Rich-Context: An Unsupervised Context-Driven Recommender System Based On User Reviews

Francisco J. Peña
B.Eng., M.Sc.

NATIONAL UNIVERSITY OF IRELAND, CORK

Faculty of Science

Department of Computer Science

Thesis submitted for the degree of Doctor of Philosophy

April 2019

Head of Department:	Prof. Cormac J. Sreenan
Supervisors:	Dr. Derek Bridge

Research supported by SFI

Contents

List of Figures

List of Tables

I, Francisco J. Peña, certify that this thesis is my own work and I have not obtained a degree in this university or elsewhere on the basis of the work submitted in this thesis.

Francisco J. Peña

Para mis papás

Acknowledgements

This has been an amazing journey, full of ups and downs, a process of self-discovery and self-improvement. During these five years I discovered my love for research and cooking. I improved my research skills, built self-discipline, improved my tennis, fell in and out of love, made great friends and lost a few. I am a completely different person after my Ph.D. and I hope I can keep changing in the coming years.

First and above all, I would like to thank my supervisor Derek Bridge for his unconditional help throughout this amazing journey. His commitment to work and the effort that he puts into everything is inspiring. At times I felt he was too demanding, but now that I see what I have achieved, I can only say thanks for believing in me and pushing me to achieve things I would have never imagined.

Next I would like to thank my battle companions (or colleagues) Diego Carraro, Mesut Kaya and Rohit Dhamane. Ever since you arrived in the lab I felt I made way more progress and that is due not only to your constant feedback and the great discussions we had, but also because you made the workplace a happier place to be in.

I would also like to thank my family and friends back in Colombia for their constant support. My mom for giving me all the recipes and teaching me how to cook at age 25, and my dad for all his advice. I have never felt alone one day of my life and that support has taken me to achieve what I wanted the most.

I cannot forget to thank all of my friends in Cork for their support during tough times and the good times we had. The BBQs, the trips, the tennis games, and the Hispanic Society were all things that kept my spirits high and had a very positive impact on my life and my Ph.D.

Finally, to all the staff at UCC for their constant help. This work has been supported by a research grant by Science Foundation Ireland under grant number SFI/12/RC/2289 which is co-funded under the European Regional Development Fund.

Francisco J. Peña
Dublin, April 2019

Abstract

In the digital era, users have, more than at any point in history, a large amount of products or services to choose from. Recommender systems help to overcome this problem by suggesting which products or services to consume based on the users' past behavior along with additional information about users, products and services. For the most part, however, they have done this in a context-insensitive way. Yet it is clear that a knowledge of the context in which a user intends to consume a product or service is desirable to ensure that the recommended products and services match the intended context.

Context-Aware Recommender Systems incorporate contextual information in order to make recommendations that take into account both the users' preferences and his/her current contextual situation. However, there are problems in building such recommender systems: most of the time, contextual information is not available; when it is available, it is limited to a very small number of predefined variables; human intervention is required to define what contextual variables there will be; and many other possible contextual variables are left out. Given that users sometimes express their context while writing reviews about consumption of a product or service, user-generated reviews present themselves as a source to extract contextual information.

In this thesis, we research the problem of how to make contextual recommendations without the need to pre-define what context is. We mine the contextual information from user-generated reviews in an unsupervised way. We present Rich-Context, an unsupervised context-driven recommender system that extracts contextual information out of reviews in order to make recommendations. By using natural language processing techniques such as part-of-speech tagging, text classification and topic modeling, Rich-Context is able to successfully extract the contextual information out of the reviews.

Experimental results on multiple publicly-available, sparse, real-world datasets from different domains show that Rich-Context has better performance in both rating and ranking prediction tasks compared to several state-of-the-art algorithms, including six Context-Aware Recommender Systems, with the advantage that no contextual keywords or other variables need to be pre-defined. Additionally, since contextual recommendations are often cold-start recommendations, we performed experiments with users that had no previous ratings, again outperforming all of the state-of-the-art recommenders.

Chapter 1

Introduction

1.1 Motivation and Background

With the arrival of the digital era, users are faced with an ever greater choice of products and services. The number of products and services (items) in many cases exceeds the processing power of a person and the time spent evaluating every single option makes the selection task unfeasible. Recommender systems help users overcome this problem by gathering information from diverse data sources in order to suggest from among the available items those that match the users' preferences or goals.

Recommender systems are traditionally classified by the source of information they use. By this perspective, there are two main types of recommender systems: content-based and collaborative-filtering. Content-based systems use information that describes the items —both the candidate items and those items that the user likes— and creates recommendations based on comparing these descriptions. Different to content-based systems, collaborative-filtering systems do not in general use descriptions of the users or items, but instead they use information about the interaction between users and items. This interaction may be found explicitly in the form of ratings, likes/dislikes, etc. or implicitly in the form of clicks, plays, etc. Recommendations are created from regularities in the interaction data. It is often an advantage of collaborative-filtering approaches over content-based approaches that they do not require item descriptions since explicit item descriptions are not always available.

In this work we research how to make a recommender system that incorporates contextual information to satisfy the user's goals. We refer to context as *"The circumstances in which the user consumed or will consume the item and which can influence how the user perceives the item"*. Context might include, for example, the time of day, the weather, other people with whom the item was consumed, and so on. Since context can alter a user's perception of an item, knowing a user's preference in the form of user-item interactions is not enough to make accurate recommendations. It can be crucial to record and use the circumstances of that interaction when making recommendations if we hope to satisfy the user's goals.

Recommenders that incorporate contextual information into their models are called Context-Aware Recommender Systems (CARS). These recommenders gather contextual information in order to make recommendations that are suited not only to their users' tastes and preferences but also to their context. Variables such as time of the day and companions are considered in order to improve the recommendations. However, as pointed out by (Pagano et al. 2016), most CARS consider a user's current context when making a recommendation, whereas it often makes more sense to consider the user's future or desired context to align the recommendations not to the current context, but to the context in which the user is planning to be when consuming the recommended item. Pagano et al. refer to such recommenders as Context-Driven Recommender Systems (2016).

Furthermore, there are several problems with current CARS. One of the major problems is not about the models per se, but about the assumptions that they make regarding the data. Datasets that contain explicit contextual information are quite rare in the real-world and obtaining explicit contextual information through user studies can be quite expensive. In addition, systems that do collect explicit contextual information are usually limited to a small set of contextual dimensions —typically companion and purpose of the trip for hotel datasets— and those contextual dimensions usually contain a small set of values (e.g. the purpose might be business or leisure, and the companions might be solo, significant other or family for the mentioned hotel datasets).

The problem here is that in the real world, there are many more contextual situations that can influence our perception of a product or service. For instance, our perception of a restaurant might change if we intend to use it as a take-away rather than eat-in, we might prefer a certain hotel because it is pet-friendly, we prefer certain bars to watch a football game, and we would choose a restaurant that has access ramps when we go with a member of our family who has walking disabilities. By predefining and limiting the amount of contextual dimensions and values, we are in fact losing valuable information that can be used to make better predictions.

Another big problem with current CARS is that they are faced with the problem of sparsity. The larger the amount of contextual dimensions incorporated into a dataset, the more sparse it will be. In some cases, the contextual information instead of helping to make better predictions does the contrary, as there are not enough records in some specific contextual situations for the algorithms to make a correct inference.

On the other hand, in recent years, user-generated reviews have started to become available as an additional source of explicit interaction information besides ratings and likes/dislikes. They come in the form of unstructured text and have been proven useful to improve the prediction performance of recommender systems. The incorporation of user-generated reviews into recommender systems is not without challenge since the reviews are typically free text and therefore may contain a mix of useful and useless information. The challenge is then to correctly extract the useful information while discarding the rest.

Review-based recommender systems deal with the problem of exploiting the information contained in user-generated reviews in order to boost the recommendation performance. They mine useful information from reviews by using different natural language processing techniques ranging from heuristic approaches to stochastic models. By doing so, they are able to identify features such as the aspects that compose the items or the sentiment that users have towards the items.

We strive to produce a better recommender by unifying the two areas of research that deal with recommenders based on user reviews and CARS. Our goal is to design a recommender capable of making context-driven recommendations that does not require contextual keywords, is able to handle sparse datasets and can work well with real-world data. This goal can be accomplished by creating context-rich datasets through the exploitation of the information contained in the users' reviews. To achieve this we design and evaluate a new recommender system called **Rich-Context**.

1.2 Contributions

In this work we present three main contributions. We also list some more specific contributions derived from the main ones.

- *We design a recommender system capable of making context-driven recommendations without the need to pre-define what context is (Chapter 3):* The main contribution of our research is that, with the help of reviews generated by users, we are able to make context-driven recommendations without the use of pre-defined keywords or variables that define what context is. To the best of our knowledge, Rich-Context is the first recommender system able to do so. This has the advantage that no expert knowledge is required and could potentially be extended to other domains outside context-driven recommendations. Rich-Context also works well with sparse and large datasets, something that, as we will see in Chapter 2, has been a major problem for current CARS. Derived from this contribution there are two specific contributions:
  - *We present a new way to represent contextual information (Chapter 3):* Traditionally, contextual information has always been represented using variables with finite domains, often binary variables. By using topic models, we are able to represent reviews as vectors of topic assignments. By identifying which topics contain contextual information, we are able to relate reviews to certain contextual situations, but instead of using binary values we use real-valued weights in the range $[0, 1]$. This allows the possibility of assigning the importance of a contextual situation within a review.
  - *We show how to use contextual information as side-information using Factorization Machines to make recommendations (Chapter 3):* After evaluating several algorithms that support side-information besides the traditional user-item-rating triplets, we show that Factorization Machines (Rendle 2010) are ideal since they deal well with sparse datasets, have a running time linear in the number of contextual dimensions and can support real-valued side information for contextual dimensions, where several algorithms only support binary values (see Chapter 3).
- *We demonstrate better rating and ranking prediction performance than other*

state-of-the-art recommender systems, including several CARS (Chapter 6): Rich-Context outperforms several state-of-the-art recommenders including six CARS after testing it on multiple publicly available, real world datasets and using a publicly available third-party recommender systems evaluation tool. This brings transparency and confidence into our results. Derived from this contribution there are two more specific contributions:

  - *We design a new methodology for offline ranking evaluation of review-based context-driven recommender systems (Chapter 6)*: We designed a new offline methodology to evaluate the performance of context-driven recommender systems. This methodology is especially important as it helps us simulate the desired context of users, which is not available in offline experiments, therefore it is not mandatory to do surveys or online experiments (which are expensive) in order to estimate the performance of the recommender. In this methodology we simulate the user's context by taking one record composed of user-item-rating-review, transform the given review into a contextual vector, use that vector to predict the ratings for all of the unseen items for that user and then take the Top-N items.

  - *We demonstrate better performance than other state-of-the-art recommender systems for brand-new users (Chapter 6):* We also evaluated Rich-Context taking users from the test set who had zero ratings in the training set; we call them *brand-new users*. Brand-new users are common visitors of websites where absolutely no information is known about them. We show that, for this type of user, Rich-Context has superior performance than all of the evaluated state-of-the-art algorithms across all of the datasets.

- *We design a method for extracting contextual information from reviews in an unsupervised way (Chapters 3 and 5):* Based on a property of the reviews written by users (discussed in Chapters 3 and 4), we are able to extract contextual information out of the reviews using unsupervised learning with the help of topic modeling algorithms. Topic modeling helps us to unveil what is being discussed in the reviews and, by distinguishing between different types of reviews, we are able to differentiate between contextual information and non-contextual information. As mentioned before, this context extraction process is unsupervised and does not require expert knowledge. Derived from this contribution there are two more specific contributions:

- *We propose a methodology to select the best topic modeling algorithm to extract contextual information (Chapter 5):* Given the wide range of available topic modeling algorithms, it was necessary to select the algorithm that produced the best topic models that we would use to produce recommendations. We designed a methodology to follow in order to select not only the best algorithm, but also the data that should be given to it.

- *We introduce new metrics to measure the quality of topic models in terms of context-richness (Chapter 5):* In order to evaluate the quality of the contextual information that we extract from the reviews, we design metrics to measure the context-richness of a topic model.

- *We improved a methodology for classifying reviews into specific and generic (Chapter 4):* With the purpose of extracting the contextual information out of the reviews, we improve on the methodology proposed by (Bauman & Tuzhilin 2014) to distinguish between specific and generic reviews. By exploring new features, testing out several classification algorithms and applying resampling techniques we improve the performance of the review classifier.

We would also like to highlight the fact that there were no assumptions made regarding the data when evaluating Rich-Context. We did not use synthetic datasets or assume that structured contextual information was available or keywords were available to give such contextual information. We evaluated Rich-Context using multiple sparse, publicly available, real-world datasets from different domains (hotels and restaurants) and the evaluation was conducted using the third-party tool `RiVaL`[1] (Said & Bellogín 2014). This brings confidence and transparency to our results as the datasets can be used by someone else to compare their approaches against Rich-Context and, by using `RiVaL`, the evaluations will be conducted in the same way (using the same number of folds, splitting strategies, shuffling, etc).

We published two papers from the work reported in this dissertation. The first paper, titled "Recommending from Experience" (Peña & Bridge 2017), was published in the recommender systems track at the 30th Florida Artificial Intelligence Research Society Conference. The second paper, titled "Unsupervised

[1]`https://github.com/recommenders/rival`

Context-Driven Recommendations Based On User Reviews" (Peña 2017), was published in the Doctoral Symposium at the 11th ACM Conference on Recommender Systems.

1.3 Structure of the Thesis

The remainder of this thesis is organized as follows.

Chapter 2 gives an introduction to recommender systems and reviews current approaches to building recommender systems that can exploit user-generated reviews and also CARS. After reviewing the literature, common drawbacks are identified unveiling the gaps in which we can make contributions.

Chapter 3 presents an overview of how recommendations are produced by Rich-Context. We explain how, by distinguishing between two types of reviews, we are able to extract contextual information from them. We also explain how Rich-Context is able to produce recommendations using the contextual information.

In Chapter 4 we give a detailed explanation of how the reviews are separated using a classifier and the techniques we use to improve the classification accuracy. We show that by using Natural Language Processing techniques we are able to distinguish between the types of reviews.

Chapter 5 explains how using an unsupervised machine learning technique called topic modeling we were able to mine contextual information out of the reviews to be incorporated in the recommendation process. We show how by using certain types of words, better contextual information can be obtained and how the selection of the topic modeling algorithm is crucial to obtain reliable contextual information.

In Chapter 6 we evaluate Rich-Context and compare it against several state-of-the-art recommenders. We show that Rich-Context is able to outperform all of the other recommenders in multiple datasets for both rating prediction and ranking prediction tasks. We also evaluate Rich-Context in scenarios where there are no previous ratings available for users, in other words for users that are brand-new to the system. Under these conditions, again Rich-Context shows better performance than the other recommenders.

Finally, Chapter 7 concludes with a summary of the main contributions of this thesis and we discuss directions for future work.

Chapter 2

State of the Art

Recommender systems have come a long way since the early approaches that simply used a dataset of ratings to make predictions of a user's ratings for an unseen item. Developments stem from the fact that there are now richer datasets that include more information than just the ratings, and also the demand from users to create recommenders that are not only able to recommend new items, but also model their goals or temporal preferences.

The presence of websites such as Amazon.com, Yelp.com, TripAdvisor.com and others have enriched the ratings dataset with reviews. These reviews, made by consumers, often offer explanations of why a user gave a certain rating to an item. Including these reviews into the recommendation models can help boost the recommender accuracy because more information is available to model the users' preferences (Chen et al. 2015).

Equally, we have witnessed the arrival of recommender systems that model the user's current context in order to make recommendations that are not only tailored to the user but also to the situation surrounding the user. These type of recommenders, called Context-Aware Recommender Systems (CARS), have made it possible to improve the accuracy of certain recommender systems (Adomavicius & Tuzhilin 2015).

In this chapter we review the works related to context-driven recommender systems based on user reviews. We divide the works into three: CARS, unsupervised review-based approaches, and works that combine these two.

2.1 Context-Aware Recommender Systems

In many real-world situations, context can influence how a person perceives a product or service. For instance, it is important to know the purpose of travel at the time of recommending a hotel, as a good hotel for children or pets may be uncomfortable for someone who travels there to attend a conference. We can find similar examples in other types of businesses: a restaurant that turns out to be great to celebrate the birthday of a six year old can be a failure for a wedding anniversary or even more a wedding proposal. Similarly, a recommendation of a movie to watch with friends may be not the most appropriate to watch with the grandparents.

Defining what context is is not a simple task. The definition of context can vary depending on the discipline where it is used. For instance, (Bazire & Brézillon 2005) analyse 150 definitions of context across different fields. Nevertheless, with the goal of stating our definition of context, we explore the work of (Adomavicius & Tuzhilin 2015). Specifically, we briefly describe the classifications provided in (Adomavicius & Tuzhilin 2015) and then we frame our recommender system within one of the classifications they provide.

Adomavicius & Tuzhilin claim that there are two important aspects to context: what is known about the context (fully observable, partially observable or unobservable) and how the context changes over time (static or dynamic) (Adomavicius & Tuzhilin 2015). Based on these two aspects, a classification of CARS is offered in which there are four classes: representational approach (context: static, fully observable), incomplete-information approach (context: static, partially observable), latent approach (context: static, unobservable), dynamic approach (context: dynamic, various observability).

In our work, the knowledge that we have about the context is unobservable: it is not explicitly available but it is inferred from the reviews that users write about product or services. In our work, the context information is also static. Therefore, our recommender system can be classified as a latent approach. It is worth clarifying, as is done also in (Adomavicius & Tuzhilin 2015), that this is not to be confused with a latent factors approach such as matrix factorization. In a matrix factorization model, the latent factors are directly associated with each user and item, In our case however, contextual latent variables do not describe users or item.

One major advantage that latent approaches like ours have over representational approaches is that they do not assume that the context is represented by a *predefined* set of contextual attributes. By definition, havng predefined contextual attributes implies that the relevant contextual information needs to be identified and acquired well ahead of time in order to make recommendations (Adomavicius & Tuzhilin 2015). This represents a major problem with most representational approaches, since most real-world datasets do not come with contextual information. As we will see later in this Chapter, many representational approaches (Karatzoglou et al. 2010, Baltrunas et al. 2011, Unger et al. 2016, Zheng et al. 2013, Zheng, Mobasher & Burke 2014) try to circumvent this by using synthetically generated datasets, or data that comes from surveys or user studies, which can be quite different to real-world data.

Adomavicius & Tuzhilin note that, for each different domain, the relevance of the contextual variables changes and it is therefore necessary to have an expert who defines which variables are relevant (Adomavicius & Tuzhilin 2015). In our approach no expertise about the domain is necessary as the latent contextual variables are learned in an unsupervised fashion by the recommendation model — as we explain in Chapters 3 and 5.

For us, a definition of context within the framework of recommender systems is *"The circumstances in which the user consumed or will consume the item and which can influence how the user perceives the item"*. Many recommender systems have focused on making accurate recommendations based on data from users and items (Adomavicius & Tuzhilin 2015), ignoring contextual information. However, for some types of items, contextual information is of great value and incorporating it will lead to more accurate recommendations.

Traditional recommender systems use a function $Pred$ (often learned from data) to estimate ratings:

$$Pred : User \times Item \rightarrow Rating \tag{2.1}$$

where $Pred$ is a function that depends on a user and an item that has not been yet rated by that user. This is also known as the 2D approach to recommendations (Adomavicius & Tuzhilin 2015).

Ratings can also be modeled as a function of the known contextual information.

The rating-based representational approach to CARS is defined as:

$$Pred : User \times Item \times Context \rightarrow Rating \tag{2.2}$$

There can be more than one contextual dimension. For instance, when recommending a restaurant, we can have several types of contextual information, such as the time, companion and the purpose. So different recommendations will be obtained when the context is Sunday brunch with friends, compared to a weekday working lunch with colleagues.

$$Pred : User \times Item \times Time \times Companion \times Purpose \rightarrow Rating \tag{2.3}$$

We can generalize the above as:

$$Pred : User \times Item \times \mathcal{C}_1 \ldots \times \mathcal{C}_{|C|} \rightarrow Rating \tag{2.4}$$

where each $\mathcal{C}_i$ is a contextual dimension. In this case, we have $|C|$ contextual dimensions, thus having a $|C| + 2$ dimensional problem.

In what has become a widely accepted classification, Adomavicius & Tuzhilin divide CARS into three main approaches (2015): context pre-filtering, context post-filtering and context modeling. In context pre-filtering and context post-filtering, non-context aware recommender systems are used, but a pre- or post-filtering task is added in order to ensure that the recommended items match the desired context. In the contextual modeling approach, the context information is incorporated into the model. This of course has several advantages as the interactions between the contextual dimensions, the users and items are modeled with all of the information being used simultaneously.

In early work, most of the proposed recommenders took either a pre-filtering or post-filtering approach, but in recent years most of the research has turned into the contextual modeling approach. The problem with pre-filtering and pos-filtering approaches is that they actually ignore the contextual information when producing recommendations (Unger et al. 2016). In this way, those types of recommenders miss out on the interactions between users, items and context. The pre-filtering approach has additional problems such as the sparsity introduced by filtering out certain interactions from the dataset before training the model (Karatzoglou et al. 2010, Chen & Chen 2015).

Contextual modeling approaches directly incorporate the contextual information into the model (Unger et al. 2016). Our own work takes this approach. For this reason, in this review of the state-of-the-art, we focus on the contextual modeling approaches and classify them into three: matrix factorization, neighbourhood-based, and probabilistic approaches.

2.1.1 Matrix factorization approaches

Collaborative filtering methods try to predict the rating a user would give to an item based on the ratings of other users (Adomavicius & Tuzhilin 2005). They are able to produce recommendations without the need for information other than the ratings. This is especially useful when there is no other information available, such as item descriptions or user demographic information (Koren & Bell 2015). Collaborative filtering methods have proven to be a suitable alternative to content-based methods, and their good performance has established them as a state-of-the-art reference (Aggarwal 2016).

Matrix factorization is a very popular collaborative filtering method that uses dimensionality reduction techniques to fill the missing entries of a rating matrix $\mathbf{R}$. They do so by transforming both users and items to the same latent factor space of dimensionality l. The goal of this transformation is to model user-item interactions as an inner product in that factor space (Koren & Bell 2015).

In a basic matrix factorization model, the ratings matrix $\mathbf{R}^{|U|\times|I|}$, where U is the set of users and I is the set of items, is approximated by multiplying two factorized matrices $\mathbf{P}^{|U|\times l}$ and $\mathbf{Q}^{|I|\times l}$, such that:

$$\mathbf{R} \approx \mathbf{P}\mathbf{Q}^{\top} \tag{2.5}$$

The rating that user u would give to item i can be predicted then by taking u's latent features vector $\mathbf{p}_u$ and multiplying it by i's latent features vector $\mathbf{q}_i$, as follows:

$$\hat{r}_{u,i} = \mathbf{p}_u \mathbf{q}_i \tag{2.6}$$

The lower dimensional matrices are learned by optimizing a cost function. Several different optimization techniques and cost functions have been proposed.

But in this section we will explore some approaches that make use of matrix factorization in order to produce context-aware recommendations.

2.1.1.1 Multiverse Recommendation

Multiverse Recommendation is presented in (Karatzoglou et al. 2010).

Goal The goal is to find a straightforward way of integrating contextual information into a recommendation model.

Contribution This is achieved by using Tensor Factorization, a $|C|$-dimensional generalization of the bi-dimensional Matrix Factorization model. By adding regularization to the Tensor Factorization model, Karatzoglou et al. are able to adapt it to the collaborative filtering case and the model supports multiple contextual dimensions. Results using three datasets show that Multiverse Recommendation is able to beat a non-contextual Matrix Factorization Model.

Differences with other approaches Other approaches that use Matrix Factorization, such as (Xiong et al. 2010) and (Koren 2009), support only one contextual dimension. In the case of (Xiong et al. 2010) and (Koren 2009), that dimension is time. Multiverse Recommendation differs because it can support multiple contextual dimensions.

Notation The Multiverse Recommendation model uses the following notation:

Model In Multiverse Recommendation, a tensor is decomposed into a set of matrices and a small core tensor. Assuming for simplicity that there is only one contextual dimension, such as time, the ratings tensor is defined as $Y \in Y^{|U|\times|I|\times|K|}$, where $|U|$ is the number of users, $|I|$ is the number of items and $|K|$ is the number of conditions that the contextual dimension can have. For instance, we could have four conditions: “morning”, “noon”, “afternoon” and “night”. The ratings are given in a 5 star scale with $Y \in \{0, \dots, 5\}^{|U|\times|I|\times|K|}$, where 0 indicates that the user did not rate an item.

Table 2.1: Notation of Multiverse recommendation.

Symbol	Description
u	A user
i	An item
k	A contextual condition
$\hat{r}_{u,i,k}$	The predicted rating that user u would give to item i under the contextual condition k
U	A set of users
I	A set of items
K	A set of contextual conditions
Y	The user-item-context ratings tensor
$\mathcal{T}$	A latent features central tensor
$\mathbf{P}$	The latent features user matrix
$\mathbf{Q}$	The latent features item matrix
$\mathbf{S}$	The latent features context matrix
l_U	The number of latent factors of $\mathbf{P}$
l_I	The number of latent factors of $\mathbf{Q}$
l_K	The number of latent factors of $\mathbf{S}$

To factorize the tensor Y, a technique called High Order Singular Value Decomposition (HOSVD) (Lathauwer et al. 2000) is used. In HOSVD the tensor is decomposed into three matrices $\mathbf{P}^{|U|\times l_U}$, $\mathbf{Q}^{|I|\times l_I}$ and $\mathbf{S}^{|K|\times l_K}$ and a central tensor $\mathcal{T}^{l_U\times l_I\times l_K}$. In this case we could see the dimensionality variables l_U, l_I and l_K as the number of latent factors that we have in the conventional matrix factorization approach. The greater the number of conditions of the dimensionality variables, the more complex the model is going to be. If we want to predict the rating for item u by user i under context k, we could use the following equation:

$$\hat{r}_{u,i,k} = \mathcal{T} \times_{\mathbf{P}} \mathbf{P}_{u*} \times_{\mathbf{Q}} \mathbf{Q}_{i*} \times_{\mathbf{S}} \mathbf{S}_{k*}, \tag{2.7}$$

where $\times_{\mathbf{P}}$ is used to indicate that the tensor on the left is going to be multiplied from the $\mathbf{P}$ side.

Similarly to matrix factorization, there is an objective function composed of error and regularization terms. To learn the parameters, a stochastic gradient descent algorithm is used.

Evaluation Multiverse Recommendation was evaluated using three datasets, one semi-synthetic dataset and two others that were collected from surveys. Mean Absolute Error (MAE) was used as the evaluation metric. The approach is compared against a non-contextual matrix factorization model and against two other context-aware recommenders: an OLAP-based method (Adomavicius et al. 2005) and a method called item-splitting (Baltrunas & Ricci 2009). Results show that they are able to outperform all of the three methods across the three datasets in terms of MAE.

Summary The contribution of Multiverse Recommendation is that it is one of the first approaches to support any number of contextual dimensions and, as shown in (Karatzoglou et al. 2010), the model performs better than three other models in terms of MAE.

However, the support of any amount of contextual dimensions comes at a cost, since the number of parameters is exponential in the number of contextual dimensions. For one contextual dimension K_1 the dimension of the central tensor $\mathcal{T}$ is $l_U \times l_I \times l_{K_1}$, but if n contextual dimensions are included, then the dimension of $\mathcal{T}$ becomes $l_U \times l_I \times l_{K_1} \times \ldots \times l_{K_n}$. The number of parameters grows geometrically with each extra contextual dimension.

Another problem of this approach is that it requires that there be a discrete and indeed finite number of contextual conditions per contextual dimension. If contextual conditions were continuous, for example, then that dimension would in principle be infinite. This removes any possibility of storing numeric continuous values unless they are first transformed into categories. For instance, if we are dealing with temperatures such 5°C or 36°C, they would have to be transformed into categorical values such as *cold=true, warm=false* in the first case and *cold=false, warm=true* in the second case.

Finally, the use of a semi-synthetic dataset does not bring much confidence to the evaluation of Multiverse Recommendation. Although there are another two datasets used to test the approach, they are quite small (the largest one having around 6000 ratings and only two contextual dimensions). Testing Multiverse Recommendation in a real-world dataset would really improve the confidence in the results shown. But as we will see in much of the rest of this chapter, not testing the models on real-world datasets is a tradition with CARS. Most of the datasets are either semi-synthetically generated or are collected from surveys. It is very rare to find an approach that has been tested on the data of a real-world system. This is mainly because of the unavailability or real-world datasets that contain ready-to-use contextual information.

2.1.1.2 Context-Aware Matrix Factorization

Context-Aware Matrix Factorization (CAMF) is presented in (Baltrunas et al. 2011).

Goal The goal of CAMF is to extend Matrix Factorization in order to produce context-aware recommendations. Baltrunas et al. model the interaction of ratings with contextual dimensions by introducing additional model parameters.

Contribution CAMF is able to model the interaction of multiple contextual dimensions with ratings using a number of parameter that grows linearly with the number of contextual dimensions, resulting in a lower computational cost than Multiverse Recommendation. Tests on three semi-synthetic datasets show that the performance of CAMF is comparable to Multiverse Recommendation in terms of MAE. Tests on two other datasets, collected from surveys, show that CAMF has a better rating prediction performance than a context-less matrix factorization model.

Differences with other approaches As we saw earlier, the number of parameters in Multiverse Recommendation grows exponentially with the number of contextual dimensions. CAMF is able to have a comparable rating prediction performance in terms of MAE using fewer parameters. Compared to other CARS, this model also introduces different levels of granularity as we will see later on. This makes the model more flexible: it is easy to adjust it in case we want to reduce the number of parameters.

Notation The CAMF model uses the following notation:

Model In (Baltrunas et al. 2011), three CAMF models are introduced to incorporate context into recommender systems. The three models are based on Matrix Factorization. The difference between the three models is the granularity of the impact that the context has over the ratings.

The first model, called CAMF-C, includes a bias parameter per contextual condition that tells how each contextual condition influences the ratings so, assuming

Table 2.2: Notation of CAMF.

Symbol	**Description**
u	A user
i	An item
K	A set of contextual conditions
$\mathcal{C}$	A contextual dimension
c	The index of a contextual dimension
U	A set of users
I	A set of items
I_g	The set of item groups
C	A set of contextual dimensions
α	The offset or average rating
β_u	The bias of user u
β_i	The bias of item i
$\beta_{i,\mathcal{C}_c}$	The bias of item i under context $\mathcal{C}_c$
$\mathbf{p}_u$	The latent features vector of user u
$\mathbf{q}_i$	The latent features vector of item i
$\mathbf{c}$	A vector containing the contextual conditions for each context dimension
$\hat{r}_{u,i,\mathbf{c}}$	The predicted rating that user u would give to item i under the contextual conditions $\mathbf{c}$

that there are $|K|$ contextual conditions, we would have $|K|$ additional parameters. The second model, called CAMF-CI, adds a bias parameter that tells how each contextual condition influences the ratings of each item. This means that we are going to have $|K| \times |I|$ additional parameters, where $|I|$ is the number of items. The third model, called CAMF-CC, adds a bias parameter that tells how each contextual condition influences the ratings of each item category, i.e., we group the items into categories and have a different parameter for each category. This means that we are going to have $|K| \times |I_g|$ additional parameters, where $|I_g|$ is the number of item categories.

The general equation to predict a rating using any of the three mentioned models above is:

$$\hat{r}_{u,i,\mathbf{c}} = \alpha + \beta_u + \beta_i + \mathbf{p}_u \mathbf{q}_i^\top + \sum_{c=1}^{|C|} \beta_{i,\mathcal{C}_c} \tag{2.8}$$

where α is the offset parameter and β_u and β_i are the user and item biases, respectively. Here $\beta_{i,\mathcal{C}_c}$ is a variable that tells how much the contextual dimension $\mathcal{C}_c$ influences the rating of the item i. $\mathbf{c} = \langle \mathcal{C}_1, \ldots, \mathcal{C}_{|C|} \rangle$ is a vector that stores the contextual conditions for each available context dimension. For instance, if we have two contextual dimensions "Time of the day" and "Companion" a possible value for it is $\mathbf{c} = \langle \text{“}night\text{”}, \text{“}friends\text{”} \rangle$.

In CAMF-CI there is a parameter $\beta_{i,\mathcal{C}_c}$ for each contextual condition of each contextual dimension and item i combination. In the CAMF-CC model there is one parameter for each contextual condition and item category pair. That means that $\beta_{i,\mathcal{C}_c} = \beta_{i',\mathcal{C}_c}$ if items i and i' belong to the same category. In CAMF-C there is just one single parameter $\beta_{\mathcal{C}_c}$ for each contextual condition. In other words, we will have $\beta_{i,\mathcal{C}_c} = \beta_{i',\mathcal{C}_c}$ for every item i and i'. If a contextual condition is unknown, i.e., $\mathcal{C}_c = 0$ then the corresponding bias parameter $\beta_{i,\mathcal{C}_c}$ is 0.

Evaluation CAMF is tested on three datasets, one semi-synthetic dataset and two datasets collected from surveys. The rating prediction performance of the model is measured using MAE. Baltrunas et al. compare CAMF against Multiverse Recommendation using the semi-synthetic dataset and find a similar performance. Using the two datasets collected from surveys, they compare the model against Matrix Factorization and find that CAMF has a lower MAE.

Summary CAMF has the advantage that it can be trained in linear time with respect to the number of data points and contextual factors. For each contextual dimension in CAMF-CI, there is a $\beta_{i,\mathcal{C}_c}$, so for each contextual dimension there will be $|I| \times |\mathcal{C}_c|$ parameters, where $|\mathcal{C}_c|$ is the number of contextual conditions in the contextual dimension $\mathcal{C}_c$. In total, the CAMF-CI model will have $I \times (\mathcal{C}_1 + \cdots + \mathcal{C}_{|C|})$ parameters to learn. This is a great advantage over Multiverse Recommendation.

On the other side, CAMF-C assumes that context has an equal effect on all the items, which is not always true. The ratings of a ski resort can be heavily affected depending on the season, whereas the season could have little to no effect on the ratings of a business hotel. CAMF-CC groups items into categories and attempts to unveil the effect of the contextual dimensions on each category. Although this works much better than CAMF-C, it requires the intervention of an expert who is knowledgeable enough to create the right categories, grouping the items correctly. Finally, CAMF-CI has the highest level of granularity in which a parameter is learned to capture how each contextual dimension affects each item. But this model comes with the highest number of learned parameters.

2.1.1.3 Latent Context Matrix Factorization Recommendation

Latent Context Matrix Factorization Recommendation (LCMF) is introduced in (Unger et al. 2016).

Goal The main goal in (Unger et al. 2016) is to design a context-aware recommender system for smartphones that is able to collect all of the available information collected from the various smartphone sensors. Unger et al. combine this “latent” contextual information collected from the mobile devices with explicit contextual information in order to produce context-aware recommendations. Another goal they have is to reduce the feature space from the data that comes from the mobile devices, since the amount of features is quite high.

Contribution Unger et al. provide a hybrid model that mixes latent contextual features with explicit contextual features. They present two methods to reduce the dimensionality space from the information gathered by the mobile devices. They are able to outperform the rating prediction performance of CAMF in terms of Root Mean Squared Error (RMSE). In their tests, they show that a CARS that uses latent contextual dimensions or a mix of latent and explicit contextual dimensions is able to beat a CARS that only uses explicit contextual features.

Differences with other approaches The LCMF model extends from CAMF and it is very similar to it, with the difference that it introduces a set of bias parameters β_{i,ℓ_l} to model the influence that the latent contextual dimensions have over the item ratings. Their methodology also varies since the data is collected from mobile devices and then transformed in order to reduce the number of dimensions.

Notation The LCMF model uses the following notation:

Model LCMF extends the CAMF model by incorporating latent features extracted from mobile phone sensors. Unger et al. call the context dimensions that accompany the CAMF model *explicit-context* dimensions and the ones collected from the mobile phone sensors *latent-context* dimensions. They record visits of users to points-of-interest (POIs) with positive and negative feedback (like or dislike) and store information from the mobile phone sensors, such as GPS, accelerometer, microphone, light, gyroscope etc. They mix the latent-context dimensions with explicit-context dimensions such as the time of the day, whether it is weekend or weekday, and the weather conditions. In their user study, they collected approximately 520 latent-context dimensions per record.

Table 2.3: Notation of LCMF.

Symbol	Description
u	A user
i	An item
$\mathcal{C}$	A contextual dimension
c	The index of a contextual dimension
ℓ	A latent contextual dimension
l	The index of a latent contextual dimension
U	A set of users
I	A set of items
C	A set of contextual dimensions
L	A set of latent contexual dimensions
α	The offset or average rating
β_u	The bias of user u
β_i	The bias of item i
$\beta_{i,\mathcal{C}_c}$	The bias of item i under context $\mathcal{C}_c$
β_{i,ℓ_l}	The bias of item i under latent context ℓ_l
$\mathbf{p}_u$	The latent features vector of user u
$\mathbf{q}_i$	The latent features vector of item i
$\mathbf{c}$	A vector containing the contextual conditions for each context dimension
$\hat{r}_{u,i,\mathbf{c}}$	The predicted rating that user u would give to item i under the contextual conditions $\mathbf{c}$

To reduce the number of dimensions they use feature engineering techniques based on principal component analysis and deep learning. They extend the CAMF-CI model to include the latent contextual dimensions in the context vector $\mathbf{c} = \langle \mathcal{C}_1, \ldots, \mathcal{C}_{|C|}, \ell_1, \ldots, \ell_{|L|} \rangle$. In LCMF, $\mathbf{c}$ stores both the explicit contextual conditions and the latent contextual conditions for each available context dimension. A predicted rating for the LCMF model is calculated as follows:

$$\hat{r}_{u,i,\mathbf{c}} = \beta_u + \beta_i + \mathbf{p}_u \mathbf{q}_i^\top + \sum_{c=1}^{|C|} \beta_{i\mathcal{C}_c} + \sum_{l=1}^{|L|} \beta_{il} \ell_l \tag{2.9}$$

As seen in Equation 2.9, the model proposed by Unger et al. is based on Equation 2.8. But the model is different because, besides including a bias parameter $\beta_{i\mathcal{C}_c}$ for each explicit contextual condition, it also includes a bias parameter $\beta_{il}\ell_l$ for each latent contextual condition. This latent bias parameter is multiplied by ℓ_l, which is a real number in the range $[0, 1]$. ℓ_l represents the signal readings from the mobile phone sensors after a feature subset selection process.

