Discovering Similar Products in Fashion E-commerce

Amber Madvariya
amber.madvariya@myntra.com

Sumit Borar
sumit.borar@myntra.com

ABSTRACT

In recent years, item-item collaborative filtering algorithms have been studied thoroughly in recommender systems. When applied in the context of fashion e-commerce these algorithms can be used to generate similar recommendations, personalize search results and to build a framework for creating clusters of similar products. The efficacy of these algorithms when applied in fashion domain, rely on accurately inferring a user's fashion taste and matching them to a product. Our work hinges around discovery of similar products using two different item-item collaborative filtering algorithms. We identify and address some unique challenges while applying these algorithms in the dynamic fashion e-commerce environment. We study and evaluate their performances through precision/recall measures, live A/B tests and baseline them against a content based approach. We also discuss effects of the transient nature of the industry on a user's fashion taste.

KEYWORDS

Recommender Systems, Item-Item Collaborative Filtering

1 INTRODUCTION

Fashion shopping is a complex intersection of product styles with users' taste where products are usually described in terms of several attributes such as silhouette, line, hem length, color, fabric, waist length and so forth. These attributes are dynamic in nature and vary as trends in fashion change. However, these attributes are not typical representatives of a user's fashion taste. Users rely more on look, feel, popularity and other intrinsic factors to identify their taste.

The advent and growth of fashion e-commerce, pose even more challenges towards providing users with relevant products. As the industry grows, there are thousands of products available in a catalogue with similar set of attributes. Also users' needs in fashion are not specific as compared to hard goods like electronics, therefore they tend to browse significantly more products on a fashion portal before clicking on a particular product. In general, at Myntra we observe the average click depth i.e. number of products viewed before the first click, to be around 90. Correctly inferring a user's intent in a session becomes critical for providing a better shopping experience. A user's click on a prod-uct is a proxy for his/her interest in that product. We can utilize this information to narrow down his/her intent. Thereafter, serving a user with similar products to the clicked one, helps them navigate through the vast catalogue and find relevant products. We also use similar products to personalize search results as majority of queries on our platform are broad in nature like 't-shirts' and ranking results based on user's interest improves the overall search relevance.

Therefore, discovering similar products in fashion e-commerce becomes an interesting data mining and business problem. In this paper, we present our work on prescribing similar products using item-item collaborative filtering algorithms in a fashion context.

Similarity between items is usually computed in terms of content based similarity, item-item based collaborative filtering [12] or the hybrid of two. Content based similarity measures rely on a curated taxonomy to represent a particular content. A widely cited example of content based system is that of Pandora radio[11], which uses the Music Genome Project to tag their songs and artists from a set of 450 manually curated attributes. Unlike music, the dynamicity of trends in fashion, makes its taxonomy ephemeral and new products continuously require new attributes to be added to the taxonomy. Maintaining and curating this taxonomy system becomes an unscalable task over time. Another limitation with this approach is that attributes are unable to capture user taste completely.

Unlike content based similarity measures, item-item collaborative filtering algorithms use user signals instead of a taxonomy to compute item-item similarity. In the domain of e-commerce users' feedback is not explicit [7] and needs to be inferred from user signals. In traditional e-commerce platforms [9], user purchases are used to compute item similarity. The challenge in using only user purchase data to compute item similarity in fashion e-commerce is that it is sparse and erratic. This is due to two primary factors, the short life cycle of products and the breadth of the catalogue available. Thus similarity measures relying solely on user purchase data tend to perform poorly in this setting. So we consider purchase data along with signals like clicking on a product and adding a product to a cart to compute item similarity.

The challenges with using these implicit feedback signals are following:

- Quantifying the chosen signals by assigning relative weights to each.
- Normalising effect of popular products i.e. products which have signals from a large number of users.

In this work, we look at two representations to model item-item collaborative filtering. We compare these two approaches with a content based approach, where we use the annotated taxonomy of a product to get a feature representation for it.

