A Scalable Approach for Cross-Domain Recommendation

Prosenjit Gupta[1], Akarsh Srivastava[1], Aman Jain[1], Ashwin Jayadev[1], Rajdeep Mukherjee[1] and Shronit Bhargava[1]

[1] Department of Computer Science and Engineering,
NIIT University, Neemrana, Rajasthan, India
prosenjit.gupta@st.niituniversity.in
akarsh372@gmail.com
amanjainddn@gmail.com
jaydevashwin7@gmail.com
rajdeep.mukherjee295@gmail.com
bhargavashronit@gmail.com

Abstract.

Recommender Systems possesses the ability to predict in advance the future behavior of a particular individual based on its past preferences. Today any individual has more than one profile that he maintains on various social media platforms and other website, and leveraging all this data on the preference of an individual from various domains (cross-domain) can help us in making better user models that can be used to make better and improved recommendation. A Cross-domain recommenders system thus aims to improve the recommendation of a target domain extracting and using the metadata from many source domains. This paper starts with a brief introduction to the concept of recommender systems including collaborative filtering, types of domains and the various challenges associated with it. In this paper we aim to tackle the problem of scalability, the biggest challenge to the implementation of cross domain recommendation. We have performed various experiments to divide the datasets into smaller clusters and then to implement our own algorithm using the attributes in the dataset to return the best suited recommendation for a user.

Keywords: Cross-Domain, Recommender Systems, Collaborative Filtering, Scalability.

1 Introduction

Recommender systems is an essential part of every single online enterprise. One buzz word we come across often now-a-days is 'Big Data'. Recommender systems in general require a lot of data. Data that is collected from users of systems help recommender systems to recommend items to users. The more the data, the better the recommendation. One obvious problem that we face in this case is the problem of scalability. As the paper 'Iterative Use of Weighted Voronoi Diagrams to Improve Scalability in Recommender Systems' state 'In CF, finding similarity amongst N users is an $O(N^2)$ process.

If N is large then similarity computation becomes quite expensive. Decomposition avoids this quadratic blowup and allows us to process bigger datasets even with limited computational resources. As for example, if we partition a region with n users into k partitions with nearly equal sizes, then the overall time required for performing collaborative filtering in all those k partitions will be proportional to $k.(n/k)^2 = (n^2/k^2).k = n^2/k$. So we can achieve a k order speed up by dividing the users' space into k partitions' [1].

Recommendation algorithms are mainly of three types. The one we use here is **Collaborative Filtering**. In collaborative filtering, we can use a single domain for recommendation or multiple domains. Here domain refers to the type of an item. For instance in our case we have used 2 domains for collaborative filtering which are 2 sources of movies. One source being **MovieLens** and another being **Kaggle.** Involving **multiple domains** in recommendation gives us a better chance at recommending items effectively. For instance if we use **movies** and **books** as the 2 domains then we can recommend movies that have the same genre as the books that the user has rated well and vice-versa. That's why **cross-domain collaborative filtering** is powerful in improving recommendations.

2 Our Contribution

2.1 Scalability of the system

For the purpose of achieving a scalable recommender system we had to partition the data to make the search of an item computationally efficient [1]. We used clustering to initially partition the data into clusters and recommending items from those clusters. Clustering hence helped us greatly reduce our search space. The clustering algorithm used was **K-Means clustering algorithm.** The entire recommendation was therefore broken up to 2 phases. **Preprocessing,** where we cluster the data and **Querying** where we actually similar movies from the appropriate cluster. Hence we are not searching the movie in the entire search space but only a subset of it.

2.2 Numeric and Non-Numeric Data Clustering

One issue we tackled was the clustering of non-numeric data. The clustering of numeric data was easy as we had to simply create vectors of the numeric fields in our datasets, however in case of non-numeric fields we had to find a way to represent non-numeric fields as vectors. This was achieved using **tf-idf vectorization** of non-numeric data. Due to the inclusion of both numeric and non-numeric data the recommendation is enhanced. Our final approach of combining both numeric and non-numeric data was to create clusters of non-numeric data first and then to create sub-clusters of numeric data inside the non-numeric clusters.

