0.1 Question 1: Using Linear Algebra for Optimization

In recommender system module, low-rank matrix factorization was used to execute latent factor modeling of movie ratings data.

Specifically, we calculated matrices U and V to solve the following optimization problem (if all ratings were given):

$$\min_{U,V} f(U,V) = \min_{U,V} \|R - VU^T\|_F^2 = \min_{U,V} \left\{ \sum_{m=1}^{M} \sum_{i=1}^{I} I_{mi}(r_{mi} - v_m u_i^T)^2 \right\},$$

where

$$I_{mi} = \begin{cases} 1, \text{ if } r_{mi} \text{ is observed} \\ 0, \text{ if } r_{mi} \text{ is missing.} \end{cases}$$

The best U and V were calculated iteratively by improving on current estimates:

$$\begin{aligned} u_i^{\text{new}} &= u_i + 2\alpha(r_{mi} - v_m u_i^T) \cdot v_m \\ v_m^{\text{new}} &= v_m + 2\alpha(r_{mi} - v_m u_i^T) \cdot u_i, \end{aligned}$$

where α is the step-size that is to be chosen by the user. (We won't discuss the role in this class, but treat it as an arbitrary, but given, parameter)

We can make calculating the updates more efficient by calculating them with matrix operations. For example, instead of calculating each deviation $\gamma_{mi} = r_{mi} - v_m u_i^T$ separately for all $m = 1, 2, \dots, M$ and $i = 1, 2, \dots, I$, matrix Γ of all deviations can be computed together using matrix operation *(verify for yourself)*:

$$\Gamma = R - VU^T$$

Similarly, updating U and V can be combined into matrix calculations which makes the optimization procedure more efficient.

First, note that updates for u_i, $i = 1, 2, \dots, I$ can be rewritten as

$$\begin{aligned} u_1^{\text{new}} &= u_1 + 2\alpha\gamma_{m1} \cdot v_m \\ u_2^{\text{new}} &= u_2 + 2\alpha\gamma_{m2} \cdot v_m \\ \vdots & \qquad \vdots \\ u_I^{\text{new}} &= u_I + 2\alpha\gamma_{mI} \cdot v_m. \end{aligned}$$

Stacking all I equations into a matrix form,

$$U^{\text{new}} = U + 2\alpha\Gamma_{m-}^T v_m,$$

where Γ_{m-} is the m-th row of Γ (use the notation Γ_{-i} for the i-th column). When evaluating U^{new}, the latest updated values of U, V, and Γ are used.

Note that there are M such update equations (one for each $m = 1, 2, \dots, M$) that can also be combined into one matrix update equation involving matrices U, V, Γ and scalars. As stated earlier, since α is assumed to be an arbitrary step-size parameter, we can replace α/M with α.

0.1.1 Question 1a: Using Linear Algebra for Optimization

Complete the following update equations:

$$U^{\text{new}} = U + 2\alpha[\text{some function of } \Gamma][\text{some function of } V]$$
$$V^{\text{new}} = V + 2\alpha[\text{some function of } \Gamma][\text{some function of } U]$$

SOLUTION

$$U^{\text{new}} = U + 2\alpha\Gamma^T V$$
$$V^{\text{new}} = V + 2\alpha\Gamma U$$

0.1.2 Question 1d: Interpret Diagnostic Plots

Following figures tell us if the optimization algorithm is working properly.

```
In [17]: import altair as alt
         logscale = alt.Scale(type='log', base=10)
         fig_rmse = \
             alt.Chart(output1['rmse'])\
             .mark_line()\
             .encode(
                 x='iteration:Q',
                 y=alt.Y('rmse:Q', scale=logscale)
             )
         fig_max_residual_change = \
             alt.Chart(output1['rmse'])\
             .mark_line()\
             .encode(
                 x='iteration:Q',
                 y=alt.Y('max residual change:Q', scale=logscale)
             )
         fig_updates = \
             alt.Chart(output1['update'])\
             .mark_line()\
             .encode(
                 x='iteration:Q',
                 y=alt.Y('max update:Q', scale=logscale)
             )
         alt.vconcat(
             fig_rmse | fig_max_residual_change,
             fig_updates
         )
```

```
Out[17]: alt.VConcatChart(...)
```

By referring back to the function used to calculate the quantities in each figure, describe what each figure is showing and interpret the behavior of the optimization algorithm.

