20231029_assignment5

November 4, 2023

[ ]:
```
!pip install pandas numpy matplotlib seaborn scikit-learn==1.2.2


# This is to disable annoying warning messages from sklearn 1.2.2
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

```
WARNING: Ignoring invalid distribution -cikit-learn
(/home/ugrad10/zarafat/src/data-mining/.venv/lib/python3.10/site-
packages)
Requirement already satisfied: pandas in /home/ugrad10/zarafat/src/data-
mining/.venv/lib/python3.10/site-packages (2.1.0)
Requirement already satisfied: numpy in /home/ugrad10/zarafat/src/data-
mining/.venv/lib/python3.10/site-packages (1.26.0)
Requirement already satisfied: matplotlib in /home/ugrad10/zarafat/src/data-
mining/.venv/lib/python3.10/site-packages (3.8.0)
Requirement already satisfied: seaborn in /home/ugrad10/zarafat/src/data-
mining/.venv/lib/python3.10/site-packages (0.12.2)
Requirement already satisfied: scikit-learn==1.2.2 in
/home/ugrad10/zarafat/src/data-mining/.venv/lib/python3.10/site-packages (1.2.2)
Requirement already satisfied: scipy>=1.3.2 in /home/ugrad10/zarafat/src/data-
mining/.venv/lib/python3.10/site-packages (from scikit-learn==1.2.2) (1.11.2)
Requirement already satisfied: joblib>=1.1.1 in /home/ugrad10/zarafat/src/data-
mining/.venv/lib/python3.10/site-packages (from scikit-learn==1.2.2) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/ugrad10/zarafat/src/data-mining/.venv/lib/python3.10/site-packages (from
scikit-learn==1.2.2) (3.2.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/home/ugrad10/zarafat/src/data-mining/.venv/lib/python3.10/site-packages (from
pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /home/ugrad10/zarafat/src/data-
mining/.venv/lib/python3.10/site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /home/ugrad10/zarafat/src/data-
mining/.venv/lib/python3.10/site-packages (from pandas) (2023.3)
Requirement already satisfied: contourpy>=1.0.1 in
```

```
/home/ugrad10/zarafat/src/data-mining/.venv/lib/python3.10/site-packages (from
matplotlib) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /home/ugrad10/zarafat/src/data-
mining/.venv/lib/python3.10/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/home/ugrad10/zarafat/src/data-mining/.venv/lib/python3.10/site-packages (from
matplotlib) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/home/ugrad10/zarafat/src/data-mining/.venv/lib/python3.10/site-packages (from
matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/home/ugrad10/zarafat/src/data-mining/.venv/lib/python3.10/site-packages (from
matplotlib) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /home/ugrad10/zarafat/src/data-
mining/.venv/lib/python3.10/site-packages (from matplotlib) (10.0.1)
Requirement already satisfied: pyparsing>=2.3.1 in
/home/ugrad10/zarafat/src/data-mining/.venv/lib/python3.10/site-packages (from
matplotlib) (3.1.1)
Requirement already satisfied: six>=1.5 in /home/ugrad10/zarafat/src/data-
mining/.venv/lib/python3.10/site-packages (from python-dateutil>=2.8.2->pandas)
(1.16.0)
WARNING: Ignoring invalid distribution -cikit-learn
(/home/ugrad10/zarafat/src/data-mining/.venv/lib/python3.10/site-
packages)

[notice] A new release of pip is
available: 23.2.1 -> 23.3.1
[notice] To update, run:
pip install --upgrade pip
```

[ ]:

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

1 Assignment 5

1.1 Ziad Arafat

1.1.1 Reading in the data

1. We read in the CSV using the pandas library and store it in a dataframe.
2. We print the data in the first two rows using the `head()` method

[ ]:

```
df_google_reviews = pd.read_csv("google_review_ratings.csv")
print(df_google_reviews.head(n=5))
```

```
     User  C1   C2    C3    C4   C5    C6   C7    C8    C9  …   C15   C16  \
0  User 1  0.0  0.0  3.63  3.65  5.0  2.92  5.0  2.35  2.33  …  1.74  0.59
1  User 2  0.0  0.0  3.63  3.65  5.0  2.92  5.0  2.64  2.33  …  1.74  0.59
2  User 3  0.0  0.0  3.63  3.63  5.0  2.92  5.0  2.64  2.33  …  1.74  0.59
3  User 4  0.0  0.5  3.63  3.63  5.0  2.92  5.0  2.35  2.33  …  1.74  0.59
4  User 5  0.0  0.0  3.63  3.63  5.0  2.92  5.0  2.64  2.33  …  1.74  0.59

   C17  C18  C19  C20  C21  C22  C23  C24
0  0.5  0.0  0.5  0.0  0.0  0.0  0.0  0.0
1  0.5  0.0  0.5  0.0  0.0  0.0  0.0  0.0
2  0.5  0.0  0.5  0.0  0.0  0.0  0.0  0.0
3  0.5  0.0  0.5  0.0  0.0  0.0  0.0  0.0
4  0.5  0.0  0.5  0.0  0.0  0.0  0.0  0.0

[5 rows x 25 columns]
```

1.1.2 Preprocessing

Scaling

1. We use minmax to normalize the data

[ ]:
```python
# Drop the 'User' column
df_google_reviews = df_google_reviews.drop('User', axis=1)
# replace NaN with 0
df_google_reviews = df_google_reviews.fillna(0)

