Proceedings of the National Conference
On Undergraduate Research (NCUR) 2018
University of Central Oklahoma
Edmond, Oklahoma
April 5-7, 2018

Recommending Recipes: A Data Enabled Framework

[1]Omar Taylor, [1]Souparni Agnihotri, [1]Yazan Okasha, [2]Charlie Hubbard, [1]Chinmay Hegde
[1]Electrical and Computer Engineering Department
Iowa State University
Ames, IA 50011, USA

[2]Hy-Vee Innovation Center
Grimes, IA 50111, USA

Faculty Advisor: Dr. Chinmay Hegde

Abstract

Recommender systems apply prediction algorithms and data science techniques to predict user's interest in various products and services. To implement the recommender system, we developed a recipe recommendation software application called PotLuck which is built with node.js, Python, Java, and MongoDB. PotLuck provides users with the tools to make and rate recipes and a list of recommendations is made using content-based filtering, which is implemented using TF-IDF and cosine similarity. Once user preferences were acquired, we implemented collaborative filtering and researched matrix completion techniques of finding the average, median, soft impute and SVD impute to account for the sparsity of our data. Moreover, various clustering methods are performed on the non-sparse matrix and some similarity measurements are made to test the effectiveness of the matrix completion and clustering methods on the recommender system. The results showcase that a particular type of preprocessing and clustering technique works well with our PotLuck data.

Keywords: Recommender systems, data mining, content-based filtering, collaborative filtering, matrix completion.

1. Introduction

Ever since the rise in technology, data has been ever-increasing as more and more users put information on the internet. There is a need to process this information in a fast and efficient manner so that the relevant information can be processed and utilized for human needs. One of the most successful information processing technologies that have been developed is the Recommender System. It can be described as a personalized information filtering system that uses information provided by the user to predict the ratings or preference that user would give to an item.

Over the years, various approaches for building recommender systems have been explored. Two of the main approaches are content-based filtering (computing similarities between items) and collaborative filtering (computing similarities between users); both of which have seen successful implementations in information filtering and e-commerce applications. In this paper, we focus on the user-based, collaborative filtering approach towards building a recommender system as well as matrix completion methods to combat the problem of sparsity for that system.

2. Overview or Background

2.1 Content-Based Filtering

Content based filtering relies on the specific content of the item as well as a specific user's preference to make recommendations. In this kind of system, the keywords used to describe an item are matched to the kind of items described on a user's profile using some similarity metric.

There are several issues to be considered while building a content-based filtering system. First, terms can either be assigned automatically or manually. Terms can be typically the words that appear in the document. When terms are assigned automatically a method has to be chosen that can extract these terms from items. Second, the terms have to be represented such that both the user profile and the items can be compared in a meaningful way. Third, a learning algorithm has to be chosen that is able to learn the user profile based on seen items and can make recommendations based on this user profile.

Pandora Radio is a popular example of a content-based recommender system that plays music with similar characteristics. In the movie recommender systems arena, Rotten Tomatoes, Internet Movie Database, and Jaman are popular examples.

2.2 Collaborative Based Filtering

A traditional collaborative filtering algorithm represents a customer as an N-dimensional vector of items, where N is the number of distinct items. The algorithm generates recommendations based on a few customers who are most like the user. Collaborative based filtering methods can be computationally expensive. It is O(MN) in the worst case where M is the number of customers and N is the number of product items. As a result, scalability can be a big issue in these types of systems.

Big content providers like Netflix and online shopping giant, Amazon, all use collaborative filtering as a means of making recommendations.

3. Research Focus

The motivation behind our research is attempting to gain a solid understanding of recommender systems. We do this through a web application named Potluck. Potluck is a web and mobile software application that recommends recipes to users based on recommender system models. To recommend recipes to users, the web application is split into two main components: the front-end and the back-end. As we see in the diagram below, we can see a high-level overview of the application. The client facing application passes information to the back-end which contains some functions which are used to generate recommendations. Once recommendations are generated they are sent back to the front end.

