Cluster Analysis with the *sklearn* ML Framework

David Gerbing
The School of Business
Portland State University
gerbing@pdx.edu

Table of Contents

The following data file contains information on 1199 products regarding gross revenue and number of customers.

Data: http://web.pdx.edu/~gerbing/data/Products.csv

```
from datetime import datetime as dt
now = dt.now()
print ("Analysis on", now.strftime("%Y-%m-%d"), "at", now.strftime("%H:%M"))
```

```
    Analysis on 2021-08-06 at 00:00
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

a. Read the data into a data frame, display its basic characteristics.

```
d = pd.read_csv('http://web.pdx.edu/~gerbing/data/Products.csv')
d.shape
```

```
    (1197, 2)
```

```
d.head()
```

	Customers	Revenue
0	5.0	15.253
1	3.0	12.710
2	4.0	3.548
3	4.0	9.874
4	12.0	1.000

b. Create the X data structure for this cluster analysis.

```
X = d[['Customers','Revenue']]
X.head()
```

	Customers	Revenue
0	5.0	15.253
1	3.0	12.710
2	4.0	3.548
3	4.0	9.874
4	12.0	1.000

```
n_features = X.shape[1]
print('Number of features:', n_features)
```

```
    Number of features: 2
```

Missing data check.

```
print (d.isna().sum())
print ('\nTotal Missing:', d.isna().sum().sum())
```

```
    Customers    0
    Revenue      0
    dtype: int64

    Total Missing: 0
```

c. Standardize the data.

```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
s_scaler = preprocessing.StandardScaler()
X = s_scaler.fit_transform(X)
```

```
X = pd.DataFrame(X, columns=['Customers','Revenue'])
X.head()
```

	Customers	Revenue
0	-0.720023	2.436975
1	-1.331928	1.819275
2	-1.025976	-0.406194
3	-1.025976	1.130405
4	1.421644	-1.025108

d. Verify the standardization.

Use `describe()` to verify the transformations correctly resulted in z-scores for each of the X variables.

```
X.describe().round(3)
```

	Customers	Revenue
count	1197.000	1197.000
mean	0.000	0.000
std	1.000	1.000
min	-1.944	-1.499
25%	-0.720	-0.842
50%	-0.108	-0.464
75%	0.810	0.671
max	2.645	2.486

The mean and standard deviation of each standardized variable are 0 and 1, respectively.

e. Do a hyper-parameter tuning from 2 to 15 clusters to help determine the number of clusters that best describe this data set. Compute inertia and silhouette for each model.

Create the *inertia* and *silhouette* data frames, initially as empty containers. Fitting the model, named *model*, with a specified number of clusters creates computed variables *model.inertia_* and *model.labels_*.

```
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.metrics import pairwise_distances
from sklearn.metrics import silhouette_samples, silhouette_score

max_nc = 15
inertia = []
silhouette = []

for i in range(2, max_nc):
    model = KMeans(n_clusters=i, init='k-means++', n_init=100, random_state=11)
    model.fit(X)
    inertia.append(model.inertia_)
    s_score = metrics.silhouette_score(X, model.labels_, metric='euclidean')
    silhouette.append(s_score)
```

f. Create a table of inertia and the silhouette index for each model.

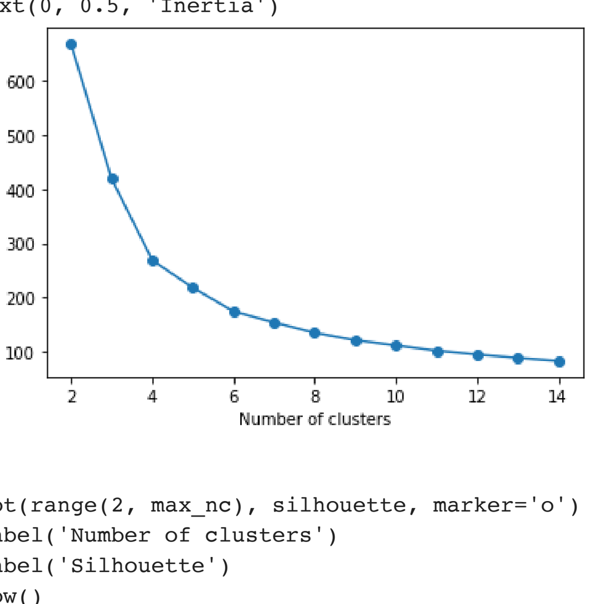
```
print('{:>2}{:>11}{:>9}'.format('nc', 'Silhouette', 'Inertia'))
print('-' * 24)
for i in range(2, max_nc):
    print('{:>2d}{:>8.3f}{:>12.3f}'.format(i, silhouette[i-2], inertia[i-2]))
```

```
    nc Silhouette  Inertia
    ------------------------
     2   0.623     668.607
     3   0.587     419.614
     4   0.509     269.091
     5   0.462     218.206
     6   0.441     174.582
     7   0.430     154.043
     8   0.415     134.885
     9   0.416     121.310
    10   0.400     111.669
    11   0.387     101.875
    12   0.384      94.910
    13   0.368      88.391
    14   0.376      82.887
```

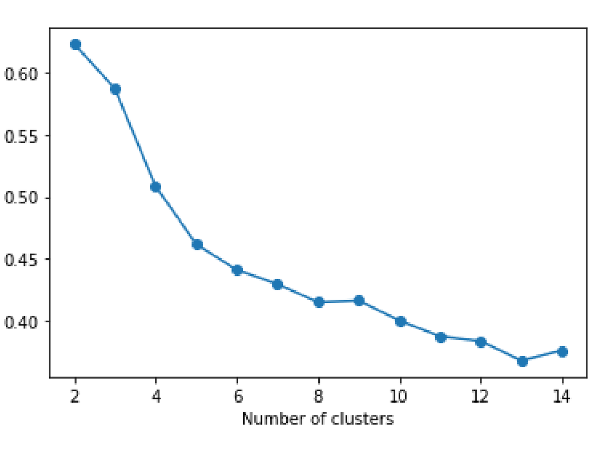
g. Create a plot of inertia and a plot of silhouette for each model.

