Ullrich Köthe: Fundamentals of Machine Learning, Winter Semester 2021/22
Notebook created by Christian Weilbach, Lorenzo Cerrone and Jens Müller

Solutions for Exercise 8

Setup

In [1]:
```
import math

from numpy.random import RandomState
import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import load_digits
from sklearn import decomposition

digits = load_digits()

X = digits["data"]/16.
Y = digits["target"]
```

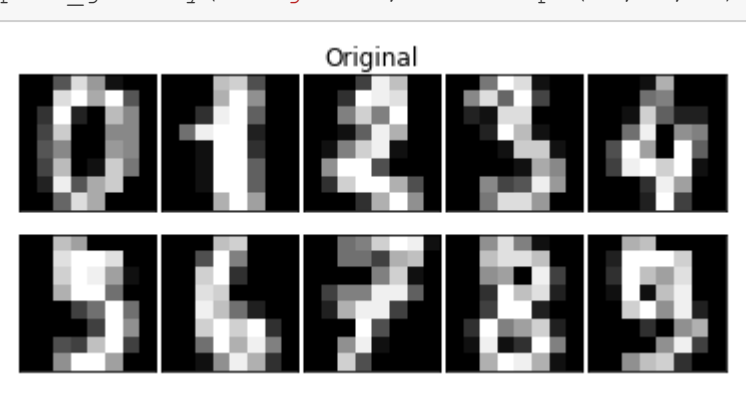
1 Non-negative matrix factorization

1.1 Comparison of scikit-learn's NMF with SVD (6 Points)

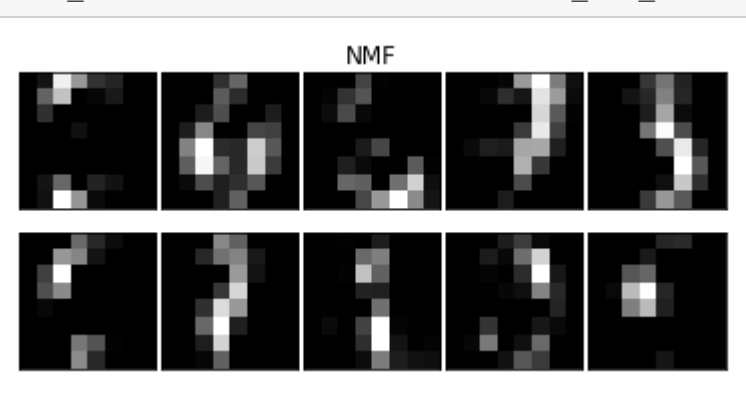
In [2]:
```
# Setup parameters
n_row, n_col = 2, 5
image_shape = (8, 8)
n_components = n_row * n_col

# Sklearn NMF decomposition
nmf = decomposition.NMF(n_components=n_components, init='nndsvda', tol=5e-3)
nmf.fit(X)
```

Out[2]: `NMF(init='nndsvda', n_components=10, tol=0.005)`

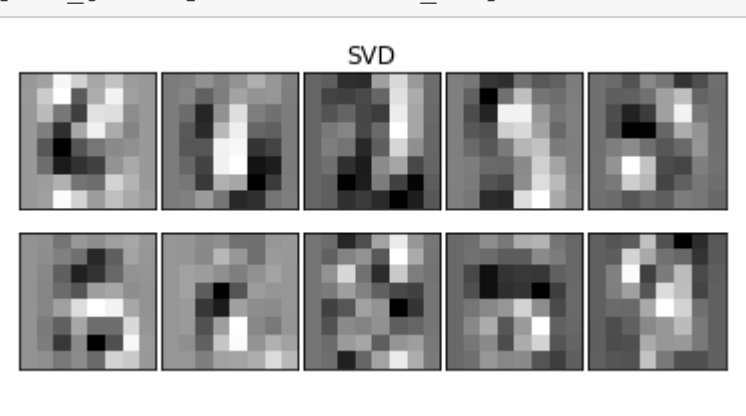
In [3]:
```
def plot_gallery(title, images, n_col=n_col, n_row=n_row):
    # generate a plot with the images tiled
    plt.figure(figsize=(1. * n_col, 1.26 * n_row))
    plt.suptitle(title)
    for i, comp in enumerate(images):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(comp.reshape(image_shape), cmap=plt.cm.gray,
                   interpolation='nearest',
                   vmin=comp.min(), vmax=comp.max())
        plt.xticks(())
        plt.yticks(())
    plt.subplots_adjust(0.01, 0.05, 0.99, 0.93, 0.04, 0.)

plot_gallery('Original', X.reshape(-1, 8, 8)[:n_components])
```