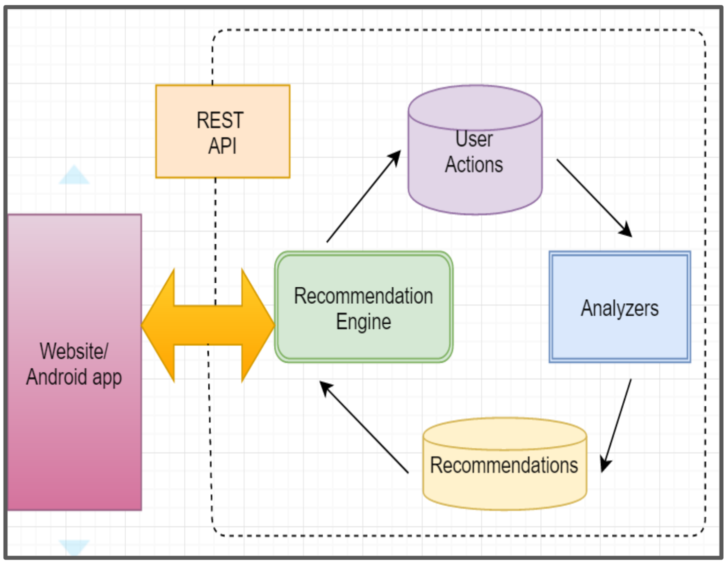

Figure 1: High level diagram of the system

The front-end acts as the main web app allowing users to log in, view recipes etc. The front-end has a significant contribution in collecting data. The data collected is from recipes displayed to users and users rating them. We can see that the data collection implementation is quite important as it will power our recommender system.

The backend will receive the collected data and feed it to a model which trains on the data. After the training phase, the backend can recommend recipes. But before that, the backend has already built a model to recommend recipes based on content-based filtering. The model consists of running TF-IDF on the recipe meta-data and finding the most similar recipes. Before we go into detail on the algorithm we will introduce TF-IDF and cosine similarity.

3.1 TF-IDF

TF-IDF[1] or Term Frequency-Inverse Document Frequency is an algorithm for converting documents of words into vectors. TF-IDF is popular since it will attempt to extract important words from the strings. It does so by giving the important words high weights and the common words a low weight.

To find and assign weights, we must find the importance of the word. This is done through Term Frequency, which is simply counting the number of times the word is repeated in the document. The importance is a little flawed right now since common words, such as “and” or “the”, are very common and will have a high count. Inverse Document Frequency will now account for this, which counts the number of documents that contain the word and a little more described below. For more details refer to [1].

3.2 Cosine Similarity

The cosine similarity is a popular and common similarity measure of 2 vectors of d dimensions, where d is an arbitrary integer. The rule derives from the dot product of two vectors, A and B where we get the angle between the two vectors as:

$$\frac{A \cdot B}{||A|| \cdot ||B||}$$

This is a powerful tool that allows us to find the similarity between two vectors. It can be used in many applications and in our case it is used in our content and collaborative based filtering algorithms.

3.3 Content-Based Filtering Algorithm

Our content-based filtering algorithm uses the TF-IDF vectors and the cosine similarity concept mentioned above. Our data was in the form of a JSON string containing all of the metadata acquired from the Spoonacular API. We used the ingredients list of all of the recipes to compute the similarities. First, the ingredients of each recipe were converted into its vector form using TF-IDF. This resulted in having a massive array of TF-IDF vectors for the ingredients corresponding to each recipe. We then performed cosine similarity on each recipe corresponding to every other recipe. We then outputted the top-k similar recipes for each recipe in our database, where k was an arbitrarily chosen number. This helped us come up with the initial set of recommendations which we could show the user on PotLuck.

4. Research Aim

No two data sets are the same. The same algorithm can perform differently on different data. Having been exposed to several clustering methodologies in class, our main research goal was to explore how well these methods worked with the data that we had gathered from PotLuck. We were particularly interested in how various matrix completion methods affect the data and the ways it is being clustered. Our final goal was to come up with a quantitative analysis of the accuracy of the different clustering methods when performed with various matrix completion methods, in order to give a detailed analysis of which method to adapt for our data.

5. Research Methodology

As mentioned above, we want to perform collaborative based filtering on the data we have collected. It is important to note that our data is unlabelled, it only contains user’s ratings for a given recipe. So it will be difficult to perform a supervised learning technique, even though supervised learning techniques are popular and have large amounts of research behind them. Once we have the data, we need to fill in missing values in our data. Once that is complete we intend to perform three common unsupervised learning techniques on our data and test the performance. The algorithms are K-means, Spectral and Hierarchical Clustering respectively. First, we will introduce the algorithms in more detail before moving forward.

5.1 Clustering

5.1.1 k-means clustering

K-means clustering is a popular clustering technique proposed by Hartigan (1975)[2]. The popular clustering technique is based on clustering n data points into k clusters. A proper formulation is given below:

Given our data points $x_1, x_2, \ldots, x_n$, we assume that they are initially clustered into disjoint sets $S_1 \ldots S_k$.

