project3

December 9, 2021

1 In this final movies project, I will demonstrate some ML methods. We revisit the same dataset already used in previous projects. This will highlight what machine learning methods can and cannot do for you, compared to Hypothesis testing and Prediction methods.

1.0.1 Dataset description

This dataset features ratings data of 400 movies from 1097 research participants.

- 1st row: Headers (Movie titles/questions) – note that the indexing in this list is from 1
- Row 2-1098: Responses from individual participants
- Columns 1-400: These columns contain the ratings for the 400 movies (0 to 4, and missing)
- Columns 401-421: These columns contain self-assessments on sensation seeking behaviors (1-5)
- Columns 422-464: These columns contain responses to personality questions (1-5)
- Columns 465-474: These columns contain self-reported movie experience ratings (1-5)
- Column 475: Gender identity (1 = female, 2 = male, 3 = self-described)
- Column 476: Only child (1 = yes, 0 = no, -1 = no response)
- Column 477: Movies are best enjoyed alone (1 = yes, 0 = no, -1 = no response)

[ ]:
```
from myModules import movieClass
import pandas as pd
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.cluster import KMeans
from yellowbrick.cluster import SilhouetteVisualizer
from sklearn.linear_model import LogisticRegression
from sklearn import model_selection
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
```

```
sns.set(style="darkgrid")

# Data preprocessing and labels
Movies = movieClass.movie(verbose=True, alpha=0.005)
usrData = pd.DataFrame(Movies.userData()).T
names = ['User ' + str(i) for i in range(len(usrData.columns))]
usrData.columns = names
data = usrData[420:474].T
factors = Movies.titles[420:474]
```

1.1 PCA for Dimensionality Reduction

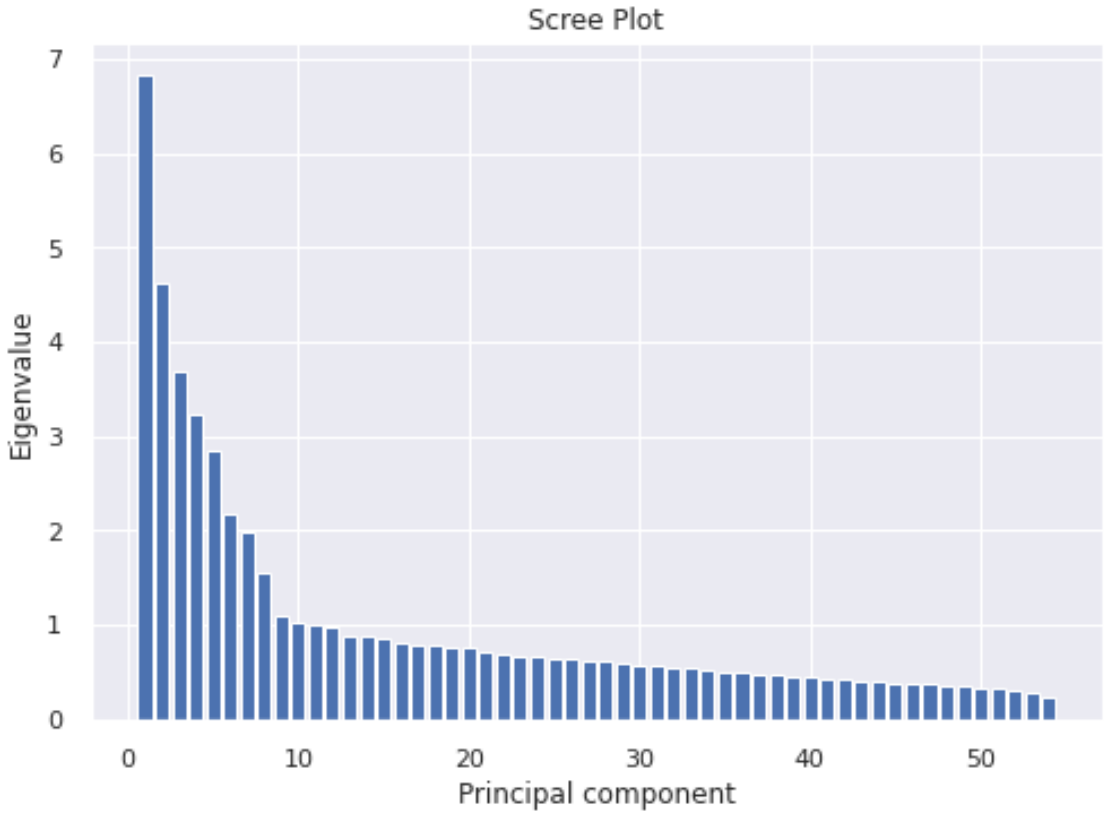
I apply an overall PCA to the data in columns 421-474. These columns contain self-report answers to personality and how these individuals experience movies, respectively. Missing data were filled with column averages as no aggressive outliers had been shifting distributions from center.

1.1.1 a) Determine the number of factors (principal components). Make a Scree plot.

1.1.2 b) Semantically interpret what those factors represent (hint: Inspect the loadings matrix). Explicitly name the factors and decided to interpret meaningfully in a).

