




 [learn-co-curriculum](#) / [dsc-market-segmentation-clustering-lab](#) Public

 View license

 1 star  256 forks

 Star

 Watch ▾

 Code

 Issues

 Pull requests

 Actions

 Projects



 Security

 Insights

 solution ▾



This branch is [7 commits ahead](#), [8 commits behind](#) master.

 lindseyberlin Update metrics path ... on Aug 25, 2020  9

[View code](#)

 README.md

Market Segmentation with Clustering - Lab

Introduction

In this lab, you'll use your knowledge of clustering to perform market segmentation on a real-world dataset!

Objectives

In this lab you will:

- Use clustering to create and interpret market segmentation on real-world data

Getting Started

In this lab, you're going to work with the [Wholesale customers dataset](#) from the UCI Machine Learning datasets repository. This dataset contains data on wholesale purchasing information from real businesses. These businesses range from small cafes and hotels to grocery stores and other retailers.

Here's the data dictionary for this dataset:

Column	Description
FRESH	Annual spending on fresh products, such as fruits and vegetables
MILK	Annual spending on milk and dairy products
GROCERY	Annual spending on grocery products
FROZEN	Annual spending on frozen products
DETERGENTS_PAPER	Annual spending on detergents, cleaning supplies, and paper products
DELICATESSEN	Annual spending on meats and delicatessen products
CHANNEL	Type of customer. 1=Hotel/Restaurant/Cafe, 2=Retailer. (This is what we'll use clustering to predict)
REGION	Region of Portugal that the customer is located in. (This column will be dropped)

One benefit of working with this dataset for practice with segmentation is that we actually have the ground-truth labels of what market segment each customer actually belongs to. For this reason, we'll borrow some methodology from supervised learning and store these labels separately, so that we can use them afterward to check how well our clustering segmentation actually performed.

Let's get started by importing everything we'll need.

In the cell below:

- Import `pandas`, `numpy`, and `matplotlib.pyplot`, and set the standard alias for each.
- Use `numpy` to set a random seed of `0`.
- Set all matplotlib visualizations to appear inline.

```
import pandas as pd
import numpy as np
np.random.seed(0)
import matplotlib.pyplot as plt
%matplotlib inline
```

Now, let's load our data and inspect it. You'll find the data stored in 'wholesale_customers_data.csv' .

In the cell below, load the data into a DataFrame and then display the first five rows to ensure everything loaded correctly.

```
raw_df = pd.read_csv('wholesale_customers_data.csv')
raw_df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

</style>

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper
0	2	3	12669	9656	7561	214	2674
1	2	3	7057	9810	9568	1762	3293
2	2	3	6353	8808	7684	2405	3516
3	1	3	13265	1196	4221	6404	507
4	2	3	22615	5410	7198	3915	1777

Now, let's go ahead and store the 'Channel' column in a separate variable and then drop both the 'Channel' and 'Region' columns. Then, display the first five rows of the new DataFrame to ensure everything worked correctly.

```
channels = raw_df['Channel']
df = raw_df.drop(['Channel', 'Region'], axis=1)
df.head()
```

```
<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

```
</style>
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	12669	9656	7561	214	2674	1338
1	7057	9810	9568	1762	3293	1776
2	6353	8808	7684	2405	3516	7844
3	13265	1196	4221	6404	507	1788
4	22615	5410	7198	3915	1777	5185

Now, let's get right down to it and begin our clustering analysis.

In the cell below:

- Import `KMeans` from `sklearn.cluster`, and then create an instance of it. Set the number of clusters to 2
- Fit it to the data (`df`)
- Get the predictions from the clustering algorithm and store them in `cluster_preds`

```
from sklearn.cluster import KMeans
```

```
k_means = KMeans(n_clusters=2)
k_means.fit(df)
cluster_preds = k_means.predict(df)
```

Now, use some of the metrics to check the performance. You'll use `calinski_harabasz_score()` and `adjusted_rand_score()`, which can both be found inside `sklearn.metrics`.

In the cell below, import these scoring functions.

```
from sklearn.metrics import calinski_harabasz_score, adjusted_rand_score
```

Now, start with CH score to get the variance ratio.

```
calinski_harabasz_score(df, cluster_preds)
```

```
171.68461633384186
```

Although you don't have any other numbers to compare this to, this is a pretty low score, suggesting that the clusters aren't great.

Since you actually have ground-truth labels, in this case you can use `adjusted_rand_score()` to check how well the clustering performed. Adjusted Rand score is meant to compare two clusterings, which the score can interpret our labels as. This will tell us how similar the predicted clusters are to the actual channels.

Adjusted Rand score is bounded between -1 and 1. A score close to 1 shows that the clusters are almost identical. A score close to 0 means that predictions are essentially random, while a score close to -1 means that the predictions are pathologically bad, since they are worse than random chance.

In the cell below, call `adjusted_rand_score()` and pass in `channels` and `cluster_preds` to see how well your first iteration of clustering performed.