3 Recommendation Algorithms

Recommendation can be done in various ways. Recommendation algorithms in general are subdivided into 3 categories based on the aforementioned criteria:

1. Content-Based Filtering
2. Collaborative Filtering
3. Hybrid Recommender Systems

3.1 Content Based Filtering

Content-Based filtering is the type of recommendation where the properties of the items themselves are used for the purpose of recommendation and nothing more. For instance if we consider a movie then the attributes or properties of the movie would include 'movie-title', 'director name', 'lead actor', 'production', 'genre' and so on. All these attributes play a significant role in recommending movies to users. For instance a person who has watched 'horror' movies and likes the 'horror' genre in general would also like another movie from the same genre.

3.2 Collaborative Filtering

Collaborative Filtering is the type of recommendation where the **users** of the items come into play and the preferences of individual users play a vital role in the recommendation. It is called collaborative because we are taking into account the preferences (for instance for movies, the previously watched and rated movies) of the same and *like-minded* users to recommend items to a particular user. For instance if **user A** likes **movie X** and the same movie is also liked by **user B** then chances are that **user A** and **user B** are alike in some respects and the movies watched and/or rated by **user B** will also be liked by **user A.** It is a based on a simple idea that users who have expressed same interests in the past will express similar interests in the future. Collaborative Filtering can be used across multiple domains.

3.3 Hybrid Recommender Systems

Hybrid recommendation as the name suggests consists of using the pros of both content based filtering and collaborative filtering. For instance in the example of movies, if we have the data corresponding to the attributes of the movies along with the likes and dislikes of users, we can not only recommend movies rated well by like-minded users but also recommend movies similar to that movie.

4 Experiments

4.1 Datasets used

For the purposes of experimenting with our methodology we used datasets from 2 different sources. The item chosen for experimentation was **movies**.

Table 1. Datasets used and their specifications

Dataset	Source	Size	Unique Entries	Columns	Description
Movies	MovieLens (domain 1)	15.4 MB	10197	33	Movies vs Movie attributes
Movies	Kaggle (domain 2)	1.44 MB	5043	28	Movies vs Movie attributes
Users	MovieLens	26.7 MB	2114	9	Users vs user ratings for movies in MovieLens

4.2 Methodology

Preprocessing. *Why:* Recall that our objective was to tackle scalability through partitioning the datasets effectively. To address this issue of scalability of large recommender systems due to their enormous data we need to partition the data so that at a time we can work with smaller portions of the data.

How: Various techniques were used and analyzed for various ways of partitioning the data. These techniques are illustrated in the various experiments that follow.

Querying. After preprocessing, querying would imply that we query the partitions of the data for similar items which in our case are movies. Since now we are searching for similar movies within a small portion with respect to the entire dataset, our query will be both faster and more effective.

4.3 Experiment 1: Genre and Rating Based Partitioning

A rather naive approach to solve the problem of scalability we used the **genre** of movies in order to partition the entire data and within each genre **ratings of movies** were used to create another level of partitions. If a genre signified one partition, a movie can be in more than one partition at the same time since one movie had many genres.

Partitions chosen for ratings for the dataset **MovieLens**

[0 – 1.5): Partition A
[1.5 – 3): Partition B
[3 – 5]: Partition C

Partitions chosen for ratings for the dataset **Kaggle**:
[0 – 2.5): Partition A
[2.5 – 5): Partition B
[5 – 7.5): Partition C
[7.5 – 10]: Partition D

Results and Drawbacks. Since genre was the only criteria used for the partitioning of data into partitions, the results were consistent only with respect to the genre and their corresponding rating. One important drawback of this approach is the lack of usage of data from the datasets.

4.4 Experiment 2: Numeric Data Clustering

In an attempt to use maximum possible number of relevant fields to partition the data, we had to come up with a proper partitioning scheme that would use a number of fields together. We therefore turned our attention to information retrieval techniques.