Evaluation To evaluate the LCMF approach, Unger et al. developed a mobile phone application that makes recommendations to users based on explicit contextual information and contextual information collected from the mobile phone sensors (latent context). A study was made with 60 uses and 227 points-of-interest. Overall, 7416 feedback records were collected. LCMF was compared against CAMF-CI and a context-less matrix factorization model on both rating and ranking prediction tasks. To measure the rating prediction error, RMSE was used; for the ranking predictions, hit rate and Normalized Discounted Cumulative Gain (nDCG) were used. The hit rate is measured as the amount of times a recommendation is liked among the top-K recommendations presented to the user. nDCG is a weighted average of the position indexes among the top-K list when the weights are decreased as a function of the rank (position) of the recommendation. In nDCG a discount is applied to "hits" with low rankings (Unger et al. 2016).

Results show an overall better performance of LCMF across all of the metrics used (RMSE, hit rate and nDCG) compared to CAMF-CI and a context-less matrix factorization model. Surprisingly, in their results, the model that incorporates only the latent-context dimensions has better performance than the model that includes both the explicit and latent dimensions. Unger et al. explain that the explicit-context dimensions can cause over-fitting.

Summary LCMF inherits the advantages from CAMF, having linear time complexity. It also adds the advantage of supporting continuous values for context dimensions, not only categorical ones, such as the ones given by the ℓ_l variable.

Although the results from (Unger et al. 2016) are promising, they use only one dataset for their results, which reduces the reliability of the evaluation.

2.1.2 Neighbourhood-based approaches

User-based neighbourhood approaches are, like matrix factorization, a type of collaborative filtering method. They use a heuristic that makes predictions for one user based on the items that other users have rated previously. The value of an unknown rating $\hat{r}_{u,i}$ can be calculated by aggregating the ratings of other users (usually, the most similar other users) for the same item i (Adomavicius & Tuzhilin 2005):

$$\hat{r}_{u,i} = \operatorname*{aggr}_{u' \in \hat{U}} r_{u'i}, \tag{2.10}$$

where $\hat{U}$ is the set of neighbours of u. A very simple example of the aggregation function is:

$$\hat{r}_{u,i} = \frac{1}{|\hat{U}|} \sum_{u' \in \hat{U}} r_{u'i} \tag{2.11}$$

In general, latent factor models like Matrix Factorization tend to have better prediction performance than neighbourhood-based models due to their ability to link various aspects of the data (Adomavicius & Tuzhilin 2015). However, neighbourhood models are still used because of their simplicity, efficiency and the fact that they can naturally express explanations about why an item is being recommended (Desrosiers & Karypis 2011). In this section we will explore the neighbourhood based approaches that have been proposed with the goal of making context-aware recommendations.

2.1.2.1 Differential Context Relaxation

Differential Context Relaxation (DCR) is presented in (Zheng et al. 2012*a*).

Goal The goal of DCR is to create a CARS that identifies the relevant contextual variables from a context-rich dataset, to reduce the dimensionality of the problem, and alleviate the sparsity problem.

Contribution Given that having a lot of contextual variables increases the dimensionality of the problem and reduces the accuracy of the recommendations, DCR introduces the concept of context relaxations, which reduce dimensionality and increase rating prediction accuracy (Zheng et al. 2012*a*).

Differences with other approaches DCR is presented as a memory-based collaborative-filtering alternative to the latent factor models that we have explored so far. Different from Multiverse Recommendation, CAMF and LCMF, DCR uses previous ratings that happened under a similar or equal context to make recommendations, whereas other approaches are limited to using only the exact context. DCR can be classified as a hybrid between pre-filtering and contextual modeling since they apply a filtering part in the selection of neighbours and also use the context similarity to calculate the predicted rating under a target context.

Notation The DCR model uses the following notation:

Table 2.4: Notation of DCR.

Symbol	Description
u	A user
i	An item
$\hat{U}$	The set of neighbours of a user
$\mathbf{c}$	A vector containing the contextual conditions for each context dimension
$\boldsymbol{\mu}$	A context relaxation

Model DCR is a user-based collaborative filtering model. The main idea is to predict the rating $\hat{r}_{u,i}$ that user u would give to item i based on the ratings that other users, similar to u, have given to item i. The $\mathcal{K}$ most similar users to user u are called the neighbours of u and are denoted by $\hat{U}$. To make a prediction, the following formula is used:

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum\limits_{u' \in \hat{U}} (r_{u',i} - \bar{r}_{u'}) \times usim(u,u')}{\sum\limits_{u' \in \hat{U}} usim(u,u')} \tag{2.12}$$

where $usim(u,u')$ is a function that calculates the similarity between users u and u'. The similarity is calculated using the Pearson correlation coefficient (Zheng et al. 2012*a*).

DCR divides the recommendation algorithm into components, allowing each part to treat the context differently. In each component, the best set of contextual dimensions to be used are selected through an optimization procedure.

One way to make the recommender from Equation 2.12 context-aware is to select only neighbours of u that have rated items under the same target context $\mathbf{c}$, thus having a neighbourhood $\hat{U}_{\mathbf{c}}$. We could also include only ratings that were made under context $\mathbf{c}$ to calculate the user baseline $\bar{r}_{\mathbf{c}}$. Finally, Equation 2.12 can be modified to include only the ratings that were made for item i by the neighbours under the target context $\mathbf{c}$, denoted $r_{u',i,\mathbf{c}}$, and the baseline for the neighbour $\bar{r}_{u',\mathbf{c}}$ can be calculated using only the target context $\mathbf{c}$. Once we have made all these modifications, the context-aware user-based collaborative filtering recommender would look like this:

$$\hat{r}_{u,i,\mathbf{c}} = \bar{r}_{u,\mathbf{c}} + \frac{\sum\limits_{u' \in \hat{U}_{\mathbf{c}}} (r_{u',i,\mathbf{c}} - \bar{r}_{u',\mathbf{c}}) \times usim(u,u')}{\sum\limits_{u' \in \hat{U}_{\mathbf{c}}} usim(u,u')} \tag{2.13}$$

The problem with the above equation is that if we limit the ratings to have exactly the same context as $\mathbf{c}$, then it could happen that in many cases we have very few or no ratings from which to make the predictions. This is problematic because in many cases we will not be able to produce recommendations and the lack of ratings can also reduce the accuracy of the recommender. For instance, if we have three contextual dimensions that are "Time", "Companion" and "Day" and we want restaurants recommendations for $\mathbf{c}_1 = \langle \text{``}Night\text{''}, \text{``}Friends\text{''}, \text{``}Saturday\text{''} \rangle$, only ratings made under that exact situation will be considered.

The main idea behind DCR is to relax the strictness of the above approach in order to also include ratings that were made under a similar context. To achieve this, Zheng et al. introduce the concept of context relaxations, that are represented by $\boldsymbol{\mu}$. Following the above example, we could introduce a relaxation for each the contextual dimensions, like this: $\boldsymbol{\mu}_x = \langle \texttt{(any time)}, \texttt{(exact companion)}, \texttt{(contained weekday/weekend day)} \rangle$. In this case, when we are looking for ratings that were made under similar contexts, we can ignore the "Time" dimension, we need to find exactly the same kind of companion, and the day must belong to the same weekday/weekend group as our target context. For instance, we could include ratings that were made under $\mathbf{c}_2 = \langle \text{``}Afternoon\text{''}, \text{``}Friends\text{''}, \text{``}Friday\text{''} \rangle$ because this $\mathbf{c}_2$ is similar to $\mathbf{c}_1$ under context relaxation $\boldsymbol{\mu}_x$. Ratings made under the context $\mathbf{c}_3 = \langle \text{``}Night\text{''}, \text{``}Husband\text{''}, \text{``}Saturday\text{''} \rangle$ will not be considered because $\mathbf{c}_3$ is not similar to $\mathbf{c}_1$ under context relaxation $\boldsymbol{\mu}_x$.

In DCR, the formulae to calculate the predicted rating are divided into three, which are neighbourhood selection, neighbourhood contribution and user baseline, like this:

$$\hat{r}_{u,i,\mathbf{c}} = \underbrace{\bar{r}_{u,\boldsymbol{\mu}_3}}_{\text{user baseline}} + \frac{\overbrace{\sum_{u' \in \hat{U}_{\boldsymbol{\mu}_1}}}^{\text{neigbourhood selection}} \overbrace{(r_{u',i,\boldsymbol{\mu}_2} - \bar{r}_{u',\boldsymbol{\mu}_2})}^{\text{neighbourhood contribution}} \times usim(u,u')}{\underbrace{\sum_{u' \in \hat{U}_{\boldsymbol{\mu}_1}}}_{\text{neigbourhood selection}} usim(u,u')} \tag{2.14}$$

Here, for each of the three components, there is a different context relaxation variable. $\boldsymbol{\mu}_1$ is used for the neighbourhood selection, $\boldsymbol{\mu}_2$ for the neighbourhood contribution and $\boldsymbol{\mu}_3$ is used for calculating the user baseline.

In the **neighbourhood selection** component of the original k-Nearest Neighbour (k-NN) algorithm, the neighborhood is composed of the $\mathcal{K}$ most similar users to u. In DCR, the neighborhood is composed of the most similar users to u who have rated item i in a context matching $\mathbf{c}$ under the relaxation $\boldsymbol{\mu}_1$.

In the **neighbourhood contribution** component, the average rating of $\bar{r}_{u'}$ is replaced by one where only the ratings that were given under the relaxation $\boldsymbol{\mu}_2$ are considered. The result is denoted by $\bar{r}_{u',\boldsymbol{\mu}_2}$ and is subtracted from $r_{u',\boldsymbol{\mu}_2}$ to calculate the contribution of user u' towards the prediction.

The calculation of the **user baseline**, $\bar{r}_u$, is the same but taking the relaxation $\boldsymbol{\mu}_3$ instead of $\boldsymbol{\mu}_2$. The result is denoted by $\bar{r}_{u,\boldsymbol{\mu}_3}$.

The goal is to find the set of the most influential contextual dimensions in order to make accurate context-aware recommendations. This is achieved through finding the best choices for context relaxations in each of the three components. If the relaxations are too strict then a sparsity problem arises and if they are too loose then the influence of contextual dimensions is lost (Zheng et al. 2012*a*).

In their original proposal of DCR (Zheng et al. 2012*a*), an exhaustive search strategy was used to find the best combination of the contextual relaxation variables ($\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$ and $\boldsymbol{\mu}_3$). This was possible for the particular dataset they were using but likely to be impractical in general. Later, in (Zheng et al. 2012*b*), they used particle swarm optimization to learn the best values for the contextual relaxation variables.

Evaluation DCR is evaluated in (Zheng et al. 2012*a*) using a dataset of hotels in the largest 120 cities in the USA crawled from TripAdvisor. The dataset contains 2565 users, 1455 hotels and a total of 9251 ratings. This is the only time we found a CARS that was evaluated in a real-world dataset (a dataset that comes from an existing real-world platform and was not collected from a survey or generated artificially). DCR was compared against a context-less user-based k-NN approach like the one in Equation 2.12 and against a k-NN using contextual pre-filtering. Experiments showed that DCR has superior rating prediction performance in terms of RMSE. However, coverage is heavily affected when using DCR going from 8% when using a context-less k-NN to 3% when using DCR.

In (Zheng et al. 2012*b*), DCR is evaluated again using another dataset collected from a survey. This dataset contains 212 users that rated 20 food menus with a total of 6360 ratings. Note that compared to the TripAdvisor dataset, this one is not sparse. This time, as is natural with a very dense dataset, the coverage increases and DCR again outperforms a context-less user-based k-NN and a k-NN context pre-filtering in terms of RMSE.

Summary DCR is able to reduce the dimensionality of a context-aware recommendation problem by introducing context relaxations. This helps to alleviate the sparsity problem and translates into a higher coverage compared to a strict context k-NN approach. It also produces more accurate rating predictions.

However, DCR has a few drawbacks. First of all, it has a reduced coverage compared to a context-less k-NN. This can be clearly seen in sparse datasets, as shown in the results presented in (Zheng et al. 2012*a*). As stated in (Zheng et al. 2013), DCR suffers from the sparsity problem when used in datasets where the context information is not very dense. Also, since the algorithm components are dependent, there is no guarantee that selection of neighbours under constraint $\boldsymbol{\mu}_1$ is going to bring ratings that satisfy constraint $\boldsymbol{\mu}_2$ used to calculate the neighbour contribution. When this happens, the algorithm is not able to make a prediction (at least a personalized one) having a negative impact on the coverage. Furthermore, the search space is exponential to the number of contextual dimensions (Zheng et al. 2012*b*), which brings doubt on how scalable is the algorithm, and if it would work on a big dataset with a high number of contextual dimensions. Zheng et al. also fail to compare DCR against Matrix Factorization, which has been proven to have better performance than k-NN (Koren & Bell 2015).

2.1.2.2 Differential Context Weighting

Differential Context Weighting (DCW) is presented in (Zheng et al. 2013).

Goal DCW is a generalization of DCR that aims to fix the sparsity problem present in DCR and at the same time improve the recommendation accuracy by reducing the dimensionality of the contextual recommendations problem.

Contribution DCW is able to produce more accurate recommendations without the loss of coverage compared to other neighbourhood-based approaches.

Differences with other approaches Instead of selecting a context in the way that DCR does, DCW assigns a weight to each contextual dimension. DCW can be classified as a hybrid between pre-filtering and contextual modeling since it applies a filtering part in the selection of neighbours and also uses the context similarity to weight the ratings of neighbours.

Notation The DCW model uses the following notation:

Table 2.5: Notation of DCW.

Symbol	**Description**
u	A user
i	An item
$r_{u,i}$	The rating that user u gave to item i
$r_{u,i,\mathbf{c}}$	The rating that user u gave to item i under the contextual conditions $\mathbf{c}$
$\hat{U}$	The set of neighbours of a user
I_u	The set of all items rated by user u
$\mathbf{c}$	A vector containing the contextual conditions for each context dimension
$\boldsymbol{\sigma}$	A context weighting vector
ϵ	A context similarity threshold
ρ	An average rating weighted by context similarity
$\hat{r}_{u,i,\mathbf{c}}$	The predicted rating that user u would give to item i under the contextual conditions $\mathbf{c}$

Model DCW introduces a context weighting vector $\boldsymbol{\sigma} = \{\boldsymbol{\sigma}_c \in \mathbb{R} \mid 0 \leq \boldsymbol{\sigma}_c \leq 1\}$ to assign a weight between 0 and 1 to each contextual dimension c. The weights control the contribution of each contextual dimension to the prediction model. For instance, we can have $\boldsymbol{\sigma} = \langle Time = 0.0, Companion = 0.9, Day = 0.4\rangle$, meaning that, when we are going to make contextual recommendations, the "Time" dimension will be completely ignored and the "Companion" dimension will be almost twice as important as the "Day" dimension.

Similarly to DCR, DCW divides the recommendations algorithm into components. In particular, for DCW, the algorithm is divided into four components, which are the *neighbourhood selection*, the *neighbour contribution*, the *user baseline* and the *user similarity*, as can be seen in Equation 2.19.

Differently from DCR, in DCW the ratings are not filtered out, but instead a score is assigned to the ratings depending on their context. In this way, the more similar the context of a rating to the context of the prediction, the more it is going to contribute. Based on $\boldsymbol{\sigma}$, similarity between two contexts $\mathbf{c}$ and $\mathbf{c}'$ is calculated like this:

$$csim(\mathbf{c}, \mathbf{c}', \boldsymbol{\sigma}) = \frac{\sum\limits_{c \in \mathbf{c} \cap \mathbf{c}'} \boldsymbol{\sigma}_c}{\sum\limits_{c \in \mathbf{c} \cup \mathbf{c}'} \boldsymbol{\sigma}_c} \tag{2.15}$$

As we will see later, the $csim$ function is used in several places in the rating prediction task. It is used to calculate the set of neighbours of a user $\hat{U}$, to calculate the similarity between two users u and u', and to calculate how much a neighbour contributes towards the rating prediction.

To avoid including ratings that were made under a context very different from $\mathbf{c}$, Zheng et al. introduce four variables $\epsilon_1, \ldots, \epsilon_4$ (one for each component) that act as a minimum threshold. The goal of these variables is to remove all ratings that have been made under contexts that are too different to the target context $\mathbf{c}$, i.e., $csim(\mathbf{c}, \mathbf{c}', \boldsymbol{\sigma}) < \epsilon$. They explain that after running experiments, it was found out that ratings made under very different context have a negative impact on the rating prediction accuracy.

To do the **neighbourhood selection**, it has to be considered that users may have rated in multiple contexts, for that reason, the maximally-similar context is chosen and then the $\boldsymbol{\sigma}_1$ threshold is applied:

$$\hat{U}_{u,\boldsymbol{\sigma},\epsilon_1} = \{u' : \max_{r_{u',i,\mathbf{c}'}}(csim(\mathbf{c}, \mathbf{c}', \boldsymbol{\sigma})) > \epsilon_1\} \tag{2.16}$$

Note that only neighbours who have rated items under a context $\mathbf{c}'$ whose similarity to the target context $\mathbf{c}$ exceeds ϵ_1 are considered.

To calculate the **neighbour contribution** each rating is going to be weighted. The weighting is done by multiplying the rating times the similarity $csim(\mathbf{c}, \mathbf{c}', \boldsymbol{\sigma})$ divided by all the similarities of the other ratings. The weighted average is calculated like this:

$$\rho_{u',i,\boldsymbol{\sigma},\epsilon_2} = \frac{\sum\limits_{r_{u',i,\mathbf{c}'}:csim(\mathbf{c},\mathbf{c}',\boldsymbol{\sigma})>\epsilon_2} r_{u',i,\mathbf{c}'} \times csim(\mathbf{c}, \mathbf{c}', \boldsymbol{\sigma})}{\sum\limits_{r_{u',i,\mathbf{c}'}:csim(\mathbf{c},\mathbf{c}',\boldsymbol{\sigma})>\epsilon_2} csim(\mathbf{c}, \mathbf{c}', \boldsymbol{\sigma})} \tag{2.17}$$

Note that Equation 2.17 is necessary only when there are multiple ratings by the same user to the same item under different contextual situations. If there is only one rating for each user-item pair, then $\rho_{u',i,\boldsymbol{\sigma},\epsilon_2} = r_{u',i,\mathbf{c}'} \times csim(\mathbf{c}, \mathbf{c}', \boldsymbol{\sigma})$.

Only ratings whose context similarity is greater than ϵ_2 are included.

To calculate the average rating of a user, the set of all items rated by user u, which we will denote as I_u, is used. The overall average across all items rated in a similar context is $\bar{\rho}$ and is calculated in this way:

$$\bar{\rho}_{u',\boldsymbol{\sigma},\epsilon_2} = \frac{\sum\limits_{i \in I_{u'}} \rho_{u',i,\boldsymbol{\sigma},\epsilon_2}}{|I_{u'}|} \tag{2.18}$$

The **user baseline** is the average of all the ratings made by user u in similar contexts, determined by ϵ_3. The user baseline is expressed as $\bar{\rho}_{u,\boldsymbol{\sigma},\epsilon_3}$.

To calculate the **user similarity** Zheng et al. weight each comparison between ratings. They create a set Λ_{ϵ_4} that contains all the items i and pairs of context $\mathbf{c}$ and $\mathbf{c}'$ for the users u and u' respectively, such that user u has rated i under context $\mathbf{c}$, and user u' has rated i under context $\mathbf{c}'$ and $csim(\mathbf{c}, \mathbf{c}', \boldsymbol{\sigma}) > \epsilon_4$.

$$\Lambda_{\epsilon_4} = \{\langle i, \mathbf{c}, \mathbf{c}' \rangle : \exists r_{u,i,\mathbf{c}}, r_{u',i,\mathbf{c}'} \wedge csim(\mathbf{c}, \mathbf{c}', \boldsymbol{\sigma}) > \epsilon_4\} \tag{2.19}$$

Once all the ratings and relevant contexts have been obtained, a weighted version of the correlation function can be computed:

$$usim(u, u', \boldsymbol{\sigma}, \epsilon_4) = \frac{\sum\limits_{\langle i,\mathbf{c},\mathbf{c}' \rangle \in \Lambda_{\epsilon_4}} (r_{u,i,\mathbf{c}} - \bar{r}_u)(r_{u',i,\mathbf{c}'} - \bar{r}_{u'}) csim(\mathbf{c}, \mathbf{c}', \boldsymbol{\sigma})}{\sqrt{\sum (r_{u,i,\mathbf{c}} - \bar{r}_u)^2 \sum (r_{u',i,\mathbf{c}'} - \bar{r}_{u'})^2 \sum\limits_{\langle i,\mathbf{c},\mathbf{c}' \rangle \in \Lambda_{\epsilon_4}} csim(\mathbf{c}, \mathbf{c}', \boldsymbol{\sigma})^2}} \tag{2.20}$$

With the four components defined above, the rating can be predicted by using the following equation:

$$\hat{r}_{u,i,\boldsymbol{\sigma}} = \underbrace{\bar{\rho}_{u,\boldsymbol{\sigma}_3,\epsilon_3}}_{\text{user baseline}} + \frac{\overbrace{\sum_{u' \in \hat{U}_{u,\boldsymbol{\sigma}_1,\epsilon_1}}}^{\text{neighbourhood selection}} \overbrace{(\rho_{u',i,\boldsymbol{\sigma}_2,\epsilon_2} - \bar{\rho}_{u',\boldsymbol{\sigma}_2,\epsilon_2})}^{\text{neighbour contribution}} \times \overbrace{usim(u, u', \boldsymbol{\sigma}_4, \epsilon_4)}^{\text{user similarity}}}{\underbrace{\sum_{u' \in \hat{U}_{u,\boldsymbol{\sigma}_1,\epsilon_1}}}_{\text{neighbourhood selection}} \underbrace{usim(u, u', \boldsymbol{\sigma}_4, \epsilon_4)}_{\text{neighbour contribution}}} \tag{2.21}$$

To find the best values for the $\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_1, \epsilon_1, \ldots, \epsilon_4$ parameters, particle swarm optimization is used with the objective of minimizing the prediction error.

Evaluation DCW was evaluated using two datasets. The first one contains 6360 ratings made by 212 users for 20 food menus and it contains 6 contextual dimensions. The second one contains 1010 ratings made by 69 users about 176 movies with 5 contextual dimensions. Both datasets were collected from surveys. Zheng et al. run rating prediction experiments and compare DCW against a context-less user-based k-NN, a neighbourhood-based pre-filtering approach and against DCR. Experiments show that DCW has better prediction performance than all of the aforementioned approaches in terms of RMSE. DCW's better prediction accuracy comes without sacrificing coverage, which is a great improvement over the DCR approach.

Summary By reducing the dimensionality of the contextual recommendations problem, DCW is able to perform better than DCR in terms of RMSE and coverage. DCW improves the coverage of DCW because it is able to keep ratings that are made under different contexts by assigning them a lower weight, thus reducing the number of situations where no neighbours are found. Zheng et al. evaluated DCW in quite small datasets, leaving open the question of whether it will perform well on datasets with high dimensionality and high sparsity.

2.1.2.3 Contextual SLIM

Contextual SLIM (CSLIM) is introduced in (Zheng, Mobasher & Burke 2014).

Goal CSLIM is a Top-N recommender for context-aware recommendations. This means that its focus is not rating prediction but rather ranking candidate items in order to make a top-N recommendation.

Contribution Zheng, Mobasher & Burke present CSLIM, the first recommender to extend the Sparse Linear Method (SLIM) (Ning & Karypis 2011, 2012) algorithm and incorporate contextual information to make contextual ranking predictions. SLIM is able to make more accurate ranking predictions than other state-of-the-art CARSs.

Differences with other approaches CSLIM is like a neighbourhood-based collaborative filtering approach, so it is different from latent factor approaches like Multiverse Recommendation, CAMF and LCMF. It is more similar to DCR and DCW but those two approaches are user-based, whereas CSLIM has both user- and item-based variants. CSLIM is different to all of the aforementioned approaches because, similarly to SLIM, it only works for ranking prediction whereas all of the other approaches are used to predict ratings.

Notation The CSLIM model uses the following notation:

Table 2.6: Notation of CSLIM.

Symbol	Description
u	A user
i	An item
k	A contextual condition
$r_{u,i}$	The rating that user u gave to item i
$r_{u,i,\mathbf{c}}$	The rating that user u gave to item i under the contextual conditions $\mathbf{c}$
$w_{i,i'}$	The similarity value between items i and i'
d	The rating deviation of item i under the contextual condition k
K	A set of contextual conditions
$\mathbf{r}_u$	A vector with all the ratings made by user u
$\mathbf{w}_i$	A vector containing the similarities of all the items with item i
$\mathbf{R}$	The ratings matrix
$\mathbf{D}$	A matrix storing all contextual rating deviations
$\mathbf{W}$	A matrix storing all the similarity values between all item pairs
$\hat{r}_{u,i,\mathbf{c}}$	The predicted rating that user u would give to item i under the contextual conditions $\mathbf{c}$
$\hat{\hat{s}}_{u,i}$	The predicted ranking score for user u and item i
$\hat{s}_{u,i,\mathbf{k}}$	The predicted ranking score for user u and item i under the contextual conditions $\mathbf{k}$
$\hat{\mathbf{R}}$	The predicted ratings matrix

Model In order to have a better understanding of the CSLIM model, we will first introduce SLIM. SLIM is an item-based neighbourhood-based collaborative filtering model. It is an improvement over the traditional item-based k-NN. An item-based k-NN makes rating predictions by looking at the ratings of items that are similar to an item i. We will let $\hat{I}_i$ be the set of $\mathcal{K}$ most similar items to i. Rating predictions can then be calculated by using the following formula:

$$\hat{r}_{u,i} = \frac{\sum\limits_{i' \in \hat{I}} r_{u,i'} \times isim(i,i')}{\sum\limits_{i' \in \hat{I}} isim(i,i')} \tag{2.22}$$

where $isim(i,i')$ is a function that calculates the similarity between items i and i'. Traditionally, cosine similarity, adjusted cosine similarity or Pearson correlation are used to calculate this similarity.

If we are only interested in making ranking predictions rather than rating predictions, then we can eliminate the denominator part of Equation 2.22, obtaining:

$$\hat{\hat{s}}_{u,i} = \sum_{i' \in \hat{I}} r_{u,i'} \times isim(i, i') \tag{2.23}$$

where $\hat{\hat{s}}_{u,i}$ is the predicted ranking score.

If the ratings are in a ratings matrix $\mathbf{R}$ and item similarity values in a matrix $\mathbf{W}$, then we can calculate the predicted ranking score $\hat{\hat{s}}_{u,i}$ using:

$$\hat{\hat{s}}_{u,i} = \mathbf{r}_u^\top \mathbf{w}_i \tag{2.24}$$

where $\mathbf{r}_u$ from $\mathbf{R}$ is a vector that contains the ratings of all the items rated by user u and $\mathbf{w}_i$ from $\mathbf{W}$ is a vector that contains the similarity values between item i and the rest of items. Since we are assuming that $s_{u,i}$ is unknown, then $s_{u,i} = 0$. To avoid trivial recommendations, where an item recommends itself, then we impose the constraint $diag(\mathbf{W}) = 0$ (Ning & Karypis 2011).

The goal of SLIM is to learn the best values of $\mathbf{W}$ that minimize an objective function so that $\hat{\mathbf{R}} = \mathbf{RW}$. Like in traditional Matrix Factorization approaches, the objective function of this model is composed of terms for prediction error and regularization to help avoid over-fitting. In (Ning & Karypis 2011), $\mathbf{W}$ is learned by using the coordinate descent method.

Zheng, Mobasher & Burke introduce CSLIM, a model based on SLIM that incorporates contextual information in order to make context-aware recommendations (Zheng, Mobasher & Burke 2014). The context is represented by a binary vector $\mathbf{k}$ of size $|K|$, where $|K|$ is the total number of contextual values. Here each $\mathbf{k}_k$ is used to represent the current contextual situation. For example, we could have two contextual dimensions Time and Occasion, and all the contextual conditions can be represented in an $|K|$-sized vector, like this: {*Time=weekend, Time=weekday, Occasion=Business, Occasion=Romance*}. Having the vector $\mathbf{k} = \langle 0, 1, 1, 0 \rangle$ means that the current context is {*Time=weekday, Occasion=business*}.

To incorporate context into the recommendations, a matrix $\mathbf{D}$ that stores the contextual rating deviations is used. Rating deviations are the differences between the rating $r_{u,i,\mathbf{k}}$ under a particular context $\mathbf{k}$ and a context-less rating $r_{u,i}$.

The $\mathbf{D}$ matrix is of size $|I| \times |K|$, where each row represents an item and each column represents a contextual condition (e.g., *Time=weekend, Time=weekday, Companion=Friends*, etc.). Each contextual rating deviation is denoted by $d_{i,k}$. Note how, in this model, it is assumed that additional to the contextual ratings $r_{u,i,\mathbf{k}}$, there are ratings that users made without the influence of a context. These context-less ratings are in the matrix $\mathbf{R}$. If a context-less rating is not available, then an average of the ratings over item i by user u in contexts different to $\mathbf{k}$ is taken. Having this, the predicted contextual rating can be calculated using:

$$\hat{r}_{u,i,\mathbf{k}} = r_{u,i} + \sum_{k=1}^{|K|} d_{i,k}\mathbf{k}_k \tag{2.25}$$

Finally, with the predicted contextual rating $\hat{r}_{u,i,\mathbf{k}}$, Equation 2.24 is modified to include the contextual ratings, like this:

$$\hat{s}_{u,i,\mathbf{k}} = \sum_{\substack{i'=1\\i'\neq i}}^{|I|} \hat{r}_{u,i,\mathbf{k}} w_{i',i} = \sum_{\substack{i'=1\\i'\neq i}}^{|I|} \left(r_{u,i} + \sum_{k=1}^{|K|} d_{i,k}\mathbf{k}_k \right) w_{i',i} \tag{2.26}$$

where $w_{i,i'}$ corresponds to the entry (i, i') of the $\mathbf{W}$ matrix.

The parameters to be learned in this model are $\mathbf{D}$ and $\mathbf{W}$. The model presented in 2.26 is called CSLIM-I-CI. There are two further variations of this model: CSLIM-I-CU and CSLIM-I-C. In these variations, what changes is the size of the $\mathbf{D}$ matrix. For instance, in CSLIM-I-CU, the matrix $\mathbf{D}$ has a size $|U| \times |K|$ and it stores how the context affects each user. In CSLIM-I-C, $\mathbf{D}$ is a vector of length $|K|$ that stores how each contextual value deviates from the ratings in a general way, i.e., not for each item, but generally, very similar to the CAMF-C model presented in Section 2.1.1.2.

Evaluation In (Zheng, Mobasher & Burke 2014), results are shown for the evaluation of CSLIM across three datasets involving ratings of food, restaurants and music. Their size is 6360, 1421 and 3010 ratings respectively and all of them were collected using surveys. CSLIM has better performance than SLIM, CAMF, context-aware splitting approaches (Zheng, Burke & Mobasher 2014) and Multiverse Recommendation in terms of precision, recall and mean average precision across the three datasets.

Summary CSLIM presents a new way to make Top-N recommendations by extending the SLIM model in order to incorporate contextual information into it. CSLIM is able to make better ranking predictions compared to other state-of-the-art approaches.

One concern about the datasets used in (Zheng, Mobasher & Burke 2014) to evaluate CSLIM is that the datasets in which CSLIM is evaluated are very dense: two of them have a user-item density of 100% and the other one of 60%. In real-world applications these levels of density are rarely seen. For instance, the MovieLens 10M[1] has a density of 0.14%, the Yelp Hotels and Yelp Restaurants[2] have densities of 1.2% and 0.2% respectively, and the Four-city dataset of TripAdvisor hotel reviews[3] has a density lower than 0.1%. Therefore, the performance of CSLIM on sparse datasets like those just mentioned remains a big concern.

2.1.3 Summary

In this section we have reviewed six Context-Aware Recommender Systems. Three of them were latent-factor models and three of them were neighbourhood-based models. A classification of the CARS that we have reviewed can be seen in Figure 2.1. We can summarize the three main weaknesses of this past work as: the dimensionality, the lack of suitable contextual datasets, and the sparsity of the datasets used to evaluate the models.

Table 2.7: Summary of the CARSs that we have reviewed.

Model	Type	Number of Parameters	Evaluation		Data Source			
			Rating	Top-N	Semi-synthetic	Survey	User study	Real-world
Multiverse	Latent factor	Exponential	✓		✓	✓		
CAMF	Latent factor	Linear	✓		✓	✓		
LCMF	Latent factor	Linear	✓	✓			✓	
DCR	Neighbour-based	Linear	✓			✓		✓
DCW	Neighbour-based	Linear	✓			✓		
CSLIM	Neighbour-based	Linear		✓		✓		

Regarding the dimensionality, most of the approaches are aware of the dimensionality problem existing in CARS and try to reduce the number of parameters

[1]`https://grouplens.org/datasets/movielens/10m/`
[2]`https://www.kaggle.com/c/yelp-recsys-2013`
[3]`http://www.cs.cmu.edu/~jiweil/html/four_city.html`

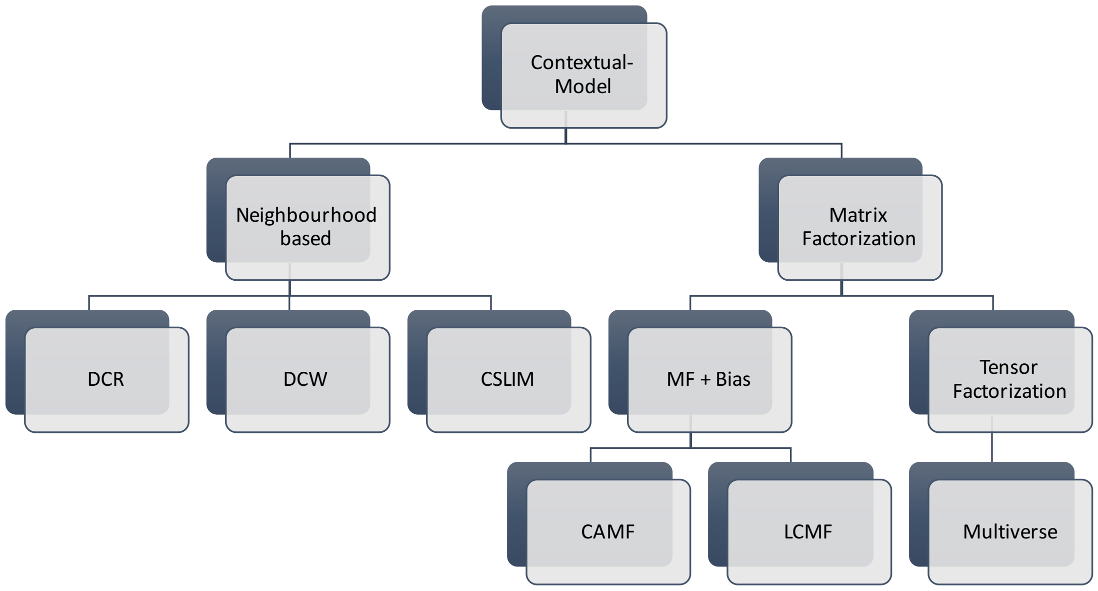


Figure 2.1: Classification of CARS

to be learned. The exception to this is the Multiverse Recommendation model in which the number of parameters is exponential to the number of contextual dimensions. In the rest of the approaches, the number of parameters grows linearly with the number of dimensions as we can see in Table 2.7.

Another problem is the lack of contextual datasets for evaluation. As we can see in Table 2.7, from all of the reviewed approaches, only one uses a real-world dataset; the rest of them use datasets that are either synthetic or come from surveys; in one other case a dataset collected from a user study was used. Using a synthetic dataset gives little confidence in the soundness of the evaluated model. Surveys may bias the behaviour of users, but also tend to result in very dense datasets, which are rare when it comes to real-world datasets. Datasets collected from user studies are better, but they are very expensive to obtain. As seen in (Unger et al. 2016), a mobile phone application had to be designed and many volunteers had to be hired to conduct the user study.

The lack of datasets also hides a third problem: the sparsity problem. This problem manifests itself in two ways, with low coverage and failing to beat context-less models using context-rich datasets. The first problem is evident in the results reported in (Zheng et al. 2012*a*) for DCR. DCR was the only approach that used a real-world dataset and it had a coverage of just 8%, meaning that in 92% of the cases a prediction could not be made. Since none of the other approaches used real-world datasets, we can not conclude how those CARS might perform compared to context-less approaches on real world datasets.

In the next section we will review recommender systems that exploit user-generated reviews. These approaches all use real-world datasets but they do not explicitly extract contextual information about the context in which the item was consumed from the reviews.

2.2 Unsupervised Review-Based Approaches

As we have seen in our reviews of context-aware approaches, contextual information is often not available, either because it cannot be obtained or because it is very expensive to obtain. There is another branch of recommender systems research that focuses on exploiting an alternative source of information: consumer reviews.

In many websites, such as Amazon, Yelp and TripAdvisor, not only may consumers give ratings on items that they have consumed, additionally they can also write reviews describing their experiences with those items. One interpretation of a consumer's review of an item is that it gives a justification of why that consumer gave that rating to that item (McAuley & Leskovec 2013). Since the users themselves are explaining why they gave a rating, this becomes a rich source of information that can be exploited to enhance the performance of recommender systems.

Including reviews as part of the recommendation model can bring several benefits, such as helping to deal with the problem of sparsity in the dataset, helping to solve the cold-start problem for new users, and lastly increasing the accuracy of the recommendations even in non-sparse datasets by including additional information such as the opinions of the consumers (Chen et al. 2015).

Many models that exploit reviews have been proposed. We will focus only on exploring review-based models that are unsupervised. This is in line with the goal of our own work, which is to use unsupervised techniques so that we do not need to use a pre-defined set of contextual variables. In this section, we will survey two types of approaches: generative models and feature extraction approaches.

2.2.1 Generative models

In discriminative models, given the data $\mathbf{x}$, a classifier tries to predict its class y, basically by estimating $p(y|\mathbf{x})$. These are called discriminative models because, given the data $\mathbf{x}$, they are able to discriminate between the different classes. Generative models work the other way around: having a class y they try do describe what the data should look like. In other words, they are able to model $p(\mathbf{x}|y)$ (Nguyen et al. 2014). They are called generative models because, given the classes, they are able to generate the data that comes from those classes.

In this section we will describe the generative models that are used to produce review-based recommendations.

2.2.1.1 Hidden Factors as Topics

Goal The goal of the work described in (McAuley & Leskovec 2013) is to create a recommender system that combines latent rating dimensions with latent review topics.

