project_134

March 19, 2024

[1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("project_134.ipynb")
```

1 Final Project

1.1 PSTAT 134 (Winter 2024)

1.2 Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the homework, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** at the top of your notebook.

Collaborators: *list collaborators here*

[2]:
```
%xmode Verbose
```

```
Exception reporting mode: Verbose
```

1.3 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):

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

where

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

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

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

where α is the step-size that is to be chosen by the user, $i = 1, 2, \ldots, I$, $m = 1, \ldots, M$. (We won't discuss the role of α 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 $m = 1, 2, \ldots, M$ and $i = 1, 2, \ldots, I$, 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, $i = 1, 2, \ldots, I$ can be rewritten as

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

Stacking all I equations into a matrix form,

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

where I_{m-} and Γ_{m-} are the m-th row of Γ and D (use the notation Γ_{-i} for the i-th column). Hadamard product (elementwise matrix product) is denoted with $\circ$. 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 $m = 1, 2, \ldots, M$) 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.

1.3.1 Question 1a: Using Linear Algebra for Optimization

Complete the following update equations:

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

SOLUTION

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

$V^{\text{new}} = V + 2\alpha(D \circ \Gamma)U$

1.3.2 Question 1b: Implementing Updates

In this problem, you will implement the updates calculated in the previous problem. Define the following three functions:

- `update_G(R, U, V)`: computes deviation $R - VU^T$
- `update_U(G, U, V, alpha=0.01)`: calculates update U^{new}
- `update_V(G, U, V, alpha=0.01)`: calculates update V^{new}

Each function should only be one line of matrix operations. Three functions is to be applied sequentially, using the most up-to-date estimates of G, U, and V.

Since some elements of `R` are `np.nan` for any missing ratings, `update_U` and `update_V` functions need to be adjusted by using `numpy.nan_to_num` function where appropriate. The function `numpy.nan_to_num` will let you replace `NaN` to some number, so that missing ratings do not interfere with updates.

[3]:
```
import numpy as np
import pandas as pd

def update_G(R_, U_, V_):
    return R_ - np.dot(V_, U_.T)

def update_U(G_, U_, V_, alpha=0.01):
    return U_ + 2 * alpha * np.dot(np.nan_to_num(G_).T, V_)

def update_V(G_, U_, V_, alpha=0.01):
    return V_ + 2 * alpha * np.dot(np.nan_to_num(G_), U_)

# small test to help debug (keep intact)
np.random.seed(1)
M_ = 5
I_ = 3
K_ = 2

R_ = np.random.rand(M_, I_).round(1)
R_[0, 0] = R_[3, 2] = np.nan
U_ = np.random.rand(I_, K_).round(1)
V_ = np.random.rand(M_, K_).round(1)
G_ = update_G(R_, U_, V_)
```

[4]:
```
grader.check("q1b")
```

[4]:
```
q1b results: All test cases passed!
```

1.3.3 Question 1c: Construct Optimization Algorithm

Combine the above functions to implement the optimization algorithm to iteratively compute U and V.

But, first, here are functions that will calculate RMSE and quantify the maximum update (in absolute value) made by `update_U` and `update_V` after they are called.

[5]:
```
def rmse(X):
    """
    Computes root-mean-square-error, ignoring nan values
    """
    return np.sqrt(np.nanmean(X**2))
```

```
def max_update(X, Y, relative=True):
    """
    Compute elementwise maximum update

    parameters:
    - X, Y: numpy arrays or vectors
    - relative: [True] compute relative magnitudes

    returns
    - maximum difference between X and Y (relative to Y)

    """
    if relative:
        updates = np.nan_to_num((X - Y)/Y)
    else:
        updates = np.nan_to_num(X - Y)

    return np.linalg.norm(updates.ravel(), np.inf)
```

A template for the optimization algorithm is given below. Fill-in the missing portions to complete the algorithm.

[6]:
```
def compute_UV(Rdf, K=5, alpha=0.01, max_iteration=5000, diff_thr=1e-3):

    R = Rdf.values
    Rone = pd.DataFrame().reindex_like(Rdf).replace(np.nan, 1) # keep data␣
 ↪frame metadata

    M, I = R.shape            # number of movies and users
    U = np.random.rand(I, K)  # initialize with random numbers
    V = np.random.rand(M, K)  # initialize with random numbers
    G = update_G(R, U, V)     # calculate residual

    track_rmse = []
    track_update = []
    for i in range(0, max_iteration):

        Unew = update_U(G,U,V, alpha)
        Gnew = update_G(R, Unew, V)

        Vnew = update_V(Gnew, Unew, V, alpha)
        Gnew = update_G(R, Unew, Vnew)

        track_rmse += [{
            'iteration':i,
            'rmse': rmse(Gnew),
```

```
            'max residual change': max_update(Gnew, G, relative=False)
        }]
        track_update += [{
            'iteration':i,
            'max update':max(max_update(Unew, U), max_update(Vnew, V))
        }]

        U = Unew
        V = Vnew
        G = Gnew

        if track_update[-1]['max update'] < diff_thr:
            break

    track_rmse = pd.DataFrame(track_rmse)
    track_update = pd.DataFrame(track_update)

    kindex = pd.Index(range(0, K), name='k')
    U = pd.DataFrame(U, index=Rdf.columns, columns=kindex)
    V = pd.DataFrame(V, index=Rdf.index, columns=kindex)

    return {
        'U':U, 'V':V,
        'rmse': track_rmse,
        'update': track_update
    }

Rsmall = pd.read_pickle('data/ratings_stacked_small.pkl').unstack()

np.random.seed(134) # set seed for tests
output1 = compute_UV(Rsmall, K=10, alpha=0.001)
```

[7]:
```
grader.check("q1c")
```

[7]:
```
q1c results: All test cases passed!
```

Running the function on a different sized problem to check if `compute_UV` adapts to changing problem sizes. There is nothing new to do here

[8]:
```
# These tests should pass if `compute_UV` works properly
np.random.seed(134) # set seed for tests
output2 = compute_UV(Rsmall.iloc[:7, :5], K=8)
```

[9]:
```
## TEST ##
output2['U'].shape
```

[9]:
```
(5, 8)
```

[10]:
```
## TEST ##
print((output2['V']@output2['U'].T).round(2))
```