# use sklearn.preprocessing.MinMaxScaler to normalize the data
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df_google_reviews = scaler.fit_transform(df_google_reviews)
```

Select best k value

1. We use sklearn metrics and kmeans to select the best k-value
2. For each k, run k-means 5 times and compute the average Silhouette coefficient across those 5 running times and clusters

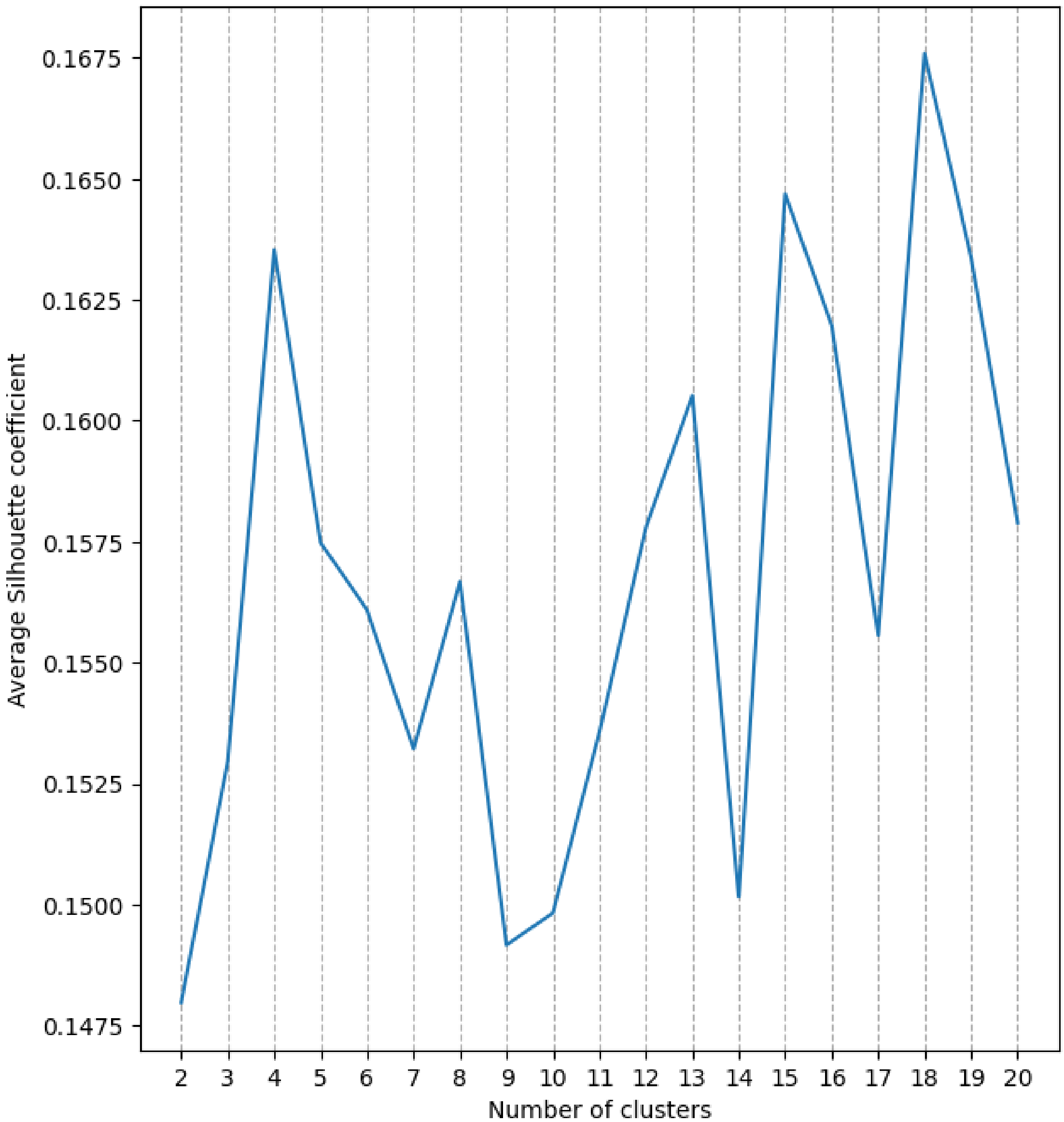
[ ]:
```python
# Use the Silhouette coefficient to select the best number of clusters (k) from␣
↪[1,20]
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import numpy as np

silhouette_scores_kmeans = []
for k in range(2, 21):
        kmeans = KMeans(n_clusters=k, random_state=42)
        silhouettes = []
```

```
        for i in range(5):
                kmeans.fit(df_google_reviews)
                silhouettes.append(silhouette_score(df_google_reviews, kmeans.
↪labels_))
        silhouette_scores_kmeans.append(np.mean(silhouettes))
```

Plot silhouette scores

1. We plot the scores by k-value to determine the best one
2. From the plot we can see that the best tested k value was 18
3. However we can also see that for all the values the score was much closer to 0 than to 1 indicating we have overall bad separation

[ ]:
```
import matplotlib.pyplot as plt

plt.plot(range(2, 21), silhouette_scores_kmeans)
plt.xlabel('Number of clusters')
plt.ylabel('Average Silhouette coefficient')
plt.xticks(range(2, 21))
# make width of plot bigger
plt.gcf().set_size_inches(7, 8)
plt.grid(axis='x', linestyle='--')
plt.show()
```



1.1.3 Training

1. Train a k-means model with the best k

[ ]:
```
best_k_kmeans = np.argmax(silhouette_scores_kmeans) + 2 # +2 because we started␣
↪from k=2

kmeans = KMeans(n_clusters=best_k_kmeans, random_state=2)
kmeans.fit(df_google_reviews)

