AC209a Data Science Project: Data Science with User Ratings and Reviews

Andrew, Sophie, Reiko, Nick

October 19, 2016

Introduction

Note to everyone: I am just spit-balling some things here for possible use later on. Please don't take this as set in stone but it might be a useful start. Feel free to add comments or change things.

Modern technologies unleash the possibilities for customising user experiences, products and thereby increase customer satisfaction by personalising content based on the past preferences of a specific user. Content based recommendation systems can further be extended to not only providing customers with content that they are most likely to enjoy and associate with but also for advertisment purposes. One could argue that it is only worthwhile advertising the right type of product and content to the right type of person. If this is not done the advertisement company not only runs the risk of alienating potential customers, but at a best case scenario simply wastes the time and money placed into delivering the specific advertisment content to the specific customer. It has been shown (*insert one of out references here*b) that potential customers are more likely to respond to content that directly appeals to them. Such is the power of a system that is able to make these structured connections from an unstructured source of data, such as user reviews and ratings.

In this project, we aim to achieve the following:

1. Select and prepare a suitable database of user ratings and reviews from ... TODO
2. Qualitatively, and quantitatively evaluate features and metrics upon which to train a statistical learning model.
3. Implement and evaluate the following prediction and recommendation engines: simple user rating prediction, content based sentiment and recommendation, clustering of product and collaborative filtering.

4. Decide upon the most appropriate recommendation based system for a particular setting and discuss some ethical considerations to be evaluated when making content based recommendations.

Project Description

Literature Review - Book Chapter 9

The application of recommendation systems is wide-ranging but most valuable to online retailers because of the ”long tail” effect. Because online vendors are not confined by physical space, they have the option of selling many more obscure items, articles, movies, etc. to their customers. However, customers may not be aware that these items exist or need help finding them because the product pool is so large. Recommendation systems fill this need.

Data is a recommendation system is usually presented in a utility matrix, with users by products. If we have numerical values representing the utility of an item to a user, for example, from user ratings, we would fill this in. However, often we can only insert a binary value here of 1 or 0, where a 1 refers to a user having bought or looked at the product or content and a 0 otherwise. Note that in general utility matrices are very sparse. In addition, the problem is often not to calculate a value for every blank in the utility matrix but rather just to generate some subset of items for which we think the utility value will be high. When combining binary and numerical values in this approach, we need to carefully choose scaling factors for the numerical values that will affect the resulting value. When computing user preference matrices from rating values, we should normalize the utilities by subtracting the average rating value for a user.

There are two basic structures for recommendation systems (any many variations in between): Content-Based Systems, wherein similarity of items is measured by a generated set of features comparing the items to other items, and Collaborative-Filtering, wherein recommendations are generated based on similarity of ratings by users who have rated both items.
In Content-Based Systems, delineating the features is often challenging. For text documents, we can compute TF.IDF scores for each word to try to determine which words most accurately characterize the document and then choose from a variety of distance measure between vectors of words. Again, sparseness in these vectors must be taken into account in computation measures. A system of feature generation that generalizes to almost any situation is user tagging. However, tagging relies on the willingness of users to go through the trouble of creating tags and requires that there are enough tags such that incorrect tags do not bias the system.

To compute recommended items based on profile vectors for both users, and items, we can compute the cosine distance between the user and item vectors, scaling non-boolean values as discussed above. Random-hyperplane and locally-sensitive-hashing are techniques we should research to place items into utility buckets.

Another method to consider for recommendations is to build a classifier that takes existing user ratings as the training set and predicts rating on other items. Decision trees are one option for this methodology, but we should look into other options as well. In the decision tree example, node values would be conditions involving one or more features of an item. Leaf values would determine whether or not an item would be liked. In general, construction of a tree like this tries to arrange that a leaf node either gets all positive examples or all negative examples. However, if the statistical significance of a minority group in a leaf node is small enough, we may leave it this way to avoid overfitting. Decision trees require a large amount of computation and a different tree for each user, so they are mostly only applicable to small data sets.

In Collaborative Filtering, we don't have to create a feature profile for users. Rather, users are defined exactly by their rating vectors. We find other users similar to these users by comparing the distance between rating vectors and then recommend to one user the items that the other user also liked.

To compute the distance between users in extremely sparse matrices, we could consider Jaccard distance, which focuses only on sets of items rated. This is a good choice in the situation where we are are inferring user preferences (i.e. have no ratings) based on whether or not a user purchased/read/watched a given item. With non-binary values, we have a few options: we could try cosine distance again, but this treats all unrated objects as unliked objects, which isn't exactly the correct interpretation. Alternatively, we can bucket numeric values into binary "like" and "dislike" based on their ranges. Again, we should also normalize numeric values.

Two complications in the lack of symmetry in the utility matrix:

1. The process of generating recommendations from similar users does not translate directly into generating recommendations from sets of similar items.
2. Users may like multiple genres and have only one of those genres in common with a similar user. We need to avoid recommending the other genre that the second user may not care for.

To predict an empty value, (I,U) in the matrix, we can find n users most similar to user U, normalize their ratings for item I, average the difference for the users who have rated I, and add it to the average rating for U. We could also find m items most similar to I and take the average rating that U has given to these items. To generate recommendations, we seek to at least generate most of the entries for a user U that are blank but have a high estimated value. In general, item-item similarity will be more accurate, because of the complication of users who like multiple genres, but this requires more computation.

To deal with the complications of the sparse matrix, we often have to cluster users and/or items into small clusters to get enough overlapping ratings.

Another way to compute blank values is by performing UV-decomposition on the utility matrix. We initialize U and V and then iterate to find choices of U and V that minimize the RMSE between values of UV and values of the actual utility matrix. Then, we can use entries of UV as a prediction for entries that are blank in the utility matrix. Intuitively, this method works off the assumption that there are only a small (comparatively) number of features that determine the final value in the utility matrix. Note that we again need to perform normalization for user rating scales and item quality before performing this algorithm. Additionally, it is possible to reach local minima in this algorithm, so we must either consider random restarts. To avoid overfitting, we can:

1. Avoid favoring the first components to be optimized by moving values only partially toward the optimized value, i.e. use a learning rate less than 1.

2. Stop the algorithm before the elements of U and V have fully converged

3. Take several UV decompositions and average the results of all of them.

Literature Review - Paper 2