Recommending from Experience*

Francisco J. Peña, Derek Bridge
Insight Centre for Data Analytics
Department of Computer Science
University College Cork, Ireland
francisco.pena@insight-centre.org, derek.bridge@insight-centre.org

Abstract

In this paper we present RC, a context-driven recommender system that mines contextual information from user-generated reviews and makes recommendations based on the users' experiences. RC mines the contextual information from the user-generated reviews using a form of topic modeling. This means that, unlike other context-aware recommender systems, RC does not have a predefined set of contextual variables. After mining the contextual information, RC makes top-n recommendations using a Factorization Machine with the contextual topics as side information. Our experiments on two datasets of ratings and reviews show that RC has higher recall than a conventional recommender.

Introduction

Context can have a great influence on how a user perceives an item. For instance, a user may assign a 5-star rating to a hotel after staying there on a business trip, but might have given the same hotel just 2-stars if she visited that hotel on a family holiday, complaining about the small room size or the lack of a swimming pool for her children. Many recommender systems have focused on modeling user preferences, sometimes overlooking the influence that the context in which the product or service is consumed has on those preferences. In several domains, the growing availability of user-generated reviews means that we now have access to contextual information from the reviews themselves. In this paper, we seek to mine this contextual information and exploit it in a recommender system.

User-generated reviews often describe *experiences*, a business dinner or a honeymoon, for example. By definition, there is no such thing as a context-less experience. We either go to a hotel during Spring, Summer, Autumn or Winter, and we go alone or as a couple or with the whole family, etc. The context has an especial influence when we are evaluating products or services, such as hotel visits or restaurant meals, in which the most important thing is the experience and not the product itself. In this case, a user is likely to include a lot of the contextual information in the review, to qualify their opinion of the product or service.

*This work is supported by the Science Foundation Ireland(SFI) under Grant Number SFI/12/RC/2289.

Of course, we are not alone in seeking to use contextual information in recommender systems. There have been many context-aware recommender systems (CARS) (Adomavicius and Tuzhilin 2011). But, the majority of them, if not all of them, use a predefined, and typically small, set of contextual variables. They might have a temporal variable, with values based on seasons, days of the week or times of the day, or a variable for the purpose-of-visit, with values such as business or pleasure, for example.

Inspection of the experiences described in user-generated reviews shows that there is a much wider range of contextual factors than those most commonly captured in existing CARS. Indeed, contextual factors are open-ended and many of them are domain-specific. For example, a restaurant with no car parking may be evaluated differently by someone who drives herself to the restaurant from someone who takes a taxi; a hotel whose rooms are inadequately wheelchair accessible may be rated differently by someone whose companions includes a person who uses a wheelchair from someone whose companions do not. In these examples, other things being equal, differences in ratings occur because the user is rating her whole experience of the restaurant meal or hotel visit, factoring in the contextual information. Parking and wheelchair accessibility are just two of numerous contextual factors, which may be revealed in user reviews. Similarly, user reviews reveal that even well-known contextual variables such as purpose-of-visit tend to have a much wider range of values than is common in existing CARS.

In fact, the kind of recommender we seek to build might be better described as *context-driven* rather than *context-aware* (Pagano et al. 2016). Currently, CARS tend to make use of the user's current situation and historic behaviours of that user and similar users in similar situations. A context-driven recommender has a concern too for what the user is trying to accomplish (their *intent*), allowing them to move beyond their past preferences. These are the kind of contextual factors we expect to mine from user-generated reviews. Our recommender system is trained on these contextual factors as well as user preferences (ratings). It is invoked by supplying not just the active user and a candidate item or items, but also by supplying a vector of these contextual factors, representing the intent of the user.

The remainder of this paper is organized as follows. We

start with a discussion of related work. Then we present an overview of our proposed RC system, and give details of each of its components. Finally, we report the results of evaluating RC on two datasets.

Related Work

There are three main ways to build a CARS: using contextual information to pre-filter, to post-filter and including it in the model (Adomavicius and Tuzhilin 2011). Our work is a model-based CARS. Other examples of model-based CARS include approaches based on matrix factorization e.g. (Baltrunas, Ludwig, and Ricci 2011), tensor factorization e.g. (Karatzoglou et al. 2010), k-nearest neighbours e.g. (Zheng, Burke, and Mobasher 2013), and the SLIM algorithm (Zheng, Mobasher, and Burke 2014). In all cases, they use a small, predefined set of contextual variables each with a small set of values. By contrast, in our work we take a more open-ended view of context.