print(best_k_kmeans)
```

18

1.1.4 Post analysis

Report Centroids

1. We print and plot the centroids to visualize how the clustering will be done

[ ]:
```
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[0.17342246 0.20700535 0.32517647 0.20106695 0.14585975 0.15589129
  0.99840123 0.29274846 0.44698684 0.49338251 0.56043288 0.77517297
  0.89375608 1.         0.78433155 0.14636364 0.14839572 0.17285561
  0.21997861 0.27474866 0.15372193 0.15486631 0.15331551 0.16834225]
 [0.52006122 0.50682993 0.48015646 0.27525735 0.18081563 0.11233234
  0.08417491 0.10544218 0.14331502 0.16516487 0.18237579 0.07135603
  0.04634052 0.06197857 0.51495238 0.34402721 0.27371429 0.2740068
  0.30731293 0.42576871 0.62291837 0.85783673 0.70680952 0.68278231]
 [0.24834921 0.46194709 0.70157672 0.70276477 0.76930944 0.50293114
  0.88358425 0.45063006 0.54839489 0.64740943 0.82881968 0.38003711
  0.28079853 0.19605421 0.54585185 0.44128042 0.12663492 0.10293122
  0.10273016 0.15191534 0.11568254 0.19146032 0.19713228 0.21091005]
 [0.30258947 0.38180351 0.59593684 0.91079978 0.87815156 0.6694809
  0.33714053 0.27246377 0.2698971  0.37129339 0.43054793 0.24793382
  0.12687155 0.08797584 0.24011228 0.2421614  0.17787368 0.11995789
  0.10497544 0.10195088 0.12906667 0.96047018 0.21458947 0.33237193]
 [0.23959344 0.36540984 0.34445902 0.32133506 0.54691567 0.75296894
  0.96419638 0.47579789 0.83385876 0.18719042 0.15173646 0.18204491
  0.17855288 0.13612744 0.45363934 0.13496393 0.1316918  0.13209836
  0.1684459  0.1526623  0.17735738 0.18612459 0.21310164 0.24078689]
 [0.29922892 0.34328916 0.38207229 0.26315333 0.19867097 0.19749435
  0.55806732 0.31136139 0.42152746 0.37186934 0.36178839 0.37526409
  0.39022758 0.67218402 0.8596747  0.94043373 0.7499759  0.44762651
  0.14263855 0.11243373 0.2556747  0.29768675 0.29172289 0.29495181]
 [0.22879487 0.32661538 0.61344615 0.72039599 0.83228919 0.90169402
  0.9037272  0.31674718 0.32871055 0.3162536  0.24877871 0.22871552
  0.25576772 0.24671626 0.27079487 0.18949231 0.14534359 0.06957436
  0.07612308 0.07978462 0.12147692 0.15855897 0.20170769 0.23865641]
 [0.26777705 0.87241967 0.52054426 0.37276408 0.3833615  0.55428379
  0.84732128 0.55162746 0.69802963 0.5092218  0.4522104  0.40456064
  0.31314188 0.86393443 0.47359344 0.15403279 0.15133115 0.14356721
  0.1313377  0.17416393 0.14630164 0.1548459  0.17921967 0.28359344]
 [0.36936634 0.47043564 0.52506931 0.75049663 0.82494641 0.46588104
  0.55351128 0.50081312 0.58589585 0.55605441 0.65513897 0.40020021
  0.97257527 0.33679712 0.29238944 0.18852805 0.19570297 0.15372937
  0.14961716 0.14481188 0.14980198 0.57171617 0.56431683 0.50218482]
 [0.10494118 0.16814379 0.31798693 0.1674817  0.10935078 0.11469832
  0.15068897 0.24198004 0.46614976 0.52030979 0.62196822 0.85664282
```

