Universität Koblenz-Landau

Research Paper - Realtime Recommandation Systems

Scaling of Recommender Engines

Sahana Maddali, Dr. Sara de Nigris

January 2020

1 Introduction

With the world shifting to online and virtual retail, recommender engines have become a commodity in their own right. Consider the problem encountered by a daily newspaper. There exists, for this newspaper, a large and diverse reader-base. By diverse, we mean that the readers like to read from a wide range of topics such as politics, history, arts, culture, science and tabloid. It the newspaper's main interest to maintain, if not grow, it's reader base. The solution to the question "Which articles get printed in the newspaper?" is a classic optimization problem, with 'reader-base satisfaction' as the variable to be maximized, and 'area available for printing' as the presented constraint. The optimal solution would be to determine weight of each topic and print most popular news articles of each topic. Thus, the newspaper succeeds in catering to the wants of the many versus that of the few.

An online newspaper however is available to each one of its users independently of the others. In other words, it can allocate the the same 'area available for printing' individually to each of its users, opening up the possibility of distributing personalised content to each of its users. Commonly referred to as the 'long-tail' phenomenon, it describes the ability of any online businesses to suggest items of retail other than those that are only popular. Thus succeeding in catering to the wants of the one versus that of the many. In these situations, it is important to analyse user profiles to prolong engagement of the user on the online platform (a website, or an app), and come up with suitable personalised articles for that user. This task is performed by recommender engines.

A recommender engine looks at historical information about the customer, or user, to detect trends in purchases with some measure of accuracy to then predict for another user what he/she is most likely to purchase. The word purchase here is used in loose terms. It could also mean buying, viewing, reading, rating etc. There are several existing techniques that determine latent trends in purchasing patterns of people, how often these patterns occur, and with what probability a future purchase is dependent on these patterns.

An important factor to consider while discussing recommender engines is speed. A lot of valuable purchase data is available during customer engagement on the online platform, and it is profitable to exploit it at that moment. We then require real-time recommender engines. As the number of items (commodities) and users (customers) interacting on the online platform grow, the recommender must 'scale' efficiently with respect to the growing data base. This means that the recommender engine cannot become both slow and inaccurate as the data data grows. As predicted, the computation time required for highly accurate and undoubtedly reasoned recommendations is very expensive, because it depends on system architecture and response to user engagement on the online platform. This now requires expertise in database management. System architecture and a comparative study of various methodology are detailed in [8], [18]. Various novel methods and technological advances in this area have been collated in [6], discussing the new standard optimal requirements to keep up with the growing volume of big data. However, all of this is synonymous with high costs for

setting up and maintenance of recommendation engines. A solution is usually economical when either accuracy or speed can be compromised.

On the discussion of methodology we have, broadly speaking, two sections of discussion. Content-based filtering, or item-item filtering, is achieved by referencing each past purchase against every other past purchase, to determine the frequency of some emerging purchase pattern among items. Collaborative filtering, or user-item filtering, looks at users-items database on the whole to determine neighbourhoods for like users, and suggest items that were not previously purchased by this user, that are highly rated by other users. This method disregards any meta-data available for the items. There exist model-based and memory-based techniques for collaborative filtering recommeder engines.

We present our paper in the context of high speed, low accuracy (and therefore low cost) memory-based techniques. Our proposed algorithm generates recommendations of the type 'Top-N', where the goal is to achieve at least 'n' recommendations for a user with a predicted rating that the user is likely to give the item once purchased. With this method, recommendations can be computed online speedily, without requiring to call on the pre-computed offline model for predictions. Essentially, with our implementation we want to achieve great speed while compromising on accuracy. It is shown that random sampling techniques applied to latent frequent-itemset generation directly from the data set does not yield meaningful recommendations when the algorithm is implemented independently. However there is good scope for usage in Hybrid recommender engines.

2 Related Work

2.1 Evaluation Metrics

Any machine learning algorithm needs to pass a statistical test which determines how well it is performing its function. So, as described in [22], [9] and [13], we endeavour to find the meaningful metric.

