

TDT 4215 – Recommender System

Group Project: Recommender System Using Adressa Dataset

Original Authors:

Lemei Zhang (lemei.zhang@ntnu.no)

Peng Liu (peng.liu@ntnu.no)

last modified: 18/01/2021

Introduction

The project dataset is a refined version of the Adressa dataset published by the SmartMedia group at NTNU in partnership with the local newspaper Adresseavisen in Trondheim.

This dataset includes anonymized user data from local digital newspaper from 01.01.2017 to 31. 03. 2017 (3 months in total). We filter 1000 most active users from the original dataset, and select 9 attributes that we think most relevant for the project. The dataset is 85M in a compressed size.

Full articles (in Norwegian) from the newspaper can also be provided upon your request if you would like to use more details about the text in your recommender system. For this, text processing and the use of natural language processing tools are required.

The information of the original dataset and documentations are available below in case you want to know more on the dataset: <http://reclab.idi.ntnu.no/dataset>

Project Dataset

Overview

The datasets are saved under the directory called *active1000* and files are saved by dates.

```

ls active1000
20170101 20170111 20170121 20170131 20170210 20170220 20170302 20170312 20170322
20170102 20170112 20170122 20170201 20170211 20170221 20170303 20170313 20170323
20170103 20170113 20170123 20170202 20170212 20170222 20170304 20170314 20170324
20170104 20170114 20170124 20170203 20170213 20170223 20170305 20170315 20170325
20170105 20170115 20170125 20170204 20170214 20170224 20170306 20170316 20170326
20170106 20170116 20170126 20170205 20170215 20170225 20170307 20170317 20170327
20170107 20170117 20170127 20170206 20170216 20170226 20170308 20170318 20170328
20170108 20170118 20170128 20170207 20170217 20170227 20170309 20170319 20170329
20170109 20170119 20170129 20170208 20170218 20170228 20170310 20170320 20170330
20170110 20170120 20170130 20170209 20170219 20170301 20170311 20170321 20170331

```

In each file, every line represents one clicking event occurred by users in JSON format. In Python, you can use *json* package and *json.loads()* to read events:

```

import json

for line in open(fname):
    obj = json.loads(line.strip())

```

Then use *pprint* package can have a glimpse at the event:

```

In [90]: import pprint

In [91]: pp = pprint.PrettyPrinter(indent=4)

In [92]: pp.pprint(obj)
{  u'activeTime': 37,
   u'category': u'nyheter|sortrondelag',
   u'documentId': u'3a77a5a627c60c02d40440ea394cb8afb2791862',
   u'eventId': 1450324853,
   u'publishtime': u'2017-01-01T20:04:30.000Z',
   u'time': 1483311582,
   u'title': u'Johanna er \xe5rets nyeste tr\xfbnder',
   u'url': u'http://adressa.no/nyheter/sortrondelag/2017/01/01/johanna-er-%c3%a5rets-nyeste-tr
%c3%b8nder-14002903.ece',
   u'userId': u'cx:hzxwfhnad3y1r0h0:1iteimihpr31'}

```

We can see there are 9 attributes in one event: *activeTime*, *category*, *documentId*, *eventId*, *publishtime*, *time*, *title*, *url* and *userId*. Note that not all attributes have values. If some attributes have no value, there will be a *None* type instead.

```

{  u'activeTime': None,
   u'category': None,
   u'documentId': None,
   u'eventId': 996850947,
   u'publishtime': None,
   u'time': 1483311602,
   u'title': None,
   u'url': u'http://adressa.no',
   u'userId': u'cx:iafvru1j9yajwnu:1ualyvwzsbtkq'}

```

Basic Statistics

The table below shows some basic statistics of the dataset.

Item	Value ¹
Total number of events (front page incl.) ²	2,207,608

¹ Values in brackets are values after dropping duplicates

² “front page” event represents that users clicked only front page “http://adressa.no”.