Equally, there has been a lot of work that has exploited user-generated reviews in recommender systems (Chen, Chen, and Wang 2015). Typically, it uses heuristics, e.g., for sentiment mining. But several use topic modeling, as we do, e.g. (McAuley and Leskovec 2013; Ling, Lyu, and King 2014; Diao et al. 2014). Our work differs in its focus on mining context from the specific review sentences.

Chen & Chen present work which, like ours, combines CARS with review mining (Chen and Chen 2015). They use heuristics to extract context-independent preferences but also context-dependent preferences and fuse the two in a linear-regression-based recommender. In a given review, they identify product aspects, given by nouns; opinions about those aspects, given by adjectives; and the contexts in which the opinions apply to the aspects. For hotels, they use just three contextual variables (companion, occasion, and time), each associated with a small set of predefined keywords. A contextual value may relate to an opinion about an aspect if the context appears in the same or a preceding sentence as the opinion. By contrast our work uses topic modeling and takes a more open-ended view of context.

Hariri et al. also present work that combines CARS with review mining (Hariri et al. 2011). Like RC, their system assumes that the user provides some sort of query. But instead of applying a topic model to the query, they use a multi-label text classifier to provide a probability distribution over a set of class labels (e.g. trip types). The classifier that they use is Labeled-LDA. In contrast to our work, this uses topic modeling in a supervised way and over a predefined set of topics (e.g. trip types). They use a nearest-neighbours classifier, with an extended way of computing similarity that takes the contextual class labels into account.

Finally, we have adopted the main ideas in (Bauman and Tuzhilin 2014)'s approach to context mining: classification into specific and generic; topic modeling on the specific text; and then a filter to retain topics that are more associated with specific text. But, following extensive experimentation, we have many differences too including: for the classification, we work at sentence-level rather than review-level, we use different features, we use a balanced training set, and we use a different classifier; for the topic mining, again we are working at sentence-level, we use NMF instead of LDA, we have placed emphasis on topic model stability by using an ensembled NMF approach, and in equations 1 and 2 we sum weights whereas they compute cardinalities. But the biggest difference is that Bauman & Tuzhiln do not 'close the loop': they do not actually use the topic models in a recommender; in their paper, it remains an aspiration. We have used the contextual topic vectors as side information in a context-driven recommender, implemented using FMs.

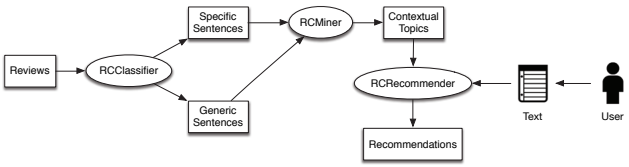

Figure 1: An overview of the RC system

The RC System

We have built a system which we call RC (standing for Rich Context), which is our first attempt at a context-sensitive recommender that uses open-ended contextual information that it mines from reviews that describe experiences. Our system has three main components called RCClassifier, RCMiner and RCRecommender. RCClassifier classifies review sentences into specific and generic; RCMiner builds a contextual topic model from the specific review sentences; and RCRecommender uses the topic model as side information in making context-driven recommendations. They are shown in overview in Figure 1.

We assume a dataset of user opinions for training the system. Each record in the dataset $\langle\langle u, i, R_{ui}\rangle, r_{ui}\rangle$ identifies a user $u \in U$ and an item (e.g. a hotel or restaurant) $i \in I$. It contains the user's review of the item, R_{ui}, and the user's rating of the item, r_{ui}.

Since the system is context-driven, it is invoked by identifying the active user u, a candidate item i or set of candidate items, and a contextual query Q. The latter is a short phrase, submitted by the user, which describes her intent, the context in which she intends to consume the recommended item, e.g. *"birthday dinner"*.

