Selecting Content-Based Features for Collaborative Filtering Recommenders

Royi Ronen, Noam Koenigstein, Elad Ziklik and Nir Nice
Microsoft Israel
{royir,noamko,eladz,nicen}@microsoft.com

ABSTRACT

We study the problem of scoring and selecting content-based features for a collaborative filtering (CF) recommender system. Content-based features play a central role in mitigating the "cold start" problem in commercial recommenders. They are also useful in other related tasks, such as recommendation explanation and visualization. However, traditional feature selection methods do not generalize well to recommender systems. As a result, commercial systems typically use manually crafted and selected features. This work presents a framework for automated selection of informative content-based features, that is independent of the type of recommender system or the type of features. We evaluate on recommenders from different domains: books, movies and smart-phone apps, and show effective results on each. In addition, we show how to use the proposed methods to generate meaningful features from text.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Data Applications - *Data Mining*.

Keywords

Feature Selection, Recommender Systems, Collaborative Filtering, Content Based, Cold Start

1. INTRODUCTION

Collaborative Filtering (CF) recommender systems have become an essential component in many in e-commerce sites and digital markets such as Amazon, Ebay and the Xbox Marketplace [8, 12, 15]. CF algorithms are typically favored over content-based algorithms [11] because of their overall higher accuracy in predicting common purchase patterns. However, by their nature, CF algorithms face challenges in modeling and recommending items with little or no usage (the "cold-start" problem [16]). Items meta-data in the form of content-based features, has been used to alleviate the cold-start problem and improve accuracy [1, 3, 4, 5]. Features are also useful for providing explanations to recommendations [17] and for visualization [7].

RecSys'13, October 12–16, 2013, Hong Kong, China.

ACM 978-1-4503-2409-0/13/09 ...$15.00.
http://dx.doi.org/10.1145/2507157.2507203.

This work studies the problem of evaluating the quality of meta-data features. We present an algorithmic framework which is independent of the specific recommendation algorithm for which the selection is made. Instead, the recommendation algorithm and other parameters are pluggable variables. Two types of selection are discussed: one for scoring meta-data attributes, and another for scoring meta-data labels (to be defined later on). Both have been successfully used to enhance recommendations in the Xbox marketplace, serving recommendations to more then 50 million users worldwide [8, 12, 14, 13].

We evaluate our methods and show their high effectiveness by experimenting with books, movies and data from windows Phone 8 smart-phone apps. In addition, we show how to use the framework to automatically generate informative labels from item descriptions and present the extracted labels for a qualitative evaluation.

1.1 Content-Based Features

Item catalogs in e-commerce marketplaces typically include meta-data features in the form of *attributes*. The attributes may be numerical, categorical, ordinal, binary, etc. For example, some common attributes are *price*, *brand* and *is-on-sale*. Another common type of features are labels, or tags, assigned to items by consumers, experts, or extracted from text by an algorithm. A label is typically a word, an n-gram, or a short phrase describing the item. Labels follow the 'bag-of-words' model. They form a closed dictionary, and every label may or may not be assigned to any item. Some examples of movie labels are: *location-usa*, *horror*, *kids* and *funny*.

In most catalogs there exists a subset of features that is highly informative with regard to the recommendation task at hand. However, many other features are often redundant or irrelevant with regard to CF. For example, in the context of books recommendations, *author* is informative, while *cover-color* is not. Despite a substantial body of work on feature selection for Data Mining algorithms, for most algorithms it is unclear how to extend them to the case of a recommendation system.

1.2 On Feature Selection

We briefly survey the three main categories of feature selection algorithms:

Wrapper methods evaluate different subsets of features by training a model for each subset and then evaluating each subset's contribution on a validation dataset. As the number of all possible subsets is factorial in the number of features, different heuristics are used to choose "promising" subsets (forward-selection, backward-elimination, tree-induction, etc.). Wrapper methods are independent of the prediction algorithm.

Filter methods are typically based on heuristic measures, such as *Mutual Information* or *Pearson Correlation*, to score features based on their information contents w.r.t. the prediction task. Similar to wrapper methods, filter methods are also independent of the algorithm in use. However, they do not require training many models and therefore scale well for large datasets. Yet, filter methods can not be naturally extended to recommender systems, in which the prediction target varies and depends both on the user's history and on the item under consideration. This work proposes a framework and algorithms to address the above difficulties.