```
 0.88532734 0.99060458 0.78173856 0.16205882 0.16538562 0.18994118
 0.30403922 0.18646405 0.09675817 0.08994771 0.08478431 0.11303268]
[0.22461883 0.29006278 0.48931839 0.35779807 0.40655772 0.75012392
 0.92029957 0.59750655 0.890307   0.55503708 0.29333943 0.21814762
 0.52695353 0.94200017 0.84455605 0.13282511 0.12409865 0.12476233
 0.13278027 0.14734529 0.16034978 0.18938117 0.18655605 0.20904933]
[0.41304124 0.47262887 0.42522165 0.30054513 0.22305107 0.20005035
 0.2293748  0.20478485 0.23016207 0.20111704 0.19714663 0.20912322
 0.28284712 0.43467954 0.42556701 0.3088299  0.42478351 0.4956701
 0.69697423 0.49376289 0.29997423 0.43429381 0.42208247 0.4501134 ]
[0.48549351 0.74458442 0.73761688 0.54612414 0.36084148 0.30824959
 0.30286016 0.17268179 0.27840129 0.13805288 0.13946421 0.1296624
 0.14021676 0.14938894 0.36684416 0.16091558 0.17725325 0.1591039
 0.37381818 0.28135065 0.18561039 0.29866234 0.2908961  0.40756494]
[0.2171991  0.49179638 0.37379638 0.260436   0.2454168  0.50816573
 0.9030473  0.73333224 0.98373869 0.97243491 0.30588021 0.21295383
 0.1529369  0.14216576 0.54602262 0.14825339 0.10299095 0.09000905
 0.11683258 0.16163801 0.17263348 0.19932127 0.20636199 0.21159729]
[0.27924119 0.82916531 0.75353388 0.76172558 0.94623949 0.67286699
 0.35828235 0.40266813 0.4456366  0.50483471 0.24248963 0.29536727
 0.22051804 0.18394948 0.28337669 0.27085637 0.08296477 0.07666667
 0.08039024 0.12325203 0.16779404 0.17862873 0.20733875 0.26133333]
[0.34050758 0.48257576 0.66656818 0.74632112 0.76657685 0.53771325
 0.53626796 0.37984922 0.36028555 0.38205142 0.44913292 0.27057303
 0.04189949 0.0521816  0.21415909 0.18205303 0.18142424 0.15166667
 0.14242424 0.14122727 0.17776515 0.67171212 0.97903788 0.51790909]
[0.26729577 0.59239437 0.55767606 0.82328503 0.57621243 0.38522394
 0.33153042 0.18689188 0.8541441  0.49524354 0.03761431 0.03494426
 0.0429694  0.07605302 0.28483099 0.34011268 0.12360563 0.12271831
 0.17122535 0.75119718 0.47426761 0.80129577 0.35330986 0.24042254]
[0.24665085 0.30385424 0.32088814 0.29290737 0.26250218 0.31292754
 0.58353136 0.65544911 0.87644638 0.86119494 0.99451763 0.35228131
 0.27578635 0.12699872 0.28661695 0.22869831 0.14851525 0.07909492
 0.06081017 0.05881356 0.09838305 0.26229831 0.2045661  0.23615254]]
```

PCA with 2 components

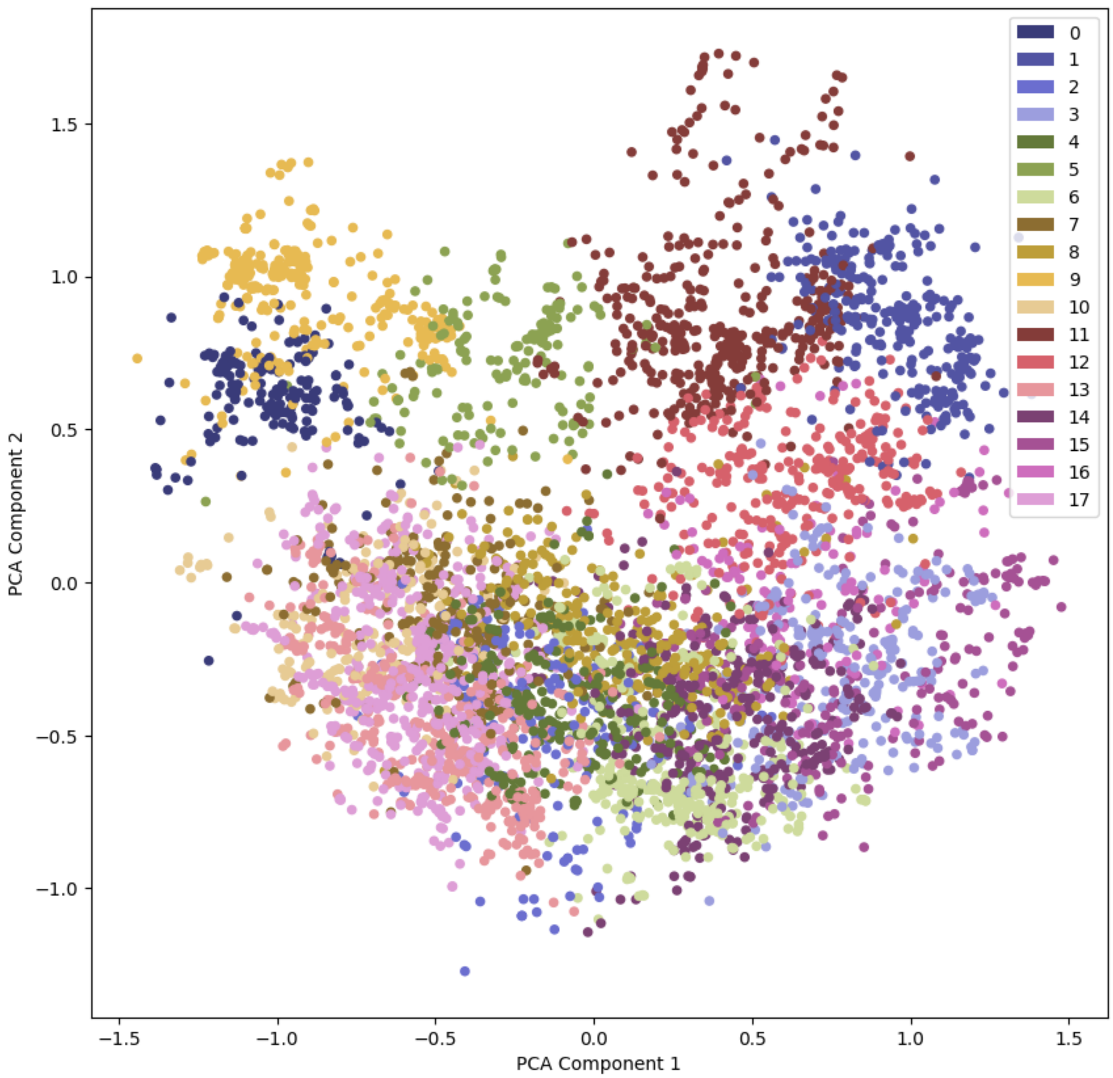
1. Project the data using PCA with two principal components

[ ]:
```python
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_google_reviews)
```

Plot clusters using Principal Components in 2D

[ ]:
```python
from matplotlib.patches import Patch

cluster_count = len(np.unique(kmeans.labels_))
```

```python
# Pick a color map with sufficiently many colors for the number of clusters
cmap = plt.cm.get_cmap('tab20b', cluster_count)

plt.scatter(
        df_pca[:, 0],
        df_pca[:, 1],
        c=kmeans.labels_,
        s=20,
        alpha=1,
        cmap=cmap
)
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
# make width of plot bigger
plt.gcf().set_size_inches(10, 10)