In [4]:
```
plot_gallery('NMF', nmf.components_[:n_components])
```



In [5]:
```
X_mu = np.mean(X, axis=0)
X_c = X - X_mu
U, S, Vt = np.linalg.svd(X_c)

plot_gallery('SVD', Vt[:n_components])
```



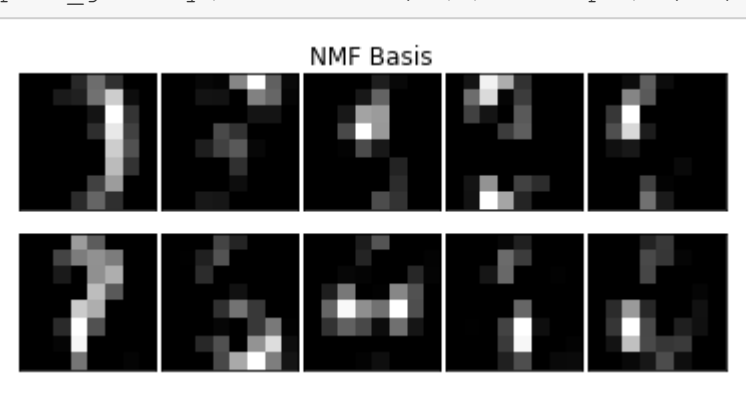
The resulting bases for NMF are much sparser than those for SVD - NMF almost clusters different features, while svd only finds suitable linear combinations. Note that while black corresponds to zero for the NMF basis, instead gray corresponds to zero for the SVD basis (because the entries of an orthogonal matrix can be in the range [-1, 1]). Hence there is also some sparsity in the SVD basis, but only in the unused pixels at the sides.

1.2 Implementation - Non-negative matrix factorization (8 Points)

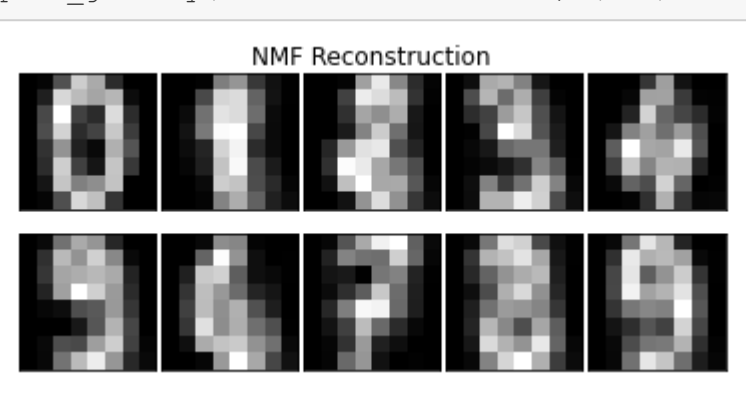
In [6]:
```
def frobenius(A, B):
    # returns the frobenius norm between two matrices
    return np.sum((A-B)**2)


def non_negative(Z, H, X, n_components=10, num_iterations=1000, verbose=False):
    # Non-Negative factorization
    X_ = Z @ H

    for i in range(num_iterations):
        H_ = np.clip(H * (Z.T@X)/(Z.T@(Z@H)), 1e-128, 1000)
        Z_ = np.clip(Z * (X @ H_.T)/((Z@H_)@H_.T), 1e-128, 1000)
        X_ = Z_ @ H_

        loss = frobenius(X, X_)

        if math.isnan(loss):
            break

        if i % 100 == 0 and verbose:
            print("Frobenius-Norm at iteration %i: %f" % (i, loss))

        H = H_
        Z = Z_

    return Z, H

# W and H initialization
Z_rand = np.abs(np.random.randn(X.shape[0], n_components))
H_rand = np.abs(np.random.randn(n_components, X.shape[1]))

# Non-negative factorization
Z, H = non_negative(Z_rand, H_rand, X, n_components=10, verbose=True)
```

```
Frobenius-Norm at iteration 0: 9611.726940
Frobenius-Norm at iteration 100: 3462.099678
Frobenius-Norm at iteration 200: 3388.734861
Frobenius-Norm at iteration 300: 3365.956635
Frobenius-Norm at iteration 400: 3355.676675
Frobenius-Norm at iteration 500: 3347.981715
Frobenius-Norm at iteration 600: 3343.125065
Frobenius-Norm at iteration 700: 3339.629670
Frobenius-Norm at iteration 800: 3336.314613
Frobenius-Norm at iteration 900: 3334.166980
```

In [7]:
```
# basis
plot_gallery('NMF Basis', (H).reshape(-1, 8, 8)[0:10], 5, 2)
```



In [8]:
```
# reconstructions
plot_gallery('NMF Reconstruction', (Z@H).reshape(-1, 8, 8)[0:10], 5, 2)
```



The calculated Frobenius above shows the expected non-increasing loss. The sklearn implementation and our own show results which are structurally similar, i.e. they exhibit the same sparseness, although the exact shape and appearance of the different sparseness vectors differs. This will mainly be due to different random initialisations, as the results of the single algorithms also vary depending on their intialisation.