Embedded methods are a family of algorithms in which the feature selection is performed in the course of the training phase. Unlike wrapper methods, they are not based on cross-validation and therefore scale with the size of the data. However, since the feature selection is an inherent property of the algorithm, an embedded method is tightly coupled with the specific model: If the recommendation algorithm is replaced, features selection needs to be revisited (e.g., in the context of recommendation systems see [9]).

2. FEATURE SCORING ALGORITHMS

As explained above, we distinct between two types of item features: *attributes* and *labels*. For attributes, we denote by $s.a$ the value that item s has for the attribute a. For labels, we denote by $s.L$ the set of labels associated with the item s ("bag-of-words").

Our algorithms depend on a pluggable features similarity function $sim(\cdot,\cdot)$ between two attribute values or between two labels. The function $sim(\cdot,\cdot)$ may be a content-based function that depends on features values. Alternatively, $sim(\cdot,\cdot)$ can be a CF similarity function based on users who purchased items with features f_1 and f_2. An example of a simple content-based similarity is $sim(f_1, f_2) = \delta(f_1, f_2)$ which equals 1 if $f_1 = f_2$ and 0 otherwise. An example of a CF similarity function is the cosine similarity based on the users of items with f_1 and f_2

We denote by $H_u = \{h_{u1}, h_{u2}, \dots, h_{un}\}$ the set of n items in user u's history. The set H_u is used by an implicit-feedback recommender to produce a set of k recommended items denoted by $R_k(H_u) = \{r_{u1}, r_{u2}, \dots, r_{uk}\}$. For the explicit-feedback case, H_u is also associated with numeric ratings.

The recommendation algorithm, $R_k(\cdot)$, is also pluggable to the framework. Different applications may benefit from different $R_k(\cdot)$'s and $sim(\cdot,\cdot)$'s. In Section 3, we discuss some of the concrete similarity measures and the recommender system we use.

2.1 Algorithms

We perform feature selection by computing a relevance score for each feature and then selecting the highest-scoring (i.e., most informative) features. Algorithm 1 and Algorithm 2 describe our algorithms for computing a relevance score for attributes and labels, respectively. Both algorithms are based on the ratio between two variables b_1 and b_2: b_1 is proportional to the similarity of the feature under consideration w.r.t. relevant items according to $R_k(\cdot)$. b_2 is a normalizer which is proportional to the similarity of the feature under consideration w.r.t. random items. The ratio b_1/b_2 measures the *normalized* relevance of the feature with regard to recommended items.

Algorithm 1 ScoreAttribute(Attribute a)

```
Take a sample of n seed user histories {H_1, H_2, ..., H_n};
for each user history H_u do
  Compute k recommended items R_k(H_u) = {r_u1, r_u2, ..., r_uk};
  for each pair (h_ui, r_uj) s.t. h_ui ∈ H_u, r_uj ∈ R_k(H_u) do
    b_1 = b_1 + sim(h_ui.a, r_uj.a);
    Pick a random item r_rand;
    b_2 = b_2 + sim(h_ui.a, r_rand.a);
  end for
end for
return b_1/b_2;
```

Algorithm 2 ScoreLabel(Label l)

```
Take a sample of n seed user histories {H_1, H_2, ..., H_n};
for each user history H_u do
  Compute k recommended items R_k(H_u) = {r_u1, r_u2, ..., r_uk};
  for each pair (h_ui, r_uj) s.t. h_ui ∈ H_u, r_uj ∈ R_k(H_u) do
    Let l_1 be the label in r_uj.L closest to l according to sim(·,·);
    b_1 = b_1 + sim(l, l_1);
    Pick a random item r_rand;
    Let l_2 be the label in r_rand closest to l according to sim(·,·);
    b_2 = b_2 + sim(l, l_2);
  end for
end for
return b_1/b_2;
```