RCClassifier

The goal of the RCClassifier is to classify review sentences as either *specific* or *generic*. (This is similar to the work in (Bauman and Tuzhilin 2014), except that they classify whole reviews instead of individual sentences.) Sentences in which the author is describing (part of) an experience with a product or service are specific; for instance the second, third and fourth sentences of this review: *"During the summer, we like to take a mini staycation. This year it was extra special as we also got engaged. Our stay at the Biltmore was just fantastic. The service was exceptional, and the food was amazing"*. Generic sentences just give general opinions and do not describe an experience with the product or service; for

instance: *"Nice hotel, all the amenities you need, great complex of pools"*. Our assumption is that, since there is no such thing as a context-less experience, sentences that describe experiences, i.e., specific ones, will tend to have more contextual information than generic sentences.

RCClassifier starts by performing part-of-speech tagging on every sentence in every review in its training set. Each tagged sentence is then represented by a vector of numeric-valued features. We experimented with a wide set of features and ultimately adopted the following four, which we found to be predictive:

- *LogWords*: log of number of words in the sentence + 1
- *Vsum*: log of number of verbs in the sentence + 1
- *VBDSum*: log of number of verbs in the past tense in the sentence + 1
- *ProRatio*: ratio of log of number of personal pronouns + 1 to *LogWords*

It makes intuitive sense that the more words there are in a sentence the more likely it is that the sentence is specific: an experience is likely to be more verbose than a summary opinion. Similarly, it is intuitive that personal pronouns are commonly-used when relating an experience. Experiences are also more likely to use verbs in the past tense (e.g. "We *went* to the hotel on my birthday", "I *ordered* the cheese burger", "The service *was* amazing"), whereas generic sentences are more likely to use verbs in the present tense ("The hotel *is* very beautiful", "They *serve* great food here" (Bauman and Tuzhilin 2014). Finally, experiences involve events and actions and so are likely to contain more verbs.

Bauman & Tuzhilin use the first three of the above features, although they are classifying whole reviews, instead of individual sentences, so their features are computed per-review (Bauman and Tuzhilin 2014). They also used two other features: the log of the number of sentences in the review plus 1 (which is not applicable to our sentence-level classifier), and the ratio of *VBDSum* to *Vsum*, which we did not find to be so predictive on our datasets. We also experimented with features based on other parts of speech but did not find anything as predictive as the four above.

We manually label the sentences in a set of reviews to give a training set. We found that in our datasets there were more generic sentences than specific ones, giving an unbalanced training set. We trained classifiers on the unbalanced training set and compared their classification accuracy against the same classifiers trained on balanced versions of the training sets. For balancing the training sets, we tried the methods from the imbalanced-learn library (Lemaître, Nogueira, and Aridas 2016). We found SMOTE+ENN (Batista, Prati, and Monard 2004) to give the best results. As we will show later, the best classifier was a Random Forest classifier trained on a set that was balanced using SMOTE+ENN.

RCMiner

RCMiner's goal is to represent reviews by their contextual topics. The methodology for determining the contextual topics is inspired by (Bauman and Tuzhilin 2014):

- RCMiner *builds* a topic model from the specific review sentences only, as identified by RCClassifier. A topic model is essentially a set of K latent factors, each represented by a weighted list of terms. The way we build the topic model is detailed in later parts of this section. Since the topic model is built only from the specific review sentences, we expect some of these topics to be contextual. However, specific review sentences do not contain only contextual information so some of the topics may still be quite general. In the next three steps, the more general topics are discarded.
- RCMiner *applies* the topic model to *all* review sentences, so that each review sentence S is represented by a vector of weights T across the K topics.
- We normalise each vector T so that the sum of its weights equals 1.
- RCMiner next determines which topics appear more frequently in specific review sentences than in generic review sentences, on the assumption that these are more likely to be the contextual topics. For each topic, $t_k, k \in [1:K]$, we calculate the sum of the weights for that topic in the specific review sentences, divided by the number of specific review sentences:

$$w^s(t_k) = \frac{\sum_{S \in specific} T[k]}{|S : S \in specific|} \quad (1)$$

We make a similar calculation for generic reviews:

$$w^g(t_k) = \frac{\sum_{S \in generic} T[k]}{|S : S \in generic|} \quad (2)$$

Finally, the likely contextual topics, CT, are those where the ratio of the two proportions exceeds a threshold β:

$$CT = \{t_k \mid \frac{w^s(t_k)}{w^g(t_k)} > \beta\} \quad (3)$$

What is left to describe is the topic modeling itself (the first bullet point above). We *build* our topic models from just the nouns in the specific review sentences, as these are the parts of speech that most capture contextual information. Furthermore, to reduce costs, we *build* the topic models from just specific sentences that are the first sentence of their review. This is justified later in the experimental evaluation by noting that these sentences are the most likely to contain contextual information. We leave it to future work to exploit more of the sentences in building the topic models. (None of this changes the three other bullet points above.) Let the total number of nouns under consideration be N and let the number of specific sentences that are the first in their review be $|S^s|$.