2 Recommender system (12 Points)

In [9]:
```
# inspired by https://beckernick.github.io/matrix-factorization-recommender/

## Recommendation

# https://web.archive.org/web/20171125033229/http://files.grouplens.org/datasets/movielens/ml-100k.zip
import pandas as pd


# column headers for the dataset
ratings_cols = ['user id','movie id','rating','timestamp']
movies_cols = ['movie id','movie title','release date',
'video release date','IMDb URL','unknown','Action',
'Adventure','Animation','Childrens','Comedy','Crime',
'Documentary','Drama','Fantasy','Film-Noir','Horror',
'Musical','Mystery','Romance ','Sci-Fi','Thriller',
'War' ,'Western']
users_cols = ['user id','age','gender','occupation',
'zip code']

users = pd.read_csv('ml-100k/u.user', sep='|',
names=users_cols, encoding='latin-1')

movies = pd.read_csv('ml-100k/u.item', sep='|',
names=movies_cols, encoding='latin-1')

ratings = pd.read_csv('ml-100k/u.data', sep='\t',
names=ratings_cols, encoding='latin-1')
```

In [10]:
```
# look at the dataframes
users.head()
```

Out[10]:

	user id	age	gender	occupation	zip code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

In [11]:
```
movies.head()
```

Out[11]:

	movie id	movie title	release date	video release date	IMDb URL	unknown	Action	Adventure	Animation	Childrens	...	Fantasy	Film-Noir	H
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	0	0	1	1	...	0	0	
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	1	1	0	0	...	0	0	
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	0	0	0	0	0	...	0	0	
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	0	1	0	0	0	...	0	0	
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0	0	0	0	0	...	0	0	

5 rows × 24 columns

In [12]:
```
ratings.head()
```

Out[12]:

	user id	movie id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

In [13]:
```
# create a joint ratings dataframe for the matrix
fill_value = 0
rat_df = ratings.pivot(index = 'user id',
                       columns ='movie id', values = 'rating').fillna(fill_value)
rat_df.head()
```

Out[13]:

movie id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676	1677	1678	1679	1680	1681	1682
user id																					
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	4.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 1682 columns

In [14]:
```
# numpy matrix to work on
rat_m = rat_df.values

Z_rand = np.abs(np.random.randn(rat_m.shape[0], n_components))
H_rand = np.abs(np.random.randn(n_components, rat_m.shape[1]))

Z, H = non_negative(Z_rand, H_rand, rat_m, n_components=10)

preds_df = pd.DataFrame(Z @ H, columns=rat_df.columns)
preds_df.head()
```

Out[14]:

movie id	1	2	3	4	5	6	7	8	9	10	...	1673	1674
0	4.204638	1.794327	1.188842	3.113615	1.189468	6.163674e-01	5.169269	2.673227	3.061191	1.890857	...	6.596764e-03	8.389533e-03
1	1.732447	0.000029	0.270319	0.157981	0.089274	4.122797e-01	1.710818	0.644681	2.049287	0.992618	...	1.356879e-22	9.089893e-69
2	0.024088	0.016772	0.073697	0.090991	0.046320	1.365289e-47	0.127793	0.027394	0.066833	0.009341	...	6.001719e-04	6.730939e-15
3	0.472751	0.027088	0.039726	0.072748	0.014769	2.477552e-47	0.335533	0.125401	0.176102	0.016863	...	1.089116e-03	6.436518e-04
4	3.910092	0.947269	0.494880	1.122838	0.395402	1.859321e-03	2.672205	1.777367	0.151653	0.281364	...	9.796038e-61	9.172262e-03

