The Million Song Dataset Challenge

Getting Started

By the end of this document, you should be ready to make a first submission in the Million Song Dataset Challenge on Kaggle.

The outline follows these five steps:

1. register on the Kaggle website,
2. acquire the training data,
3. write a Python script that computes song popularity,
4. create and verify the solution file, and
5. upload it to Kaggle.

A stand-alone implementation of the code described in this document can be found on the MSD Challenge github under the examples directory: popularity.py.

1. Registering on Kaggle

The contest is located at http://www.kaggle.com/c/msdchallenge. Make sure to read the general statement and the rules about the challenge before moving forward. When you're ready, click on **sign up** to create an account. Then go back to the MSD challenge page and click "**ENTER THE COMPETITION**". Finally, agree to the rules and download the data.

That's it. Welcome to the MSD Challenge!

2. Acquiring the data

Before you can really get started, you will need to download the core data: triplets (*user, song, play count*). Each user's listening history is defined in terms of these triplets. There are ~1,110,000 unique users, split into two subsets:

- ~1M users as training data, for which you have access all triplets, and
- 110K users as testing data, for which you you have access to only half of their triplets.

In this tutorial, we will ignore the full training data, but you can access everything from the Million Song Dataset website.

From http://www.kaggle.com/c/msdchallenge/data, you will need to download the following files:

- `kaggle_visible_evaluation_triplets.txt`
- `kaggle_users.txt`
- `kaggle_songs.txt`

The first file contains the visible part of the testing data. The second file is the canonical list of user identifiers, which must be used to sort the predictions in our submission file. The third file is the canonical list of songs, which contains all song IDs from the training data and the test data (both visible and hidden parts).

3. Computing Popularity

Our example algorithm uses the raw popularity of songs to form recommendations. Here we only use the testing part because it is smaller, but a better system would use the full training set. We use python, we recommend installing IPython.

In the following lines of code, we open the file, create a mapping from a song ID to the number of times this song appears, and close the file.

```
In [1]: f = open('kaggle_visible_evaluation_triplets.txt', 'r')
In [2]: song_to_count = dict()
In [3]: for line in f:
   ...:     _, song, _ = line.strip().split('\t')
   ...:     if song in song_to_count:
   ...:         song_to_count[song] += 1
   ...:     else:
   ...:         song_to_count[song] = 1
   ...:
In [4]: f.close()
```

For instance, the following song appears 13 times:

```
In [5]: song_to_count['SONZTNP12A8C1321DF']
Out[5]: 13
```

Next, we re-order the songs by decreasing popularity:

```
In [6]: songs_ordered = sorted(song_to_count.keys(),
                               key=lambda s: song_to_count[s],
                               reverse=True)
```

We will recommend the most popular songs to every user, but we must filter out songs already in the user's library. Reopening the triplets file, we will create a map from user to songs they have listened to.

```
In [7]: f = open('kaggle_visible_evaluation_triplets.txt', 'r')
In [8]: user_to_songs = dict()
In [9]: for line in f:
   ...:     user, song, _ = line.strip().split('\t')
   ...:     if user in user_to_songs:
   ...:         user_to_songs[user].add(song)
   ...:     else:
   ...:         user_to_songs[user] = set([song])
   ...:
In [15]: f.close()
```

For each user, we now have a list of songs:

```
user_to_songs['d7083f5e1d50c264277d624340edaaf3dc16095b']
Out[16]:
['SOUVUHC12A67020E3B',
 'SOUQERE12A58A75633',
 'SOIPJAX12A8C141A2D',
 'SOEFCDJ12AB0185FA0',
 'SOATCSU12A8C13393A',
 'SOZPZGN12A8C135B45',
 'SOPFVWP12A6D4FC636',
 'SOHEKND12A8AE481D0',
 'SOPSVVG12A8C13B444',
 'SODSKZZ12AB0188524',
 'SONZTNP12A8C1321DF',
 'SOVVLKF12A8C1424F0',
 'SOMLKZO12AB017F4AE',
 'SOACRJG12A8C137A8D',
 'SONJVYU12A8AE44F9E',
 'SOSOUKN12A8C13AB79']
```

Ok, we now have the songs ordered by popularity, and listening history for each user. To produce our submission file, we'll need to load the canonical ordering of users:

```
In [17]: f = open('kaggle_users.txt', 'r')
In [18]: canonical_users = map(lambda line: line.strip(),
                               f.readlines())
In [19]: f.close()
```

We now have the list of users in the order expected in the submission. The first two users are:

```
In [20]: canonical_users[:2]
Out[20]:
['fd50c4007b68a3737fe052d5a4f78ce8aa117f3d',
 'd7083f5e1d50c264277d624340edaaf3dc16095b']
```

We are almost there, but we're missing one more thing. To reduce the size of submission files, we do not submit a list of song IDs such as `SOSOUKN12A8C13AB79`, but rather their index in the canonical list of songs.

Let's create the map from song ID to song index (for those unfamiliar with python, this line is even more magic than before).

```
In [27]: f = open('kaggle_songs.txt', 'r')
In [28]: song_to_index = dict(map(lambda line:
                                  line.strip().split(' '),
                                  f.readlines()))
In [29]: f.close()
```

Now, for any given song ID, we have the corresponding integer index:

```
In [31]: song_to_index['SOSOUKN12A8C13AB79']
Out[31]: '283892'
```

Finally, we are ready to create the submission file. For each user in the canonical list, recommend the songs in order of popularity, except those already in the user's profile.

```
In [67]: f = open('submission.txt', 'w')
In [68]: for user in canonical_users:
   ....:     songs_to_recommend = []
   ....:     for song in songs_ordered:
   ....:         if len(songs_to_recommend) >= 500:
   ....:             break
   ....:         if not song in user_to_songs[user]:
   ....:             songs_to_recommend.append(song)
   ....:     # Transform song IDs to song indexes
   ....:     indices = map(lambda s: song_to_index[s],
                           songs_to_recommend)
   ....:     # Write line for that user
   ....:     f.write(' '.join(indices) + '\n')
   ....:
In [69]: f.close()
```

And that's it! We have our submission file: `submission.txt`

4. Verifying the Submission File

As a final sanity check, we provide a test script to verify that your submission file is properly formatted. You can find it on the MSD Challenge github: https://github.com/bmcfee/msd_challenge in the SubmissionUtils/ directory, file: validate_submission.py

Download the file, and launch it:

```
python validate_submission.py submission.txt
```

Everything checks out? Great! We can now upload the file to Kaggle!

5. Uploading to Kaggle

Once you're ready to upload your submission file, follow these simple steps:

1. (Optional) Compress the `submission.txt` file (.zip / .tar.gz / .7z).
2. Go to the "submission" section of the MSD Challenge Kaggle page.
3. Upload your (compressed) submission file.
4. Kick back and wait for your score!

If everything was successful, you should receive a score of around **0.02255**.

Congratulations! Now you're ready for the fun part!

Good luck!

-- The MSD Challenge Team