[ ]:
```
# Z-score the data: same as StandardScaler().fit_transform(data.values)
zscoredData = stats.zscore(data)

# Run the PCA:
pca = PCA().fit(zscoredData)

# Eigenvalues: Single vector of eigenvalues in decreasing order of magnitude
eigVals = pca.explained_variance_

# Loadings (eigenvectors): Weights per factor in terms of the original data.␣
 ↪Where do the
# principal components point, in terms of the 54 questions?
loadings = pca.components_

# Rotated Data: Simply the transformed data - we had 1097 particpants (rows) in
# terms of 54 variables (each question is a column); now we have 1097␣
 ↪participants in terms of 54
# factors ordered by decreasing eigenvalue
rotatedData = pca.fit_transform(zscoredData)

fig1 = plt.figure()
numClasses = data.shape[1]
plt.bar(np.linspace(1, numClasses, numClasses), eigVals)
plt.xlabel('Principal component')
plt.ylabel('Eigenvalue')
```

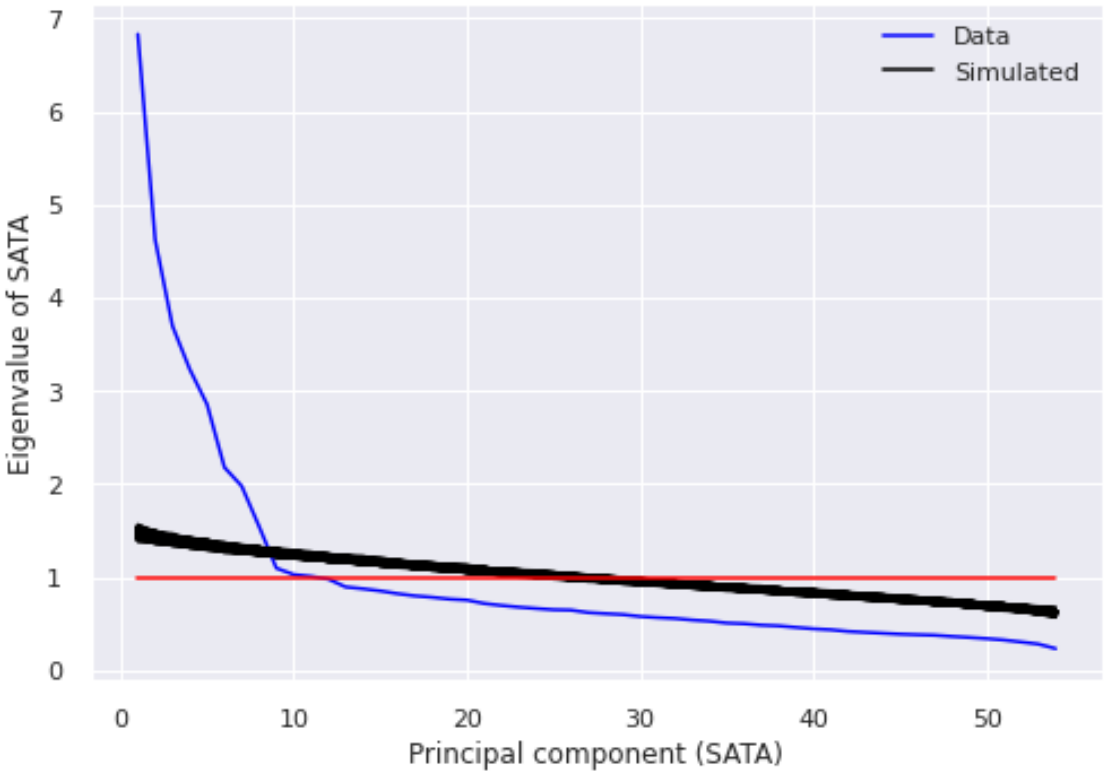
```
plt.title('Scree Plot')
plt.savefig("scree.png")
plt.show()
```



1.1.3 We use Horn's method to determine the number of relevant factors to consider; here, only 8 factors were above the noise and are thus chosen. The components are listed in subplots to follow.

[ ]:
```
# Horns Method for choosng 8 factors (above noise): Simulate noise␣
 ↪distributions to see which factors exceed
# what you would expect from noise. Resampling/Bootstrap-based. This is␣
 ↪explained in the NDS book, in
# the PCA chapter as described on page 239ff of the NDS book

# Initialize variables:
nDraws = 2000  # How many repetitions per resampling?
# How many rows to recreate the dimensionality of the original data?
numRows = data.shape[0]
# How many columns to recreate the dimensionality of the original data?
numColumns = data.shape[1]
```

```
# Initialize array to keep eigenvalues of sata
eigSata = np.empty([nDraws, numColumns])
eigSata[:] = np.NaN  # Convert to NaN

for i in range(nDraws):
    # Draw the sata from a normal distribution:
    sata = np.random.normal(0, 1, [numRows, numColumns])
    # Run the PCA on the sata:
    pca = PCA()
    pca.fit(sata)
    # Keep the eigenvalues:
    temp = pca.explained_variance_
    eigSata[i] = temp

# Make a plot of that and superimpose the real data on top of the sata:
plt.plot(np.linspace(1, numColumns, numColumns), eigVals,
         color='blue')  # plot eigVals from section 4
plt.plot(np.linspace(1, numColumns, numColumns),
         np.transpose(eigSata), color='black')  # plot eigSata
plt.plot([1, numColumns], [1, 1], color='red')  # Kaiser criterion line (for␣
 ↪reference, where eigenvalues > 1)
plt.xlabel('Principal component (SATA)')
plt.ylabel('Eigenvalue of SATA')
plt.legend(['Data', 'Simulated'])
plt.savefig('horns.png')
plt.show()
```



1.1.4 We look at loadings for all principal components to determine the greatest contributions of features (our questions) in a given direction (we use the L1 norm for eigenvectors to plot below).

