Recommender Systems: Data Notes

Aaron Tuor

August 21, 2015

Contents

1 SNAP Amazon Reviews Data

1.1 Overview

Attributes:

- Reviews: 34,686,770
- Users: 6,643,669
- Products: 2,441,053
- Users with $>$ 50 Reviews: 56,772
- Product Categories: 28
- Timespan: Jun 1995 - Mar 2013
- Product description file in metadata

Review File Data Format

product/productId: B00006HAXW

product/title: Rock Rhythm and Doo Wop: Greatest Early Rock

product/price: unknown

review/userId: A1RSDE90N6RSZF

review/profileName: Joseph M. Kotow

review/helpfulness: 9/9

review/score: 5.0

review/time: 1042502400

review/summary: Pittsburgh - Home of the OLDIES

review/text: I have all of the doo wop DVD's and this one is as good or better than the 1st ones. Remember once these performers are gone, we'll never get to see them again. Rhino did an excellent job and if you like or love doo wop and Rock n Roll you'll LOVE this DVD !!

Product Description File Data Format

product/productId: B00006HAXW

product/description: Whether you're hoping to obtain a raise from your boss...(no new lines till end of description)

The original interest in this data set was for experimenting with models for cross-domain collaborative filtering. People in the group have also used this data set for developing topic and sentiment models using review text. Currently we are using the numerical review data along with the associated product descriptions to develop matrix factorization based models for addressing the cold start problem.

The data was originally obtained from:

https://snap.stanford.edu/data/web-Amazon.html

There is a newer incarnation of this dataset here:

http://jmcauley.ucsd.edu/data/amazon/

1.2 Data preparation

The models we are interested depend on two matrices, a user-item utility matrix R, and a document term occurrence matrix, D.

The entries in D are such that D_{ij} = the number of times word i occurs in the description of item j.

The entries in R are such that R_{ij} = user i's review (a score between 1 and 5) of item j.

The first step in data preparation was to put the data in a tab delimited sparse matrix format that is friendly with matlab.

Issues

1. There is a user labeled "unknown" which must be removed from the dataset. (Amazon used to allow unregistered users to rate and review items, all of which got lumped into the label "unknown".) This unknown user has on the order of a million ratings in the data set. There are other users which have on the order of 10,000 ratings in the data set. These high numbers of ratings are due to the fact that Amazon has some reviewers that receive items from companies in order to review them. The most prolific of these semi-professional reviewers have around 30,000 reviews. We thought about removing these reviewers from the dataset but I am averse to doing so unless we have some principled way (something that doesn't seem arbitrary) of determining the cutoff.
2. User and Item ID's must be mapped to integers between 1 and num users and 1 and numitems respectively
3. We are only concerned with ratings for items which have an associated product description in the metadata. So, some ratings must be filtered out when there is no product description in the metadata.
4. In order to construct the document term occurrence matrix we have to do some text processing.
   i Normalize the text (Remove punctuation).
   ii Create a dictionary: Some words are too uncommon to have meaningful correlations with other words (uncommon mispellings, random integers, etc.). A dictionary needs to be created that contains the words we wish to count from the product descriptions.
   iii Count words and print files.

For the most part these issues were unanticipated. As a result they were dealt with one by one, as they were discovered and so the particular solutions outlined below (which work) are open to improvement.

Steps: Below are the steps for processing Amazon Movies Review data. The scripts used, unless otherwise mentioned are at:

`/home/hutch_research/projects/recommender_systems/aaron_copy/scripts/data_processing/`

Processing Reviews Data

1. Use `AmazonSparseMaker.py` to generate a tab delimited sparse format file from `Movies_TV.txt` (review file data format). The usage is:

   `python ...path.../AmazonSparseMaker.py <reviewfilepath> <sparsefilepath> <filename>`

   where:

   `<reviewfilepath>` is the location of `Movies_TV.txt`

   `<sparsefilepath>` is the location where you want the sparse data file which will be created named `<filename>`.

   This script automatically filters out the unknown user. The script will create a folder if one doesn't already exist at `<sparsefilepath>`. A `readme` is created which gives the stats on the reviews.
   A list of items contained in the review file is also created at `<sparsefilepath>` for use in the next step.

2. Use `AmazonProductDescListMaker.py` to generate a file containing product descriptions for products contained in `Movies_TV.txt`. The usage is:

   `python ...path.../AmazonProductDescListMaker.py <readfilepath> <writefilepath> <namestub>`

   where:

   `<readfilepath>` is the path to `descriptions.txt` including filename. (for us `/home/hutch_research/data/amazon_reviews/descriptions.txt`).

   `<writefilepath>` is the path to `Movies_TV.description` file you wish to create (without filename).

   `<namestub>` is the name of the .sparse file minus the file extension (in this case `Movies_TV`) created in step 1.