SOLUTION

RMSE (Root Mean Square Error) Plot: This plot displays the root mean square error of the residuals (the differences between the predicted and actual ratings) on a logarithmic scale.We can observe a steady decrease in RMSE as the number of iterations increases, which indicates that the optimization algorithm is effectively minimizing the error in the predictions. The RMSE is converging towards a lower value, suggesting that the model is learning and improving its predictions over time. ##### Maximum Residual Change Plot: This plot illustrates the maximum change in the residuals from one iteration to the next, again on a logarithmic scale. The plot shows a sharp decrease initially, which levels off as the iterations progress.

This indicates that the biggest improvements in the model's predictions occur in the initial iterations, and subsequent updates result in progressively smaller changes to the residuals. The algorithm is stabilizing as the updates are having less impact on the residuals. ##### Maximum Update Plot: This plot shows the maximum update to the U and V. matrices at each iteration, also on a logarithmic scale. This is the maximum absolute change in the values of U and V after an iteration of updates. The maximum update values are quite high at the beginning and decrease rapidly, which is expected as the matrices start to converge towards their optimal values. However, there are spikes in the max update values at certain iterations. These spikes may indicate that the algorithm is exploring different parts of the solution space, possibly due to the non-convex nature of the optimization problem. Over time, the updates should become smaller, indicating that the matrices are nearing their optimal values. If the spikes continue without a clear downward trend, it may suggest that the algorithm is not stable or that the learning rate α needs to be adjusted.

0.1.3 Question 1e: Analyze Large Dataset

Following code will analyze a larger dataset:

```
In [18]: # run on larger dataset: ratings for 100 movies
         Rbig = pd.read_pickle('data/ratings_stacked.pkl').unstack().iloc[:100]

         np.random.seed(14) # set seed for tests
         output3 = compute_UV(Rbig, K=5, alpha=0.001, max_iteration=500)

         Rhatbig = output3['V']@output3['U'].T
```

```
In [19]: fit_vs_obs = pd.concat([
             Rhatbig.rename(columns={'rating':'fit'}),
             Rbig.rename(columns={'rating':'observed'}),
         ], axis=1).stack().dropna().reset_index()[['fit','observed']]

         fit_vs_obs = fit_vs_obs.iloc[np.random.choice(len(fit_vs_obs), 5000)]

         alt.Chart(fit_vs_obs).transform_density(
             density='fit',
             bandwidth=0.01,
             groupby=['observed'],
             extent= [0, 6]
         ).mark_bar().encode(
             alt.X('value:Q'),
             alt.Y('density:Q'),
             alt.Row('observed:N')
         ).properties(width=800, height=50)
```

```
Out[19]: alt.Chart(...)
```

Consider the above plot. By reading the code, comment on what the plot is illustrating. What happens when you add `counts=True` to `transform_density`? What can you conclude?

SOLUTION

The plot visualizes the distribution of predicted ratings (fit) for each actual observed rating (observed). Each row in the chart corresponds to an observed rating value (from 1 to 5), and the horizontal bars represent the distribution of predicted ratings for that observed value. The `value` on the x-axis refers to the predicted rating, while the `density` on the y-axis represents the probability density of the predictions.

The transform_density in the Altair chart is used to estimate the probability density function of the predicted ratings for each observed rating value. It gives us a visual estimate of how often each predicted rating occurs for each actual rating.

Each bar's width represents a range of predicted ratings, and the height represents the estimated density of predictions within that range. The extent `[0, 6]` in the transform_density function limits the range of the predicted values visualized, even though the actual predictions may not necessarily fall within this range.

When `counts=True` is added to the `transform_density`, the chart will display the actual number of observations in each bin rather than the probability density. This would change the y-axis to represent counts, which can be useful for understanding the absolute number of data points contributing to each bin, providing a more granular view of the distribution.

