Methods of classification on dimensionality reduction 23.02.2022

Project 1

Paweł Lorek

Deadline: 23.04.2022

You should provide: A report in a `.pdf` file and a file (or files) in Python (ver 3). The report should be comprehensible also for a person not attending the lecture: thus, not only the results should be reported, but also the statement of the problem, explanations of your solution, etc.

Project can be done by one person or a group of two persons.

Remark: Below you will find what should be present in your project. Your project should include all these elements, but it does not have to be limited only to them.

Recommender system
Part 1: SVD + NMF (15 points)

Download a file ml-latest-small.zip from `https://grouplens.org/datasets/movielens/`. Extract from it `ratings.csv` (we are interested only in this file). There are around 100000 ratings – around 600 users rated around 9000 movies. Its format is following:

```
userId,movieId,rating,timestamp
1,1,4.0,964982703
1,3,4.0,964981247
1,6,4.0,964982224
1,47,5.0,964983815
1,50,5.0,964982931
...,
```

where `userId` is a unique user id, `MovieId` is a unique movie id, `rating` is the rating 0–5 (integer). (there is also `timestamp` – we will not use it).

Split the file `ratings.csv` into two files `train_ratings.csv` (train set) and `test_ratings.csv` (test set) randomly, so that `training_ratings.csv` contains around 90% of ratings of each user (and `test_ratings.csv` contains the remaining ones).

The task is to "predict" ratings from `test_ratings.csv`.

It may be convenient to convert the above data into matrices. Let $\mathbf{Z}$ be a matrix containing training ratings – a matrix of size $n \times d$, where n is the number of users and d is the number of movies. Thus, the presented fragment of `ratings.csv` – assuming it is all in a training set – is converted to: (recall, numbering in Python starts with 0)

$$\mathbf{Z}[0,0] = 4.0,\ \mathbf{Z}[0,2] = 4.0,\ \mathbf{Z}[0,5] = 4.0,\ \mathbf{Z}[0,46] = 5.0,\ \mathbf{Z}[0,49] = 5.0, \ldots$$

Of course $\mathbf{Z}$ is *sparse* – many entries of $\mathbf{Z}$ are not defined (either there is no such pair at all, or it is in the test set).

Let $\mathcal{T}$ denote a set of pairs (u, m) with ratings present in a test set – then the rating is given by $\mathbf{T}[u, m]$.

(In other words, from the file `train_ratings.csv` we create the matrix $\mathbf{Z}$, whereas from the file `test_ratings.csv` we analogously create the matrix $\mathbf{T}$).

Quality of the system. Assume that your algorithm after training on $\mathbf{Z}$ computes $\mathbf{Z}'$, a matrix containing elements $\mathbf{Z}'[u, m]$ for $(u, m) \in \mathcal{T}$. Then *the quality* is computed as **root-mean square error**:

$$\texttt{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,m)\in\mathcal{T}} (\mathbf{Z}'[u, m] - \mathbf{T}[u, m])^2}. \qquad (1)$$

Your program should contain command-line option `--alg`, which is either NMF, SVD1, SVD2 or SGD.

- NMF: Recall that NMF approximates $\mathbf{Z}$ of size $n \times d$ as

  $$\mathbf{Z} \approx \mathbf{WH},$$

  where $\mathbf{W}$ is of size $n \times r$ and $\mathbf{H}$ is of size $r \times d$ (r is a parameter).

  Typical usage in Python:

```
from sklearn.decomposition import NMF
import numpy as np

model = NMF(n_components=r, init='random', random_state=0)
W = model.fit_transform(Z)
H = model.components_
Z_approximated = np.dot(W,H)
```

  More on NMF in scikit-learn (Python's library) and its parameters:

  `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html`

  This implementation works only for a full matrix $\mathbf{Z}$. That is why first missing values should be imputed. You should come up with some solution (e.g., you can put 0 everywhere; the mean rating (of all users) of the film can be placed in missing one, or only the mean rating of a single user, etc. Choose whatever you think will give the best RMSE).

