pd.concat((customers,labels),axis=1)
labeledCustomers = labeledCustomers.rename({0:'labels'},axis=1)
```

Finally, lets make some swarm plots to see how our clustering performed.



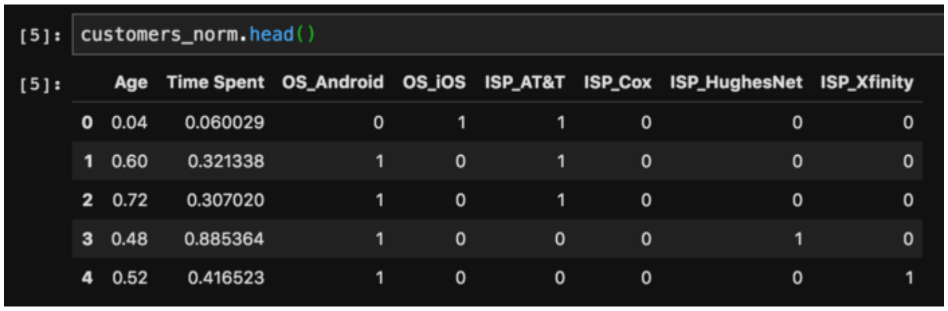
It worked! It looks like we have 3 fairly distinct clusters. Our first cluster seems to represent users that spent a lot of time on the website. Our second cluster has mostly Android users who have medium/ low time spent. Finally, our third cluster has mostly iOS users who also have medium/ low time spent.

Method 2: K-Means with One Hot Encoding

An alternative to using the K-prototypes algorithm is using the K-means algorithm and one hot encoding categorical variables. **One hot encoding** involves creating a new column for each categorical value in the dataset. Then a 1 or 0 is assigned depending on if that categorical value is in the data or not. Lets one hot encode the categorical values in our dataset using the pandas "get_dummies" function.

```
customers_norm = pd.get_dummies(customers_norm, columns=["OS","ISP"])
```

Now, our data looks like this. COOL! Time to do more clustering.



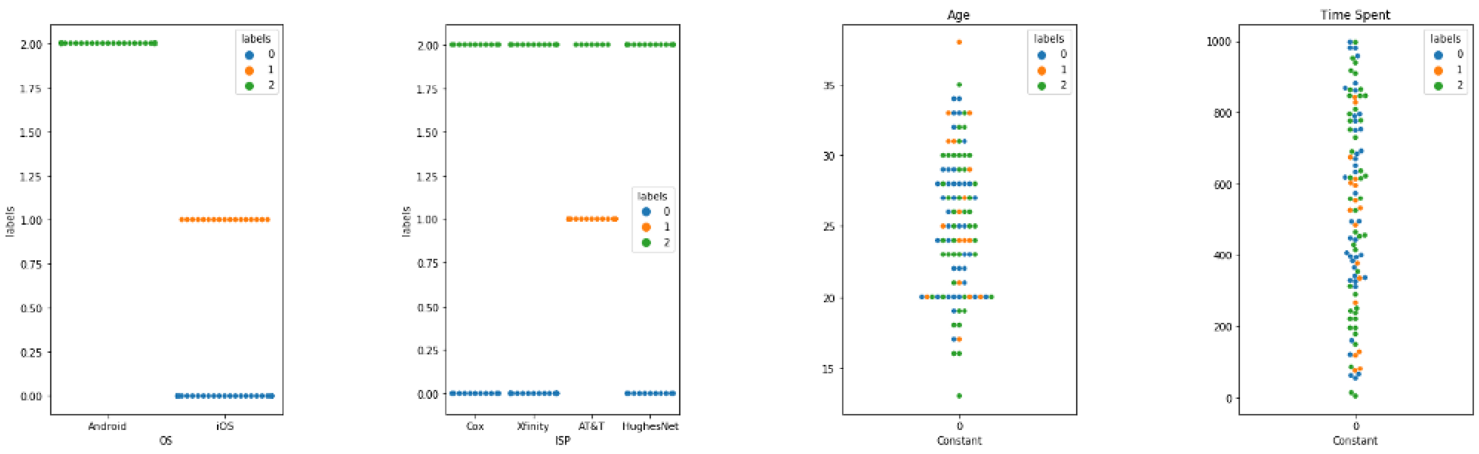
```
[5]: customers_norm.head()
```

[5]:

	Age	Time Spent	OS_Android	OS_iOS	ISP_AT&T	ISP_Cox	ISP_HughesNet	ISP_Xfinity
0	0.04	0.060029	0	1	1	0	0	0
1	0.60	0.321338	1	0	1	0	0	0
2	0.72	0.307020	1	0	1	0	0	0
3	0.48	0.885364	1	0	0	0	1	0
4	0.52	0.416523	1	0	0	0	0	1

Lets cluster this data using a basic implementation of K-means, and make some more swarm plots to see how this method works.

```
from sklearn.cluster import KMeans

kmeans = KMeans(3)
clusters = kmeans.fit_predict(customers_norm)
labels = pd.DataFrame(clusters)
labeledCustomers = pd.concat((customers,labels),axis=1)
labeledCustomers = labeledCustomers.rename({0:'labels'},axis=1)
```



Okay! It is interesting to see how much different these clusters look from the previously shown K-prototypes clusters. At first glance, it looks like the categorical features in our data contributed significantly more to our clusters, and our continuous features did not contribute much at all. The first cluster contains iOS users who DO NOT use AT&T. Our second cluster contains iOS users who DO use AT&T. Finally, our third cluster contains all Android users. Meanwhile, Age and Time Spent have a large distribution in all clusters.

Conclusion

In this post, we looked at two different methods of clustering mixed categorical/continuous data in Python. To begin with, we implemented the **K-prototypes** algorithm, an algorithm specifically designed to cluster mixed data using a combination of the K-means/K-modes. As an alternative, we tried using the **K-means algorithm with one hot encoding.**

Using our fake dataset, there are significant differences in the clusters determined by these two methods. K-prototypes seemed to evenly consider categorical and continuous features. Meanwhile, K-means seemed to weigh categorical features MUCH more heavily, which would likely be undesirable.

How do these methods work with your mixed dataset? Lets discuss!

Github repo with all code/visuals here:
https://github.com/ryankemmer/ClusteringMixedData

References

[1] de Vos, Nelis J. kmodes categorical clustering library. 2015–2021, https://github.com/nicodv/kmodes

[2] Huang, Z.: Clustering large data sets with mixed numeric and categorical values, Proceedings of the First Pacific Asia Knowledge Discovery and Data Mining Conference, Singapore, pp. 21–34, 1997.

Data Science Clustering Mixed Data Types Python K Means