Recommender Systems

Abstract

Wide variety of algorithms exist for generating recommendations using data sets of past purchases or ratings given by users. This paper aims to explore some of the key concepts and approaches for building effective recommender systems.

Introduction

The aim of the recommender or recommendation system is to predict the "rating" or "preference" a user would give to an item by application of various supervised and unsupervised machine learning techniques on transactions or ratings data.

Recommender systems have become increasingly popular in recent years, and are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general.

Recommender systems apply statistical and knowledge discovery techniques to the problem of making product recommendations based on previously recorded data. Such recommendations can help to

- Improve the conversion rate by helping the customer to find products she/he wants to buy faster
- Promote cross-selling by suggesting additional products
- Improve customer loyalty through creating a value-added relationship
- Aid in call preparation and effectiveness of sales representatives

Content Based versus Collaborative Filtering Algorithms

Content based approaches try to identify preference structure of a customer (user) concerning product (item) attributes. These attributes are then used for recommending products which rank high in user preference. For example, we define a product at multiple levels of hierarchy like Product Family >> Category >> Flavor Format >> Package Size >> Base Product. By looking at these attributes of preferred items by a user, we can recommend other products which have similar attributes.

Collaborative Filtering approaches leverage sales or rating data by many users to predict new opportunities and/or for creating top-N recommendation list for a given user. These algorithms are broadly divided into two categories-

User Based- User-based CF is a algorithm which tries to mimics word-of-mouth by analyzing rating data from many individuals. It is also called as Memory-based CF because it uses the whole user database to create recommendations. The assumption is that users with similar preferences will rate items similarly.

Thus missing ratings for a user can be predicted by first finding a neighborhood of similar users and then aggregate the ratings of these users to form a prediction.

The neighborhood is defined in terms of similarity between users, either by taking a given number of most similar users (k nearest neighbors) or all users within a given similarity threshold. Popular similarity measures for CF are the Pearson correlation coefficient and the Cosine similarity.

Item Based- Item-based CF is an approach which produces recommendations based on the relationship between items inferred from the rating matrix. The assumption behind this approach is that users will prefer items that are similar to other items they like. The model-building step consists of calculating a similarity matrix containing all item-to-item similarities using a given similarity measure. Popular are again Pearson correlation and Cosine similarity.

Item-based CF is more efficient than user-based CF since the model (reduced similarity matrix) is relatively small ($N \times k$) and can be fully precomputed. Item-based CF is known to only produce slightly inferior results compared to user-based CF and higher order models which take the joint distribution of sets of items into account are possible. Furthermore, item-based CF is successfully applied in large scale recommender systems (e.g., by Amazon.com).

Collaborative Filtering on Binary (0-1) Data

In situations wherein we don't have the feedback metric like rating or purchase quantity available, the only information available with us is whether a user purchased a product or not. This situation leads to binary data or more exactly 0-1 data where 1 means that we inferred that the user has a preference for an item and 0 means that either the user does not like the item or does not know about it.

We can deal with this situation with two strategies-
- All Zeros are negative examples
- All Zeros are unknown

If we assume that users typically favor only a small fraction of the items and thus most items with no rating will be indeed negative examples. Then we have no missing values and can use the approaches described above for real valued rating data.

However, if we assume all zeroes are missing values, then this lead to the problem that we cannot compute similarities using Pearson correlation or Cosine similarity since the not missing parts of the vectors only contains ones. A similarity measure which only focuses on matching ones and thus prevents the problem with zeroes is the Jaccard index :

$$sim_{Jaccard}(X, Y) = |X \cap Y| / |X \cup Y|$$

where X and Y are the sets of the items with a 1 in user profiles u_a and u_b, respectively. The Jaccard index can be used between users for user-based filtering and between items for item-based filtering as described above.

Association Rules for recommendations on binary data

Association rules are rules that help in uncovering purchase patterns of users leveraging data with their past transaction information. They indicate the likely occurrence of an item based on the occurrences of other items in the transaction.

A classical example of one such rule is- { Diaper} -> {Beer} . Evidently, young fathers would make a late-night run to the store to pick up Pampers and get some Beer while they were there. Placing the two disparate items together in a store might increase the likelihood of increased sales.