In (Bauman and Tuzhilin 2014), Latent Dirichlet Allocation (LDA) (Blei, Ng, and Jordan 2003) is used for the topic modeling. We choose instead to use Non-Negative Matrix Factorization (NMF) (Lee and Seung 1999), which has been applied to topic modeling in, e.g., (Arora, Ge, and Moitra 2012). When using NMF for topic modeling, the $|S^s| \times N$ document-term matrix (in our case, $|S^s|$ specific review sentences and their N nouns) is approximated by two non-negative matrices, $\mathbf{W}$ and $\mathbf{H}$, where $\mathbf{W}$ is a $|S^s| \times K$ topic

membership matrix and $\mathbf{H}$ is a $N \times K$ term membership matrix for the K topics. As stated in (Belford, Namee, and Greene 2016), one advantage of using NMF over LDA is that it involves fewer parameter choices.

But, there is a significant *instability* problem with both LDA and NMF: different runs of these algorithms (with different initial random weights) can result in very different topic models (Belford, Namee, and Greene 2016). Belford et al. have recently proposed an *ensembling* technique that can much reduce the instability of NMF topic models, and we adopt their methods in RCMiner. In overview, they propose to produce a set of base topic models by multiple runs of NMF on the same original document-term matrix with different random initialisations. Due to the instability of NMF, this produces a diverse set of topic models. These diverse topic models are combined. This is done by stacking each of the $\mathbf{H}$ matrices: each row is a topic from one of the base models and each column is a term. Then, NMF is run again but on this combined matrix to produce the ensembled topic model. This model can have K' factors, which need not be the same as the K used in the base models. However, in their experiments and in ours $K' = K$. Belford et al. give further details and also demonstrate the improved stability of their approach (Belford, Namee, and Greene 2016). As explained in a later section, we have empirically verified that this produces much more stable models on our datasets.

At the end of this process, the first sentence of every review in the training set (irrespective of whether it is specific or generic) has been associated with a vector of length $|CT|$, $c_1 \ldots, c_{|CT|}$, whose values designate the affinity of the sentence to each of the $|CT|$ contextual topics. The original dataset of records $\langle\langle u, i, R_{ui}\rangle, r_{ui}\rangle$ is transformed to one in which reviews are represented by their corresponding contextual topic vectors, $\langle\langle u, i, c_1, \ldots, c_{|CT|}\rangle, r_{ui}\rangle$.

RCRecommender

The contextual topic vector can be viewed as a form of *side information* for a recommender system. There are many ways to incorporate side information into a recommender system (Shi, Larson, and Hanjalic 2014). We use Factorization Machines (FM) (Rendle 2012). For this, we one-hot encode the user and item ids, as is done in (Rendle 2012), giving training examples $\langle \mathbf{x}, r_{ui}\rangle$ where $\mathbf{x} \in \mathcal{R}^{|U|+|I|+|CT|}$. We use FMs of order 2, which attempt to model the interactions between each variable $x_j \in \mathbf{x}$ and the dependent variable r_{ui} but also the interactions between pairs of variables $x_j x_{j'}$ and the dependent variable r_{ui}. However, the interactions between pairs of variables is not modeled by one parameter per pair but by a low rank approximation, which makes the FM work well with the kind of sparse data we have in these domains.

As mentioned earlier, at recommendation time, after the FM has been trained, we assume we have an active user u, a candidate item i and a contextual query Q, the latter being a short phrase that expresses the context in which the user intends to consume the recommended item. We apply the topic model to Q, so that it too will be represented by a vector of contextual topics, $c_1, \ldots, c_{|CT|}$. For each candidate item, we use the FM to predict u's rating and we recommend the n items with highest predicted ratings.