# add legend using cluster id as label
# Create a custom legend using Patch objects
legend_elements = [
        Patch(facecolor=cmap(i),
              label=str(i))
        for i in range(cluster_count)
]
plt.legend(handles=legend_elements)

plt.show()
```



1.1.5 Analysis of plot

1. We can see from the plot that most of the clusters have very bad seperation.
2. This is consistent with the very low silhouette coefficients we obtained earlier.

1.2 Question 2

1.2.1 Training GMM

Recording silhouette scores

[ ]:
```
from sklearn.mixture import GaussianMixture



# Record silhouette scores for each k
silhouette_scores_gmm = []
```

```
for k in range(2, 21):
        silhouettes = []
        for i in range(5):
                gmm = GaussianMixture(n_components=k, random_state=2)
                gmm.fit(df_google_reviews)
                silhouettes.append(
                        silhouette_score(
                                df_google_reviews,
                                gmm.predict(df_google_reviews)
                        )
                )
        silhouette_scores_gmm.append(np.mean(silhouettes))
```

Fitting using best k

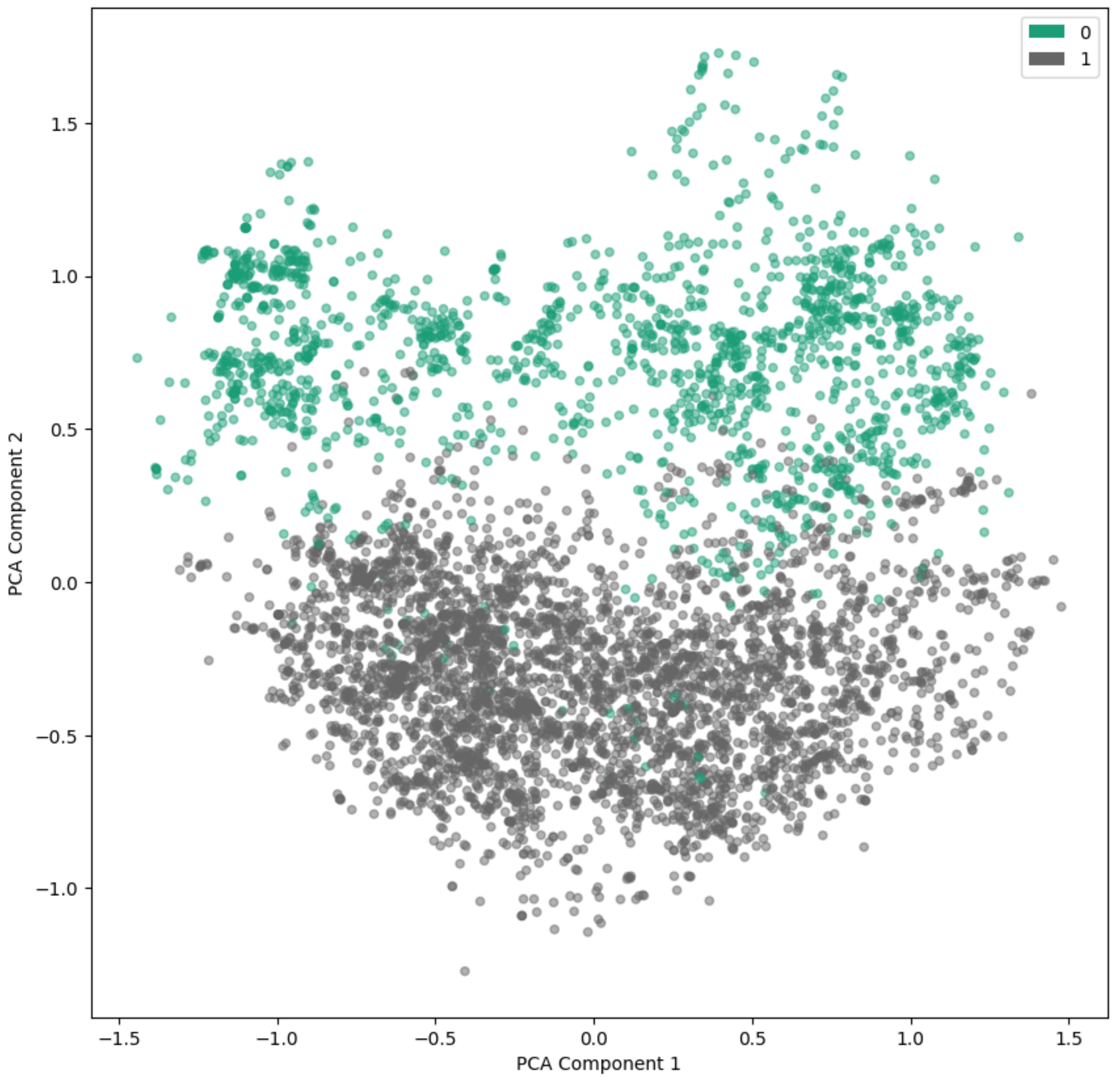
[ ]:
```
best_k_gmm = np.argmax(silhouette_scores_gmm) + 2 # +2 because we started from␣
  ↪k=2

gmm = GaussianMixture(n_components=best_k_gmm, random_state=2)
gmm.fit(df_google_reviews)

print(best_k_gmm)
```

2

1.2.2 Plotting clusters

[ ]:
```
from matplotlib.patches import Patch

cluster_count = len(np.unique(gmm.predict(df_google_reviews)))