5 rows × 1682 columns

In [15]:
```
def recommend_movies(predictions, user_id, movies, original_ratings, num_recom=5):
    # Get and sort the user's predictions
    user_row_number = user_id - 1 # UserID starts at 1, not 0
    sorted_user_predictions = (predictions.iloc[user_row_number].
                               sort_values(ascending=False))
    # Get the user's data and merge in the movie information.
    user_data = original_ratings[original_ratings["user id"] == (user_id)]
    user_full = (user_data.merge(movies, how = 'left',
                                 left_on = 'movie id', right_on = 'movie id').
                 sort_values(['rating'], ascending=False))

    # leave out movies already seen by user
    recommendations = (movies[~movies['movie id'].isin(user_full['movie id'])].
         merge(pd.DataFrame(sorted_user_predictions).reset_index(), how = 'left',
               left_on = 'movie id',
               right_on = 'movie id').
         rename(columns = {user_row_number: 'predictions'}).
         sort_values('predictions', ascending = False).iloc[:num_recom, :-1])

    return user_full, recommendations

# Prediction recommendation for user 1
already_rated, predictions = recommend_movies(preds_df, 1, movies, ratings, 10)
```

In [16]:
```
predictions
```

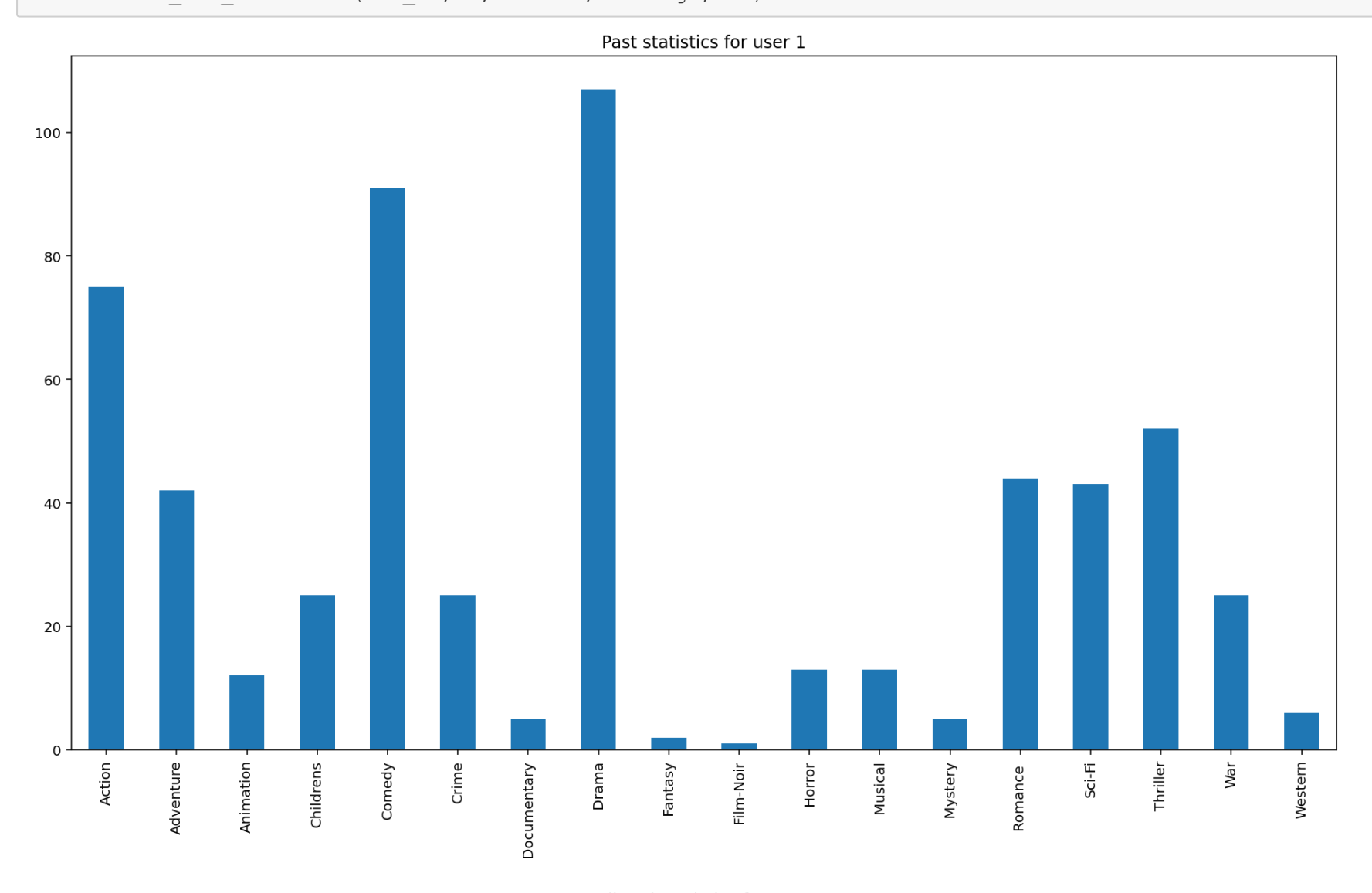
Out[16]:

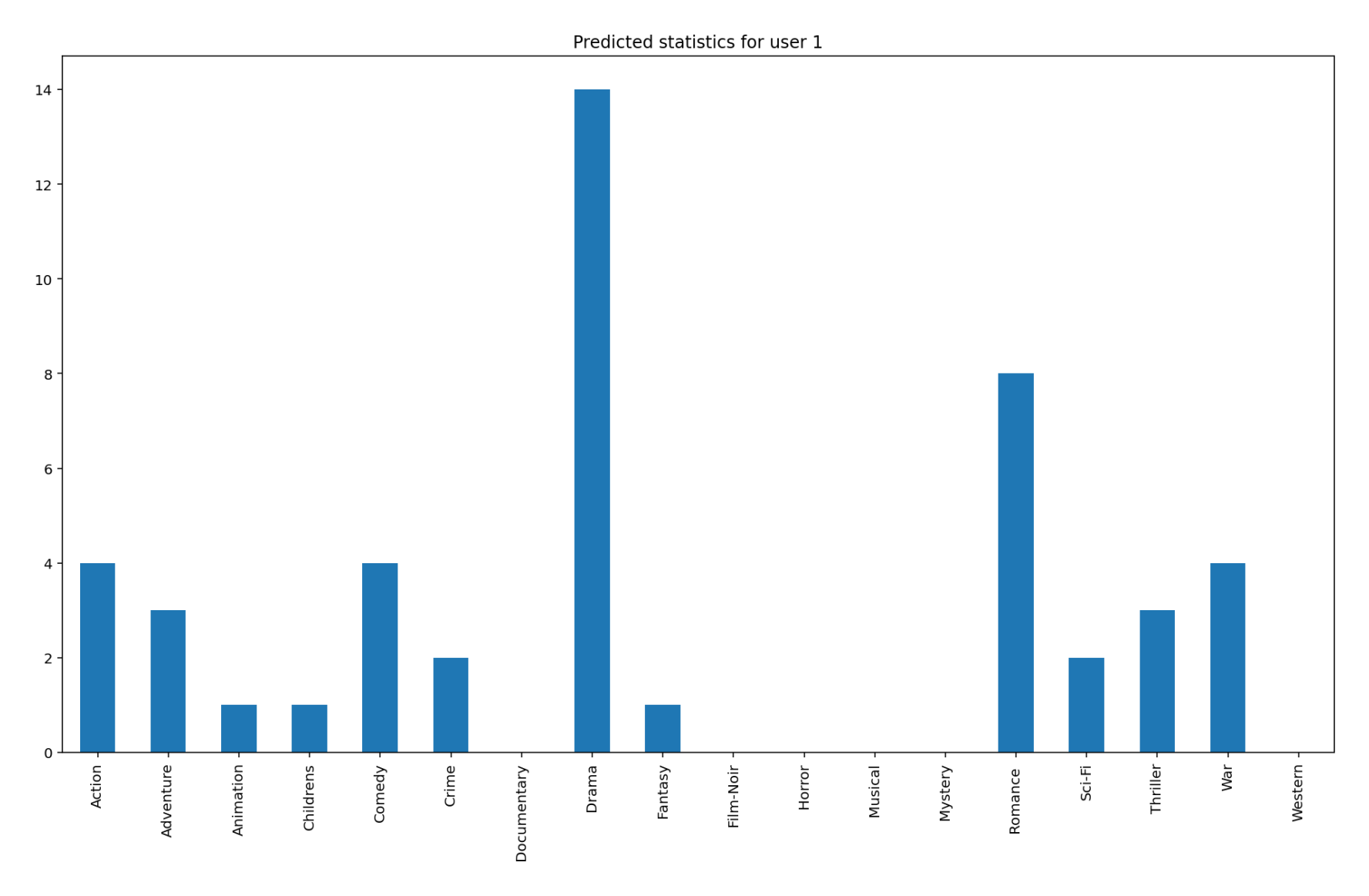
	movie id	movie title	release date	video release date	IMDb URL	unknown	Action	Adventure	Animation	Childrens	...	Fantasy	Film-Noir
202	475	Trainspotting (1996)	19-Jul-1996	NaN	http://us.imdb.com/Title?Trainspotting+(1996)	0	0	0	0	0	...	0	0
2	275	Sense and Sensibility (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Sense%20and%2...	0	0	0	0	0	...	0	0
12	285	Secrets & Lies (1996)	04-Oct-1996	NaN	http://us.imdb.com/M/title-exact?Secrets%20&%2...	0	0	0	0	0	...	0	0
135	408	Close Shave, A (1995)	28-Apr-1996	NaN	http://us.imdb.com/M/title-exact?Close%20Shave...	0	0	0	1	0	...	0	0
13	286	English Patient, The (1996)	15-Nov-1996	NaN	http://us.imdb.com/M/title-exact?English%20Pat...	0	0	0	0	0	...	0	0
160	433	Heathers (1989)	01-Jan-1989	NaN	http://us.imdb.com/M/title-exact?Heathers%20(1...	0	0	0	0	0	...	0	0
3	276	Leaving Las Vegas (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Leaving%20Las...	0	0	0	0	0	...	0	0
382	655	Stand by Me (1986)	01-Jan-1986	NaN	http://us.imdb.com/M/title-exact?Stand%20by%20...	0	0	1	0	0	...	0	0
150	423	E.T. the Extra-Terrestrial (1982)	01-Jan-1982	NaN	http://us.imdb.com/M/title-exact?E%2ET%2E%20th...	0	0	0	0	1	...	1	0
130	403	Batman (1989)	01-Jan-1989	NaN	http://us.imdb.com/M/title-exact?Batman%20(1989)	0	1	1	0	0	...	0	0