An unsurprising question would now be : Is it even a recommender engine if it's not accurate? Several purposes for which recommender engines are built require high accuracy in prediction/recommendations. The above mentioned online newspaper is one such example. The quality of a recommendation with respect to accuracy is usually tested with rankings and ratings of the recommendations it generates. Ranking is the order in which the recommended items are presented, i.e. from most probable to be purchased to least. Rating is the review for the item by the user, usually interpreted over some numerical scale.

Although we began with the image of an e-commerce recommender engine, recommender engines have a wide array of applications. As an example, consider the recommender engine installed at a law firm for research purposes, that recommends cases to a lawyer to look for precedent. The recommender must suggest cases from diverse topics (i.e. civil law, corporate law, environment law to name a few). Meaning that, in this case we do not require recommendations

based on nearest neighbourhood formation to back up a legal argument. This requires arguably lower accuracy and higher coverage of topics than an online magazine that targets a user's preferences of articles.

Further, trends in purchases are susceptible to recurrent purchases of some items as compared to others. That is, some items may generally be more popular as compared to others, and are likely to get even more popular with time. Recommender engines are naturally sensitive to high magnitude in weight of popular items, causing recommendations lack in diversity. As detailed in [16], other than choice of data-set, the extent of accuracy (including factors of serendipity and coverage) of recommendations are important aspects to consider while interpreting the performance of a recommendations engine.

We argue that although our algorithm performs poorly when tested using accuracy metrics (such as MAE, RMSE, F1-score), it performs exceptionally well when tested using non-accuracy metrics (such as diversity, coverage, serendipity, novelty and relevance) and for time. Achieved earlier by [7], with hybrid reduced-dimension-space techniques, the result was improved scalability of the collaborative filtering algorithm, at the cost of accuracy of recommendations.

2.2 Data sets

An important aspect regarding the problem statement is to determine long-term and short-term preferences. As expanded in [12], [17], the period of existence of the data is important. Different data collected at different times describe and evolving profile of user-item interactions. A real-time recommender engine must be sensitive to these changes and also easily adaptable to them. Long-term preferences are dependent on historical user-data, for which recommendations can be computed with high accuracy using several model-based techniques: neighbourhood formation, any of several matrix factorisation methods (ALS, SGD, NMF to name a few), and some very basic logistic regression methods. A special case is recommender engines for text data (i.e. news, tweets etc.), where popular practice is to use Latent Dirichlet Models [21], which involve calculating 'a priori' probability distributions of topics for documents.These models can be trained on new up-to-date data regularly : once every week, day, hour, or in case of some very speedy matrix-factorization (backed up with advanced system architecture), some minutes. Impressive as it is, still not feasible for a real time recommendation engine. Short term preferences are real-time purchases of users on the online platform. These preferences are small amounts of input data that can be referenced against large amounts of historical data in real-time. The latter is the area that our approach explores.

Data not describing the rating of an item by a user is said to be implicit, usually encoded as 1 if purchased or 0 if not. While data which includes item ratings is said to be explicit, usually describe over some range of numbers. As an important note, one recognises that most businesses start off with little to no data for rating. Explicit data is widely accepted as being more accurate in user profiling [4]. Difference between implicit and explicit data, as well as a detailed study of estimating user preferences from implicit and explicit data have been

conducted [4]. Another desired feature of a recommender engine is that it treats changes in numerical scale, i.e. from discrete (1s and 0s) to continuous or resized discrete (0 to 10) with indifference. We therefore conduct experiments for our approach for explicit and implicit data sets.

3 New Approach

The recommender engine algorithm discussed in this paper is designed for real-time purposes and uses a memory-based collaborative filtering method for items. Recommendation engines are essentially a class of machine learning algorithms that more often than not determine patterns from the input data, or in other words, use unsupervised learning. As explained above, learning accurate pattern emergence from the data is time-intensive.

An accepted technique, described in [7], applies learning in a reduced dimensional space, effectively reducing the net amount of computation to learn emerging patterns, to ultimately improve scalability of the algorithm with data. The approach proposed in this paper proposes something similar, but also bypassing actual 'learning' for the recommender engine at run time.