In the following sections of this work, we describe these three approaches in detail and compare their performance via A/B tests and precision-recall test. Finally, we try to conclude by inferring whether the transient nature of the industry affects user taste or not.

2 METHODOLOGY

For the remaining sections of this paper, we will use the following terminology. Let P be the set of all products and S be the set of all user sessions. A session contains all activity by a user within a 30 minute window from the time he/she logs into the portal annotated by time-stamp. We record user signals like product list views, product clicks, addition to carts, orders placed and so on.

For all the three approaches mentioned below, we split our data at the article type (e.g Men-Tshirts, Men-Shirts, Women-Dresses) level, since products similar to a product p_i should be from the same article type. Splitting the data at the article type level has the added advantage that it reduces the set of products from which we find similar products for p_i.

2.1 Product Attribute Vectors

Each product in our system is annotated with attributes generated from a manually curated taxonomy. These attributes are broadly classified into two types, general attributes like brand, color, price bands etc which are applicable for all article-types and article-type specific attributes (ATSA) like collar type, sleeve length, neck type for T-shirts. Each attribute is represented as a key-value pair e.g. collar type of a t-shirt is the key and different collar types like round-neck, v-neck, polo-neck are values. Each product is represented as a vector in the real space, where each dimension represents an attribute. We use binary values to populate a particular dimension i.e. if the attribute represented by the dimension is a part of the set of attributes which were annotated to the product, then we populate the dimension as 1, else we populate it as 0. Along with product attributes, we extract relevant bi-grams using log likelihood ratio scores[6] from the product descriptions. We utilize a L2 norm to normalize these product vectors.In order to generate N similar products for a product p_i, we discover the N closest neighbors of p_i, keeping cosine distance as our distance metric.

We analyse this approach here to benchmark the performance of item-item collaborative filtering algorithms. The major challenge with this approach is that the set of attributes representing a product is not comprehensive and could results in products not being annotated by some key information.

2.2 Item-Item Weighted Graph

In this approach, we use an undirected weighted graph representation to model product relationships in the system. In this representation nodes represent products, edges represent associativity between products and edge weights represent degree of associativity between products. This approach was first introduced by YouTube [4], however our formulation to calculate edge weight is different than the one showed in that work.

In order to to generate this graph, we use a well-known technique known as association rule mining[1] or co-visitation counts. We consider a user session s_i, where $s_i \subset S$, and generate a set of products $\{p_1, p_2, \ldots p_n\}$ clicked in that session. Within this set, we compute all pair combinations of products (p_i,p_j) which were co-browsed together. We count all the occurrences where p_i and p_j were co-browsed together, across S and denote the total count of co-occurrences of (p_i,p_j) as c_{ij}. We assign edge weights as w_{ij} and calculate it using the following formulation of normalised point-wise mutual information (NPMI) [2]:

$$w_{ij} = \frac{-1}{\log p(i,j)} * \log \frac{p(i,j)}{p(i)p(j)}$$

where $p(i,j)$ is the probability of occurrence of the pair (p_i,p_j) and $p(i)$ and $p(j)$ is the probability of occurrence of p_i and p_j respectively. We can use the following formulations for the values of $p(i,j)$, $p(i)$, $p(j)$:

$$p(i,j) = \frac{c_{ij}}{C} \qquad p(i) = \frac{c_i}{C} \qquad p(j) = \frac{c_j}{C}$$

where, c_{ij} = count of occurrences of pair (i,j),

$$c_i = \sum_{k \subset P} c_{ik} \qquad \text{and} \qquad C = \sum_{i,j \subset P} c_{ij}$$

Then, to find similar products for a product p_i, we locate p_i in the graph, we consider it's adjacent nodes and among them pick the top N neighbors, sorted in descending order by edge weight. Here N is a hyper-parameter which denotes the number of similar products we want to find for p_i.