```
adjusted_rand_score(channels, cluster_preds)
```

```
-0.03060891241109425
```

According to these results, the clusterings were essentially no better than random chance. Let's see if you can improve this.

Scaling our dataset

Recall that k-means clustering is heavily affected by scaling. Since the clustering algorithm is distance-based, this makes sense. Let's use `StandardScaler` to scale our dataset and then try our clustering again and see if the results are different.

In the cells below:

- Import and instantiate `StandardScaler` and use it to transform the dataset
- Instantiate and fit k-means to this scaled data, and then use it to predict clusters
- Calculate the adjusted Rand score for these new predictions

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df)
```

```
scaled_k_means = KMeans(n_clusters=2)
scaled_k_means.fit(scaled_df)
scaled_preds = scaled_k_means.predict(scaled_df)
```

```
adjusted_rand_score(channels, scaled_preds)
```

```
0.212843835451224
```

That's a big improvement! Although it's not perfect, we can see that scaling our data had a significant effect on the quality of our clusters.

Incorporating PCA

Since clustering algorithms are distance-based, this means that dimensionality has a definite effect on their performance. The greater the dimensionality of the dataset, the greater the total area that we have to worry about our clusters existing in. Let's try using Principal Component Analysis to transform our data and see if this affects the performance of our clustering algorithm.

Since you've already seen PCA in a previous section, we will let you figure this out by yourself.

In the cells below:

- Import `PCA` from the appropriate module in sklearn

- Create a `PCA` instance and use it to transform our scaled data
- Investigate the explained variance ratio for each Principal Component. Consider dropping certain components to reduce dimensionality if you feel it is worth the loss of information
- Create a new `KMeans` object, fit it to our PCA-transformed data, and check the adjusted Rand score of the predictions it makes.

NOTE: Your overall goal here is to get the highest possible adjusted Rand score. Don't be afraid to change parameters and rerun things to see how it changes.

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=4)
pca_df = pca.fit_transform(scaled_df)
```

```
np.cumsum(pca.explained_variance_ratio_)
```

```
array([0.44082893, 0.72459292, 0.84793705, 0.94189209])
```

```
pca_k_means = KMeans(n_clusters=2)
pca_k_means.fit(pca_df)
pca_preds = pca_k_means.predict(pca_df)
```

```
adjusted_rand_score(channels, pca_preds)
```

```
0.13885372763476506
```

Question: What was the Highest Adjusted Rand Score you achieved? Interpret this score and determine the overall quality of the clustering. Did PCA affect the performance overall? How many principal components resulted in the best overall clustering performance? Why do you think this is?

Write your answer below this line:

```
# The highest ARS should be ~0.23, which suggests that the clusters are better than  
# but far from perfect. Overall, the quality of the clustering algorithm did a lot  
# first algorithm we ran on unscaled data. The best performance was achieved when re  
# number of principal components down to 4. The increase in model performance is lik  
# reduction in dimensionality. Although dropping the last 2 PCs means that we lose a  
# our explained variance, this proved to be a net-positive trade-off for the reducti  
# it provided.
```

Optional (Level up)

Hierarchical Agglomerative Clustering

Now that we've tried doing market segmentation with k-means clustering, let's end this lab by trying with HAC!

In the cells below, use [Agglomerative clustering](#) to make cluster predictions on the datasets we've created and see how HAC's performance compares to k-mean's performance.

NOTE: Don't just try HAC on the PCA-transformed dataset -- also compare algorithm performance on the scaled and unscaled datasets, as well!

```
from sklearn.cluster import AgglomerativeClustering
```

```
hac = AgglomerativeClustering(n_clusters=2)  
hac.fit(pca_df)  
hac_pca_preds = hac.labels_
```

```
adjusted_rand_score(channels, hac_pca_preds)
```

```
0.04822381910875346
```

```
hac2 = AgglomerativeClustering(n_clusters=2)  
hac2.fit(scaled_df)  
hac_scaled_preds = hac2.labels_
```

```
adjusted_rand_score(channels, hac_scaled_preds)
```



```
0.022565317001188977
```

```
hac3 = AgglomerativeClustering(n_clusters=2)
hac3.fit(df)
hac__preds = hac3.labels_
```

```
adjusted_rand_score(channels, hac__preds)
```

```
-0.01923156414375716
```

Summary

In this lab, you used your knowledge of clustering to perform a market segmentation on a real-world dataset. You started with a cluster analysis with poor performance, and then implemented some changes to iteratively improve the performance of the clustering analysis!

Releases

No releases published

Packages

No packages published

Contributors 7



Languages

● Jupyter Notebook 100.0%