3. Use `filterSparse.py` To remove entries for items not found in `descriptions.txt` from the sparse format file created in step 1. Usage:

   `python ...path.../filterSparse.py <path> <sparsefilename> <itemDescriptionFile>`

   where:

   `<itemDescriptionFile>` is the name of the .descriptions file made in step 2

   `<sparsefilename>` is the name of the .sparse file made in step 1

<path> is the path where the above files are located

4. Use `sparsemap.py` to create .sparse file where Amazon IDs have been mapped to integer IDs. Usage:

   `python ...path.../sparsemap.py <path> <filename>`

   where:

   <filename> is the name of the filtered .sparse file made in step 2

   <path> is the path where the above file is located

   Three files will be created at <path>. A .sparse file (named with `_mapped` infixed between the namestub and extension of the original file), a .txt file recording the user ID mappings and a .txt file recording the item ID mappings.

Processing Product Description Data

1. See step 2 in Processing Reviews Data above
2. Use `descriptionMap.py` to map amazon IDs in `Movies_TV.descriptions` file to corresponding integer IDs. Mappings are found in `Watches_filtered_itemmap.txt`. Usage:

   `python ...path.../descriptionMap.py <path> <descriptionFilename> <itemMapFilename`

   where:

   <descriptionFilename> is the name of the product description file you wish to map Ids for.

   <itemMapFilename> is the name of the file containing the mappings.

   <path> is the location of the above files.

   A new description file with integer IDs replacing amazon IDs will be created at <path> using the following naming convention. `_mapped` will be infixed between the namestub and file extension of the original file name. For instance if `Movies_TV.descriptions` is the name of the original file then `Movies_TV_norm.descriptions` is the name of the file containing the normalized text.

3. Use `normalize.py` to remove punctuation that is not important to the semantics of individual word tokens. Usage:

   `python ...path.../normalize.py <infile>`

   where:

   <infile> is the name of the text file to be normalized (in this case `Movies_TV_mapped.descriptions`).

A new file of the normalized text will be created automatically by the following convention. `_norm` will be infixed between the namestub and file extension of the original file name. For instance if `Movies_TV_mapped.descriptions` is the name of the original file then `Movies_TV_mapped_norm.descriptions` is the name of the file containing the normalized text.

4. Use `generateVocab.pl` to create a dictionary of words contained in .descriptions file. We don't want the item IDs contained in our dictionary so the first step is to remove the first token from each line of the .descriptions file. This can be accomplished using the Unix terminal command below:

   `cut -f2- -d" " <descriptionFile> > <cutDescriptionFile>`

   where:

   `<descriptionFile>` is the name of the .description file we wish to remove item IDs from.

   `<cutdescriptionFile>` is the name of the file were the descriptions minus the item IDs will be written to.

   Next you need to type in the terminal:

   `export PERL5LIB=$PERL5LIB:/home/hutch_research/lib`

   I'm not sure what this does exactly but I think it ensures that the script can access the appropriate libraries. Now we can use `generateVocab.pl` located at `/home/hutch_research/bin/` on the cut .description file. Usage:

   `/home/hutch_research/bin/generateVocab.pl -c <integer> <descriptions> > <dictionary`

   where:

   `<integer>` is the number of occurrences a word must have to be in the dictionary.

   `<descriptions>` is the name of the file the dictionary is being compiled from.

   `<dictionary>` is the name of the resulting dictionary file.

5. Use `generateTermDoc.py` to create a .sparse file of term/Doc counts for each product. Format: `<term> <doc> <count>`. Usage:

   `python ...path.../generateTermDoc.py <path> <dictionaryfilename> <descriptionfilena`
   `<termDocfilename>`

   where:

   `<dictionaryfilename>` is the name of the dictionary file created in step 4.

`<descriptionfilename>` is the name of the normalized .description file created in step 3.

`<termDocfilename>` is the name of the .sparse file being created.

`<path>` is the location of the above files.

1.3 Automated Processing

The script which automates the above steps is `processAmazon.py`. Usage:

```
python ...path.../processAmazon.py <reviewfilename> <datadirectory>
```

where:

1.4 Data Split

The data is split up in a particular way in order to train and test for cold start scenarios. Once the data is prepared in the .sparse format the ratings observations are split as follows:

`userdevcold` $\approx$ 5 percent of ratings where entire rows of user ratings are removed from training data.
`usertestcold` $\approx$ 5 percent of ratings where entire rows of user ratings are removed from training data.
`itemdevcold` $\approx$ 5 percent of ratings where entire columns of item ratings are removed from training data.
`itemtestcold` $\approx$ 5 percent of ratings where entire columns of item ratings are removed from training data.
`bothdevcold` $\approx$ .5 percent of ratings
`bothtestcold` $\approx$ .5 percent of ratings
`dev` $\approx$ 5 percent of remaining ratings after removal of `devcold` and `testcold` columns.
`test` $\approx$ 5 percent of remaining ratings after removal of `devcold`, `testcold`, and `dev`.
`train` $\approx$ the remaining 80 percent of ratings.

This split is accomplished by running the matlab script `coldsplitword.m`:

Usage:

`matlab` (enter matlab environment)
`coldsplitword(fullratings.sparse, termDoc.sparse, newfile.mat)`;

where
`fullRatings.sparse` is the .sparse file created in the previous steps for ratings

`termDoc.sparse` is the .sparse file created in the previous steps for words.

`newfile.mat` is the name of the .mat file with the created matrices.