[ ]:
```
fig2, axs = plt.subplots(4, 2, figsize=(15, 10), sharey=True, sharex=True)
x = np.linspace(1, data.shape[1], data.shape[1])
pc = 0
for row in range(4):
    for col in range(2):
        y = np.abs(loadings[pc, :])
        axs[row, col].bar(x, y)
        title = 'PC' + str(pc+1) + ': ' + \
            str(factors[np.argmax(np.abs(loadings[pc, :]))])
        axs[row, col].set_title(title[0:95], fontdict={'fontsize': 8}, pad=2.0)
        pc += 1

plt.subplots_adjust(left=0.125,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.05,
                    hspace=0.5)
```

```
fig2.text(0.5, 0.04, 'Question', ha='center')
fig2.text(0.04, 0.5, 'Eigenvector L1 Norm', va='center', rotation='vertical')
plt.savefig("PCnorms.png")
plt.show()
```

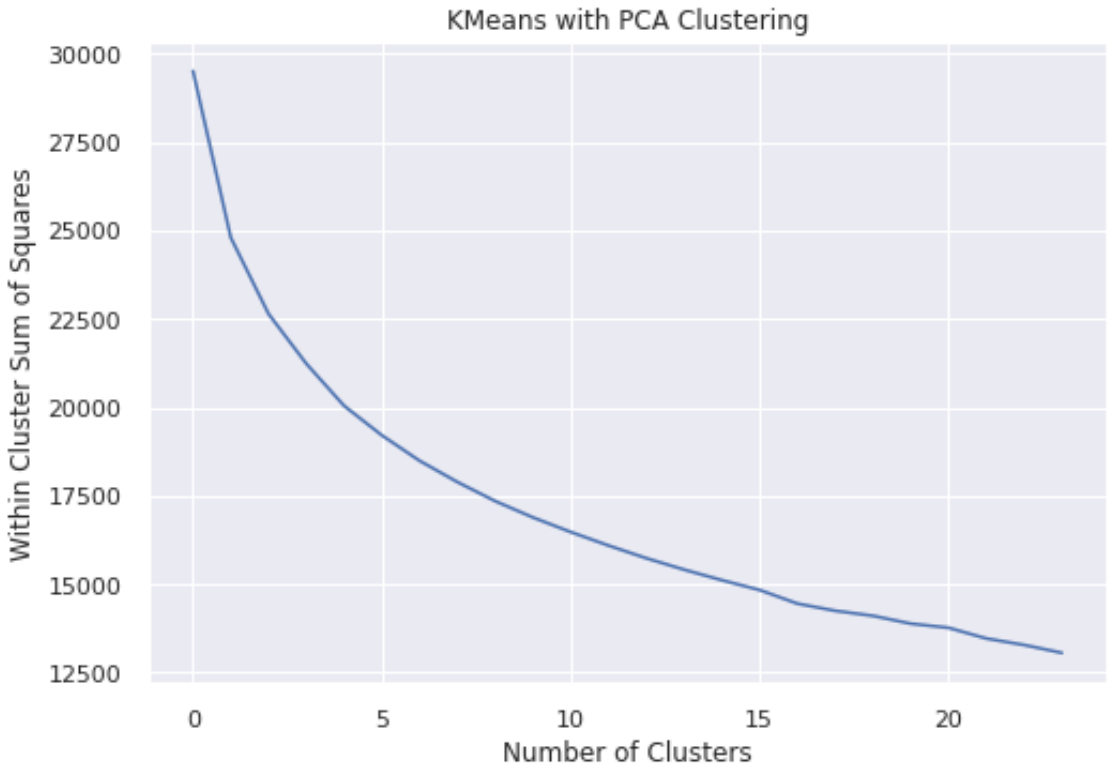
1.1.5 Identify clusters in this new space (rotated PCA frame). kMeans is chosen as the clustering algorithm.

Note: The intuition is that PCA seeks to represent all n data vectors as linear combinations of a small number of eigenvectors, and does it to minimize the mean-squared reconstruction error. In contrast, K-means seeks to represent all n data vectors via small number of cluster centroids, i.e. to represent them as linear combinations of a small number of cluster centroid vectors where linear combination weights must be all zero except for the single 1. This is also done to minimize the mean-squared reconstruction error. Therefore K-means can be seen as a super-sparse PCA.