Classification techniques in Information Retrieval

1. Supervised
2. Unsupervised

Supervised. In case of supervised classification technique, the classes or labels as they are more formally called, appear within the data itself and hence there is no need of external classification of the data into classes. Supervised classification techniques are widely used in Naïve Bayes classification and Machine Learning.

Unsupervised. In case of unsupervised classification technique, the classes are not already present in the data itself and needs to found out using an appropriate technique like clustering. The strict assumption is, 'Items in the same cluster are as similar as possible and items in different clusters are as dissimilar as possible'.

Clustering is done in such a way that all attributes of an item in a dataset essentially become, dimensions in an n-dimensional hyperspace where n is the number of attributes, thus each item becomes a point in that n-dimensional hyperspace. After that any number of different clustering techniques can be applied to classify the points (which actually represent items) into clusters.

Method. Following the clustering technique, we first considered it appropriate to consider all the numeric data present in the datasets as it would be easy to consider them as dimensions in the n-dimensional hyperspace. These points can be thought of as n-dimensional vectors. In our case after conversion of the items to vectors we normalize the vectors and then we used **K Means clustering algorithm** to cluster those points (representative of the items present in the dataset).

K-Means Clustering Algorithm. K-Means is one of the many clustering algorithms available to cluster the points in an n-dimensional hyperspace. K-means clustering aims to partition x observations (where x is the total number of observations or in our case the total number of movies present in one dataset) into k clusters. K-Means clustering is applied in domains like Information Retrieval, Recommender systems and the like.

Numeric Clustering in MovieLens Dataset. A total of 14 [1]attributes were chosen for numeric clustering in MovieLens Dataset.

All these attributes are considered for the purpose of vectorizing each item. This means that each attribute essentially become a dimension in the n-dimensional hyperspace where n is the total number of attributes which in our case is 14.

The scale of all the dimensions may not be the same and hence it is necessary to apply a normalization of the vectors. In our case we applied L2-normalization in order to normalize the vectors.

After all points are found and normalized the k-means algorithm is applied in this 14-dimensional hyperspace in order to cluster the points and hence the movies into clusters.

Numeric Clustering in Kaggle Dataset. A total of 7 [1]attributes were chosen for numeric clustering in Kaggle Dataset.

The clustering was applied in the same way in the Kaggle Dataset as well separately.

Results and Drawbacks of this Approach. After the clustering of the 2 datasets into 2 separate set of clusters, clusters were stored in a csv file. Although no querying was done on this dataset it was clear that the optimum result could not be achieved by just using the numeric data present in the datasets. We initially had taken the numeric data because it could be easily converted to vectors without any prior processing. We were not considering some of the more potent fields like actors in the movie, director, genre and so on.

4.5 Experiment 3: Non-Numeric Data Representation and Nearest Neighbor Approach

The technique we used for including non-numeric data was **tf-idf vectorization.** tf-idf vectorization stands for term-frequency document-frequency vectorization. These are information retrieval techniques used in order to retrieve relevant data from documents (unstructured data).

A document consists of terms which are unique words present in the document. The term frequency weight is given by:

[1] For checking all the attributes that were used for numeric data clustering check out our github repository: github.com/CrossDomainCollaborativeFiltering/Clustering/tree/master/src/NumericClustering

$$w_{t,d} = 1 + log_{10}\, tf_{t,d} \;\; if\; tf_{t,d} > 0 \tag{1}$$

and,

$$w_{t,d} = 0\; otherwise \tag{2}$$

$Tf_{t,d}$ refers to to the term frequency for a term t for a document d.

Document Frequency. Document frequency refers to the number of documents in the entire collection that has a particular term in it.

Inverse Document Frequency. The inverse document frequency is the inverse of the document frequency. It is given by:

$$idf_t = log_{10} \frac{N}{dft} \tag{3}$$

Here N is the total number of documents in the collection. df_t represents the document frequency for a term in the collection. In general inverse document frequency is used in order to reduce the weightage of frequent terms in the collection.