10 rows × 24 columns

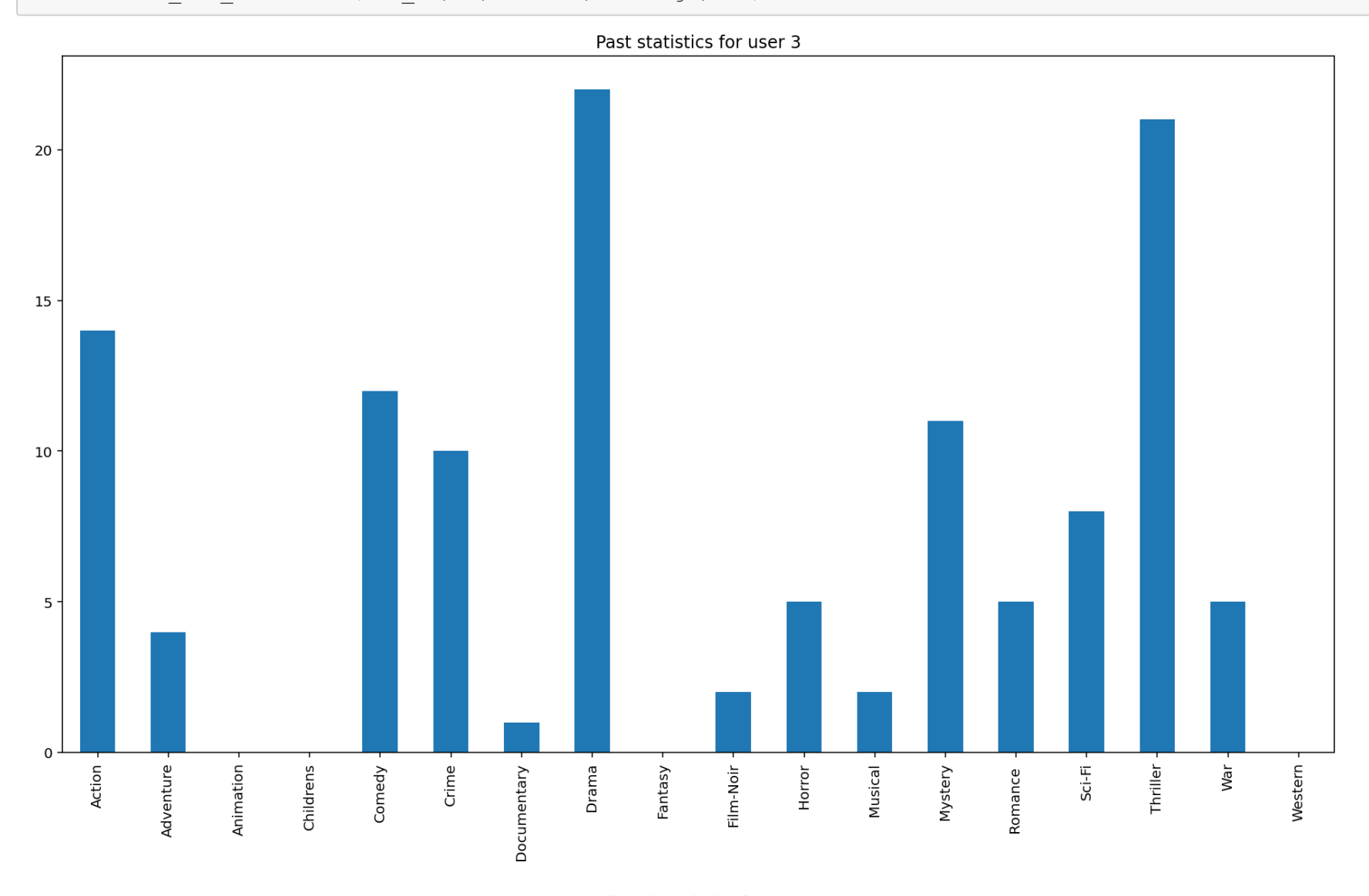
In [17]:
```
unwanted_keys_pred = ["movie id", "movie title", "release date", "video release date",
                      "IMDb URL", "unknown"]
unwanted_keys_gt = ["user id", "timestamp", "rating", "movie id", "movie title",
                    "release date", "video release date", "IMDb URL", "unknown"]

def reconstruct_and_recommend(rat_df, user_id, movies, ratings, num_recom=5):
    # numpy matrix to work on
    rat_m = rat_df.values

    Z_rand = np.abs(np.random.randn(rat_m.shape[0], n_components))
    H_rand = np.abs(np.random.randn(n_components, rat_m.shape[1]))

    Z, H = non_negative(Z_rand, H_rand, rat_m,
                        n_components=num_recom,
                        num_iterations=1000)
    preds_df = pd.DataFrame(Z @ H, columns=rat_df.columns)

    # Prediction recommendation for user
    already_rated, predictions = recommend_movies(preds_df, user_id, movies, ratings, num_recom)

    # Prune data frame
    for key in unwanted_keys_pred:
        del predictions[key]

    for key in unwanted_keys_gt:
        del already_rated[key]

    # Plot predictions
    already_rated.sum().plot(kind='bar', figsize=(16, 9), title=f"Past statistics for user {user_id}")
    plt.show()

    predictions.sum().plot(kind='bar', figsize=(16, 9), title=f"Predicted statistics for user {user_id}
")
    plt.show()
```