In older 'a priori' estimation algorithms for pattern determination such as A-priori and FP-Growth, it is found how frequent combinations of items occur in purchases across the whole data set. The higher the frequency, the more recurrent that combination (or itemset) is believed to be, and stronger is that determined pattern of purchase. Based on a user-given threshold, the algorithm determines an itemset to be a 'potential itemset', a desirable itemset, if its frequency is above that threshold. Calculating recurrence of each combination of items for larger volumes of data tests the limit of system architecture and ultimately fails. See Figure 1.

3.1 Big Data

Over the years, the definition of bid data has evolved, as most likely will continue to do so. In the 1990s, accepted norm was to measure data with respect to volume, or the amount of memory required by a divide to store and process it. During the 2010s, big data had already surpassed sizes of zetta bytes, and so the metric changed from size to computational type of the system handling the data, see [14]. Generally recognised by one of its five features as enumerated in [19], its historical progression requires careful understanding and innovative insights, [23],[6], that of course need to keep up with its growing volume. Clearly, handling bid data requires sophisticated, and to re-iterate, expensive system architecture.

With this in mind, we look back at Figure 1 and notice that frequent itemset generation fails relatively quickly. As observed, the system architecture is not advanced and these algorithms are unsuitable for low-cost real-time applications. The approach discussed in this paper bargains for accuracy by implementing

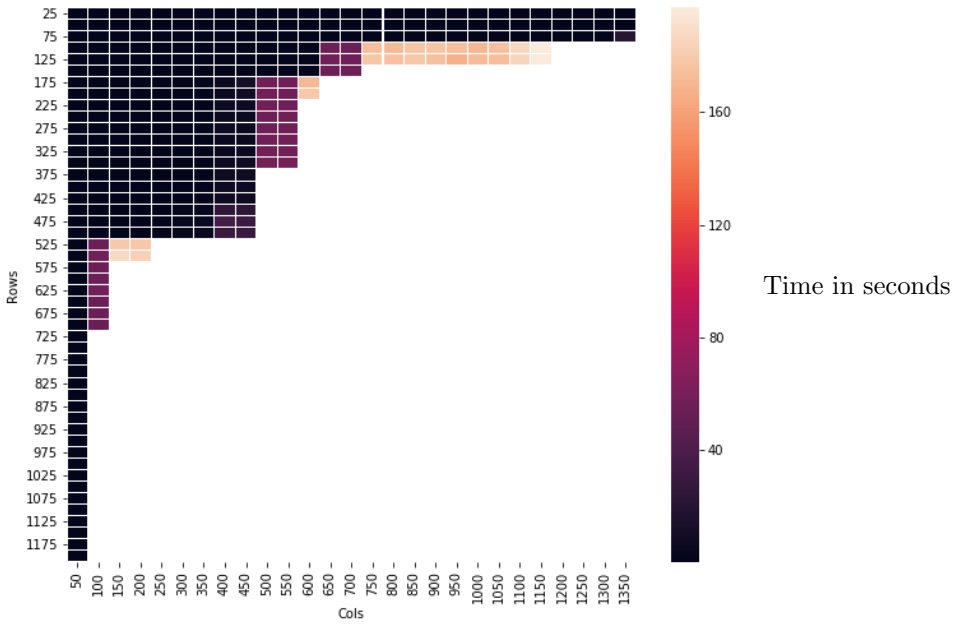

Figure 1: **Performance of FP-Growth algorithm as size of data scales up.** *The task was to understand how the FP-Growth algorithm behaves as we increase the number of items (Cols) and the number of users (Rows). We used the Online Retail data set [11]. Machine description : (Describe machine used to perform tests here). As we can observe, the algorithm doesn't return discernible patterns after just a few hundred users and items. (Perform better experiments with GPU if possible, to show difference between serial and parallel computing requirements.)*

a fast method to grow potential itemsets from a subset of data for real-time application.