In the provided plot, the distributions are relatively broad, and the peaks do not always align with the diagonal, indicating that while there is some correlation between observed and predicted ratings, there is also a significant variance. The model does not always predict the exact observed rating but rather a range of ratings around the observed rating. This spread in the distribution of predicted ratings suggests that the model might have limitations in capturing all the nuances of the data.

0.1.4 Question 1f: Make Recommendation

What movies would you recommend to `user id` 601? Do you see any similarities to movies the user rated high?

SOLUTION

```
In [20]: Rhatbig.iloc[:,600].head().sort_values(ascending=False).head()
```

```
Out[20]: movie id  movie title
         3         Four Rooms (1995)    2.984696
         4         Get Shorty (1995)    2.634451
         1         Toy Story (1995)     2.119271
         5         Copycat (1995)       2.079726
         2         GoldenEye (1995)     1.911324
         Name: (rating, 601), dtype: float64
```

The movies that the user rated high are all from 1995, and the user's highest rating is below 3.

0.1.5 Question 2a: Derive New Gradients and Update Rules

Based on the new objective function $g(U, V)$, derive its gradients and update rules for U^{new} and V^{new}.

SOLUTION

In a vector form, gradients are

$$\frac{\partial}{\partial u_i} g(u_i, v_m) = -2 \sum_{m=1}^{M} I_{mi}(r_{mi} - v_m u_i^T) v_m + 2\lambda u_i$$
$$\frac{\partial}{\partial v_m} g(u_i, v_m) = -2 \sum_{i=1}^{I} I_{mi}(r_{mi} - v_m u_i^T) u_i + 2\lambda v_m$$

By stacking equations, gradients can be expressed in a matrix form as

$$\frac{\partial}{\partial U} g(U, V) = -2\Gamma^T V + 2\lambda U$$
$$\frac{\partial}{\partial V} g(U, V) = -2\Gamma U + 2\lambda V$$

Finally, the update rules are (in a matrix form)

$$U^{\text{new}} = U + 2\alpha(\Gamma^T V - \lambda U)$$
$$V^{\text{new}} = V + 2\alpha(\Gamma U - \lambda V)$$

0.1.6 Question 2d: Investigating the Effects of Regularization

Adding the regularization terms to the objective function will affect the estimates of U and V. Here, we consider comparing the user matrix U.

Using the dataset *Rsmall*, obtain two estimated user matrices, say $\hat{U}$ for a non-regularized model and $\hat{U}_{\text{reg}}$ for a regularized model. Select $K = 20$ and $\lambda = 5$. Come up with an effective visualization for comparing $\hat{U}$ and $\hat{U}_{\text{reg}}$, and describe any differences you notice. Additionally, analyze whether the observed differences in patterns align with the concept of regularization.

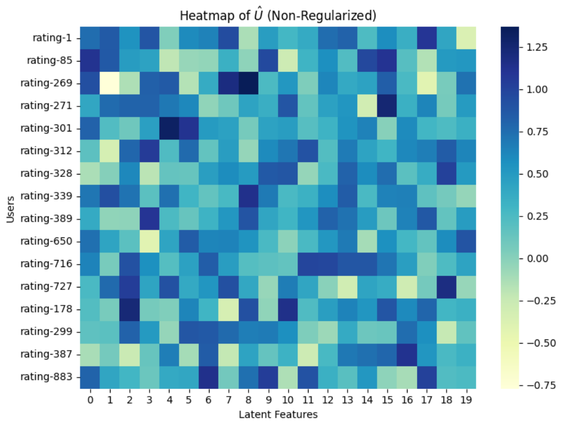
Provide reasoning supported by evidence, such as code implementation and results.