Dataset	Reviews	Users	Items	Sparsity
Hotels	4098	3420	102	0.989
Restaurants	148721	35158	2557	0.998

Table 1: The datasets

Evaluation

We evaluated RC and its components on two datasets: the Yelp hotels dataset and Yelp restaurants dataset that were provided for the RecSys 2013 Challenge[1]. Both datasets contain records that identify a user, an item (i.e. hotel or restaurant), the user's review of the item and the user's rating of the item. We removed records that referred to items that had fewer than 10 reviews. Table 1 describes the datasets after these records were removed. We proceeded to evaluate RCClassifier for classification accuracy, RCMiner for topic stability, and RCRecommender for top-n recall.

RCClassifier

To evaluate RCClassifier, we randomly selected 300 reviews from each dataset. We manually labeled all sentences in these 300 reviews as *specific* or *generic*, giving us a ground-truth. For hotels, there were 3264 sentences, of which we labeled 1264 (39%) as specific and 2000 (61%) as generic; for restaurants, there were 2772 sentences, 1084 (40%) specific and 1645 (60%) generic. Training sets sampled from these labeled datasets were thus likely to be unbalanced. Preliminary experiments with different balancing methods (including none) and different classifiers found that balancing the datasets using the SMOTE+ENN method gave the best results, so we adopted this.

We then used nested 10-fold cross-validation to train and test different classification algorithms. The outer cross-validation estimates accuracy using hyperparameter values chosen by the inner cross-validation. In this way, we made sure that, for example, a k-nearest neighbours classifier had its accuracy estimated with a good value for k. We measured average classification accuracy, i.e. the proportion of the records in the test sets whose labels were correctly predicted by the classifier.

We experimented with different feature sets with the different classifiers. We found highest accuracy using the four features mentioned earlier (*LogWords*, *Vsum*, *VBDSum* and *ProRatio*) and a Random Forest classifier. Accuracy on the hotels dataset was 94%; for restaurants, it was 92%.

For evaluating RCMiner and RCClassifier, we trained a final RandomForest classifier on all 300 of the manually labeled reviews.

RCMiner

To help with the evaluation of RCMiner, we manually constructed two vocabularies of contextual words, comprising

[1] `https://www.kaggle.com/c/yelp-recsys-2013`

147 words for hotels and 108 for restaurants. We emphasize that these vocabularies are not used in building any of the models; they are used only to help confirm that RCMiner does build models that capture contextual information. We are not claiming that the vocabularies are some kind of complete ground truth; indeed, this would be at variance with our belief that contextual factors are open-ended. We are using the vocabularies simply as an indicative evaluation of the degree to which topics are contextual.

In these experiments, we need to build lots of topic models (on different splits of the datasets), but constructing topic models is expensive. We are using the ensembled NMF topic models described earlier. In each ensemble, we have 100 base models and each of these is trained over 200 iterations of NMF. These are then stacked for a final run of NMF for another 200 iterations. Because of the high costs, as mentioned earlier, these proof-of-concept experiments train the topic models on just the first sentence of each review. This is justified by noting that in the hotels dataset 28% of the first sentences of reviews contain words from our manually-constructed vocabulary of hotel contextual words, compared with just 10% of other sentences. In the restaurants dataset, the values are 19% and 10%, respectively. In future work, we will endeavour to use all sentences in our experiments.

Using grid search, we found highest top-10 recommender system recall using $K = K' = 78$ for hotels and $K = K' = 30$ for restaurants.

We scored different topic models according to the extent to which they included words from the manually-constructed vocabularies using the topic score ts of a topic t_k relative to a manually-constructed vocabulary V:

$$ts(t_k, V) = \sum_{w \in V} p(w|t_k) \tag{4}$$

where $p(w|t_k)$ is given by the topic model: the affinity of word w to topic t_k. Computing ts for the highest-weighted 5 words, LDA topic models had topic scores of around 18% for the hotels dataset and 11% for the restaurants dataset; ensembled NMF models had scores of 18% and 16% respectively. Informally, we found we had topics among whose top-ranked words were {*weekend*, *getaway*}, {*summer*, *family*, *kids*} and {*work*, *conference*} for hotels, and {*dinner*, *girlfriend*} and {*friends*, *Saturday*, *evening*} for restaurants.