Reference: https://stats.stackexchange.com/questions/183236/what-is-the-relation-between-k-means-clustering-and-pca

[ ]:
```
# Now using the 8 components we selected
pca = PCA(n_components=8).fit(zscoredData)
rotatedData = pca.fit_transform(zscoredData)
```

```
n_clusters = 25
cost = []
for i in range(1, n_clusters):
    kmean = KMeans(i)
    kmean.fit(rotatedData)
    cost.append(kmean.inertia_)
plt.ylabel('Within Cluster Sum of Squares')
plt.xlabel('Number of Clusters')
plt.title('KMeans with PCA Clustering')
plt.plot(cost, 'bx-')
plt.savefig("BadElbow.png")
plt.show()
```

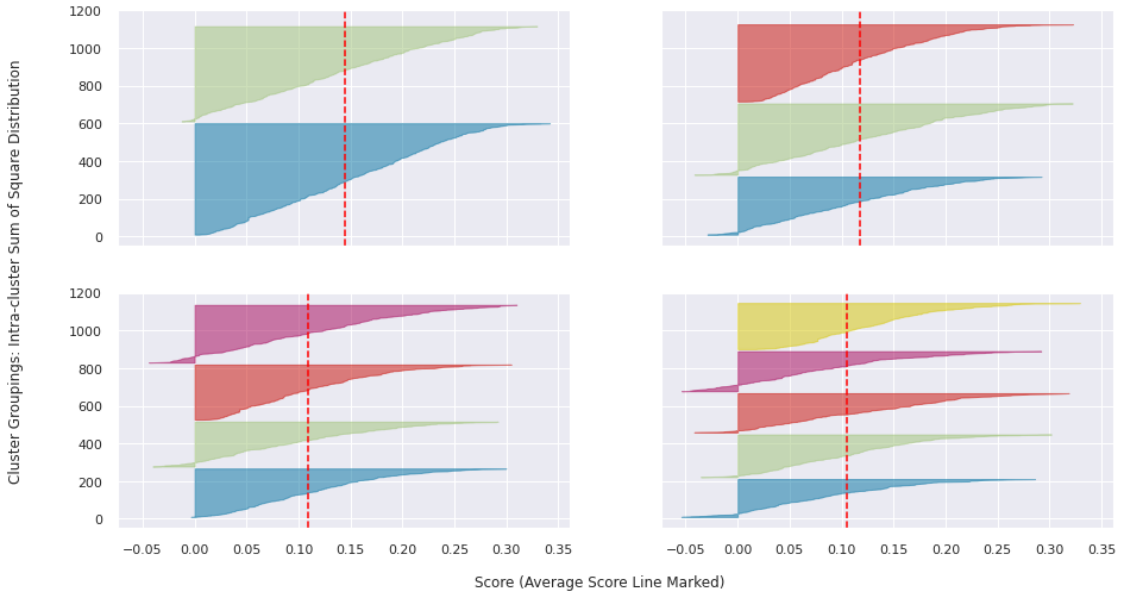


1.1.6 A Bad Elbow

At first the elbow method was considered but it was difficult to discern the joint at which the clustering groups were chosen. We then go into Silihouette score comparisons that help is make a more informed decision.

[ ]:
```
# see: https://vitalflux.com/
 ↪kmeans-silhouette-score-explained-with-python-example/
fig3, ax = plt.subplots(2, 2, figsize=(15, 8), sharex=True, sharey=True)
```

```
for i in [2, 3, 4, 5]:
    km = KMeans(n_clusters=i, init='k-means++',
                n_init=10, max_iter=400, random_state=42)
    q, mod = divmod(i, 2)
    visualizer = SilhouetteVisualizer(
        km, colors='yellowbrick', ax=ax[q-1][mod])
    visualizer.fit(rotatedData)
fig3.text(0.5, 0.04, 'Score (Average Score Line Marked)', ha='center')
fig3.text(0.04, 0.5, 'Cluster Groupings: Intra-cluster Sum of Square␣
↪Distribution',
          va='center', rotation='vertical')
plt.savefig("silhouette.png")
plt.show()
```



1.1.7 Better Option: Silhouette Analysis

Here is the Silhouette analysis done on the above plots with an aim to select an optimal value for cluster groups.

Choosing to cluster in groups of 4 and 5 looks to be suboptimal for the given data as their average cluster scores are less positive than the ones for 2 and 3 clusters. Wide fluctuations in the size of the silhouette plots. There is also some fluctuation in their distributions. We also see more pronounced misclassification (negative scores) in their plots.