Conversion of Non-Numeric Data to vectors. Non-numeric fields considered in the MovieLens Dataset: **ActorName, Genre, DirectorName.**

The other non-numeric fields were not chosen as such they had little relevance with respect to similarity of movies and what a typical viewer with a specific would like to see.

Non-numeric fields considered in the Kaggle Dataset: **DirectorName, Actor2Name, Genre, Actor1Name, Actor3Name.**

Conversion of data in fields to vectors:

Conversion of field values to documents. For each item (movie), the values present in the fields were converted to paragraphs such that each value in the field becomes one term in the paragraph and the paragraph essentially became a document. For instance considering the MovieLens Dataset.

Table 2. Sample data from dataset

movieName	ActorName	Genre	DirectorName
Titanic	Leonardo di Caprio, Kate Winslet	Drama, Romance	James Cameron

Document Name: Titanic

Document contents: **LeonardodiCaprio**<space>**KateWinslet**<space>**Drama**<space>**Romance**<space>**JamesCameron**

Hence the document consists of 5 terms. This was done with each and every movie and hence each movie was considered to be one document.

Generation of a term document matrix. The term document matrix can then be generated in the following manner. For instance considering the MovieLens dataset:

Table 3. Sample data from dataset

MovieName	ActorName	Genre	DirectorName
Titanic	LeonardodiCaprio, Kate Winslet	Drama, Romance	James Cameron
Tomorrow Never Dies	Colin Salmon, Geoffrey Palmer	Action, Romance, Thriller	Roger Spottiswoode
Twisted	Richard T Jones, Andy Garcia	Comedy, Drama	Philip Kaufman

Mind you that the total number of terms (even for a single movie) here is 14. Needless to say that we are getting an extremely sparse matrix here.

Table 4. Term document matrix

MovieName	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
Titanic	1	1	0	0	0	0	1	1	0	0	0	1	0	0
TomoroowNeverDies	0	0	1	1	0	0	0	1	1	1	0	0	1	0
Twisted	0	0	0	0	1	1	1	0	0	0	1	0	0	1

Once we get these term frequencies for each movie (here document) these can be considered as coordinates for an n-dimensional hyperspace where n is the number of terms present in the entire collection and the vectors are documents themselves.

Normalization of the vectors. The vectors are next normalized following the L-2 Normalization technique or Euclidean normalization technique.

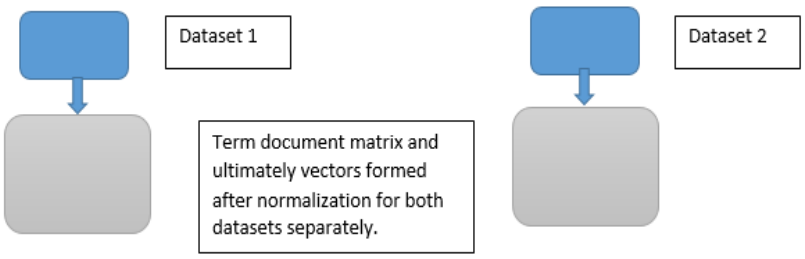

Fig. 1. Vectorization of movies

Querying of movies using nearest neighbor approach. Since all vectors are unit length the similarity of 2 items (movies) can simply be computed by figuring out the dot product between the 2 items. The similar one movie is with respect to another movie, the lower the angle between them, the higher the cosine of the angle and hence higher the dot product.

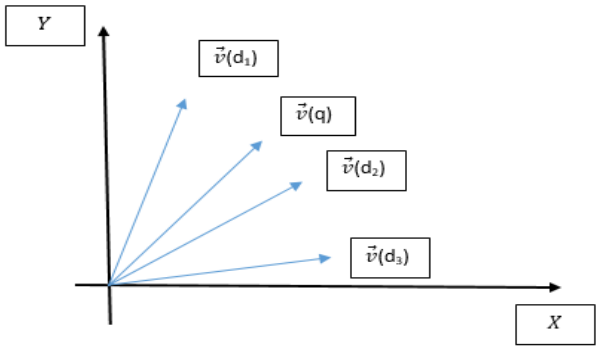

Fig. 2. Nearest neighbor approach

Note in the above diagram, if the query document vector is $\vec{v}$ (q), then $\vec{v}$ (d_2) is closer to the query document than $\vec{v}$ (d_1) and hence would have a higher dot product.