The advantage of using this formulation to generate edge weights, is that it normalises the effect of popular products. If we consider a pair of products (p_i,p_j) where p_j is browsed more across all sessions as compared to p_i, then the value of c_j will be high, resulting in a high value of $p(j)$. The high value of $p(j)$ results in the PMI of (p_i,p_j) turning out to be low, even though the co-occurrence counts of (p_i,p_j) might have been higher as compared to other pairs containing p_i.

One challenge with using this approach is the noise prevalent in the input data. Some pairs might have a low co-browsing count or low PMI score to form a meaningful edge. To tackle this challenge, we put a threshold on both the co-browsing counts and the edge weights generated using PMI scores. After experimenting, we found 5 to be a suitable threshold for co-browsing count and 0.15 to be a suitable threshold for PMI score. These thresholds change depending upon the number of products and user sessions.

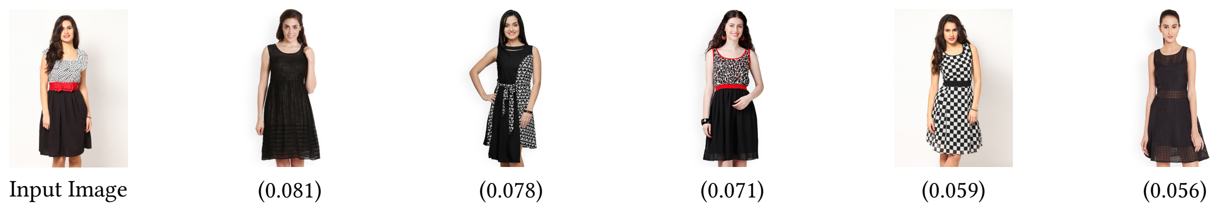

Figure 1: Example of similar products found using product attributes. The numbers at the bottom of a given image represent the similarity between the given product and the input product.

Another source of noise is the session in consideration itself. This entire approach hinges around the assumption that products browsed in a session are similar i.e. there is some context to products browsed by a user in a session. If the number of products browsed in a session are low, say 2 or 3, or if different article types were browsed randomly in a session, without any intent, then these sessions will add noise to our system. To tackle this problem, we use the concept of coherent sessions. We define a session as coherent, if in that session, the user has browsed at least three products of the same article type, and not more than two article types.

As we have considered sessions to construct the graph, it results in edges being formed only between products which were present in the system at the same time. Therefore, this graph captures the ephemeral nature of fashion trends, because products present across two different points in time would never form an edge. So, while finding similar products from this graph for p_i, the output products represent a trend from the time when p_i was present.

2.3 Product-User Vectors

In this approach, we represent each product as a vector of user signals in the real space, with each dimension representing a user. We consider three user signals i.e. product clicks, addition of products to cart and checkout for populating values in each dimension. We start by aggregating these user signals across S, for a product-user combination (p_i,u_i), and generate a chronologically sorted list of all the identified signals that user u_i has generated for product p_i, where $p_i \in P$. If we denote a product click by C, addition to cart as T and checkout as O, one example of such representation could be

$$(p_i, u_i) = \{C, C, C, T, O\}$$

After generating this list, we quantify our user signals, based on past data. We estimate the relationship between number of product clicks, add to carts and checkout using a linear classification model with each session as a data point. In the linear classification model, we use product clicks and add to cart signals as input and the checkout event as the target value.

The weight for checkout event is considered as 1, since it is the target value in the classification model. So, for the products which a user has bought, we populate a value of 1 in that user's dimension for those products. For products, which the user hasn't bought but has clicked or added to cart, we populate it's vector with weights that we obtained from the model. For users, who have not generated any of the above signals for a product, we don't populate any value in their dimensions. This results in the vector for each product being sparse, since the subset of users who have generated signals for a product is very small as compared to all users.

After computing the vector for a product p_i, we take it's L2 norm and find it's N similar products by finding it's N nearest neighbors using the following formulation of cosine similarity as the distance metric.

$$cos(i, j) = \frac{\vec{p_i} \cdot \vec{p_j}}{|p_i| * |p_j|}$$

where $\vec{p_i}$ and $|p_i|$ represent the vector of p_i and it's magnitude, "$\cdot$" represents the dot product between the two vectors.