```
                                    rating
user id                                  1     85    269   271   301
movie id movie title
132      Wizard of Oz, The (1939)     4.00  5.00  5.00  5.00  4.01
238      Raising Arizona (1987)       4.00  2.00  5.00  4.00  3.01
748      Saint, The (1997)            1.92  1.53  1.97  1.47  1.52
196      Dead Poets Society (1989)    5.00  4.00  1.00  4.00  4.00
197      Graduate, The (1967)         5.00  5.00  5.00  4.00  5.01
185      Psycho (1960)                4.00  3.67  5.00  3.00  3.80
194      Sting, The (1973)            4.01  4.01  5.00  5.00  3.99
```

[11]:
```
## TEST ##
output2['V'].shape
```

[11]:
```
(7, 8)
```

[12]:
```
## TEST ##
output2['U'].index
```

[12]:
```
MultiIndex([('rating',   1),
            ('rating',  85),
            ('rating', 269),
            ('rating', 271),
            ('rating', 301)],
           names=[None, 'user id'])
```

[13]:
```
## TEST ##
output2['V'].index
```

[13]:
```
MultiIndex([(132,  'Wizard of Oz, The (1939)'),
            (238,    'Raising Arizona (1987)'),
            (748,         'Saint, The (1997)'),
            (196, 'Dead Poets Society (1989)'),
            (197,      'Graduate, The (1967)'),
            (185,             'Psycho (1960)'),
            (194,         'Sting, The (1973)')],
           names=['movie id', 'movie title'])
```

[14]:
```
## TEST ##
output2['U'].columns
```

[14]:
```
RangeIndex(start=0, stop=8, step=1, name='k')
```

[15]:
```
## TEST ##
output2['V'].columns
```

[15]:
```
RangeIndex(start=0, stop=8, step=1, name='k')
```

1.3.4 Question 1d: Interpret Diagnostic Plots

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

[16]:
```
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
)
```

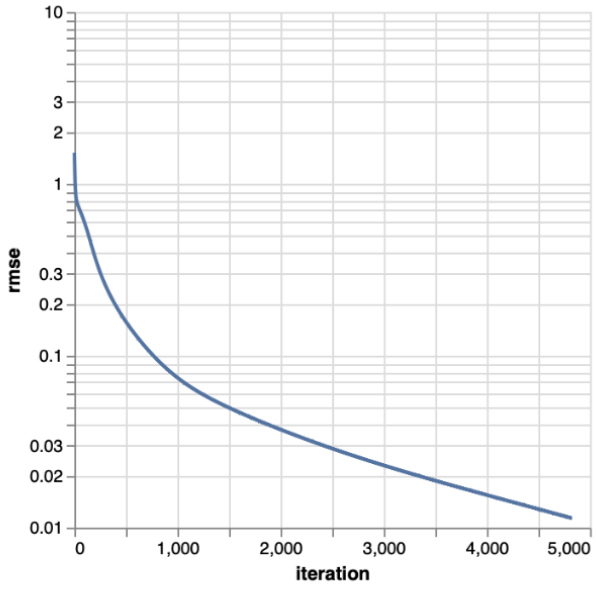
[16]:
```
alt.VConcatChart(…)
```

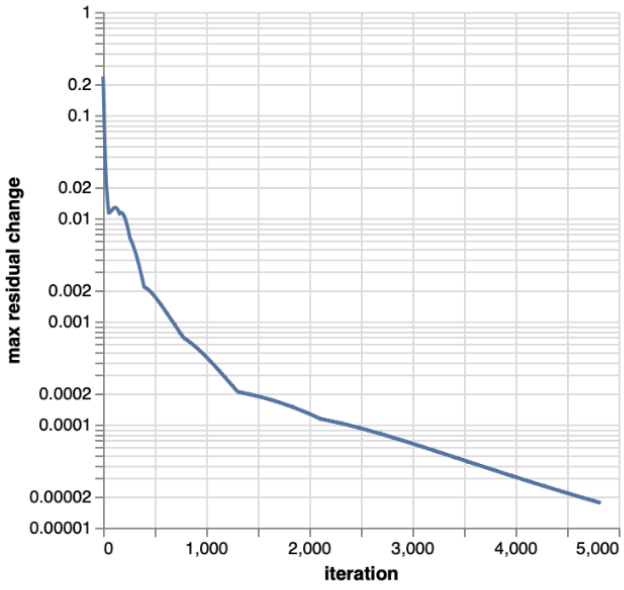
By referring back to the function used to calculate the quantities in each figure, describe what each figure is showing and interpret the behavior of the optimization algorithm.

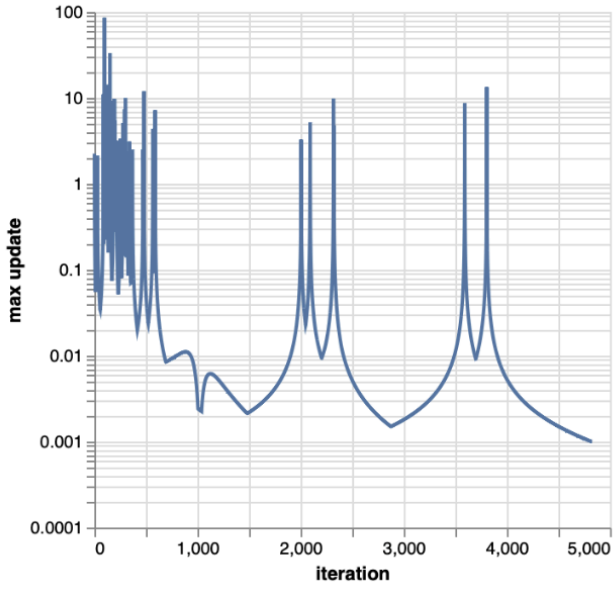
SOLUTION

The first plot visualizes the Root Mean Square Error (RMSE) over iterations. RMSE is a standard measure to assess the accuracy of a model, representing the square root of the average squared differences between the predicted and actual values. In the context of matrix factorization the X-axis represents the iteration number, indicating the progression of the optimization algorithm over time. The Y-axis (log scale) shows the RMSE value on a logarithmic scale. Using a log

1.3.4 Question 1d: Diagnostic Plots







scale helps in highlighting changes across a wide range of error magnitudes, especially useful when improvements become more subtle in later stages. The plot shows a decreasing trend which would indicate that the model is successfully minimizing the prediction error, suggesting effective learning and adjustment of U and V matrices.

The second plot tracks the maximum change in residuals (the differences between actual and predicted ratings) across all observed ratings between successive iterations. The X-axis denotes iteration number and the Y-axis (log scale) denotes the maximum residual change, again presented on a logarithmic scale to better visualize a range of change magnitudes. This plot focuses on the magnitude of changes in the prediction errors. The decreasing values suggest that the model is stabilizing, with successive iterations making smaller adjustments to U and V, indicating that the model is approaching an optimal state.

The third plot shows the maximum update value across all elements of matrices U and V in each iteration. This metric measures the size of the adjustments made to the model parameters. The X-axis is the iteration number and the Y-axis (log scale) is the maximum update magnitude, displayed on a logarithmic scale to accommodate a broad spectrum of update sizes. This plot provides insight into how significantly the model parameters (U and V) are being adjusted at each step. A decreasing trend suggests that updates are becoming more refined, with smaller changes being made as the model converges towards an optimal solution. It's a sign of the algorithm's convergence when the updates become very small, indicating that further iterations are unlikely to significantly improve the model. Though in our case, we have large, fluctuating update values indicate ongoing learning or potential instability in the optimization process.

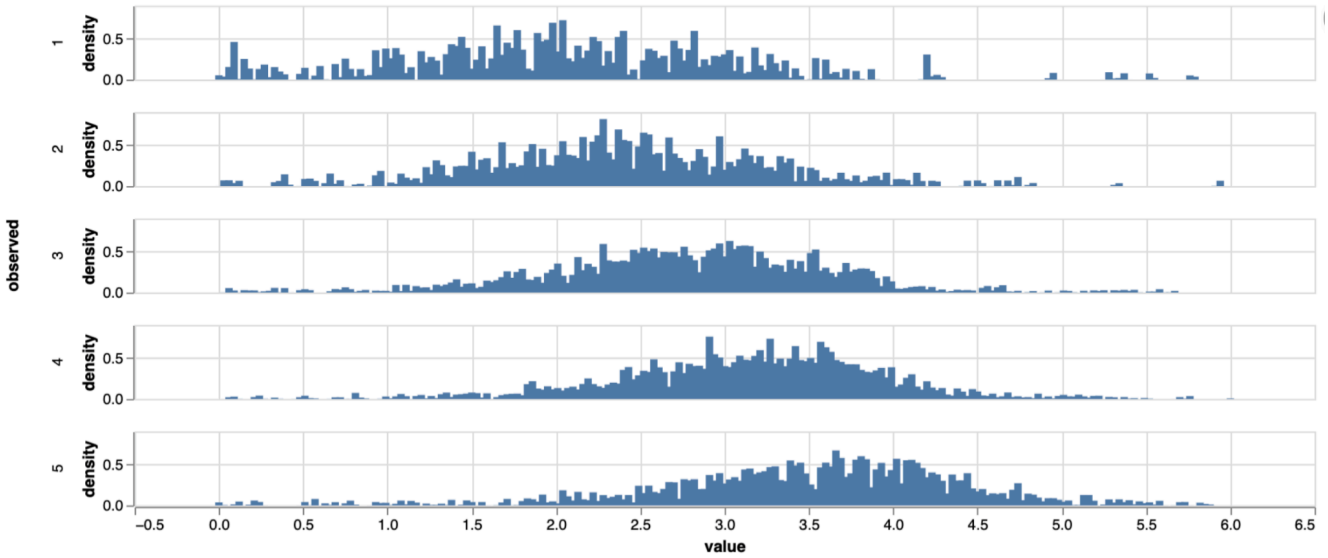
1.3.5 Question 1e: Analyze Large Dataset

Following code will analyze a larger dataset:

[17]:
```
# run on larger dataset
Rbig = pd.read_pickle('data/ratings_stacked.pkl').unstack()[100:]

np.random.seed(134) # set seed for tests
output3 = compute_UV(Rbig, K=5, alpha=0.001, max_iteration=500)

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

[18]:
```
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(
```

1.3.5 Question 1e: Large Dataset Plots



```
    alt.X('value:Q'),
    alt.Y('density:Q'),
    alt.Row('observed:N')
).properties(width=800, height=50)
```

```
/tmp/ipykernel_96/1847306426.py:4: FutureWarning: The previous implementation of
stack is deprecated and will be removed in a future version of pandas. See the
What's New notes for pandas 2.1.0 for details. Specify future_stack=True to
adopt the new implementation and silence this warning.
  ], axis=1).stack().dropna().reset_index()[['fit','observed']]
```

[18]:
```
alt.Chart(…)
```

Question: 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 created by Altair visualizes the density of predicted ratings 'fit' for each observed rating value 'observed'. It does this by grouping the density plots according to the observed rating values. This visualization aims to show how well the predicted ratings align with the actual observed ratings across different rating levels. Ideally, for each group of observed ratings, you'd want the density of predicted ratings to be concentrated around the actual observed rating value, indicating accurate predictions by the model. For each observed rating level, the plot displays a separate bar chart since `mark_bar()` is used, showing the distribution of predicted ratings. The extent [0, 6] limits the x-axis to show predicted ratings within this range. The bandwidth parameter controls the smoothness of the density estimation. A smaller bandwidth results in a more detailed plot that might highlight minor variations in density.

When you add `counts=True` to the `transform_density` method, Altair modifies the density calculation to reflect the raw counts within each bin of the density plot, rather than normalizing these values to form a continuous density estimate. This adjustment makes the plot emphasize the actual number of observations falling within each bin range, offering a more literal representation of the data distribution as opposed to a smoothed density estimate. In conclusion, the original visualization helps assess the model's predictive performance across different rating levels by showing how closely the distributions of predicted ratings align with actual ratings. Adding `counts=True` shifts the focus towards the raw number of predictions within each range, providing an additional layer of understanding regarding the distribution and volume of predicted versus observed ratings.