Figure 1: Computing attribute and label scores

2.2 Generalizing Lift

Interestingly, our methods can be seen as a generalization of the lift measure commonly used outside the context of recommendation systems. For the particular case where $sim(v_1, v_2) = \delta(v_1, v_2)$, the parameter b_1 counts co-occurrences of the feature in user histories and in their relevant recommended items. The parameter b_2 counts co-occurrences of the feature in user histories and in random items. Let E_1 be the event where a recommended item r is a "good" recommendation to a user with history H_u (i.e., appears in the top-k recommendations). Let E_2 be the event where a recommended item r, not necessarily "good", has the same feature as an item in H_u. It can be easily shown that under this particular settings, ranking according to b_1/b_2 coincides with the empirical $lift(E_1 \Rightarrow E_2)$:

$$lift(E_1 \Rightarrow E_2) \stackrel{\text{def}}{=} \frac{P_r(E_2|E_1)}{P_r(E_2)} = \frac{b_1}{b_2}. \quad (1)$$

3. IMPLEMENTATION AND EVALUATION

Feature scoring for recommendation systems is not well studied. We thus evaluate the feature scoring algorithm using a novel cold item representation task for Matrix Factorization (MF) models. MF models represent items and users by trait vectors in a low dimensional latent space. The inner product between a user vector and an item vector is

	Author	Publisher	Year of Publication
Score	3.4	1.67	1.15
RMSE	0.98	1.72	2.19

Table 1: Reconstruction results for the books dataset.

proportional to the affinity between the user and the item [10]. We perform our evaluation as follows: After training a MF model we iteratively remove a random item vector from the model and attempt to reconstruct its trait vector based on other item vectors having a similar features (labels or attribute values). We repeat this reconstruction process for N items, each time reconstructing a different item vector.

Formally, let $q_i \in \mathbb{R}^D$ be an item vector of a removed item i from a trained MF model with dimensionality D. We measure the ability of each single feature f to reconstruct q_i based on other items which are similar to i according to the feature f. For each test item i, we find a set of items $S_f(i)$ whose f values are similar to that of item i's value for f. We then compute a reconstructed vector $\hat{q}_i^f$ for i according to $S_f(i)$ as follows:

$$\hat{q}_i^f = \frac{1}{|S_f(i)|} \sum_{j \in S_f(i)} q_j. \quad (2)$$

Finally, we measure the quality of a reconstruction according to the Root Mean Squared Error (RMSE) and score the features quality by averaging all reconstructed item vectors:

$$\text{RMSE}(f) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} ||q_i - \hat{q}_i^f||^2}. \quad (3)$$

where N is the number of reconstructed vectors and $|| \cdot ||^2$ denotes the $\mathcal{L}_2$ norm.

This reconstruction process is designed to cope with the cold-start problem for items. When a new item i is introduced into the catalog, we wish to construct a trait vector in order to integrate i into an existing model even before having any usage data for i. This process is successfully used in the Xbox movies recommender. We note that while this evaluation process is specific to the cold-start representation problem for items in a MF model, the presented feature scoring framework is general to any task and any recommendation algorithm.

3.1 Results

We evaluate with two datasets: The book-crossing dataset [18] and a dataset from the Xbox Movies recommender (smartphone apps are considered in Section 4). As explained, we are interested in implicit-feedback. We use the algorithm of [12] to learn trait vectors for items, as well as for ranking recommended items in Algorithms 1 and 2.

Books: Each book in [18] is associated with three attributes: *author*, *publisher* and *year*. For scoring attributes, we choose $sim(\cdot,\cdot)$ to be the cosine similarity between attribute values. For example, for *author*, $sim(v_1, v_2)$ is the cosine similarity on users who read author v_1 and author v_2. We reconstruct a sample of 500 books based on each of these attributes. Table 1 summarizes the reconstruction results. As intuitively expected, *author* has the highest score, followed by *publisher*, and then *year*. RMSE results coincide with these scores (i.e., higher RMSE for low scores).