We also measured topic model stability. We used the topic model agreement score, defined in (Greene, O'Callaghan, and Cunningham 2014), which rewards term overlap but also agreement on term ranking. For the values we have already given (78 topic for hotels, 30 topic for restaurants, 100 base models in the ensembles), the agreement scores for LDA were 0.28 for hotels and 0.21 for restaurants; for ensembled NMF, they were 0.95 and 0.74, respectively. This justifies the use of the ensembled approach.

RCRecommender

We start by describing the part of the RCRecommender evaluation methodology that is common to both datasets. We use the (now widely-accepted) methodology proposed in (Cremonesi, Koren, and Turrin 2010). We split the dataset randomly into training and probe sets. We build the topic model on the reviews from the training set and then prepare the training set by putting it into the format required by the Factorization Machine (see earlier section). From the probe set, we select records that have 5-star ratings to create a test set. For each of the test records, $\langle\langle u, i, R_{ui}\rangle, 5\rangle$, we find an additional C items that have not been rated by user u. This gives us $C + 1$ candidates (the test item and the C unrated items). We use the recommender system that we have built from the training set to predict the ratings of these $C + 1$ items. As mentioned previously, RC is a context-driven recommender, requiring the user to supply a contextual query Q, to which we also apply the topic model. In reality, a user will submit this query (a few words that describes the context in which she intends to consume the item). In these offline experiments, we need a way of obtaining these contextual queries. Hence, for Q, we simply use the nouns of the first sentence of the test item's review, R_{ui}. The same query is used for each of i's additional candidate items. Once it has predicted their ratings, RC recommends the top-n candidates with the highest predicted ratings ($n = 10$). If i (the test item, which the user had rated 5-stars) is among the top-10 recommendations, we count this as a hit. We measure the recall (hit rate): the proportion of items in the test set for which we get a hit. (In future work, we plan other ways of obtaining these queries, e.g. an ablation study and from human participants.)

We are using the LibFM implementation of FMs (Rendle 2012) with order 2 (i.e. it considers interactions between pairs of variables) but with different low rank approximations for the weights of the interactions.

Because of the very different sizes of the datasets, we have different methodologies for obtaining our results. In the case of the smaller hotels dataset, we use nested 10-fold cross-validation. The inner cross-validation mines topic models and builds FMs with different ranks (1, 2, 4, 8, 16, 32). It supplies the best of these to the outer cross-validation for recall estimation using the method described above. Because the dataset has a small number of hotels, the largest value of C we can use is $C = 90$: every user has 90 unrated hotels.

For the larger restaurants dataset, it is too computationally expensive to build so many topic models and to test so many test items. So, we split the dataset into two equal-sized parts and built an ensembled NMF topic model on the first half. We then used nested 10-fold cross-validation on the second half of the dataset to build FMs of different ranks (1, 2, 4, 8, 16, 32) and to compute recall, as described above. In the cross-validation, in order to reduce the testing times, we limit the size of the test set to 200 elements; these are chosen randomly from the 5-star ratings in the probe set. But in these experiments, the number of additional candidate items C for each test item is $C = 1000$, as proposed in (Cremonesi, Koren, and Turrin 2010).

We compare RC with an FM without contextual side information (using equation (41) from (Rendle 2012)). Its top-10 recall is computed using nested cross-validation, using 1, 2, 4, 8, 16 and 32 as different values for the number of latent factors, designated FM in Table 2.

Results in Table 2 show that there is an improvement by incorporating contextual information into the recommendation model in both datasets: an improvement of 3.5% for the

Dataset	FM	RC	Improvement
Hotels	0.4286	0.4436	+3.5%
Restaurants	0.0465	0.0500	+7.5%

Table 2: Top-10 recall results

hotels dataset and of 7.5% for the restaurants dataset.

Conclusions

In this paper we have presented RC, a context-driven recommender system that uses reviews and ratings to make recommendations. We have shown that we can outperform conventional systems by mining contextual information from the reviews. Our contributions are: an improvement in the task of classifying review sentences into specific and generic, the use of a stable ensembled topic modeling method to extract contextual information from user reviews without the need to predefine contextual variables and their values, and the use of this contextual information as side information in a Factorization Machine for recommendation.