1.3.6 Question 1f: Make Recommendation

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

[124]:
```
# Extract and rename columns for the user of interest
predicted_ratings = Rhatbig.iloc[:, 600].rename('predicted')
observed_ratings = Rbig.iloc[:, 600].rename('observed')

# Combine the predicted and observed ratings into a single DataFrame
```

```
ratings_combined = pd.DataFrame({'predicted': predicted_ratings, 'observed':␣
 ↪observed_ratings})

# Sort by predicted ratings, descending
ratings_sorted = ratings_combined.sort_values(by='predicted', ascending=False)

# Filter to find unobserved ratings and select the top 5 for recommendation
recommendations = ratings_sorted[ratings_sorted['observed'].isna()].head(10)

# The 'recommendations' DataFrame now holds the top 10 recommendations.
recommendations
```

[124]:

movie id	movie title	predicted	observed
1500	Santa with Muscles (1996)	4.932997	NaN
1467	Saint of Fort Washington, The (1993)	4.016834	NaN
408	Close Shave, A (1995)	3.973421	NaN
216	When Harry Met Sally… (1989)	3.822423	NaN
194	Sting, The (1973)	3.494979	NaN
169	Wrong Trousers, The (1993)	3.476869	NaN
1656	Little City (1998)	3.476402	NaN
1138	Best Men (1997)	3.407717	NaN
180	Apocalypse Now (1979)	3.395319	NaN
1616	Desert Winds (1995)	3.387911	NaN

SOLUTION

To user 601 I would reccomend the movies Santa with Muscles, The Saint of Fort Washington, A Close Shave, When Harry Met Sally and The Sting. There are a few similiarties between the movies we recommended and the movies this user has rated highly, this user tends to like comedy movies, movies from the 1990s and Wallace and Gromit.

1.4 Question 2: Regularization

One of the common problems in machine learning is overfitting, and a common method that remedies overfitting is regularization.

Recall that we solved the following optimization problem

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

where

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

To prevent overfitting, we can introduce L_2 regularization on both the user matrix and the movie matrix. Then the new optimization problem is

$$\min_{U,V} g(U,V) = \min_{U,V} \|R - VU^T\|_F^2 + \lambda(\|U\|_F^2 + \|V\|_F^2)$$
$$= \min_{U,V} \left\{ \sum_{m=1}^{M} \sum_{i=1}^{I} D_{mi}(r_{mi} - v_m u_i^T)^2 + \lambda(\sum_{i=1}^{I} \|u_i\|_2^2 + \sum_{m=1}^{M} \|v_m\|_2^2) \right\}$$

where λ is a tuning parameter that determines the strength of regularization.

1.4.1 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}.

Given the objective function

$$g(U,V) = \sum_{m=1}^{M} \sum_{i=1}^{I} D_{mi}(r_{mi} - (Vu_i)_m^T)^2 + \lambda \left(\sum_{i=1}^{I} \|u_i\|_2^2 + \sum_{m=1}^{M} \|v_m\|_2^2 \right)$$

where $(Vu_i)_m^T$ represents the m-th element of the vector Vu_i, we want to find the gradients with respect to u_i and v_m.

The gradient of $g(U,V)$ with respect to u_i is given by

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

and with respect to v_m by

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

Using these gradients, we can update the matrices U and V using the gradient descent update rules:

$$u_i^{\text{new}} = u_i - \alpha \frac{\partial g(U,V)}{\partial u_i}$$
$$v_m^{\text{new}} = v_m - \alpha \frac{\partial g(U,V)}{\partial v_m}$$

where α is the learning rate.

Our new update functions are:

$$U_i^{\text{new}} = U_i - \alpha \left(-2 \sum_{m=1}^{M} D_{mi}(R_{mi} - (V_m U_i)^T)V_m + 2\lambda U_i \right)$$

$$V_m^{\text{new}} = V_m - \alpha \left(-2 \sum_{i=1}^{I} D_{mi}(R_{mi} - (V_m U_i)^T)U_i + 2\lambda V_m \right)$$

1.4.2 Question 2b: Implementing Updates

Implement new update functions similarly as in q1b.

[21]:
```
import numpy as np
import pandas as pd

def update_G_reg(R_, U_, V_):

    return R_ - np.dot(V_, U_.T)

def update_U_reg(G_, U_, V_, lam, alpha=0.01):

    return U_ + alpha * (2 * (np.nan_to_num(G_).T @ V_) - 2 * lam * U_)

def update_V_reg(G_, U_, V_, lam, alpha=0.01):

    return V_ + alpha * (2 * (np.nan_to_num(G_) @ U_) - 2 * lam * V_)



# small test to help debug (keep intact)
np.random.seed(1)

M_ = 5
I_ = 3
K_ = 2
lam = 5.0

R_ = np.random.rand(M_, I_).round(1)
R_[0, 0] = R_[3, 2] = np.nan
U_ = np.random.rand(I_, K_).round(1)
V_ = np.random.rand(M_, K_).round(1)
G_ = update_G_reg(R_, U_, V_)
```

[22]:
```
grader.check("q2b")
```

[22]:
```
q2b results: All test cases passed!
```

1.4.3 Question 2c: Construct Optimization Algorithm

Combine the above functions to implement the optimization algorithm to iteratively compute U and V.

[23]:
```
def compute_UV_reg(Rdf, K=5, lam=0.5, alpha=0.01, max_iteration=5000,␣
↪diff_thr=1e-3):

    R = Rdf.values
    Rone = pd.DataFrame().reindex_like(Rdf).replace(np.nan, 1) # keep data␣
↪frame metadata
```

```
    M, I = R.shape            # number of movies and users
    U = np.random.rand(I, K)  # initialize with random numbers
    V = np.random.rand(M, K)  # initialize with random numbers
    G = update_G(R, U, V)     # calculate residual

    track_rmse = []
    track_update = []
    for i in range(0, max_iteration):

        Unew = update_U_reg(G,U,V,lam, alpha)
        Gnew = update_G_reg(R, Unew, V)

        Vnew = update_V_reg(Gnew, Unew, V, lam, alpha)
        Gnew = update_G_reg(R, Unew, Vnew)

        track_rmse += [{
            'iteration':i,
            'rmse': rmse(Gnew),
            'max residual change': max_update(Gnew, G, relative=False)
        }]
        track_update += [{
            'iteration':i,
            'max update':max(max_update(Unew, U), max_update(Vnew, V))
        }]

        U = Unew
        V = Vnew
        G = Gnew

        if track_update[-1]['max update'] < diff_thr:
            break

    track_rmse = pd.DataFrame(track_rmse)
    track_update = pd.DataFrame(track_update)

    kindex = pd.Index(range(0, K), name='k')
    U = pd.DataFrame(U, index=Rdf.columns, columns=kindex)
    V = pd.DataFrame(V, index=Rdf.index, columns=kindex)

    return {
        'U':U, 'V':V,
        'rmse': track_rmse,
        'update': track_update
    }

Rsmall = pd.read_pickle('data/ratings_stacked_small.pkl').unstack()
```

```
np.random.seed(134) # set seed for tests
output4 = compute_UV_reg(Rsmall, K=10, lam=0.5, alpha=0.001)
```

[24]:
```
grader.check("q2c")
```

[24]:
```
q2c results: All test cases passed!
```

1.4.4 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.

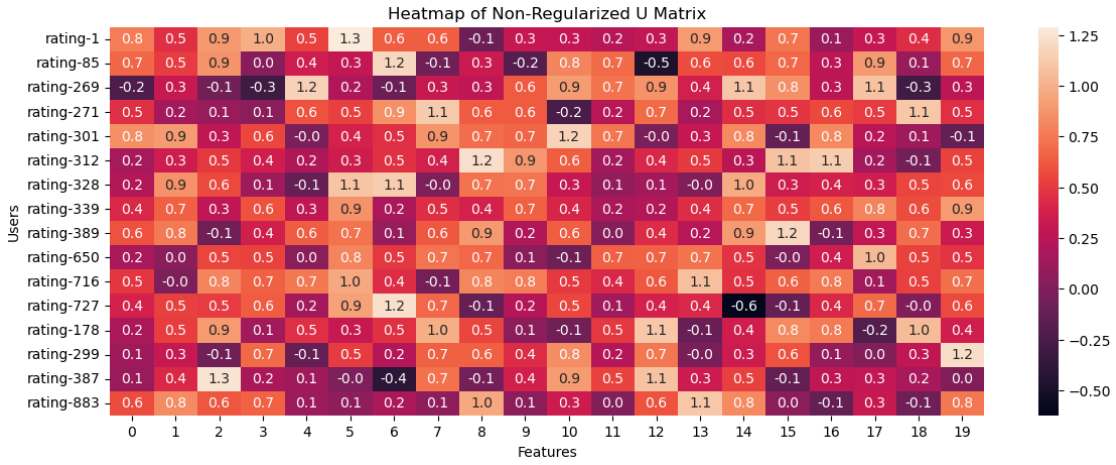
Using the dataset *Rsmall*, obtain two estimated user matrices, say $\hat{U}$ for a non-regularized model and $\hat{U}_{\text{reg}}$ for a regularized model. Select $K = 20$ and $\lambda = 5$. Come up with an effective visualization for comparing $\hat{U}$ and $\hat{U}_{\text{reg}}$, and describe any 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.