Contribution Hidden Factors as Topics (HFT) is able to match latent rating dimensions with latent topics, which helps to explain why a user likes or dislikes a certain item. HFT is also able to make more accurate recommendations than other review-based recommender systems with a totally unsupervised approach that also does not use heuristics and does not require any form of preprocessing.

Differences with other approaches HFT is different to other approaches in the sentiment analysis field (Dong et al. 2013, 2014, 2016, Hu & Liu 2004) because in sentiment analysis the goal is to predict the sentiment of a user towards an item from a review, whereas the goal of HFT is to predict ratings of items that the user has not seen. It is also different from approaches in the aspect extraction field (Ganu et al. 2013, Levi et al. 2012) because many of those proposals require human annotators, whereas HFT is unsupervised.

Notation The HFT model uses the notation presented in Table 2.8:

Table 2.8: Notation of HFT.

Symbol	**Description**
u	A user
i	An item
$r_{u,i}$	The rating that user u gave to item i
$\tau_{u,i}$	The review that user u wrote about item i
$w_{\tau,j}$	j^{th} word of review τ
$z_{\tau,j}$	Topic for the j^{th} word of review τ
α	The offset or average rating
β_u	The bias of user u
β_i	The bias of item i
$\boldsymbol{\tau}_i$	A document containing all the reviews written about item i
$\mathbf{p}_u$	The latent features vector of user u
$\mathbf{q}_i$	The latent features vector of item i
θ_i	The vector containing the topics distribution for item i
ϕ_t	The vector containing the words distribution for topic t
$\hat{r}_{u,i}$	The predicted rating that user u would give to item i
$\|T\|$	The number of latent dimensions/topics

Model McAuley & Leskovec propose a model in which the vector of latent features that describes an item i (or user u) is learned while having regard to the probability vector of topics i that discuss that item in item reviews. The latent features matrices are optimized using gradient descent, but instead of placing regularization parameters that punish the complexity of the model, the topic model probability is used as the regularization parameter. To learn the topic model, McAuley & Leskovec use a transformation function that relates each latent features vector $\mathbf{q}_i$ to a topic document probability vector θ_i.

Hence, HFT discovers topics that are correlated with item and user latent factors, $\mathbf{q}_i$ and $\mathbf{p}_u$. Documents are modeled as the set of all reviews about an item. Document $\boldsymbol{\tau}_i$ contains all the reviews about item i.

For each item i, a topic distribution θ_i is learned. θ_i contains the extent to which each of the k topics is being discussed across all reviews for item i. Implicitly, this assumes that the number of topics in the reviews is equal to the number of latent features in the user/item space.

The rating parameters $\mathbf{q}_i$ and the review parameters θ_i are linked through a transformation function, given by:

$$\theta_{i,t} = \frac{\exp(\kappa \mathbf{q}_{it})}{\sum_{t'} \exp(\kappa \mathbf{q}_{it'})} \tag{2.27}$$

where κ is a parameter that controls the 'peakiness' of the transformation. As $\kappa \to \infty$, θ_i will approach a vector that has a value of 1 only for the largest index of $\mathbf{q}_i$; as $\kappa \to 0$, θ_i approaches a uniform distribution. Intuitively, large κ means that users only discuss the most important topic, while a small κ means that users discuss all the topics evenly.

Thanks to the transformation function, there is no point in learning both θ_i and $\mathbf{q}_i$, since one uniquely defines the other. In practice the value of $\mathbf{q}_i$ is learned. The transformation function is the key contribution in this model.

The HFT model is based on the idea that factors should be useful to predict ratings, but also once those factors are transformed into topics, we should obtain a good generative model of the text review corpus using θ_i. The objective function for a corpus $\mathcal{S}$, which is composed of ratings and reviews, is defined as:

$$f(\mathcal{S}|\Theta, \Phi, \kappa, z) = \sum_{r_{u,i} \in \mathcal{S}} \underbrace{(\hat{r}_{u,i} - r_{u,i})^2}_{\text{rating error}} - \mu \cdot \underbrace{l(\mathcal{S}|\theta, \phi, z)}_{\text{corpus likelihood}} \tag{2.28}$$

The first part of the above equation is the prediction error of the ratings and the second part is the likelihood that the words in the corpus $\mathcal{S}$ can be generated with the learned parameters. μ is a hyperparameter that trades off the importance between rating prediction and the likelihood of generating the corpus. The corpus likelihood is very important in this equation, even if the model is just being used to predict ratings and not obtaining topics, since it acts like a regularization term.

To infer the values of the rating parameters $\Theta = \{\alpha, \beta_u, \beta_i, \mathbf{p}_u, \mathbf{q}_i\}$, and the parameters associated with topics $\Phi = \{\theta, \phi\}$, McAuley & Leskovec attempt to minimize the function defined in equation 2.28. That is:

$$\underset{\Theta,\Phi,\kappa,z}{\text{argmin}} f(\mathcal{S}|\Theta, \Phi, \kappa, z) \tag{2.29}$$

Since $\mathbf{q}_i$ and θ_i are linked through the transformation function, a change in $\mathbf{q}_i$ modifies the values of both the rating prediction and the corpus likelihood. For that reason Θ and Φ cannot be optimized independently.

The optimization procedure to obtain the parameter is composed of two steps:

$$\text{update } \Theta^{(t)}, \Phi^{(t)}, \kappa^{(t)} = \underset{\Theta,\Phi,\kappa,z}{\text{argmin}} f(\mathcal{S}|\Theta, \Phi, \kappa, z) \tag{2.30}$$

$$\text{sample } z_{\tau,j}^{(t)} \text{with probability} \quad p(z_{\tau,j}^{(t)} = k) = \phi_{k,w_{\tau,j}}^{(t)} \tag{2.31}$$

In the first step, the values of z (the topic assignments for each word) are fixed and the remaining variables (Θ, Φ and κ) are fitted using gradient descent. In the second step an iteration is performed over all the words from all the documents and their topics assignments are updated. The two steps are executed until convergence.

Note that the second step is very similar to what is done in Latent Dirichlet Allocation (LDA) (Blei et al. 2003). In LDA, topic assignments θ are updated by sampling from a Dirichlet distribution. Instead, in HFT, the topics assignments are obtained by transforming the $\mathbf{q}_i$ parameter, which is calculated in step one.

Evaluation In (McAuley & Leskovec 2013), HFT is compared against an offset baseline recommender, matrix factorization and LDA in terms of rating prediction using Mean Squared Error (MSE) as the evaluation metric. Twenty-seven real-world datasets are used to evaluate the performance of the models. All of the datasets are composed of reviews and ratings, which in total add up to nearly 42 million reviews. Twenty of those datasets come from Amazon and the rest come from Ratebeer, Beeradvocate, Citysearch and Yelp Phoenix. Results show that HFT can produce more accurate recommendations than all of the other approaches across all 27 real-world datasets.

Summary One advantage of this model is that it is good for users who have very few ratings but have provided reviews. When a user has a small amount of ratings, collaborative filtering models can not make accurate predictions, but when topics are used, reviews can give a better insight of what the preferences of the user are, even with a low number of reviews.

One fundamental disadvantage of this approach is that the number of topics of the topic model has to be the same as the number of factors. It has been shown that a quite high number of factors will translate into a better recommender (Koren 2008, Unger et al. 2016). But the same does not apply to topic models. It is well known that a large number of topics will result in “over-clustering” a topic model, where many topics will overlap, having many words in common (Greene et al. 2014). In (McAuley & Leskovec 2013), it is mentioned that future work would be to add additional latent factors that are not constrained to topics in the topic model, so that the number of latent factor is independent of the number of topics. Ideally this would permit to do matrix factorization with a higher number of factors and use a topic model with the right number of topics (which does not have to be so high). This argument applies especially because McAuley & Leskovec talk about the interpretability of the latent dimensions, which would require a good topic model. McAuley & Leskovec claim to obtain highly interpretable textual labels for latent recommendations, but this can only be achieved if the topics are good. As stated in (Ling et al. 2014), *"there is a discrepancy between the item topic distribution θ_i in LDA and the item feature vector $\mathbf{q}_i$ in matrix factorization. The former is a distribution which is all positive and sums up to 1 while the latter can assume any real value"*.

2.2.1.2 Ratings Meet Reviews

Goal In (Ling et al. 2014), the Ratings Meet Reviews (RMR) model is presented. The goal of Ling et al. is to build a recommender system that alleviates the cold-start problem through the use of reviews and to provide recommendations that are interpretable.

Contribution There are two contributions presented in (Ling et al. 2014). First, a new method to combine content-based filtering with collaborative filtering is presented. This method models the text reviews and numeric ratings simultaneously. Second, the RMR approach is shown to be more accurate than other baseline and state-of-the-art approaches.

Differences with other approaches RMR is very similar to HFT in the sense that they are both fitting a latent factor rating matrix and a topic model review corpus simultaneously. The difference lies in the fact that HFT is a Matrix Factorization model that replaces the regularization by the corpus likelihood. HFT also has a transformation function that links the items' latent factors matrix to the latent topics. RMR, on the other hand, uses a mixture of Gaussians to model the ratings assuming that the mixture proportion has the same distribution as the topic distribution. In this way, the need for a transformation function is also avoided.

Notation The RMR model uses the following notation:

Table 2.9: Notation of RMR.

Symbol	**Description**
u	A user
i	An item
$r_{u,i}$	The rating that user u gave to item i
$w_{\tau,j}$	j^{th} word of review τ
$z_{\tau,j}$	Topic for the j^{th} word of review τ
$\boldsymbol{\tau}_i$	A document containing all the reviews written about item i
$\mathbf{p}_u$	The latent features vector of user u
θ_i	The vector containing the topics distribution for item i
ϕ_t	The vector containing the words distribution for topic t

Model In (Ling et al. 2014) a model that combines content-based filtering with collaborative filtering using information from ratings and reviews is presented. This approach, called RMR, extracts topics from text reviews and aligns the topics with rating dimensions in order to improve prediction accuracy. With this approach, latent topics become interpretable and it can help to recommend cold-start items. To help solve the problem of interpretation present with latent features, (Ling et al. 2014) tag each latent dimension with a word cloud that explains the meaning of the dimension and align the latent topic spaces with rating spaces. Similarly to HFT, in RMR the reviews are grouped per item.

As in (McAuley & Leskovec 2013), the goal of RMR is to find the best values for the parameters $\Phi = \{\theta, \phi, \mathbf{p}\}$, Ling et al. do this through a generative process in which they 'tune' the parameters using a Gibbs sampler. The generative process is the following:

Algorithm 1 Ratings Meet Reviews

```
    Input: collection of documents $D$, number of topics $k$.
    Output: tuned parameters $\Phi = \{\theta, \phi, \mathbf{p}\}$.
 1: for all $u \in U$ do                          ▷ For each user
 2:     for all $t \in T$ do                      ▷ For each latent topic
 3:         $\mathbf{p}_{u,t} \sim$ Gaussian$(\mu_0, \sigma_0^2)$
 4: for all $t \in T$ do                          ▷ For each latent topic
 5:     $\phi_t \sim$ Dirichlet$(\zeta)$
 6: for all $i \in I$ do                          ▷ For each item
 7:     $\theta_i \sim$ Dirichlet$(\eta)$
 8:     for all $w_{i,j}$ do                      ▷ For each description word
 9:         $z_{\boldsymbol{\tau}_i,j} \sim$ Multinomial$(\theta_i)$          ▷ Draw topic assignment
10:         $w_{\boldsymbol{\tau}_i,j} \sim$ Multinomial$(\phi_{z_{\boldsymbol{\tau}_i,j}})$      ▷ Draw word
11:     for all $r_{u,i}$ do                      ▷ For each observed rating assigned by $u$ to $i$
12:         $f_{u,i} \sim$ Multinomial$(\theta_i)$            ▷ Draw topic assignment
13:         $r_{u,i} \sim$ Gaussian$(\mathbf{p}_{u,f_{u,i}}, \sigma^2)$      ▷ Draw the rating
```

RMR uses a mixture of Gaussians (instead of Matrix Factorization) to model the ratings. These user-topic specific Gaussian distributions describe how a user values the aspects denoted by each latent topic. The item is modeled as a distribution of topics which, together with the user-topic specific Gaussian distributions, determine how a user would rate the item. The items and review text are connected by the same item topic distribution θ. The more a user talks about certain aspects concerning an item, the higher the distribution will be on these topics, which in turn affects the rating that the user would assign to the item.

To obtain these distributions, Ling et al. use a Gibbs sampler to learn the parameters θ, ϕ and $\mathbf{p}$, and finally they can make predictions about an item using the formula:

$$\hat{r}_{u,i} = \sum_t \theta_{i,t} \mathbf{p}_{u,t} \tag{2.32}$$

Evaluation RMR was tested using the exact same datasets as in (McAuley & Leskovec 2013) for HFT: in summary, a collection of 27 datasets that contain close to 42 million reviews and ratings. RMR was evaluated against HFT, LDA, Matrix Factorization and the model proposed in (Wang & Blei 2011) in terms of rating prediction accuracy using RMSE as the evaluation metric. The results presented show that RMR is more accurate in terms of RMSE than the other approaches. It also shows that approaches that use text reviews perform much better than the ones that do not when the sparsity is very high.

Summary One of the main advantages of this model is that it lines-up both reviews and ratings using a generative model, removing the need to use a transformation function as needed in (McAuley & Leskovec 2013). Where there are very few ratings, the topic distribution θ can still be learned accurately from the reviews. Another advantage that this method has over methods that use Matrix Factorization is that in this method the latent features obtained from the ratings and from the reviews have a direct mapping, whereas the other methods such as the one presented in (McAuley & Leskovec 2013) need to develop a transformation function to perform this mapping. On the other hand, although topics can be interpretable, Ling et al. do not provide a methodology or a metric to measure the quality of the generated topics.

2.2.1.3 Jointly Modeling Aspects, Ratings and Sentiments

The Jointly Modeling Aspects, Ratings and Sentiments (JMARS) model is presented in (Diao et al. 2014).

Goal Diao et al. build an unsupervised movie recommender system based on ratings and reviews. The goal of JMARS is to predict the rating that a user will give to an item, and also to predict the review that the user will write for that item. The predicted rating is, of course, in the form of a real number, and the predicted review is presented as a collection of words. The predicted ratings and reviews are modeled jointly.

Contribution The contribution in (Diao et al. 2014) is a recommender that is able to outperform other state-of-the-art recommenders. The model is able to obtain a representations of user interests and movie properties and it is able to uncover aspect-specific sentiments.

Differences with other approaches JMARS is different to HFT and RMR in the sense that JMARS includes several language models for the reviews. For instance, it has one language model for movie descriptions, another one for aspects of the movies, another one for the sentiments associated to the movies and so on. JMARS also incorporates aspects and sentiment into its model, whereas HFT and RMR model reviews and ratings, associating each topic dimension to one latent factor.

Notation The JMARS model uses the following notation:

Table 2.10: Notation of JMARS.

Symbol	Description
u	A user
i	An item
$r_{u,i}$	The rating that user u gave to item i
α	The offset or average rating
β_u	The bias of user u
β_i	The bias of item i
$\mathbf{p}_u$	The latent features vector of user u
$\mathbf{q}_i$	The latent features vector of item i
$\hat{r}_{u,i}$	The predicted rating that user u would give to item i
$\boldsymbol{\mathcal{A}}$	A scaling matrix of aspects

Model In JMARS, θ_u is the distribution that encodes the interest of users towards the aspects they write and care about. Similarly, θ_i encodes the amount of the aspects that the item i contains.

The presented model assumes that users u and items i are characterized by latent factor vectors $\mathbf{p}_u$ and $\mathbf{q}_i$ respectively that are drawn from zero-mean spherical Gaussian priors.

$$\mathbf{p}_u \sim \mathcal{N}(0, \sigma_u^2 \mathbf{I}) \quad \text{and} \quad \mathbf{q}_i \sim \mathcal{N}(0, \sigma_i^2 \mathbf{I}) \tag{2.33}$$

where σ_u^2 and σ_i^2 are the hyperparameters that represent the variances related to the users and items.

This model assumes that items are composed of aspects and that the same item can be rated differently, depending on the aspect on which the item is being evaluated. Thus, for each aspect there is going to be a different rating $r_{u,i,a}$, and the final rating $r_{u,i}$ of the item is going to be given by aggregating all the aspect-ratings.

The aspects are captured by $\boldsymbol{\mathcal{A}}$, and they are useful to see if the properties of an item i match the expectations of the user u when viewed under a specific aspect. In other words, the rating $r_{u,i,a}$ will reveal if a user appreciates a particular aspect of an item such as, for instance, the plot or the special effects of a movie.

To calculate the rating that a user would give to an item under a certain aspect the following equation is used:

$$\hat{r}_{u,i,a} = \alpha + \beta_u + \beta_i + \mathbf{p}_u^\top \boldsymbol{\mathcal{A}} \mathbf{q}_i \tag{2.34}$$

where β_u and β_i are the user and item biases, and α is a common bias. The goal of the matrix $\boldsymbol{\mathcal{A}}$ is to emphasize the aspect-specific properties, while the vectors $\mathbf{p}_u$ and $\mathbf{q}_i$ contain the general profile. In this way, one could say that, even if an item is very good, it could not excel in some aspects. For instance, a movie can be very good overall but have poor special-effects.

Each element of $\boldsymbol{\mathcal{A}}$, $\mathbf{p}_u$, $\mathbf{q}_i$, β_u and β_i is assumed to follow a Gaussian distribution with a fixed variance. In other words, the variance is a fixed real number and does not come from a distribution.

Diao et al. also assume that users have an interest distribution θ_u related to the aspects they write and care about. The same applies for the items, which contain a number of aspects, as indicated by θ_u.

To make inferences, the Gibbs-EM method is used. This method alternates between collapsed Gibbs sampling and gradient-descent to estimate the parameters in the model.

The parameters to be learned in this model are the biases α, β_u and β_i, the hidden factor vectors $\mathbf{p}_u$ and $\mathbf{q}_i$, the aspects $\boldsymbol{\mathcal{A}}$ and sentiments. They are learned in order to model the user ratings and to be able to maximize the probability of generating the text reviews. The likelihood of generating the reviews and predicting the ratings is given by the negative log posterior, defined as:

$$\mathcal{L} := -\log p(R, W|\Upsilon, \Omega) \tag{2.35}$$

where R denotes the ratings, W denotes the words, Υ denotes the Gaussian hyperparameters and Ω denotes the Dirichlet hyperparameters.

The terms of Equation 2.35 are decomposed for the purpose of using the inference algorithm, obtaining:

$$\mathcal{L} = \sum_{r_{u,i} \in R} [\epsilon^{-2}(r_{u,i} - \hat{r}_{u,i})^2 - \log p(w_{u,i}|\Upsilon, \Omega)]. \tag{2.36}$$

The left part of the above equation denotes the prediction error on user ratings. The right part denotes the probability of observing the generated reviews conditioned on priors.

Evaluation JMARS is evaluated using a movie dataset with ratings and reviews collected from IMDb[4]. The dataset contains 348415 movie reviews and ratings made by 54671 users about 22380 items, having a total density of 0.03%. The dataset is not publicly available.

Diao et al. evaluate the rating prediction performance using MSE as the evaluation metric. They compare JMARS against Probabilistic Matrix Factorization and Hidden Factors as Topics. The results show that they have a better performance against both methods in terms of MSE.

[4]`https://www.imdb.com`

A qualitative evaluation of JMARS is also presented, showing the learned aspects from the reviews, the extracted sentiment words and the movie specific words. No comparison is made with other approaches for the qualitative evaluation.

Summary JMARS is a model that brings together ratings and reviews to make rating predictions. Results show that it is able to beat other state-of-the-art recommender systems in terms of rating prediction accuracy. It is also completely unsupervised and, unlike other aspect-based recommendation models (Chen & Chen 2015, Levi et al. 2012), it does not need the aspects to be explicitly defined.

Even though this work looks very promising, it has a few drawbacks. Only one dataset is used to evaluate JMARS, which is a big weakness. The model is quite complex and makes many assumptions. Even though it is a probabilistic model, given the number of assumptions, it is more similar to an heuristic-based model. Negation cases, such as “not good”, are not incorporated.

To evaluate the aspects ratings of the reviews, only one review is presented, there are no defined metrics, and showing one positive case does not constitute a proof that the model is working properly. No methodology is presented to measure the quality of the extracted aspects.

2.2.1.4 Topic-Criteria

Goal The goal of the work described in (Rossetti et al. 2013) is to extend the topic modeling method in order to make interpretable user and item models that can explain user preferences and produce recommendations. Two models are introduced: Topic-Criteria (TC) and Topic-Sentiment Criteria (TSC).

Contribution The contribution of (Rossetti et al. 2013) is that they create two new recommendation models that are based on the topic modeling method. These models are able to outperform different baselines on rating prediction on two real-world datasets. The presented recommendation models make use of both ratings and reviews in order to predict ratings, and a user model is built using only reviews in order to describe the preferences of users.

Differences with other approaches The difference between HFT (McAuley & Leskovec 2013), RMR (Ling et al. 2014), JMARS (Diao et al. 2014) and the models presented in (Rossetti et al. 2013) is that the former approaches learn their models using reviews and ratings jointly, whereas TC and TSC first build a topic model using LDA (Blei et al. 2003) or Joint Sentiment-Topic Model (JST) (Lin & He 2009) and then they use that topic model along with ratings to be able to make predictions.

Notation The TC model uses the following notation:

Table 2.11: Notation of TC.

Symbol	Description
u	A user
i	An item
$\tau_{u,i}$	The review that user u wrote about item i
z	A topic
Υ_u	The set of reviews made by user u
Υ_i	The set of reviews made about item i

Model We will limit ourselves to describing just the TC approach as both TC and TSC are based on the same principle. We refer to the reader to (Rossetti et al. 2013) to find a detailed description of TSC. TC is based on the idea of building profiles for both users and items and then matching those profiles in order to make predictions. User profiles are built using only reviews whereas item profiles are built using both reviews and ratings.

In TC the user profile is built by averaging the topic weights across all the items the user has written reviews about, as follows:

$$UP(u,z) = \frac{\sum_{\tau_{u,i} \in \Upsilon_u} p(z|\tau_{u,i})}{|\Upsilon_u|} \tag{2.37}$$

where $p(z|\tau_{u,i})$ is the strength of topic z in review $\tau_{u,i}$ and Υ_u is the set of reviews written by user u.

Items profiles are built using both topic weights and ratings:

$$IP(i,z) = \frac{\sum_{\tau_{u,i} \in \Upsilon_i} p(z|\tau_{u,i}) \cdot r_{u,i}}{\sum_{\tau_{u,i} \in \Upsilon_i} p(z|\tau_{u,i})} \tag{2.38}$$

where Υ_i is the set of reviews written about item i.

In summary, Rossetti et al. are mapping both users and items to a common aspect space that is determined by the number of topics k. The user profile will reveal the aspects that the user cares about and the item profile will reveal how satisfied the users are with each of the aspects of a given item.

In order to predict ratings, the sum of the product for each topic in the user and item profiles is calculated, as indicated by 2.39:

$$\hat{r}_{u,i} = \sum_{z=1}^{k} UP(u,z)IP(i,z)\varphi_z \tag{2.39}$$

where φ_z is a weight that is optimized to minimize the rating prediction error. φ_z represents the strength that topic z contributes to the rating.

Evaluation Rossetti et al. evaluate TC using two real-world datasets from the Yelp and TripAdvisor websites. The Yelp dataset is part of the Yelp Data Challenge and contains business reviews mainly about restaurants, and the TripAdvisor dataset is composed of hotel reviews. In order to have different levels of sparsity, the authors removed items and users that had at least n reviews, resulting in four datasets: Yelp-5-5, Yelp-10-10, TA-3-3 and TA-5-5. Yelp-5-5 contains 145735 reviews made by 9382 users about 3733 items, Yelp-10-10 contains 101416 reviews made by 2413 users about 3802 items, TA-3-3 contains 83395 reviews made by 12342 users about 13048 items, and TA-5-5 contains 14656 reviews made by 1774 users about 1850 items. RMSE is used to evaluate the performance of both TC and TSC. In both Yelp datasets, TC has better performance than TSC, k-NN and Probabilistic Matrix Factorization (Salakhutdinov & Mnih 2008). In both TripAdvisor datasets, TSC outperforms all of the other approaches.

Rossetti et al. also evaluate TC and TSC to predict the ratings of reviews using only the text. For this task, TSC shows the best performance. This makes sense since TSC exploits sentiment words which reflect the polarity of the reviews, thus helping to better predict ratings.

Summary In (Rossetti et al. 2013), two recommendation models that are built on top of topic models are presented. These models build user and item profiles that have the size of the number of topics and then match them together in order to make rating predictions. The TSC approach is especially useful for rating prediction from reviews: by exploiting sentiment words, better predictions can be made. The advantage of the presented models is that they are completely unsupervised.

We saw how users and items are mapped into a common latent space that is determined by the topic model, which is built using only reviews. The model could be improved by also capturing the interaction between users and items by using ratings directly, as is done in matrix factorization. This would result in a model that contains both a review-based latent space and a rating-based latent space.

Besides using their model for rating predictions, Rossetti et al. also use their model to analyze and interpret topics and to suggest ratings for reviews. The analysis part can be useful for explanations. However, no evaluation metrics regarding the interpretation of models are presented (only one selected example is presented), so there is no way to know how good or bad this feature is. This is a common drawback that we also found in (McAuley & Leskovec 2013, Ling et al. 2014, Diao et al. 2014).

2.2.2 Feature extraction approaches

Feature extraction approaches deal with the task of extracting useful features from free-form text reviews. These features are used, along with ratings, to make predictions. Because reviews are written in an unstructured way, feature extraction can become complex and techniques from Natural Language Processing (NLP) are often used. It is common to see techniques such as part-of-speech (POS) tagging, stop-word removal, stemming, lemmatizing and representing documents as bags of words (Aggarwal 2016). These techniques can be used to extract product and user features, sentiments or even to predict the quality of the reviews themselves (Chen et al. 2015).

In this section we will explore the approaches that use feature extraction techniques to exploit the information contained in reviews and produce better recommendations.

2.2.2.1 Cold Start Context-Based Hotel Recommender

Goal The goal of the model presented in (Levi et al. 2012) is to build a recommender system that can provide cold-start recommendations for hotels.

Contribution Cold Start Context-Based Hotel Recommender (CSCB) analyzes the information contained in reviews to extract common traits of user groups in order to make recommendations for users that share a set of characteristics. These characteristics include the intent of the trip and the nationality of the users. CSCB extracts these traits and uses them to make recommendations that bring higher satisfaction to users compared to hotel web services.

Differences with other approaches CSCB is different to other approaches that use demographics to make recommendations, such as (Lam et al. 2008, Park & Chu 2009), because CSCB incorporates features extracted from reviews into the recommendation model, whereas (Lam et al. 2008, Park & Chu 2009) require that the features are already given.

Notation The CSCB model uses the following notation:

Table 2.12: Notation of CSCB.

Symbol	Description
u	A user
i	An item
r	A review
s	A sentence
f	A feature
op	An opinion word
or_{op}	The sentiment orientation of the opinion word op

Model Levi et al. propose an heuristic-based recommender system that extracts information from textual reviews in order to recommend hotels (Levi et al. 2012). The recommender system is called CSCB. The steps CSCB takes are: extract features, calculate feature weights for nationalities and intents, build aspects from features and calculate the weight of aspects, calculate the feature opinion orientations, and finally calculate the hotel score. In CSCB users are asked to provide information about their nationality, trip intent and preferences towards the mined aspects.

The first thing Levi et al. do is extract features from reviews at a sentence level. So they perform part-of-speech tagging for each sentence and keep only the nouns and noun phrases. Subsequently, Levi et al. try to determine the importance of each feature for each nationality group and intent group. They do this by assigning a weight to each feature depending on how frequent the feature appears in the sentences written by the user or nationality group or intent group. At the end of this process they obtain a series of weights φ_c^f where f is a feature and c can be either a nationality or an intent.

The next step is to determine the aspects that compose the items. This is done by clustering the features into six clusters; each one of the clusters represents an aspect. One thing to note about the clustering algorithm is that features can belong to one and only one cluster.

To calculate the feature opinion orientation, Levi et al. first manually define a seed list of adjectives and assign an orientation score of (-1,+1) to each of them. Then, the seed list is enlarged by finding synonyms and antonyms using WordNet (Miller et al. 1990). The seed list grows in the process. In that way, all opinion words will have an orientation. To calculate the score of a feature f in a sentence s the following function is used:

$$score(f, s) = \sum_{op \in s} \frac{or_{op}}{d(op, f)}, \tag{2.40}$$

where or_{op} is the orientation $(+1, -1)$ of the opinion word op, $dist(op, f)$ is the distance (word count) between the opinion word or and the feature f in the sentence.

To calculate the review score, first a total weight must be calculated. This is done by:

$$\varphi_u^f = \varphi_{u_p}^f + \varphi_{u_n}^f + \varphi_{u_{pref}}^f, \tag{2.41}$$

where φ_{u_n} is the nationality weight, φ_{u_p} is the intent weight and $\varphi_{u_{pref}}$ is the weight based on the user preferences. To produce the score of a review, the score of each feature f in each sentence s of review τ written by user u is calculated like this:

$$score(\tau, u) = \sum_{s \in \tau} \sum_{f \in s} \varphi_u^f \tag{2.42}$$

Finally, using an aggregation function (see (Levi et al. 2012) for details), the hotel score is calculated.

Evaluation CSCB is evaluated on two datasets, one from TripAdvisor that contains 84968 reviews about 1930 hotels, and another dataset from Venere that contains 52266 reviews about 1845 hotels. The data was collected from four cities in Europe: Munich, Berlin, Rome and Milan. The datasets are not publicly available.

To evaluate CSCB, Levi et al. run a user study using an online evaluation methodology similar to (Hayes & Cunningham 2002). In the study, CSCB was compared against Venere and TripAdvisor. The system was evaluated by 150 users.

In each experiment, the user is asked for the intent of her trip, her nationality, her preferences towards the aspects and a price range. Then the user is presented with a list of six hotels. Some of the hotels are recommended by CSCB and others are recommended by the other systems. For each of the hotels, the user is asked “Would you select this hotel?” and is also asked to provide a rating for the recommended hotel. Results show that 60.2% of the recommendations provided by CSCB are accepted by the users compared to 50.8% of the recommendations provided by the other systems.

Summary The main advantage of this work is that features and aspects are extracted in an unsupervised fashion. These aspects are then integrated into the recommendation model leading to increased user satisfaction.

We note that Levi et al. have referred to the nationality and intent group as contextual information, but in our view the nationality is not contextual. Therefore, we did not include this review as part of the context-aware approaches (Section 2.3).

Many things can be improved. The first is that it is hard to pin down the effect of context in this work because Levi et al. have a rather questionable definition of what context is. They define a personal characteristic of a user –the nationality– as a contextual dimension. If we more narrowly, treat contexts as the circumstances of item consumption, then the nationality of a user, which is something that does not change (often) over time, should not be considered as a contextual dimension, even if it has an influence on how a user perceives an item.

Another disadvantage is that user intent has a pre-defined and rather limited set of values (constrained by the datasets used in the evaluation). The only permissible values are for the following five scenarios: family, couple, group, single and business. We understand that this information is obtained from the datasets used, but in the real world, intent is wider than just those five options. Other models such as (Chen & Chen 2015) have successfully extracted contextual situations from user reviews.

We also consider that the way aspects are created can be improved. The fact that one feature can only belong to one aspect is a serious drawback, since one feature can be used in different contexts. For instance, the feature "swimming pool" can be used in the context of spa, but also in the context of summer family holidays with kids. Having equal importance for all the features within the same aspect is also a problem, since one feature may represent an aspect better than another one. Both problems can be solved by removing the constraint of having only one feature per aspect and assigning weights to each feature within a cluster. This type of approach is commonly used in topic modeling, where a word has a probability of belonging to a topic, and each word belongs to all of the topics with a different probability.

Finally, the CSCB is never compared against state-of-the-art methods. CSCB is compared against the TripAdvisor.com and Venere.com websites, but we do not have any details about how the recommender systems of those websites are implemented. A better designed user study would have compared their implementation with at least one state-of-the-art method used as a baseline. In the evaluation only an aggregate number of the performance of the other systems is provided. Levi et al. state that the user satisfaction for the other systems is 50.8%. It would be interesting to know what was the user satisfaction for the recommendations provided by TripAdvisor and Venere individually.

2.2.2.2 Opinionated Product Recommender

Goal The goal of (Dong et al. 2013, 2014, 2016) is to build a recommender system that exploits the information contained in user reviews to make product recommendations. The resulting recommender system is called an Opinionated Product Recommender (OPR). The goal of OPR is to extract both features and sentiment to provide recommendations for a query-based system.

Contribution OPR demonstrates that combining feature similarity between items and sentiment can lead to better recommendations. OPR succeeds at extracting item features from user-generated reviews, allocating sentiment to those features and then finding similar items given the item features. This approach can be used to find items that have higher sentiment scores across all of the features of a query item.

Differences with other approaches Along with CSCB, OPR is one of the first approaches to exploit the information contained in user-generated reviews to provide recommendations. But, different from CSCB, OPR does not need the user to provide personal information. Nevertheless, the model requires a query item in order to provide recommendations. OPR also differs from CSCB in that the provided recommendations are not personalized: they are not made for specific users; they are made using a query product as a seed.

Notation The OPR model uses the following notation:

Table 2.13: Notation of OPR.

Symbol	**Description**
i	An item
τ	A review
s	A sentence
f	A feature
F_i	The set of features that compose item i
Υ_i	The set of reviews made for item i

Model Dong et al. present a recommender system called OPR that extracts features and sentiments from user reviews, uses them to generate cases to include them in a case-based reasoning model, and uses that model in order to make better Top-N recommendations (2016).

Dong et al. preprocess all the reviews and assign part-of-speech tags to each word. They identify two types of features: *bi-grams* and *single nouns*. The bi-grams they consider are in the form adjective-noun (AN) (e.g. *hard drive, wide angle*) or noun-noun (NN) (e.g. video mode). For the AN form, bi-grams whose adjective is a sentiment word are discarded to avoid including ANs that are actually a single-noun feature accompanied by an opinion (e.g. *excellent, awful*). Single nouns that are rarely associated with sentiment words are unlikely to describe item features (Hu & Liu 2004) and, for that reason, they are excluded from the single noun features.

The sentiment of a feature is assigned at a sentence level. For each feature f in a sentence s, OPR looks for any sentiment words. In case there are none, they assign f a neutral score, but in case there are, they take the closest sentiment word w_{min} to f and assign f the sentiment of w_{min}.

There are a couple of exceptions in which the sentiment can be changed. In the first scenario, if there are any negation words within a 4-word distance of w_{min}, then the sentiment is inverted. The second exception is based on the *opinion pattern technique* introduced by (Moghaddam & Ester 2010). This technique identifies POS tags patterns between a sentiment word w_{min} and a feature f. For example, the sentence *"the speaker has great audio quality"* has a pattern JJ-FEATURE (where JJ stands for adjective). Here *"great"* is the adjective and *"audio quality"* is bi-gram feature. The frequency for every pattern that appears is counted and then the patterns that are below a certain frequency threshold are eliminated. For each occurrence of the eliminated patterns, neutral sentiments are assigned.

The next step consists of generating experiential item cases to add to a case base. Each item case is composed of the features of the item $f \in F_i$, the sentiment associated to each of the features and the popularity of each of the features for that item. Features that are mentioned in less than 10% of the reviews are removed.

$$sentiment(i) = \frac{positives(f,i) - negatives(f,i)}{positives(f,i) + negatives(f,i) + neutral(f,i)}, \tag{2.43}$$

$$popularity(f,i) = \frac{|\{\tau \in \Upsilon_i : f \in \tau\}|}{|\Upsilon_i|}, \tag{2.44}$$

$$case(i) = \{[f, sentiment(f,i), popularity(f,i)] : f \in F_i\}, \tag{2.45}$$

where Υ_i is the set of reviews made for item i, and $positives(f,i)$, $negatives(f,i)$ and $neutral(f,i)$ are functions that count the number of times feature f has a positive, negative or neutral sentiment associated with it in the reviews for item i, respectively.

Once the cases have been built, recommendations can be made. There are three ways of doing this: using a content-based recommendation strategy, exploiting the information about the sentiment of the features, and using a hybrid approach.

The content-based recommendation strategy uses the cosine metric to measure the similarity between a query item i and a candidate item i' in the case base based on the popularity levels of each of the items' features, like this:

$$similarity(i,i') = \frac{\sum_{f \in F_i \cup F_{i'}} popularity(f,i) \cdot popularity(f,i')}{\sqrt{\sum_{f \in F_i} popularity(f,i)^2} \cdot \sqrt{\sum_{f \in F_{i'}} popularity(f,i')^2}} \tag{2.46}$$

In (Dong et al. 2013), Dong et al. also attempted to measure the similarity based on the sentiment by replacing popularity by sentiment in equation 2.46.

For the sentiment based recommender, the improvement of the feature f of a candidate item i' over a query item i is calculated by using the following formula:

$$better(f,i,i') = \frac{sentiment(f,i') - sentiment(f,i)}{2} \tag{2.47}$$

Subsequently they calculate an overall 'better' score by averaging the scores of all the features that i' and i have in common:

$$b1(i,i') = \frac{\sum_{f \in F_i \cap F_{i'}} better(f,i,i')}{|F_i \cap F_{i'}|} \tag{2.48}$$

An alternative formula includes all the features of i' and sets the sentiment value of non-shared features to zero:

$$b1(i, i') = \frac{\sum_{f \in F_i \cup F_{i'}} better(f, i, i')}{|F_i \cup F_{i'}|} \tag{2.49}$$

Finally, one last recommendation method is proposed, this method merges both content-based and sentiment-based approaches:

$$score(i, i') = (1 - \lambda) \cdot similarity(i, i') + \lambda \cdot \left(\frac{sentiment(i, i') + 1}{2} \right), \tag{2.50}$$

where λ controls the weight given to the case-based part versus the sentiment part. Note that Dong et al. add 1 to the right part of the equation and divide by 2. This is done to normalize its values, because the $sentiment$ function returns values in the range $[-1, 1]$ whereas the $similarity$ function returns values in the range $[0, 1]$.

Evaluation To evaluate OPR, Dong et al. used six datasets from six different types of devices: cameras, GPS, laptops, phones, printers and tablets. All of the reviews were collected from Amazon. The datasets are not publicly available. A summary of the datasets can be seen in Table 2.14.

Table 2.14: Datasets used to evaluate OPR.