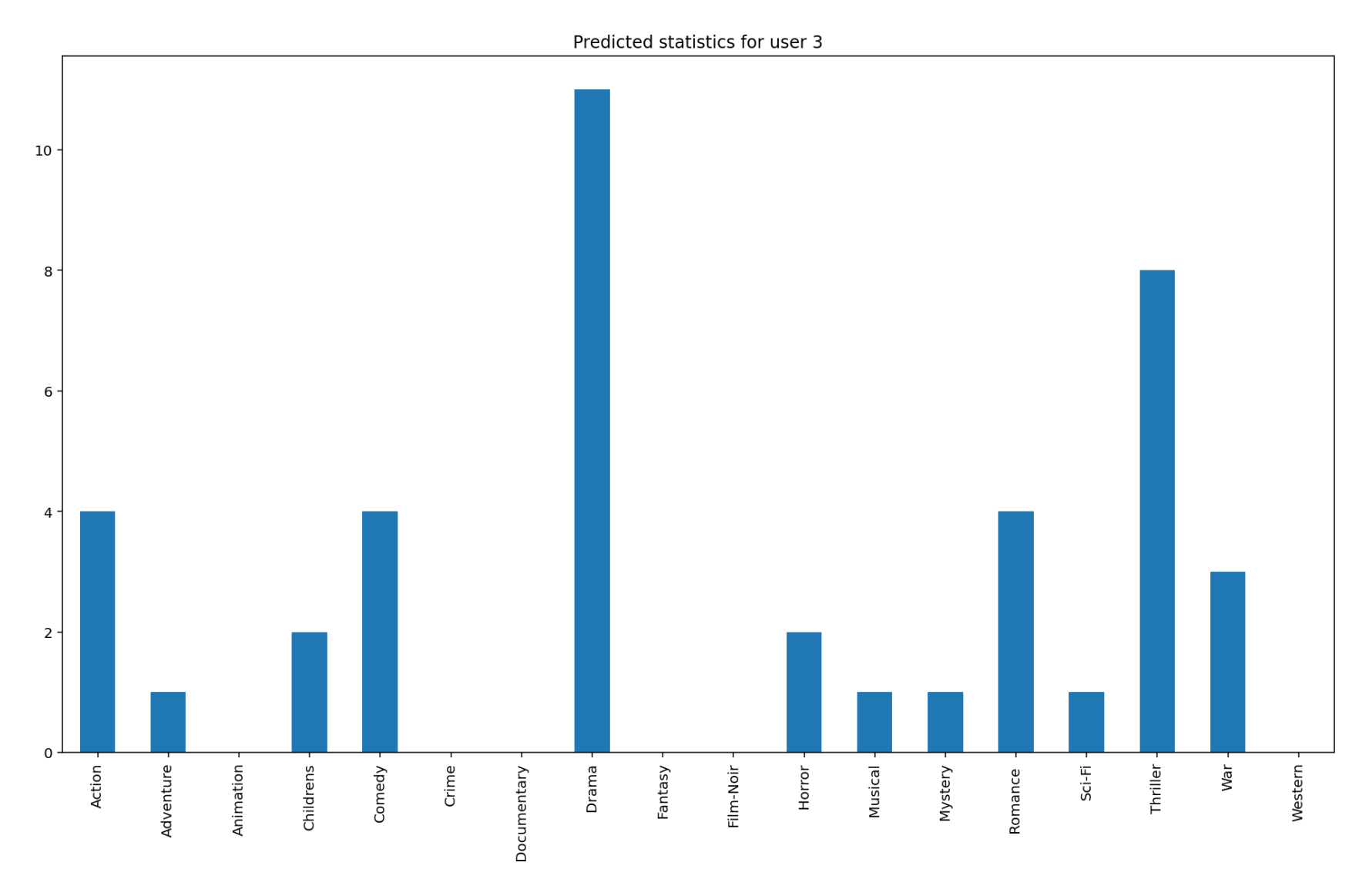
In [18]:
```
reconstruct_and_recommend(rat_df, 1, movies, ratings, 20)
```