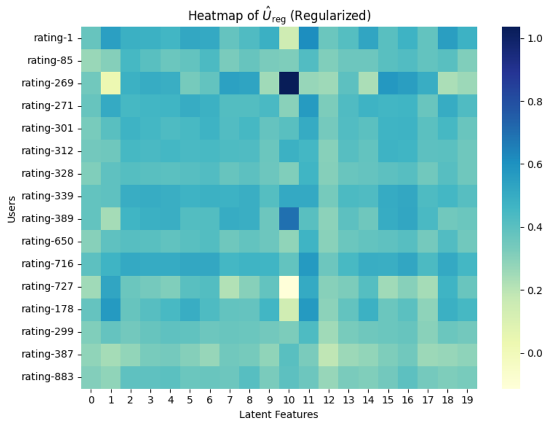
SOLUTION

```
In [25]: np.random.seed(134) # set seed for tests
         output_noreg = compute_UV(Rsmall, K=10, alpha=0.001)
         output_reg = compute_UV_reg(Rsmall, K=10, lam=1.0, alpha=0.001)
```

```
In [26]: output_noreg = compute_UV(Rsmall, K=20, alpha=0.001)
         output_reg = compute_UV_reg(Rsmall, K=20, lam=5, alpha=0.001)

         U_noreg = output_noreg['U']
         U_reg = output_reg['U']
```

```
In [27]: import matplotlib.pyplot as plt
         import seaborn as sns

         # Creating subplots
         fig, axes = plt.subplots(1, 2, figsize=(16, 6))

         # Plotting heatmap for non-regularized U matrix
         sns.heatmap(U_noreg, ax=axes[0], cmap="YlGnBu", cbar=True)
         axes[0].set_title(r'Heatmap of $\hat{U}$ (Non-Regularized)')

         # Plotting heatmap for regularized U matrix
         sns.heatmap(U_reg, ax=axes[1], cmap="YlGnBu", cbar=True)
         axes[1].set_title(r'Heatmap of $\hat{U}_{\text{reg}}$ (Regularized)')

         # Setting labels
         for ax in axes:
             ax.set_xlabel('Latent Features')
             ax.set_ylabel('Users')

         plt.tight_layout()
         plt.show()
```





The produced heatmaps provide a visual comparison between the non-regularized and regularized user matrices, denoted as $\hat{U}$ and $\hat{U}_{\text{reg}}$, respectively. Each row represents a user, each column represents a latent feature, and the color intensity indicates the magnitude of the feature value for that user.

The heatmap of the regularized matrix $\hat{U}_{\text{reg}}$ shows lighter shades overall, indicating lower value magnitudes due to the effect of regularization. This suggests that the regularization process successfully constrains the feature values, which is expected to help in mitigating overfitting.

There is a noticeable uniformity in the feature values across different users in the regularized matrix, implying a reduction in the noise and potential overfitting present in the non-regularized matrix $\hat{U}$.

The regularization introduces a bias toward smaller values, which tends to produce a model that generalizes better to unseen data, aligning with the theoretical benefits of regularization in machine learning.

The comparison indicates that regularization has the anticipated impact on the user matrix, leading to a model that potentially offers a more generalized and robust representation of user preferences, essential for effective recommendations.

0.1.7 Question 2e: Practical Aspects

In the previous question, a specific values for K and λ were provided. Now, try applying various K's and λ's. Specifically, try the following:

- While keeping K constant, experiment with different values of λ. What do you notice? Why do you think this happens?
- While keeping λ constant, experiment with different values of K. What do you notice? Why do you think this happens?

If your optimization algorithm is correctly implemented, you will notice that the choice of K and λ has a significant impact on the final estimates. Hence, selecting appropriate values for K and λ is crucial when applying the recommendation algorithm in practice. As a practitioner, how would you approach choosing K and λ?

Provide reasoning supported by evidence, such as code implementation and results.

