Music Recommender Documentation

Project description

General description

`TODO`

Using the recommender

The final webpage is made to be as intuitive to use as possible. In theory, no instructions should be necessary aside from the ones given on the web page itself. However, a more detailed user guide is available in the GitHub repository.

Technical description

The dataset

Our data mainly stems from the Million Song Dataset (hereafter: 1M), which is a freely available dataset covering roughly one million songs in contemporary music. The dataset records various information of the actual audio, like length, tempo and loudness, as well as metadata like release year, genre tags and artist information for each song. The basis of our project was to use these fields to construct a feature space in which the recommender could perform some kind of search. Since the dataset comprises over 300 GB of data, we only worked with a 1% subset to keep it manageable. In theory, the project can be scaled up to the full dataset on a powerful enough machine.

In total, 1M provides over 50 fields for each song, most of which are not useful for our purposes. So in a first step, we selected -- by reasonable intuition -- those fields that might be predictive or informative for a recommender. It was important to only choose numerical fields or fields that could be transformed reasonably to numerical values, as they needed to be able to be implemented into a continuous features space. This was trivial for properties like year or tempo, however not so for genre tags, which are lists of arbitrary length. For these, we calculated a proxy value by counting the number of tags that overlap with the tags for each input song. This means that overlapping genres is a mutable field that needs to be recalculated everytime a new input song is chosen.

Some fields were very informative in theory, but could not be chosen because of data quality. Among those are artist and song hotttness—which is more or less a measure of virality—and danceability. These scores are the result of some third party analysis and they were missing from so many songs that data imputation was infeasible.

In the end, we chose five informative fields for our feature space: overlapping genres, location (latitude and longitude), year and tempo. Some fields were left out to reduce computation cost like loudness, duration, time signature and artist terms—which is similar to genre tags. However, these fields are totally viable and could be reintroduced into a scaled up version of this project.

For our empirical analytics, we also used two complementary datasets, the Thisismyjam Likes Dataset and the Echo Nest Taste Profile Subset (hereafter: Echo set). They both contain user data, namely which songs a given user likes / listens to. Our idea was to use these datasets for validating our recommender by checking if the recommendation matches up with the user data.

The recommendation algorithm

The basis of our recommendation algorithm is the simple intuition that if a user likes a song with some set of features, the measure of how much they like other songs is a function of how close their feature set is to the input song's features. In other words: People like songs that are similar to other songs they like. This means that the goal of our recommender is to find songs that sit close to the input songs in the feature space.

The feature space is constructed using the fields collected from 1M as explained in `The Dataset`. However, since the recommender is supposed to recommend new songs to the user, we filter out all songs that from the arists of the input songs, as the user likely knows them already. This subset is then used to populate the feature space using an unsupervised Nearest Neighbors learner. Since Nearest Neighbors can only take one input song into account, we first generate possible recommendation candidates for each input song seperately. In this step, we purposely overgenerate, as the optimal recommendation for the whole set of input songs might in some cases be very suboptimal for a single input song.

Finally, every candidate is checked against every input song by calculating their squared distance in the feature space. We use squared distance here to promote songs that are equidistant from all input songs since the idea of the recommender is to take all input songs into account equally. The sum of all the squared distances then shows how good a candidate is, with a lower distance meaning better recommendation. In the end, the recommender outputs the 5 songs with the lowest distances.

Evaluation of the recommendations

Since music taste is a personal opinion that can't be measured directly as a quantitative value the evaluation was done using datasets containing information about what songs users had pressed "like" on on platforms separate from 1M. The primary plan was to use data from Thisismyjam, which used to be a social music website which allowed users to post their current favorite song, or "jam". The process can be described with the following steps:

1. Pick the users that have liked the song the user of the music recommender selected
2. Check if the users have also liked the songs that the recommender recommended
3. For each recommended song, display #users that liked the user selected and recommended song / #users that liked the user selected song

```
                users that liked the user selected and recommended song
evalutation = ----------------------------------------------------------
                         users that liked the user selected song
```

The song IDs from 1M had already been paired with those from Thisismyjam using fuzzy string matching with the Python search engine library Whoosh. Unfortunately, it was noticed that the amount of matching songs in our project's subset of songs was not large enough to perform a good evaluation, so new data had to be found. The second plan was to perform a similar process with data from the Echo taste profile dataset, which contained similar data for 1 million users. However, it was noticed that each user seemed to only have a few songs on their list (1.69 on average), which was insufficient for our needs. An empirical analysis where different combinations of attributes that were considered in the recommendation part was performed to see if a better subset of attributes that would yield a subset of users that would have a larger number of songs in their lists could be found, but did not produce any better results. A working version of the evaluation code was still completed and has been left in the program for documentation purposes.

Panel interface

`TODO`

Deployment

`TODO`

Reflection

What didn't work

The empirical analysis with the Echo Nest Taste Profile Subset did not yield any useful results. There could be several reasons for that: One the one hand, our data was incredibly sparse. Only using the 1% 1M subset meant that we also could only use ~1% of the Echo set. This was reduced even more, as a sizeable number of user data either only covered one artist or had very varied taste, both of which are not handled by our recommender. It would be interesting to see this validation on the whole 1M, where most of the sparsity-related problems would not occur.

On the other hand, the results may show that our recommendation algorithm is too naïve or unrefined, since even given this sparse dataset, one would expect at least marginally better results. This will be discussed further in the segment `Possible improvements`.

Possible improvements

There are many ways to improve the recommendation algorithm. While our current approach is apt for finding songs with similar characteristics, given input songs that are all similar themselves, its limitations quickly show for users with varied taste. If a user likes both metal and chorales, it should not be assumed that they also like whatever the average of those two styles is. However, by choosing one of each as their input songs as their input songs, the song with least distance in the feature space to will be exactly that and thus not pose a good recommendation. To avoid this, we could cluster the input songs based on some maximum distance and then give separate recommendations for each cluster. On the other hand, we could embrace this nonlinearity with which taste maps onto the feature space and wholesale replace the nearest-neighbor recommender with e.g. a neural network. It could be trained on the input songs' characteristics, with gold labels acquired through a dataset like Thisismyjam or Echonest likes. Given how sparsely populated these datasets are in the whole 1M Song feature space, this would be a sizeable task requiring a lot of compute.

Learning Outcomes

A big part of this project was learning how to compromise between our expections of the project scope and what was realizable, given the dataset and the time constraints. A thorough analysis of the data helps with estimating the possibilities—we saw this concretely with the sparsity of the Thisismyjam dataset, having to switch to the Echo Nest dataset for our empirical validation. Having to navigate the dataset and adjust the project and implementation accordingly was a continuous red thread guiding our project work. In the same vein, we needed to learn how to prioritize our work. In the end, a lot of time was spent on fixing the appearance of the deployed website and adjusting minutiae in the css, which would have been better spent on improving the recommender algorithm.