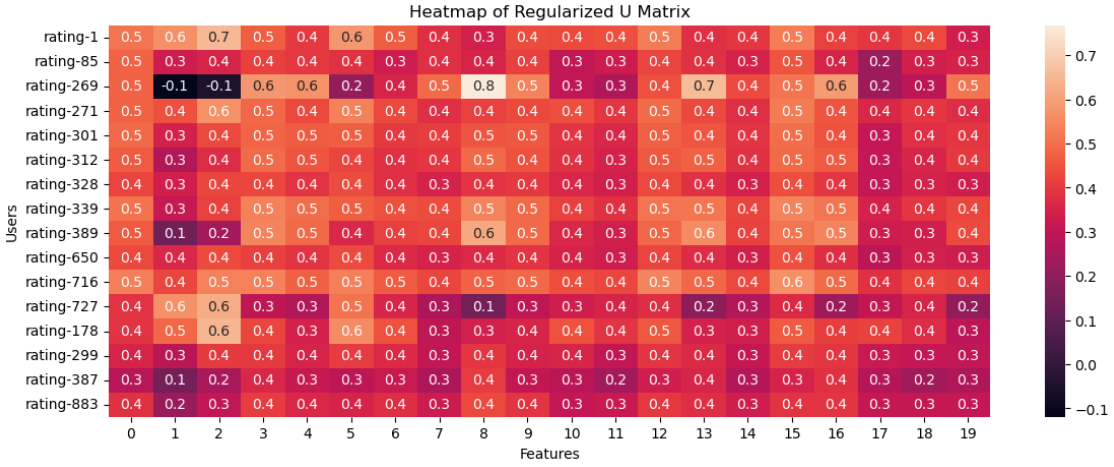
SOLUTION

[25]:
```
np.random.seed(134) # set seed for tests
output_noreg = compute_UV(Rsmall, K=20, alpha=0.001)
output_reg = compute_UV_reg(Rsmall, K=20, lam=5.0, alpha=0.001)
```

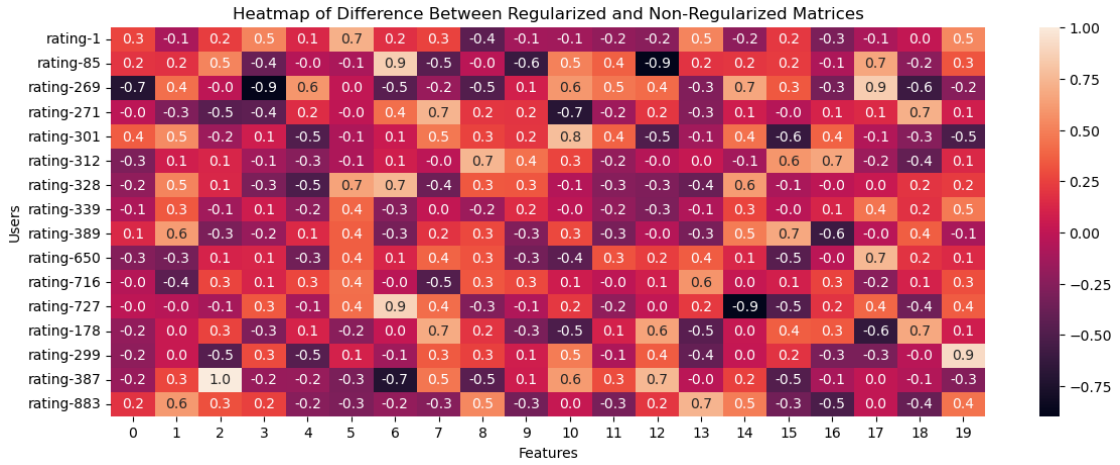
[26]:
```
import matplotlib.pyplot as plt
import seaborn as sns

#Heatmap of Non-Regularized U Matrix
plt.figure(figsize=(14,5))
sns.heatmap(output_noreg['U'], annot=True, fmt=".1f")
plt.title("Heatmap of Non-Regularized U Matrix")
plt.xlabel("Features")
plt.ylabel("Users")
plt.show()
```



[27]:
```
#Heatmap of Regularized U Matrix
plt.figure(figsize=(14,5))
sns.heatmap(output_reg['U'], annot=True, fmt=".1f")
plt.title("Heatmap of Regularized U Matrix")
plt.xlabel("Features")
plt.ylabel("Users")
plt.show()
```



[28]:
```
#Heatmap of Difference Between Regularized and Non-Regularized Matrices
plt.figure(figsize=(14,5))
sns.heatmap(output_noreg['U'] - output_reg['U'], annot=True, fmt=".1f")
plt.title("Heatmap of Difference Between Regularized and Non-Regularized␣
 ↪Matrices")
plt.xlabel("Features")
plt.ylabel("Users")
plt.show()
```



In the non-regularized matrix heatmap, the values span a wider range, with some entries even being negative. In the regularized matrix heatmap, the values appear more uniform and tightly bound around a narrower range, which is closer to zero. This suggests that regularization is indeed shrinking the parameter values towards zero. This behavior is expected because regularization penalizes large weights in the cost function, thereby discouraging the model from assigning too much importance to any single feature or user. The non-regularized heatmap shows greater variance in color intensity, indicating a higher variance in the values of the matrix. Whereas the regularized heatmap has less variance in color intensity, showing that the values are more homogenized. This also aligns with the concept of regularization, which should lead to more regularization as less emphasis is placed on fitting the model to the training data, thereby reducing variance. The difference heatmap emphasizes the impact of regularization. Positive values indicate where the non-regularized matrix had larger values compared to the regularized matrix, and vice versa for negative values. Areas with larger positive differences suggest that those features or interactions were significantly reduced by regularization, which may indicate that the non-regularized matrix was overfitting to these particular aspects of the data. Regularization introduces a bias towards smaller, simpler models, and this is typically evident in the reduced magnitude of the user and item features. In summary, the observed differences do align with the concept of regularization. The regularized matrix's constrained range and reduced variance are indicative of a model that is potentially less likely to overfit, showing a preference for simplicity and possibly better generalization to unseen data.

1.4.5 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

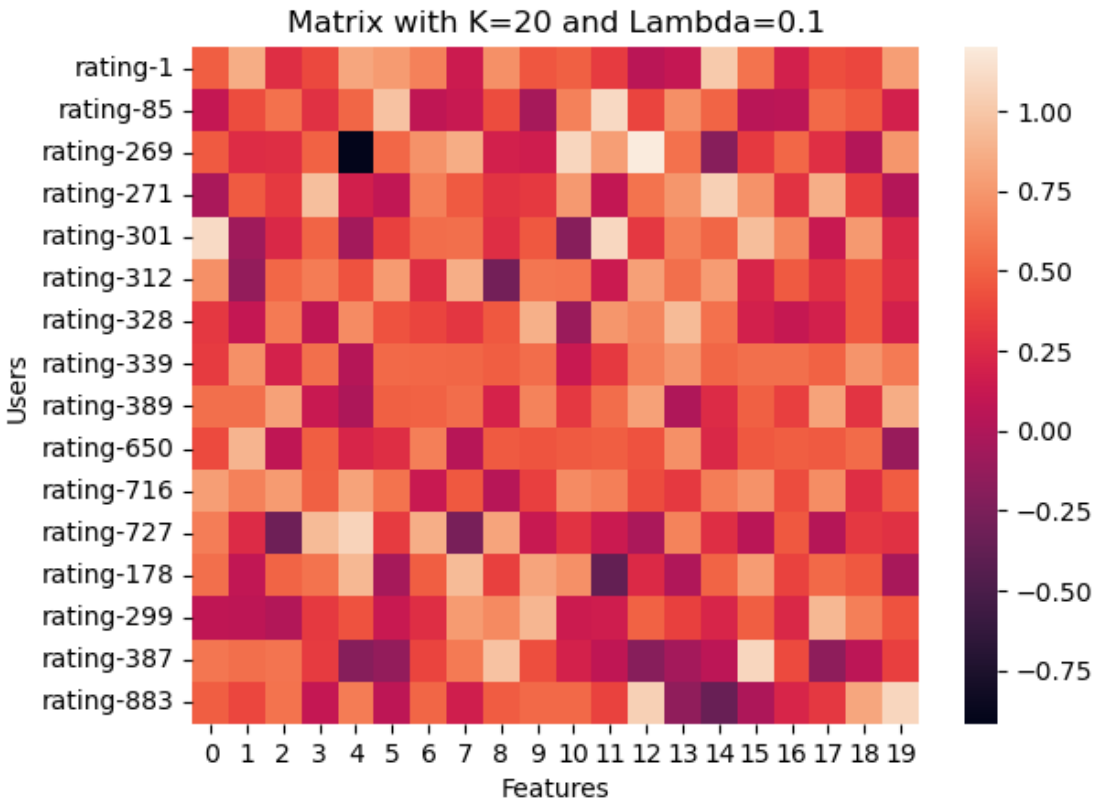
is crucial when applying the recommendation algorithm in practice. As a practitioner, how would you approach choosing K and λ?