In order to get accurate cosine similarities, we utilise a compressed sparse row (CSR) [3] matrix representation of the vectors, instead of approximate nearest neighbor algorithms[10]. Each row of this matrix represents a product and each column represents a user. We multiply this matrix by its transpose, such that the (*i,j*)th entry of the resultant matrix gives us the cosine similarity between p_i and p_j. Another advantage of using this representation is that it reduces computation time because of inherent sparse matrix optimisations.

We observe that products which are globally popular are dense as compared to products which are less popular. This results in popular products being close neighbors to many products, if their vectors are not normalised. Taking a L2 norm reduces the magnitude along each dimension of popular product and normalizes for global popularity.

In this approach, since we have aggregated user signals across S, it results in the system being agnostic of fashion trends. Unlike the item graph approach, we find multiple instances of linkage of products across time. Hence, while finding similar products for a product p_i using this approach, we find that the output products do not necessarily reflect trends from the same time-frame as when p_i was present in the system.

3 ANALYSIS AND RESULTS

3.1 Data

We split our sessions data into two non overlapping sets, a training set, which is used to generate the similar products, and a test set, which is used to evaluate the approaches. On an average, each session contains 5 products. The training set consists of ~200M sessions with ~12M unique users. Using this training set, we find similar products for ~1.3M products. The test set is generated from ~10M sessions with ~500K unique users. To generate the test set of products for p_i, we take each session in the test data, where

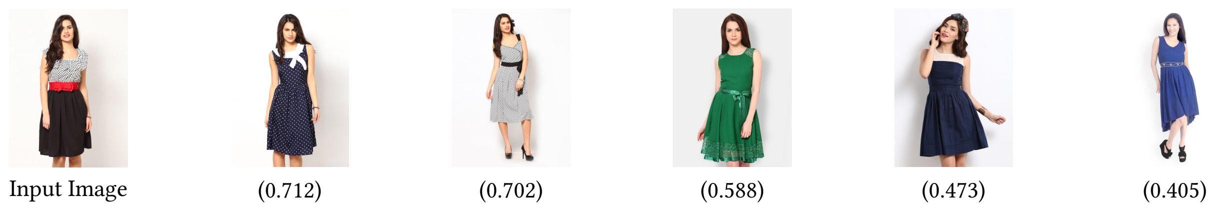

Figure 2: Example of similar products found using item-item graph. The first image is of the input product and the rest of the images are it's top 5 adjacent nodes. The numbers at the bottom of a given image are the PMI scores between the given product and the input product

p_i was clicked, and we consider the product which was clicked immediately after p_i in this session. Aggregating across all the sessions in the test data, we create a set of products which were clicked immediately after p_i. Using the session test data, we were able to generate test sets for ~330K products, with the average size of the set being 164.

We use a series of MapReduce computations [5] to generate the item graph and to aggregate user signals for each product user combination. Further, to create the CSR matrix for product user vectors, we use the CSR sparse matrix routine in scipy [8].

3.2 Analysis

3.2.1 ***Browsing behaviour by Gender.*** To analyse browsing behaviour by gender, we look at density statistics from both the item-item graph and product-user vectors.

For item-item graph, we define, the density of a graph as

$$d = \frac{e}{n * (n-1)/2}$$

where e is the number of edges and n is the number of nodes. Density here represents the ratio of number of edges formed in a graph to all possible edges in the graph. Below we tabulate density statistics of graphs for some prominent article types

Article Type	**Number of nodes**	**Number of edges**	**Density of Graph**
Women-Tops	62727	7325578	0.0037
Men-Tshirts	74763	9092631	0.0033
Women-Jeans	7255	303871	0.0115
Men-Jeans	19958	1519793	0.0076
Women-Casual Shoes	6932	287098	0.0120
Men-Casual Shoes	26402	2279818	0.0065
Women-Sports Shoes	2124	31327	0.0139
Men-Sports Shoes	8918	341966	0.0086

As can be observed from the graph, for two comparable article types like Men-Jeans and Women-Jeans, the women article types have higher density as compared to men article types, despite women article types having lower number of nodes. It can be inferred from the higher density of women article types that women browse more products than men before making a selection.