SOLUTION

```
In [28]: # Experimenting with different values of lambda while keeping K constant

         # Setting the values of lambda and K
         lambdas = [0.1, 1, 5, 10]
         K_constant = 10

         # Running the model for different values of lambda
         rmse_lambda = {}
         for lam in lambdas:
             output = compute_UV_reg(Rsmall, K=K_constant, lam=lam, alpha=0.001)
             rmse_lambda[lam] = output['rmse'].iloc[-1]['rmse']

         # Displaying the final RMSE for each lambda
         rmse_lambda
```

```
Out[28]: {0.1: 0.03037088710692119,
          1: 0.2563644087142672,
          5: 0.913801915700897,
          10: 1.2973255720094965}
```

These results align with the expectations of how regularization affects a model. As λ increases, the regularization term becomes more dominant in the objective function, leading to a stronger penalization of large values in the matrices U and V. This typically results in a more generalized model but can increase the RMSE, as the model becomes less flexible to fit the complex patterns in the data.

```
In [29]: # Setting the values of K and lambda
         K_values = [5, 10, 20, 40]
         lambda_constant = 1

         # Running the model for different values of K
         U_matrices_K = {}
         rmse_K = {}
         for K in K_values:
             output = compute_UV_reg(Rsmall, K=K, lam=lambda_constant, alpha=0.001)
             U_matrices_K[K] = output['U']
             rmse_K[K] = output['rmse'].iloc[-1]['rmse']

         rmse_K
```

```
Out[29]: {5: 0.36536508102715937,
          10: 0.25733731068405513,
          20: 0.2563171479045487,
          40: 0.2561679334215519}
```

These results indicate that increasing the number of latent features (K) has a relatively minor impact on RMSE, especially when compared to the effect of varying λ. First, increasing K enhances the model's capacity to capture more complex patterns. However, beyond a certain point, additional latent features may not contribute significantly to capturing additional useful information, especially if the underlying structure of the data is not highly complex. In addition, with more latent features, there's a risk of overfitting, but since λ is set to a moderate value, it helps in regularizing the model, thus balancing between model complexity and overfitting. Last but not least, The characteristics of the `Rsmall` dataset may also play a role. If the dataset doesn't contain very complex user-item interactions, then increasing K might not lead to substantial improvement.

0.1.8 Question 3a: Concatenate matrix factors and cluster

Entries in either matrix factors are just points in k-dimensional latent variable space. We will use both U and V for segmentation by combining them into one large clustering problem.

Once clusters are identified, you will qualitatively inspect the users and movies in the cluster and decide on a "representative" movie from each cluster.

Consider concatenating U and V into one large matrix. Since these matrices have arbitrary scaling, it would be a good idea to standardize the columns before concatenating them. Standardize U and V separately, then concatenate with numpy's `concatenate` method. Call this concatenated matrix, `UVstd`.

Apply hierarchical and K-means clustering methods on `UVstd`. For each clustering method, identify 5 clusters. Compare the clustering results by applying three different cluster validation metrics to evaluate the clustering performance.

Which cluster performance metrics can you use? Do we have true labels? Does one performance metric seem to clearly be better than another? Why would you choose one metric over another? What interpretation, if any, does each metric have in the context of our problem? Explain.

SOLUTION

```
In [30]: from sklearn.preprocessing import StandardScaler
         from sklearn.cluster import KMeans, AgglomerativeClustering
         from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score

         U_reg_latest = output_reg['U']
         V_reg_latest = output_reg['V']

         scaler_U = StandardScaler()
         U_std_latest = scaler_U.fit_transform(U_reg_latest)

         scaler_V = StandardScaler()
         V_std_latest = scaler_V.fit_transform(V_reg_latest)

         # Concatenating U and V
         UVstd = np.concatenate((U_std_latest, V_std_latest), axis=0)

         # Applying Hierarchical and K-means Clustering
         kmeans_latest = KMeans(n_clusters=5, random_state=42, n_init=1000).fit(UVstd)
         hierarchical_latest = AgglomerativeClustering(n_clusters=5).fit(UVstd)

         # Extracting cluster labels
         labels_kmeans_latest = kmeans_latest.labels_
         labels_hierarchical_latest = hierarchical_latest.labels_

         # Computing cluster validation metrics
         silhouette_kmeans_latest = silhouette_score(UVstd, labels_kmeans_latest)
```

```python
silhouette_hierarchical_latest = silhouette_score(UVstd, labels_hierarchical_latest)

calinski_harabasz_kmeans_latest = calinski_harabasz_score(UVstd, labels_kmeans_latest)
calinski_harabasz_hierarchical_latest = calinski_harabasz_score(UVstd, labels_hierarchical_lat

davies_bouldin_kmeans_latest = davies_bouldin_score(UVstd, labels_kmeans_latest)
davies_bouldin_hierarchical_latest = davies_bouldin_score(UVstd, labels_hierarchical_latest)

# Results
clustering_metrics_latest = {
    "Silhouette Score": {"K-Means": silhouette_kmeans_latest, "Hierarchical": silhouette_hiera
    "Calinski-Harabasz Index": {"K-Means": calinski_harabasz_kmeans_latest, "Hierarchical": ca
    "Davies-Bouldin Index": {"K-Means": davies_bouldin_kmeans_latest, "Hierarchical": davies_b
}

clustering_metrics_latest
```