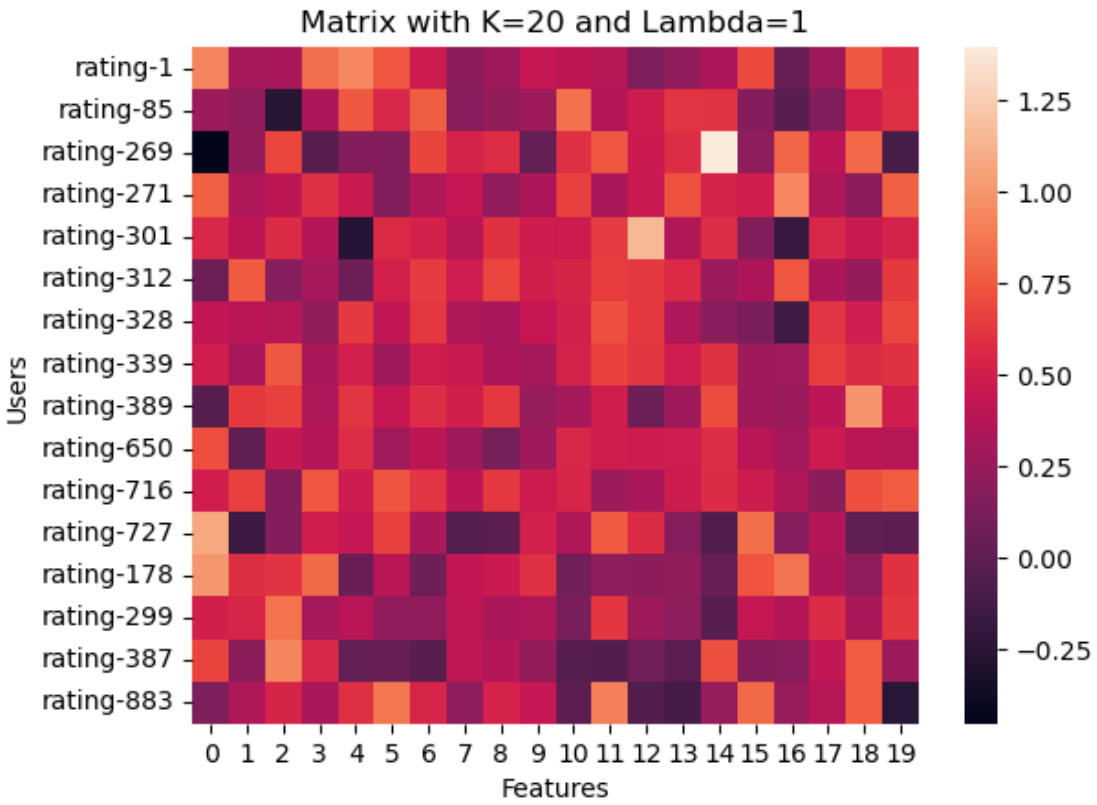
Provide reasoning supported **by evidence**, such as code implementation and results.

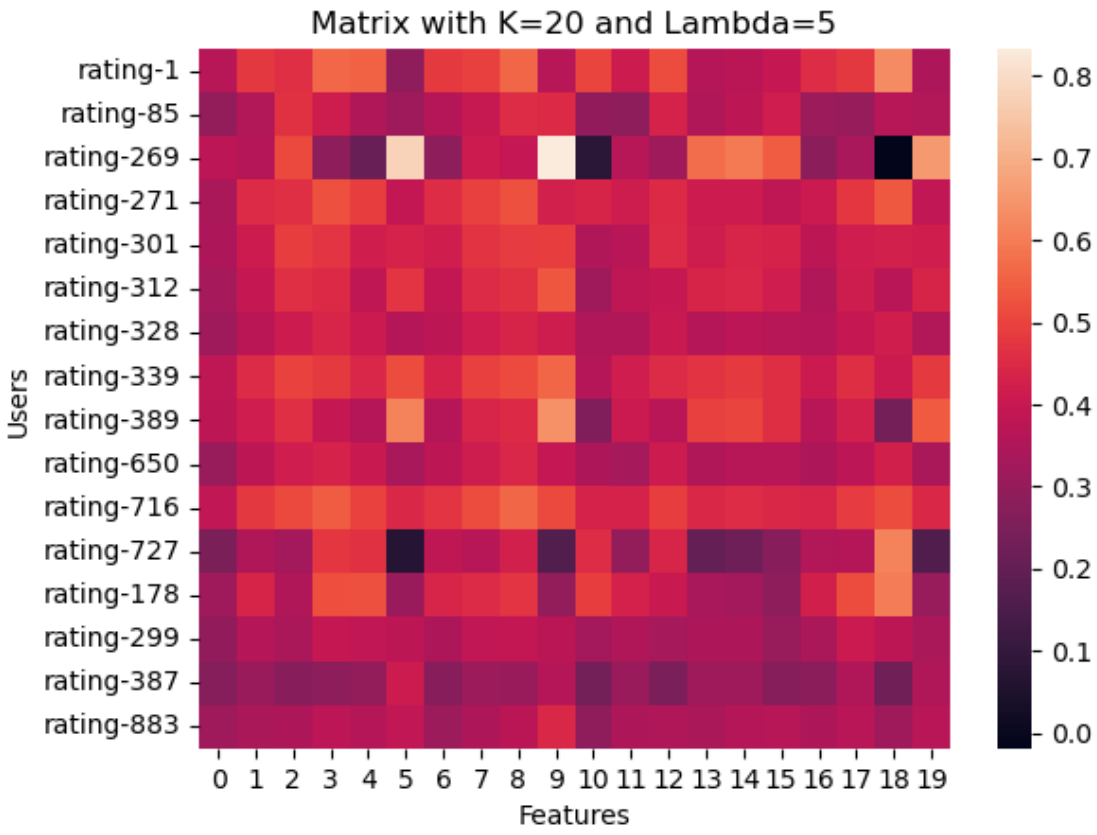
SOLUTION

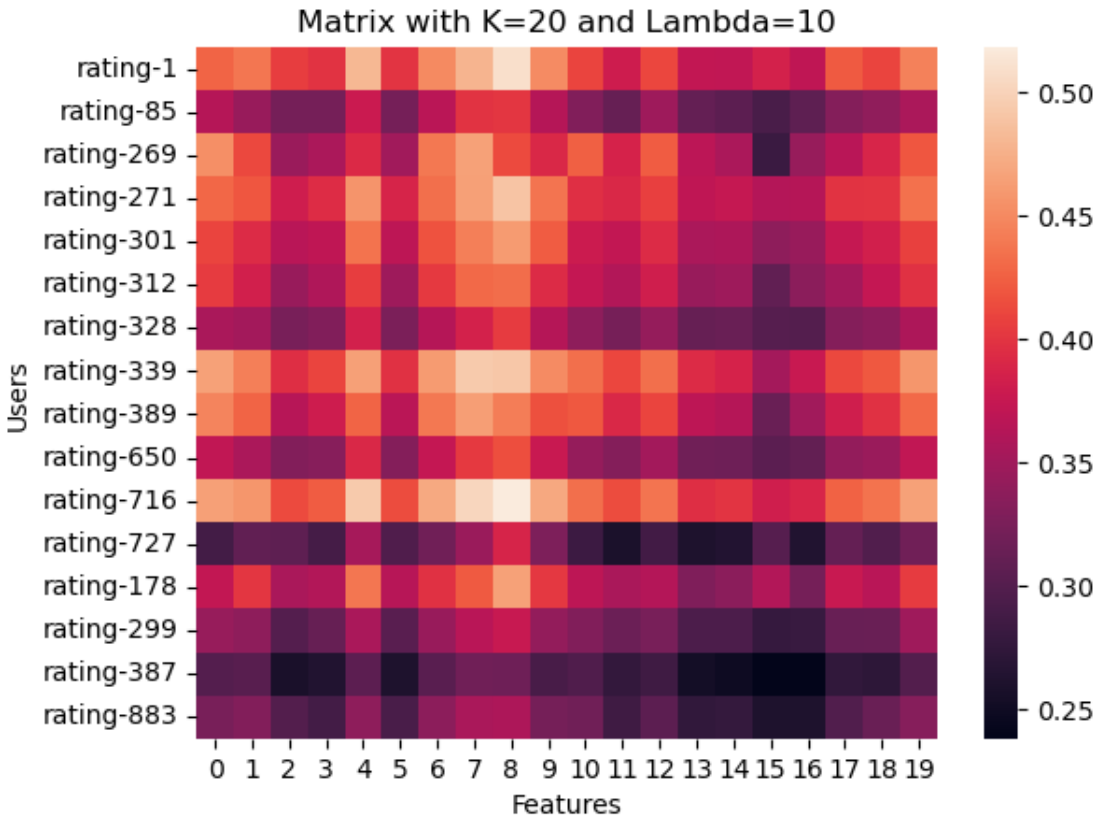
As λ increases, the model's complexity decreases since the regularization term penalizes large weights. The model also becomes less prone to overfitting the training data, as the regularization forces the learned user and item feature vectors to contain smaller values. There's a trade-off however, if λ is too high, the model may become too simple, potentially leading to underfitting and poor predictive performance. Increasing K allows the model to capture more complex user-item interactions because it has more dimensions over which to spread the representation of tastes and preferences. If K is too small, the model may not be able to encapsulate all the nuances of the data, leading to underfitting. Conversely, if K is too large, it might start fitting to the noise in the data, leading to overfitting. In such cases, the model might perform exceptionally well on training data but poorly on validation or test data. As a practitioner we are going to select appropriate K and λ values using RMSE as the evaluation metric. This involves choosing values of K and λ that minimizes the evaluation metric. Additionally we will be using heatmaps for different values of K and λ as a visualization tool to identify trends and the point of diminishing returns or increased overfitting. We also want to keep our model interpretable which means avoiding any very high K values or any very low λ values.