For product user vectors, density of a sparse vector is defined as the ratio of number of non-zero values in a vector to it's length. In the table below, we tabulate density statistics for vectors of different article types.

Article Type	**Number of Vectors**	**Number of Users**	**Average Vector Density**
Women-Tops	68553	5038512	0.00042
Men-Tshirts	85123	7155934	0.00027
Women-Jeans	8070	2497915	0.0013
Men-Jeans	22099	4648243	0.00072
Women-Casual Shoes	5942	2508693	0.0011
Men-Casual Shoes	28398	6984123	0.00057
Women-Sports Shoes	2459	1768939	0.0019
Men-Sports Shoes	9527	4738348	0.0011

Similar to item-item graph, we notice that women article types are more dense as compared to men products. The higher density of women product vectors corroborates our earlier inference that women products are browsed more as compared to men products.

3.2.2 ***Difference in Style Catalogued Date.*** Each product in our system is tagged with a style catalogued date (SCD), which represents the date when the product was introduced to the system. We calculate the average difference between SCD of a product and it's similar products and further average it over an article type.

Fig 8 represents average difference in SCD, between input products and their similar products averaged over article types for the two approaches. It can be observed from the plot that average difference in SCD for product-user vectors is 2-3 times more than that of item-item graph. Hence, it can be inferred from the plot that similar products found using item-item graph are from the same time-frame, while similar products found using product user vectors link across time. This validates our claim that similar products found using item-item graph are cohesive of fashion trends, while those found using product user vectors are agnostic of them.

3.3 Results

3.3.1 ***Precision Scores.*** Denoting the test set of p_i by T_i and the set of similar products for it as R_i, we use the following formulation to calculate precision:

$$precision = \frac{|R_i \cap T_i|}{|R_i|}$$

The size of R_i is a hyper-parameter, since we can set the number of similar products that we want to find for p_i, and we denote it as N.

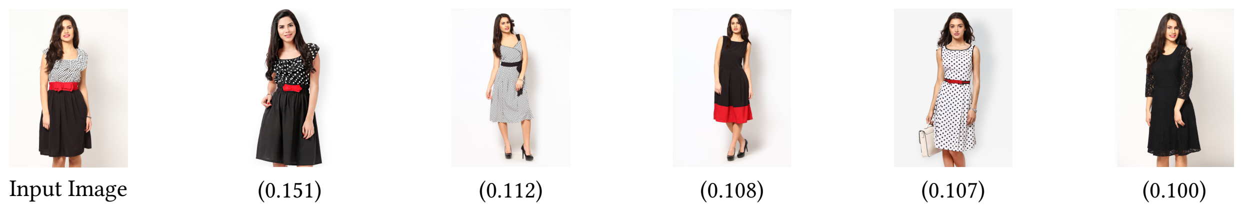

Figure 3: Example of similar products found using product user vectors. The first image is of the input product and the rest of the images are it's top 5 nearest neighbors. The numbers at the bottom of an image are the cosine similarities between the given product and the input product