Out[30]:
```
{'Silhouette Score': {'K-Means': 0.3362913820323857,
  'Hierarchical': 0.3199431666406211},
 'Calinski-Harabasz Index': {'K-Means': 19.23503697788666,
  'Hierarchical': 18.5590774408883},
 'Davies-Bouldin Index': {'K-Means': 0.9324789932973024,
  'Hierarchical': 0.9555143859502723}}
```

Cluster metric used: 1. Silhouette Score: Measures how similar an object is to its own cluster compared to other clusters. A high silhouette score indicates well-clustered data. It is useful when the cluster separation and cohesion are important. It gives an idea of how distinct the clusters are. For this data, both k-means and hierarchical methods have relatively low Silhouette Scores, suggesting that the clusters might not be highly distinct or well-separated. However, the Hierarchical method performs slightly better than K-Means in this aspect. 2. Calinski-Harabasz Index: Also known as the Variance Ratio Criterion, this metric evaluates clusters based on the mean between-cluster variance and the mean within-cluster variance. Higher values generally indicate better-defined clusters. It is beneficial when we have a relatively large dataset and need to understand the cluster's spread and separation. Under our context, the Calinski-Harabasz Index is higher when clusters are dense and well-separated, which relates to a higher value being better. Both methods have similar scores here, suggesting a moderate level of separation and density within the clusters. 3. Davies-Bouldin Index: This metric evaluates the average similarity between each cluster and the most similar one. Lower values indicate better clustering. It is good for identifying sets of clusters that are far apart and not overlapping much. After calculation, the Davies-Bouldin Index indicates the average 'similarity' between clusters, where lower values mean better clustering. Both methods have similar scores, indicating a moderate level of cluster quality, with the Hierarchical method performing slightly better.

We want each ad to appeal distinctly to a different segment of potential customers. Hence, metrics indicating well-separated and cohesive clusters (like Silhouette Score) might be more relevant. On the other hand, considering the computational efficiency and ease of interpretation, K-Means with a silhouette score might be a practical choice.

Given these results, we can deduce that while there is some level of clustering happening, it might not be very pronounced or clear-cut. This could be due to several factors, such as the inherent structure of the data, the choice of number of clusters, or the nature of the latent features derived from the matrix factorization.