# Pick a color map with sufficiently many colors for the number of clusters
cmap = plt.cm.get_cmap('Dark2', cluster_count)

plt.scatter(
        df_pca[:, 0],
        df_pca[:, 1],
        c=gmm.predict(df_google_reviews),
        s=20,
        alpha=0.5,
        cmap=cmap
)
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
# make width of plot bigger
plt.gcf().set_size_inches(10, 10)

legend_elements = [
```

```
        Patch(facecolor=cmap(i),
              label=str(i))
        for i in range(cluster_count)
]

plt.legend(handles=legend_elements)

plt.show()
```



1.2.3 Analysis of cluster seperation

1. Although we get really low silhouette values the clustering visually does not look so bad minus some overlap.
2. However we cannot expect 2 clusters to suffice for describing a complex dataset like this

3. Additionally it is clear that there are clusters existing that have not been defined

1.2.4 Training Spectral Clustering

Recording silhouette scores

[ ]:
```
from sklearn.cluster import SpectralClustering

# repeat for SpectralClustering
silhouette_scores_spectral = []
for k in range(2, 21):
        silhouettes = []
        for i in range(5):
                spectral = SpectralClustering(n_clusters=k, random_state=2)
                spectral.fit(df_google_reviews)
                silhouettes.append(
                        silhouette_score(
                                df_google_reviews,
                                spectral.labels_
                        )
                )
        silhouette_scores_spectral.append(np.mean(silhouettes))
```

Fitting using best k

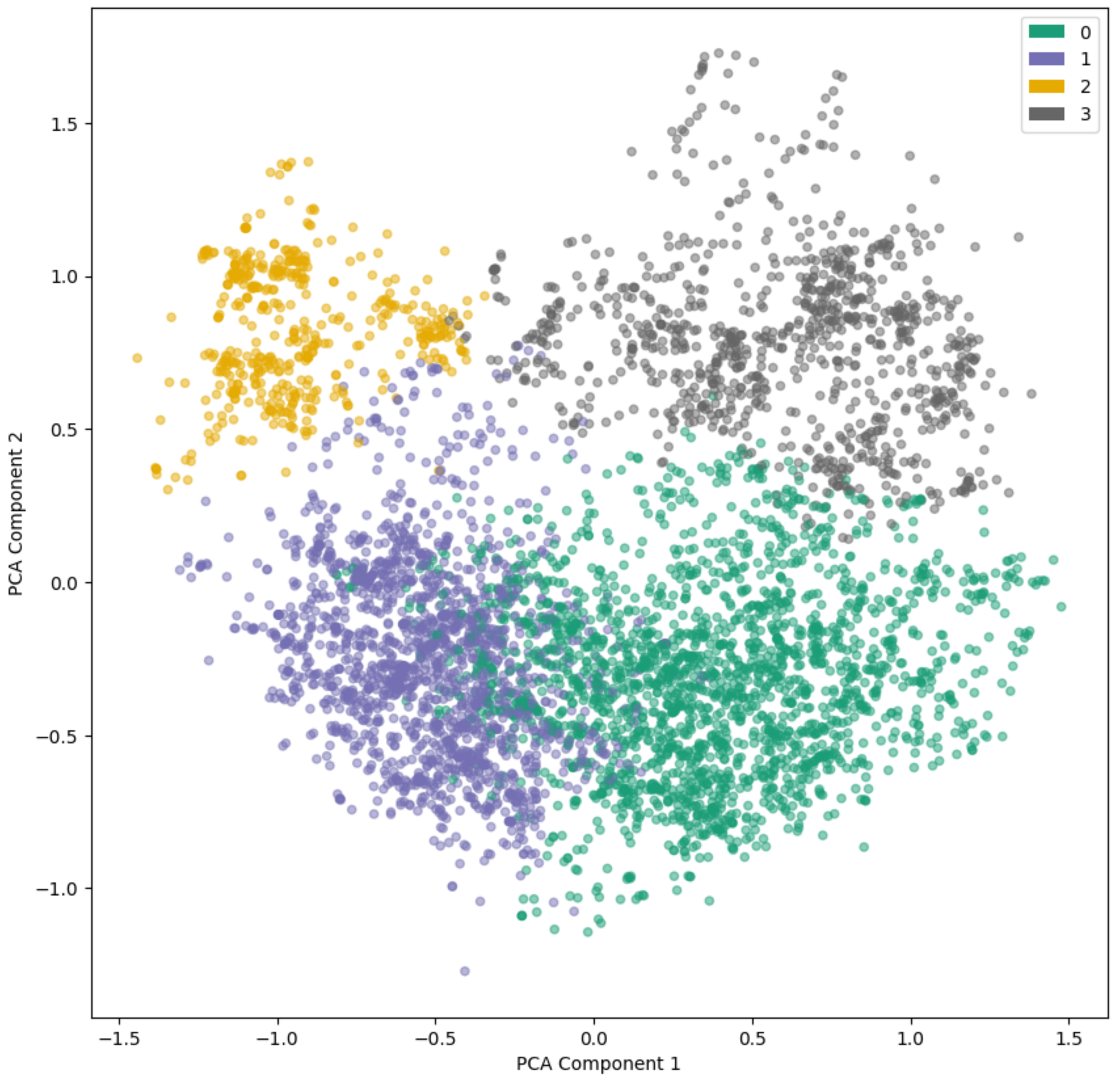
[ ]:
```
best_k_spectral = np.argmax(silhouette_scores_spectral) + 2 # +2 because we␣
 ↪started from k=2

spectral = SpectralClustering(n_clusters=best_k_spectral, random_state=2)

spectral.fit(df_google_reviews)

print(best_k_spectral)
```

```
4
```

1.2.5 Plotting spectral clusters

[ ]:
```
cluster_count = len(np.unique(spectral.labels_))