- SVD1

  The same what above, but instead of NMF use SVD. Recall that SVD approximates $\mathbf{Z}$ as

  $$\mathbf{Z} \approx \mathbf{U}_r \mathbf{\Lambda}_r \mathbf{V}_r^T$$

(see the lecture notes), let us denote $\mathbf{W} = \mathbf{U}_r, \mathbf{H} = \mathbf{\Lambda}_r \mathbf{V}_r^T$, to be consistent with NMF.

This is how you may use SVD and "extract" matrices $\mathbf{W}$ and $\mathbf{H}$ in Python:

```
from sklearn.decomposition import TruncatedSVD
import numpy as np

svd=TruncatedSVD(n_components=r,random_state=42)
svd.fit(Z)
Sigma2=np.diag(svd.singular_values_)
VT=svd.components_

W=svd.transform(Z)/svd.singular_values_
H= np.dot(Sigma2,VT);
```

More details:

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

- SVD2

  Use the procedure described will be described during lecture or labs.

- SGD: described in last section

Python's script. The script should be named `recom_system_IndexNr.py`, where `IndexNr` is a student index number (in case of a group work – please name it `recom_system_IndexNr1_IndexNr2.py`)

It must take the following parameters (you should use `argparse` library).

```
python3 recom_system_IndexNr.py --train train_file --test test_file \\
--alg ALG --result result_file
```

where

- `train_file` is a file with the training data (`train_ratings.csv`)
- `test_file` is a file with the test data (`test_ratings.csv`)
- `ALG` is one of the algorithms `NMF, SVD1, SVD1, SGD` (note – uppercase)
- `result_file` is a file where a final RMSE will be saved (only this number should appear in this file)

REMARK: Both `train_file` and `test_file` will be of the same format as `ratings.csv` – in particular the first row will be `userId,movieId,raging,timestamp`; it will still contain column `timestamp`; ratings will be stored as floats, e.g. 4.0 (although they are integers).

Evaluation of your work. Except evaluation of a `.pdf` file itself, described ideas etc., the results (RMSEs) of your script will be taken into account. It will be evaluated on randomly chosen pair (or pairs – then the average RMSE will be computed) (`train_ratings.csv`,`test_ratings.csv`) for each option NMF, SVD1, SVD2 separately. In report you can describe several ideas you tried, but you should implement only the one which you think will give the best (i.e., the smallest) RMSE, for each option NMF, SVD1, SVD2 separately.

Part 2: Stochastic Gradient Descent SGD (5 points)

In previous methods we needed first to impute missing values in the matrix $\mathbf{Z}$. Let us reformulate the problem: For a given $\mathbf{Z}$ of size $n \times d$ we want to find matrices $\mathbf{W}$ of size $n \times r$ and $\mathbf{H}$ of size $r \times d$ in the following way:

$$\arg\min_{\mathbf{W},\mathbf{H}} \sum_{(i,j):z_{ij}\neq '?'} (z_{ij} - \mathbf{w}_i^T \mathbf{h}_j)^2$$

or ($\lambda > 0$ is a parameter)

$$\arg\min_{\mathbf{W},\mathbf{H}} \sum_{(i,j):z_{ij}\neq '?'} (z_{ij} - \mathbf{w}_i^T \mathbf{h}_j)^2 + \lambda \left(||\mathbf{w}_i||^2 + ||\mathbf{h}_j||^2 \right)$$

(here $\mathbf{h}_j$ is j-th column of $\mathbf{h}$, whereas $\mathbf{w}_i^T$ is i-th row of $\mathbf{W}$).

There are many methods for finding a minimum (or its approximation) of a function of this type (it is a function of matrices $\mathbf{W}$ and $\mathbf{H}$, but thinking that this is $f(w_{11}, w_{12}, \ldots, h_{11}, h_{12}, \ldots)$ – it is simply a function of $r(n+d)$ variables). *Stochastic gradient descent* (SGD) is one of the methods.
Find how it works in more details and implement it. Tune parameters so that the final RMSE is “the best”.