3.2 The Algorithm

Data: item, rating
Result: recommendations with rating = $\min_r ating$
initialisation;
while *not minimum number of recommendations generated* **do**
  Step 1. *Find users* : find all users who have purchased item in the past;
  Step 2. *Find items* : find all items rated as 'rating' bought by these users;
  Step 3. *Potential itemsets* : find the largest intersection of all itemsets;
  Step 4. *Update item* : update to potential itemset
end
return (Final potential itemset);
Algorithm 1: Frequent itemset generation for Big and Wide Data

3.3 The Explanation

This approach is a quick and dirty fix to time consuming comparisons and accuracy computations. Consider an item of purchase i_0 to be of interest. This item could be the final item purchased by the user, using which we endeavour to recommend new items based on collaboratively filtering other items. We start by finding all users who have bought this item, which is the set of users

$$U_0 = [u_{0,1}, u_{0,2}, ..., u_{0,n}]$$

In loose terms this means we have found a neighbourhood of people with respect to this item. Next, we look at other historical purchases of this user-neighbourhood and find items with the same rating as our initial item. This determines items $i_{1,1}, i_{1,2}, ..., i_{1,n}$, using which we can form a set of n potential itemsets

$$I_1 = [(i_0, i_{1,1}), (i_0, i_{1,2}), ..., (i_0, i_{1,n})]$$

Each potential itemset contains the initial element i_0 and one similar item $i_{1,k}, k \in 1, ..., n, n \in \mathbb{N}$ Each potential itemset is some item-neighbourhood where people have bought and rated the items the same.

We now have a set of n potential itemsets with two items each. We repeat the above process to grow these itemsets if possible. We find m new user-neighbourhoods

$$U_1 = [u_{1,1}, u_{1,2}, ..., u_{1,m}]$$

, where $m <= n$. The new user-neighbourhood is always smaller than or of the same size as the older user-neighbourhood, because it might occur that none of the users may have bought the combination of items in some of the n potential itemsets. The thus formed new item-neighbourhood is of the form

$$I_2 = [(i_0, i_{1,1}, i_{2,1}), (i_0, i_{1,2}, i_{2,2}), ..., (i_0, i_{1,n}, i_{2,m})]$$

, each potential itemset with three elements.

This process is repeated for p iterations, until the number of unique items in I_p does not fall below a pre-defined threshold. I_p is now the selected recommendation for item i_0.

4 Notes on new Approach

This section draws attention to some features of the new approach, and how they are better when compared to other comparable real-time recommender engines.

- **Complexity** As one would immediately notice, the algorithms design has no taxing similarity computations. All neighbourhood formations can be achieved using simple set operations. Best case complexity for existing methods are as shown in Table 1. The new approach is much faster in returning recommendations as it only queries the database during runtime, and does not actually run expensive computations. Complexity of the model is flexible, as it is the sample size specified.

Algorithm	Neighbourhood-Formation	Matrix-Factorisation	Regression models
Complexity	$O(N^2S)$ [20]	$poly(nm^{O(r^2)})$ [5]	$O(p^2n + p^23)$ [2]

Table 1: Table to test captions and labels (Explain variables)

- **Implementation and Portability** The algorithm is simple enough to implement in several high level languages such as c++ and python. For this reason it can be developed or exported onto several platforms easily.

- **Integration** This algorithm can be used as a standalone recommender engine or in hybrid recommender engines as explained later in this paper. Consider, as a workaround for the low accuracy problem, that we are able to run an offline time-intensive accurate model on a regular basis, say every one week. The algorithm discussed in this paper could play the part of the real-time recommeder, sampling from pre-computed accurate results from an algorithm like ALS or SGD. This would allow for higher accuracy and also improved coverage of items in recommendations.

5 Experiments

The MovieLens100k data set [15] is the smaller of the GroupLens data sets that uses data from about 943 users and covers 1682 movies, producing 1,00,000 ratings. Algorithm 1 is used to determine recommendations for items rated on a 5 point scale, with half-point increments (1.0 points - 5.0 points). The

Book-Crossings data set [10] contains 1.1 million ratings of 270,000 books by 90,000 users. The ratings are on a scale from 1 to 10 (implicit ratings are also included). Algorithm 1 is used to determine recommendations for items rated on a 11 point scale, with one-point increments (0 points - 10 points). As the Book-Crossings data set contains ratings of value 0, this includes implicit information. We can appropriate pre-process the data set to create a purely implicit data set. For larger data sets, one requires still more advanced systems.