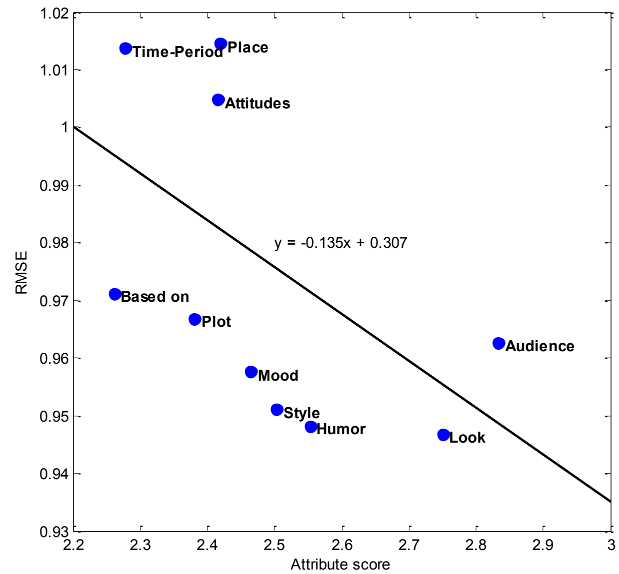

Figure 2: Xbox movies attributes vs. RMSE

Movies: Movies in the Xbox movies dataset are associated with labels. Each label is associated with a category, e.g., *Audience*, *Mood*, *Plot*. Every movie in the catalog has zero or more labels from each category. The *Audience* category has labels such as *Kids*, *Girls Night* and *Family*; The *Look* category has labels such as *3D*, *Black and White* and *Animation*; The *Time-Period* category has labels indicating the time in which the plot takes place. We use this dataset for evaluating both attribute and label ranking as follows:

Movies Attributes. We treat each of the label categories as a distinct attribute, and the unordered set of labels which belong to this category as the attribute value. Here we defined $sim(v_1, v_2)$ to be Jaccard similarity on labels belonging to movies v_1 and v_2. We then reconstruct a sample of 1,500 movies, and compute RMSE for each category. Figure 2 depicts the RMSE results vs. the attribute scores. As expected, categories like *Audience* and *Look* were found to be more informative than categories like *Time-Period*. Notice the negative correlation between the scores and RMSE.

Movies Labels. In this experiment, we evaluate our label scoring algorithm. Our experience shows that different labels from the same category may have a different informative value. For example, the *Place* category is in general non-informative, as for most movies it simply take the label *USA*. Nevertheless, for a small subset of movies, this category carries a more informative label such as *Ghetto* which highly predicts whom might watch the movie. Hence, we evaluate labels separately, ignoring categories. Here, we used $sim(i, j) = \delta(i, j)$. Figure 3 depicts the RMSE results vs. the label scores. Again, we see a clear trend line indicating a negative correlation between the scores and RMSE.

4. GENERATING FEATURES

Crafting a list of descriptive features and assigning them to catalog items is a labor-intensive process, typically performed manually by content specialists. Next, we demonstrate how to use the presented framework to automatically generate (and select) label features extracted from text descriptions of items. We model text descriptions using the

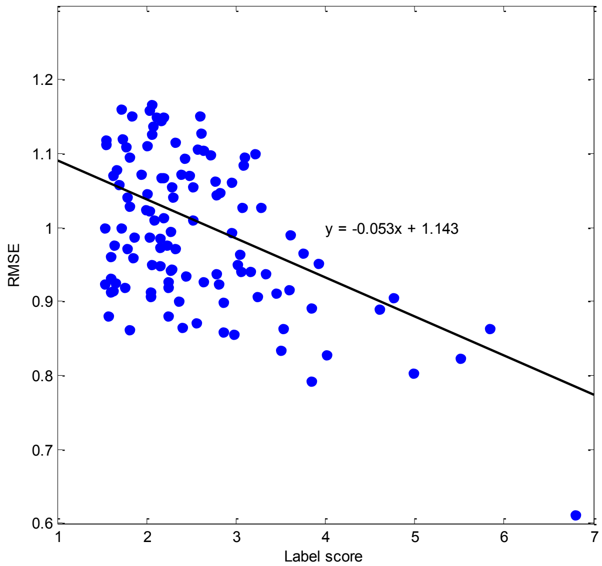

Figure 3: Xbox movies - labels vs. RMSE

Rank	Label feature	Score
1	enemy	2.342
2	ringtone	2.331
3	weapon	2.216
4	girl	2.191
5	youtube	2.183
6	child	2.182
7	kid	2.102
8	music	2.092
9	video	2.092
10	ball	2.086
· · · labels 11 through 1330 · · ·		
1331	version	1.158
1332	add	1.153
1333	time	1.147
1334	simple	1.147
1335	app	1.113

Table 2: Ranked labels from apps descriptions

binary bag of words model, and perform a simple filtering process, in which we remove stop words, words with very high inverse-document frequency, and words with very low frequency [2]. The remaining terms are normalized with a stemmer. We treat each word as a label and perform feature selection using Algorithm 2.