The left hand side (LHS) of this rule is called the antecedent – {Diaper} and the right hand side (RHS) is called the consequent {Beer}. Below criteria are used for identifying such rules-

- Support (s)- Fraction of transactions that contain both LHS and RHS
- Confidence(c)- Measures how often items in RHS appear in transactions that contain LHS
- Lift(l)- Ratio of actual confidence and expected confidence. A lift value greater than 1 indicates that LHS and RHS appear more often together than expected

Recommender systems using association rules produce recommendations based on a dependency model for items given by a set of association rules. The binary profile matrix R is seen as a database where each user is treated as a transaction that contains the subset of items in I with a rating of 1. To build the dependency model, a set of association rules R is mined from R. Association rules are rules of the form $X \rightarrow Y$. For the model we only use association rules with a single item in the right-hand-side of the rule. To select a set of useful association rules, thresholds on measures of significance and interestingness are used. The set of rules R that satisfy these constraints form the dependency model. Although finding all association rules given thresholds on support and confidence is a hard problem. Also, model size can be controlled by l, s and c.

To make a recommendation for an active user given the set of items Ta the user likes and the set of association rules R (dependency model), the following steps are necessary:
1. Find all matching rules $X \rightarrow Y$ for which $X \subseteq Ta$ in R.
2. Recommend N unique right-hand-sides (Y) of the matching rules with the highest confidence (or another measure of interestingness).

The dependency model is very similar to item-based CF with conditional probability-based similarity. It can be fully precomputed and rules with more than one items in the left-hand-side (X), it incorporates higher order effects between more than two items.

Evaluation of Recommender Algorithms

There are various approaches which can be taken for evaluating recommender algorithms. These approaches require splitting data into training and testing data sets-

- Splitting: We can randomly assign a predefined proportion of the users to the training set and all others to the test set.

- Bootstrap sampling: We can sample from test set with replacement to create the training set and then use the users not in the training set as the test set. This procedure has the advantage that for smaller data sets we can create larger training sets and still have users left for testing.

- k-fold cross-validation: Here we split U into k sets (called folds) of approximately the same size. Then we evaluate k times, always using one fold for testing and all other folds for learning. The k results can be averaged. This approach makes sure that each user is at least once in the test set and the averaging produces more robust results and error estimates.

For evaluating predicted ratings, we can compute the deviation of the prediction from the true value using Mean Average Error (MAE) or Root Mean Square Error (RMSE). RMSE penalizes larger errors stronger that MAE and thus is suitable for situations where small prediction errors are not very important.

For evaluating Top-N recommendations- A confusion matrix for all test users can created with the predicted top-N lists and the withheld items liked by the user. From the confusion matrix several performance measures can be derived. For the data mining task of a recommender system the performance of an algorithm depends on its ability to learn significant patterns in the data set. Performance measures used to evaluate these algorithms have their root in machine learning. A commonly used measure is accuracy, the fraction of correct recommendations to total possible recommendations.

Accuracy = correct recommendations/total possible recommendations

Tools and Libraries

There are multiple tools which can used for building a recommendation system based on the volume of data-

- Python-
  - GraphLab, pandas and sklearn are some of the key libraries
  - Surprise is a Python scikit library for building and analyzing recommender systems.
  - The name *SurPRISE* stands for Simple Python RecommendatIon System Engine

- R-
  - arules (shortform for Association Rules) is a R library with apriori algorithm for mining association rules based on lift, support and confidence
  - Recommenderlab- R extension package geared towards developing and testing recommender algorithms

- Spark- ML-Lib- ALS Factorization
  - Spark framework can also be used in R and Python using libraries like Sparklyr and PySpark
  - When using a Matrix Factorization approach to implement a recommendation algorithm we decompose the large user/item matrix into lower dimensional user factors and item factors. In the most simple approach we can then estimate the user rating (or in general preference) by multiplying those factors and minimizing the quadratic loss functions.

Footnotes and references

- http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=34AEEE06F0C2428083376C26C71D7CFF?doi=10.1.1.167.5120&rep=rep1&type=pdf
- https://spark.apache.org/docs/2.3.0/mllib-collaborative-filtering.html
- https://link.springer.com/article/10.1007/s10618-010-0204-8
- https://mahout.apache.org/users/recommender/intro-als-hadoop.html
- Recommenderlab, surprise and arules package documentation

Affiliation:

Aditya Mehta
Data Scientist, GBS Data Science Team
General Mills