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

Literature Review - The BellKor Solution to the Netflix Grand Prize

The grand prize solution to the Netflix challenge is important to consider, because despite its complexity, it serves as a kind of gold standard for recommendation systems. Also, the fact that the solution is a mixture of many models means that studying it will help teach us about a wide variety of models.

There are a few techniques the BellKor team used that were important across models, though.

One technique was computing baseline ratings for movies (which measures their overall quality, independently from a specific viewer) and users (which measures how critically they tend to rate movies generally, independently from any specific movies).

They computed these baseline movie ratings (which they denote b_i) and user ratings (which they denote b_u) by solving a least squares problem in terms of the average overall rating μ and each individual rating r_{ui}, where they try to find b_u and b_i such that the following quantity is minimized:

$$\sum_{\text{all ratings}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda \left(\sum_u b_u^2 + \sum_i b_i^2 \right)$$

where λ is a regularization parameter for the Ridge-regression-style penalty on large b_u and b_is, and we can think of the squared term as representing the true rating r_{ui} minus its user and movie-specific baseline $b_{ui} \equiv \mu + b_u + b_i$, whose difference should never be 0 (because users do have preferences not explainable by the movie's overall quality and their overall criticality), but which should intuitively be relatively small.

Once baselines are factored away, we can start predicting better and more consistent estimates of how much a user might actually like a movie.

The BellKor team actually took this idea of baselines a bit further and made them time-dependent. It's reasonable to expect that a movie's general popularity might change over time (e.g. as it becomes more or less culturally relevant) or that a user's baseline might shift from day to day (e.g. depending on their mood). So they actually compute baselines for every movie and every user as a function of time, although in practice they don't use continuous functions, but rather a set of discrete estimates for time bins spanning the total time range of the dataset.

The timescale over which these quantities vary is different; we expect movie baselines to shift slowly over months or years and user baselines to shift much more frequently. For movies, they bin their baselines in 10-week intervals. For users, because there isn't enough data to fully establish 1-day bins, they do a more complicated calculation that allows for gradual drift effects over long time periods as well as sudden "spikes" that are computed from day-specific data. This allows them to account for both long and short term changes in user baselines (which they note may

even be caused by multiple people using the same user account, which many of us know from experience is a fairly common practice).

One other important term they factor into their (movie and user-specific) baseline rating calculation is an adjustment to the movie baseline based on the number of other ratings users who rated that movie also gave on the same day. The intuitive justification they give for this adjustment is that certain movies tend to be rated differently when a user is rating them individually vs. alongside a lot of other movies in bulk. Those two processes, from a user-psychological perspective, are very distinct, yet they appear commingled in the same dataset. In particular, the paper argues that when users are bulk-rating movies, they tend to only rate movies that are particularly memorable over a long period of time, either for positive or negative reasons. In these cases, it may be that 50% of the population will like the movie unmemorably, but another 50% will hate it very memorably, so that only the negative ratings show up during bulk-rating sessions (which may be the most common type of rating session). This can bias recommendations in an unhelpful way, and so the extra term they include which controls for frequency helps remove that bias.

Note that although most of this time-specific information is most useful for filling in predictions about the past, rather than the future (which we want to predict), the calculated baselines still help the model make better future predictions, on average.

So, finally, all of these baseline components are computed again by a (more complex) gradient descent solution to a least-squares problem, with additional λ terms to regularize the extra parameters (stil using effectively a Ridge penalty).

After obtaining the baseline factors, they trained a variety of models to predict $r_{ui} - b_{ui}$, i.e. the difference between the true rating and the baseline. Models they used include time-modified versions of matrix factorization, nearest neighbor algorithms, restricted Boltzmann machines, and more. Their prediction for a given rating was a blend (weighted average) of the predictions of each of these models, and the weights of that average were also user and movie-specific. The way they compute the blend weights for each user and movie is complex (and will require reading several more papers to start grasping), but they used gradient boosted decision trees for some of the blend weight calculations.