0.1.9 Question 3b: Visualizing Clusters in Latent Space

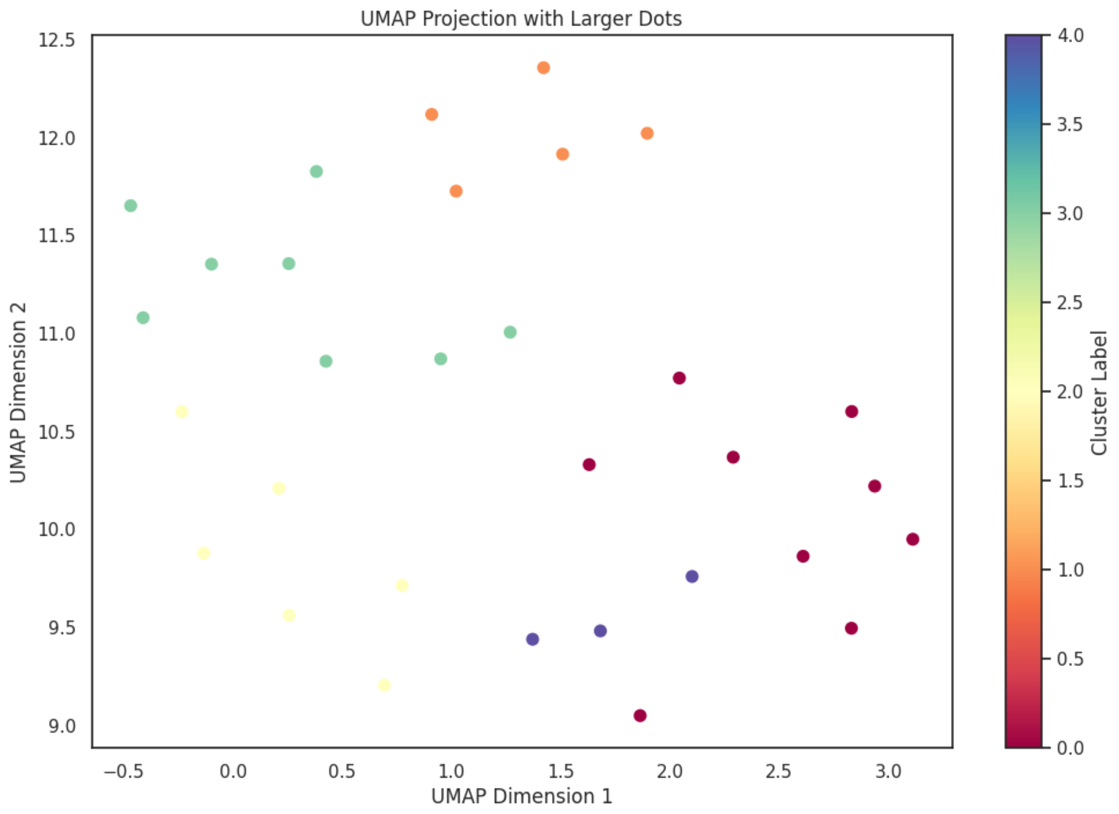
Select the clustering method based on the evaluation results in q3a and visualize the clusters using UMAP. Are the clusters and UMAP projection consistent?

SOLUTION

```
In [71]: # install umap
         # !pip install umap-learn
```

```
In [38]: !pip install umap-learn

         import umap
         import matplotlib.pyplot as plt

         reducer = umap.UMAP(n_neighbors=30, min_dist=0.1, n_jobs=-1)
         embedding = reducer.fit_transform(UVstd)

         # Visualize the clusters with larger dots
         plt.figure(figsize=(12, 8))
         scatter = plt.scatter(embedding[:, 0], embedding[:, 1], c=labels_hierarchical_latest, cmap='Sp
         plt.title('UMAP Projection with Larger Dots')
         plt.colorbar(scatter, label='Cluster Label')
         plt.xlabel('UMAP Dimension 1')
         plt.ylabel('UMAP Dimension 2')
         plt.show()
```

```
Collecting umap-learn
  Using cached umap_learn-0.5.5-py3-none-any.whl
Requirement already satisfied: numpy>=1.17 in /opt/conda/lib/python3.11/site-packages (from umap-learn)
Requirement already satisfied: scipy>=1.3.1 in /opt/conda/lib/python3.11/site-packages (from umap-learn
Requirement already satisfied: scikit-learn>=0.22 in /opt/conda/lib/python3.11/site-packages (from umap
Collecting numba>=0.51.2 (from umap-learn)
  Using cached numba-0.58.1-cp311-cp311-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (2.7 kB
Collecting pynndescent>=0.5 (from umap-learn)
  Using cached pynndescent-0.5.11-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.11/site-packages (from umap-learn) (4.66.
Collecting llvmlite<0.42,>=0.41.0dev0 (from numba>=0.51.2->umap-learn)
  Using cached llvmlite-0.41.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.8
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.11/site-packages (from pynndescen
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.11/site-packages (from sc
Using cached numba-0.58.1-cp311-cp311-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (3.6 MB)
Using cached pynndescent-0.5.11-py3-none-any.whl (55 kB)
Using cached llvmlite-0.41.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (43.6 MB)
Installing collected packages: llvmlite, numba, pynndescent, umap-learn
Successfully installed llvmlite-0.41.1 numba-0.58.1 pynndescent-0.5.11 umap-learn-0.5.5
2023-12-06 12:02:20.854950: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to re
2023-12-06 12:02:20.854987: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to reg
2023-12-06 12:02:20.855706: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to r
2023-12-06 12:02:20.860384: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropr
2023-12-06 12:02:21.680190: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Coul
```