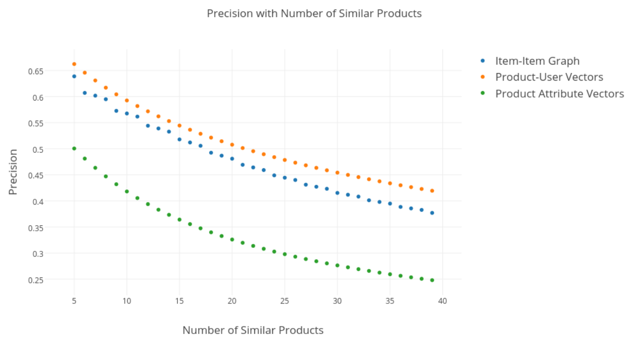

Fig 4: Precision versus N for the three approaches

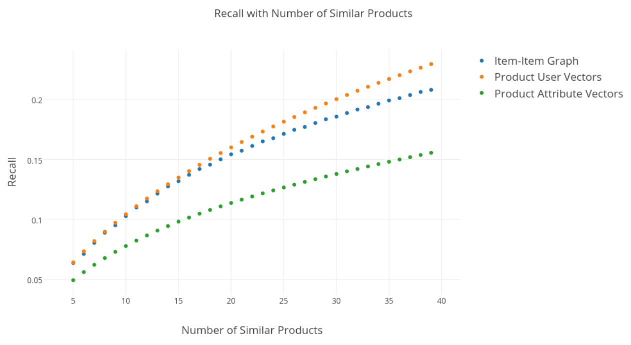

Fig 5: Recall versus N for the three approaches

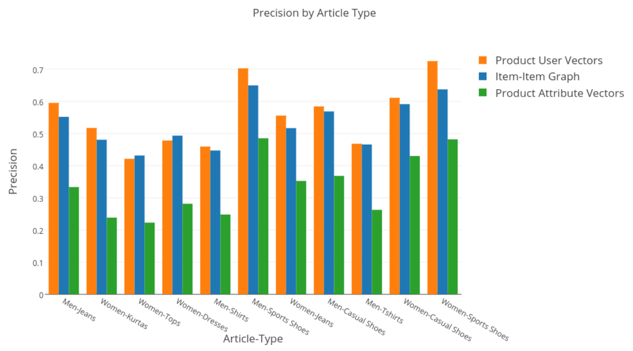

Fig 6: Precision for different article types

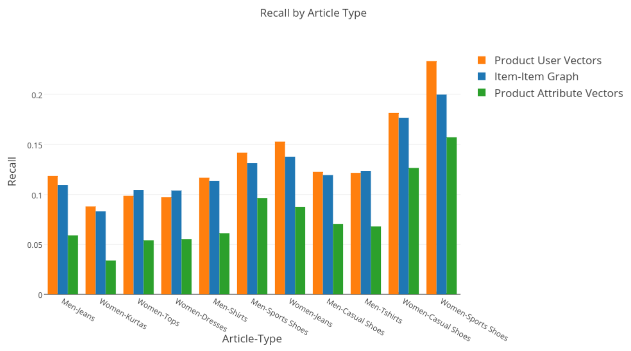

Fig 7: Recall for different article types

Fig 4 shows the average precision of the three approaches for different values of *N*. Fig 6 shows precision by article type for the three approaches calculated with a value of *N* as 25.

3.3.2 ***Recall Scores**.* Going with the same notations as above, we use the following formulation to calculate recall:

$$recall = \frac{|R_i \cap T_i|}{|T_i|}$$

Fig 5 shows the average recall of the three approaches for different values of *N*. Fig 7 shows recall by article type for the three approaches, with the same value of *N* as for Fig 6 i.e. 25.

It is evident from Fig 4-7 that the two collaborative filtering approaches significantly out perform the content based approach. Among the two item-item collaborative filtering approaches we calculate from the numbers of Fig 4 & 5 that product user vectors show an average improvement of 7% and 5.1% over item-item graph for precision and recall, respectively. Also we observe that product user vectors have better precision recall numbers across majority of article types as compared to item-item graph.

3.3.3 ***A/B Tests**.* For the A/B test, we render the similar products on the product details page of the input product. We test the above three approaches by assigning randomly selected 10% of traffic to each treatment (100k users for each treatment). We recorded the number of product views and the number of clicks for the three approaches over a period of three weeks. Figure 9 shows the CTR (click through rate defined as ratio of clicks to views) recorded during the test period for the three approaches averaged over each day of the week. From the test, we observe that product user vectors show an average CTR improvement of 5% over item-item graph with a p-value[13] of 0.019. Also, we notice that item-item graph and product user vectors show an improvement of 50% and 58.4% over product attribute vectors with a p-value of $1.2*10^{-5}$ and $6.6*10^{-6}$ respectively.