# Pick a color map with sufficiently many colors for the number of clusters
cmap = plt.cm.get_cmap('Dark2', cluster_count)

plt.scatter(
        df_pca[:, 0],
        df_pca[:, 1],
        c=spectral.labels_,
        s=20,
        alpha=0.5,
        cmap=cmap
```

```
)
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
# make width of plot bigger
plt.gcf().set_size_inches(10, 10)

legend_elements = [
        Patch(facecolor=cmap(i),
              label=str(i))
        for i in range(cluster_count)
]

plt.legend(handles=legend_elements)

plt.show()
```

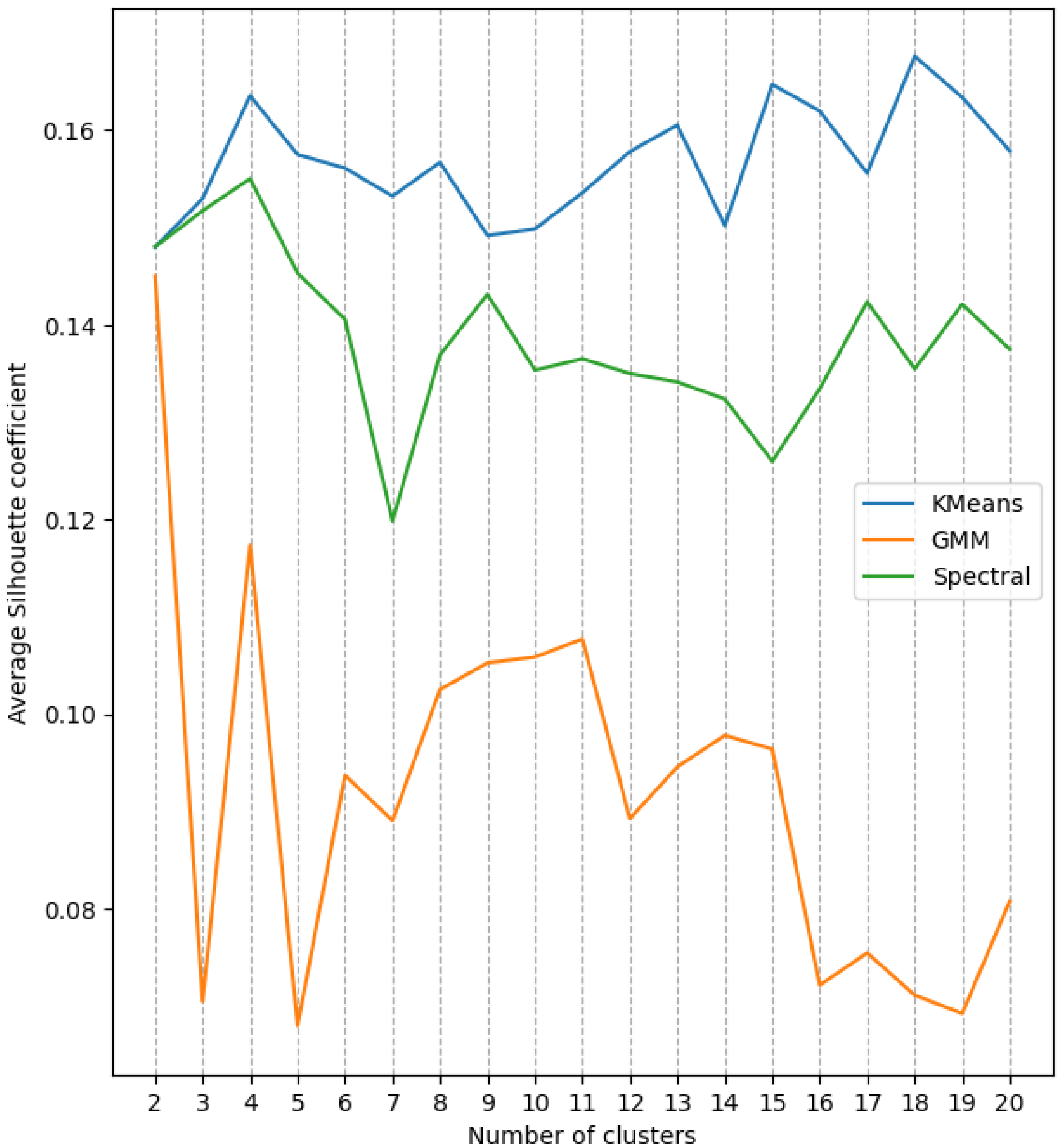


1.2.6 Plotting all the silhouette values

1. The graph shows that although kmeans has such a low silhouette value, it is still performing better than the other two algorithms in terms of silhouette score
2. Despite this, visually it does not seem that way but this is likely because the other algorithms had much lower `best_k` values

[ ]:

