3. Embedding and Recommending

Objectives

In this lab, you will be able to get hands-on experience with popular methods for the following tasks:

1. dimensionality reduction,
2. recommender systems, and
3. clustering.

In the process, you will also be exposed to some challenges that arise when dealing with large, real-world, noisy data:

- some of the algorithms are computationally demanding for the size of datasets that you will handle, and you will need to distribute the processing using Spark.
- real-world data is sometimes messy, and results need to be interpreted carefully. Use your judgment, don't jump to conclusions, and keep in mind that you can always preprocess the data (removing outliers, normalizing values) if needed.

We expect you to be curious and proactive—we intentionally keep this handout short and want you to look up for the appropriate resources online when needed (e.g., on using Spark and various Python libraries). **Do not hesitate to make use of the TAs**, whether to ask questions or to double-check your code and analyses.

Finally, we expect you to take this lab almost as a small project. *There is no single right way to solve the questions in this lab*, and you should feel free to come up with creative approaches.

Deliverables

Just like in the previous lab, we expect you to hand in an HTML export of the three Jupyter notebooks containing the result of your analyses,

- `lab3-dimred.ipynb`
- `lab3-recsys.ipynb`
- `lab3-cluster.ipynb`

as well as any supplementary file that might be necessary to run your analyses (e.g., Python modules). Keep in mind the four criteria that we use for grading:

1. general **correctness** of the analyses,
2. **code readability** and clarity,
3. **presentation** of the notebook and the figures, and
4. bonus points for **creativity**.

Dataset

The dataset used throughout this lab comes from a movie recommender system, MovieLens (`https://movielens.org/`). Information about the dataset can be found online[1]. It consists of four files, available on HDFS in the folder `/ix/ml-20m`:

[1] See: `http://files.grouplens.org/datasets/movielens/ml-20m-README.html`. Note that we slightly changed the format of the dataset to make it easier for you to process.

movies.txt Each line describes a movie by a `movieId` a `title` and a list of `genres`.

genome-tags.txt Each line describes a tag by a `tagId` and a `tag`.

genome-scores.txt Each line gives a score (`relevance`) for a movie / tag pair (`movieId` and `tagId`).

ratings.txt Each line gives a `rating` for a `userId` / `movieId` pair, as well as a `timestamp`.

It is useful to spend some time getting acquainted with the dataset.

- How many different movies are there? How many tags, tag scores, movie ratings?
- How many movies have at least one tag?
- How many different movie genres are there?
- What are the shortest and longest movie titles? Tag names?

Note: the answer to these questions is not needed for the hand-in.

3.1 Dimensionality reduction

Using the files `genome-tags.txt` and `genome-scores.txt`, a subset of the movies can be expressed in a high-dimensional "tag space". The goal of this part is to embed the movies in a low-dimensional "concept space" based on these tags, using principal component analysis.

Note: there are many ways to compute the eigenvalue decomposition, for example you might want to use `numpy.linalg.eigh`.

Exercise 3.1

Construct an $M \times N$ data matrix, where M is the number of tags and N the number of movies. Do *not* include movies which do not have any tag.

- Plot the variance of each dimension (represented by tags) across all movies.
- Plot the eigenvalues of the $M \times M$ covariance matrix.
- Explain the implications for dimensionality reduction.

For example, how many principal directions do you need to capture $2/3$ of the variability in the data? ■

Exercise 3.2 Concept space

For the five first principal directions, find the 10 tags that have the highest and lowest coordinates in that direction.

- What *concepts* would you use to describe these dimensions?

Create a Python `dict` which maps every tag (name) to its coordinates in the 5 first principal directions and save it to disk. This will be used later for the clustering exercise. ■

We have prepared a short list of movies in the pickle `selected-movies.pickle`[2]. Each movie is described by its movie ID, title, and Rotten Tomatoes score[3]

Exercise 3.3 Movie visualization

Project the movies of `selected-movies.pickle` on the first two principal directions.