All experiments were perfromed on 16GB RAM 4-core 64-bit system.

5.1 Accuracy Metrics

When tested as a stand-alone recommender engine, the algorithm performed very poorly with respect to Accuracy metrics. Sampling with sample sizes 90% of the Book-Crossings data set yielded almost no true positive predictions, and the same was observed for the MovieLens20M data set. Shown below is a table of confusion matrices for the Book-Crossings data set and MovieLens20M data sets. As is evident, the algorithm is capable of rejecting items better than it is able to correctly predict items for a user.

	Predicted Positive	Predicted Negative
Actual Positive	0.031712	1.677590
Actual Negative	18.322410	18.322410

Table 2: *Book-Crossings data set* : averaged confusion matrix for most popular item

	Predicted Positive	Predicted Negative
Actual Positive	4.545508	10.405776
Actual Negative	18.193617	9.594224

Table 3: *MovieLens100k data set* : averaged confusion matrix for most popular item

5.2 Non-accuracy Metrics

Coverage is defined as the percentage of items recommended, denoted by :

$$\boldsymbol{c} = \frac{n}{N} * 100$$

where, c = *coverage*, n = number of items recommended and N = total number of items, [3].

As the Book-Crossings data set has both implicit and explicit recommendations, we get two measures of coverage. Table (4) gives the average coverage of the recommender tested for 20 most popular items, with N = all items (rated and not rated). Table (5) gives the average coverage of the recommender tested for 20 most popular items, with N = all rated items. The coverage of items over sample sizes 50% to 85% are shown as follows. ALS has proven to show low item coverage [1]. The following results show higher than usual item-coverage when compared to model-based techniques.

Sample size	$\boldsymbol{c}$
0.5	0.000080
0.55	0.000084
0.6	0.000095
0.65	0.000099
0.7	0.000094
0.75	0.000095
0.8	0.000106
0.85	0.000106

Table 4: Book-Crossings Data set : Average coverage over all items

Sample size	$\boldsymbol{c}$
0.5	0.000147
0.55	0.000154
0.6	0.000174
0.65	0.000181
0.7	0.000173
0.75	0.000173
0.8	0.000193
0.85	0.000194

Table 5: Book-Crossings Data set : Average coverage over rated items

The algorithm showed positive results when tested for time. Sampling with sample sizes 55% upto 85% of the Book-Crossings data set was achieved between 40-180 seconds, and for the MovieLens20M data set was achieved between 190-1100 seconds.

Sample size	c
0.5	0.004560
0.55	0.004398
0.6	0.005221
0.65	0.005289
0.7	0.004938
0.75	0.004992
0.8	0.005572
0.85	0.005545

Table 6: MovieLens100k Data set : Average coverage over rated items

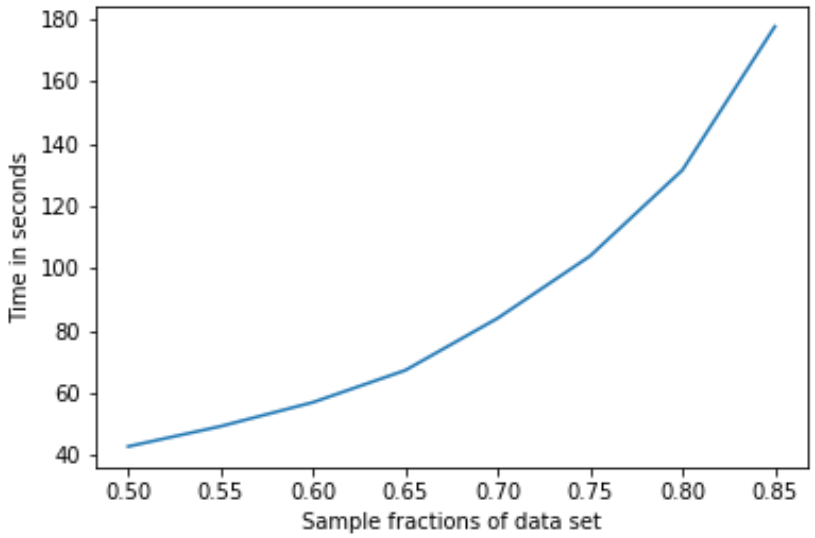

Figure 2: **Performance of Algorithm with respect to time, given a sampling size; Book-Crossings Data set**

