Exercise 9

Deadline: 24.1.2018

Regulations

Please hand in the following:

- Jupyter notebook as well as an exported PDF `nmf-comparison.*` that cover Question 1.
- PDF (hand written or typed) `method-nmf.pdf` that answers Question 2.
- Jupyter notebook as well as an exported PDF `nmf-implementation.*` that contain all parts of Question 3 and 4.

Zip all files into a single archive with naming convention (sorted alphabetically by last names)

`lastname1-firstname1_lastname2-firstname2_exercise09.zip`

or (if you work in a team of three)

`lastname1-firstname1_lastname2-firstname2_lastname3-firstname3_exercise09.zip`

and upload it to Moodle before the given deadline.

1 Setup

First load the dataset and import scikit-learn's decomposition module:

```
import math
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"]/255.
Y = digits["target"]
```

2 Non-negative matrix factorization vs. SVD comparison (6 Points)

Use the decomposition module to compare non-negative matrix factorization (NMF) with singular value decomposition (SVD, `np.linalg.svd`) on the digits dataset, where the methods factorize $\mathbf{X}$ in the following way:

$$\mathbf{X} = \mathbf{W} \cdot \mathbf{H} \text{ (NMF)} \tag{1}$$

$$\mathbf{X} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \text{ (SVD)} \tag{2}$$

$\mathbf{X}, \mathbf{W}, \mathbf{H} \in \mathbb{R}_+$. If $\mathbf{X} \in \mathbb{R}_+^{k \times n}$ and your number of latent components is m then $\mathbf{W} \in \mathbb{R}_+^{k \times m}$ and $\mathbf{H} \in \mathbb{R}_+^{m \times n}$. Use SVD with full rank and at least 10 components for NMF. You should use centered data for SVD (not for NMF of course) and add the means back to the basis vectors. Plot and compare the first 6 components (rows) of $\mathbf{H}$ with the ones of $\mathbf{V}$. You can view these matrices as a basis of the vector space of the digit dataset. What do you observe?

3 Method - Non-negative matrix factorization (12 Points)

If we assume that the observation matrix $\mathbf{X}$ differs from the decomposition $\mathbf{WH}$ by additive Gaussian noise, the objective of NMF for $\mathbf{H}$ can be written in terms of the the Frobenius norm of the difference:

$$\min_{\mathbf{H}} ||\mathbf{X} - \mathbf{WH}||_F^2 \tag{3}$$

First of all we observe symmetry in $\mathbf{W}$ and $\mathbf{H}$ through transposition, so we can derive the gradient for one and yield similarly the gradient for the other. We also simplify by just minimizing over a column j, $\mathbf{x}_j$, in $\mathbf{X}$ and column $\mathbf{h}_j$ in $\mathbf{H}$:

$$\min_{\mathbf{h}_j} ||\mathbf{x}_j - \mathbf{W}\mathbf{h}_j||^2 \tag{4}$$

We skip the j index from now on and just look at one column. **Derive the gradient** for $\nabla\mathbf{h}$ under the euclidean norm and **verify** that a gradient descent step $t \to t+1$

$$\mathbf{h}_{t+1} \leftarrow \mathbf{h}_t - \eta\nabla\mathbf{h}_t \tag{5}$$

with learning rate $\eta = \frac{\mathbf{h}}{\mathbf{W}^T\mathbf{W}\mathbf{h}}$ yields:

$$\mathbf{h}_{t+1} = \mathbf{h}_t \frac{\mathbf{W}_t^T\mathbf{X}}{\mathbf{W}_t^T\mathbf{W}_t\mathbf{h}_t} \tag{6}$$

Under this learning rate the objective is guaranteed to be non-increasing and converges to a local minimum[1]. Note that division is here understood element-wise. Show also that updates according to equation (6) preserve the positivity constraints on $\mathbf{H}$ and $\mathbf{W}$ when these constraints were fulfilled at time step t, in other words it ensures feasibility.

4 Implementation - Non-negative matrix factorization (10 Points)

Given the update rules from the previous exercise, implement an iterative procedure `non_negative(data, num_components)` that calculates a non-negative matrix factorization with the updates:

$$\mathbf{H}_{t+1} \leftarrow \mathbf{H}_t \frac{\mathbf{W}_t^T\mathbf{X}}{\mathbf{W}_t^T\mathbf{W}_t\mathbf{H}_n} \tag{7}$$

$$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t \frac{\mathbf{X}\mathbf{H}_{t+1}^T}{\mathbf{W}_t\mathbf{H}_{t+1}\mathbf{H}_{t+1}^T} \tag{8}$$

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 at a small positive values with `np.clip`. Run your code and compare it with two plots of the basis vectors to the `NMF`

[1] `https://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf`

of `scikit-learn` on the digits. It should perform comparably. Can you observe the non-increasing property of your loss?

Hint: Initialize $\mathbf{W}$ and $\mathbf{H}$ positively, e.g by taking the absolute value of standard normal random variables (RV) with `np.random`.

5 Recommender system (12 Points)

Use your code to implement a recommendation system. We will use the `movielens-100k` dataset with pandas, which you can download from Moodle.

```
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 $\widehat{\mathbf{X}} = \mathbf{WH}$ 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 the user has not seen yet:

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

You can also add ratings for an additional user (yourself) and check if the resulting recommendations make sense. How much does the result vary (due to random initialization) between different runs of the NMF for selected users? Explain why you think this is happening and how good the quality of your results is. Also try to identify rows in $\mathbf{H}$ that can be interpreted as prototypical user preferences (e.g. "comedy fan").

Sidenote: Netflix is still using a similar `SVD++` reconstruction together with a restricted Boltzmann machine (RBM) to give recommendations. [2]

[2] `https://www.quora.com/How-does-the-Netflix-movie-recommendation-algorithm-work`