- Create an interactive plot that displays the 2D-projection of the movies using `bokeh`

[2] Note that the file uses the UTF-8 encoding. See `snippets.ipynb` for instructions on how to read such pickles.

[3] This score corresponds to the percentage of newspaper reviews that rate the movie favorably. See https://www.rottentomatoes.com/

and its `hover` tool. Color the nodes by their Rotten Tomatoes score.
- Based on your knowledge of these movies and information that you can get from the web, explain the coordinates of a few of the movies.
- How do the PCA directions correlate with the Rotten Tomatoes score?

Try projecting the movies on subsequent principal directions. Does it make sense based on your knowledge of these movies? (*Not needed for the hand-in*). ■

3.2 Recommender systems

We now turn our attention to the task of recommending movies to users. We focus exclusively on the *collaborative filtering* approach, i.e., extracting knowledge from the user / movie rating matrix. This data is given in the `ratings.txt` file.

Important note: *never* load raw rating data in memory. Always use Spark RDD transformations to aggregate the data over users or movies before calling `collect()`.

Exercise 3.4 Basic statistics
First, we will look at basic statistics on this dataset.

- Plot the number of ratings for each user.
- Plot the number of ratings for each item.

Is the number of ratings balanced uniformly across users and movies? ■

To make the exercise more interesting, we are providing a script, `rate-movies.py` that you can use to create a personalized taste profile. Simply execute the script and rate a few movies.

Exercise 3.5 Partitioning the dataset
In order to tune hyperparameters and evaluate the recommender system's performance, we will split the data into a *training* and a *validation* set.

- Append your personal ratings to the MovieLens dataset RDD.
- Partition the data into two sets: $\approx 80\%$ for training, and $\approx 20\%$ for validation.

Hint: A convenient way to split the data is to filter on the last digit of the rating's `timestamp`
■

3.2.1 Baseline recommender system

First, we will consider the following simple model. Let N be the total number of ratings, and N_u and N_m be the number of ratings for user u and movie m, respectively. We predict the rating of user u for movie m as

$$\hat{r}_{um} = \mu + \alpha_u + \beta_m \tag{3.1}$$

where

$$\mu = \frac{1}{N}\sum_{u,m} r_{um} \qquad \text{global average rating,}$$
$$\alpha_u = \frac{1}{N_u}\sum_{m}(r_{um} - \mu) \qquad \text{user bias,}$$
$$\beta_m = \frac{1}{N_m}\sum_{u}(r_{um} - \alpha_u - \mu) \qquad \text{remaining item bias.}$$

Exercise 3.6 Baseline model
You will first implement a recommender system based on model (3.1).

- Compute the global mean μ, the user biases $\{\alpha_u\}$ and the item biases $\{\beta_m\}$ using a sequence of RDD transformations on the training set.
- How many parameters does this model have?
- Predict the rating of every user / movie pair in the validation set.

Note: some users / movies do not have any ratings. Use sensible default values for α_u and β_m. ■

To evaluate the recommender system, we will use the average of each user's root mean square error over the validation set. Letting U be the number of users,

$$\text{error} = \frac{1}{U}\sum_u \sqrt{\frac{1}{N_u}\sum_m (\hat{r}_{um} - r_{um})^2} \tag{3.2}$$

Exercise 3.7 Evaluation
Implement a function `error()` that takes an RDD containing `(userId, movieId, rating)` triplets and computes the error (3.2) with respect to the validation set.

- Use this function to evaluate the baseline predictions that you computed previously.

You should obtain an error of approximately 0.867. ■

3.2.2 Matrix-factorization model

We now consider a more powerful class of models based on factorizing the rating matrix. Spark provides the function `pyspark.mllib.recommendation.ALS()` to learn such models. A typical loop will consist of:

1. learning a model with `ALS()`
2. predicting ratings on the validation set using `model.predictAll()`
3. evaluating the predictions using `error()`

Exercise 3.8 Regularization
Decide on a rank for your model, e.g., between 5 and 25. What is the trade-off between choosing a lower and a higher rank?