In the future we will look specifically at cold-start issues since, as (Pagano et al. 2016) describes, these are significant for context-driven systems. We will also look at hierarchical topic modeling to give a better representation of contextual information, along with the incorporation of other forms of side information that we can mine from the reviews.

References

Adomavicius, G., and Tuzhilin, A. 2011. Context-aware recommender systems. In Ricci, F.; Rokach, L.; Shapira, B.; and Kantor, P. B., eds., *Recommender Systems Handbook*. Springer. 217–253.

Arora, S.; Ge, R.; and Moitra, A. 2012. Learning topic models – going beyond SVD. In *IEEE 53rd Annual Symposium on Foundations of Computer Science*, 1–10.

Baltrunas, L.; Ludwig, B.; and Ricci, F. 2011. Matrix factorization techniques for context aware recommendation. In *5th ACM Conference on Recommender Systems*, 301–304.

Batista, G. E. A. P. A.; Prati, R. C.; and Monard, M. C. 2004. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.* 6(1):20–29.

Bauman, K., and Tuzhilin, A. 2014. Discovering contextual information from user reviews for recommendation purposes. In *1st Workshop on New Trends in Content-based Recommender Systems at the 8th ACM Conference on Recommender Systems*, 2–9.

Belford, M.; Namee, B. M.; and Greene, D. 2016. Ensemble topic modeling via matrix factorization. In *24th Irish Conference on Artificial Intelligence and Cognitive Science*.

Blei, D. M.; Ng, A. Y.; and Jordan, M. I. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.* 3:993–1022.

Chen, G., and Chen, L. 2015. Augmenting service recommender systems by incorporating contextual opinions from user reviews. *User Modeling and User-Adapted Interaction* 25(3):295–329.

Chen, L.; Chen, G.; and Wang, F. 2015. Recommender systems based on user reviews: The state of the art. *User Modeling and User-Adapted Interaction* 25(2):99–154.

Cremonesi, P.; Koren, Y.; and Turrin, R. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *4th ACM Conference on Recommender Systems*, 39–46.

Diao, Q.; Qiu, M.; Wu, C.-Y.; Smola, A. J.; Jiang, J.; and Wang, C. 2014. Jointly modeling aspects, ratings and sentiments for movie recommendation (JMARS). In *20th ACM International Conference on Knowledge Discovery and Data Mining*, 193–202.

Greene, D.; O'Callaghan, D.; and Cunningham, P. 2014. How many topics? stability analysis for topic models. In *European Conference on Machine Learning and Knowledge Discovery in Databases*, 498–513.

Hariri, N.; Zheng, Y.; Mobasher, B.; and Burke, R. 2011. Context-aware recommendation based on review mining. In *9th Workshop on Intelligent Techniques for Web Personalization & Recommender Systems*, 30–36.

Karatzoglou, A.; Amatriain, X.; Baltrunas, L.; and Oliver, N. 2010. Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering. In *4th ACM Conference on Recommender Systems*, 79–86.

Lee, D. D., and Seung, S. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature* 401.

Lemaître, G.; Nogueira, F.; and Aridas, C. K. 2016. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *CoRR* abs/1609.06570.

Ling, G.; Lyu, M. R.; and King, I. 2014. Ratings meet reviews, a combined approach to recommend. In *8th ACM Conference on Recommender Systems*, 105–112.

McAuley, J., and Leskovec, J. 2013. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *7th ACM Conference on Recommender Systems*, 165–172.

Pagano, R.; Cremonesi, P.; Larson, M.; Hidasi, B.; Tikk, D.; Karatzoglou, A.; and Quadrana, M. 2016. The contextual turn: From context-aware to context-driven recommender systems. In *10th ACM Conference on Recommender Systems*, 249–252.

Rendle, S. 2012. Factorization machines with libfm. *ACM Trans. Intell. Syst. Technol.* 3(3):57:1–57:22.

Shi, Y.; Larson, M.; and Hanjalic, A. 2014. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Comput. Surv.* 47(1):3:1–3:45.

Zheng, Y.; Burke, R.; and Mobasher, B. 2013. Recommendation with differential context weighting. In *User Modeling, Adaptation, and Personalization*. Springer. 152–164.

Zheng, Y.; Mobasher, B.; and Burke, R. 2014. Cslim: Contextual slim recommendation algorithms. In *8th ACM Conference on Recommender Systems*, 301–304.