Category	# Reviews	# Products
Cameras	9355	103
GPS	12155	119
Laptops	12431	314
Phones	14860	257
Printers	24369	233
Tablets	17936	166

Dong et al. propose a way of evaluating the recommendations by comparing their recommendations against Amazon's recommendations. To achieve this, an item is taken as the seed and Amazon will suggest other items based on the seed item. OPR then tries to find items that are equally similar or more similar to the seed item (the similarity being measured based on the mined features) and have higher overall ratings. Results show that indeed OPR is able to find items that are equally or more similar to the ones that Amazon suggested and have higher overall ratings. In (Dong et al. 2014), the model is also tested using a TripAdvisor dataset having similar results.

Summary OPR is able to extract item features from reviews and link the features with sentiments in order to provide recommendations for similar items. One strength is that OPR discards nouns that do not relate to sentiments, leaving only features that users qualify with sentiments. It also considers negation words. Another strength is that it measures the score for each feature of the item. This allows it to recommend another item that has higher scores in all of the features or in the ones that are important for the user. The feature extraction is an unsupervised process. Without the need for explicit item descriptions, this approach is capable of finding similar items by mining item features from the reviews.

A disadvantage of OPR is that it gives the same weight to all sentiment words. For example, the phrases “slow service” and “horrible service” would have the same sentiment score; “good screen resolution” and “amazing screen resolution” would also have the same score. The formulae used in OPR do not consider the distance between the feature and the sentiment word in the sentence, which seems questionable. An obvious disadvantage of OPR is that the recommendations are not personalized; a seed item is needed to make recommendations. As a matter of fact, the user is never included as part of the model. Finally, the major disadvantage of OPR is that it does not incorporate ratings information into the recommendation model, losing a large amount of valuable information that has proven to be useful for recommending items.

On the evaluation side, one major disadvantage is that the results are compared in relation to Amazon’s performance, as if Amazon’s recommendations were the ground truth. This is a very unusual way to present recommendation results. The real ground truth are the users’ preferences, not Amazon’s predictions. This way of evaluating does not permit to know if the suggestions of this model are better than Amazon’s suggestions.

2.2.2.3 Multi-Criteria Aspect-Based Sentiment Analysis

Goal The goal of (Musto et al. 2017) is build a multi-criteria recommender system that extracts information from user reviews in order to have a multi-faceted representation of the users’ interests. To achieve this goal, they present the Multi-Criteria Aspect-Based Sentiment Analysis (MCABSA) recommender system.

Contribution The contributions of (Musto et al. 2017) are that the MCABSA recommender is able to extract the aspects from the reviews in an unsupervised way. Furthermore, they are able to identify subaspects, which gives MCABSA a finer-grained representation of the users' interests. MCABSA is able to outperform several state-of-the-art techniques in terms of prediction accuracy.

Differences with other approaches Similarly to the OPR model presented in (Dong et al. 2013, 2014, 2016), MCABSA is able to extract aspects (which are called features in OPR) from reviews in an unsupervised way. Also similarly to OPR, MCABSA assigns sentiment scores to the extracted aspects (or features) using the information contained in the reviews. Differently from OPR, MCABSA incorporates the ratings into the recommendation model, making it possible to produce personalized recommendations. Unlike OPR and CSCB, MCABSA is also able to predict ratings; the former two models are only useful to predict rankings of items.

Notation The MCABSA model uses the following notation:

Table 2.15: Notation of MCABSA.

Symbol	**Description**
u	A user
i	An item
$r_{u,i}$	The rating that user u gave to item i
w	A word
a	An aspect
$\varsigma_{a,u',i}$	The sentiment score that user u has for aspect a of item i
$\mathcal{S}$	A corpus containing a collection of documents

Model Musto et al. present MCABSA, a recommender built on top of the Sentiment Aspect-Based Retrieval (SABRE) framework (Caputo et al. 2017). This framework takes a list of documents and extracts a list of aspects and the sentiments associated with those aspects. Having that, Musto et al. use a k-NN-based algorithm to make rating predictions.

SABRE extracts aspects from documents based on the idea that different words are used when describing general topics from those used to describe more specific topics (Caputo et al. 2017). The goal is to identify the aspects that have a different distribution within a specific collection of documents, such as a dataset of hotel reviews, compared to the aspects in a general language dataset, such as the British National Corpus[5]. Two types of aspects are extracted, called main aspects and sub-aspects, that follow a hierarchical relation. Main aspects are more general, whereas sub-aspects are more specific and belong to a main aspect.

To find the aspects that are relevant to a specific domain, Caputo et al. first perform part-of-speech tagging on the documents and then consider only nouns as the candidate words for aspects. Subsequently, they calculate the *Kullback-Leiber divergence* to find out how relevant the word w is for the specific domain. To calculate the pointwise Kullback-Leiber divergence (giving the relevance of a word), they use:

$$\delta_w(\mathcal{S}_a \| \mathcal{S}_b) = p(w, \mathcal{S}_a) log \frac{p(w, \mathcal{S}_a)}{p(w, \mathcal{S}_b)}, \tag{2.51}$$

where w is a word and $\mathcal{S}_a$ and $\mathcal{S}_b$ are two corpora. In this work, $\mathcal{S}_a$ is the reviews corpus and $\mathcal{S}_b$ is the British National Corpus. $p(w, \mathcal{S}_a)$ can be calculated as the count of times that word w appears in corpus $\mathcal{S}_a$, divided by the total count of words:

$$p(w, \mathcal{S}) = \frac{freq(w)}{\sum_{w' \in \mathcal{S}} freq(w')}. \tag{2.52}$$

When the relevance δ_w of a word w for corpus $\mathcal{S}_a$ is higher than a given threshold ϵ, the word is labeled as a main aspect; otherwise, the word is discarded. In this way, every review τ will contain a set of main aspects $a_{\tau,j} \in A$, where $a_{\tau,j}$ is the j-th main aspect of review τ, and A is the set of all main aspects in the corpus.

To extract the sub-aspects, Caputo et al. calculate the *phraseness* and *informativeness* of each word that appears in a review τ, as given by (Tomokiyo & Hurst 2003). Then, for each word the phraseness and informativeness are aggregated together into a variable called relevance weight, denoted by φ. For each main aspect $a_{\tau,j}$, they go through all the words w_k (the nouns in this particular case, even the ones labeled as main aspects) in the review τ and calculate the rele-

[5]http://www.natcorp.ox.ac.uk/

vance weight φ. If φ is greater than a threshold γ, then the word w_k is labeled as a sub-aspect of $a_{\tau,j}$; otherwise, w_k is discarded.

Once the main aspects and sub-aspects have been extracted, they are assigned a sentiment score by a lexicon-based algorithm that uses the AFINN wordlist (Musto et al. 2014). Thus, for every aspect (both main aspects and sub-aspects), a sentiment score $\varsigma_{a,\tau}$ for aspect a in a review τ is obtained.

Having the sentiment score, a k-NN rating prediction is made. First the similarity between two users is measured as the inverse of the distance, which is calculated as:

$$dist(u,u') = \frac{1}{|I(u,u')|} \cdot \sum_{i \in I(u,u')} overall\_dist(r_{u,i}, r_{u',i}) \tag{2.53}$$

where $I(u,u')$ are the items that u and u' have rated jointly and $r_{u,i}$ is user u's rating for item i. Then, the overall distance is calculated for the n aspects mentioned in both reviews, as in (Adomavicius & Kwon 2007), like this:

$$overall\_dist(r_{u,i}, r_{u',i}) = \sqrt{\sum_{a=1}^{n} |\varsigma_{a,u,i} - \varsigma_{a,u',i}|^2} \tag{2.54}$$

where $\varsigma_{a,u,i}$ is the sentiment score for aspect a in the review about item i made by user u. Finally, the rating prediction is made by using the *weighted sum approach* as in (Adomavicius & Kwon 2007), like this:

$$\hat{r}_{u,i} = \sum_{u' \in \hat{U}} \frac{sim(u,u') \cdot r_{u',i}}{|sim(u,u')|} \tag{2.55}$$

Evaluation To evaluate MCABSA, Musto et al. use three datasets from three different platforms: Yelp, TripAdvisor and Amazon. The Yelp dataset contains 229906 reviews and ratings made by 45981 users about 11537 items. The TripAdvisor dataset contains 796958 reviews and ratings made by 536952 users about 3945 items. The Amazon dataset contains 1324759 reviews and ratings made by 826773 users about 50210 items. MAE is used to evaluate the performance of MCABSA, and it is shown to have better performance than Matrix Factorization approaches.

Summary This work has a few advantages, such as the fact that the unsupervised extraction of aspects and sub-aspects is novel. Regarding the evaluation, the approach is tested using three different publicly available datasets and `RiVaL` (Said & Bellogín 2014) is used, which brings confidence to the results. For all of the datasets, they have a better performance in terms of MAE than other state-of-the-art approaches.

2.2.3 Summary

In this section, we have reviewed seven review-based recommender systems; four of them are generative models (McAuley & Leskovec 2013, Ling et al. 2014, Diao et al. 2014, Rossetti et al. 2013) and three of them are feature extraction models (Levi et al. 2012, Dong et al. 2016, Musto et al. 2017). A classification of these seven review-based recommenders can be seen in Figure 2.2.

All of the four generative models are probabilistic models that use or are derived from the LDA model (Blei et al. 2003). Only (Diao et al. 2014) does not constrain the number of latent factors to be the same as the number of topics. On the other side, all of the heuristic models are focused on extracting features and assign scores to those features by identifying sentiment words and looking at the distance of the sentiment words to the features. The main properties of the models reviewed in this section are summarized in Table 2.16.

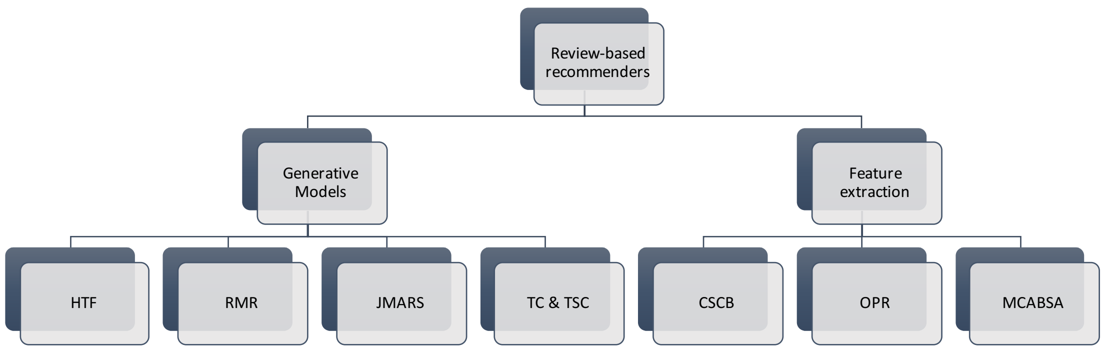


Figure 2.2: Classification of review-based recommenders

On the evaluation side, as we can see in Table 2.16, these recommender systems fix one of the most common problems present in the evaluation methodology of CARS, the lack of real-world data. All seven used real-world datasets in their evaluation. However, one problem regarding the evaluation methodology of all of the recommenders, is that only one of them evaluates its performance using Top-N metrics.

Table 2.16: Summary of the unsupervised review-based recommenders that we have reviewed.

Model	**Type**	**Topic-factor**	**Feature/aspect**	**Sentiment**	**Evaluation**		**Real-world**
		Independence	**extraction**	**analysis**	**Rating**	**Top-N**	**data**
HFT	Generative				✓		✓
RMR	Generative				✓		✓
JMARS	Generative	✓	✓	✓	✓		✓
TC & TSC	Generative			✓	✓		✓
CSCB	Heuristic		✓	✓			✓
OPR	Heuristic		✓	✓		✓	✓
MCABSA	Heuristic		✓	✓	✓		✓

In the next section we will review two approaches that are context-aware but also are able to extract information from user-generated reviews. These are the closest approaches to what we want to achieve with our own research.

2.3 Review-Based Context-Aware Approaches

This section explores the proposals closest to what we want to achieve: context-aware recommendations that exploit consumer reviews. We take a look at the papers that merge these two concepts with the intention of extracting new ideas and identifying the gaps that will lead us to produce a novel model.

2.3.1 Context Discovery

Goal The goal of (Bauman & Tuzhilin 2014) is to develop a method to discover contextual information from user-generated reviews. This contextual information can then be used to provide better recommendations. The model that Bauman & Tuzhilin introduced is called Context Discovery.

Contribution The contribution of (Bauman & Tuzhilin 2014) is that it presents two methods for the extraction of contextual information. The first method is based on word semantics and the second one is based on topic modeling. Bauman & Tuzhilin state that these methods are complementary. The most important contribution of this paper is that both methods are unsupervised and do not require a set of pre-defined keywords to extract the contextual information, which most CARS do.

Differences with other approaches Although the ultimate goal of this work is to produce better recommendations, the paper focuses on the extraction of contextual information from reviews, leaving open the question of how this might be used in a recommender system. Having said that, this method is different from all of the proposals that we reviewed in Section 2.1 because it does not need the contextual information to be pre-defined. The way the context is extracted is totally unsupervised. It is also different from (Levi et al. 2012) and (Chen & Chen 2015) because it does not requires a list of manually defined words. By contrast, the work in (Levi et al. 2012) requires a list of seed adjectives for the sentiment words and the work in (Chen & Chen 2015) requires a list nouns for the aspects. In that sense, Context Discovery is more similar to (Dong et al. 2016) and (Musto et al. 2017), both of which do not require pre-defined keywords.

Notation The Context Discovery model uses the following notation:

Table 2.17: Notation of Context Discovery.

Symbol	Description
u	A user
i	An item
w	A word
t	A topic
τ	A review
W_τ	The set of words that belong to review τ
T_τ	The set of topics that belong to review τ

Model Bauman & Tuzhilin propose a method called Context Discovery to extract contextual information from reviews. The goal of this paper, unlike the ones described up until now, is not to produce recommendations, but to discover contextual information from user-generated reviews. They propose two methods to achieve this, a word-based approach and a LDA-based approach. Both methods are based on the assumption that there are two types of reviews: *specific* and *generic*. Specific reviews describe detailed experiences about visits to a place (hotel, restaurant, spa, etc.) whereas generic reviews give short impressions of the places.

Based on this, the first step in the proposed model is to separate specific from generic reviews. To do this, Bauman & Tuzhilin use the well-known k-means clustering method with $k = 2$, placing specific reviews in one of the clusters and the generic ones in the other. To make the clustering possible, certain features of the reviews are extracted. For this paper in particular, the number of sentences, the number of words, the number of verbs, the number of verbs in the past-tense and the ratio between the number of verbs and the number of verbs in the past-tense are used. To all of them the logarithm function is applied.

Once the reviews have been clustered, the word-based or the LDA based approach can be used to extract the contextual information.

For the word-based approach the words that are nouns in the reviews are identified and, for each word, the weight of that word among specific reviews $\varphi^s(w)$, among generic reviews $\varphi^g(w)$ and among all reviews $\varphi(w)$ are calculated. The weight is calculated by counting the frequency of w among that type of review (specific, generic or all) and dividing it by the total count of that type of review. In other words:

$$\varphi^{type}(w) = \frac{|\tau : \tau \in type \text{ and } w \in W_\tau|}{|\tau : \tau \in type|}, \tag{2.56}$$

where τ is a review and W_τ is the set of words that belong to review τ.

Once this is done, the words w that have an overall frequency lower than a certain threshold α ($\varphi^{all}(w) < \alpha$) are removed. In (Bauman & Tuzhilin 2014), α was set to 0.005. For each word w, the ratio between specific and generic weights is also calculated as $ratio(w) = \frac{\varphi^s(w)}{\varphi^g(w)}$ and words with a ratio lower than a threshold β ($ratio(w) < \beta$) are removed. In (Bauman & Tuzhilin 2014), β was set to 1.0.

For each of the remaining words, a set of senses is found using WordNet. The set of senses is the set of meanings that a word can have; for instance a bank could mean the place where people store money or the land alongside of a river. Then WordNet is used to determine the senses which have very close meanings and they are grouped together. Words with several distinct meanings can be represented in several distinct groups. Then for each group g the frequency weight $\varphi^{type}(\mathbf{g})$ is calculated as:

$$\varphi^{type}(\mathbf{g}) = \frac{|\tau : \tau \in type \text{ and } \mathbf{g} \cap W_\tau \neq 0|}{|\tau : \tau \in type|} \tag{2.57}$$

Then the ratio between specific and generic reviews is calculated for each group g as $ratio(\mathbf{g}) = \frac{\varphi^s(\mathbf{g})}{\varphi^g(\mathbf{g})}$. Finally the groups are sorted by $ratio(\mathbf{g})$ and presented in descending order. Groups that have a higher ratio are supposed to contain more contextual information than the ones that have a lower ratio.

For the LDA-based approach, a topic model is built using LDA on only the specific reviews. Then the topic model is applied to all of the reviews. This results in every review being represented by a stochastic vector of size k, where k is the number of topics. Each cell of that vector will contain the probability of the topic t for the given review τ, $p(t|\tau)$. For each review τ, the topics with probability lower than threshold ϵ are discarded $p(t|\tau) < \epsilon$.

The result of the word-based approach is a list of words that are sorted by ratio in descending order.

Similarly to the word-based approach, in the LDA-based approach, the weights of the topics t are calculated like this:

$$\varphi^{type}(t) = \frac{|\tau : \tau \in type \text{ and } t \in T_\tau|}{|\tau : \tau \in type|} \tag{2.58}$$

Then the topics with overall frequency lower than α are removed $\varphi^{all}(t) < \alpha$. Subsequently, and similarly to the word-based approach, the ratio of the topics is calculated as $ratio(t) = \frac{\varphi^s(t)}{\varphi^g(t)}$ and then topics with a ratio lower than a threshold β are removed $ratio(t) < \beta$. As in the word-based approach, $\alpha = 0.005$ and $\beta = 1.0$. The value of ϵ is not disclosed. Finally, the topics are sorted by ratio and are presented in inverse order. The topics with higher values of $ratio(t)$ are supposed to contain more contextual information than the ones with low values of $ratio(t)$.

The result of the LDA-based approach is a list of topics that are sorted by ratio in descending order.

Evaluation To evaluate Context Discovery, Bauman & Tuzhilin use three datasets from Yelp that were provided for the ACM Conference on Recomender

Systems Challenge 2013. They contain reviews and ratings for restaurants, hotels and beauty & spas. The restaurants dataset contains 158430 reviews by 36473 users about 4503 items. The hotels dataset contains 5034 reviews by 4184 users about 284 items. The beauty & spas dataset contains 5579 reviews by 4272 users about 764 items. To evaluate the performance of the clustering, they manually labeled 300 reviews as specific or generic for each of the datasets (900 in total). Using the clusters, they were able to correctly identify the reviews in 89%, 88% and 90% of the cases for the hotels, restaurants and beauty & spas datasets, respectively.

To evaluate the word-based approach, the resulting list of words sorted by ratio in descending order is taken. Bauman & Tuzhilin manually identify the first occurrence of a contextual word and assign it to a contextual category that has been previously defined. They show that for most of the categories, the algorithm has extracted contextual words related to each category. A similar approach is used to evaluate the LDA-based approach. However, Bauman & Tuzhilin did not compare the Context Discovery method against any other method.

Summary Context Discovery extracts contextual information from user-generated reviews. The biggest advantage of Context Discovery, for both the word-based and LDA-based methods, is that it does not require a pre-defined set of contextual dimensions or contextual keywords; it learns the context. However, the model has several drawbacks. First of all, it cannot produce recommendations by itself. It does not address the problem of topic modeling that, when a topic model is trained twice on the same data with different random seeds, the results can be very different, which lead to results that are not easy or impossible to replicate. And finally, it does not incorporate ratings information into the model.

Regarding the evaluation methodology, the performance metrics presented are too vague and virtually impossible to reproduce. Finally, Bauman & Tuzhilin do not compare Context Discovery against any system, not even a random baseline.

2.3.2 Contextual Opinions Recommender

Goal The goal of the model presented in (Chen & Chen 2015) is to introduce a recommender with improved performance by obtaining the aspect-level contextual preferences from user-generated reviews and ratings.

Contribution Contextual Opinions Recommender (COR) is able to extract item aspects from reviews and relate contextual situations directly to aspects. This gives a more fine-grained representation of the users' preferences. We believe that COR is the first approach to model aspect-context preferences.

Differences with other approaches Similarly to the OPR model proposed in (Dong et al. 2016) and to the MCABSA model proposed in (Musto et al. 2017), COR extracts aspects (or features) from user-generated reviews. But differently from the two former approaches, COR requires a list of keywords to determine the aspects that are part of a review. As mentioned earlier, this approach is unique in the sense that it models the aspect-context preferences of users, and it uses context-dependent and context-independent preferences to produce Top-N recommendations.

Notation The COR model uses the following notation:

Table 2.18: Notation of COR.

Symbol	Description
u	A user
i	An item
$\tau_{u,i}$	The review that user u wrote about item i
s	A sentence
a	An aspect
f	An aspect-related term
op	An opinion word
k	A contextual condition
$\tau_{u,i}$	The review that user u wrote about item i
ς_{op}	The sentiment score associated with opinion word op
K	A set of contextual conditions
$\lvert A\rvert$	The number of aspects
$\mathbf{k}$	A vector of contextual values
$\boldsymbol{\varphi}_u$	A vector containing the weights that user u gives to each aspect

Model The approach presented in (Chen & Chen 2015) can be presented in four parts. The first part deals with the extraction of context from the user-generated reviews. In the second part, the context-independent preferences of the users are inferred. In the third part, the context-dependent preferences are inferred. The final part is the ranking and recommendation part, where the context-independent and the context-dependent preferences are combined to produce recommendations.

Part I: Extracting context To extract aspect-related contextual opinions, Chen & Chen propose a method composed of four steps. The first step deals with the identification of aspects and uses a heuristic-based technique based on bootstrapping to extract the aspects from the reviews. In this technique, a number of aspects is first defined and then each aspect is assigned a set of terms as seed words. Then, candidate terms are searched and selected by measuring the dependency between a candidate and a seed term using the chi-square statistic as proposed in (Yang & Pedersen 1997). Only nouns and noun phrases are considered as candidate terms. As a result, a set of aspect-related terms are obtained. An aspect-related term is a word that is directly correlated with an aspect. For instance, the aspect "service" of a restaurant can have aspect-related terms like "service", "staff", "waiter", etc.

The second step extracts the opinion orientation for each aspect. This opinion orientation is extracted by looking at adjectives. Using an opinion lexicon, each adjective is assigned a numeric score of $+1$ or -1 depending on whether it is positive or negative. Then, for each aspect-related term f, and each sentence s, a score is calculated as $score(s, f) = \sum_{op \in s} \varsigma_{op} / dist(op, f)$, where op is an opinion word in a sentence, ς_{op} is the sentiment score of op and $dist(op, f)$ is the number of words between opinion word op to aspect f plus one. Additionally, when a negation word is found in the sentence or a "but" is found, then the opinion score is inverted.

The third step extracts the context at a sentence level. To do this, a set of keywords are defined for every contextual situation. If any of the keywords of a contextual situation are present in a sentence, then that sentence is labeled with the corresponding context value.

The fourth and last step relates aspects with context, which is done in two ways. First, if an aspect-level opinion and a context are together in the same sentence, then they are related. If there is no context in the sentence, then the aspect level opinion is related to the context of the nearest previous sentence. Then for each context, all the opinions of an aspect that are related to that context are summed.

Part II: Inferring context-independent preferences The context-independent preferences are the preferences of the user towards the aspects, and are represented by $\boldsymbol{\varphi}_u = \langle \varphi_{u,1}, \ldots \varphi_{u,|A|} \rangle$, which is a vector that contains the weights that user u gives to each aspect. To infer the context-independent preferences, Chen & Chen introduce two variants, a Linear Regression Model (LRM) based model and Probabilistic Regression Model (PRM) based model.

In LRM the ratings are modeled as a weighted function of the aspects, like this:

$$r_{\tau_{u,i}} = \boldsymbol{\varphi}_u^\top \mathbf{r}_{u,\tau_{u,i}} + \varepsilon \tag{2.59}$$

where $\mathbf{r}_{u,\tau_{u,i}} = \langle r_{a_1}, \ldots, r_{a_{|A|}} \rangle$ is a rating vector, in which $r_{a_j}(1 \leq j \leq |A|)$ represents the opinion that user u assigned to aspect a_j in review $\tau_{u,i}$, $|A|$ is the number of aspects, and ε denotes the error term. In this paper, linear least-square regression is used to infer the context-independent preferences.

Similarly to LRM, PRM models the relation between the rating and the aspects' opinions as a regression problem. PRM models this as a Bayesian problem in order to incorporate prior knowledge. ε is drawn from a Gaussian distribution with mean 0 and variance σ^2: $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. The overall rating is treated as a sample drawn from a Gaussian distribution with mean $\boldsymbol{\varphi}_u^\top \mathbf{r}_{u,\tau_{u,i}}$ and variance σ^2, like this:

$$p(r_{\tau_{u,i}} | \boldsymbol{\varphi}_u, \mathbf{r}_{u,\tau_{u,i}}) = \mathcal{N}(r_{\tau_{u,i}} | \boldsymbol{\varphi}_u^\top \mathbf{r}_{u,\tau_{u,i}}, \sigma^2) \tag{2.60}$$

Chen & Chen solve the PRM model using the expected maximization algorithm.

Part III: Inferring context-dependent preferences To obtain the context-dependent preferences, the focus is turned on learning the aspect-context weights $\varphi_{a,k}$. It is very important to note that the context-dependent preferences are not user-specific. The learning of the aspect-context weights is based on the observation that if an aspect appears more frequently than others in reviews that belong to a certain context, then this aspect should have a higher weight because it is more important than the others. Based on this, the following formula is introduced:

$$freq_{a,k} = \frac{\sum_{\tau \in \Upsilon} \sum_{s \in \tau} \Delta_{s,k} \cdot \left(\sum_{f \in s} \Theta_{f,a} \right)}{\sum_{\tau \in \Upsilon} \sum_{s \in \tau} \Delta_{s,c} \cdot \left(\sum_{f \in s} 1 \right)} \tag{2.61}$$

where f is an aspect-related term, s is a sentence and τ is a review, Υ denotes the set of all reviews, $\Delta_{s,k}$ is an indicator function whose value is 1 when the sentence s is related to context k and 0 otherwise, and $\Theta_{f,a}$ is an indicator function whose value is 1 if the term f is related to aspect a and 0 otherwise. Then the aspect's average frequency is calculated as $avg_a = \sum_{k \in K} freq_{a,k}/|K|$ and the standard deviation as $stdv_a = \sqrt{\sum_{k \in K}(freq_{a,k} - avg_a)^2/|K|}$, where K is the set of context values and $dev_{a,k} = freq_{a,k} - avg_a$. Finally the aspect-context weight $\varphi_{a,k}$ is calculated using the following equation:

$$\varphi_{a,k} = \begin{cases} 1, & \text{if } |dev_{a,k}| < stdv_a \\ Max\left(0.1, 1/\left|\frac{dev_{a,k}}{stdv_a}\right|\right), & \text{if } \frac{dev_{a,k}}{stdv_a} < -1 \\ Min\left(3, \frac{dev_{a,k}}{stdv_a}\right), & else \end{cases} \tag{2.62}$$

Chen & Chen note that equation 2.61 does not distinguish the relative importance of the aspect-related term in different contexts, as users may assign different importance to an aspect-related term depending on the context. To overcome this, they try out three techniques from the text categorization field. In summary, they replace the $\Theta_{f,a}$ function by functions that calculate mutual information, information gain and the chi-square statistic.

Part IV: Producing recommendations Once both context-independent and context-dependent weights have been obtained, recommendations can be produced. The following equation is used to calculate the score that the target user

u would give to item i based on the review $\tau_{u',i}$:

$$score(u,\tau_{u',i},\mathbf{k}) = \overbrace{\sum_{\langle i,\tau_{u',i},a,Con_{u',i,a}\rangle \in S(\tau_{u',i})}}^{\text{(a) iterate through context-opinion tuples}} \overbrace{\left(\prod_{k\in\mathbf{k}}(1+\alpha_{a,k}\cdot\varphi_{a,k})\right)}^{\text{(b) calculate context-dependent preferences}}$$
$$\cdot \underbrace{\varphi_{u,a}\cdot\varsigma_a\cdot g(Con_u, Con_{u',i,a})}_{\text{(c) calculate context-independent preferences}} \tag{2.63}$$

where $S(\tau_{u',i})$ is the set of contextual opinion tuples derived from review $\tau_{u',i}$, ς_a is the opinion score of aspect a contained in the contextual opinion tuple $\langle i,\tau_{u',i},a,Con_{u',i,a}\rangle$, $\mathbf{k}$ is a vector that contains the current target context of user u, Con_u is the boolean vector form of $\mathbf{k}$, $Con_{u',i,a}$ is a boolean vector whose element value is 1 when the associated context occurs and 0 otherwise, and $g(Con_u, Con_{u',i,a})$ is an indicator function whose value is 1 if $Con_u \cdot Con_{u',i,a} \neq 0$, and 0 otherwise. In other words the g function makes sure that we are only including information that is relevant to the target context for user u. Note that to combine both types of preferences there is a parameter $\alpha_{a,k}$ that controls the contribution of each type of preferences. This parameter is learned using a stochastic gradient descent method.

To have a better understanding of Equation 2.63 we divided it in three parts. Part (a) simply iterates through all of the context-opinion tuples derived from review $\tau_{u',i}$ and sums the result. Part (b) iterates through all of the target contexts and aggregates all the context-dependent preferences by multiplying them. Finally, part (c) multiplies the preferences of user u towards aspect a by the opinion score of aspect a, ς_a, but only including instances where aspect a appeared as part of the current target context of user u.

Subsequently, to calculate the score that user u will give to item i, we take the average score of all of the reviews written about i.

$$score(u,i) = avg_{\tau_{u',i}\in R(i)}[score(u,\tau_{u',i},)] \tag{2.64}$$

In the end, for a user u, the score for all candidate items is calculated and only the top-N items with the highest scores are presented to the user.

Evaluation To evaluate COR, Chen & Chen use two datasets. The first dataset contains 357113 hotel reviews crawled from TripAdvisor, made by 30039 users about 11405 items. This dataset is not publicly available. The second dataset contains 237077 restaurant reviews made by 23152 users about 11485 items. This last dataset is from Yelp and was published as part of the ACM Conference on Recommender Systems Challenge 2013. COR is evaluated in terms of Top-N performance using Hit Ratio and Mean Reciprocal Rank as the metrics. COR is not compared against any other approaches, but the results report the performance of different variations of COR, such as using PRM compared to LRM or using different weighting methods like mutual information, information gain or chi-square statistics. Across both datasets, the PRM approach combined with the chi-square weighting method showed superior ranking prediction performance.

Summary The novelty of COR lies in that it is able to model the users' aspect-context preferences and use them to make recommendations. An advantage of COR is that both aspects of the items and the context are extracted from user reviews. It is also able to learn both context-independent preferences and context-dependent preferences. Nevertheless, it has a few disadvantages, such as the fact that the context is pre-defined as a set of keywords and aspects require a set of terms as seed words. No comparison of the model is made against any state-of-the-art recommender system, nor a random baseline.

A few things can be improved. The preferences could be modeled both at an aspect-based level and an item-based level, as is often done in matrix factorization, where a bias is learned for the items and then also the factorized parameters are learned. A very important thing to note is that Chen & Chen state that the context-dependent preferences are not user-specific. Having said this, it would be good to explore the performance of the model if context-dependent preferences are also dependent on user, thus learning weights in the form $\varphi_{u,f,a}$. Overall, this is a very good paper and it is the closest one we have found to our topic of extracting context from reviews in order to produce recommendations. Our goal, however, is to extract the context in an unsupervised way so that we remove the need for pre-defined contextual variables and pre-defined contextual keywords.

2.4 Summary

In this chapter we have explored a variety of models with the goal of identifying elements that can help us create our own context-driven recommender that is able to extract the contextual information from user-generated reviews without the need to pre-define contextual variables or contextual keywords, allowing us to have open-ended contextual information.

In Table 2.19 we have summarized the properties that we consider should be included as part of an unsupervised context-driven recommender model. Similarly, Table 2.20 contains the evaluation properties that we considered a recommender should be subject to. Here, for the first time we mention Rich-Context, our proposed recommender. As we can see in the last row of the two aforementioned tables, we make sure that Rich-Context meets all of the properties that we have considered important.

Table 2.19: Summary of the model properties of the papers that we have reviewed.

Model	**Contextual recommendations**	**Incorporates reviews**	**Learns context**	**Keyword-free**	**Prediction Type**
Multiverse	✓			✓	Rating
CAMF	✓			✓	Rating
LCMF	✓			✓	Rating
DCR	✓			✓	Rating
DCW	✓			✓	Rating
CSLIM	✓			✓	Ranking
HFT		✓		✓	Rating
RMR		✓		✓	Rating
JMARS		✓		✓	Rating
TC & TSC		✓		✓	Rating
CSCB		✓			Ranking
OPR		✓		✓	Ranking
MCABSA		✓		✓	Rating
Context Discovery		✓	✓	✓	–
COR	✓	✓	✓		Rating
Rich-Context	✓	✓	✓	✓	Rating

The work we have reviewed falls, for the most part, into one of two categories: context-aware recommendations or review-based recommendations. Only COR spans these two categories but, as we have explained before, COR requires that a set of contextual keywords be pre-defined, which prevents contextual information from becoming open-ended. Even though, Context Discovery is a hybrid between review-based and context-aware approaches, as we explained previously, it does not produce recommendations.

After reviewing all of the papers that are mentioned in this chapter, we have

Table 2.20: Summary of the characteristics of the evaluations in papers that we have reviewed.

Model	**SOTA**	**Datasets**			
	comparison	**Sparse**	**Real-world**	**Multiple**	**Publicly available**
Multiverse	✓			✓	
CAMF	✓			✓	
LCMF	✓				
DCR	✓	✓	✓	✓	
DCW	✓			✓	
CSLIM	✓			✓	
HFT	✓	✓	✓	✓	✓
RMR	✓	✓	✓	✓	✓
JMARS	✓	✓	✓		
TC & TSC	✓	✓	✓	✓	✓
CSCB		✓	✓	✓	
OPR		✓	✓	✓	
MCABSA	✓	✓	✓	✓	✓
Context Discovery		✓	✓	✓	✓
COR		✓	✓	✓	✓
Rich-Context	✓	✓	✓	✓	✓

identified a gap where we can make a contribution to the field. As we will see in the following chapters, we will borrow from many of these proposals to build Rich-Context, our own unsupervised context-driven review-based recommender.

The next chapter introduces Rich-Context and gives a general overview of its operation.

Chapter 3

Rich-Context

After reviewing the state of the art, we identify ways we can make a contribution to work on CARS, based on lifting assumptions or limitations of the existing work. Based on this, we set ourselves the goal of creating a recommender system with the following inter-related features:

Treat contexts as open-ended The recommender should be capable of capturing any type of contextual situation. Most Context-Aware Recommender Systems only focus on making recommendations on a small, pre-defined set of contextual variables, such as the purpose of a trip and the companion, and each with a small set of possible values (contextual conditions). In real life, there are more contextual variables and conditions that can affect how we perceive an item (for instance, a restaurant might be pet-friendly, birthday, girls night out, for children, etc.). Therefore, we want to capture all of these types of contextual situations and conditions.

Work in an unsupervised fashion (no need for keywords) The recommender should be able to extract all of the contextual information from the user-generated reviews without the need for a set of pre-defined contextual keywords. In other words it should be able to identify that a review is discussing a certain context (for instance, a family trip) without needing keywords that describe that context. This implies that the recommender system is able to act by itself without the intervention of an expert who defines the keywords. As we will see in Chapter 4, the only moment that human intervention is needed is at the time of labeling reviews as specific or generic to train one of the components of the recommender system.

Achieve good performance on sparse datasets The recommender should be able to work on sparse datasets. This is very important as most real-world datasets tend to be very sparse and, with the introduction of contextual variables, they become even more sparse. Approaches such as (Zheng et al. 2013) have shown good performance on dense datasets but have not been tested on sparse datasets. We want a recommender that is able to overcome the sparsity problem.

Regarding the evaluation of the recommender, we set ourselves the goal of having a recommender that performs better than the state of the art on sparse, real-world datasets. We also evaluate on multiple datasets for robustness of results. And we choose to use publicly-available datasets.

We designed and implemented a recommender system that covers all the aforementioned features. This recommender system is called Rich-Context and in this chapter we give an overview of how it works.

3.1 Overview of the Model

Rich-Context is a context-driven recommender system that extracts contextual information from user-generated reviews. Differently to CARS, context-driven recommender systems aim to produce recommendations for upcoming contextual situations, while CARS do it for the context in which the user finds herself/himself at the moment. This allows the user to plan for consumption of an item in the future (Pagano et al. 2016).

To extract the contextual information, Rich-Context works based on the principle first exposed in (Bauman & Tuzhilin 2014) that there are two types of reviews called specific and generic. Specific reviews tend to contain more contextual information than generic ones. Therefore, by identifying and separating the reviews by type we are able to extract contextual information from user reviews.

Our approach then consists of separating the reviews, extracting the context and producing recommendations. To achieve this Rich-Context is composed of

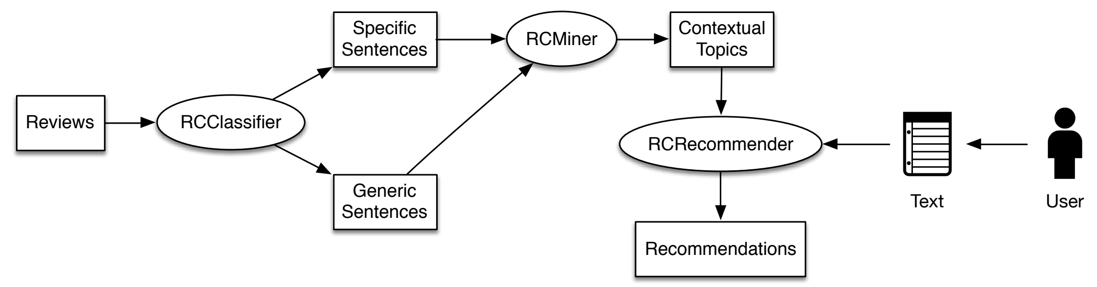


Figure 3.1: An overview of the RC system

three main components called RCClassifier, RCMiner and RCRecommender. An overview of Rich-Context can be seen in Figure 3.1.

To train Rich-Context we use a dataset composed of reviews and ratings in which each review can be seen as the justification of why that rating was given (McAuley & Leskovec 2013). Each record in the dataset $\langle\langle u, i, \tau_{u,i}\rangle, r_{u,i}\rangle$ 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, $\tau_{u,i}$, and the user's rating of the item, $r_{u,i}$.