[140]:
```
#Lambda Values to Experiment With
lam_vals = [0.1, 1, 5, 10]

#Loop to Plot Heat Maps with Different Lambda Values
for lam in lam_vals:
    lam_output = compute_UV_reg(Rsmall, K=20, lam=lam, alpha=0.001)
    sns.heatmap(lam_output['U'])
    plt.title(f"Matrix with K=20 and Lambda={lam}")
    plt.xlabel("Features")
    plt.ylabel("Users")
    plt.show()

#Loop to Store RMSE for Different Lambda Values
lambda_rmse = {}
for lam in lam_vals:
    output_lam = compute_UV_reg(Rsmall, K=20, lam=lam, alpha=0.001)
    lambda_rmse[(20,lam)]=output_lam['rmse']['rmse'].iloc[-1]

#Convert to DataFrame and Print
pir_lam=pd.DataFrame(lambda_rmse.items(), columns=['K Lam', "RMSE"])
print(pir_lam)
```


Matrix with K=20 and Lambda=0.1
rating-1
rating-85
rating-269
rating-271
rating-301
rating-312
rating-328
rating-339
rating-389
rating-650
rating-716
rating-727
rating-178
rating-299
rating-387
rating-883
Users
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Features
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75


Matrix with K=20 and Lambda=1
rating-1
rating-85
rating-269
rating-271
rating-301
rating-312
rating-328
rating-339
rating-389
rating-650
rating-716
rating-727
rating-178
rating-299
rating-387
rating-883
Users
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Features
1.25
1.00
0.75
0.50
0.25
0.00
−0.25

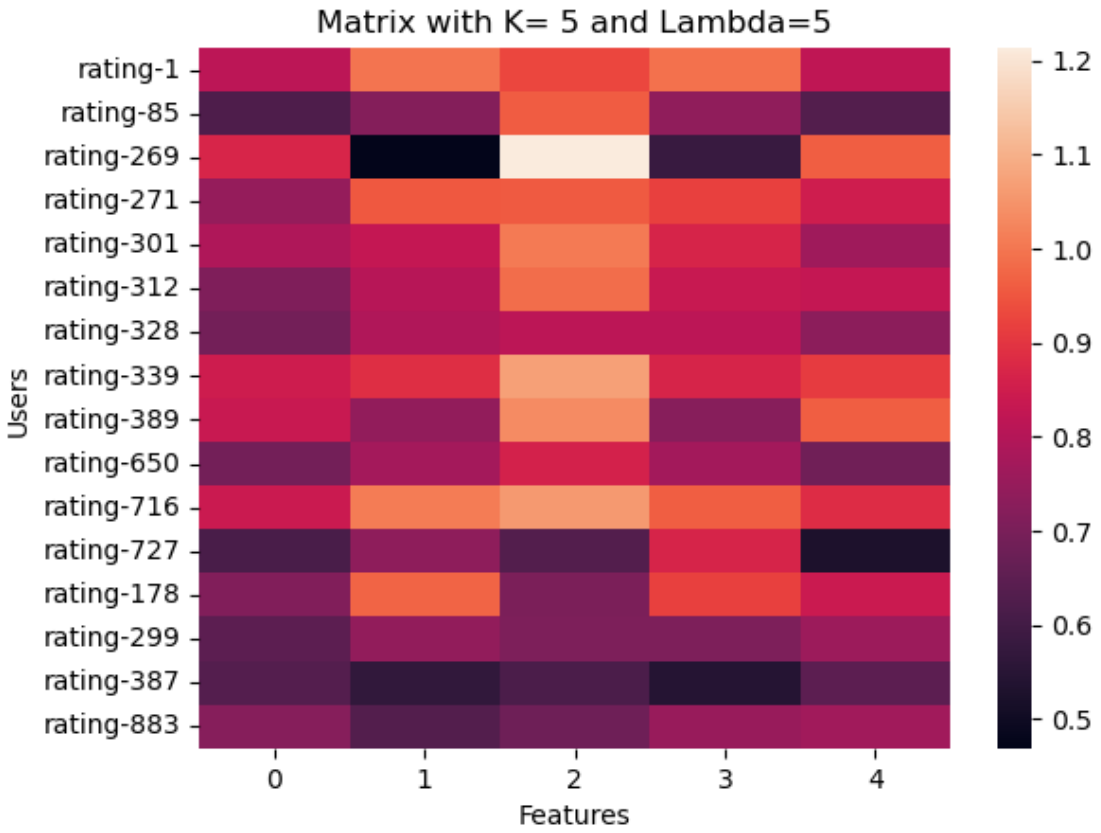

Matrix with K=20 and Lambda=5
rating-1
rating-85
rating-269
rating-271
rating-301
rating-312
rating-328
rating-339
rating-389
rating-650
rating-716
rating-727
rating-178
rating-299
rating-387
rating-883
Users
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Features
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0

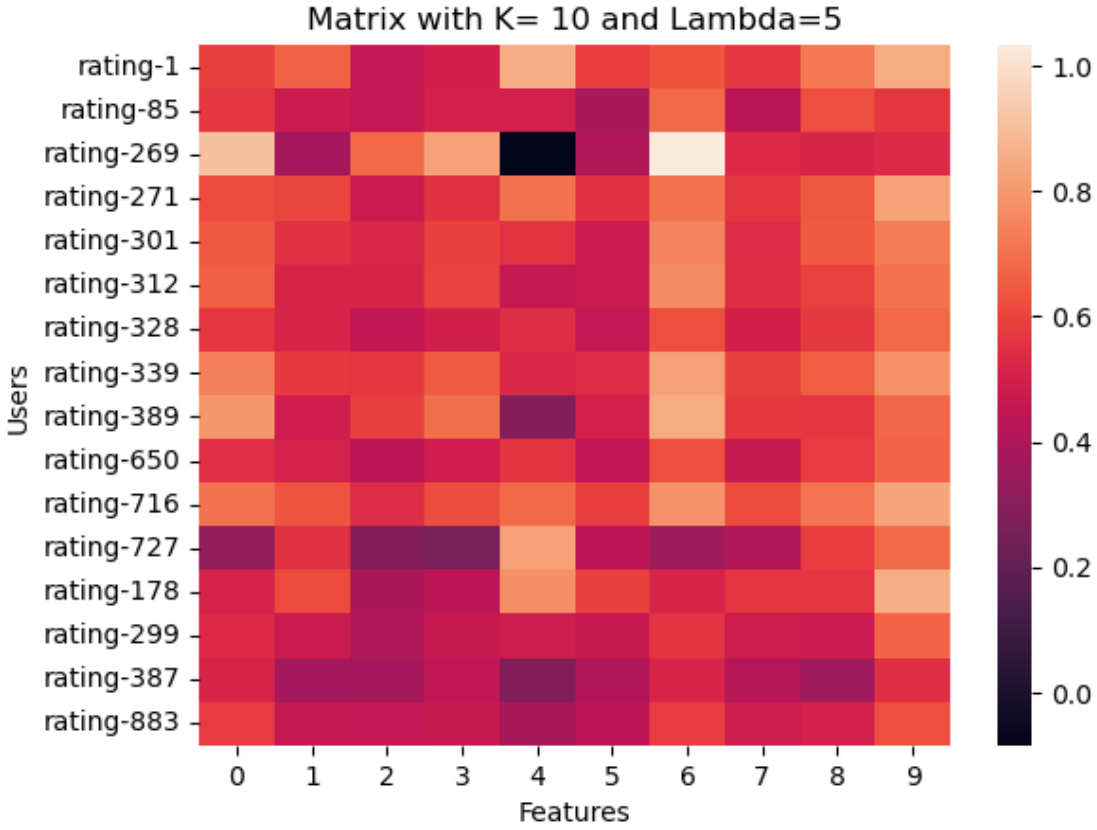


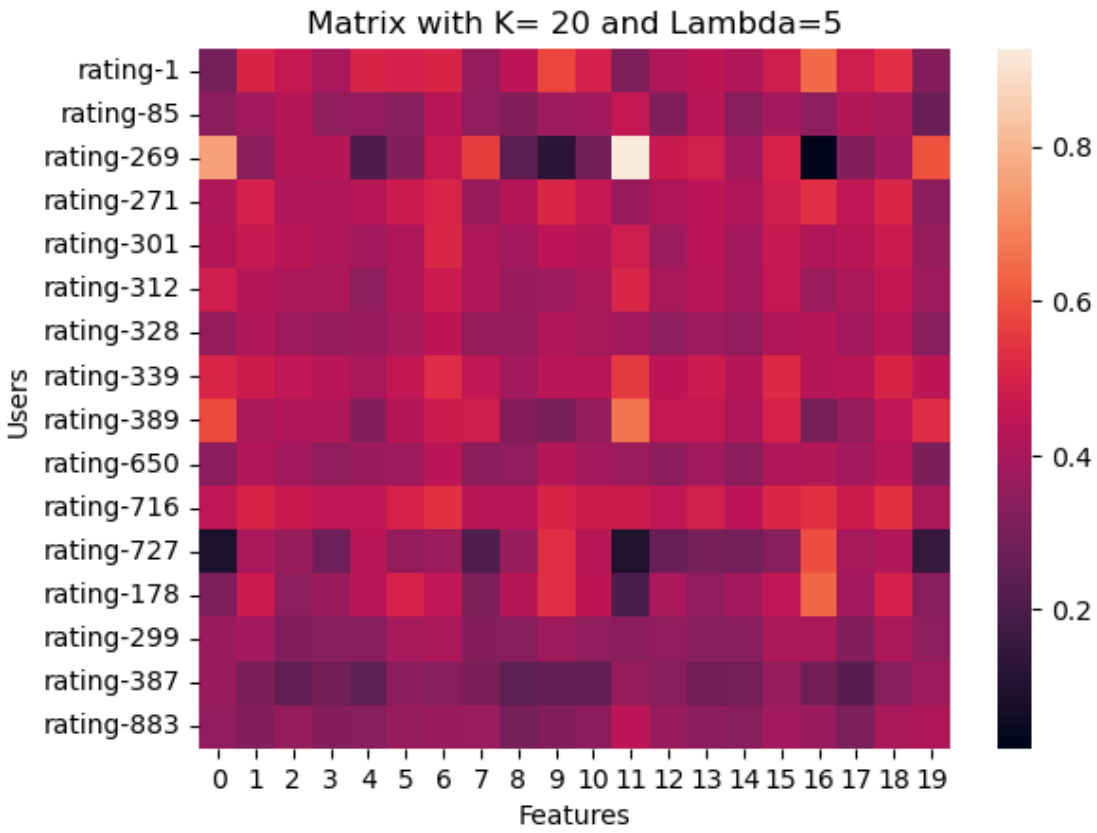
	K Lam	RMSE
0	(20, 0.1)	0.027368
1	(20, 1)	0.256173
2	(20, 5)	0.912350
3	(20, 10)	1.296822