```
# plot all three silhouette scores together
plt.plot(range(2, 21), silhouette_scores_kmeans, label='KMeans')
plt.plot(range(2, 21), silhouette_scores_gmm, label='GMM')
plt.plot(range(2, 21), silhouette_scores_spectral, label='Spectral')
plt.xlabel('Number of clusters')
plt.ylabel('Average Silhouette coefficient')
plt.xticks(range(2, 21))
# make width of plot bigger
plt.gcf().set_size_inches(7, 8)
plt.grid(axis='x', linestyle='--')
plt.legend()
plt.show()
```



[ ]:
```
# repeat process with dbscan

from sklearn.cluster import DBSCAN

# Record silhouette scores for each k
silhouette_scores_dbscan = []

for eps in np.arange(0.1, 1.1, 0.1):
        silhouettes = []
        for i in range(5):
```

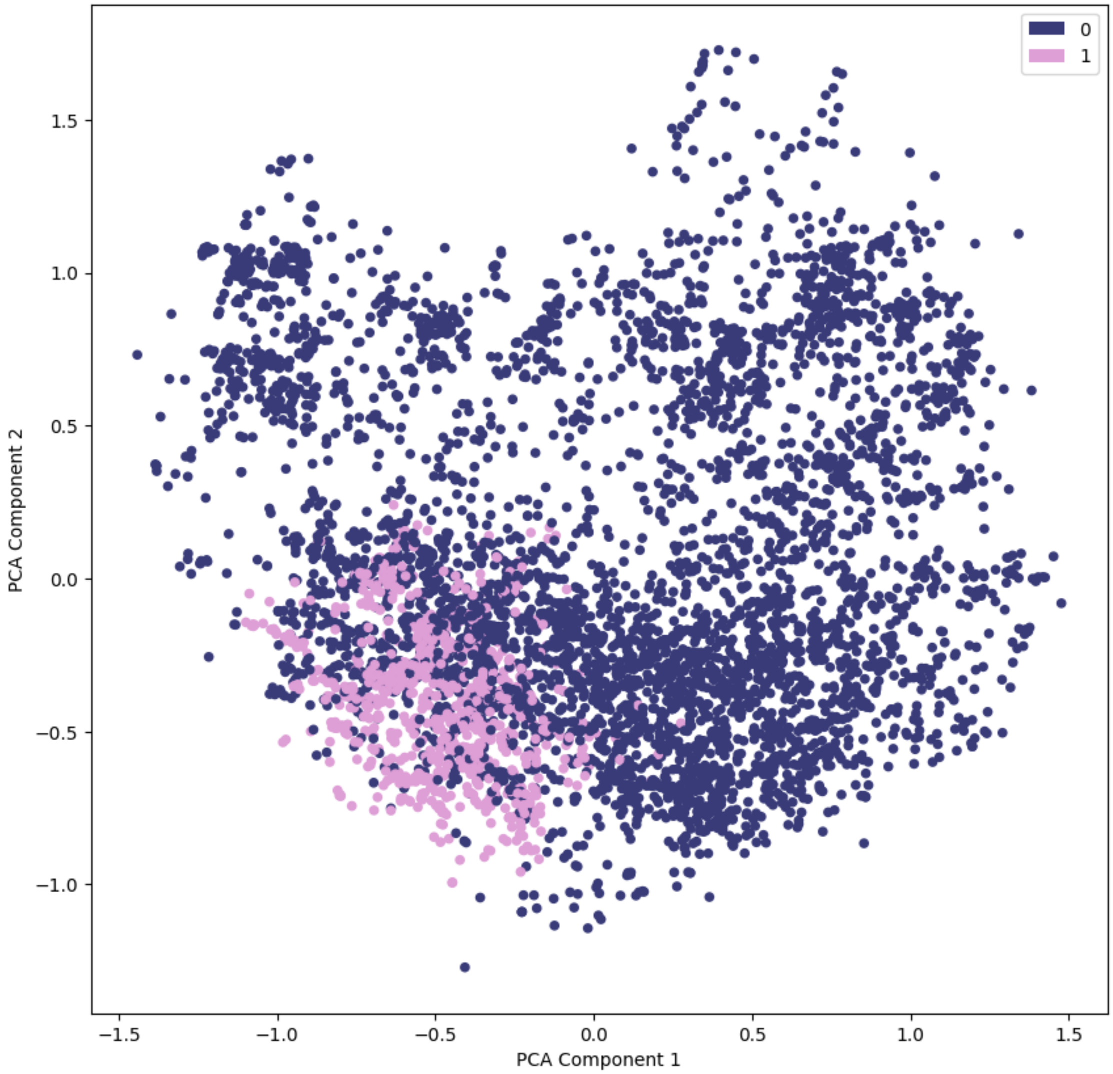
```
                dbscan = DBSCAN(eps=eps)
                dbscan.fit(df_google_reviews)
                silhouettes.append(
                        silhouette_score(
                                df_google_reviews,
                                dbscan.labels_
                        )
                )
        silhouette_scores_dbscan.append(np.mean(silhouettes))
```

[ ]:
```
eps_values = [i for i in np.arange(0.1, 1.1, 0.1)]
best_k_dbscan = eps_values[np.argmax(silhouette_scores_dbscan)]

#fit the model with the best eps value
dbscan = DBSCAN(eps=best_k_dbscan, metric='euclidean', min_samples=320)
dbscan.fit(df_google_reviews)

print(best_k_dbscan)
```

```
1.0
```

[ ]:
```
# plot the clusters
cluster_count = len(np.unique(dbscan.labels_))

# Pick a color map with sufficiently many colors for the number of clusters
cmap = plt.cm.get_cmap('tab20b', cluster_count)

plt.scatter(
        df_pca[:, 0],
        df_pca[:, 1],
        c=dbscan.labels_,
        s=20,
        alpha=1,
        cmap=cmap
)
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
# make width of plot bigger
plt.gcf().set_size_inches(10, 10)

legend_elements = [
        Patch(facecolor=cmap(i),
              label=str(i))
        for i in range(cluster_count)
]

plt.legend(handles=legend_elements)
```

```
plt.show()
```



[ ]:
```
print(max(silhouette_scores_dbscan))
```

0.1407142487783989

[ ]:
```
```