Since Rich-Context is a context-driven recommender system the user u will have to state his/her desired context through a query q. This query is in the form of a short sentence describing the user's intent, i.e. the context in which she intends to consume the recommended item, e.g. *"birthday dinner"*. Based on this query, the recommender system will retrieve a set of candidate items.

3.2 Classifying Reviews

The goal of RCClassifier is to classify reviews as *specific* or *generic*, similarly to (Bauman & Tuzhilin 2014). Specific reviews are ones that describe experiences whereas generic reviews give an overall overview of a product or service. Our assumption is that, since there is no such thing as a context-less experience, reviews that describe experiences, i.e., specific ones, will tend to have more contextual information than generic reviews.

To classify the reviews, RCClassifier is first trained using a dataset of 300 manually-labeled reviews. RCClassifier starts by performing part-of-speech tagging on every review in the training set. Each tagged review is then represented by a vector of numeric-valued features. Finally these features are used to classify the reviews. A detailed explanation of how RCClassifier works and the description of the features is given in Chapter 4. Note that we *classify* reviews into specific and generic, whereas Bauman & Tuzhilin *cluster* reviews into two groups.

3.3 Context Extraction

Having classified the reviews as specific or generic, the goal of RCMiner is then to identify which of the information contained in the reviews is contextual and extract it. To achieve this, RCMiner relies on topic models.

RCMiner uses the ensemble topic modeling algorithm presented in Chapter 5 to build a topic model using only the specific reviews, as seen in Figure 3.1. After that, the topic model is applied to *all* of the reviews (both specific and generic) with the purpose of identifying which topics are discussed more frequently in specific reviews than in generic ones. We treat the topics discussed more frequently in specific reviews as contextual topics and the remaining ones as non-contextual. At the end of this process, each document is described by a vector that contains the weights of each of the topics discussed within the document. Since we are only concerned with the contextual topics, the remaining topic weights are set to zero. We proceed to give a little more detail on the operation of RCMiner.

RCMiner *builds* a topic model from the specific reviews 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 reviews, we expect some of these topics to be contextual. However, specific reviews 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* reviews, so that each review τ is represented by a vector of weights $\mathbf{t}_\tau$ across the k topics.

We normalise each vector $\mathbf{t}_\tau$ so that the sum of its weights equals 1.

RCMiner next determines which topics appear more frequently in specific reviews than in generic reviews, on the assumption that these are more likely to be the contextual topics. For each topic, $t \in T$, we calculate the sum of the weights for that topic in the specific reviews, divided by the number of specific reviews:

$$\varphi^s(t) = \frac{\sum_{\tau \in \textit{specific}} \mathbf{t}_{\tau,t}}{|\tau : \tau \in \textit{specific}|} \tag{3.1}$$

We make a similar calculation for generic reviews:

$$\varphi^g(t) = \frac{\sum_{\tau \in \textit{generic}} \mathbf{t}_{\tau,t}}{|\tau : \tau \in \textit{generic}|} \tag{3.2}$$

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

$$CT = \left\{ t \mid \frac{\varphi^s(t)}{\varphi^g(t)} > \beta \right\} \tag{3.3}$$

What is left to describe is the topic modeling itself. We *build* our topic models from just the nouns in the specific reviews, as these are the parts of speech that most capture contextual information.

At the end of this process, every review in the training set (irrespective of whether it is specific or generic) has been associated with a vector of length $|CT|$, $\mathbf{c}_1 \ldots, \mathbf{c}_{|CT|}$, whose values designate the affinity of the review to each of the $|CT|$ contextual topics. The original dataset of records $\langle\langle u, i, \tau_{u,i}\rangle, r_{u,i}\rangle$ is transformed to one in which reviews are represented by their corresponding contextual topic vectors, $\langle\langle u, i, \mathbf{c}_1, \ldots, \mathbf{c}_{|CT|}\rangle, r_{u,i}\rangle$.

3.4 Recommending

3.4.1 Recommendations with side-information

To produce recommendations using the newly-extracted contextual information we needed a recommendation algorithm that can support side-information. Shi et al. describes many of the recommendation algorithms that can handle side information (2014). They categorize these algorithms into two types: ones that can support extra information about the users and items, such as demographics or genres respectively; and ones that can support information about the interaction between users and items. We are more interested in the latter since context has an effect on the user-item interaction.

After exploring several candidate algorithms, in particular the neighborhood-based approaches DCW (Zheng et al. 2013) and CSLIM (Zheng, Mobasher & Burke 2014), and the factorization-based approaches Tensor Factorization (Karatzoglou et al. 2010), CAMF (Baltrunas et al. 2011), we decided to use Factorization Machines (Rendle 2010, 2012), which we describe in the next paragraphs.

A Factorization Machine is a latent-factor model in which the interaction between each pair of variables is represented by a factorized parameterization. It takes real-valued feature vectors as inputs and finds factorized interactions between variables, which enables it to make accurate predictions in environments with high sparsity (Rendle 2012).

In Factorization Machines the data for the prediction problem is assumed to be in the form of a matrix $X \in \mathbb{R}^{n \times l}$ where $\mathbf{x}_i$ is a real-valued vector with p variables describing the i-th case and where y_i is the prediction target for the i-th case. They are able to model all nested interactions up to order d between the p input variables in $\mathbf{x}$ (Rendle 2012). Factorization Machines of order two are defined as:

$$\hat{y}(\mathbf{x}) := \varphi_0 + \sum_{j=1}^{p} \varphi_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \sum_{f=1}^{k} \psi_{j,f} \psi_{j',f} \tag{3.4}$$

where l is the number of latent factors and the model parameters $\{\varphi_0, \varphi_1, \ldots, \psi_{1,1}, \psi_{p,l}\}$ are:

$$\varphi_0 \in \mathbb{R}, \qquad \boldsymbol{\varphi} \in \mathbb{R}^p, \qquad \boldsymbol{\Psi} \in \mathbb{R}^{p \times k} \tag{3.5}$$

Similarly to a linear regression model, the first part of Equation 3.4 models the unary interactions between each of the variables of x_j and the target variable.

The number of parameters of a Factorization Machine is linear to the number of predictor variables and to the number of latent factors. For our contextual recommender this means that the number of parameters is going to be linear to the number of users, items and contextual situations. The total number of parameters in our problem is $1 + |U| + |I| + |CT| + l(|U| + |I| + |CT|)$.

Unlike a polynomial linear regression, where the interactions are modeled by independent parameters $\varphi_{j,j'}$, Factorization Machines model these interactions with a factorized parametrization $\varphi_{j,j'} \approx \langle \psi_j, \psi_{j'} \rangle = \sum_{f=1}^{k} \psi_{j,f} \psi_{j',f}$. This assumes that the pairwise interactions have a low rank, which is ideal for sparse datasets such as contextual datasets.

Three things led us to choose Factorization Machines over other prediction models. First, it has support for any form of side-information, including any number of contextual dimensions. Second, it is able to learn factorized parameters, which is ideal for sparse datasets. Third, the number of parameters is linear with the number of users, items and contextual situations. None of the other reviewed models offered those three features simultaneously.

In the next section we describe how we use Factorization Machines to produce recommendations.

3.4.2 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 et al. 2014). As explained in the previous section, we opted to use Factorization Machines (Rendle 2012). For this, we one-hot encode the user and item ids, as is done in (Rendle 2012), giving training examples $\langle \mathbf{x}, r_{u,i} \rangle$ where $\mathbf{x} \in \mathbf{R}^{|U|+|I|+|CT|}$. We use Factorization Machines of order 2, which attempt to model the interactions between each variable $x_j \in \mathbf{x}$ and the dependent variable $r_{u,i}$ but also the interactions between pairs of variables $x_j x_{j'}$ and the dependent variable $r_{u,i}$. However, the interactions between pairs of variables is not modeled by one parameter per pair but by a low rank approximation, which makes Factorization Machines work well with the kind of sparse data we have in these domains.

As mentioned earlier, at recommendation time, after the Factorization Machine has been trained, we assume we have an active user u 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, $\mathbf{c}_1, \ldots, \mathbf{c}_{|CT|}$. For each candidate item i, we use the Factorization Machine to predict u's rating and we recommend the n items with highest predicted ratings. A summary of the recommendation process can be seen in Figure 3.2.

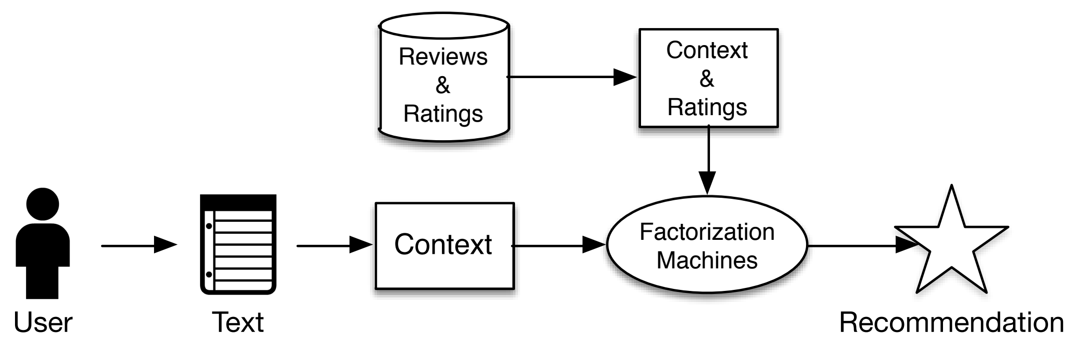


Figure 3.2: The RCRecommender component

We evaluate Rich-Context in Chapter 6 both in terms of rating prediction and ranking prediction performance. We rely on the `RiVaL` framework (Said & Bellogín 2014) to perform the evaluation. But, before that, in Chapters 4 and 5, we give more details about two of Rich-Context's more complex components: the review classification and the content extraction using topic modeling, respectively.

Chapter 4

Classification of User Reviews

As we have stated in Chapter 3, one of the goals of our work is to extract the contextual information in an unsupervised way from user-generated reviews. We want to remove human intervention in the task of deciding which is the contextual information. We do not want, e.g., to generate a dictionary of keywords as in (Chen & Chen 2015) or depend on a dataset that has fixed predefined contextual dimensions as done in (Baltrunas et al. 2011, Hariri et al. 2011, Karatzoglou et al. 2010, Zheng et al. 2012*a*, 2013, Zheng, Mobasher & Burke 2014).

Besides the benefit of automating the context definition process by removing human intervention, this is going to bring other contextual variables that were not considered before, or that were latent in the original dataset of reviews. These contextual variables could indicate things such as: is this place good for birthday celebrations, for conferences, for breakfast, take-away versus eat in, etc.

When users go to websites to write reviews about items, normally they are rating the experience they have had with the item, and writing a justification of why they gave that numeric score. The reviews that users write can be categorized into two types, specific and generic. Specific reviews are the ones that tell stories about a particular experience with an item and generic reviews are the ones that give a general description of an item. An example of a specific review is: *"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"*. In contrast, in generic reviews it is very hard to relate the review to a particular situation; for instance:

"Nice hotel, all the amenities you need, great complex of pools".

In real life, there is no such thing as a context-less experience. Every experience we undergo happens under a certain context. For instance, when we go to a hotel, we go during spring, summer, fall or winter; we go by ourselves, with our significant other, friends or colleagues; we go because of work or for holidays; our stay was planned in advance or we were about to fall asleep when driving and stopped at a motel, etc. There is always a context associated to each experience. Based on this premise, we assume that reviews that describe experiences (specific reviews) have more contextual information than reviews that do not describe experiences (generic reviews).

Our goal in this chapter is then to successfully separate specific reviews from generic ones. Once they have been separated it will be easier to mine the contextual information. To do this we will label a set of reviews as specific or generic and then train a classifier using this labeled dataset to help us identify the types of reviews.

4.1 Manual Labeling

The first part of this classification process deals with creating a dataset that serves as the ground truth to identify specific and generic reviews. To carry on with this process, we asked three volunteers to manually label 302 reviews for two datasets. We used the hotels and restaurants datasets from the ACM Conference on Recommender Systems Challenge 2013[1] and randomly sampled 302 reviews from each. They were given the following instructions:

Reviews labeling

You will be presented with a series of documents which you will have to classify in one of either 4 categories: specific, generic, unknown or empty.

Specific documents are the ones that describe experiences. If the document is telling a story about a particular situation in the past, then it

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

is specific. Generic reviews give a general overview of a restaurant or hotel, but one wouldn't be able to tell in which particular situation the user visited the hotel or restaurant. Unknown is only assigned when you can't really tell if the document is specific or generic. Beware that if the document is mixed (contains both specific and generic parts), then the category corresponding to the biggest part in the document should be assigned. The empty category is used for cases when documents contain no information.

In general, the unknown and empty categories should be avoided. Try to use them as little as possible. If you are unsure about the category of a document, use the one that fits the best.

Examples

Specific 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 exceptional, the food amazing- it was great at the pool, Wrights and also at Frank and Alberts. The only reason I am not giving it a full 5 stars is the 'upgraded' room was just a nice basic room. Though it was certainly nice, it wasn't what I expected for being the Biltmore. However, everything else certainly lived up to that expectation.*

Generic review *Nice hotel, all the amenities you need, great complex of pools. Just make sure your room is as far from the Vista Lounge as possible; otherwise you'll be bombarded with crappy live music, fully audible from the lounge to all the surrounding rooms above it, for four hours a day. Horrible.*

When the volunteers were completed with the labeling, we assigned each review the label that the majority of the volunteers had chosen for that review. There were a few rare cases in which every volunteer assigned a different label to a review, for instance one assigned *'specific'*, another one *'generic'* and the last one *'unknown'*. Reviews that did not have a majority label were discarded. In the end, seven reviews were discarded in the restaurants dataset and two reviews were discarded in the hotels dataset. In Table 4.1 a summary of the labeled reviews is presented. The percentage of agreement between the three volunteer was 68% and we measure the inter-rater agreement using Fleiss' kappa (Fleiss 1971). Fleiss' kappa is a statistic designed to measure agreement in the case where there are more than two raters, whereas the better-known Cohen's kappa is only for two raters. The kappa coefficient for the agreement between the three volunteers was 0.37.

Table 4.1: The manually labeled datasets

Dataset	**Agreement**			**Specific**	**Generic**
	Unanimous	**Partial**	**None**		
Hotels	157(52.0%)	143(47.3%)	2(0.7%)	126(42%)	174(58%)
Restaurants	150(49.7%)	145(48.0%)	7(2.3%)	186(63%)	109(37%)

4.2 Features

In order to be able to classify reviews as either specific or generic, we extracted some features from the reviews that would help the classifier differentiate between a specific and a generic review. These were partly based on the work of (Bauman & Tuzhilin 2014). After trying out several features, the following ones helped the classifier achieve the highest classification accuracy:

- *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*

Once we extracted the features, we rescaled them using min-max scaling.

Just to give the reader a graphical representation of the usefulness of these features in discriminating between specific and generic reviews, we have decided to plot the two most relevant features in term of classification that we used in one of our datasets. In Figure 4.1 specific reviews are marked with red crosses and generic reviews are marked with blue circles. On the horizontal axis, we have *LogWords*; and on the vertical axis we have *VDBSum*. After using this data to train a logistic regression classifier, the classifier separates the reviews using the green line. Reviews below the line are classified as generic and reviews above the line are classified as specific.

We can see clearly how the longer a review is, the higher the chances that the review is specific. This makes perfect sense as people tend to use more words to describe their experiences than when they are giving a general description. Usually, people who describe experiences give more details, which makes the reviews longer. We can also see that the more there are verbs in the past tense, the higher the chances that the review is specific. Again this makes a lot of sense since people tend to use a lot of verbs in the past tense when they are describing their experiences. It is very common to find sentences like "we *ordered* the pizza, but it *was* not so good" or "we *booked* the sea-side suite and the view *was* fantastic, we *had* an amazing time". On the other hand, when general descriptions are given, then we less often find verbs in the past tense, e.g. "The bar is rather noisy".

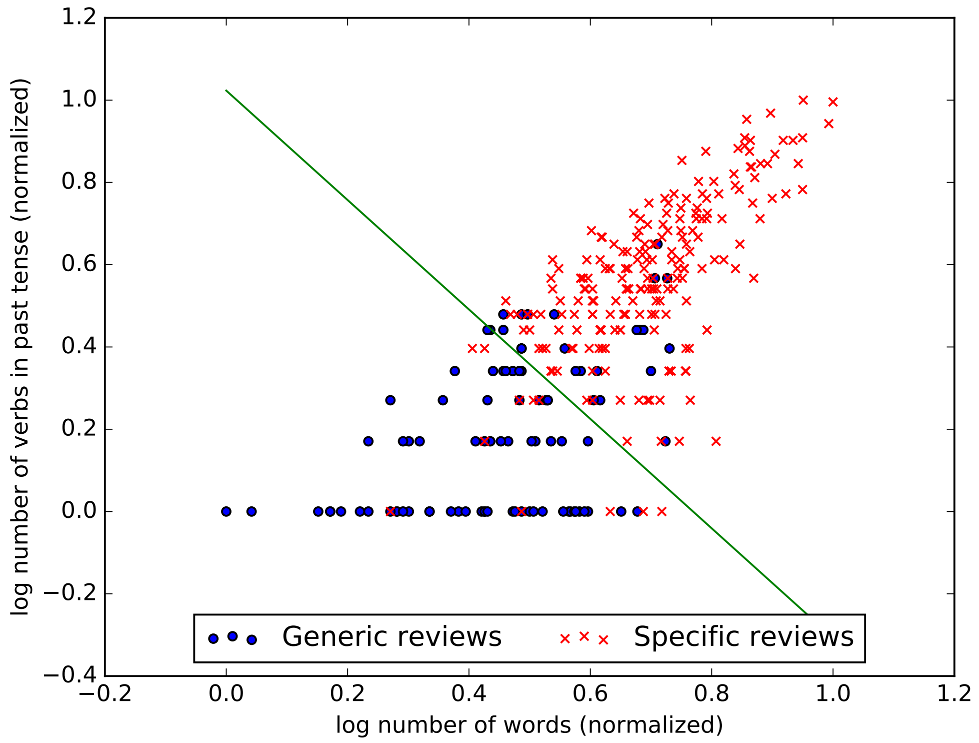


Figure 4.1: Reviews classification

4.3 Classifiers

For the classification of the reviews, we tried several types of classifiers, including a dummy baseline that always predicts the most frequent value. In Table

4.2 we provide a brief description of each of them.

Table 4.2: The classification algorithms that we compared

Classifier	**Description**
Most Frequent	This always predicts the most frequent class.
Logistic Regression	This is a linear regression model where the dependent variable is categorical.
SVC	This is a support vector machine based classifier.
k-Nearest Neighbours	This is a classifier that calculates the similarities between neighbours and takes the class of the majority of the k-nearest neighbors.
Decision Tree	This is a classifier that works by generating decision rules based on the values of the features to predict the class of the target variable.
Random Forest	This is an ensemble learning classifier that constructs multiple decision trees and returns the class that the majority of the decision trees predicts.

4.4 Dataset Balancing

In a dataset the classes might not be evenly distributed. It is possible to introduce bias into a classifier if one of the classes outnumbers the other ones by a large proportion. A typical scenario that exemplifies this kind of situation is when a classifier is trained to predict a rare disease based on some features. If the disease only happens in 0.5% of the cases, a classifier that just predicts every single time that a patient does not have the disease will have a 99.5% accuracy, but it will be useless to predict the disease. In this kind of scenario, data balancing techniques can be useful. They distribute the training data more evenly with the goal of boosting classier accuracy. They also help to reduce the class overlapping problem by removing noisy data points from the data, in this way creating more well-defined clusters (Batista et al. 2004). Data balancing is commonly used in the medical field to predict a disease and in banking to improve fraud detection (Chawla et al. 2002).

Basically, there are three ways to balance a dataset: *under-sampling* the majority class, *over-sampling* the minority class, or having a combination of the two. We will describe the approaches that we have tested in order to improve the classification of reviews.

4.4.1 Under-sampling techniques

Random under-sampler This is the most basic under-sampling technique. It randomly removes data samples from the majority class until a desired data balance is achieved. The drawback of this method is that it can discard potentially useful samples that describe the majority class really well while leaving outliers or noisy samples.

Tomek Links (Tomek 1976) This takes two samples e_i and e_j that belong to different classes, then the distance $d(e_i, e_j)$ between them is measured. If $d(e_i, e_l) > d(e_i, e_j) \forall l \neq i, j$ or $d(e_j, e_l) > d(e_i, e_j) \forall l \neq i, j$ then it is said that e_i and e_j form a Tomek link. If two samples form a Tomek link, then either one of them is noise or they are both borderline because it is unusual for two samples to be so close and belong to different classes. If Tomek links are used as an under-sampling technique, then only the samples that belong to the majority class are removed, i.e., if e_i belongs to the majority class then it is removed, otherwise e_j is removed. If used as a cleaning technique then samples from all of the classes are removed, i.e., both e_i and e_j are removed.

Edited Nearest Neighbors (ENN) (Wilson & Martinez 2000) This removes any sample that contradicts the classification of at least two of its three nearest neighbors.

Neighborhood Cleaning Rule (NCL) (Laurikkala 2001) This takes a sample e_i and retrieves its three nearest neighbors. If e_i is in the majority class and at least two of its neighbors are in the minority class, then e_i is removed. If e_i is in the minority class and at least two of its neighbors are in the majority class, then neighbors that belong to the majority class are removed.

4.4.2 Over-sampling techniques

Random over-sampler This is the most basic over-sampling technique. It creates new samples of the minority class by sampling with replacement. In other words, it creates copies of the existing minority class samples at random. This has the disadvantage that it can lead to overfitting (Batista et al. 2004).

Synthetic Minority Over-sampling Technique (SMOTE) (Chawla et al. 2002) This over-samples the minority class by creating new synthetic samples rather than by over-sampling with replacement. The samples are created by selecting random points in the the space between a minority class and its nearest neighbour. The goal of SMOTE is to increase the number of minority samples in order to make the minority class more general.

4.4.3 Combined techniques

SMOTE + Tomek Links (Batista et al. 2004) This uses SMOTE to over-sample the minority class, then, Tomek links are used to clean the over-sampled dataset, removing noise and borderline points from *both* types of classes.

SMOTE + ENN (Batista et al. 2004) This is very similar to SMOTE + Tomek links. It first applies over-sampling to the data using SMOTE and then ENN is used to perform data cleaning, removing samples from the majority and minority classes.

4.5 Summary

In this chapter we have seen how, based on a manually labeled dataset, we can build classifiers that are able to predict whether a review describes an experience or not, i.e. if it is specific or generic. We presented a methodology that will guide users to label reviews as specific or generic. Using a POS tagger, we were able to extract relevant features to help us differentiate between a specific and a generic review. In Chapter 6 we will test the classifiers that we have mentioned to identify the most accurate for our datasets. Furthermore, we will use data balancing techniques to further improve the performance of the classifier in terms of accuracy. Finally, the best classifier configuration in terms of accuracy for our datasets will be presented.

Chapter 5

Topic Modeling

With the increasing amount of digital documents, several challenges have arisen. Initially, the main concern was how to index the documents so that they could be found later by using queries, but recently more challenging operations are needed, such as automatic summarization or unsupervised retrieval of documents. Topic models have surged as an alternative approach to traditional methods of the information retrieval field.

Topic modeling is a technique that aims to find a latent semantic structure between terms based on their co-occurrence within documents without relying on any form of labeled data. The terms are grouped together into *topics* that typically represent a concept or a theme. In topic modeling, each topic is modeled as a mixture of terms and each document is modeled as a mixture of topics.

Topic modeling can be used for several purposes, such as indexing and retrieval of documents (Deerwester et al. 1990, Hofmann 1999*b*), semantic analysis (Lee & Seung 1999, Pauca et al. 2004, Hofmann 1999*a*, Blei et al. 2003), classification of documents, clustering of documents (Xu et al. 2003, Pauca et al. 2004), among others.

Generally, the data used by topic modeling algorithms has two characteristics. First, each document is represented using a bag-of-words approach in which the order of the words within the documents is lost, and second, there is no chronological order kept for the documents themselves.

In this chapter we will start with a brief description of the precursors of topic modeling, so that the reader has an idea of how topic models came to exist. In the subsequent sections we will review the two main types of topic modeling approaches: probabilistic topic models and factorization-based methods. Then, we will describe the current problems existing in topic modeling, describe the approach we have used to overcome some of the problems and present some of the evaluation techniques used to measure the quality of the topic models. Finally, we will summarize the chapter.

5.1 Precursors of Topic Modeling

One field that has researched the analysis of documents with the goal of organizing and structuring the documents is information retrieval. Information retrieval is concerned with improving the search for documents based on their content.

5.1.1 Term Frequency - Inverse Document Frequency

One of the most important techniques in the information retrieval field is Term Frequency - Inverse Document Frequency (TF-IDF) (Salton & Buckley 1988). The goal of TF-IDF is to index documents based on their content, but also taking into account the content of the other documents in the collection. In TF-IDF, each term has a weight that indicates its importance within a document. The basic idea behind this, is that the weight of a term increases proportionally to the number of times that the term appears in the document, but it decreases proportionally to the number of times that the term appears in other documents. Terms that are *unique* within a document will have a higher weight than terms that appear commonly in other documents.

The calculation of the TF-IDF score for a term is composed by two scores: the *term frequency* score and the *inverse document frequency* score. Given a term w, a document d and a collection of documents D, the term frequency score can be calculated as follows:

$$tf(w, d) = \frac{f_{w,d}}{\sum_{w' \in d} f_{w',d}}, \tag{5.1}$$

where $f_{w,d}$ is the number of times that the term w appears in the document d.

To calculate the inverse document frequency score the following is used:

$$\textit{idf}(w, D) = log\left(\frac{|D|}{\{d \in D : w \in d\}}\right), \tag{5.2}$$

then, the TF-IDF score is calculated as:

$$\textit{tf-idf}(w, d, D) = tf(w, d) \cdot idf(w, D) \tag{5.3}$$

A high TF-IDF score is reached when a term has a high frequency in a given document and appears rarely in the whole collection of documents.

Several variants of the above weighting scheme can be found in the literature where the term frequency score is changed to a binary representation that states whether the term appears in the document, for example. Other variants are also available for the inverse document frequency score, where several weighting schemes can be applied, that include various types of normalization (Pincombe 2004, Lee et al. 2005).

Using TF-IDF, it is possible to represent a document as a vector of term weights that indicate the importance of each term for that document. This vector can then be used as an index to find other similar documents. For instance, we can use cosine similarity to calculate how similar two documents are based on their TF-IDF vectors.

In the same way that we can represent a document using a TF-IDF vector, we can also represent a collection of documents using a TF-IDF matrix, in which the rows are the documents and the columns are the terms. Each cell (w, d) will then contain the TF-IDF weight of term w within document d. We will call this matrix the *document-term matrix*.

TF-IDF representation has several advantages. It uses a fixed-length representation of the documents since each document is represented as a vector with length equal to the number of terms in the documents collection. The fact that it can be used to compare the similarity between two documents in an unsupervised fashion makes it ideal for automatic text retrieval.

Nevertheless, there are several drawbacks of information retrieval using TF-IDF, such as that it fails to capture the inter- and intra-document relations between terms, it fails to retrieve documents if synonyms are used, and it can also retrieve the wrong documents given the polysemous nature of some words, i.e., words with multiple meanings.

5.1.2 Latent Semantic Analysis

In (Deerwester et al. 1990) a new method for document indexing is proposed. This method, called Latent Semantic Analysis (LSA), exploits the semantic structure present in documents in order to have a more compact representation of documents that also leads to better matching between pairs of documents and queries and documents.

Deerwester et al. uses Singular Value Decomposition (SVD) as a dimensionality reduction technique to approximate the document-term matrix by using a small number of orthogonal factors or derived dimensions. These factors may be seen as artificial concepts. Each term and document is then represented by a vector of weights indicating its strength of association with each of the latent concepts.

LSA decomposes the document-term matrix $\mathbf{A}$ into the product of three other matrices:

$$\mathbf{A} = \mathbf{H}_0 \times \mathbf{\Sigma}_0 \times \mathbf{W}_0^\top, \tag{5.4}$$

where $\mathbf{H}_0$ and $\mathbf{W}_0$ have orthonormal columns and $\mathbf{\Sigma}_0$ is a diagonal matrix.

If the singular values in $\mathbf{\Sigma}_0$ are ordered by size, the first k may be kept and the remaining ones set to zero. After multiplying the resulting matrices, an approximation matrix $\hat{\mathbf{A}}$ can be obtained which is approximately equal to $\mathbf{A}$, but of rank k. $\mathbf{\Sigma}_0$ can be simplified by removing the zero rows and columns to obtain a new diagonal matrix $\mathbf{\Sigma}$. The corresponding columns of $\mathbf{H}_0$ and $\mathbf{W}_0$ can also be deleted to obtain $\mathbf{H}$ and $\mathbf{W}$, respectively. This results in a reduced model:

$$\mathbf{A} \approx \hat{\mathbf{A}} = \mathbf{H}\mathbf{\Sigma}\mathbf{W}^\top \tag{5.5}$$

The advantages of this model include that all documents have a very compact representation, i.e. by vectors of size k. Deerwester et al. also explain that the reduction of the dimensional space helps to reduce noise and thus leads to a better generalization of the data. The model also deals well with synonyms. On the other hand, the model fails to capture the polysemous nature of the words and there is no interpretability of the matrices.

5.2 Probabilistic Topic Models

5.2.1 Probabilistic Latent Semantic Analysis

Based on the LSA work in (Deerwester et al. 1990), Hofmann proposed the Probabilistic Latent Semantic Analysis (pLSA) model (Hofmann 1999*b*,*a*, 2001). pLSA takes a probabilistic approach to model the interaction between words and documents. pLSA links documents and terms through a latent variable z that represents the k topics. Hofmann proposes a generative model where he introduces a latent unobserved variable $z_t \in z_1, \ldots, z_k$ to model the interaction between the observed variables, that is, the interaction between the documents and the terms. Here, each of the unobserved variables z_t can be interpreted as the topics. Documents are modeled as convex mixtures of topics, and topics are probability distributions over terms.

$p(w_j|z_t)$ denotes the probability that the word w_j belongs to the topic z_t and $p(z_t|d_i)$ denotes the probability that the topic z_t is present in the document d_i.

One of the key facts of this model is that Hofmann defines d_i and w_j to be conditionally independent given z_t. In that way, he is linking the observed variables (documents and terms) through an unobserved or latent variable (the topics).

Based on the above, Hofmann defines the following generative model for document/term co-occurrences:

Algorithm 2 Probabilistic Latent Semantic Analysis

```
    Input: collection of documents $D$, number of topics $k$.
    Output: topic assignments $\mathbf{z}$.
1:  for $d \in D$ do
2:      Choose a topic $z_t$ from $p(z_t|d_i)$
3:      Choose a word $w_j$ from $p(w_j|z_t)$
```

The result of this generative model is that one obtains a (d_i, w_j) while the unobserved topic variable z_t is discarded.

pLSA has a compact representation, can deal with synonyms and polysemous words and is highly interpretable, as documents can be seen as convex mixtures of topics and topics can be seen as probability distributions over terms (Hofmann 2001).

Nevertheless, there are several disadvantages for the pLSA model. There is no natural way to assign probability to a previously unseen document, which is very inconvenient for query-based search engines. No assumptions are made about the probability distribution of the documents, which leads to having a model where the number of parameters grows linearly with the number of documents and can also lead to overfitting the data (Blei 2012, Steyvers & Griffiths 2007).

5.2.2 Latent Dirichlet Allocation

LDA, proposed by (Blei 2012), extends the pLSA model by making an assumption about the probability distributions of the documents.

To simplify notation, let $\theta^{(d_i)} = p(z|d_i)$ refer to the multinomial distribution over topics for document d_i and $\phi^{(t)} = p(w|z_t)$ refer to the multinomial distribution over terms for topic z_t. The parameters θ and ϕ indicate which topics are important for a particular document and which terms are important for which topics, respectively.

LDA places a Dirichlet prior on θ, which simplifies the problem of statistical inference. Also, it is a very convenient choice for a prior since the Dirichlet distribution is a conjugate to the multinomial.

The generative model proposed in (Blei et al. 2003) is as follows:

Algorithm 3 Latent Dirichlet Allocation

```
    Input: collection of documents $D$, number of topics $k$.
    Output: topic assignments z.
 1: for $d \in D$ do
 2:     Choose $|d| \sim$ Poisson($\xi$)                ▷ $|d|$ is the length of the document
 3:     Choose $\theta \sim$ Dirichlet($\alpha$)
 4:     for $j \leftarrow 1, |d|$ do
 5:         Choose a topic $z_t \sim$ Multinomial($\theta$)
 6:         Choose a word $w_j$ from $p(w_j|z_t, \gamma)$      ▷ a multinomial probability
    conditioned on the topic $z_t$
```

Having a prior distribution on θ means that the model generalizes more easily to new documents. LDA has also the advantage that the number of parameters does not grow linearly with the size of the training corpus (Blei et al. 2003); it just depends on the number of terms and the number of topics. Additionally, it deals well with both synonyms and polysemous words.

To approximate the probability distributions, inference algorithms, such as Expected Maximization or Gibbs Sampling, are used (Blei et al. 2003, Griffiths & Steyvers 2004).

5.3 Factorization-Based Methods

The goal of factorization-based methods for topic modeling is to approximate the document-term matrix $\mathbf{A}^{m \times n}$ by decomposing it into two smaller matrices $\mathbf{W}^{m \times k}$ and $\mathbf{H}^{k \times n}$ with $k \ll min(m,n)$. An approximation to the $\mathbf{A}$ matrix is found by multiplying $\mathbf{W}$ and $\mathbf{H}$, $\mathbf{A} \approx \hat{\mathbf{A}} = \mathbf{W} \times \mathbf{H}$. Factorization-based methods assume that $\mathbf{A}$ is of low rank and thus the multiplication of $\mathbf{W}$ and $\mathbf{H}$ will give a close approximation of $\mathbf{A}$.

Based on LSA, Lee & Seung also proposed to factorize the document-term matrix, but this time using Non-negative Matrix Factorization (NMF) to do so (Lee & Seung 1999). NMF is a matrix factorization approach that reduces the dimensionality of a non-negative matrix, such as the document-term matrix. The goal of NMF is to approximate a matrix $\mathbf{A}$ as the product of two non-negative k-dimensional matrices $\mathbf{W}$ and $\mathbf{H}$ (Belford et al. 2018). These k-dimensions can be interpreted as k topics (Belford et al. 2016). $\mathbf{H}$ represents the topic-term matrix, in which each of the k topics is a vector that contains the weight of each word within the topic. $\mathbf{W}$ represents the document-topic matrix, in which each of the m documents is a vector that contains the strength of each topic within the document. In other words, each cell of the document-type vector is telling us how much of that topic is being discussed in the document. This interpretation of latent factors as topics can not be made if a matrix factorization algorithm that allows negative values is used, because it does not make much sense to say that a topic is being discussed with a negative value in a document or that a term has a negative value within a topic. Other approaches that have used NMF to produce topic models include (Pauca et al. 2004, Arora et al. 2012, Xu et al. 2003, Belford et al. 2016, 2018).

To generate the $\mathbf{H}$ and $\mathbf{W}$ matrices, optimization algorithms, such as Alternating Least Squares and Gradient Descent, are used (Pauca et al. 2004, Belford et al. 2018).

The advantages of using NMF to create topic models is that they deal well with synonyms and polysemous words, have a very compact representation and can use the wide range of available NMF methods. More importantly, the model has interpretability, as the rows of the $\mathbf{H}$ matrix can be interpreted as vectors that contain the weights of each term in a topic, and the rows of the $\mathbf{W}$ can be interpreted as vectors that contain the weights of each topic in a document.

There are a few advantages that NMF have over the LDA method. First, there is the fact that LDA can only operate on document-term matrices that have raw frequency counts (Greene et al. 2014, Belford et al. 2018), whereas NMF can work with document-term matrices that have been preprocessed using different weighting functions, such as TF-IDF, log-entropy, binary, etc., or that have been subject to document length normalization (Greene et al. 2014). Second, there are fewer parameter choices for NMF than for LDA. Finally, as reported by (O'Callaghan et al. 2015), LDA has a tendency to produce more generic and less semantically-coherent topics than NMF.

5.4 Current Problems with Topic Models

In the previous sections we have described some of the most traditional algorithms used to produce topic models and for each of them we have presented advantages and disadvantages.

One thing to note is that topic modeling is in a way very similar to clustering: they are both unsupervised strategies that attempt to find a hidden structure in the data and in many cases they are stochastic methods that use optimization or inference techniques that converge on a solution. Many problems faced in clustering, such as cluster instability, deciding on the number of clusters (for algorithms like k-means) and cluster initialization are also common in topic modeling. In this section we will focus on the problems of instability of the topic models and how to decide on the number of topics. These problems are common to all of the topic modeling algorithms we have presented.

5.4.1 Instability

Given the stochastic nature of topic modeling, it is common that topic models produced using the same data, the same algorithm but with a different random seed are different. This is called the *instability* problem. Instability happens because the algorithms converge to different local minimums in the optimization or inference process (Belford et al. 2018). This is revealed in two ways: the topics are composed of different terms or the order of the terms has changed, and the documents have different assignments to topics. This problem has also been studied in clustering, where some clustering algorithms are initialized with random values and then different clusters can be produced when running the algorithm on the same data with different random seeds.

Instability is a big problem for many reasons. The topic models generated can become unpredictable; the experiments can not be reproduced; if used for document retrieval, different documents will be retrieved for the same query if the random seed is changed; and so on. It is common in many applications to assume that the generated topic models are definitive, even when different runs of the same algorithm can result in different topic models (Belford et al. 2018).

In our particular case, since we are going to use topic models to extract contextual information out of the reviews, we need topic models that we can rely on so that we can produce good recommendations. The quality of our recommender can not rely on a “lucky” random seed.

In Section 5.5 we will describe the approach we have used to overcome the instability problem and in Section 5.6 we describe a set of metrics to measure the stability of a topic model.

5.4.2 Deciding on the number of topics

Another common problem that researchers face when building topic models is deciding on the number of topics. When the number of topics is too low, the generated topics become too general. If the number of topics is too large, repeated topics begin to appear; this is known as the “over-clustering problem”. The accuracy of a topic model is sensitive to the number of topics. It then becomes necessary to define metrics that help us decide what is the best number of topics.