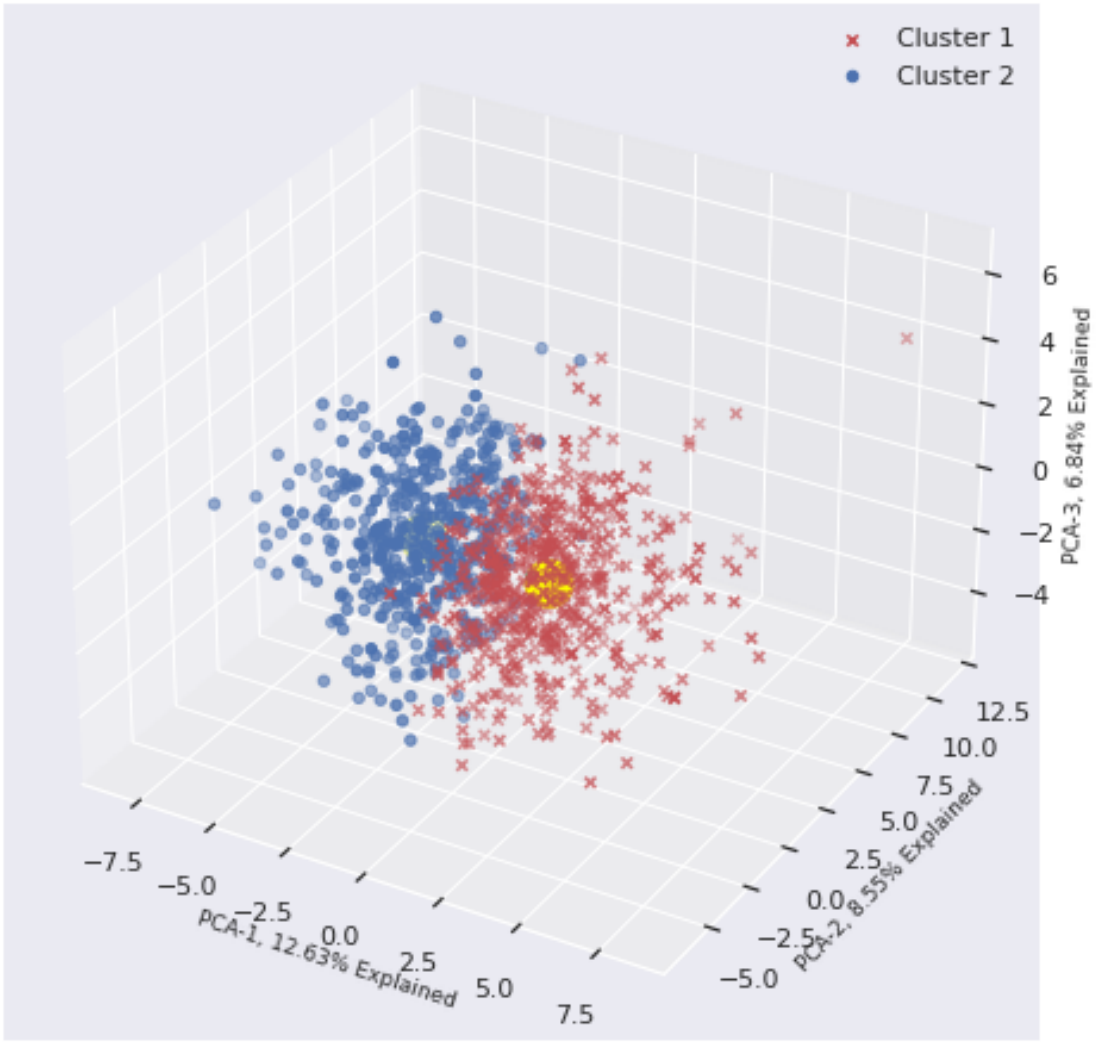
The value of 2 and 3 for the number of clusters looks to be optimal. The silhouette score for each cluster is above average silhouette scores. While the fluctuation in size is similar, the thickness is more uniform with 2 clusters than that with 3. We also see a greater propostion of misclassification possible with 3 clusters and thus 2 are chosen.

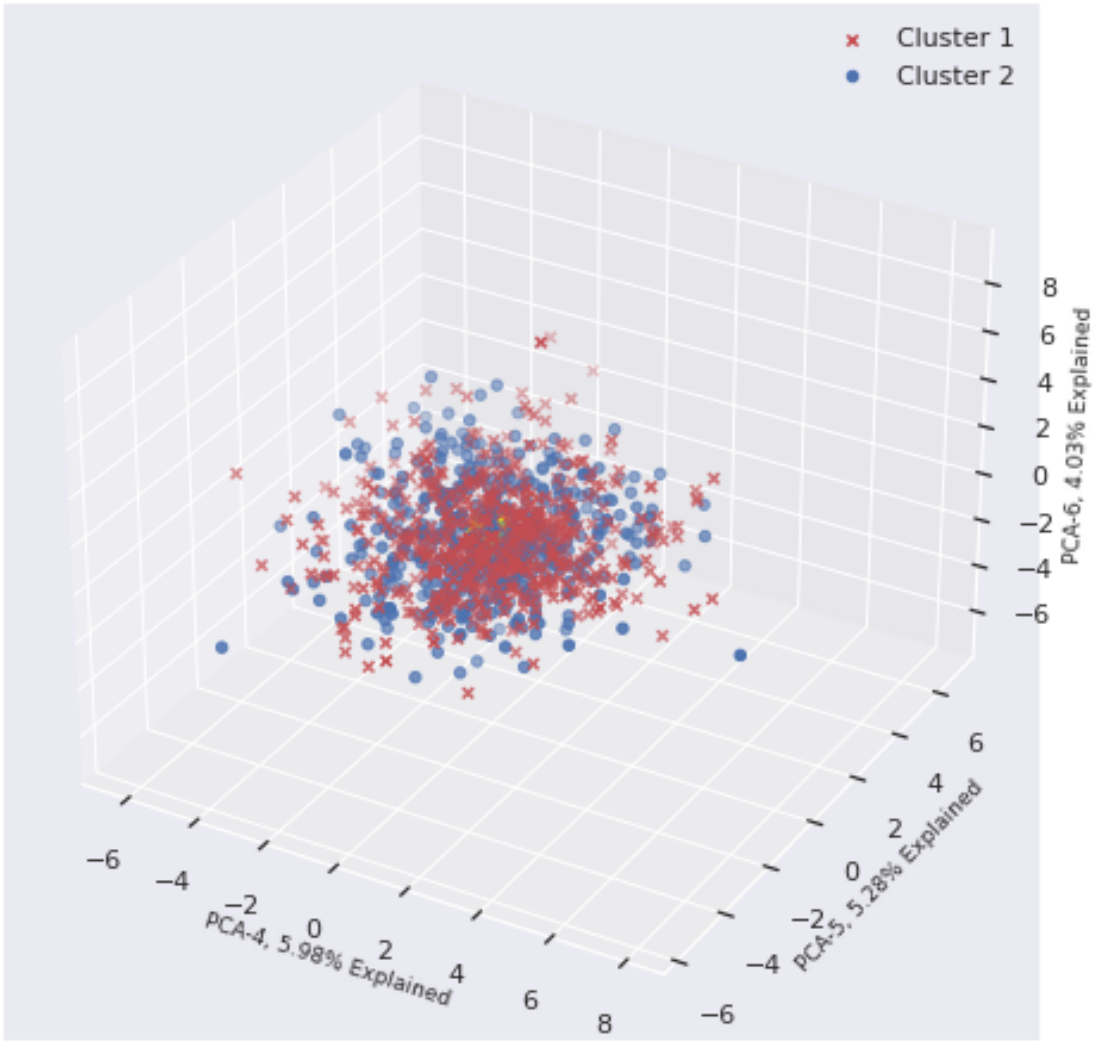
1.1.8 We plot the data from columns 421-474 in the new coordinate system, where each dot represents a person, and the axes represent the 8 factors from above.