Total number of events (without front page)	788,931
Number of events (drop duplicates) ³	679,355
Number of documents (news articles)	20,344
Number of users	1000
Max number of events per user	7,960 (7,958)
Min number of events per user	181 (59)
Average number of events per user	788.931 (679.355)
Sparsity	3.878% (3.339%)

In table above, Sparsity equals 3.878% means that 3.878% of user-ratings have a value. Note that, although we fill the missing values with 0, we should not assume that these values are truly zero.

Examples

We offer two recommendation examples. Source code for Python 2.7 is available online (http://reclab.idi.ntnu.no/project_example.py, <http://reclab.idi.ntnu.no/ExplicitMF.py>) and on BlackBoard for Python 3.

1. Collaborative Filtering

Collaborative Filtering (CF) is a widely adopted recommendation algorithm. The fundamental assumption of CF is that if user X and Y rate n items similarly, or have similar behaviors (such as buying, rating, clicking, listening), and hence will rate or act on other items similarly.

There are many kinds of CF and CF extended algorithms online nowadays. In this doc, we will introduce the *Explicit Matrix Factorization (MF)* as an example. Students can realize their own algorithms based on this code. MF is based on the assumptions of: 1) each user can be described by k features; 2) each item can be described by k attributes; 3) predicted value of rating or clicking probability of an item can be represented by the summation of each multiplication of user feature value and item feature value. We will not elaborate MF in this doc. Students can google with keyword of *Explicit Matrix Factorization* or [here](#). Our code implementation of MF is also based on this blog. Difference is that we assume the ratings of clicked items are 1 and otherwise 0 in user-item matrix.

Before MF, we split our data into training and test sets by randomly choosing a fraction of ratings per user from the whole dataset in function `train_test_split(ratings, fraction)`.

³“drop duplicates” here means drop duplicate events according to userId and documentId. This operation based on the assumption that user refreshes web page will also bring in new event.

```

def train_test_split(ratings, fraction=0.2):
    """Leave out a fraction of dataset for test use"""
    test = np.zeros(ratings.shape)
    train = ratings.copy()
    for user in xrange(ratings.shape[0]):
        size = int(len(ratings[user, :].nonzero()[0]) * fraction)
        test_ratings = np.random.choice(ratings[user, :].nonzero()[0],
                                       size=size,
                                       replace=False)
        train[user, test_ratings] = 0.
        test[user, test_ratings] = ratings[user, test_ratings]
    return train, test

```

The evaluation of MF is according to *MSE* (detailed definition is in the next chapter). The output results of each iteration are shown below:

```

Iteration: 1
Train mse: 0.6027160643041712
Test mse: 0.6896122222140042
Iteration: 2
Train mse: 0.5356027170531528
Test mse: 0.6425526901669532
Iteration: 5
Train mse: 0.5137733963167571
Test mse: 0.6274973344350155
Iteration: 10
Train mse: 0.5107244574314569
Test mse: 0.6252450668436269
Iteration: 25
Current iteration: 10
Train mse: 0.50972724333495
Test mse: 0.624080568994183
Iteration: 50
Current iteration: 10
Current iteration: 20
Train mse: 0.5096093258024226
Test mse: 0.6238224694470061
Iteration: 100
Current iteration: 10
Current iteration: 20
Current iteration: 30
Current iteration: 40
Current iteration: 50
Train mse: 0.509587065936661
Test mse: 0.6237738948389052

```

2. Content-based Recommendations

Content-based recommendations are another popular used recommendation methods. They make recommendations by analysing the content of textual information and finding regularities in the content. In this example, we adopt TF-IDF (Term Frequency – Inverse Document Frequency) for feature selection and Cosine similarity to find the most similar items with user clicking before.