As shown in the scatter plot, the clusters appear to have some space between them, implying that they are somewhat distinct from one another in the reduced dimensional space. Some clusters seem to have points that are closer together, indicating a degree of cohesion within those clusters. However, other clusters appear more spread out, which could imply that the cohesion within those groups is weaker. Nevertheless, there are some points that lie far from any central grouping, which could be considered outliers. This is normal in most datasets, but if there are many outliers, it may indicate that the clustering or the dimensionality reduction hasn't fully captured the structure of the data.

In summary, the clusters appear to have a degree of consistency, but they are not highly distinct. There is visual separation between some of the clusters, yet the boundaries between them are not sharply defined. This indicates that while there is some level of grouping present, the clusters are not completely isolated from each other.

0.1.10 (PSTAT 234) Question 3c: Making decisions

To make actionable decisions, there are practical considerations to take into account.

1. How will you choose a “representative” movie from each cluster?
2. How many of each poster do you estimate you will need? Assume the ad campaign will serve 10 million users and 0.01% people will respond. What other assumption do you need to make?
3. Which clustering method will you use as the final method?

SOLUTION

0.2 (PSTAT 234) Question 4: Improving the Model

0.2.1 Question 4a: Logistic function

Note the reconstructed ratings can be smaller than 1 and greater than 5. To confine ratings to between the allowed range, we can use the logistic function. Logistic function is defined as

$$h(x) = \frac{1}{1+e^{-x}}.$$

It is straightforward to show the derivative is

$$h'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = h(x)(1-h(x)).$$

Therefore, we can rescale the ratings from $r_{mi} \in [1,5]$ to $r_{mi} \in [0,1]$. Then, we can find the best U and V to optimize the following:

$$\min_{U,V} \|R - h(VU^T)\|_F^2 = \sum_{m,i} I_{mi}(r_{mi} - h(v_m u_i^T))^2,$$

where function h is applied elementwise and

$$I_{mi} = \begin{cases} 1, \text{ if } r_{mi} \text{ is observed} \\ 0, \text{ if } r_{mi} \text{ is missing.} \end{cases}$$

Derive new update expressions for the new objective function.

SOLUTION

0.2.2 Question 4c: Analyze a Large Dataset

Following code will analyze a larger dataset:

```
In [ ]: # run on larger dataset: ratings for 100 movies
        Rbig = pd.read_pickle('data/ratings_stacked.pkl').unstack().iloc[:100]

        np.random.seed(14) # set seed for tests
        output6 = compute_logistic_UV(Rbig, K=5, alpha=0.05, max_iteration=500)

        Rhatbig = logistic_rating(output6['U'], output6['V'])
```

```
In [ ]: fit_vs_obs_2 = pd.concat([
            Rhatbig.rename(columns={'rating':'fit'}),
            Rbig.rename(columns={'rating':'observed'}),
        ], axis=1).stack().dropna().reset_index()[['fit','observed']]

        fit_vs_obs_2 = fit_vs_obs_2.iloc[np.random.choice(len(fit_vs_obs_2), 5000)]

        alt.Chart(fit_vs_obs_2).transform_density(
            density='fit',
            bandwidth=0.01,
            groupby=['observed'],
            extent= [0, 6]
        ).mark_bar().encode(
            alt.X('value:Q'),
            alt.Y('density:Q'),
            alt.Row('observed:N')
        ).properties(width=800, height=50)
```

Consider the above plot. By reading the code, comment on what the plot is illustrating. How does this plot look different than part 1.e?

SOLUTION