Greene et al. proposes to use stability metrics to decide on the number of topics. The idea behind this proposal is that a topic model with high stability is more robust to perturbations in the data (2014). Greene et al. uses stability metrics to predict the number of topics. They showed their predictions correlated well for several corpora where the number of categories was known beforehand. Based on those results we will also use the stability metrics to help us choose the ideal number of topics.

Arun et al. also propose a metric to find the ideal number of topics (2010). Arun et al. note that when the topics are orthogonal to each other, i.e., well-separated, the singular values that come from the Singular Value Decomposition of the topic-term matrix are equal to the row L_2 norms of the topic-term matrix, which can be interpreted as the proportion of topics assigned to the corpus. They also note that as the number of topics increases, the topics start to become more and more orthogonal. If a value for the number of topics k is reached where the topics are well-separated, they will continue to be well-separated for every $k' > k$. The goal in (Arun et al. 2010) then becomes to find the smallest value of k where the topics are well-separated (or orthogonal).

5.5 Topic Ensembling

Given that instability is a big problem, we decided to use the topic ensembling approach presented in (Belford et al. 2016, 2018). Our decision to integrate this into Rich-Context has been one of the most helpful elements towards out-performing other recommender systems. Before using the topic ensembling approach our results were very inconsistent.

Belford et al. borrow techniques from the clustering world and apply them to topic modeling in order to obtain stable topic models (2018). Their idea is very simple: generate a lot of topic models and then aggregate them together into a final topic model in order to reduce variability. Their approach for generating stable topic models is divided in two steps: *ensemble generation* and *ensemble integration*, as illustrated in Figure 5.1.

In the ensemble generation step, r topic models are generated from the same corpus using different random seeds each time but preserving the value for the number of topics k. The topic models are generated using NMF, as described in Section 5.3. These topic models are called the *base topic models*. After each

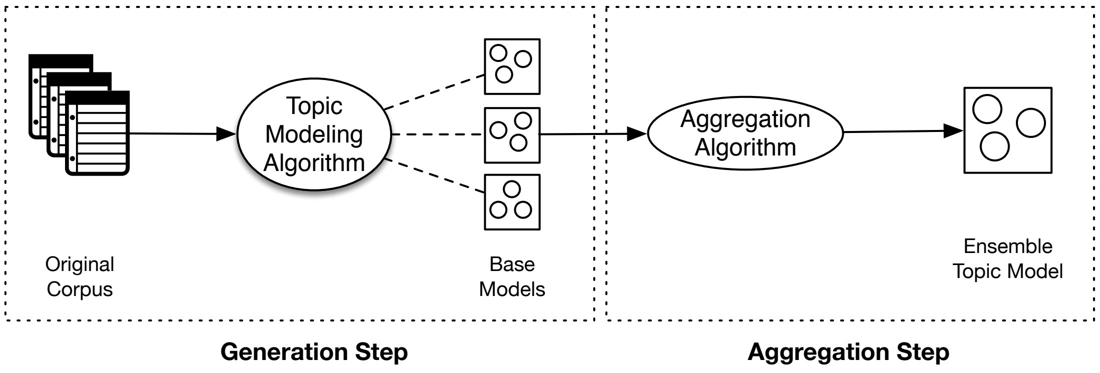


Figure 5.1: Illustration of the topic ensemble approach.

run, the topic-term matrix **H** of the base topic model is kept for later use.

In the ensemble integration step, a new representation of the corpus is created in the form of a topic-term matrix **M**. This matrix is created by stacking together the transpose of each topic-term matrix **H** created in the ensemble generation step, forming a matrix of dimensions $r \cdot |D| \times |W|$, where r is the number of base topic models, $|D|$ is the number of documents and $|W|$ is the number of terms. Belford et al. note that the order in which the matrices are stacked is irrelevant to the output of the algorithm (Belford et al. 2018).

After the matrix **M** has been created, then NMF is used to decompose **M** into two matrices using k factors. The newly obtained **H**′ is going to be the final topic-term matrix that contains the weight of each term inside a topic.

Belford et al. propose a variant of the ensembling topic model described above that is based on the popular k-fold cross-validation strategy used in machine learning (Belford et al. 2018). In this method, instead of using the whole corpus for creating each of the base models, the corpus is divided into f folds of equal size. In each turn, one of the f folds is removed, leaving $(f - 1)$ folds which are used to generate the topic model using NMF. At the end of the process, f topic models are obtained. To reduce variability, the process is repeated p times, which means that at the end of the whole cycle, $f \cdot p$ base topic models will have been generated. Finally all the base topic-term matrices are stacked together in a matrix **M** and NMF is applied to this stacked matrix to generate the final topic-term matrix **H**′ in the same way as described before.

The results reported in (Belford et al. 2018) show that the k-fold version of the topic ensembling approach generates the most stable topic models. In our work, we use the k-fold version of the topic ensembling algorithm.

5.6 Topic Model Evaluation

Determining the quality of a topic model remains an open problem in research and there is no definite way to say that one topic model is better than another. This is similar to clustering, in which is difficult to say if a clustering is better than another. Having said that, we do use metrics to measure the quality of our topic models and our ultimate goal is to predict the performance of the recommender based on the quality of our topic model. In this section we will present four metrics, two for topic model stability and two, designed by us, to measure the quality of the topic models based on their content of contextual words.

We have created a two-step methodology to help us determine which is the best topic model for our recommendation goal. The first step in this methodology is to decide on the topic modeling algorithm. This decision is made based on the stability metrics described in Section 5.6.1. Our goal is to select the algorithm that returns the most stable results so that we can be confident about our results and we are able to reproduce them in the future. We do not want our results to depend on a "lucky" random seed.

Once we have selected the topic modeling algorithm, we are going to decide which types of words to use based on the part-of-speech tags. In this case we will use the context-richness metrics described in Section 5.6.2. We want to use the types of words that produce the most context-rich topic models to facilitate the context extraction process from the topic models.

The number of topics is a highly relevant hyperparameter when building a topic model and, in our case, will affect the performance of the overall recommender system. However, despite several attempts, we could not find a metric for choosing the number of topics that would correlate well with the recommendation performance, either for ranking or rating prediction. We tried, for example, using the divergence metric proposed by (Arun et al. 2010) without success. In the end, the number of topics remains as a hyperparameter of Rich-Context and we set it by grid-search. As we will show in Chapter 6, it has a great impact on the recommendation performance.

5.6.1 Topic model stability

The topic stability metrics will help us to choose the best topic modeling algorithm. Our choice is based on stability metrics as we want to obtain results that we can reproduce in the future. Once we have found a stable topic model algorithm we will explore the hyperparameter space to improve the quality of the generated topic models. The metrics presented in this section were proposed in (Belford et al. 2018). One advantage of these metrics is that they are unsupervised.

5.6.1.1 Descriptor set difference

Belford et al. argue that if two topic models are very similar, then there should be little or no difference between the top p terms of their topics (Belford et al. 2018). Based on this, for a topic model $\mathcal{M}_i$ a set of terms W_i is created. W_i contains the union of the top p terms of all of the topics in $\mathcal{M}_i$. Then for two models, the amount of different terms that they have in common is compared by using the following formula:

$$DSD = (\mathcal{M}_i, \mathcal{M}_j) = \frac{W_i \triangle W_j}{p \cdot k} \tag{5.6}$$

where $\triangle$ is the symmetric set difference, i.e., $W_i \triangle W_j = (W_i - W_j) \cup (W_j - W_i)$.

This equation returns a value in $[0, 1]$ where 0 means that there are no term differences between the two topic models and 1 means that the two topic models share no terms.

Having this, we can produce the Average Descriptor Set Difference (ADSD) score for r topic models $\{\mathcal{M}_1, \ldots, \mathcal{M}_r\}$ by using the following equation:

$$ADSD = \frac{1}{r \cdot (r-1)} \sum_{i,j i \neq j}^{r} DSD(\mathcal{M}_i, \mathcal{M}_j) \tag{5.7}$$

5.6.1.2 Topic-term stability

To capture the variance at an individual topic level, Belford et al. propose the topic-term stability metric that makes pairwise topic comparisons to detect how much topics change individually (Belford et al. 2018).

To measure how similar two topics are, Belford et al. propose to use a similarity metric based on the Jaccard index. The topics are represented by their top p terms.

$$Jaccard(R_i, R_j) = \frac{R_i \cap R_j}{R_i \cup R_j}, \tag{5.8}$$

where R_i contains the top p terms of the i-th topic. Based on the Jaccard metric, a pairwise similarity matrix can be built to record the similarity value of every i, j pair of topics. The next step is to find the matching topics for two topic models $\mathcal{M}_i$ and $\mathcal{M}_j$. This is done by finding a permutation π in $\mathcal{M}_j$ for the topics in $\mathcal{M}_i$ (Belford et al. 2018). Having this, we can calculate the Term Stability (TS) score:

$$TS(\mathcal{M}_i, \mathcal{M}_j) = \frac{1}{k} \sum_{x=1}^{k} Jaccard(R_{ix}, \pi(R_{ix})) \tag{5.9}$$

where $\pi(R_{ix})$ represents the topic in topic model $\mathcal{M}_j$ that corresponds to the i-th topic in topic model $\mathcal{M}_i$ by the permutation π. The TS metric has a range in $[0, 1]$. A score of 1 means that we have two k-way identical topic models, where k is the number of topics.

Having this, we can produce the Average Term Stability (ATS) score for r topic models $\{\mathcal{M}_1, \ldots, \mathcal{M}_r\}$ by using the following equation:

$$ATS = \frac{1}{r \cdot (r-1)} \sum_{i,j i \neq j}^{r} TS(\mathcal{M}_i, \mathcal{M}_j), \tag{5.10}$$

where a score of 1 indicates that all pairs of topic descriptors were matched identically across the r topic models (Belford et al. 2018).

5.6.2 Context richness

Even though we have some metrics to measure the quality of topics models in general, in our particular problem, we are worried about context-driven recommendations, and for that we need topic models that are rich in contextual words. If we have topic models with high quality but a poor amount or no contextual words at all, that topic model is not going to be useful for our context-driven recommendations. Therefore, we also need metrics that reveal how rich is a topic model in terms of contextual words. In this section we will describe the metrics we have designed to measure the amount of context richness in our topic model.

The proposed metrics require that we manually define a vocabulary V of contextual words. We stress that this vocabulary is not necessary for the recommendation task, nowhere is it used in the Rich-Context system. It is only used in evaluation: to confirm that our method learns context-rich topic models.

5.6.2.1 Contextual topic score

Contextual Topic Score (CTS) helps us determine the context-richness of a topic t and is the sum of the weights of the contextual words that are part of the contextual vocabulary V. This metric assumes that the sum of all the entries in the topic t add up to one.

$$CTS(t) = \sum_{w \in V} \varphi_{w,t} \quad \forall w \in V \tag{5.11}$$

where $\varphi_{w,t}$ denotes the weight that the word φ has in topic t and V is the vocabulary of all contextual words.

The score of this metric is in $[0, 1]$. 1 means that every word in the topic was contextual and 0 means that none of the words in the topic were contextual. For our purposes the higher the value of the contextual topic score, the better.

5.6.2.2 Contextual topic model score

Contextual Topic Model Score (CTMS) averages the contextual topic scores of all the topics in a topic model $\mathcal{M}$.

$$CTMS(\mathcal{M}) = \frac{\sum_{t \in \mathcal{M}} CTS(t)}{k}, \tag{5.12}$$

where k is the number of topics of topic model $\mathcal{M}$.

5.6.3 Human evaluation of topic models

Besides the methods that we have shown to evaluate the quality of topic models, there is another family of methods that relies on humans to evaluate the quality of topic models. We do not incorporate human evaluation of topic models into our work at this stage because our work currently focuses on recommendation accuracy and not other factors. But, we feel it is important to acknowledge this family of approaches to evaluation, because, after all, reviews are written by humans and this type of evaluation is particularly useful if we were to extend our work to incorporate explanations.

The first work to introduce this type of evaluation is (Chang et al. 2009), where two metrics to evaluate the coherence of topic models are presented: word intrusion and topic intrusion. The *word intrusion metric* is used to measure the coherence of words within a topic, and it presents users with a set of six words in random order. This set of words consist of the five words with highest probability in a topic plus a word called the *intruder word* that is selected randomly from the words with lowest probability in that topic. To ensure that the intruder word is not ignored, e.g. in the case where it is a rare word, the intruder must be a word with high probability in another topic. The user must then select the intruder word. The coherence of a topic is measured by the percentage of times that users correctly select the intruder word.

The *topic intrusion metric* measures how well a topic model assigns topics to documents. It presents users with a set of four topics in random order along with a document. This set of topics consist of the three topics with highest probability in the document plus a topic called the *intruder topic* that is selected randomly from the topics with lowest probability of the document. The user must then select intruder topic. The quality of the topic model is measured by the percentage of times that users correctly select the intruder topic.

Other authors to use the word intrusion and topic intrusion methodology include (Lau et al. 2014) and (Arnold et al. 2016).

(Newman et al. 2010) introduces a more simplistic approach in which only the quality of the topics is rated using a scale from 1 to 3. The approach is also used in (Mimno et al. 2011) and (Aletras & Stevenson 2013). We note that the purpose of (Newman et al. 2010) is to find a metric that correlates with the human evaluation of topic models in order to evaluate topic models automatically.

Finally, (Lee et al. 2017) introduces another approach in which users evaluate three aspects of topic models: the clarity of the topic words, the consistency of the document set, and the topic-document correlation. These three aspects are rated by the user using a scale from 1 to 10.

In our work the evaluation of the quality of the topic models happens indirectly, through the accuracy of the recommender. In other words, the best topic models are the ones that lead us to better recommendations. There are two reasons why we did not use human-based evaluations for our topic models. First, we are more interested in producing accurate recommendations than producing interpretable topic models. Second, the costs of having humans evaluate our topic models is too high at the stage of hyperparameter selection. As we will see in Chapter 6, we had to try with dozens of numbers of topics for each dataset to improve the accuracy of Rich-Context, something that is unfeasible when human evaluation is involved.

The presented works are highly relevant if in the future we want to extend Rich-Context to incorporate item summaries, user profiles or explanations — all are tasks in which interpretability and coherence are crucial.

5.7 Summary

In this chapter, we have explored the two main approaches for topic modeling algorithms: probabilistic topic models and factorization-based approaches. We have described in detail how they work and their drawbacks and benefits. Then, we described a new topic modeling approach that overcomes one of the biggest problems in topic modeling, the instability problem. We also introduced the reader to several metrics to evaluate the quality of topic models. In Chapter 6 we run experiments across two datasets to determine the best way to build the topic models based on the presented metrics. We will show that, despite its popularity, LDA is not suitable because of its high instability and we have opted for the topic ensembling approach.

In this chapter we have given an explanation of what topic modeling is, how it works and how to choose the ideal algorithm and its hyperparameters. Based on the topic models derived from the work of this chapter, we will extract the contextual information that will enable us to make context-driven recommendations as we will see in the next chapter.

Chapter 6

Evaluation

In this chapter we present the results of evaluating our model. We start by describing the datasets we used for the testing, followed by the preprocessing steps we executed to ensure the quality of our data. The subsequent sections of the chapter present the results of evaluating the three main components of Rich-Context: RCClassifier, RCMiner and RCRecommender. For each of the three components, we describe the methodology and the metrics we have used for the evaluation and then present the results. In the last part we describe the implementation details such as the languages, the packages and libraries that we have used.

The goal of this chapter is to show the reader the performance of the three main components of Rich-Context and also guide the reader through the decisions we have made regarding the selection of algorithms and establishing a methodology based on our findings. Finally, we will show how, after following these decisions, we were able to beat the state of the art in both rating and ranking prediction.

6.1 Datasets

As seen in Chapter 2, one of our major criticisms of the evaluation methodologies of much of the work that we reviewed was the use of non-real-world datasets. These datasets were sometimes synthetically generated by the authors of the papers (Karatzoglou et al. 2010, Baltrunas et al. 2011) or sometimes came from surveys (Karatzoglou et al. 2010, Baltrunas et al. 2011, Zheng et al. 2012*a*, 2013, Zheng, Mobasher & Burke 2014) or user studies (Unger et al. 2016). Evaluating a model using a synthetic dataset gives no guarantee that the model would work in the real-world. Datasets that come from surveys or user studies have the risk of biasing the users and tend to be quite small and not sparse. Some other authors introduced datasets that were not publicly available (Karatzoglou et al. 2010, Baltrunas et al. 2011, Unger et al. 2016, Zheng et al. 2012*a*, 2013, Zheng, Mobasher & Burke 2014, Diao et al. 2014, Levi et al. 2012, Dong et al. 2016), limiting the opportunity for making comparisons. Other approaches were only evaluated using one dataset (Unger et al. 2016, Diao et al. 2014), which raised concerns about the ability of the model to generalize to other datasets.

Based on the concerns explained above, to evaluate Rich-Context we looked for datasets with the following four characteristics. First, the dataset should contain ratings and reviews. This is required since we want to extract the contextual information out of the reviews and the ratings will help us evaluate the prediction performance of our model. Second, the dataset should be sparse. This is the natural situation in CARS, given that an item will have been rated under only a small number of times under a given context. We want to evaluate our model under those sparse conditions. Third, the dataset should be a real-world dataset. We want to know how our model performs under real-world conditions, where reviews are unstructured and explicit contextual information is unavailable. Generally this condition goes in hand with the sparsity condition. Fourth, the dataset should be publicly available. We want a dataset that is accessible to other authors so that they can compare the performance of their models against Rich-Context. A final point is that we want to evaluate Rich-Context using multiple datasets, to see how well the model generalizes to various datasets.

Taking into account the four aforementioned characteristics, we conducted our experiments using three datasets, Yelp Hotels and Yelp Restaurants that were part of the ACM Recommender Systems Challenge 2013[1], and TripAdvisor Hotels that is part of the Four-city datasets[2]. All of the datasets contained the user ID, the item ID, the user's rating for the items and the user's review for the rated item. We can see a summary of the datasets in Table 6.1.

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

[2]http://www.cs.cmu.edu/~jiweil/html/four_city.html

Table 6.1: Description of the datasets.

Dataset	Reviews	Users	Items	Sparsity
Yelp Hotels	5 034	4 148	284	0.9957
Yelp Restaurants	158 430	36 473	4 503	0.9990
TripAdvisor Hotels	878 561	576 689	3 945	0.9996

6.2 Data Preprocessing

Every time a dataset is going to be analyzed, regardless of what the purpose of the analysis is, the data has to go through a preprocessing step. This preprocessing step is especially important when working with real-world data, in which it is very usual to see dirty or malformed records, which, if they were used in the data analytics process, would bring us to misleading results. A well-known phrase in the data analytics field is “garbage in, garbage out”, which reveals the importance of cleaning and correctly transforming all the data before starting to work with it (Adomavicius et al. 2011).

6.2.1 Cleaning

In our work, we are dealing with real-world datasets in which it is frequent to find missing data, empty fields or fields with errors that can lead to a misrepresentation of the data (for instance, when a value for a rating is 50 instead of 5.0, or when we have hundreds of empty reviews made by a test user that developers forgot to delete from the production database). For this reason, it was necessary for us to perform a simple but yet effective cleaning process.

6.2.1.1 Missing and erroneous IDs

In order to detect which reviews were not good, we first counted the number of reviews per user and the number of reviews per item. The purpose of this is to discover outliers and detect if they have a high count of reviews because of some mistake. This led us to discover that in some datasets reviews whose user ID was an empty field had a high frequency as well as other reviews made by some *‘default’* user. For the Yelp Hotels and Yelp Restaurants datasets, we found no records with missing or erroneous IDs, but the TripAdvisor Hotels contained $77\,365$ records whose user ID was empty or equal to `‘CATID_’`, a user ID with almost 300 reviews that we assumed was a default for when user information was unavailable. Analyzing the number of reviews by the rest of the users, `'CATID_'` was clearly an outlier. In the end, we removed all records with missing or erroneous IDs.

6.2.1.2 Removing duplicates

One of the assumptions that we make in this work is that there is only one review per user-item pair, although this might not be the case in real life, where one user can make several reviews to the same place based on different visits. We decided to simplify our model so as not to include repeated reviews. In the majority of real-world datasets that we have, there is only one review per item. In the datasets where there is more than one, we decided to take the first review in chronological order and discard the rest. For the Yelp Hotels and Yelp Restaurants datasets there was only one review per user-item pair. However the TripAvisor Hotels dataset did contain more than one review per user-item pair. In total, we discarded 7 305 records from the TripAdvisor Hotels dataset.

6.2.1.3 Low frequency users and items

Another common practice in data preprocessing is to remove users and items that have a low number of reviews. Examples of this can be seen in (Chen & Chen 2015, Diao et al. 2014, Jakob et al. 2009, Wang et al. 2012, Wu & Ester 2015). This is done because it has been shown that models that are trained with data that comes from users and items with a certain minimum number of interactions work better than the ones that include users and items with a very low number of interactions. In our work we only remove items that have fewer than 10 reviews. Infrequent users are left because we want to see how well our model performs for cold-start users. For the Yelp Hotels dataset, we discarded 186 items that corresponded to 911 records. For the Yelp Restaurants dataset, we discarded 1 953 items that corresponded to 9 697 records. For the TripAvisor Hotels we discarded 606 items that corresponded to 2 555 records.

6.2.1.4 Removing foreign reviews

Another problem with some datasets, especially the ones that have been crawled from the web, is that it is frequent to find reviews that have been written in a language different from English. This of course poses a problem because it is going to generate noise when using the foreign language reviews to train the model. It is necessary then to first detect the language of each review in the dataset and then keep only the reviews that have been written in English. To identify the language of the reviews, we have used the `Python` package `langdetect` 1.0.7. We found $16, 574$ and 64910 records that were not in English language in the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels dataset, respectively, and we removed them.

6.2.1.5 Removing records from the classification training set

Finally, for the Yelp Hotels and Yelp Restaurants datasets, the records that were used to train RCClassifier were removed. This was done in order not to bias the recommender. For the Yelp Hotels dataset, 298 records were removed that correspond to the 300 used to train the classifier minus 2 records that were previously removed because they belonged to items with fewer than 10 reviews. Similarly, for the Yelp Restaurants dataset, 295 records were removed that correspond to the 300 used to train the classifier minus 5 records that were previously removed because they belonged to items with fewer than 10 reviews. To train the classifier for the TripAdvisor Hotels dataset we used the same set of records from the Yelp Hotels dataset because they were also reviews of hotels, so no additional records were removed from the TripAdvisor Hotels dataset.

6.2.1.6 Summary

Table 6.2 presents a summary of the datasets after the cleaning process.

Table 6.2: Summary of the datasets after the cleaning process.

Dataset	**Reviews**	**Users**	**Items**	**Sparsity**	**Discarded records**
Yelp Hotels	3 809	3 205	98	0.9879	1 225 (24.3%)
Yelp Restaurants	147 864	35 021	2 550	0.9983	10 566 (6.7%)
TripAdvisor Hotels	726 426	526 717	3 299	0.9996	152 135 (17.3%)

6.2.2 Transforming

Once we have the records that we are going to use, we have to transform the data in order to make it easier to process and to give it to other components in the model. Particularly, in this work, we performed POS tagging and lemmatization, and we built bag of words representations, which we will describe in the following subsections.

6.2.2.1 Part-of-speech tagging

Part-of-speech (POS) tagging consists of labeling each word of a sentence with its corresponding POS tag. In other words, we want to identify the nouns, pronouns, verbs, adverbs, adjectives, etc. of the sentence. Due to the ambiguity of English language (or any other natural language), this task is more complicated than it appears. Depending on the syntactic context of a word, its meaning can change, but also its POS tag. For instance, let's take a look at the word *light* in these three sentences: *"There was no light in the room"*, *"Her backpack was very light"* and *"The damp wood was useless to light the fire"*. In the first sentence, *light* is a noun, in the second one it is an adjective and in the third one it is a verb. A good POS tagger will use the syntactic context in order to assign the correct tag.

To label the words in a sentence, we tried several packages including the `scikit-learn` and `Pattern` packages. After inspecting the POS tags assigned to the reviews by each package, we decided to use `Pattern` as it had fewer mistakes in the assigned tags.

6.2.2.2 Stemming and lemmatization

It is frequent when processing large text corpora to encounter words that can be grouped together because they are syntactic variants of the same base word. For instance, if we are interested in seeing which nouns appear more frequently in a corpus, it would be a good idea to group the words *friend* and *friends* together as they are variants of each other. In order to achieve that, there are two popular techniques within natural language processing that we can use: stemming and lemmatization.

Stemming is the task of reducing terms to a stem by taking off the affixes. For instance, “automate”, “automates”, “automatic” and “automation” all reduce to “automat”. There are a few inconveniences with stemming, however. First of all, by taking off the affixes words that have different meanings but that share the same root may be reduced to the same stem. For instance, *organization* and *organs* are both reduced to *organ*, which is not the desired behaviour. Another minor inconvenience is that sometimes terms can be reduced to non-existing words, for instance the word *lazy* is reduced to *laz* and the word *arguing* is reduced to *argu*. To overcome this, we can use another natural language processing technique called lemmatization.

Different to stemming, lemmatization attempts to reduce a word to its base form, this being an actual dictionary word. Lemmatization relies on POS tagging and morphological analysis. For instance, the word *feet* has *foot* as its lemma, whereas stemming misses that link.

In this work we use lemmatization based on the `Pattern`[3] `Python` module.

6.2.2.3 Building bags of words

Once we tagged and lemmatized every word in our text corpus, we represented each review as a bag of words. Note that in a bag of words representation, the order of the words in the document is lost.

For our approach, we only included nouns as contextual information is described mostly using nouns. Adjectives, verbs and other types of words are rarely used to describe a contextual situation. In Section 6.4.3.2 we see how using only nouns translates into context-richer topic models.

In the next sections, we will describe how we evaluated Rich-Context and each of its components: RCClassifier, RCMiner and RCRecommender. We will present the evaluation methodologies along with the results.

6.3 Evaluation of RCClassifier

As described in Chapter 4, the goal of RCClassifier is to discriminate between specific reviews and generic ones. We do this because specific reviews contain more contextual information than generic ones and this will help RCMiner to

[3] `https://www.clips.uantwerpen.be/pattern`

extract the contextual information by building topic models using only specific reviews.

In Chapter 4, we saw that there are many factors that can have an influence on the classification performance, mainly being the features that we give to the classifier, the selection of the right classification algorithm, the data balancing technique that we select, and the value of the hyperparameters that are given to the classification algorithm. In this section we will describe the experiments we conducted that lead us to select a final combination of the mentioned factors based on the classification accuracy.

6.3.1 Evaluation methodology

For our experiments, we used the nested cross-validation strategy. For the *outer* fold, we split the data into five folds and left one fold as the test fold; the rest was used for training. For the *inner* fold, we took the training data of the *outer* fold and split it into another five folds, leaving four for training and one for validation. Then we performed hyperparameter tuning using grid search to find the best set of hyperparameter values that maximize the score (in this case the accuracy) in the validation set. The data balancing techniques were also included as part of the hyperparameters. In Table 6.3, we can see the list of candidate hyperparameter values that we used for each type of classifier. For each fold in the *outer* fold, having found the set of hyperparameter values that performed the best, these are the ones we used to train the outer fold model and test on the outer test fold the accuracy of the algorithm. In the end, we did nested cross-validation for every type of classifier and kept the one with the highest estimated accuracy.

Finally, after selecting the best classification algorithm, we need the best hyperparameter values for the winning algorithm. Here we also use grid search. Again we split the dataset into five parts, using four for training and one for testing. For each combination of hyperparameter values, we train a model using the four parts of the dataset and test it on the remaining part. We repeat this five times and select the hyperparameter value configuration with the highest average accuracy across the five folds. In Section 6.3.3 we report the values for these hyperparameters.

It is very important to note that even though we have three datasets, only two of them were manually labeled: Yelp Hotels and Yelp Restaurants. This is due

Table 6.3: The list of candidate hyperparameters used to train the classifiers

Classifier	**Parameter**	**Values**
Most Frequent	–	–
Logistic Regression	Inverse of regularization	$0.1, 1.0, 10, 100, 1000$
SVC	Kernel	RBF, linear
	Inverse of regularization	$0.1, 1.0, 10, 100, 1000$
k-Nearest Neighbors	Number of neigbours	1, 2, 5, 10, 20
	Weights	uniform, distance
Decision Tree	Maximum depth of the tree	None, $2, 3, 5, 10$
	Min samples to split a node	$2, 5, 10$
Random Forest	Number of estimators	$10, 50, 100, 200$

to the fact that the Yelp Hotels and TripAdvisor Hotels datasets are from the same domain. This means, that when we train the classifier to classify reviews for the TripAdvisor Hotels dataset, it is trained using the Yelp Hotels manually labeled reviews.

6.3.2 Metrics

The metric we decided to use to measure the performance of the classifier is *accuracy*, which is defined as the proportion of correct predictions over the test set. In other words:

$$accuracy = \frac{t_p + t_n}{|E_{test}|}, \tag{6.1}$$

where t_p stands for true positives, t_n stands for true negatives and $|E_{test}|$ is the number of samples, i.e. the number of reviews in the test set.

6.3.3 Results

In Figure 6.1, we can see a summary of the accuracies of the classification algorithms we have used. The best classifier for the hotels dataset was Logistic Regression with an inverse regularization value of 10 and no dataset balancing. This classifier had an accuracy of 75%. For the restaurants dataset, it was the k-Nearest Neighbours classifier with 2 neighbours and uniform weights and using SMOTE-ENN to balance the dataset. This classifier had an accuracy of 80%.

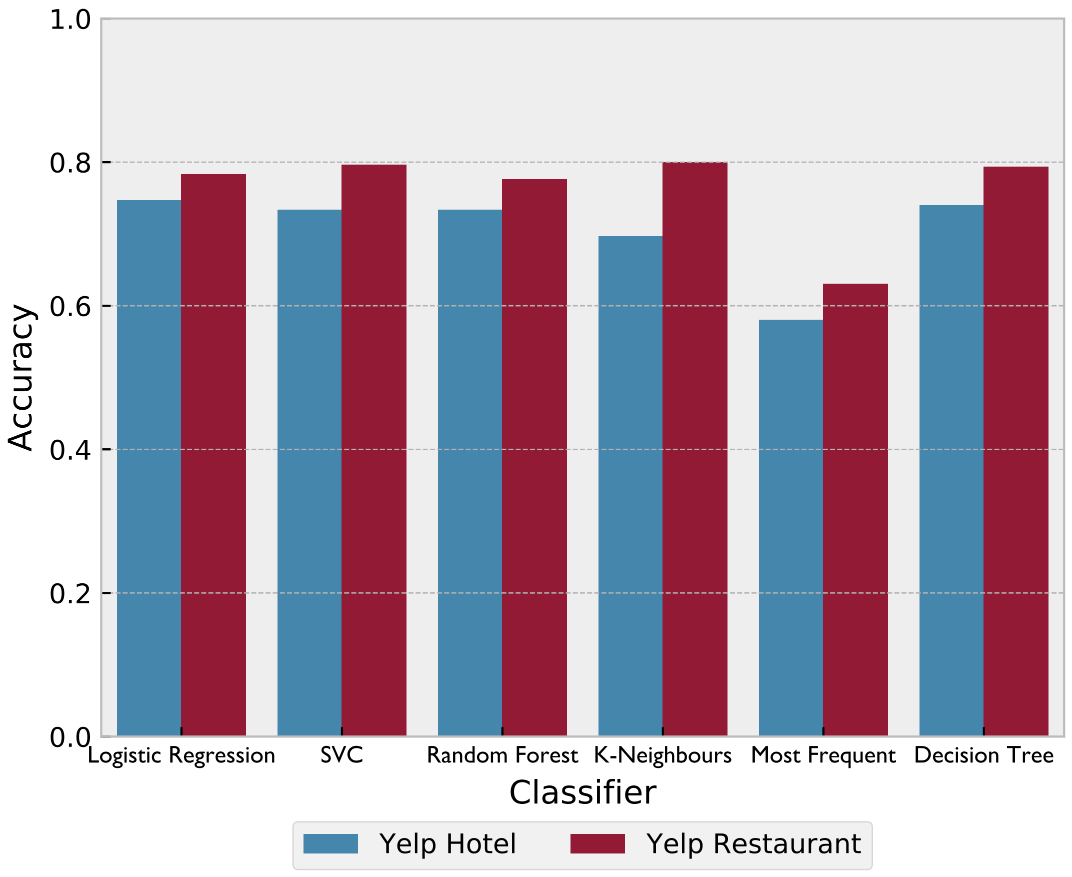


Figure 6.1: Accuracy of reviews classification

6.4 Evaluation of RCMiner

As we saw in Chapter 3, the goal of RCMiner is to extract contextual information out of the user-generated reviews. It does so by building topic models using only specific reviews and discarding topics that appear more frequently in specific reviews than in generic ones. The remaining topics are then labeled as contextual topics.

There are many ways in which a topic model can be built. Many factors such as the selection of the algorithm, the input data of the algorithm and the hyperparameter values can influence the quality of a topic model. In this section we will describe the results obtained in our quest to find the best topic models.

6.4.1 Evaluation methodology

The goal of evaluating RCMiner is to determine in which way we can build the best possible topic models. This is done in two steps. First, based on stability metrics, we determine which algorithm to use. We will choose the algorithm with the highest stability. Second, once we have chosen our algorithm, the next step is to determine what kind of data to be used by the topic modeling algorithm. The goal of this step is to create context-rich topic models. We will vary the types of words we use to create the topic models along with the types of reviews. The details of the metrics are presented in the next section.

To evaluate the stability of a topic modeling algorithm, in our experiments, we used the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels dataset to compare LDA, NMF and the topic ensembling approach. We created 100 topic models with the entire dataset, each with a different random seed and applied the ADSD and ATS metrics. We chose the topic modeling algorithm with the highest stability.

To evaluate the context richness of a topic model we want to determine what type of data to be used by the topic modeling algorithm. In this case, we want to explore which POS types are worth including in the construction of our topic model. In particular we will compare using nouns-only (NN), verbs-only (VB), adjectives-only (JJ) and all types of words to see how they impact the context-richness of the topic models. We want to have very context-rich topic models in order to extract the contextual information from them. As we saw in Section 5.6.2, context-richness measures the weights of contextual words within a topic model. To evaluate which types of words we should use, we relied on the contextual topic model score metric introduced in Section 5.6.2.2. We also explored how context-richness varies when using only specific reviews to build the topic model versus using only generic reviews. We evaluated the context-richness using the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels datasets. We built the topic models using the entire datasets.

6.4.2 Metrics

To evaluate the generated topics model we will use the metrics we described in Section 5.6. Specifically we will use three metrics, which are Average Descriptor Set Difference (ADSD), Average Term Stability (ATS) and Contextual Topic

Model Score (CTMS).

ADSD measures the term differences between two topic models and uses the set union of the top p terms of each topic in a topic model. We use the metric given by Equation 5.7. If $ADSD(\mathcal{M}_i, \mathcal{M}_j) = 0$, it means that there is no difference between $\mathcal{M}_i$ and $\mathcal{M}_j$. If $ADSD(\mathcal{M}_i, \mathcal{M}_j) = 1$, it means that $\mathcal{M}_i$ and $\mathcal{M}_j$ are completely different. For ADSD the lower (closer to 0), the better.

Differently to ADSD, ATS captures the variance at an individual topic level. ATS makes pairwise topic comparisons to detect how much topics differ. We use the metric given by Equation 5.10. If $ATS(\mathcal{M}_i, \mathcal{M}_j) = 0$, it means the topics from $\mathcal{M}_i$ are completely different from the topics from $\mathcal{M}_j$. If $ATS(\mathcal{M}_i, \mathcal{M}_j) = 1$, it means that all of the topics are the same. For ATS, the higher (closer to 1), the better, as it means that the topic models produced by the topic modeling algorithm are more contextual.

CTMS measures the ratio (given by the term weights) of contextual terms within a topic model. We use the metric given by Equation 5.12. If $CTMS(\mathcal{M}) = 1$, it means that the sum of the contextual term weights for all the topics in $\mathcal{M}$ is 1; in other words, it means that all of the terms that have a weight greater to 0 are contextual. If $CTMS(\mathcal{M}) = 0$, it means that none of the contextual terms have weights greater than 0. For CTMS the higher (closer to 1), the better, as it means that the topic models produced by the topic modeling algorithm have more contextual information.

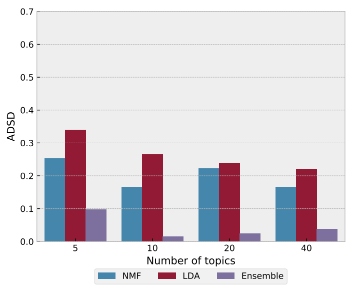
6.4.3 Results

In this section, we will present the results regarding topic model stability and context-richness.

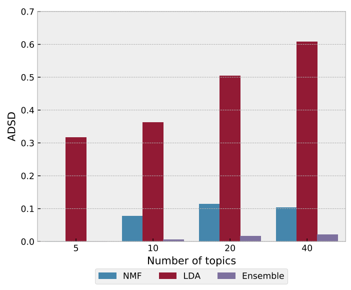
6.4.3.1 Stability

As stated previously, we used the stability metrics to decide on the topic modeling algorithm.

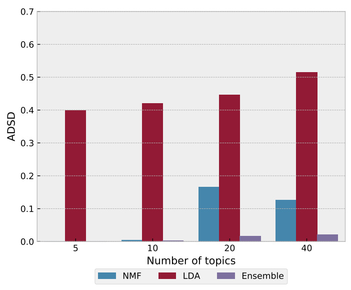
Figures 6.2a, 6.2b and 6.2c contain the ADSD results on the Yelp Hotels and Yelp Restaurants dataset respectively. Lower values of ADSD indicate higher topic model stability. We can clearly see how poor the ADSD results are for LDA across both datasets. NMF is better, but it is clear that the topic ensembling approach has the best ADSD results across both datasets.



(a) ADSD on the Yelp Hotel dataset



(b) ADSD on the Yelp Restaurant dataset

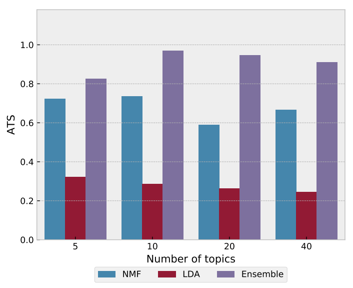


(c) ADSD on the TripAdvisor Hotel dataset

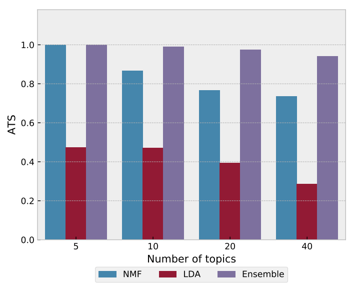
Figure 6.2: Topic model term difference

We can see a similar behavior when we measure the stability using ATS, as shown in Figures 6.3a, 6.3b and 6.3c for the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels dataset respectively. The topic ensembling approach clearly produces the most stable topic models, with NMF the second best method by this measure. LDA suffers a lot from the instability problem.