- Set the regularization parameter `lambda_` to 10^{-4}. How accurate are the predicted ratings? Can you explain the phenomenon?
- Now set `lambda_` to 10.0. What happens to the predicted ratings, and why?
- Find the value of `lambda_` that minimizes the validation error.

What improvement do you get over the error of the baseline recommender system? ■

Exercise 3.9 Recommendation
Create a Python `dict` that maps from movie IDs to movie titles based on the file `movies.txt`.

- Recommend 10 movies for user 123 using `model.recommendProducts()`. What kind of movies does the model think the user will like?
- Recommend 10 movies to yourself (user ID: 138494).

What do you think of your recommendations? :-) ■

The matrix factorization model can also be intepreted as embedding users and items in a low-dimensional space.

Exercise 3.10 Visualisation
Learn a rank-2 matrix-factorization model using `ALS()`.

- Extract the features for all the movies in `selected-movies.pickle`.
- Create an interactive plot that embeds the movies along the 2 directions defined by the factorization.
- Describe what you observe. Can you give a name to the dimensions? Do you recognize cluster of movies that are alike?

Note: you can reuse some of the code that you wrote for Exercise 3.3. ■

3.3 Clustering

To conclude this lab, we will turn to clustering algorithms in order of surface additional structure in the MovieLens data.

1. First, you will use k-means on the low-dimensional embedding of tags that your extracted earlier.
2. Second, you will implement a variant of k-means that enables the use of non-Euclidean distances, and you will cluster movies based on genres.

For the first part, you can either implement k-means yourself or use an existing implementation (e.g. in `scikit.cluster`).

Exercise 3.11 Clustering tags
Load the tag embedding that you created at the end of Exercise 3.2.

- Cluster the data using the k-means clustering algorithm. Try $k \in \{2,\dots,5\}$.
- Visualize the clusters using an interactive `bokeh` plot.
- Try projecting on different principal directions.

Which principal directions separate the clusters well? ■

Now, we consider the task of clustering the space of movies based on the set of genres associated to each movie. For this, we will use an alternative distance based on the *Jaccard index*, defined for two sets A and B as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

From the Jaccard index one can define a distance as $d(A,B) = 1 - J(A,B)$. Compared to a Euclidean space equipped with the standard ℓ^2-distance, two challenges arise:

1. there is no more a clear concept of *center* of a set of points, and
2. visualizing the space becomes difficult.

To overcome the first challenge, we adapt the M-step of the k-means algorithm: instead of computing cluster centers, we pick a representative point that belongs to the cluster and that minimizes the distance to all other points. The resulting method is described in Algorithm 3.1.

Algorithm 3.1 k-medioids

Require: set of points X, distance function $d : X \to X$, number of clusters k

for i = 1, ..., k **do** ▷ Initialize medioids.
  $m_i \leftarrow \text{random}(X)$
repeat
  $C_1, \ldots, C_k \leftarrow \varnothing$
  for $x \in X$ **do** ▷ Assign points to clusters.
    $i \leftarrow \arg\min_i \{d(m_i, x)\}$
    $C_i \leftarrow C_i \cup \{x\}$
  for $i = 1, \ldots, k$ **do** ▷ Recompute medioids.
    $m_i \leftarrow \arg\min_{x \in C_i} \{\sum_{y \in C_i} d(x, y)\}$
until convergence

Exercise 3.12 **Clustering movies**

Create a `dict` that maps movie IDs to set of genres from the data in `movies.txt`.

- Implement the k-medioids algorithm with the Jaccard distance.
- Cluster the set of movies in the file `most-rated.pickle`, using $k = 2$.
- Find a good way to visualize the results of the clustering. For example, you could try to visually represent the frequency of the genres in each of the clusters.

How do you intepret the two clusters? ■