Similarly now we can use a query movie and find out movie the query is most similar to by finding out the dot product of that movie with every other movie.

This querying of movies was done in both datasets separately and the final result was simply the combination of the 2. If dataset 1 returned 3 movies : movie1, movie2 and movie3 and dataset 2 returned 3 movies movie4, movie5 and movie6 then the final result was simply shown as movie1, movie2, movie3, movie4, movie5 and movie6.

Results and Drawback of this approach. The results obtained were not as convincing as the expected. For instance when the query movie Spiderman was given the top 5 most similar movies were expected to be sequels of Spiderman. However that was not the case.

Although it seems to be an excellent approach time is a major factor in reducing the importance and relevance of this approach. Partitioning the data was required especially in order to tackle scalability. However we were unable to tackle scalability as we were not partitioning the data into appropriate partitions. We were simply searching in the entire collection for better results. This was causing an O(n) algorithm at a high level even during the querying phase. This would quickly become an unfeasible solution when the dataset would be large.

4.6 Experiment 4: Non Numeric Data Clustering and Subsequent Querying

The next obvious approach was to amalgamate the ideas of both non-numeric data representation and clustering for tackling the scalability issue. Hence we considered K-Means clustering to cluster the data.

Conversion of data to vectors. The conversion of data to vectors was done in exactly the same way as the previous experiment, hence we are not reiterating the explanation here.

Generation of term document matrix. The term-document matrix was generated in exactly the same way as the previous experiment.

Normalization of vectors. Once again we used l2 normalization for the normalization of the vectors in exactly the same way as the previous experiment.

Application of K-Means Clustering. The difference between the previous approach and this one comes here. Here we apply K Means clustering to the formed vectors in order to cluster the data. Recall that specifics of K Means clustering have already been discussed in a previous experiment. Once the clusters are formed it is now easy to search the similar movies among the specific cluster movies. This solution would therefore tackle the issue of scalability. The clustering was done on both datasets separately. (see figure 3)

Storage of Clusters. All the movies present in a specific cluster were stored in a csv file where each row of the file was one cluster and the ids of the movies were stored in each row.

Number of Clusters (size of K). Many articles on K means clustering denote that the number of clusters to be formed or the value of k depends on the data itself. In our case we found that for the Kaggle dataset number of clusters = 100 yielded optimum results and in the case of MovieLens dataset it was also found that number of clusters = 100 yielded optimum results.

Results: Querying and Cluster Evaluation

We have a total of 3 datasets: the movies dataset of MovieLens, the movies dataset of Kaggle and the user dataset of MovieLens. For the purposes of querying and cluster evaluation, the following steps were employed:

1. A user was selected from the user dataset
2. The top 3 highest rated movies were selected for that user
3. The clusters of those movies were found in both datasets separately

4. For the purposes of cluster evaluation, we return the intersection of the movies that the user has watched and rated and the movies that are present in that cluster. This was done separately for both datasets and for all three movies.