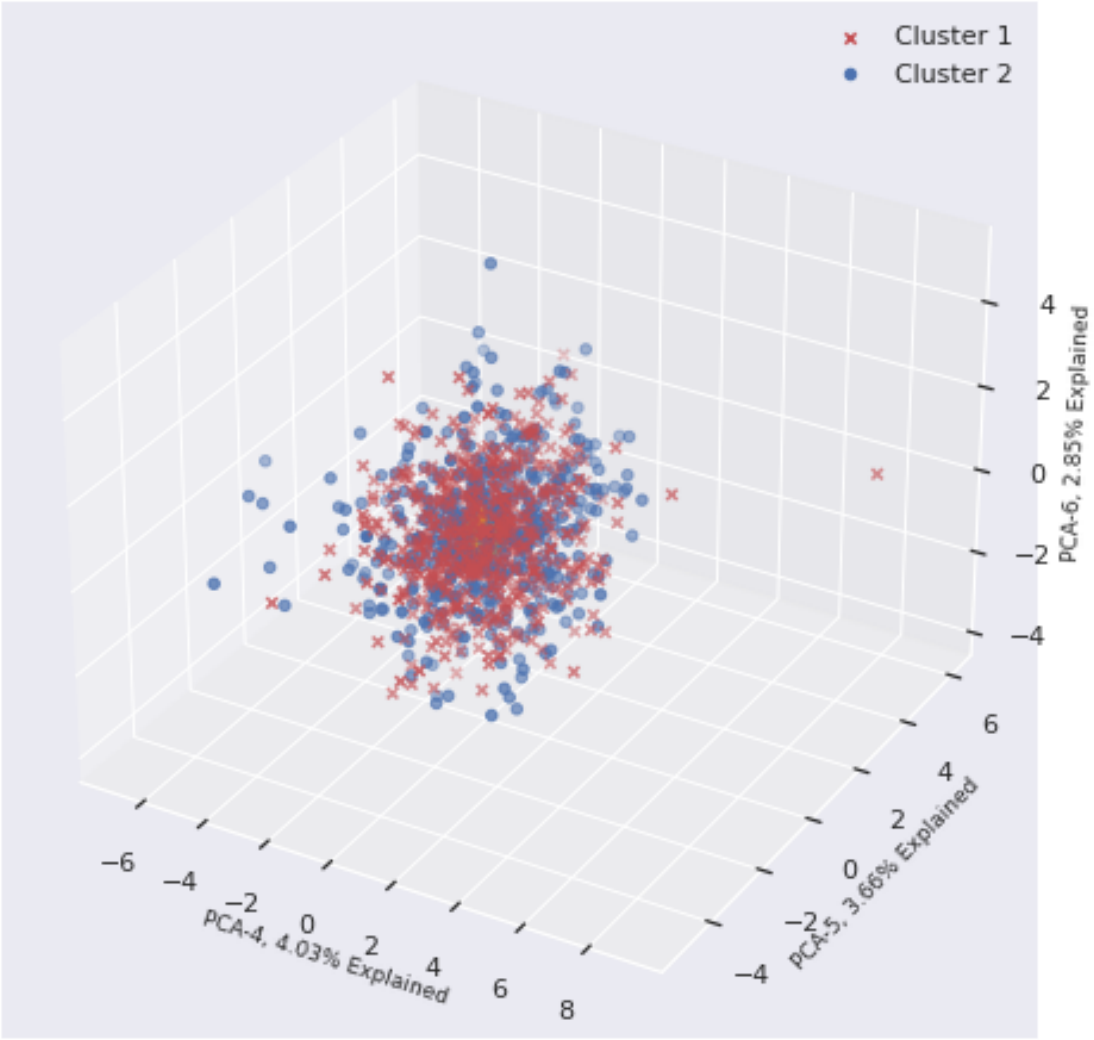
[ ]:
```
# We select only 2 clusters as we see uniform thickness and few possible␣
↪misclassifications (due to negative scores)

kmeans_pca = KMeans(n_clusters=2, init='k-means++', random_state=42,␣
↪max_iter=800)
kmeans_pca.fit(rotatedData)
df_pca_kmeans = pd.concat(
    [data.reset_index(drop=False), pd.DataFrame(rotatedData)], axis=1)

df_pca_kmeans.rename(columns={0: 'PC1', 1: 'PC2', 2: 'PC3', 3: 'PC4', 4: 'PC5',
                              5: 'PC6', 6: 'PC7', 7: 'PC8'}, inplace=True)
df_pca_kmeans['Cluster for KMeans PCA'] = kmeans_pca.labels_
df_pca_kmeans['Cluster'] = df_pca_kmeans['Cluster for KMeans PCA'].map({
    0: 'Cluster 1', 1: 'Cluster 2'})

# Analyze relationship between first 3 PCs - not much total variance explained␣
↪(~25%)
fig4 = plt.figure(figsize=(10, 8))
ax = fig4.add_subplot(111, projection='3d')
x_axis1 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 1', 'PC1']
y_axis1 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 1', 'PC2']
z_axis1 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 1', 'PC3']
x_axis2 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 2', 'PC1']
y_axis2 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 2', 'PC2']
z_axis2 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 2', 'PC3']
# We will then label the three axes using the percentages explained for each␣
↪major component.
ax.set_xlabel(
    'PCA-1, ' + str(round(pca.explained_variance_ratio_[0]*100, 2)) + '%␣
↪Explained', fontsize=9)
ax.set_ylabel(
    'PCA-2, ' + str(round(pca.explained_variance_ratio_[1]*100, 2)) + '%␣
↪Explained', fontsize=9)
ax.set_zlabel(
    'PCA-3, ' + str(round(pca.explained_variance_ratio_[2]*100, 2)) + '%␣
↪Explained', fontsize=9)
ax.scatter(x_axis1, y_axis1, z_axis1, marker='x', color='r')
ax.scatter(x_axis2, y_axis2, z_axis2, marker='o', color='b')
```

```
ax.scatter(kmeans_pca.cluster_centers_[:, 0], kmeans_pca.cluster_centers_[
           :, 1], kmeans_pca.cluster_centers_[:, 2], s=400, c='yellow',␣
 ↪label='Centroids')
plt.legend(labels=['Cluster 1', 'Cluster 2'])
plt.savefig("pc123.png")
plt.show()

# PCA 4,5,6
fig5 = plt.figure(figsize=(10, 8))
ax = fig5.add_subplot(111, projection='3d')
x_axis1 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 1', 'PC4']
y_axis1 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 1', 'PC5']
z_axis1 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 1', 'PC6']
x_axis2 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 2', 'PC4']
y_axis2 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 2', 'PC5']