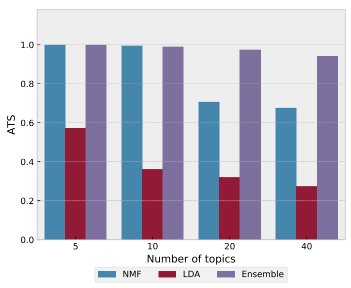
It is clear that if we want to be able to reproduce our results in the future, LDA is not a very good choice, NMF performs much better, but in the end it is the topic ensembling approach that has the highest stability. For this reason, we chose the topic ensembling algorithm to create the topic models that our recommender system will use to extract contextual information. In particular, we chose the k-fold version of the topic ensembling approach described in (Belford et al. 2018).



(a) ATS on the Yelp Hotel dataset



(b) ATS on the Yelp Restaurant dataset



(c) ATS on the TripAdvisor Hotel dataset

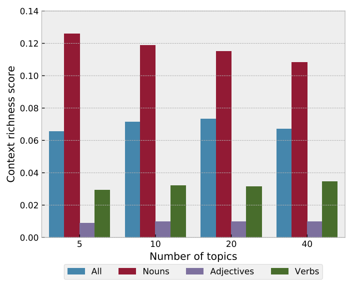
Figure 6.3: Topic model pairwise stability

As a side note, it is alarming to see how unstable are the topic models produced by LDA. In our implementation, we used the `gensim` package for `Python`[4], and despite increasing the number of iterations, the stability results stayed the same. In their work, (Belford et al. 2018) also measured the stability of LDA but instead using the `mallet` implementation[5]. In their tests, LDA showed higher values for stability but still remained behind NMF and topic ensembling.

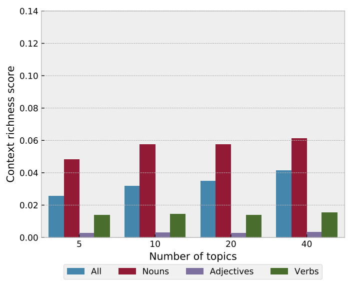
6.4.3.2 Context-richness

As stated previously, we used the context-richness metrics to decide which types of words and which types of reviews are going to be used to build the topic models.

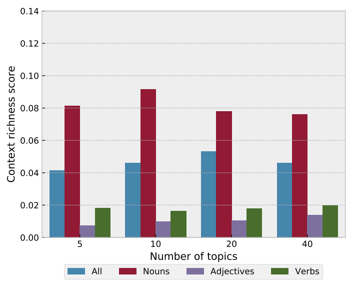
Figures 6.4a, 6.4b and 6.4c show the context-richness for the different types of words on the Yelp Hotels and Yelp Restaurants dataset, respectively.



(a) Context-richness of topic models on the Yelp Hotel dataset



(b) Context-richness of topic models on the Yelp Restaurant dataset



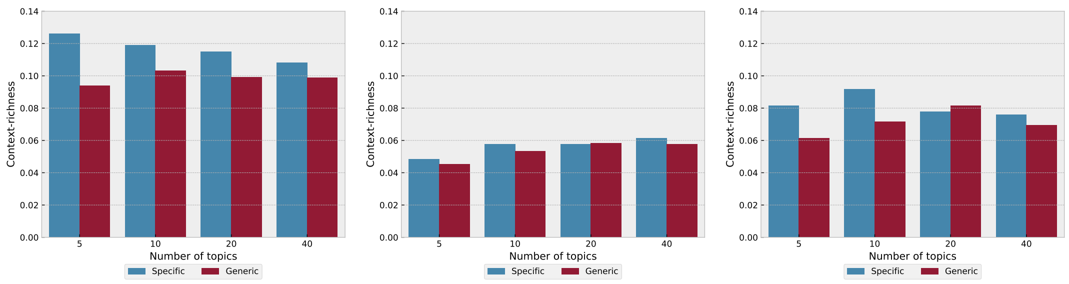
(c) Context-richness of topic models on the TripAdvisor Hotel dataset

Figure 6.4: Topic model context-richness compared by part-of-speech

We can see how using only nouns yields higher values for context-richness, regardless of the chosen number of topics. This does seem to match with intuitions when one thinks about words such as “birthday”, “conference”, “friends”, “wife”, “children”, “pets”, etc.

We can see in Figures 6.5a, 6.5b and 6.5c that in all but one of the cases (where there is a virtual draw), building topic models with specific reviews translates into context-richer topic models. It is also evident that users use more contextual words when writing reviews about hotels than when writing reviews about restaurants.

[4]`https://radimrehurek.com/gensim/`
[5]`http://mallet.cs.umass.edu/`



(a) Context-richness of topic models on the Yelp Hotel dataset (b) Context-richness of topic models on the Yelp Restaurant dataset (c) Context-richness of topic models on the TripAdvisor Hotel dataset

Figure 6.5: Topic model context-richness compared by review type

Based on these results, we decided to use only noun words and specific reviews to create our topic models.

6.5 Evaluation of RCRecommender

6.5.1 Evaluation methodology

After an initial shuffling of the dataset, we split the dataset into two parts: one for training the topic model and another one to use with the recommender. The reason for doing this is to reduce computational time, as creating an ensemble topic model is very expensive (for each topic model, 100 runs of NMF have to be executed), so creating a topic model for each hyperparameter combination in each cross-validation fold was too expensive. So with one half of the dataset we train the topic model and with the other half we train the recommender.

To evaluate RCRecommender, we used the *relevant-plus-unseen* methodology proposed in (Cremonesi et al. 2013). For the recommender dataset, we split the dataset into two parts, a training set $Train$ and a probe set. We get rid of all of the ratings below 5 stars in the probe set and refer to the remainder as the test set $Test$. In this way, we are ensuring that $Test$ is composed only of items that are relevant for the users.

We use the reviews from the training set $Train$ to train the topic model and transform the reviews and ratings into the format required by the Factorization Machine. For each of the records in the test set $Test$, $\langle\langle u, i, \tau_{u,i}\rangle, 5\rangle$, we find an additional m items that have not been rated by user u. This gives us $m + 1$

candidates (the preferred item from the test set plus the m unrated items). After this, we use the Factorization Machine that we have trained with the training set $Train$ and make rating predictions for each of the $m + 1$ items. Once all of the predictions have been made, then the top-n items are analyzed ($n = 10$). If the preferred item is among the top-n, then we count that as a hit; otherwise, we count it as a miss. We do that for every preferred item in the $Test$ set and produce the average hit rate, which in this case is the recall or the proportion of items in the test set in which we get a hit.

A thing to note here is that, as mentioned before, Rich-Context is a context-driven recommender that requires the user to supply a contextual query q, to which we also apply the topic model. In reality the user will supply this query in the form of a few words that describe the desired contextual situation. To conduct the offline experiments that we are presenting here, we assigned $q = \tau_{u,i_p}$, that is, the review written by user u about the preferred item i_p. All of the unrated items i will use the same value of $q = \tau_{u,i_p}$ to predict ratings.

To evaluate and choose the best hyperparameter values we do model selection with the help of grid search (in a similar fashion to what we did in Section 6.3.1). We split the dataset into five parts, using four for training and one for testing. For each combination of hyperparameter values, we train a model using the four parts of the dataset and test it on the remaining part. We repeat this five times and select the model with the best performance (measured in RMSE or recall) across the five folds. To evaluate RC we only varied one hyperparameter variable: the number of topics. The number of latent factors was always set to 10. For the Yelp Hotels dataset, we evaluated a variety of number of topics ranging from 2 to 60. For the Yelp Restaurants dataset we evaluated the range from 2 to 40 topics. For the TripAdvisor Hotels dataset we tried a different number of topics depending on the evaluation metrics because testing using the *relevant-plus-unseen* methodology took several days for a single run. For the rating prediction evaluation we evaluated the range from 2 to 40 topics, for the ranking prediction, we evaluated with 10, 20, 30 and 40 topics. In Section 6.4.3 we report the best values for the hyperparameters.

For all of our experiments, we used 5-fold cross-validation and relied on the `RiVaL` framework (Said & Bellogín 2014) to do the splits and the performance evaluation part. We used the `libFM` implementation of Factorization Machines (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.

6.5.2 Metrics

We measured the performance of our recommender using two types of metrics: rating prediction and ranking prediction. In general, ranking prediction metrics are preferred over rating prediction because they are better suited to evaluating top-n recommendation (Steck 2013, Liu & Yang 2008, Karatzoglou et al. 2013). However, we still use rating prediction metrics for the sake of broadening our comparisons with other approaches and because it has little extra cost in doing it. For rating prediction we used the RMSE. For ranking prediction we used the Recall metrics.

RMSE is defined as:

$$RMSE = \sqrt{\frac{\sum(\hat{r} - r)^2}{|R|}}, \quad \forall\, r \in R \tag{6.2}$$

where $\hat{r}$ is the predicted rating, r is the true rating and $|R|$ is the total number of known ratings.

The recall metric is computed using the *relevant-plus-unseen* methodology proposed in (Cremonesi et al. 2010) that we just explained.

Recall is defined as:

$$recall = \frac{\#\text{hits}}{m} \tag{6.3}$$

In our experiments, $m = min(1000, |I_{Test}|)$, where $|I_{Test}|$ is size of the set of items that are part of the $Test$ set. If there are more than 1000 unseen items for a user then 1000 are taken at random, otherwise, all of the unseen items are considered.

In the Yelp Hotels dataset there are fewer than 1000 items (98 in total as seen on Table 6.2). For this dataset we set $m = 97$, as $|I_{Test}| = 98$, thus having 1 relevant item plus up to 97 unseen items. This translates into higher recall numbers because it is easier to have a hit when there are 98 items than when there are 1000.

6.5.3 Other recommenders

We compared Rich-Context against three non-context-aware recommenders and against six other CARS which were described in Chapter 2. The three non-context-aware recommenders are Factorization Machines (Rendle 2010), Bayesian Probabilistic Matrix Factorization (BPMF) (Salakhutdinov & Mnih 2008) and Biased MF (Koren 2009). The six CARS are CAMF-C, CAMF-CI, CAMF-CU, CAMF-CUCI (Baltrunas et al. 2011), DCR (Zheng et al. 2012*a*) and DCW (Zheng et al. 2013).

For our experiments, the non-context-aware recommenders have been trained using only the ratings information and no contextual information from reviews. We trained several recommenders using the hyperparameter values presented in Table 6.4 and report here the results of the best performance of each algorithm. Since the contextual information is not explicitly available in the datasets we are using, and it is required by all of the six CARS algorithms we are using (in binary format), we designed two strategies to fill the required contextual values.

The first strategy, which we call *predefined context*, consists in defining a list of possible contextual situations and a set of keywords associated to each of the situations. Each contextual situation represents a feature in our feature vector and by default all of the contextual features are set to zero. We then search through the review trying to match any of the contextual keywords. When we find a contextual keyword we switch the value of the contextual situation associated to that keyword from zero to one. In Tables 6.5 and 6.6, we can see the list of contextual situations and the keywords associated to it for hotels and restaurants. These are contextual situations that we thought could influence the user's choice. Each contextual situation is defined by a set of keywords.

The second strategy, which we call *top topic words*, consists in creating a topic model with 10 topics. For this strategy, we will have 10 contextual features that are set to zero by default. Then the words with the highest weight in each topic are taken and the rest are discarded. We call these the *top words*. If a top word is present in the review, then we switch its corresponding feature to one.

Table 6.4: The list of candidate hyperparameters used to train the classifiers

Algorithm	**Parameter**	**Context support**	**Values**
Factorization Machine	Number of factors	No	$5, 10, 15, 20$
BPMF	Number of factors	No	$10, 20, 30, 40$
Biased MF	Number of factors	No	$10, 20, 30, 40$
CAMF-C	Number of factors	Yes	$10, 20, 30, 40$
CAMF-CI	Number of factors	Yes	$10, 20, 30, 40$
CAMF-CU	Number of factors	Yes	$10, 20, 30, 40$
CAMF-CUCI	Number of factors	Yes	$10, 20, 30, 40$
DCR	Number of PSO particles	Yes	$3, 5$
DCW	Number of PSO particles	Yes	$3, 5$
	Context similarity threshold	Yes	$0.5, 0.8$

6.5.4 Results for all users

Ranking prediction In Figures 6.6a 6.6b and 6.6c, we can see the ranking prediction performance (Recall@10) of Rich-Context on the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels datasets, respectively. We compare the performance of Rich-Context against three non-context-aware recommenders and against the six aforementioned CARS.

We can see how the performance of Rich-Context varies depending on the number of topics. Note that for the Yelp Restaurants and TripAdvisor Hotels datasets, Rich-Context is able to beat all the other state-of-the-art recommenders regardless of the number of topics. This is not the case for the Yelp Hotels dataset, where only with a couple of choices (41 and 53 topics) is Rich-Context able to beat Factorization Machines. The lower performance on the Yelp Hotels dataset can be explained by the fact that the dataset is quite small and there is a lot of sparsity when the contextual variables are introduced. In this scenario, there are not enough ratings under the same contextual situations for Rich-Context to find a pattern. Regardless of that, there are two cases in which Rich-Context is able to beat Factorization Machines (with 41 and 53 topics). DCR and DCW were unable to produce recommendations for the ranking strategy (which require to make predictions for all of the unseen items in the dataset for a user) in the Yelp Restaurants dataset due to running out of memory. None of the CARS, BPMF and Biased MF were able to cope with the TripAvisor hotels dataset for ranking predictions due to its size. All of them ran out of memory. The *predefined context* strategy was the one that returned the best ranking prediction

Table 6.5: List of keywords that refer to hotel-related contextual situations

Contextual Situation	**Keywords**
Accessibility	Handicap, wheelchair, ramp
Airport	Flight, bus, airport, plane, shuttle, transportation
Anniversary	Wife, anniversary, hubby, weekend, husband
Business	Colleague, business, work, coworker, job
Conference	Conference, attended, group, convention, meeting
Discount	Deal, groupon, discount, hotwire, priceline
Fall	September, november, october, halloween, fall
Family	Family, child, son, sibling, grandparent, girl, father, grandmother, dad, parent, grandpa, mom, grandma, kid, boy, sister, daughter, brother, grandfather, aunt, mother, uncle
Festivities	Thanksgiving, holiday, christmas
Gambling	Slot, casino, gamble, poker, roulette
Holiday	Getaway, vacation, staycation, holiday
Outdoor	Horse, court, kart, cart, tenni, field, bike, fitness, golf, cabana, training, exercise, cycle
Parking	Car, driver, valet, parking
Party	Dj, group, music, nightlife, party, friend
Pets	Pet, dog, cat
Relax	Stress, relaxing, relax, getaway, quiet, jacuzzi, treatment, facial, spa, steam, massage, relief
Romantic	Bf, gf, romantic, wife, fiancee, getaway, anniversary, romance, fiance, date, girlfriend, hubby, weekend, husband, boyfriend
Sport event	Basketball, tournament, football, field, game, baseball, ticket, match
Spring	May, spring, easter, march, april
Summer	Summer, july, june, august
Wedding	Ceremony,marriage, reception, wedding
Weekday	Monday, tuesday, weekday, thursday, wednesday
Weekend	Sunday, friday, weekend, saturday
Winter	December, january, february, winter, christmas

results across all of the tested CARS.

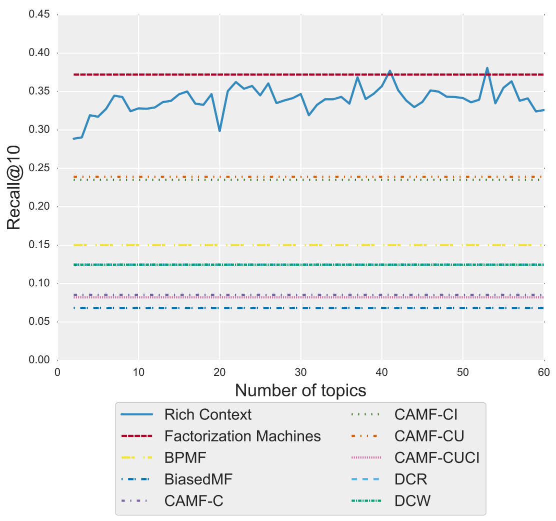
Table 6.7 summarizes the ranking prediction results, showing that in its best version, Rich-Context improves the performance of the best state-of-the-art algorithm (which in all cases was Factorization Machines) by 2.32%, 55.07% and 30.15% on the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels datasets, respectively.

Table 6.6: List of keywords that refer to restaurant-related contextual situations

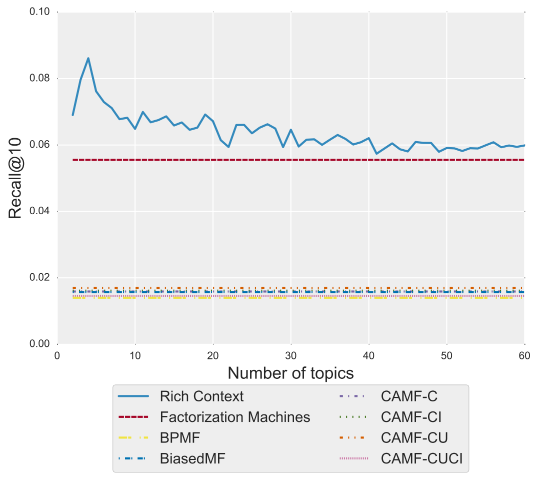
Contextual Situation	**Keywords**
Accessibility	Handicap, wheelchair, ramp
Birthday	Birthday, event, celebration
Breakfast	Omelette, pancakes, morning, breakfast, waffle, brunch
Dinner	Dinner, evening, night
Discount	Deal, coupon, groupon, discount
Fall	September, november, october, halloween, fall
Family	Dad, sister, daughter, parent, mother, grandma, father, brother, son, grandmother, sibling, aunt, mom, grandparent, grandpa, uncle, grandfather
Friends	Group, boy, guy, girl, friend
Karaoke	Song, music, karaoke
Kids	Family, boy, child, girl, kid
Lunch	Noon, lunch, afternoon
Outdoor	Outdoor, summer, outside, patio
Parking	Car, driver, valet, parking
Party	Group, people, club, disco, music, nightlife, night, party, guy, friend
Romantic	Bf, gf, romantic, wife, fiancee, anniversary, fiance, night, girlfriend, hubby, weekend, date, husband, boyfriend
Sports	Basketball, tv, nfl, football, sports, game, baseball, match
Spring	May, spring, easter, march, april
Summer	Summer, july, june, august
Takeaway	Delivery, deliver, drive, takeout, takeaway, thru
Weekday	Monday, tuesday, weekday, thursday, wednesday
Weekend	Sunday, friday, weekend, saturday
Winter	December, january, february, winter, christmas
Work	Colleague, office, coworker, job, workplace, business, meeting

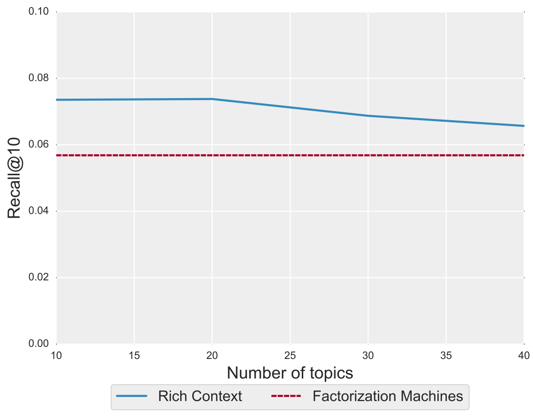
We used the Wilcoxon signed rank test to measure the statistical significance of the ranking prediction results. We found out that the results for the Yelp Restaurants and TripAdvisor Hotels are statistically significant (Wilcoxon signed rank with $p < 0.05$), while there was not statistical significance for the Yelp Hotels dataset.

To calculate the ranking performance improvement we used the following equa-



(a) Yelp Hotels





(b) Yelp Restaurants (c) TripAdvisor Hotels

Figure 6.6: Rating predictions

tion:

$$recall\_improvement = \frac{rc\_recall}{fm\_recall} - 1 \tag{6.4}$$

Rating prediction In Figures 6.7a 6.7b and 6.7c, we can see the rating prediction performance (RMSE) of Rich-Context on the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels datasets, respectively. Again, we compare the performance of Rich-Context against three non-context-aware recommenders and against the six aforementioned CARS. For all of the datasets, we can see that Rich-Context has a better rating prediction than all of the state-of-the-art

Table 6.7: Recall@10 comparison for all users.

Algorithm	**Recall@10**		
	Yelp		**TripAdvisor**
	Hotels	**Restaurants**	**Hotels**
Rich-Context	**0.381**	**0.086**	**0.074**
Factorization Machine	0.372	0.056	0.057
BPMF	0.150	0.014	–
Biased MF	0.068	0.016	–
CAMF-C	0.085	0.016	–
CAMF-CI	0.235	0.016	–
CAMF-CU	0.239	0.017	–
CAMF-CUCI	0.082	0.015	–
DCR	0.125	–	–
DCW	0.125	–	–
Improvement over SOTA	**2.32%**	**55.07%**	**30.15%**

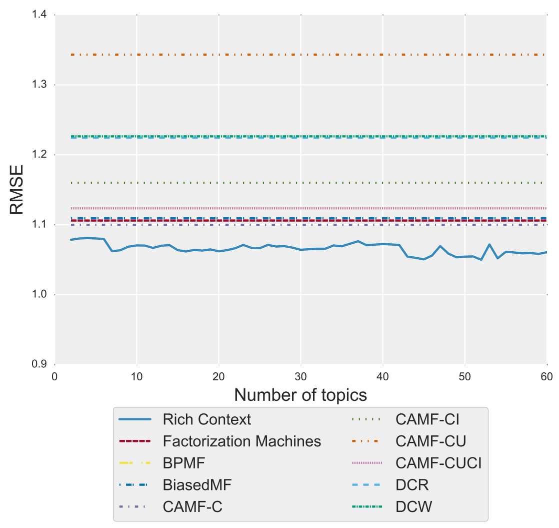
recommenders regardless of the chosen number of topics, except in the Yelp Restaurants dataset, in which, for the first 11 topics, the results of Rich-Context are almost the same as Factorization Machines, but after using 12 topics or more Rich-Context outperforms Factorization Machines. DCR and DCW ran out of memory and were unable to produce recommendations for the rating strategy using the TripAdvisor Hotels dataset. DCR also ran out of memory and was unable to produce recommendations using the Yelp Restaurants. The *top words* strategy was the one that returned the best rating prediction results across all of the tested CARS.

To calculate the rating performance improvement we used the following equation:

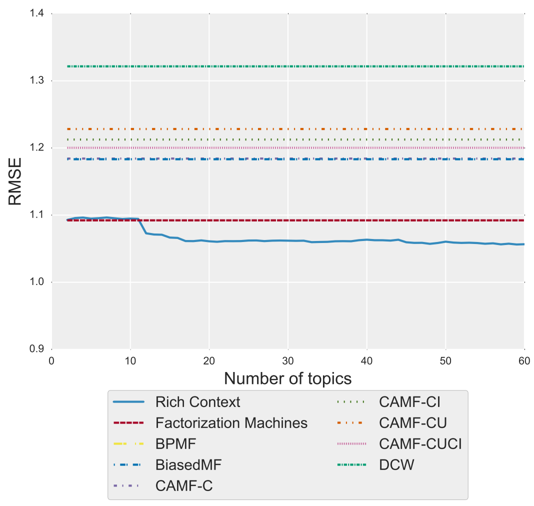
$$rmse\_improvement = 1 - \frac{rc\_rmse}{fm\_rmse} \tag{6.5}$$

Table 6.8 summarizes the rating prediction results, showing that, in its best version, Rich-Context improves the performance of the best state-of-the-art algorithm by 4.45%, 3.29% and 8.99% on the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels datasets, respectively.

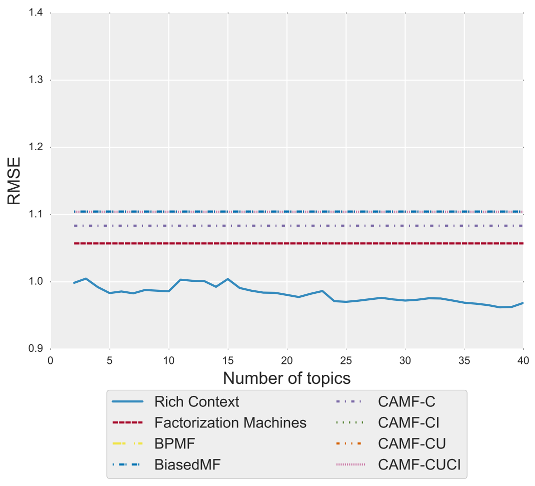
We used the two-sided Student-t test to measure the statistical significance of the rating prediction results (note that to measure statistical significance, we



(a) Yelp Hotels



(b) Yelp Restaurants



(c) TripAdvisor Hotels

Figure 6.7: Rating predictions for all users

use Student-t for rating prediction and Wilcoxon for ranking prediction). We found out that the results across all of the datasets are statistically significant (two-sided Student t-test with $p < 0.05$).

We note that the results we present here are slightly different from the ones presented in our previously-published work (Peña & Bridge 2017). The differences come due to the fact that, in order to speed the evaluation process, we reduced the number of cross-validation iterations from 10 to 5. For the Yelp Hotels dataset, we made another modification: we increased the number of additional items used in the recall evaluation from 90 to 98. The consequence is a lower recall: it is easier to predict 1 item among 90 than among 98. This

Table 6.8: RMSE comparison for all users.

Algorithm	**RMSE**		
	Yelp		**TripAdvisor**
	Hotels	**Restaurants**	**Hotels**
Rich-Context	**1.050**	**1.056**	**0.962**
Factorization Machine	1.106	1.092	1.057
BPMF	1.495	1.405	1.435
Biased MF	1.109	1.183	1.104
CAMF-C	1.100	1.184	1.084
CAMF-CI	1.160	1.213	1.104
CAMF-CU	1.343	1.228	1.420
CAMF-CUCI	1.123	1.200	1.104
DCR	1.224	–	–
DCW	1.226	1.321	–
Improvement over SOTA	**4.45%**	**3.29%**	**8.99%**

change in the number of additional items explains why the difference is bigger in the Yelp Hotels dataset compared to the Yelp Restaurants dataset.

6.5.5 Recommendations for brand-new users

It is frequent to have visitors to websites where no information is known about them. In these cases recommender systems can struggle to make accurate recommendations. We are not talking here about cold-start users, about whom little is known. But users about whom we know nothing. They may not log in, or they may not even have a user account with the system. Instead, they arrive at the website and expect to enter a query (e.g. their travel intention), and expect to receive useful context-driven recommendations. We refer to these users as brand-new users. In this section we explore how Rich-Context performs against other models when making recommendations to brand-new users.

To see if Rich-Context could help improve the recommendations made to brand-new users, i.e., users who have rated no items, we repeated all of the experiments but with a test set composed only of users that were not present in the training set. The methodology we used for running the brand-new users experiments is exactly the same as the one we used for the previous experiments and the obtained behavior is very similar to the previous experiments.

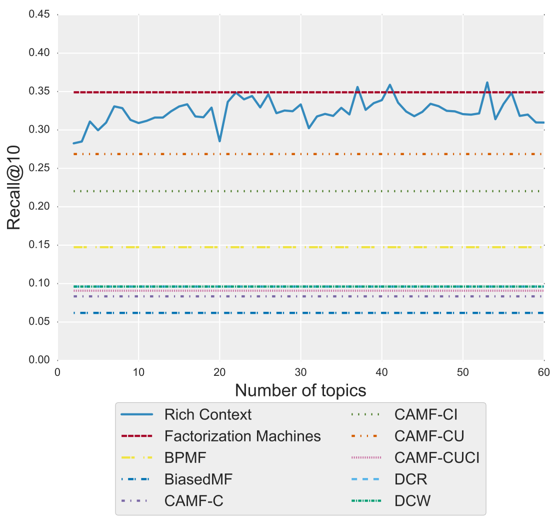
Ranking prediction In Figures 6.8a 6.8b and 6.8c, we can see the ranking prediction performance (Recall@10) of Rich-Context on the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels datasets, respectively. We compare the performance of Rich-Context against three non-context-aware recommenders and against the six aforementioned CARS.

Again, we can see how the performance of Rich-Context varies depending on the number of topics. Note that for the Yelp Restaurants and TripAdvisor Hotels datasets, Rich-Context is able to beat all the other state-of-the-art recommenders regardless of the number of topics. This is not the case for the Yelp Hotels dataset, where only with three choices (37, 41 and 53 topics) is Rich-Context able to beat Factorization Machines. The lower performance on the Yelp Hotels dataset can be explained by the fact that the dataset is quite small and there is a lot of sparsity when the contextual variables are introduced. In this scenario, there are not enough ratings under the same contextual situations for Rich-Context to find a pattern. Regardless of that, there are three cases in which Rich-Context is able to beat Factorization Machines (with 37, 41 and 53 topics). DCR and DCW were unable to produce recommendations for the ranking strategy in the Yelp Restaurants dataset due to running out of memory. None of the CARS, BPMF and Biased MF were able to cope with the TripAvisor hotels dataset for ranking predictions due to its size. All of them ran out of memory. The *predefined context* strategy was the one that returned the best ranking prediction results across all of the tested CARS.

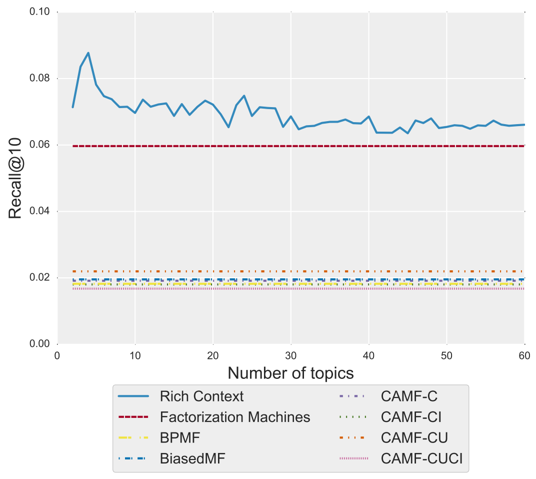
Table 6.9 summarizes the brand-new-users ranking prediction results, showing that in its best version, Rich-Context improves the performance of the the best state-of-the-art algorithm (which in all cases was Factorization Machines) by 3.67%, 47.00% and 28.5% on the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels datasets, respectively.

As before, we used the Wilcoxon signed rank test to measure the statistical significance of the ranking prediction results. We found out that the results for the Yelp Restaurants and TripAdvisor Hotels are statistically significant (Wilcoxon signed rank with $p < 0.05$), while there was not statistical significance for the Yelp Hotels dataset.

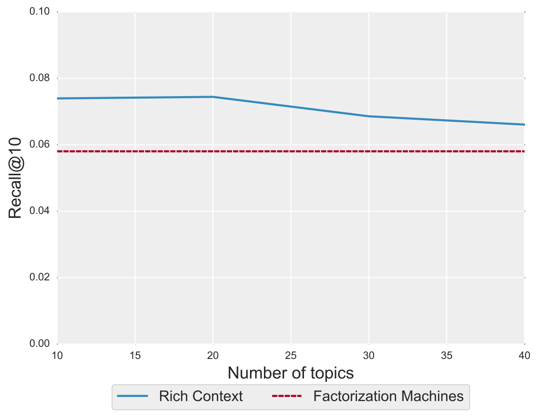
Rating prediction In Figures 6.9a 6.9b and 6.9c, we can see the rating prediction performance (RMSE) of Rich-Context on the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels datasets, respectively. For all of the datasets, we can see



(a) Yelp Hotels



(b) Yelp Restaurants



(c) TripAdvisor Hotels

Figure 6.8: Ranking predictions for brand-new users

that Rich-Context has a better rating prediction than all of the state-of-the-art recommenders regardless of the chosen number of topics, except in the Yelp Restaurants dataset, in which, for the first 11 topics, the results of Rich-Context are almost the same as Factorization Machines, but after using 12 topics or more Rich-Context outperforms Factorization Machines. DCR and DCW ran out of memory and were unable to produce recommendations for the rating task in the Yelp Restaurants and TripAdvisor Hotels datasets. The *top words* strategy was the one that returned the best rating prediction results across all of the tested CARS.

Table 6.10 summarizes the rating prediction results, showing that in its best version, Rich-Context improves the performance of the best state-of-the-art al-

Table 6.9: Recall@10 comparison for brand-new users.

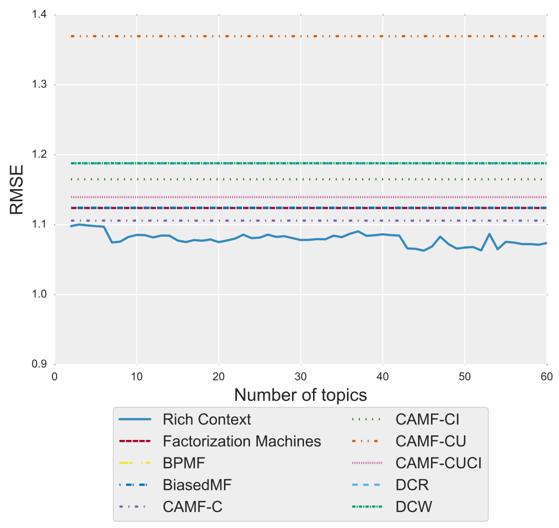
Algorithm	**Recall@10**		
	Yelp		**TripAdvisor**
	Hotels	**Restaurants**	**Hotels**
Rich-Context	**0.362**	**0.088**	**0.075**
Factorization Machine	0.349	0.060	0.058
BPMF	0.147	0.018	–
Biased MF	0.062	0.020	–
CAMF-C	0.083	0.019	–
CAMF-CI	0.220	0.018	–
CAMF-CU	0.269	0.022	–
CAMF-CUCI	0.091	0.017	–
DCR	0.096	–	–
DCW	0.096	–	–
Improvement over SOTA	**3.67%**	**47.00%**	**20.50%**

gorithm by 5.43%, 4.54% and 9.96% on the Yelp Hotels, Yelp Restaurants and TripAdvisor Hotels datasets, respectively.

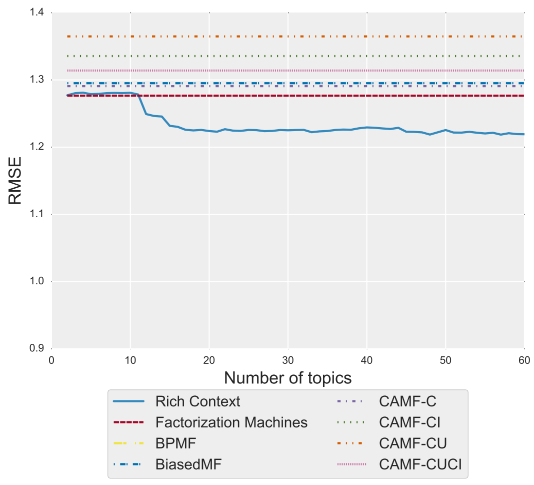
As before, we used the two-sided Student-t test to measure the statistical significance of the rating prediction results. We found out that the results across all of the datasets are statistically significant (two-sided Student t-test with $p < 0.05$).

Table 6.10: RMSE comparison for new users across all datasets.

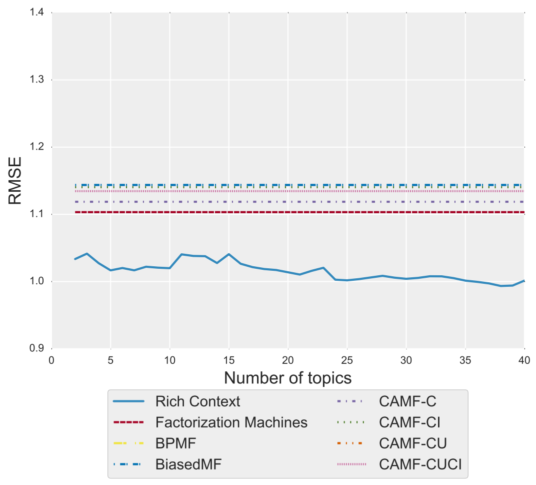
Algorithm	**RMSE**		
	Yelp		**TripAdvisor**
	Hotels	**Restaurants**	**Hotels**
Rich-Context	**1.063**	**1.219**	**0.993**
Factorization Machine	1.124	1.277	1.103
BPMF	1.543	1.653	1.509
Biased MF	1.124	1.295	1.144
CAMF-C	1.106	1.291	1.119
CAMF-CI	1.165	1.335	1.140
CAMF-CU	1.370	1.365	1.471
CAMF-CUCI	1.139	1.314	1.135
DCR	1.188	–	–
DCW	1.188	–	–
Improvement over SOTA	**5.43%**	**4.54%**	**9.96%**



(a) Yelp Hotels



(b) Yelp Restaurants



(c) TripAdvisor Hotels

Figure 6.9: Rating predictions for new users

6.6 Implementation

To implement our model, we used `Python` 2.7.11. In particular we used the version that comes with `Anaconda`[6] 2-5.0. To identify the language in which a review was written we used the `langdectet` 1.0.7 package. To produce the part-of-speech tags for the words, we used `nltk`'s[7] `PerceptronTagger`. To lemmatize the text, we used the `Pattern`[8] 2.6 package. All of the classifiers we used to discriminate between specific and generic reviews come from the

[6]`https://anaconda.org/anaconda/python`
[7]`https://www.nltk.org/`
[8]`https://www.clips.uantwerpen.be/pages/pattern-en`

`scikit-learn`[9] 0.19.0 package in Python. The LDA implementation we used comes from `gensim`[10] 0.12.2. We used Derek Greene's topic ensemble implementation for topic modeling[11]. For splitting the data and evaluating the recommendation results, we used `RiVaL`[12] 0.3. We used Steffen Rendle's implementation of Factorization Machines, which he calls `libFM`[13] 1.4.2. For the baseline CARS (CAMF-C, CAMF-CI, CAMF-CU, CAMF-CUCI, DCR, DCW), BPMF and `Biased MF` , we used `CARSKit`[14]

6.7 Summary

Overall we can see that for all of the datasets we worked with, Rich-Context is able to outperform the state-of-the-art. In the two largest datasets, Yelp Restaurants and TripAdvisor Hotels, the ranking prediction improvements are significantly bigger as there is less contextual sparsity. The improvements over the rating predictions are also significant across the three datasets. These results show that Rich-Context is not only very accurate, but also very consistent. Comparing Rich-Context against several state-of-the-art recommenders, including six that support contextual information, shows how good our recommender is. Moreover, the fact that we are using the `RiVaL` evaluation framework to make the splits of the data and the evaluation of the results gives us the confidence to state that our recommender successfully exploits contextual information to make more accurate recommendations.