Since this is a random sampling technique, in principle we can restrict the sample sizes to be small, and and and run the algorithm $n <= M < \infty$ number of times, where M is some upper bound on the number of iterations. This would give us a higher number of recommendations for lesser run-time. As described in the iPython Notebooks, we sample at a small sample size, but several times, which gives better estimate for true positives. The low accuracy problem could be improved in this manner. However, we increase the time

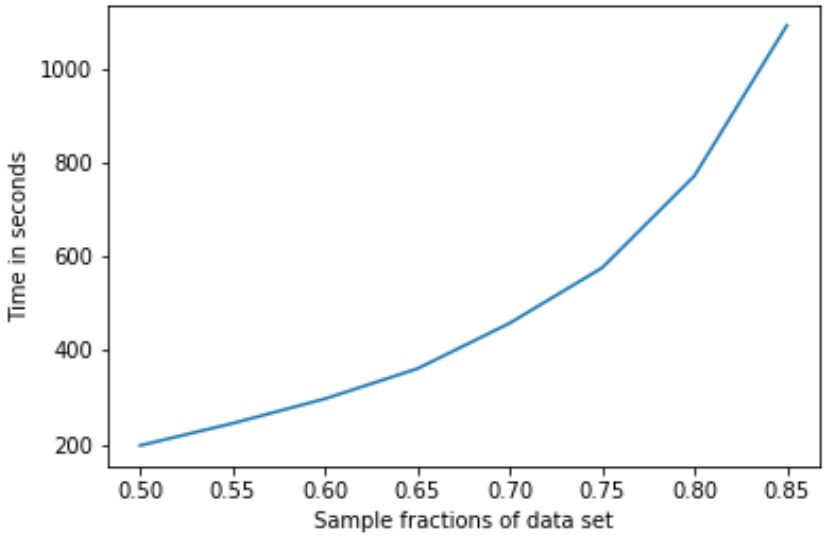

Figure 3: **Performance of Algorithm with respect to time, given a sampling size; MovieLens100k Data set**

required to recommend by the number of experiments we perform. This is again undesirable. A better option would be to use this algorithm in tandem with an offline time-intensive model.

6 Hybrid Recommender Engines

Based on the experimental results in the previous section, we can conclude that this algorithm is not useful as a stand-alone recommender engine. However, the low accuracy problem can be fixed when this algorithm is used in tandem with offline model-based techniques. Assume that we use a matrix-factorisation model (such as ALS) to generate highly accurate recommendations on a regular basis. Assume that we generate enough recommendations for each user, i.e. say < 10 recommendations per user. The random sampling algorithm can then be added as a final layer to the recommender to introduce higher coverage of recommendations. This is equivalent to saying that the random sampling algorithm will add some error to highly accurate recommendations in order to achieve higher coverage of items.
While accurate recommendations can be generated offline at leisure, the random sampling algorithm could easily be implemented online in order to generate fast and slightly less accurate recommendations. This approach could not be tested because implementation of matrix factorisation requires requires very

large RAM.

7 Conclusion

The algorithm proposed in this paper scales with size of data, and when used in tandem with offline model-based techniques generates recommendations of higher item-coverage faster than when only offline model-based techniques are implemented. When tested on two standard data sets (MovieLens100k and Book-Crossings) the results showed that the algorithm has extremely poor accuracy as a stand-alone recommender, but high item coverage and reduced recommendation time. The algorithm could function much better in Hybrid recommender engines. It was observed that for very large data sets, such as the MovieLens20M [15], the algorithm discussed in this paper also tends to fail with respect to speed as the system architecture was still not advanced enough.