We can find the mean in the following manner, $\mu_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$

Assuming that we have the correct $\mu_1 \cdots \mu_k$, our optimization problem reduces to –

min $\sum_{x_i \in X} ||x_i - \mu_j||^2$which can be thought of as a Nearest Neighbor Problem.

For each x_i. the optimal cluster for it j^* is: $j^* = \underset{j}{\operatorname{argmin}} ||x_i - \mu_j||$

One simple algorithm which makes use of this is Lloyd’s algorithm [3]. Which is defined below:

```
1. $initialize\ clusters\ \mu_1 \ldots \mu_k$
2.$for\ each\ x_i\ compute\ it's\ j^* = \underset{j}{\operatorname{argmin}} ||x_i - x_j||$
3.$recompute\ u_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$
4.$repeat\ until\ no\ change$
```

5.1.2 spectral clustering

Spectral clustering treats clustering as a graph partitioning problem without making assumptions about the form of the clusters. Spectral clustering clusters together points using the eigenvectors of the matrices derived from the data. Unlike K-means, spectral clustering does not require the data points to be in convex boundaries. This is one of its main advantages, in addition to being easy to implement and being reasonably fast on sparse datasets. However, spectral clustering may be sensitive to the choice of parameters and may be computationally intensive on large datasets.
More information on Spectral Clustering and the algorithm can be found in the paper referenced. [4]

5.1.3 hierarchical clustering

Hierarchical clustering is another popular clustering technique which assumes that there is a tree structure to collected data. Building on the assumption, Hierarchical clustering splits the data set into singleton sets and groups sets iteratively based on a user-defined constraint. In our case, we want to minimize the distance for each grouping. We can define the distance between two sets in many ways, but in our research, we defined it to be the ward distance [5]. Ward’s distance will attempt to minimize the total within-set variance by finding a way to calculate the distance between two sets.

After talking about clustering algorithms, we need to fill in missing data since our data has large quantities of missing values.

5.2 Matrix Completion

5.2.1 SVD impute and soft impute

The SVD Impute [7] algorithm is a matrix completion algorithm that uses the properties of SVD to estimate the values of the matrix. It does so by initializing missing entries by a weak estimate then iteratively improving upon the estimate. For more details on the algorithm please refer to the [7].

The Soft Impute[6] algorithm is a common and effective algorithm which attempts to fill in missing of a matrix. However, it assumes that the true matrix rank must be low. This is a fair assumption since human taste can be classified into a small number of classes. The Soft Impute algorithm attempts to optimize the error between the true matrix and original matrix.

$$\sum_{(i,j)\in\Omega}(X_{ij} - Z_{ij})^2 \leq \delta$$

Where X is the observed matrix, Z is the true matrix and δ is the error constraint. To optimize this problem, Soft Impute makes use of SVD Impute and iteratively arrives at a solution. For more details please refer to [6].

5.3 Testing

The effectiveness of the various matrix completion and clustering algorithms were tested by simply computing cosine similarity on the clusters obtained. The process we followed was to trace back the recipes in each cluster. From each cluster, we traced back and found the set of all recipes that the users in that cluster liked and a set of all recipes that they disliked. From that, we obtained the ingredients of each recipe and performed TF-IDF on them to transform them into vectors. Cosine similarity was then computed on these vectors to find a quantitative measure of the similarity of these recipes.

For example, let's say we had 7 users and they were clustered into 3 clusters.

Clustering output: $(1, 1, 3, 3, 2, 2, 1)$

This shows that user 1 was clustered into cluster 1, user 2 into cluster 1, user 3 into cluster 3… and so on.
Then we trace back to find all of the users in each cluster, and a list of the recipe ingredients that each user liked or disliked.
We pass the list of the ingredients that the users liked into the cosine similarity measures to get a quantitative measure of how closely related the ingredients are. The same thing is done for the list of dislikes. This helps us analyze how well our clustering methods worked for each matrix completion algorithm.
Our testing results are outputted as a list of tuples where each tuple represents the cosine similarity measure of the recipes liked and disliked in each cluster.

