Bonus-Exercise

Deadline: 12.08.2024, 16:00

Regulations

Please hand in your solution as a Jupyter notebook `nmf.ipynb` along with exported HTML.

Zip all files into a single archive `ex08.zip` and upload this file to MaMPF before the given deadline. Moreover, please set your **Anzeigename/display name** and **Name in Uebungsgruppen/name in tutorials** in MaMPF to your real name, which should be identical to your name in `muesli` and make sure you **join the submission** of your team via the invitation code before the submission deadline. Check out `https://mampf.blog/handing-in-homework-assignments` for instructions.

1 Non-negative matrix factorization (20 points)

1.1 Implementation

We learned in the lecture that the NMF can be found by alternating updates of the form

$$\mathbf{H}_{t+1} \leftarrow \mathbf{H}_t \frac{\mathbf{Z}_t^T \mathbf{X}}{\mathbf{Z}_t^T \mathbf{Z}_t \mathbf{H}_t} \tag{1}$$

$$\mathbf{Z}_{t+1} \leftarrow \mathbf{Z}_t \frac{\mathbf{X} \mathbf{H}_{t+1}^T}{\mathbf{Z}_t \mathbf{H}_{t+1} \mathbf{H}_{t+1}^T} \tag{2}$$

Numerators and denominators of the fractions are matrix multiplications, whereas the divisions and multiplicative updates must be executed element-wise. Implement a function `non_negative(data, num_components)` that calculates a non-negative matrix factorization with these updates, where `num_components` is the desired number of features M after decomposition. Initialize $\mathbf{Z_0}$ and $\mathbf{H_0}$ positively, e.g by taking the absolute value of standard normal random variables (RV) with `np.random.randn`. Iterate until reasonable convergence, e.g. for $t = 1000$ steps. Note that you might have to ensure numerical stability by avoiding division by zero. You can achieve this by clipping denominators at a small positive value with `np.clip`. Run your code on the digits data, plot the resulting basis vectors and compare with the `NMF` results from `scikit-learn` (results should be similar). Can you confirm that the squared loss $\|\mathbf{X} - \mathbf{Z_t} \cdot \mathbf{H_t}\|_2^2$ is non-increasing as a function of t?

1.2 Recommender system

Use your code to implement a recommendation system. We will use the `movielens-100k` dataset with pandas, which you can download at `https://tinyurl.com/HD-EML-ex08-material-zip` or as an "external link" on MaMPF.

```
import pandas as pd   # install pandas via conda


#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')

# peek at the dataframes, if you like :)
users.head()
movies.head()
ratings.head()

# 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()
```

The data matrix $\mathbf{X}$ is called `rat_df` in the code. It is sparse because each user only rated a few movies. The variable `fill_value = 0` determines the default value of missing ratings. You can play with this value (e.g. set it to the average rating of all movies, or to the average of each specific movie instead of a constant).

Now compute the non-negative matrix factorization. Play with the number of components m in your factorisation. You should choose m such that the reconstruction $\widehat{\mathbf{X}} = \mathbf{Z} \cdot \mathbf{H}$ is less sparse than the actual rating matrix. This allows the recommender system to suggest a movie to a user when that movie has not been rated in $\mathbf{X}$ by him/her, but is predicted in $\widehat{\mathbf{X}}$ to receive a high rating. Write a method to give movie recommendations for movies, which user `user_id` has not yet seen (or at least rated):

```
reconstruction = pd.DataFrame(Z @ H, columns = rat_df.columns)
predictions = recommend_movies(reconstruction, user_id, movies, ratings)
```

You can also add some ratings for additional users (yourself) and check if the resulting recommendations make sense. Show (e.g. with histograms) that the "genre-statistics" vary between already rated movies and predicted movies (e.g. with 20 predictions) for selected users. What is the difference and how do you explain it? Also try to identify rows in $\mathbf{H}$ that can be interpreted as prototypical user preferences (e.g. "comedy fan").

Sidenote: At least until around 2012, Netflix was using a similar `SVD++` reconstruction together with a restricted Boltzmann machine (RBM) to give recommendations. [1]

[1] `https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429`