References

[1] Bias disparity in collaborative recommendation:algorithmic evaluation and comparison.

[2] Computational complexity of machine learning algorithms.

[3] Recommender systems — it's not all about the accuracy.

[4] Xavier Amatriain, Josep M. Pujol, and Nuria Oliver. I like it... i like it not: Evaluating user ratings noise in recommender systems. In Geert-Jan Houben, Gord McCalla, Fabio Pianesi, and Massimo Zancanaro, editors, *User Modeling, Adaptation, and Personalization*, pages 247–258, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[5] Sanjeev Arora, Rong Ge, Ravi Kannan, and Ankur Moitra. Computing a nonnegative matrix factorization – provably. *CoRR*, abs/1111.0952, 2011.

[6] Bahaaldine Azarmi. *Scalable Big Data Architecture*. 01 2016.

[7] J. Konstan B. Sarwar, G. Karypis and J. Riedl. Analysis of recommendation algorithms for e-commerce.

[8] Carneiro V. Fernandez D. Formoso Cacheda, F. Comparison of collaborative ltering al-gorithms: Limitations of current techniques and proposals for scalable, high-performance recommendersystems. *ACM Trans. Web 5*, 2011.

[9] et al Cai-Nicolas Zeigler. Improving recommendation lists through topic diversification. *http://files.grouplens.org/papers/ziegler-www05.pdf*, 2005.

[10] Joseph A. Konstan Georg Lausen Cai-Nicolas Ziegler, Sean M. McNee. Improving recommendation lists through topic diversification. *Proceedings of the 14th International World Wide Web Conference (WWW '05)*, 2005.

[11] Kun Guo Daqing Chen, Sai Liang Sain. Data mining for the online retail industry: A case study of rfm model-based customer segmentation using data mining. *Journal of Database Marketing and Customer Strategy Management, Vol. 19, No. 3*, 2012.

[12] Elena Viorica Epure, Benjamin Kille, Jon Espen Ingvaldsen, Rebecca Deneckere, Camille Salinesi, and Sahin Albayrak. Recommending personalized news in short user sessions. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 121–129. ACM, 2017.

[13] YC Zhang et al. Auralist: Introducing serendipity into music recommendation. *http://www.cs.ucl.ac.uk/fileadmin/UCL-CS/research/Research$_N$otes/RN$_1$1$_2$1.pdf*, 2011.

[14] Charles Fox. *"Data Science" and "Big Data"*, pages 1–14. Springer International Publishing, Cham, 2018.

[15] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems 5, 4, Article 19 December 2015*, 2015.

[16] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004.

[17] Peter Dolan Jiahui Liu and Elin Rønby Pedersen. Personalizednews recommendation based on click behavior.

[18] Siddharth Dinesh Joeran Beel. Real-world recommender systems for academia: The pain and gain in building, operating, and researching them. *https://arxiv.org/pdf/1704.00156.pdf*.

[19] Stephen H. Kaisler, Frank Armour, J. Alberto Espinosa, and William H. Money. Big data: Issues and challenges moving forward. *2013 46th Hawaii International Conference on System Sciences*, pages 995–1004, 2013.

[20] C. Lu, T. Masuzawa, and M. Mosbah. *Principles of Distributed Systems: 14th International Conference, OPODIS 2010, Tozeur, Tunisia, December 14-17, 2010. Proceedings*. LNCS sublibrary: Theoretical computer science and general issues. Springer, 2010.

[21] Talel Abdessalem Anthony Barré Antoine Cornuéjols Marie Al-Ghossein, Pierre-Alexandre Murena. Adaptive collaborative topic modeling for online recommendation.

[22] et al Sean McNee. Being accurate is not enough: How accuracy metrics have hurt recommender systems. *http://grouplens.org/site-content/uploads/accurate-CHI-20061.pd*, 2006.

[23] Zibin Zheng, Jieming Zhu, and Michael Lyu. Service-generated big data and big data-as-a-service: An overview. pages 403–410, 06 2013.