Cosine similarity output: $[(0.25, 0.78), (0.34, 0.45), (0.75, 0.65) \ldots, (0.65, 0.89)]$

6. Data Collection And Results

In the data collection phase, we used python and its packages (sklearn, Scikit) for our implementation. The data that was in MongoDB was extracted using mongoexport. The specific information we extracted was the user ratings and the unique user IDs. Then the data was formatted into a matrix which was m X n dimensional, where m represents the users and n represents the recipes. Each entry in the matrix contained either a 0 or 1 or '?'. The '?' represented the empty data - the recipes that the users had not rated. The next task was to fill up this data which we did using the following four algorithms:

Mean - The first algorithm was averaging over the columns, which would essentially iterate over all the missing entries (i,j) and replace the '?' with the average of the jth column.

Median - This involves simply taking the median over the columns. Each missing entry (i,j) is replaced with the median of the jth column.

SoftImpute - Algorithm for softImpute is provided in part 5.2.1. The same algorithm was used to fill in the missing data

SVDImpute - Using the algorithm described in 5.2.1 we extracted the estimated value of our matrix X. Then once we have $U \; \Sigma \; V$ we can rebuild the matrix X using these matrices such that now we do not have any missing values in our matrix.

Using all of the algorithms described above, our data matrix X was made non-sparse and we began clustering. The following parameters and constraints were used for certain clustering measures:

For K-means, the value of 'K' was 6.

Wards distance was used as our distance measure for hierarchical clustering and we stopped at 8 clusters.

The graphs below represents the data collected:

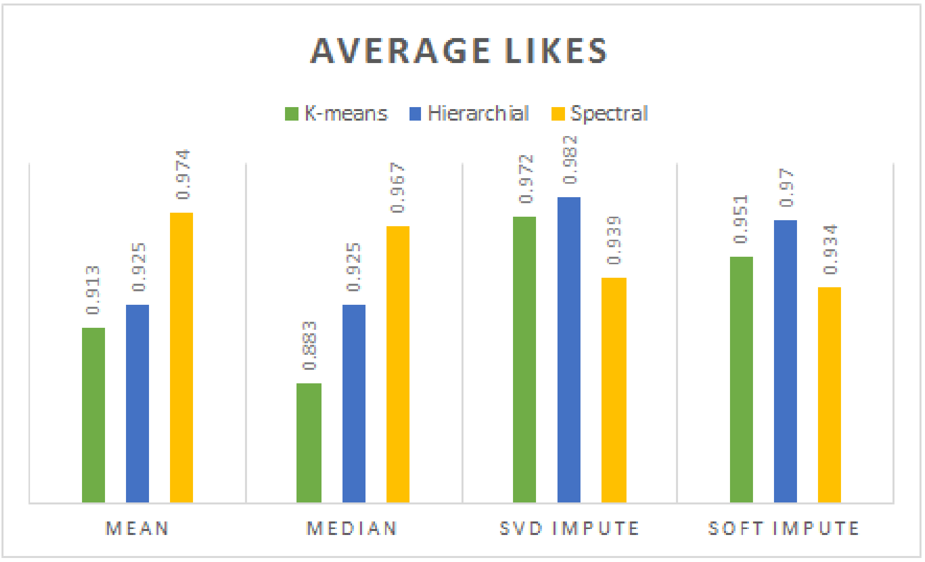

Figure 2: A comparison of the performance of the different matrix completion measures - mean, median, SVD Impute and Soft Impute on the different clustering methods - K-means, Hierarchical and Spectral clustering specifically for all of the items that the users “Liked”.