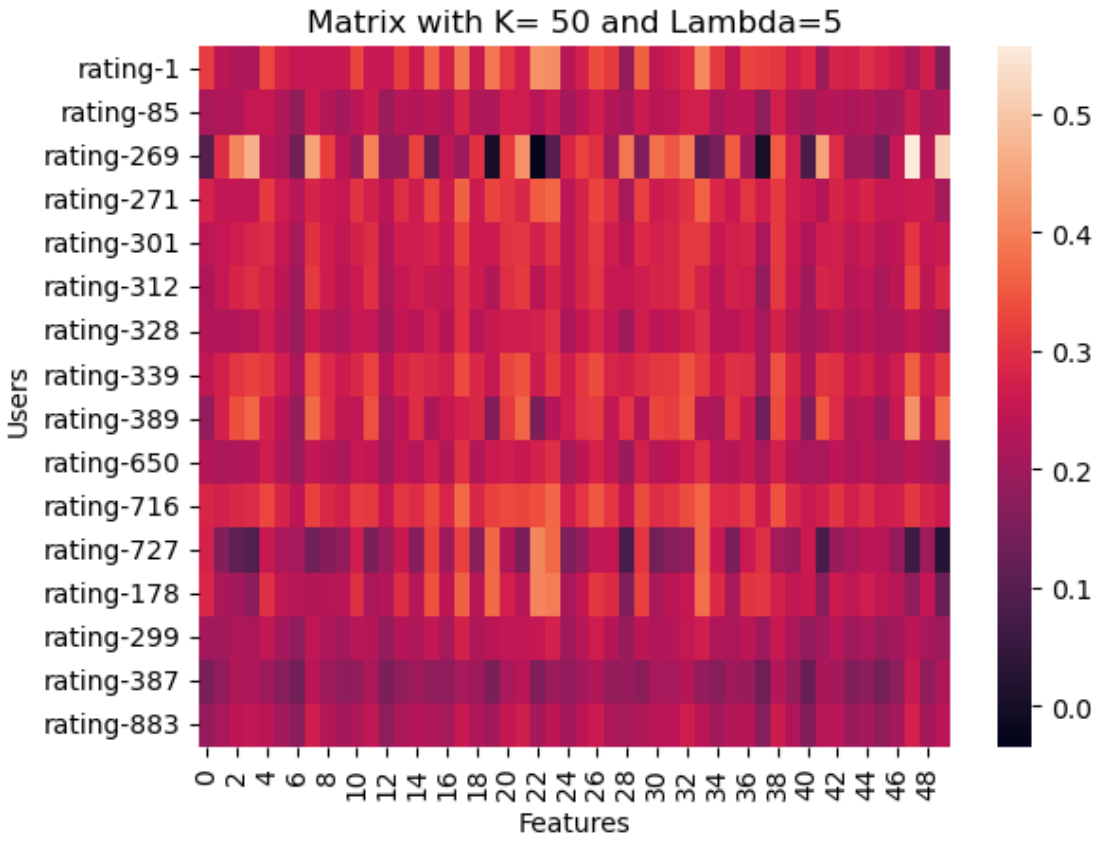
At $\lambda = 0.1$ this heatmap shows a moderate level of regularization. The values have some variability, but not as wide-ranging as you might expect with no regularization. There's a visible pattern, and it seems the model is still allowed some flexibility to fit the data. At $\lambda = 1$ the regularization is stronger, which appears to have a smoothing effect on the matrix values. The color intensity is more uniform than in the previous matrix, suggesting that the user features' magnitudes have been penalized towards smaller values. At $\lambda = 5$ the regularization effect is more pronounced. The color variations are less intense, indicating that the values of the latent features are more constrained. This level of regularization could be dampening the influence of individual preferences, potentially avoiding overfitting. At $\lambda = 10$ the values across the heatmap are the most uniform, indicating a strong shrinkage effect on the user features. The model is likely very conservative, focusing on the most significant patterns that emerge across all users, while downplaying more nuanced user-specific preferences. In conclusion, based off of the visualizations and the calculated RMSEs we can say that the most appropriate λ value is approximately 5.

[141]:
```
#K Values to Experiment With
K_vals = [5, 10, 20, 50]

#Loop to Plot Heat Maps with Different K Values
for K in K_vals:
    K_output = compute_UV_reg(Rsmall, K=K, lam=5, alpha=0.001)
    sns.heatmap(K_output['U'])
    plt.title(f"Matrix with K= {K} and Lambda=5")
    plt.xlabel("Features")
    plt.ylabel("Users")
    plt.show()

#Loop to Store RMSE for Different K Values
K_rmse = {}
for K in K_vals:
    output_K = compute_UV_reg(Rsmall, K=K, lam=5, alpha=0.001)
    K_rmse[(K,5)]=output_K['rmse']['rmse'].iloc[-1]

#Convert to DataFrame and Print
pir_K=pd.DataFrame(K_rmse.items(), columns=['K Lam', "RMSE"])
print(pir_K)
```




Matrix with K= 10 and Lambda=5
rating-1
rating-85
rating-269
rating-271
rating-301
rating-312
rating-328
rating-339
rating-389
rating-650
rating-716
rating-727
rating-178
rating-299
rating-387
rating-883
Users
0
1
2
3
4
5
6
7
8
9
Features
1.0
0.8
0.6
0.4
0.2
0.0


Matrix with K= 20 and Lambda=5
rating-1
rating-85
rating-269
rating-271
rating-301
rating-312
rating-328
rating-339
rating-389
rating-650
rating-716
rating-727
rating-178
rating-299
rating-387
rating-883
Users
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Features
0.8
0.6
0.4
0.2



	K Lam	RMSE
0	(5, 5)	0.919067
1	(10, 5)	0.921512
2	(20, 5)	0.911028
3	(50, 5)	0.907084

With only five latent factors, the model has a very limited capacity to capture the complexity of user preferences or item attributes. The heatmap for $K = 5$ shows relatively large blocks of color, suggesting that users are represented by only a few broad features. Doubling the number of latent factors to ten allows the model to express a more nuanced understanding of the user-item interactions. The heatmap exhibits a little more variation compared to $K = 5$, indicating that users can now be distinguished based on a greater variety of features. With twenty latent factors, the heatmap shows even more granularity. This suggests that the model can discern finer distinctions in user preferences and item characteristics. With more features, the model can capture more complex patterns. The heatmap for $K = 50$ has many fine-grained color changes, which indicates a high level of detail in the user and item representations. At this point, the model has a high capacity for capturing subtleties in the data. However, the constant λ helps to control overfitting, which could be a risk with so many factors. In conclusion, based off of the visualizations and the calculated RMSEs we can say that the most appropriate K value is approximately 20.

1.5 Question 3: Segmentation in Latent Factor Space

Now that we have user matrix U and movie matrix V, suppose we want to use the newly learned representation for an advertising campaign.

Suppose you are leading the planning of an online advertising campaign and you have a fixed budget. With the budget, you can create 5 variations of an ad, and you want to create the variations based on a representative movie each group likes.

The advertisements will entice the viewer to sign up for a mailing list by offering a free poster. The goal of the advertising campaign is two fold:

1. Get potential customers to sign up using their email address by offering a free poster among the 5 "representative" movies
2. Learn their user segment placement preference to use for the starting point for movie recommendations once they sign up

In order to achieve this goal, we want to 1. Produce clusterings of users 2. Balance performance metric of clustering results and practical considerations to choose one of the clustering results.

We will tackle this step by step.

1.5.1 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.

Note: In this part, - Creating a new data `Rmedium` by `ratings_stacked.pkl` and `Rsmall`, 1. Loading `rantings_stacked.pkl` and adding the `user id` having less than 134 NA's in it to `user id` in `Rsmall`, 2. Adding movies with `movie id` $134-234$ to `movie id` in `Rsmall`, 3. Name the new dataset as `Rmedium` and use `Rmedium` to do clustering. - Using the 'best' model you select in Question 2e to compute `UVstd`.