```
plt.plot(range(2, max_nc), inertia, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

```
    Text(0, 0.5, 'Inertia')
```




```
plt.plot(range(2, max_nc), silhouette, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette')
plt.show()
```



h. What appears to be the optimal number of clusters? Why?

A clear three-cluster solution. Inertia drops quickly to three clusters, then levels off. Average silhouette score drops quicly after three clusters.

i. Do the cluster analysis just for the optimal number of clusters.

```
model = KMeans(n_clusters=3, init='k-means++', n_init=100, random_state=11)
model.fit(X)
```

```
    KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
           n_clusters=3, n_init=100, n_jobs=None, precompute_distances='auto',
           random_state=11, tol=0.0001, verbose=0)
```

j. Evaluate fit just for this one cluster analysis.

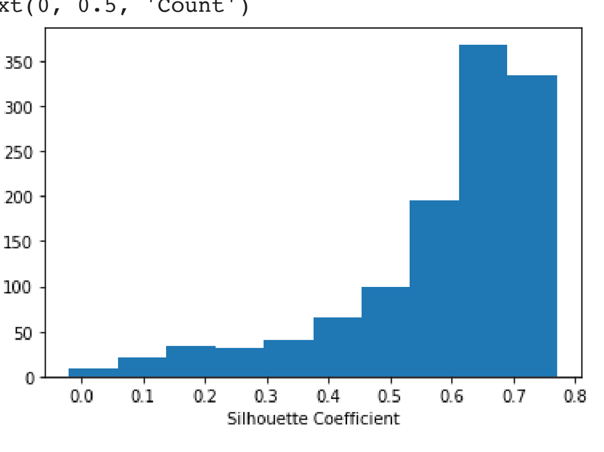
```
s_score = metrics.silhouette_score(X, model.labels_, metric='euclidean')
print('Error: %.3f ' % model.inertia_)
print('Mean silhouette score: %.3f' % s_score)
```

```
    Error: 419.614
    Mean silhouette score: 0.587
```

k. Construct the histogram of the individual silhouette coefficients. Comment on fit.

```
s_values = silhouette_samples(X, model.labels_)
plt.hist(s_values.round(3))
plt.xlabel('Silhouette Coefficient')
plt.ylabel('Count')
```

```
    Text(0, 0.5, 'Count')
```



The four-cluster solution appears to fit the data well. Most individual silhouette are above 0.5, with the largest group, by far with values above 0.7. Moreover, there are no negative values.

l. Assign the cluster membership and corresponding silhouette score to each sample in the data frame of the original data.

```
d['Cluster'] = model.labels_
d['S'] = s_values.round(3)
#d.to_csv("Clustered.csv", header=True)
d.head()
```

	Customers	Revenue	Cluster	S
0	5.0	15.253	2	0.565
1	3.0	12.710	2	0.749
2	4.0	3.548	1	0.436
3	4.0	9.874	1	0.037
4	12.0	1.000	0	0.703

m. How many samples are in each cluster?

```
lab = pd.DataFrame(model.labels_, columns=['labels'])
count = lab['labels'].value_counts()
count
```

```
    0    610
    1    431
    2    156
    Name: labels, dtype: int64
```

n. Show the standardized cluster centers. Interpret.

```
dcc = pd.DataFrame(model.cluster_centers_,
          columns=['Customers','Revenue']).round(3)
dcc['Freq'] = count
dcc.sort_values('Freq', ascending=False)
```

	Customers	Revenue	Freq
0	0.822	-0.838	610
1	-0.690	0.505	431
2	-1.308	1.881	156

Negative standardized values are below the mean. Cluster 0 drivers drive below average distance and have below average speeding.

*o. Get the cluster centroids for the original, unstandardized data.

```
avg = d.groupby(d['Cluster']).mean().round(2)
d_avg = pd.DataFrame(avg, columns=['Customers','Revenue']).round(3)
d_avg['Freq'] = count
d_avg = d_avg.sort_values('Customers', ascending=False)
#d_avg.to_csv("ClustCenters.csv", header=True)
d_avg = d_avg.reset_index()
d_avg
```

	Cluster	Customers	Revenue	Freq
0	0	10.04	1.77	610
1	1	5.10	7.30	431
2	2	3.08	12.96	156

Cluster 0 drivers drive on average about 50 miles less than Cluster 1 drivers, per trip. Cluster 0 drivers speed 5 mpg over teh speed limit about 9 and one-half percent less than Cluster 1 drivers.

p. Plot the scatterplot of the data and the cluster centroids.

```
if n_features == 2:
    sns.scatterplot(d['Customers'], d['Revenue'], s=50,
                    hue=d['Cluster'], palette=sns.color_palette(n_colors=3))
    sns.scatterplot(d_avg['Customers'], d_avg['Revenue'], s=125,
                    color="DimGray", legend=False)
    plt.grid()
```

```
    /usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning
      FutureWarning
    /usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning
      FutureWarning
```