4 CONCLUSION

In this paper, we have identified and addressed some of the challenges faced by item-item collaborative filtering approaches which are unique to fashion e-commerce domain. We formulate a new method to combine and quantify various user signals, which can be used as input to these approaches, in e-commerce domain.

We propose a new method to evaluate similar products in an offline setting. We compare and evaluate three approaches to discovering similar products in both offline and online tests. We observe a significant improvement in the performance of item-item collaborative filtering approaches as compared to the content based approach. Furthermore, among the two item-item collaborative filtering approaches product user vectors perform noticeably better than item-item graph. We also compare the effects of fashion trends with respect to the two approaches and based on results of the tests, infer that a user's preference to products is independent of rapidly changing fashion trends.

5 ACKNOWLEDGEMENTS

We thank Deepak Warrier, Ankul Batra, Sagar Arora, Ullas Nambiar, Ashay Tamhane and Kunal Sachdeva for their contributions

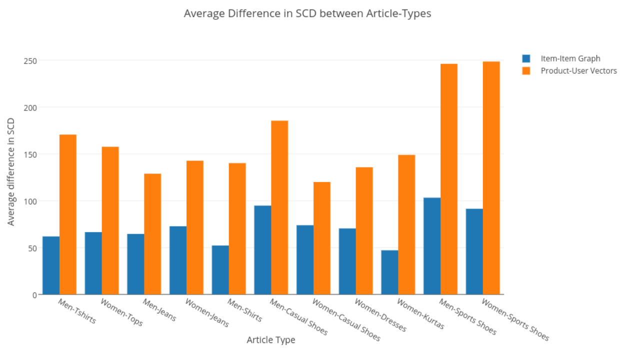

Fig 8: Average difference in SCD by article type

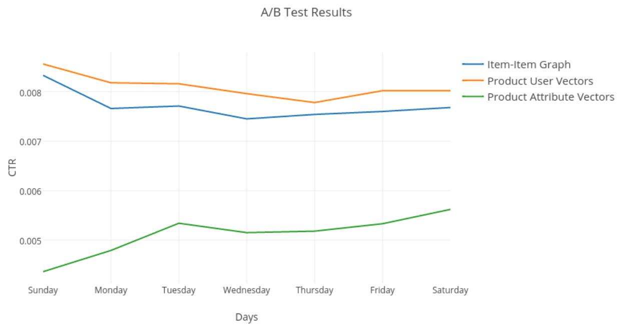

Fig 9: Average CTR over a period of 3 weeks

in reviewing this work and their inputs to algorithm design and evaluation.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In *Acm sigmod record*, Vol. 22. ACM, 207–216.

[2] Gerlof Bouma. 2009. Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL* (2009), 31–40.

[3] Aydin Buluç, Jeremy T Fineman, Matteo Frigo, John R Gilbert, and Charles E Leiserson. 2009. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*. ACM, 233–244.

[4] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. 2010. The YouTube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 293–296.

[5] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[6] Ted Dunning. 1993. Accurate methods for the statistics of surprise and coincidence. *Computational linguistics* 19, 1 (1993), 61–74.

[7] Tv Genius. 2011. An Integrated Approach to TV & VOD Recommendations. (2011).

[8] Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. (2001–). http://www.scipy.org/ [Online; accessed 2016-10-14].

[9] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.

[10] Marius Muja and David G Lowe. 2009. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. *VISAPP (1)* 2, 331-340 (2009), 2.

[11] Pandora Radio. [n. d.]. Music Genome Project. ([n. d.]). https://www.pandora.com/about/mgp

[12] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. ACM, 285–295.

[13] Bernard L Welch. 1947. The generalization ofstudent's' problem when several different population variances are involved. *Biometrika* 34, 1/2 (1947), 28–35.