SOLUTION

[85]:
```
np.random.seed(134) # set seed for tests

# Unstack the ratings DataFrame
```

```
ratings_unstacked = pd.read_pickle('data/ratings_stacked.pkl').unstack()

# Find users with fewer than 134 NaN values in the relevant ratings and adjust␣
 ↪user IDs
valid_users_mask = ratings_unstacked.iloc[Rsmall.index.get_level_values("movie␣
 ↪id")].isna().sum(axis=0) < 134
user_ids = np.where(valid_users_mask)[0]  # Adjusts user IDs based on condition

# Combine and adjust user IDs from Rsmall and the calculated valid user IDs
user_id_combined = np.unique(np.concatenate((user_ids, Rsmall.columns.
 ↪get_level_values('user id') - 1)))

# Calculate movie IDs and adjust them
movie_id_combined = np.unique(np.concatenate([Rsmall.index.
 ↪get_level_values("movie id") - 1, np.arange(134, 235) - 1]))

# Create Rmedium by selecting the relevant rows and columns from␣
 ↪ratings_unstacked
Rmedium = ratings_unstacked.iloc[movie_id_combined, user_id_combined]
```

[86]:
```
Rmedium_reg = compute_UV_reg(Rmedium, K=20, lam=5.0, alpha=0.001)
```

[108]:
```
#Standardizing U and V
Umean = np.mean(Rmedium_reg['U'],axis = 0)
Ustd = np.std(Rmedium_reg['U'], axis = 0)
Vmean = np.mean(Rmedium_reg['V'], axis = 0)
Vstd = np.std(Rmedium_reg['V'], axis = 0)
Ustd = (Rmedium_reg['U'] - Umean) / Ustd
Vstd = (Rmedium_reg['V'] - Vmean) / Vstd
UVstd = pd.concat([Ustd, Vstd])
```

[107]:
```
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score,␣
 ↪calinski_harabasz_score

# Apply K-means clustering
kmeans = KMeans(n_clusters=5, random_state=42)
kmeans_labels = kmeans.fit_predict(UVstd)

# Apply Hierarchical clustering
hierarchical = AgglomerativeClustering(n_clusters=5)
hierarchical_labels = hierarchical.fit_predict(UVstd)

# Evaluate clustering performance with different metrics
# Silhouette Score
silhouette_kmeans = silhouette_score(UVstd, kmeans_labels)
```

```
silhouette_hierarchical = silhouette_score(UVstd, hierarchical_labels)

# Davies-Bouldin Index
davies_bouldin_kmeans = davies_bouldin_score(UVstd, kmeans_labels)
davies_bouldin_hierarchical = davies_bouldin_score(UVstd, hierarchical_labels)

# Calinski-Harabasz Index
calinski_harabasz_kmeans = calinski_harabasz_score(UVstd, kmeans_labels)
calinski_harabasz_hierarchical = calinski_harabasz_score(UVstd,␣
 ↪hierarchical_labels)

# Print out the results for comparison
print(f"K-means Clustering Metrics:\nSilhouette Score:␣
 ↪{silhouette_kmeans}\nDavies-Bouldin Index:␣
 ↪{davies_bouldin_kmeans}\nCalinski-Harabasz Index:␣
 ↪{calinski_harabasz_kmeans}\n")
print(f"Hierarchical Clustering Metrics:\nSilhouette Score:␣
 ↪{silhouette_hierarchical}\nDavies-Bouldin Index:␣
 ↪{davies_bouldin_hierarchical}\nCalinski-Harabasz Index:␣
 ↪{calinski_harabasz_hierarchical}")
```

```
K-means Clustering Metrics:
Silhouette Score: 0.10520504897557259
Davies-Bouldin Index: 2.894456185464325
Calinski-Harabasz Index: 92.38963004592705

Hierarchical Clustering Metrics:
Silhouette Score: 0.07767504627923852
Davies-Bouldin Index: 3.523095438786136
Calinski-Harabasz Index: 77.96059699714998
```

The metrics we used are Silhouette Score, Davies-Bouldin Index and the Calinski-Harabasz Index. Silhouette Score measures how similar an object is to its own cluster compared to other clusters, with a range from -1 to 1. A high Silhouette Score indicates that objects are well matched to their own clusters and poorly matched to neighboring clusters, implying that the clusters are distinct and well-separated. In our context, a high Silhouette Score would mean that each user and movie cluster is distinct, which is desirable for targeting specific user preferences with advertising. Davies-Bouldin Index measures the average 'similarity' between clusters, where similarity is the ratio of the sum of within-cluster distances to between-cluster distances. Lower values indicate that clusters are more compact and better separated. For our advertising campaign, a low Davies-Bouldin Index would suggest that the clusters are well-defined, enabling more precise targeting. Calinski-Harabasz Index also known as the Variance Ratio Criterion, this index measures the ratio of the sum of between-clusters dispersion to the sum of within-cluster dispersion. A higher value indicates that the clusters are well-separated and cohesive. In the advertising context, higher values mean we can more confidently target users with ads for movies that are representative of their cluster's preferences. Based on these metrics, K-means appears to be the better choice for clustering in this context, as it consistently outperforms hierarchical clustering across all three evaluated metrics. A higher silhouette score, lower Davies-Bouldin index, and higher Calinski-Harabasz index all

indicate that K-means produces more distinct and cohesive clusters, which is desirable for our goal of segmenting users and movies effectively for targeted advertising.

Since we do not have true labels in our scenario, we cannot use supervised metrics like purity or adjusted Rand index. Our choice is limited to unsupervised metrics. The choice between these metrics depends on the specific qualities of clustering that are most relevant to our goals. If we value the distinctness of clusters, the Silhouette Score provides direct insight into how well-separated and cohesive the clusters are. If our primary concern is ensuring that clusters do not overlap and are well-defined, the Davies-Bouldin Index is useful since it penalizes clusters that are not well-separated. When we want a balance of cluster separation and cohesion, and have a larger dataset, the Calinski-Harabasz Index offers a measure of both qualities. In the context of planning an online advertising campaign, we aim to create distinct segments of users based on their movie preferences. Therefore, we might prioritize a metric that best reflects the separation and cohesion within clusters, as this will allow us to more accurately target users with ads that resonate with their preferences. The Silhouette Score is particularly relevant here, as it directly measures both how cohesive each cluster is internally and how well-separated it is from other clusters, directly informing our ability to target advertising effectively.

1.5.2 Question 3b: Visualizing Clusters in Latent Space

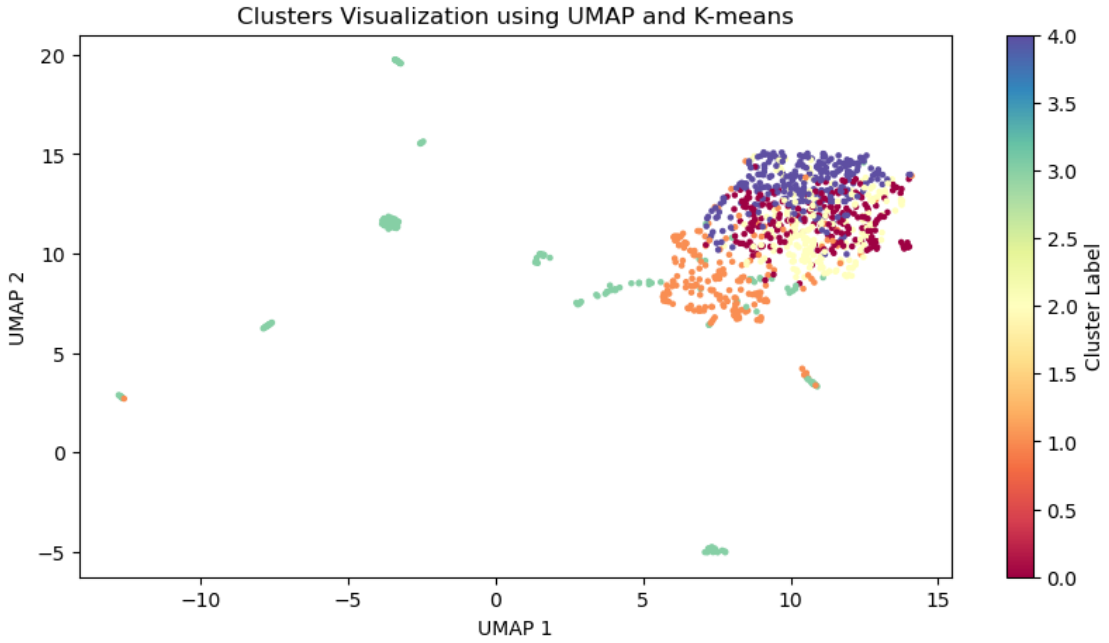
Select the clustering method based on the evaluation results in q3a and visualize the clusters using UMAP. Are the clusters and UMAP projection consistent?

SOLUTION

[109]:
```
# install umap
# !pip install umap-learn
```

[123]:
```
from umap import UMAP

# Perform UMAP dimensionality reduction
umap = UMAP(n_neighbors=5, min_dist=0.3, random_state=42)
UVstd_umap = umap.fit_transform(UVstd)

# Plot the clusters formed by K-means
plt.figure(figsize=(10, 5))
plt.scatter(UVstd_umap[:, 0], UVstd_umap[:, 1], c=kmeans_labels,␣
 ↪cmap='Spectral', s=5)
plt.title('Clusters Visualization using UMAP and K-means')
plt.xlabel('UMAP 1')
plt.ylabel('UMAP 2')
plt.colorbar(label='Cluster Label')
plt.show()
```

```
/opt/conda/lib/python3.11/site-packages/umap/umap_.py:1943: UserWarning: n_jobs
value -1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```



It seems that the clusters are fairly distinct, with some degree of overlap between neighboring clusters, which is normal in a reduced dimensional space, especially if the original data has complex structures. The dense regions of color indicate areas where members of the same cluster are close to each other, demonstrating good clustering cohesion. There are also clear boundaries between most of the clusters, which is an indicator of good cluster separation. The presence of some points that appear distant from their primary cluster group might suggest a few outliers within the data, or they could represent transitional data points that share similarities with multiple clusters. Overall, the UMAP projection appears consistent with well-defined clusters, suggesting that the K-means algorithm has done a reasonably good job of identifying distinct user and movie segments.

1.6 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

Download the zip file and submit to Gradescope.

[143]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export(run_tests=True)
```

```
Running your submission against local test cases…


Your submission received the following results when run against available test
cases:
```

```
    q1b results: All test cases passed!

    q1c results: All test cases passed!

    q2b results: All test cases passed!

    q2c results: All test cases passed!
<IPython.core.display.HTML object>
```