Exercise 8

Deadline: 30.1.2019

Regulations

Please hand in your solution as a Jupyter notebook `nmf.npynb` along with exported HTML. Zip these files along with your comments on exercise 07 into a single archive with naming convention (sorted alphabetically by last names)

`lastname1-firstname1_lastname2-firstname2_exercise08.zip`

or (if you work in a team of three)

`lastname1-firstname1_lastname2-firstname2_lastname3-firstname3_exercise08.zip`

and upload it to Moodle before the given deadline.

1 Comment on your solution to exercise 7

Study the sample solutions
`kernelized-ridge-regression-solution.html` and `fitting-circles-solution.html`
provided on Moodle and use them to comment on your own solution to this exercise. Specifically, copy your original notebooks to `kernelized-ridge-regression-commented.ipynb` and `fitting-circles-commented.ipynb` and export them to HTML in the end. Insert comments as markdown cells starting with

```
<span style="color:green;font-weight:bold">Comment</span>
```

in order to clearly distinguish your comments from other cell types. The point of these comments is that you identify your errors and bugs yourselves, so that you learn from your mistakes. In addition, the tutor will have an easier time distinguishing between the first mistake and consequential errors caused by the first one and will only deduct points for the former. If you fail to hand in comments, the tutor is not required to make this distinction and will deduct points for all errors alike.

2 Non-negative matrix factorization

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"]
```

2.1 Comparison of scikit-learn's NMF with SVD (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 $\mathbf{X}$ (the matrix of flattened digit images) in the following way:

$$\mathbf{X} = \mathbf{Z} \cdot \mathbf{H} \qquad \text{(NMF)} \tag{1}$$
$$\mathbf{X} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \qquad \text{(SVD)} \tag{2}$$

$\mathbf{X}, \mathbf{Z}, \mathbf{H} \in \mathbb{R}_+$. If $\mathbf{X} \in \mathbb{R}_+^{N \times D}$ and your number of latent components is M then $\mathbf{Z} \in \mathbb{R}_+^{N \times M}$ and $\mathbf{H} \in \mathbb{R}_+^{M \times D}$. Run SVD with full rank and then select the the 6 columns from $\mathbf{V^T}$ corresponding to the largest singular values. Use at least 10 components for NMF and extract the 6 most important rows from $\mathbf{H}$. Note that you must use centered data for SVD (but not for NMF, of course) and add the mean back to the basis vectors. Reshape the selected basis vectors from $\mathbf{H}$ and $\mathbf{V^T}$ into 2D images and plot them. One can interpret these images as a basis for the vector space spaned by the digit dataset. Compare the bases resulting from SVD and NMF and comment on interesting obervations.

2.2 Implementation (8 Points)

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

$$\mathbf{H}_{t+1} \leftarrow \mathbf{H}_t \frac{\mathbf{Z}_t^T \mathbf{X}}{\mathbf{Z}_t^T \mathbf{Z}_t \mathbf{H}_n} \tag{3}$$
$$\mathbf{Z}_{t+1} \leftarrow \mathbf{Z}_t \frac{\mathbf{X} \mathbf{H}_{t+1}^T}{\mathbf{Z}_t \mathbf{H}_{t+1} \mathbf{H}_{t+1}^T} \tag{4}$$

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`. 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?

3 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 $\widehat{\mathbf{X}} = \mathbf{Z} \cdot \mathbf{H}$ 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. How much do results 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: At least until recently, Netflix was using a similar `SVD++` reconstruction together with a restricted Boltzmann machine (RBM) to give recommendations. [1]

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