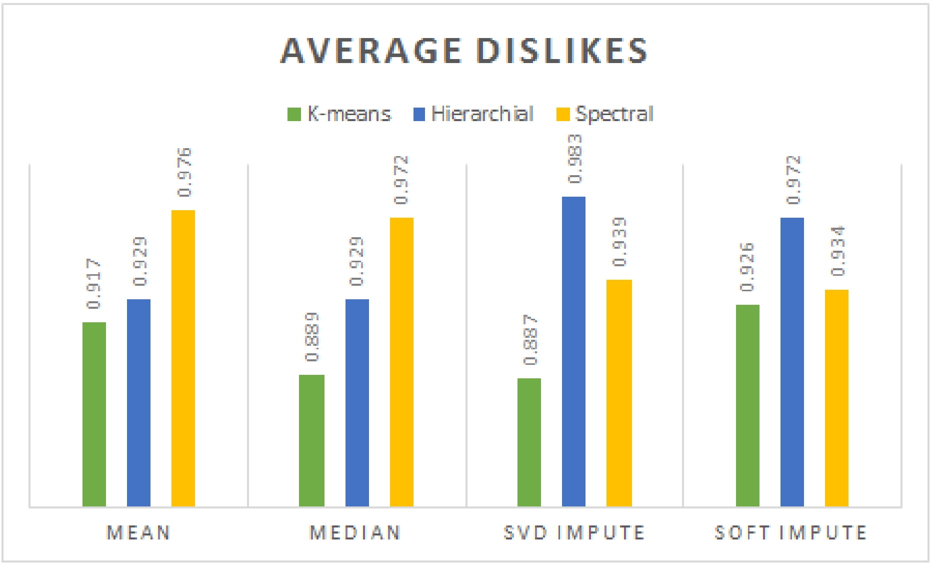

Figure 3: A comparison of the performance of the different matrix completion measures - mean, median, SVD Impute and Soft Impute on the different clustering methods - K-means, Hierarchical and Spectral clustering specifically for all of the items that the users “DIsliked”.

The numbers on the bars represents the average cosine similarity score obtained across all the clusters by each of the clustering algorithms.

7. Data Analysis

From our results, we can see that our average similarities are quite high. This can be due to several factors. The matrix completion methods used indicate that SVD and soft impute work best with K-means clustering, giving an average “like” similarity of greater than 95%. However, there is a discrepancy in the similarity results obtained for the “dislikes”, which is almost 10% lower than the “likes” similarity scores. The reason for this discrepancy is because we don’t have many data points and we have a large number of missing entries. Intuitively, we should have a much lower accuracy than our current data, but due to sparsity, we are getting very high similarity scores.

It is also interesting to note that the mean and median methods adopted for the matrix completion gave more consistent results overall than SVD and soft impute. This is because each column in the matrix was filled with the same value of the mean or median of that column whereas each value would differ in the SVD and soft impute methods. This also leads to the conclusion that both SVD and soft impute resulted in better estimations than did the mean or median methods.

We must remember that we are comparing recipes against each other to get our similarity measure. We must convert our ingredients into TF-IDF vectors and when we do, we get a high dimension vector with many 0s and a small amount of 1s. So, when comparing two vectors, most of the values compared will be the same and only a few values will differ resulting in a high similarity value.

In addition, our results would be more accurate given more users and ratings. The larger the user-rating matrix, the better the accuracy of our computations.

8. Conclusion

This report presented some of the algorithms used to build collaborative filtering based recommender systems. It is important to note that each algorithm and methodology used has its advantages and disadvantages. User-based algorithms, in general, are accurate but not scalable. We mainly focused on finding ways to improve the performance and accuracy of our algorithms by exploring different matrix completion methods that we can use on our data and how well each completed matrix worked on different clustering algorithms.

In conclusion, we believe that adopting one of the matrix completion and clustering methods into our PotLuck recommendation system, we will be able to give better recommendations to our users.

9. Future work

In the future, we plan on improving our recommender system by combining both our content-based filtering algorithm and our collaborative based filtering algorithm together. To do this, we must first decide which clustering algorithm we want and which matrix completion algorithm to use.

We also plan on experimenting more with matrix completion algorithms. Either by adapting the soft Impute algorithm or by combining two algorithms together.

Lastly, in order to obtain more accurate results, we need more user data.

10. Acknowledgements

This work was supported in part by grants from the National Science Foundation and NVIDIA, and start-up grants from the College of Engineering and the Office of the VPR at Iowa State University.

11. References

1. John Ramos. “Using TF-IDF to Determine Word Relevance in Document Queries”. Rutgers University
2. Hartigan, John A. “Clustering algorithms”. New York, NY: Wiley, 1975.
3. Lloyd, Stuart P. “Least squares quantization in PCM”. 1982. IEEE Transactions on Information Theory.
4. Ng, Andrew. Jordan, Michael. Weiss, Yair. “On Spectral Clustering: Analysis and an algorithm”.
5. Joe H Ward.”Hierarchical grouping to optimize an objective function”. Journal of the American Statistical Association.
6. Rahul Mazumder, Trevor Hastie, Robert Tibshirani. “Spectral Regularization Algorithms for Learning Large Incomplete Matrices”.
7. Troyanskaya O et al. “Missing value estimation methods for DNA microarrays”.
8. Content-based filtering, “recommender-systems.org/content-based-filtering/’. Date of access, 16 April 2018
9. Spectral clustering, “cvl.isy.liu.se:82/education/graduate/spectral-clustering/SC_course_part1.pdf**”.** Date of access, 16 April 2018