TF-IDF can be implemented with help of scikit-learn, a useful Python package for machine learning tasks. Specifically, `TfidfVectorizer` Convert a collection of raw documents to a matrix of TF-IDF features.

```

from sklearn.feature_extraction.text import TfidfVectorizer

```

```
# select features/words using TF-IDF
tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0)
tfidf_matrix = tf.fit_transform(df_item['category'])
print('Dimension of feature vector: {}'.format(tfidf_matrix.shape))
```

output the dimension of feature matrix:

```
| Dimension of feature vector: (20393, 169)
```

Then we use cosine similarity to measure the similarity of two items:

```
from sklearn.metrics.pairwise import linear_kernel

cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

The recommendation results can be a rank list of all candidate items according to the cosine similarities with the last clicking item. The evaluation is according to *Recall@k* and *ARHR@k*. The detailed definition of Recall and ARHR can be found in the next chapter. The results are shown as bellow (k=20):

```
Recall@20 is 0.0070
ARHR@20 is 0.0006
```

Evaluation Criteria

1. Recall (Hit Rate)

True positive (tp): the number of positive instances that are correctly predicted.

True negative (tn): the number of negative instances that are correctly predicted.

False negative (fn): the number of mispredicted negative instances.

False positive (fp): the number of mispredicted positive instances.

Recall is used to measure the fraction of positive instances that are correctly predicted, which can be defined as

$$Recall = \frac{tp}{tp + fn}$$

2. CTR (Click Through Rate) is the number of recommendations produced by a participating system that are clicked by users normalised by the total number of requests for recommendations that were sent to that system. Example: Participant “rocking recommendations” receives 100,000 recommendation requests.

The system manages to provide valid, in-time suggestions in 95,000 cases. Users click on 4,500 suggestions. We compute a CTR of $4,500 / 100,000 = 4.5\%$.

3. ARHR

The third measure that is commonly used, is the average reciprocal hit rate (ARHR). This measure is designed for implicit feedback data sets, in which each value of $r_{+} \in \{0,1\}$. Therefore, a value of $r_{+} = 1$ represents a “hit” where a customer has bought or clicked on an item. A value of $r_{+} = 0$ corresponds to a situation where a customer has not bought or clicked on an item. In this implicit feedback setting, missing values in the ratings matrix are assumed to be 0. Then, the ARHR metric for the user u is defined as follows:

$$ARHR(u) = \sum_{j \in I_u} \frac{r_{uj}}{v_j}$$

where v_j is the rank of item j in the recommended list, I_u represents the set of items rated by user u .

4. MSE

Mean Squared Error (MSE) is a widely used predictive accuracy metric. It takes the sum of the squared difference between the user’s rating/score and the predicted rating/score and divides it by the number of items considered.

$$MSE = \frac{1}{|I|} \sum_{b \in I} (r(b) - \hat{r}(b))^2$$

Where I represents the items in the test dataset, r represents the observed value, \hat{r} represents the predicted value.

Links you may find useful

- LensKit library: <https://lenskit.org/>
- Machine Learning in Python - scikit-learn: <http://scikit-learn.org/stable/>
- Natural Language Toolkit: <http://www.nltk.org/>
- Numpy – Scientific computing with Python: <http://www.numpy.org/>
- Pandas – Python Data Analysis Library: <https://pandas.pydata.org>
- Apache Mahout: <https://mahout.apache.org/>
- Apache Spark – Mlib: <https://spark.apache.org/mllib/>
- TensorFlow: <https://www.tensorflow.org/>
- GNU Octave and Octave-Forge for various scientific programming:
<https://www.gnu.org/software/octave/>
<https://octave.sourceforge.io/packages.php>
- Turi (GraphLab) Create: <https://github.com/apple/turicreate>
- Keras – Deep learning library: <https://keras.io/>
- Theano: <https://github.com/Theano/Theano>
- Weka – Data mining in Java: <https://www.cs.waikato.ac.nz/ml/weka/>
- PyTorch: <https://pytorch.org/tutorials/>

More sources from the book, Recommender Systems: The Textbook, Charu C. Aggarwal:

<http://charuaggarwal.net/Recommender-Systems.htm>