Illustration. If a user A has watched and rated the following movies:

userA: {m1: 4, m2: 3.5, m3: 3.5, m4: 3.5, m5: 4, m6: 4, m7: 5, m8: 4, m9: 4, m10: 5, m11: 4.5}

and the top 3 highest rated movies are:

{m7: 5, m10: 5, m11: 4.5}

Let the clusters for these 3 movies be:

{cluster7, cluster88, cluster94}

Let the movies in the clusters for the MovieLens dataset be the following:

Cluster 7. Contents of this cluster are:

m1, m2, m4, m7, m15, m88, m54

Cluster 88: Contents of this cluster are:

m10, m78, m41, m6, m84

Cluster 94: Contents of this cluster are:

m11, m3, m44, m71, m89, m27, m26, m35

Hence for the first movie m7 the movies we get after querying is the intersection of the movies the user has watched and rated and the movies present in the cluster of movie m7, in this case cluster 7. Therefore we get movies:

{m1, m2, m4, m7, m15, m88, m54} ∩ *{m1, m2, m3, m4, m5, m6, m7, m8, m9, m10, m11}.*

Which is movies:

{m1, m2, m4, m7}

The same approach was also followed for Kaggle Dataset.

After we get the required movies we compute the average rating of all movies that we have got as results. Here rating is the user's rating of the movies. Now if the cluster quality is good enough the clusters should contain movies which that same user has preferably rated well which in turn means that when we compute the average rating for each and every movie that we queried, in this case movies:

{m7, m10, m11}

Illustration. For the first movie m7 we got movies:

$$\{m1, m2, m4, m7\}$$

These movies have the following ratings given by the user:

$$\{m1: 4, m2: 3.5, m4: 3.5, m7: 5\}$$

Averaging these ratings we get:

$$\frac{(4+3.5+3.5+5)}{4} = 4 \quad (4)$$

The ratings are computed out of 5 hence 4 turns out to be a pretty good for a rating.

This measure gives us an idea of the cluster quality, there may be movies with lower ratings given by the user. But since the cluster should contain similar movies to the movies that the user has rated high, it is obvious that the average rating we got should also be high enough. Therefore a rating of 4/5 is pretty good for cluster quality.

The above average rating was computed for all three top movies separately to evaluate cluster quality. This same approach was also followed to evaluate cluster quality of the Kaggle dataset.

Fine Tuning Experiment 4: Inclusion of numeric data. A drawback of the previous approach was that we were using only non-numeric data in the previous experiment. This is not leading to full utilization of the datasets we had as we were not using the numeric data.

This approach involved 2 levels of clustering. At the first level clusters were formed using non-numeric data just like in the previous experiment. At the next level sub-clusters were formed using the numeric data of the movies present in each cluster. Numeric data was done in exactly the same way which is described in Experiment 2.

This was done in both datasets separately.

5 Conclusion

Recommender systems if well implemented can work wonders. According to JP Mangalindan's article on Fortune[2], 'Amazon reported a 29% sales increase to $12.83 billion during its second fiscal quarter, up from $9.9 billion during the same time last year. A lot of that growth arguably has to do with the way Amazon has integrated recommendations into nearly every part of the purchasing process from product discovery to checkout'. Clearly improving recommender systems is a giant leap in the direction of better lifestyles. With the ever increasing population data from all corners of the world is going to increase and this paper suggests a way to achieve scalable solutions to address the problem of cross-domain collaborative filtering.

[2] Check it out at: http://fortune.com/2012/07/30/amazons-recommendation-secret

References

1. Das, J, Majumder, S, Dutta, D, Gupta, P.: Iterative Use of Weighted Voronoi Diagrams to Improve Scalability in Recommender Systems. The 19th Pacific-Asia Conference on Knowledge Discovery and Data Mining, At Ho Chi Minh City, Viet Nam, Volume: 9077 of the series Lecture Notes in Computer Science pp 605-617. DOI: 10.1007/978-3-319-18038-0 47 (2015)
2. Larson, Y.S.M, Hanjalic, A.: Collaborative Filtering beyond the User-Item Matrix: A Survey of the State of the Art and Future Challenges. ACM Computer Surveys. DOI: http://dx.doi.org/10.1145/2556270 (2014)
3. Berkovsky, S, Kuflik, T, Ricci, F.: Cross-Domain Mediation in Collaborative Filtering. User Modeling 2007, 11th International Conference, UM 2007, Corfu, Greece, June 25-29, 2007, Proceedings. DOI: 10.1007/978-3-540-73078-1_44 (2007)