We can also see from these results that Rich-Context outperforms current CARS in both small and large datasets, but it is clear that in large datasets the gap between Rich-Context and the state-of-the-art CARS is much bigger.

[9] `http://scikit-learn.org/`
[10] `https://radimrehurek.com/gensim/`
[11] `https://github.com/derekgreene/topic-ensemble`
[12] `https://github.com/recommenders/rival`
[13] `http://www.libfm.org/`
[14] `https://github.com/irecsys/CARSKit`

Chapter 7

Conclusions

This chapter summarizes what has been achieved in this thesis and discusses future work.

7.1 Summary

In this thesis we addressed the problem of making context-driven recommendations. Given the fact that explicit contextual information is rarely available in real-world datasets, we exploited user-generated reviews to extract contextual information in order to make recommendations that correspond with the users' goals. We introduced Rich-Context, a context-driven recommender system that mines contextual information from user-generated reviews to produce context-driven recommendations.

To be able to extract the contextual information from the reviews, we made the assumption that there are two types of reviews, ones that describe experiences, which we call *specific* reviews, and others that do not, which we call *generic*. Given the fact that every experience has a context associated to it, specific reviews tend to have more contextual information than their generic counterpart.

To discriminate between specific and generic reviews, we created RCClassifier. RCClassifier assigns POS tags to the words in the reviews and creates features based on those tags that help to distinguish between the two types of reviews. By trying out different classification algorithms and using resampling techniques, we were able to boost the performance of RCClassifier.

To extract the contextual information out of the reviews, we designed RCMiner. RCMiner builds a topic model using only the specific reviews, and after applying the topic models to all of the reviews, it determines which topics appear more frequently in specific reviews than in generic ones and labels them as contextual topics. Once the contextual topics have been identified, every document can then be represented as a vector of weights of the contextual topics.

One of the factors crucial to the correct extraction of contextual information out of the reviews was to produce high quality topic models. To achieve this, we designed a methodology based on metrics first to measure the stability of the topic modeling algorithms, and then to measure the amount of contextual information contained in the resulting topic models. The former metrics helped us to decide which topic modeling algorithm should be used while the latter helped us decide what type of data should be given to the topic modeling algorithm. In this way, we were able to extract contextual information out of the reviews that was useful for the purpose of producing recommendations.

RCMiner has the additional advantage that it is able to extract context in an unsupervised way, without the need for human intervention or the need to pre-define contextual keywords. This has the extra benefit that the extracted context is open-ended. The extracted context reflects what the users have written about and is not constrained to pre-defined contextual situations, as in most CARS.

Once the contextual information has been obtained, RCRecommender takes it along with the ratings and produces recommendations. After experimenting with several algorithms that support side-information, we found that Factorization Machines produced the best results both in ranking and rating prediction performance and in terms of coverage.

Evaluation of Rich-Context compared to nine state-of-the-art recommender systems, including six CARS, shows that Rich-Context has superior performance in both rating and ranking prediction across multiple publicly available, sparse, real-world datasets. Rich-Context is able to beat all of the state-of-the-art recommender across all of the datasets. By using the recommender systems evaluation library `RiVaL` (Said & Bellogín 2014), we bring transparency and confidence to our results.

Finally, Rich-Context was also evaluated for users who were present in the test set but not in the training set, whom we call *brand-new users*. This type of user is quite common on websites and are also known as first-time visitors. We compared Rich-Context to nine state-of-the-art recommender systems, including six CARS, and the results were very similar to the case where we did the evaluation on all users. Again Rich-Context showed superior performance in terms of rating and ranking prediction across all of the datasets.

7.2 Future Work

Provision of explanations Given the fact that Rich-Context creates topic models in order to produce recommendations and the fact that topic models contain information about what is being discussed about items, we can exploit this, along with the already extracted contextual information, to produce explanations about why a certain item is being recommended. The advantage of producing explanations in this way is that they are consistent with the recommendation mechanism. This will result in a more transparent recommender system in which users are able to understand why a certain item was suggested, increasing their overall trust in the recommender system.

Improving the metrics that measure the quality of topic models One of the drawbacks of Rich-Context and, for that matter, any other recommender system that is based on unstructured text corpora is that the training time of the algorithm can be quite high when dealing with a large number of documents (for instance the TripAdvisor hotels dataset). At the time of using the recommender, this is not a major issue since the prediction time is very low. However, during the process of hyperparameter tuning this becomes a problem, in particular at the time of selecting the best value for the number of topics k.

As seen in Chapter 6, the number of topics used to build the topic model has a direct impact on the performance of Rich-Context both for rating and ranking prediction tasks. Differently to the number of factors in a latent-factor model, a higher number of topics does not lead to better recommendations. It is therefore crucial to design a metric that can reveal the quality of a topic model for recommendation purposes, so that Rich-Context does not have to be trained for each candidate value of k. So far, we were able to use metrics to help us choose the best topic modeling algorithm (see Section 5.6.1) and to determine the data that should be used to train the topic models (see Section 5.6.2). A metric to determine the best value for k is necessary.

Mixing context extraction with feature extraction and sentiment analysis With the goal of improving further the accuracy of the recommendations and with the additional benefit of improving the above proposed explanations for recommendations, features that describe the items can also be extracted from the reviews along with the sentiment towards those features. Two recent approaches that are able to extract features are (Musto et al. 2017) and (Chen et al. 2015), while (Diao et al. 2014) extracts both sentiments and features. If a future version of Rich-Context were to mix the mined features with the context that Rich-Context can already extract, better recommendations might be produced. Not only that, but by extracting the features and sentiments, along with the contextual information that is relevant for users, there is a deeper understanding of what things are important for each user. This understanding can then be used to produce better, clearer and more transparent explanations that increase the trust that users have in the recommender.

Applications to other domains Topic models have been successfully used to make predictions by exploiting unstructured text in other domains. For instance, in (Lehman et al. 2012, 2014, Zalewski et al. 2017) a model that makes predictions based on hospital discharge summaries of intensive care units is presented. Their goal is to identify patients that have a high mortality risk at the time of being discharged based on vital signs and discharge summaries. Similarly to Rich-Context, this approach does not require expert intervention in order to define which information is relevant. This problem can be modeled as a recommendation problem in which the users are the patients, the items are the treatments to recommend and the interaction is the mortality of the patient after one year of being discharged. Following a variation of the methodology used by Rich-Context, recommendations for treatments can be suggested with the goal of reducing the mortality rate. The application of Rich-Context in other areas with unstructured text such as law or news could also be explored.

Acronyms

ADSD Average Descriptor Set Difference.

ATS Average Term Stability.

BPMF Bayesian Probabilistic Matrix Factorization.

CAMF Context-Aware Matrix Factorization.

CARS Context-Aware Recommender Systems.

COR Contextual Opinions Recommender.

CSCB Cold Start Context-Based Hotel Recommender.

CSLIM Contextual SLIM.

CTMS Contextual Topic Model Score.

CTS Contextual Topic Score.

DCR Differential Context Relaxation.

DCW Differential Context Weighting.

ENN Edited Nearest Neighbors.

FM Factorization Machine.

HFT Hidden Factors as Topics.

HOSVD High Order Singular Value Decomposition.

JMARS Jointly Modeling Aspects, Ratings and Sentiments.

JST Joint Sentiment-Topic Model.

k-NN k-Nearest Neighbour.

LCMF Latent Context Matrix Factorization Recommendation.

LDA	Latent Dirichlet Allocation.
LRM	Linear Regression Model.
LSA	Latent Semantic Analysis.
MAE	Mean Absolute Error.
MCABSA	Multi-Criteria Aspect-Based Sentiment Analysis.
MSE	Mean Squared Error.
NCL	Neighborhood Cleaning Rule.
nDCG	Normalized Discounted Cumulative Gain.
NLP	Natural Language Processing.
NMF	Non-negative Matrix Factorization.
OPR	Opinionated Product Recommender.
pLSA	Probabilistic Latent Semantic Analysis.
POS	part-of-speech.
PRM	Probabilistic Regression Model.
PSO	Particle Swarm Optimization.
RC	Rich-Context.
RMR	Ratings Meet Reviews.
RMSE	Root Mean Squared Error.
SABRE	Sentiment Aspect-Based Retrieval.
SLIM	Sparse Linear Method.
SMOTE	Synthetic Minority Over-sampling Technique.
SOTA	state-of-the-art.
SVD	Singular Value Decomposition.
TC	Topic-Criteria.
TF-IDF	Term Frequency - Inverse Document Frequency.
TSC	Topic-Sentiment Criteria.

Nomenclature

$\mathbf{A}$ The document-term matrix.

$\mathcal{C}$ A contextual dimension.

C A set of contextual dimensions.

$\mathbf{c}$ A vector containing the contextual conditions for each context dimension.

c The index of a contextual dimension.

D A set of documents.

d A document.

$\mathbf{H}$ The topic-term matrix.

I A set of items.

i An item.

k The number of topics.

l The number of latent factors in a latent factor model.

$\mathcal{M}$ A topic model.

$\mathbf{M}$ An esemble matrix.

$\mathbf{P}$ The latent features user matrix.

$\mathbf{p}_u$ The latent features vector of user u.

$\mathbf{Q}$ The latent features item matrix.

$\mathbf{q}_i$ The latent features vector of item i.

$\mathbf{R}$ The ratings matrix.

R A set of ratings.

$\mathbf{r}$ A vector of ratings.

$r_{u,i}$ The rating that user u gave to item i.

$r_{u,i,\mathbf{c}}$ The rating that user u gave to item i under the contextual conditions $\mathbf{c}$.

$\hat{r}_{u,i}$ The predicted rating that user u would give to item i.

$\hat{r}_{u,i,\mathbf{c}}$ The predicted rating that user u would give to item i under the contextual conditions $\mathbf{c}$.

T A set of topics.

t A topic.

U A set of users.

u A user.

V A vocabulary.

$\mathbf{W}$ The document-topic matrix.

W A set of words.

w A word.

α The offset or average rating.

β_i The bias of item i.

β_u The bias of user u.

$\tau_{u,i}$ The review that user u wrote about item i.

Υ A set of reviews.

$\boldsymbol{\varphi}$ A vector of weights.

φ A weight.

References

Adomavicius, G. & Kwon, Y. (2007), ‘New recommendation techniques for multicriteria rating systems’, *IEEE Intelligent Systems* **22**(3), 48–55.

Adomavicius, G., Manouselis, N. & Kwon, Y. (2011), Multi-criteria recommender systems, *in* F. Ricci, L. Rokach, B. Shapira & P. B. Kantor, eds, ‘Recommender Systems Handbook’, Springer US, pp. 769–803.
URL: *http://dx.doi.org/10.1007/978-0-387-85820-3_24*

Adomavicius, G., Sankaranarayanan, R., Sen, S. & Tuzhilin, A. (2005), ‘Incorporating contextual information in recommender systems using a multidimensional approach’, *ACM Trans. Inf. Syst.* **23**(1), 103–145.
URL: *http://doi.acm.org/10.1145/1055709.1055714*

Adomavicius, G. & Tuzhilin, A. (2005), ‘Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions’, *IEEE Trans. on Knowl. and Data Eng.* **17**(6), 734–749.
URL: *https://doi.org/10.1109/TKDE.2005.99*

Adomavicius, G. & Tuzhilin, A. (2015), Context-aware recommender systems, *in* F. Ricci, L. Rokach & B. Shapira, eds, ‘Recommender Systems Handbook’, Springer US, Boston, MA, pp. 191–226.
URL: *http://dx.doi.org/10.1007/978-1-4899-7637-6_6*

Aggarwal, C. C. (2016), *Recommender Systems: The Textbook*, 1st edn, Springer Publishing Company, Incorporated.

Aletras, N. & Stevenson, M. (2013), Evaluating topic coherence using distributional semantics, *in* ‘Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013) – Long Papers’, Association for Computational Linguistics, Potsdam, Germany, pp. 13–22.
URL: *https://www.aclweb.org/anthology/W13-0102*

Arnold, C. W., Oh, A., Chen, S. & Speier, W. (2016), ‘Evaluating topic model

interpretability from a primary care physician perspective', *Computer Methods and Programs in Biomedicine* **124**, 67 – 75.
URL: *http://www.sciencedirect.com/science/article/pii/S0169260715002746*

Arora, S., Ge, R. & Moitra, A. (2012), Learning topic models – going beyond SVD, *in* 'Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science', FOCS '12, IEEE Computer Society, Washington, DC, USA, pp. 1–10.
URL: *http://dx.doi.org/10.1109/FOCS.2012.49*

Arun, R., Suresh, V., Veni Madhavan, C. E. & Narasimha Murthy, M. N. (2010), On finding the natural number of topics with latent dirichlet allocation: Some observations, *in* M. J. Zaki, J. X. Yu, B. Ravindran & V. Pudi, eds, 'Advances in Knowledge Discovery and Data Mining: 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part I', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 391–402.
URL: *https://doi.org/10.1007/978-3-642-13657-3_43*

Baltrunas, L., Ludwig, B. & Ricci, F. (2011), Matrix factorization techniques for context aware recommendation, *in* 'Proceedings of the Fifth ACM Conference on Recommender Systems', RecSys '11, ACM, New York, NY, USA, pp. 301–304.
URL: *http://doi.acm.org/10.1145/2043932.2043988*

Baltrunas, L. & Ricci, F. (2009), Context-based splitting of item ratings in collaborative filtering, *in* 'Proceedings of the Third ACM Conference on Recommender Systems', RecSys '09, ACM, New York, NY, USA, pp. 245–248.
URL: *http://doi.acm.org/10.1145/1639714.1639759*

Batista, G. E. A. P. A., Prati, R. C. & 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.
URL: *http://doi.acm.org/10.1145/1007730.1007735*

Bauman, K. & Tuzhilin, A. (2014), Discovering contextual information from user reviews for recommendation purposes, *in* '1st Workshop on New Trends in Content-based Recommender Systems co-located with the 8th ACM Conference on Recommender Systems', ACM, Foster City, Silicon Valley, California, USA, pp. 2–9.

Bazire, M. & Brézillon, P. (2005), Understanding context before using it, *in*

A. Dey, B. Kokinov, D. Leake & R. Turner, eds, 'Modeling and Using Context', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 29–40.

Belford, M., Namee, B. M. & Greene, D. (2016), Ensemble topic modeling via matrix factorization, *in* '24th Irish Conference on Artificial Intelligence and Cognitive Science'.

Belford, M., Namee, B. M. & Greene, D. (2018), 'Stability of topic modeling via matrix factorization', *Expert Systems with Applications* **91**, 159 – 169.
URL: *http://www.sciencedirect.com/science/article/pii/S0957417417305948*

Blei, D. M. (2012), 'Probabilistic topic models', *Commun. ACM* **55**(4), 77–84.
URL: *http://doi.acm.org/10.1145/2133806.2133826*

Blei, D. M., Ng, A. Y. & Jordan, M. I. (2003), 'Latent dirichlet allocation', *J. Mach. Learn. Res.* **3**, 993–1022.
URL: *http://dl.acm.org/citation.cfm?id=944919.944937*

Caputo, A., Basile, P., de Gemmis, M., Lops, P., Semeraro, G. & Rossiello, G. (2017), SABRE: A sentiment aspect-based retrieval engine, *in* C. Lai, A. Giuliani & G. Semeraro, eds, 'Information Filtering and Retrieval: DART 2014: Revised and Invited Papers', Springer International Publishing, Cham, pp. 63–78.
URL: *https://doi.org/10.1007/978-3-319-46135-9_4*

Chang, J., Gerrish, S., Wang, C., Boyd-graber, J. L. & Blei, D. M. (2009), Reading tea leaves: How humans interpret topic models, *in* Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams & A. Culotta, eds, 'Advances in Neural Information Processing Systems 22', Curran Associates, Inc., pp. 288–296.
URL: *http://papers.nips.cc/paper/3700-reading-tea-leaves-how-humans-interpret-topic-models.pdf*

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. (2002), 'Smote: Synthetic minority over-sampling technique', *J. Artif. Int. Res.* **16**(1), 321–357.
URL: *http://dl.acm.org/citation.cfm?id=1622407.1622416*

Chen, G. & Chen, L. (2015), 'Augmenting service recommender systems by incorporating contextual opinions from user reviews', *User Modeling and User-*

Adapted Interaction **25**(3), 295–329.
URL: *http://dx.doi.org/10.1007/s11257-015-9157-3*

Chen, L., Chen, G. & Wang, F. (2015), 'Recommender systems based on user reviews: the state of the art', *User Modeling and User-Adapted Interaction* **25**(2), 99–154.
URL: *http://dx.doi.org/10.1007/s11257-015-9155-5*

Cremonesi, P., Garzotto, F. & Quadrana, M. (2013), Evaluating top-n recommendations "when the best are gone", *in* 'Proceedings of the 7th ACM Conference on Recommender Systems', RecSys '13, ACM, New York, NY, USA, pp. 339–342.
URL: *http://doi.acm.org/10.1145/2507157.2507225*

Cremonesi, P., Koren, Y. & Turrin, R. (2010), Performance of recommender algorithms on top-n recommendation tasks, *in* 'Proceedings of the Fourth ACM Conference on Recommender Systems', RecSys '10, ACM, New York, NY, USA, pp. 39–46.
URL: *http://doi.acm.org/10.1145/1864708.1864721*

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. & Harshman, R. (1990), 'Indexing by latent semantic analysis', *Journal of the American Society for Information Science* **41**(6), 391–407.
URL: *http://dx.doi.org/10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9*

Desrosiers, C. & Karypis, G. (2011), A comprehensive survey of neighborhood-based recommendation methods, *in* F. Ricci, L. Rokach, B. Shapira & P. B. Kantor, eds, 'Recommender Systems Handbook', Springer US, pp. 107–144.
URL: *http://dx.doi.org/10.1007/978-0-387-85820-3_4*

Diao, Q., Qiu, M., Wu, C.-Y., Smola, A. J., Jiang, J. & Wang, C. (2014), Jointly modeling aspects, ratings and sentiments for movie recommendation (jmars), *in* 'Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '14, ACM, New York, NY, USA, pp. 193–202.
URL: *http://doi.acm.org/10.1145/2623330.2623758*

Dong, R., O'Mahony, M. P., Schaal, M., McCarthy, K. & Smyth, B. (2016), 'Combining similarity and sentiment in opinion mining for product recommenda-

tion', *Journal of Intelligent Information Systems* **46**(2), 285–312.
URL: *https://doi.org/10.1007/s10844-015-0379-y*

Dong, R., O'Mahony, M. P. & Smyth, B. (2014), Further experiments in opinionated product recommendation, *in* L. Lamontagne & E. Plaza, eds, 'Case-Based Reasoning Research and Development: 22nd International Conference, ICCBR 2014, Cork, Ireland, September 29, 2014 - October 1, 2014. Proceedings', Springer International Publishing, Cham, pp. 110–124.
URL: *https://doi.org/10.1007/978-3-319-11209-1_9*

Dong, R., Schaal, M., O'Mahony, M. P., McCarthy, K. & Smyth, B. (2013), Opinionated product recommendation, *in* S. J. Delany & S. Ontañón, eds, 'Case-Based Reasoning Research and Development: 21st International Conference, ICCBR 2013, Saratoga Springs, NY, USA, July 8-11, 2013. Proceedings', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 44–58.
URL: *https://doi.org/10.1007/978-3-642-39056-2_4*

Fleiss, J. L. (1971), 'Measuring nominal scale agreement among many raters.', *Psychological Bulletin* **76**(5), 378–382.

Ganu, G., Kakodkar, Y. & Marian, A. (2013), 'Improving the quality of predictions using textual information in online user reviews', *Inf. Syst.* **38**(1), 1–15.
URL: *http://dx.doi.org/10.1016/j.is.2012.03.001*

Greene, D., O'Callaghan, D. & Cunningham, P. (2014), How many topics? stability analysis for topic models, *in* 'Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 8724', ECML PKDD 2014, Springer-Verlag New York, Inc., New York, NY, USA, pp. 498–513.
URL: *http://dx.doi.org/10.1007/978-3-662-44848-9_32*

Griffiths, T. L. & Steyvers, M. (2004), 'Finding scientific topics', *Proceedings of the National Academy of Sciences* **101**(Suppl. 1), 5228–5235.

Hariri, N., Mobasher, B., Burke, R. & Zheng, Y. (2011), Context-aware recommendation based on review mining, *in* 'Proceedings of the 9th Workshop on Intelligent Techniques for Web Personalization and Recommender Systems (ITWP 2011)', p. 30.
URL: *http://ceur-ws.org/Vol-756/*

Hayes, C. & Cunningham, P. (2002), An on-line evaluation framework for recommender systems, Technical report, Trinity College Dublin, Department of

Computer Science.
URL: *http://www.tara.tcd.ie/handle/2262/13178*

Hofmann, T. (1999*a*), Probabilistic latent semantic analysis, *in* 'Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence', UAI'99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 289–296.
URL: *http://dl.acm.org/citation.cfm?id=2073796.2073829*

Hofmann, T. (1999*b*), Probabilistic latent semantic indexing, *in* 'Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval', SIGIR '99, ACM, New York, NY, USA, pp. 50–57.
URL: *http://doi.acm.org/10.1145/312624.312649*

Hofmann, T. (2001), 'Unsupervised learning by probabilistic latent semantic analysis', *Mach. Learn.* **42**(1/2), 177–196.
URL: *http://dx.doi.org/10.1023/A:1007617005950*

Hu, M. & Liu, B. (2004), Mining and summarizing customer reviews, *in* 'Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '04, ACM, New York, NY, USA, pp. 168–177.
URL: *http://doi.acm.org/10.1145/1014052.1014073*

Jakob, N., Weber, S. H., Müller, M. C. & Gurevych, I. (2009), Beyond the stars: Exploiting free-text user reviews to improve the accuracy of movie recommendations, *in* 'Proceedings of the 1st International CIKM Workshop on Topic-sentiment Analysis for Mass Opinion', TSA '09, ACM, New York, NY, USA, pp. 57–64.
URL: *http://doi.acm.org/10.1145/1651461.1651473*

Karatzoglou, A., Amatriain, X., Baltrunas, L. & Oliver, N. (2010), Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering, *in* 'Proceedings of the Fourth ACM Conference on Recommender Systems', RecSys '10, ACM, New York, NY, USA, pp. 79–86.
URL: *http://doi.acm.org/10.1145/1864708.1864727*

Karatzoglou, A., Baltrunas, L. & Shi, Y. (2013), Learning to rank for recommender systems, *in* 'Proceedings of the 7th ACM Conference on Recommender Systems', RecSys '13, ACM, New York, NY, USA, pp. 493–494.
URL: *http://doi.acm.org/10.1145/2507157.2508063*

Koren, Y. (2008), Factorization meets the neighborhood: A multifaceted collaborative filtering model, *in* 'Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '08, ACM, New York, NY, USA, pp. 426–434.
URL: *http://doi.acm.org/10.1145/1401890.1401944*

Koren, Y. (2009), Collaborative filtering with temporal dynamics, *in* 'Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '09, ACM, New York, NY, USA, pp. 447–456.
URL: *http://doi.acm.org/10.1145/1557019.1557072*

Koren, Y. & Bell, R. (2015), Advances in collaborative filtering, *in* F. Ricci, L. Rokach & B. Shapira, eds, 'Recommender Systems Handbook', Springer US, Boston, MA, pp. 77–118.
URL: *https://doi.org/10.1007/978-1-4899-7637-6_3*

Lam, X. N., Vu, T., Le, T. D. & Duong, A. D. (2008), Addressing cold-start problem in recommendation systems, *in* 'Proceedings of the 2Nd International Conference on Ubiquitous Information Management and Communication', ICUIMC '08, ACM, New York, NY, USA, pp. 208–211.
URL: *http://doi.acm.org.ucc.idm.oclc.org/10.1145/1352793.1352837*

Lathauwer, L. D., Moor, B. D. & Vandewalle, J. (2000), 'A multilinear singular value decomposition', *SIAM J. Matrix Anal. Appl.* **21**(4), 1253–1278.
URL: *https://doi.org/10.1137/S0895479896305696*

Lau, J. H., Newman, D. & Baldwin, T. (2014), Machine reading tea leaves: Automatically evaluating topic coherence and topic model quality, *in* 'Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics', pp. 530–539.

Laurikkala, J. (2001), Improving identification of difficult small classes by balancing class distribution, *in* S. Quaglini, P. Barahona & S. Andreassen, eds, 'Artificial Intelligence in Medicine', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 63–66.

Lee, D. D. & Seung, H. S. (1999), 'Learning the parts of objects by non-negative matrix factorization', *Nature* **401**, 788–791.
URL: *http://dx.doi.org/10.1038/44565*

Lee, M. D., Pincombe, B. & Welsh, M. (2005), An empirical evaluation of models of text document similarity, *in* 'Proceedings of the 27th Annual Conference of the Cognitive Science Society', Erlbaum, pp. 1254–1259.

Lee, T. Y., Smith, A., Seppi, K., Elmqvist, N., Boyd-Graber, J. & Findlater, L. (2017), 'The human touch: How non-expert users perceive, interpret, and fix topic models', *International Journal of Human-Computer Studies* **105**, 28 – 42.
URL: *http://www.sciencedirect.com/science/article/pii/S1071581917300472*

Lehman, L. W., Long, W., Saeed, M. & Mark, R. (2014), 'Latent topic discovery of clinical concepts from hospital discharge summaries of a heterogeneous patient cohort', *Conf Proc IEEE Eng Med Biol Soc* **2014**, 1773–1776.

Lehman, L. W., Saeed, M., Long, W., Lee, J. & Mark, R. (2012), 'Risk stratification of ICU patients using topic models inferred from unstructured progress notes', *AMIA Annu Symp Proc* **2012**, 505–511.

Levi, A., Mokryn, O., Diot, C. & Taft, N. (2012), Finding a needle in a haystack of reviews: Cold start context-based hotel recommender system, *in* 'Proceedings of the Sixth ACM Conference on Recommender Systems', RecSys '12, ACM, New York, NY, USA, pp. 115–122.
URL: *http://doi.acm.org/10.1145/2365952.2365977*

Lin, C. & He, Y. (2009), Joint sentiment/topic model for sentiment analysis, *in* 'Proceedings of the 18th ACM Conference on Information and Knowledge Management', CIKM '09, ACM, New York, NY, USA, pp. 375–384.
URL: *http://doi.acm.org/10.1145/1645953.1646003*

Ling, G., Lyu, M. R. & King, I. (2014), Ratings meet reviews, a combined approach to recommend, *in* 'Proceedings of the 8th ACM Conference on Recommender Systems', RecSys '14, ACM, New York, NY, USA, pp. 105–112.
URL: *http://doi.acm.org/10.1145/2645710.2645728*

Liu, N. N. & Yang, Q. (2008), Eigenrank: A ranking-oriented approach to collaborative filtering, *in* 'Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval', SIGIR '08, ACM, New York, NY, USA, pp. 83–90.
URL: *http://doi.acm.org/10.1145/1390334.1390351*

McAuley, J. & Leskovec, J. (2013), Hidden factors and hidden topics: Understanding rating dimensions with review text, *in* 'Proceedings of the 7th ACM Conference on Recommender Systems', RecSys '13, ACM, New York, NY, USA,

pp. 165–172.
URL: *http://doi.acm.org/10.1145/2507157.2507163*

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D. & Miller, K. (1990), 'Wordnet: An on-line lexical database', *International Journal of Lexicography* **3**, 235–244.

Mimno, D., Wallach, H. M., Talley, E., Leenders, M. & McCallum, A. (2011), Optimizing semantic coherence in topic models, *in* 'Proceedings of the Conference on Empirical Methods in Natural Language Processing', EMNLP '11, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 262–272.
URL: *http://dl.acm.org/citation.cfm?id=2145432.2145462*

Moghaddam, S. & Ester, M. (2010), Opinion digger: an unsupervised opinion miner from unstructured product reviews, *in* 'Proceedings of the 19th ACM international conference on Information and knowledge management', ACM, pp. 1825–1828.

Musto, C., de Gemmis, M., Semeraro, G. & Lops, P. (2017), A multi-criteria recommender system exploiting aspect-based sentiment analysis of users' reviews, *in* 'Proceedings of the Eleventh ACM Conference on Recommender Systems', RecSys '17, ACM, New York, NY, USA, pp. 321–325.
URL: *http://doi.acm.org/10.1145/3109859.3109905*

Musto, C., Semeraro, G. & Polignano, M. (2014), A comparison of lexicon-based approaches for sentiment analysis of microblog, *in* 'DART 2014: Information Filtering and Retrieval. CEUR-WS.org', Vol. 1314, pp. 59–68.

Newman, D., Lau, J. H., Grieser, K. & Baldwin, T. (2010), Automatic evaluation of topic coherence, *in* 'Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics', HLT '10, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 100–108.
URL: *http://dl.acm.org/citation.cfm?id=1857999.1858011*

Nguyen, T. V., Karatzoglou, A. & Baltrunas, L. (2014), Gaussian process factorization machines for context-aware recommendations, *in* 'Proceedings of the 37th International ACM SIGIR Conference on Research &#38; Development in Information Retrieval', SIGIR '14, ACM, New York, NY, USA, pp. 63–72.
URL: *http://doi.acm.org/10.1145/2600428.2609623*

Ning, X. & Karypis, G. (2011), Slim: Sparse linear methods for top-n recommender systems, *in* 'Proceedings of the 2011 IEEE 11th International Conference on Data Mining', ICDM '11, IEEE Computer Society, Washington, DC, USA, pp. 497–506.
URL: *http://dx.doi.org/10.1109/ICDM.2011.134*

Ning, X. & Karypis, G. (2012), Sparse linear methods with side information for top-n recommendations, *in* 'Proceedings of the Sixth ACM Conference on Recommender Systems', RecSys '12, ACM, New York, NY, USA, pp. 155–162.
URL: *http://doi.acm.org/10.1145/2365952.2365983*

O'Callaghan, D., Greene, D., Carthy, J. & Cunningham, P. (2015), 'An analysis of the coherence of descriptors in topic modeling', *Expert Systems with Applications* **42**(13), 5645 – 5657.
URL: *http://www.sciencedirect.com/science/article/pii/S0957417415001633*

Pagano, R., Cremonesi, P., Larson, M., Hidasi, B., Tikk, D., Karatzoglou, A. & Quadrana, M. (2016), The contextual turn: From context-aware to context-driven recommender systems, *in* 'Proceedings of the 10th ACM Conference on Recommender Systems', RecSys '16, ACM, New York, NY, USA, pp. 249–252.
URL: *http://doi.acm.org/10.1145/2959100.2959136*

Park, S.-T. & Chu, W. (2009), Pairwise preference regression for cold-start recommendation, *in* 'Proceedings of the Third ACM Conference on Recommender Systems', RecSys '09, ACM, New York, NY, USA, pp. 21–28.
URL: *http://doi.acm.org.ucc.idm.oclc.org/10.1145/1639714.1639720*

Pauca, V. P., Shahnaz, F., Berry, M. W. & Plemmons, R. J. (2004), Text mining using non-negative matrix factorizations, *in* 'Proceedings of the 2004 SIAM International Conference on Data Mining', pp. 452–456.
URL: *http://epubs.siam.org/doi/abs/10.1137/1.9781611972740.45*

Peña, F. J. (2017), Unsupervised context-driven recommendations based on user reviews, *in* 'Proceedings of the Eleventh ACM Conference on Recommender Systems', RecSys '17, ACM, New York, NY, USA, pp. 426–430.
URL: *http://doi.acm.org/10.1145/3109859.3109865*

Peña, F. J. & Bridge, D. (2017), Recommending from experience.
URL: *https://aaai.org/ocs/index.php/FLAIRS/FLAIRS17/paper/view/15439*

Pincombe, B. (2004), Comparison of human and latent semantic analysis (lsa) judgements of pairwise document similarities for a news corpus, Technical report, Defense Science and Technology Organization Salisbury (Australia).

Rendle, S. (2010), Factorization machines, *in* '2010 IEEE International Conference on Data Mining', pp. 995–1000.

Rendle, S. (2012), 'Factorization machines with libfm', *ACM Trans. Intell. Syst. Technol.* **3**(3), 57:1–57:22.
URL: *http://doi.acm.org/10.1145/2168752.2168771*

Rossetti, M., Stella, F. & Zanker, M. (2013), Towards explaining latent factors with topic models in collaborative recommender systems, *in* '2013 24th International Workshop on Database and Expert Systems Applications', pp. 162–167.

Said, A. & Bellogín, A. (2014), Comparative recommender system evaluation: Benchmarking recommendation frameworks, *in* 'Proceedings of the 8th ACM Conference on Recommender Systems', RecSys '14, ACM, New York, NY, USA, pp. 129–136.
URL: *http://doi.acm.org/10.1145/2645710.2645746*

Salakhutdinov, R. & Mnih, A. (2008), Bayesian probabilistic matrix factorization using markov chain monte carlo, *in* 'Proceedings of the 25th International Conference on Machine Learning', ICML '08, ACM, New York, NY, USA, pp. 880–887.
URL: *http://doi.acm.org/10.1145/1390156.1390267*

Salton, G. & Buckley, C. (1988), 'Term-weighting approaches in automatic text retrieval', *Information Processing & Management* **24**(5), 513 – 523.
URL: *http://www.sciencedirect.com/science/article/pii/0306457388900210*

Shi, Y., Larson, M. & 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.

Steck, H. (2013), Evaluation of recommendations: Rating-prediction and ranking, *in* 'Proceedings of the 7th ACM Conference on Recommender Systems',

RecSys ’13, ACM, New York, NY, USA, pp. 213–220.
URL: *http://doi.acm.org/10.1145/2507157.2507160*

Steyvers, M. & Griffiths, T. (2007), Probabilistic Topic Models, *in* T. Landauer, D. Mcnamara, S. Dennis & W. Kintsch, eds, ‘Handbook of Latent Semantic Analysis’, Lawrence Erlbaum Associates.
URL: *http://www.worldcat.org/isbn/1410615340*

Tomek, I. (1976), ‘Two modifications of cnn’, *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-6**(11), 769–772.

Tomokiyo, T. & Hurst, M. (2003), A language model approach to keyphrase extraction, *in* ‘Proceedings of the ACL 2003 Workshop on Multiword Expressions: Analysis, Acquisition and Treatment - Volume 18’, MWE ’03, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 33–40.
URL: *https://doi.org/10.3115/1119282.1119287*

Unger, M., Bar, A., Shapira, B. & Rokach, L. (2016), ‘Towards latent context-aware recommendation systems’, *Knowledge-Based Systems* **104**(Supplement C), 165 – 178.
URL: *http://www.sciencedirect.com/science/article/pii/S0950705116300727*

Wang, C. & Blei, D. M. (2011), Collaborative topic modeling for recommending scientific articles, *in* ‘Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining’, KDD ’11, ACM, New York, NY, USA, pp. 448–456.
URL: *http://doi.acm.org/10.1145/2020408.2020480*

Wang, Y., Liu, Y. & Yu, X. (2012), Collaborative filtering with aspect-based opinion mining: A tensor factorization approach, *in* ‘2012 IEEE 12th International Conference on Data Mining’, pp. 1152–1157.

Wilson, D. R. & Martinez, T. R. (2000), ‘Reduction techniques for instance-basedlearning algorithms’, *Mach. Learn.* **38**(3), 257–286.
URL: *https://doi.org/10.1023/A:1007626913721*

Wu, Y. & Ester, M. (2015), Flame: A probabilistic model combining aspect based opinion mining and collaborative filtering, *in* ‘Proceedings of the Eighth ACM International Conference on Web Search and Data Mining’, WSDM ’15, ACM, New York, NY, USA, pp. 199–208.
URL: *http://doi.acm.org/10.1145/2684822.2685291*

Xiong, L., Chen, X., Huang, T. K., Schneider, J. & Carbonell, J. G. (2010), Temporal collaborative filtering with bayesian probabilistic tensor factorization, *in* 'SIAM Data Mining 2010 (SDM 10)'.

Xu, W., Liu, X. & Gong, Y. (2003), Document clustering based on non-negative matrix factorization, *in* 'Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval', SIGIR '03, ACM, New York, NY, USA, pp. 267–273.
URL: *http://doi.acm.org/10.1145/860435.860485*

Yang, Y. & Pedersen, J. O. (1997), A comparative study on feature selection in text categorization, *in* 'Proceedings of the Fourteenth International Conference on Machine Learning', ICML '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 412–420.
URL: *http://dl.acm.org/citation.cfm?id=645526.657137*

Zalewski, A., Long, W., Johnson, A. E. W., Mark, R. G. & Lehman, L. H. (2017), 'Estimating Patient's Health State Using Latent Structure Inferred from Clinical Time Series and Text', *IEEE EMBS Int Conf Biomed Health Inform* **2017**, 449–452.

Zheng, Y., Burke, R. & Mobasher, B. (2012*a*), Differential context relaxation for context-aware travel recommendation, *in* C. Huemer & P. Lops, eds, 'E-Commerce and Web Technologies: 13th International Conference, EC-Web 2012, Vienna, Austria, September 4-5, 2012. Proceedings', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 88–99.
URL: *https://doi.org/10.1007/978-3-642-32273-0_8*

Zheng, Y., Burke, R. & Mobasher, B. (2012*b*), Optimal feature selection for context-aware recommendation using differential relaxation, *in* 'In ACM RecSys' 12, Proceedings of the 4th International Workshop on Context-Aware Recommender Systems (CARS 2012). ACM'.

Zheng, Y., Burke, R. & Mobasher, B. (2013), Recommendation with differential context weighting, *in* S. Carberry, S. Weibelzahl, A. Micarelli & G. Semeraro, eds, 'User Modeling, Adaptation, and Personalization', Vol. 7899 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 152–164.
URL: *http://dx.doi.org/10.1007/978-3-642-38844-6_13*

Zheng, Y., Burke, R. & Mobasher, B. (2014), Splitting approaches for context-aware recommendation: An empirical study, *in* 'Proceedings of the 29th An-

nual ACM Symposium on Applied Computing', SAC '14, ACM, New York, NY, USA, pp. 274–279.
URL: *http://doi.acm.org/10.1145/2554850.2554989*

Zheng, Y., Mobasher, B. & Burke, R. (2014), Cslim: Contextual slim recommendation algorithms, *in* 'Proceedings of the 8th ACM Conference on Recommender Systems', RecSys '14, ACM, New York, NY, USA, pp. 301–304.
URL: *http://doi.acm.org/10.1145/2645710.2645756*