We present a ranked list labels from Windows Phone 8 apps descriptions[1]. After filtering, a total of 1335 labels were scored using Algorithm 2 with $sim(f_1, f_2) = \delta(f_1, f_2)$ and $R_k(\cdot)$ from [12]. Table 4 presents the 10 top ranking and the 5 low ranking labels. Observe that high ranking labels describe apps genres (*music*, *video*, and *ball*, which indicates a game) and demographic features (*kid*, *girl*), that are naturally informative for CF. Labels such as *app*, *version* and *time*, that clearly not informative, are ranked low.

[1]Avail. on `http://www.windowsphone.com/en-us/store`

5. CONCLUSIONS

Feature selection for recommendation systems is a relatively new field. This paper presents a generalized framework for selecting informative features for CF. The framework is successfully utilized to enhance recommendation in the Xbox Marketplace. We hope this work will inspire future research on feature selection for recommendation systems.

6. REFERENCES

[1] Fab: content-based, collaborative recommendation. M. Balabanovic and Y. Shoham. Comm. ACM, 1997.

[2] R. Baeza-Yates and B. Ribeiro-Neto. Modern Information Retrieval, 2011.

[3] Hybrid Recommender Systems: Survey and Experiments. Robin D. Burke. in User Model. User-Adapt. Interact.

[4] Combining content-based and collaborative filters in an online newspaper. M. Claypool and A. Gokhale, et al. Recommender Systems Workshop, 1999.

[5] Yahoo! Music Recommendations: Modeling Music Ratings with Temporal Dynamics and Item Taxonomy. G. Dror, N. Koenigstein, Y. Koren. RecSys 2011.

[6] The Yahoo! Music Dataset and KDD-Cup'11. G. Dror, N. Koenigstein, Y. Koren, M. Weimer. Journal Of Machine Learning Research, 2012.

[7] Map Based Visualization of Product Catalogs. M. Kagie, M. van Wezel and P.J.F. Groenen. Recommender Systems Handbook, 2011.

[8] The Xbox Recommender System. N. Koenigstein, N. Nice, U. Paquet, N. Schleyen. In RecSys 2012.

[9] Xbox Movies Recommendations: Variational Bayes Matrix Factorization with Embedded Feature Selection. N. Koenigstein, U. Paquet. In RecSys 2013.

[10] Matrix Factorization Techniques for Recommender Systems. Y. Koren, R. M. Bell, and C. Volinsky. IEEE Computer, 2009.

[11] Content-based Recommender Systems: State of the Art and Trends. P. Lops, M. de Gemmis, G. Semeraro. Recommender Systems Handbook, 2011.

[12] One-class Collaborative Filtering with Random Graphs. U. Paquet, N. Koenigstein. In WWW 2013, 999–1008.

[13] Sage - Recommender Engine as a Cloud Service. R. Ronen, N. Koenigstein, E. Ziklik, M. Sitruk, R. Yaari, N. Haiby-Weiss. In RecSys 2013.

[14] Automatic Feature Selection for Recommender Systems. R. Ronen, N. Koenigstein, E. Ziklik, N. Nice. Microsoft Patent 339085.01, 2013.

[15] J. B. Schafer, J. A. Konstan, J. Riedl. E-Commerce Recommendation Applications. Data Min. Knowl. Discov. 5(1/2): 115-153 (2001).

[16] Methods and metrics for cold-start recommendations. A. I. Schein, A. Popescul, L. H. Ungar and D. M. Pennock. SIGIR 2002.

[17] Designing and Evaluating Explanations for Recommender Systems. N. Tintarev and J. Masthoff. Recommender Systems Handbook, 2011.

[18] Improving recommendation lists through topic diversification. C. Ziegler, S. M. McNee, J. A. Konstan, G. Lausen. WWW 2005.