In this session:

- We'll have a quick look at the surprise library for creating recommenders in python
- We'll compare our implementation of CF with surprise's
- We'll run a matrix factorization algorithm in surprise, and investigate whether the latent factors it discovers make any sense.

1 Surprise

Surpsise is a Python library that contains a number of implementations of recommendation algorithms, and auxiliary code, for example for loading rating datasets and for evaluating recommenders.

You can install it with `pip` or whatever:

```
$ pip install numpy
$ pip install scikit-surprise
```

Try the test in the surprise homepage, `http://surpriselib.com/`:

```
from surprise import SVD
from surprise import Dataset
from surprise.model_selection import cross_validate

# Load the movielens-100k dataset (download it if needed).
data = Dataset.load_builtin('ml-100k')

# Use the famous SVD algorithm.
algo = SVD()

# Run 5-fold cross-validation and print results.
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

You see that surprise has its own notion of Dataset, and has some built-in datasets but can read external ones as well.

2 Task 1

Starting from this scheme,

- Go to the Documentation page, check the Nearest-Neighbor-based prediction algorithms, and see the differences.
- `surprise.prediction_algorithms.knns.KNNWithMeans` is probably the closest to the user-to-user CF algorithm that you implemented last week.
- Get surprise to read the same movie rating file that you used the other day (check the `Reader` class).
- And see if the results you get are similar to those you obtained last week.

In the last point, we do not expect a formal evaluation using MAE or cross-validation (since you did not implement it last week). Write code that lets you enter a userid, a moveid, gets the rating predicted by surprise's algorithm, and prints it.

If the algorithms are similar, movies that your system recommends to a particular userid should have high rating by the new algorithm too. Try also ratings of movies that look very different from those that a user liked.

Do at least 2-3 tests.

3 SVD, recommending via matrix factorization

You can see the code of the SVD recommendation algorithm that we tested first here:

`https://github.com/NicolasHug/Surprise/blob/master/surprise/prediction_algorithms/matrix_factorization.pyx`

These recommendation have a `fit(self, trainset)` method that does the matrix decomposition, and a `estimate(self, u, i)` method that then predicts the rating of item i by user u.

If you look at the documentation of the code you will see that a fitted SVD object has two attributes among others:

```
...
pu(numpy array of size (n_users, n_factors)): The user factors (only
            exists if ``fit()`` has been called)
qi(numpy array of size (n_items, n_factors)): The item factors (only
            exists if ``fit()`` has been called)
...
```

These are the two matrices with orthonormal columns discussed in the course slides. They are created in `fit`, using gradient descent, and then used in `estimate`.

Read the code of `estimate`. The recommender can be biased or unbiased, via a flag set at inicialization. The unbiased one is like in the course slides. The predictor is biased by default; it corrects the prediction using user averages and item averages. It makes the predictions more stable if the user or item has few ratings, and it also allows to give some decent prediction if user or item have no ratings.

We propose the following:

- Train a SVD predictor on the same dataset and a moderate number of factors (5 to 10, say)
- Get the `qi` attribute of the predictor. Each column of this attribute is the description of one of the latent factors.
- Write code that, for each factor, gets the top-something movies with highest weight in the factor, and prints the movie names and weight.
- Do they make some sense? For example, these could make sense:
  - "Four weddings and a funeral"+"Pretty Woman"+"Bridget Jones's diary"
  - "Alien"+"2001, a space odissey"+"Interstellar"
  - "Nightmare in Elm Street"+"Alien"+"The Texas Chain Saw Massacre"
  - "Gladiator"+"The Seven Samurais"+"Macbeth"

(These are imaginary examples. We have not even checked if all these movies are in the dataset. But you get the idea)

Give some examples in your report of what you find.

4 Deliverables

Rules: Same rules as in previous labs about working solo/in pairs and plagiarism.

To deliver: You must deliver a zip file containing 1) the code you wrote (probably several files) 2) a report (2 pages max) with the examples you tried and results; the difficulties you found (only if you had any significant difficulty), plus anything you found interesting or frustrating about this lab.

Procedure: Submit your work through the Racó as a single zipped file.

Deadline: Work must be delivered within **2 weeks** from the lab. Late deliveries risk being penalized or not accepted at all. If you anticipate problems with the deadline, tell me as soon as possible.