z_axis2 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                            == 'Cluster 2', 'PC6']
# We will then label the three axes using the percentages explained for each␣
 ↪major component.
ax.set_xlabel(
    'PCA-4, ' + str(round(pca.explained_variance_ratio_[3]*100, 2)) + '%␣
 ↪Explained', fontsize=9)
ax.set_ylabel(
    'PCA-5, ' + str(round(pca.explained_variance_ratio_[4]*100, 2)) + '%␣
 ↪Explained', fontsize=9)
ax.set_zlabel(
    'PCA-6, ' + str(round(pca.explained_variance_ratio_[5]*100, 2)) + '%␣
 ↪Explained', fontsize=9)
ax.scatter(x_axis1, y_axis1, z_axis1, marker='x', color='r')
ax.scatter(x_axis2, y_axis2, z_axis2, marker='o', color='b')
ax.scatter(kmeans_pca.cluster_centers_[:, 3], kmeans_pca.cluster_centers_[
           :, 4], kmeans_pca.cluster_centers_[:, 5], s=400, c='yellow',␣
 ↪label='Centroids')
plt.legend(labels=['Cluster 1', 'Cluster 2'])
plt.savefig("pc456.png")
plt.show()

# PCA 6,7,8
fig5 = plt.figure(figsize=(10, 8))
ax = fig5.add_subplot(111, projection='3d')
x_axis1 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
```

```
                              == 'Cluster 1', 'PC6']
y_axis1 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                              == 'Cluster 1', 'PC7']
z_axis1 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                              == 'Cluster 1', 'PC8']
x_axis2 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                              == 'Cluster 2', 'PC6']
y_axis2 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                              == 'Cluster 2', 'PC7']
z_axis2 = df_pca_kmeans.loc[df_pca_kmeans['Cluster']
                              == 'Cluster 2', 'PC8']
# We will then label the three axes using the percentages explained for each␣
 ↪major component.
ax.set_xlabel(
    'PCA-4, ' + str(round(pca.explained_variance_ratio_[5]*100, 2)) + '%␣
 ↪Explained', fontsize=9)
ax.set_ylabel(
    'PCA-5, ' + str(round(pca.explained_variance_ratio_[6]*100, 2)) + '%␣
 ↪Explained', fontsize=9)
ax.set_zlabel(
    'PCA-6, ' + str(round(pca.explained_variance_ratio_[7]*100, 2)) + '%␣
 ↪Explained', fontsize=9)
ax.scatter(x_axis1, y_axis1, z_axis1, marker='x', color='r')
ax.scatter(x_axis2, y_axis2, z_axis2, marker='o', color='b')
ax.scatter(kmeans_pca.cluster_centers_[:, 5], kmeans_pca.cluster_centers_[
           :, 6], kmeans_pca.cluster_centers_[:, 7], s=400, c='yellow',␣
 ↪label='Centroids')
plt.legend(labels=['Cluster 1', 'Cluster 2'])
plt.savefig("pc678.png")
plt.show()
```