In [19]:
```
reconstruct_and_recommend(rat_df, 3, movies, ratings, 20)
```





The comparison of genre statistics for reconstructed data and original data shows a similar behaviour to the basis observed for the digits dataset: The general structure is conserved, while the entries for different genres are sparser.

To find out whether a certain row of H correspond to a certain genre, we multiply the selected row of H with a vector that is 1 for all movies in this genre and 0 elsewhere. If this value is big, we consider the vectors to be similar.

Out of these considerations we create a heatmap. In the heatmap we have entries $a_{j,i} = < H_{j\cdot}, m_i >$ where $H_{j\cdot}$ corresponds to the jth row in H and m_i to the vector which is 1 for movies in genre i and 0 elsewhere. Note that in the code below we normalize the vectors before computing the scalar product.

In [20]:
```
mv = movies.copy()
columns = mv.columns
for a in ['movie id', 'movie title', 'release date', 'video release date','IMDb URL', 'unknown']:
    del mv[a]

# The following list will evolve into a matrix which represents the genres of each movie
# Rows represent different genres and columns different movies
movies_list = []
for c in mv.columns:
    matrix = mv[c].to_numpy().reshape(-1,1)
    # We make sure every 'genre-vector' adds up to 1
    # Therefore we take into account when certain genres are overrepresented
    matrix = matrix/np.linalg.norm(matrix, 2)
    movies_list.append(matrix)
out = np.array(movies_list)
out = np.swapaxes(out, 0, 1).reshape(-1, 18)

n_comp = 10
# Non negative matrix factorization with n_comp components
Z_rand = np.abs(np.random.randn(rat_m.shape[0], n_comp))
H_rand = np.abs(np.random.randn(n_comp, rat_m.shape[1]))
Z, H = non_negative(Z_rand, H_rand, rat_m,
                    n_components=n_comp,
                    num_iterations=1000)

# We also normalize the rows of H
for i in range(H.shape[0]):
    H[i,:] = H[i,:]/np.linalg.norm(H[i,:], 2)

fig, ax = plt.subplots(figsize=(10, 10))

# Similar to the intuition stated above, but in matrix form
# A matrix that represents similarities between rows of H and film genres
out =np.dot(H, out)

# We enforce the values in one row to lie in [0,1]
for i in range(n_comp):
    out[i,:] /= out[i,:].max()

# What follows is a heatmap showing the similiarity between rows in H and different genres
im = ax.imshow(out)

ax.set_xticks(np.arange(18))
ax.set_yticks(np.arange(n_comp))
ax.set_xticklabels(mv.columns)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
for i in range(n_comp):
    for j in range(18):
        text = ax.text(j, i, round(out[i, j],2),
        ha="center", va="center", color="w", size='smaller')

ax.set_title("Rows of H and Film Genres")
```

Out[20]:
```
Text(0.5, 1.0, 'Rows of H and Film Genres')
```

For instance row 0 (counting starts from 0) of H corresponds to the user preference of comedy movies. Other rows can be categorized simiarly.