Cluster 1
Cluster 2
PCA-1, 12.63% Explained
PCA-2, 8.55% Explained
PCA-3, 6.84% Explained
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
−4
−2
0
2
4
6


Cluster 1
Cluster 2
PCA-4, 5.98% Explained
PCA-5, 5.28% Explained
PCA-6, 4.03% Explained



1.1.9 Results and Interpretation

Though the first 3 principal components exlained roughly 26% of the data, they show the greatest cluster separation using kMeans along the principal components eigenvector axes (directions of greatest variance). The centroids are colored in yellow. Both cluster centroids are roughly adjacent for the other PCA plots, suggesting that misclassification portion we saw from the previous score plot (the negative silhouette scores) come from components explaining relatively little of the data. This would prompt us to reconsider how many factors we should consider without making our model overly complex in future iterations. We can also decide to use another clustering algoirthm in future work to compare against kMeans performance.

1.1.10 Use these principal components and/or clusters identified (8) to build a classification model for logistic regression, where you predict the movie ratings of all movies from the personality factors identified before. We use cross-validation methods to avoid overfitting and state the accuracy of your model by its AUC.

[ ]:
```
pc_factors = ['Is full of energy', 'The emotions on the screen "rub off" on me␣
 ↪- for instance if something sad is happening I get sad or if something␣
 ↪frightening is happening I get scared',
              'Tends to be quiet', 'Perseveres until the task is finished', 'Is␣
 ↪original/comes up with new ideas', 'Has few artistic interests', 'I have␣
 ↪trouble following the story of a movie',
              'As a movie unfolds I start to have problems keeping track of␣
 ↪events that happened earlier']

idx = [i+420 for i, val in enumerate(factors) if val in pc_factors]
X = usrData.T.iloc[:, idx]
X_np = X.values
y_np = df_pca_kmeans['Cluster for KMeans PCA'].values

# see: https://stackoverflow.com/questions/42305862/
 ↪logistic-regression-and-cross-validation-in-python-with-sklearn

logreg = LogisticRegression(penalty='l2')
# Scores does the following:
# 1. Get the data and estimator (logreg, X_train, y_train)
# 2. From here on, we will use X_train as X_cv and y_train as y_cv (because␣
 ↪cross_val_predict doesnt know that its our training data) - Doubts??
# 3. Split X_cv, y_cv into X_cv_train, X_cv_test, y_cv_train, y_cv_test by␣
 ↪using its internal cv
# 4. Use X_cv_train, y_cv_train for fitting 'logreg'
# 5. y_cv_pred = logreg.predict(X_cv_test) and calculate accuracy with␣
 ↪y_cv_test.
# 6. Repeat steps 1 to 5 for cv_iterations = 10 (KFolds w/shuffling)
# 7. Return array of accuracies calculated in step 5.
X_s, y_s = shuffle(X_np, y_np)
scores = model_selection.cross_val_score(
    logreg, X_s, y_s, cv=10, scoring='roc_auc')

# Find out average of returned accuracies to see the model performance
scores = scores.mean()
print(scores)
```

```
0.8921672648720506
```

1.1.11 Result and Interpretation

The PCA and kMeans data was used to come up with classifier predictions for the movie data and was able to discern them fairly well using an L2 penalty term for logistic regression. The AUC

came out to around 0.89.

1.1.12 Create a neural network model of your choice to predict movie ratings, using information from all 477 columns.

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X_np, y_np, random_state=42)
# Multi-layer Perceptron is sensitive to feature scaling, so scale data
scaler = MinMaxScaler()
X_train_norm = scaler.fit_transform(X_train)
X_test_norm = scaler.fit_transform(X_test)
regr = MLPRegressor(random_state=42, max_iter=40000).fit(X_train_norm, y_train)
# These are scaled y_pred
y_predictions = regr.predict(X_test_norm)
# COD for predict(X) wrt. y.
R2_NN = regr.score(X_test_norm, y_test)
print(R2_NN)
```

```
0.4551253320553621
```

1.1.13 Results and Interpretation

We use a Multi-layer perceptron model (uses backprop). This regressor model optimizes the squared error using LBFGS or stochastic gradient descent. We scale the data as NN tend to be sensitive to feature scaling. The coefficient of determination of 0.45 makes this a poor predictive model with the current setup and more care should go into NN tuning in the future.

Reference: https://scikit-learn.org/stable/modules/neural_networks_supervised.html