AALBORG UNIVERSITET

mi107f19:

Attribute-enhanced Collaborative Topic Modelling

AALBORG UNIVERSITY
STUDENT REPORT

Software
Aalborg University
http://www.aau.dk

Title:
Attribute-enhanced Collaborative Topic Modelling

Theme:
Master Thesis

Project Period:
Spring Semester 2019

Project Group:
mi107f19

Participant(s):
Victor Holberg Haugan

Supervisor(s):
Thomas D. Nielsen

Copies: 0

Number of Pages: 87

Date of Completion:
June 14, 2019

Abstract:

This project explores possible solutions for the item cold-start problem for recommender systems. The project uses a dataset collected from the Japanese internship recruitment website 01Intern. The model CoAWILDA+ is proposed based on the idea of combining different types of item attributes with the advantages that collaborative filtering offers. CoAWILDA+ combines topic modelling of text documents using *Adaptive Window Incremental Latent Dirichlet Allocation* with the Attribute-enhancement model, which is a simple modification of matrix factorisation to include info for item attributes. The topic modelling component AWILDA is evaluated separately

Preface

Reading guide

Source reference. This report uses the Vancouver-referring system [1] for citation. The number in square brackets is the source number. This number is simultaneously a hyperlink to bibliography.

Figure reference. This report uses figures and illustrations which are referred to with a chapter number and a figure number separated by a period, e.g. 0.1, where 0 is the chapter and 1 is index of the figure. An example of this can be seen in Figure 1.

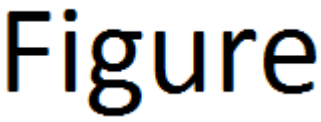


Figure 1: Sample figure

Table reference. This report uses tables and these are referred to in a similar manner to figures. An example can be seen in Table 1.

Table Item Class 1	Table Item Class 2
Item A	Item X
Item B	Item Y
Item C	Item Z

Table 1: Sample Table

Code listing. This report uses code samples to highlight code for explanations. An example can be seen in Listing 1.

```
1  public class HelloWorld {
2
3      public static void main(String[] args) {
4          // Prints "Hello, World" to the standard output.
5          System.out.println("Hello, World");
6      }
7  }
```

Listing 1: Java Code Snippet

Acronyms

This is a list of acronyms used throughout the report. Some of these acronyms may be known to the reader, but they are included here anyway as a convenience for readers unfamiliar with them.

AAU Aalborg University

ADWIN Adaptive Windowing

AE Attribute-Enhancement

ALS Alternating Least Squares

AWILDA Adaptive Windowing based Incremental Latent Dirichlet Allocation

BPR Bayesian Personalised Ranking

DCG Discounted Cumulative Gain

ELBO Evidence Lower Bound

FVA Factor Vector Alignment

IMDb Internet Movie Database

ISGD Incremental Stochastic Gradient Descent

KL Kullback-Leibler

LDA Latent Dirichlet Allocation

LSI Latent Semantic Indexing

MF Matrix Factorization

ML-100K MovieLens 100K

NLP Natural Language Processing

SGD Stochastic Gradient Descent

SVD Singular Value Decomposition

Contents

Part I

Introduction

1. Introduction

Today many web-based platforms face the problem of information overload: the abundance of products and documents makes it difficult for users to find the items that are relevant to them by traditional means. Sorting through everything becomes nearly impossible - which is why numerous methods are being developed that help overcome this problem. Information retrieval systems are designed to retrieve documents that are highly relevant to the needs of a user, while information filtering systems leverage various information to filter out information and improve the quality of displayed items. Among information filtering systems are recommender systems, which provide users of a service recommendations for items they might be interested in, often presented as a list ranked by relevance. Recommender systems are almost ubiquitous today, some providing movie recommendations on your favourite movie streaming platforms and others suggesting that you try out a product that seems to your taste. Recommender systems are a widely studied area of information filtering, with a lot of research being stimulated by the Netflix Prize competition which began in 2006.

01Intern is a Japanese internship recruitment website. While the service has plenty of functionality that allows users to formulate queries and search for relevant jobs, it offers little in terms of providing users with personalised recommendations. During an internship at the company that offers 01Intern, Salt Inc., the author had the good fortune of working with a dataset of activity on 01Intern, with the purpose of developing a recommender system for the website. While a simple recommender system was proposed, the problem cannot yet be considered solved, as many aspects of the dataset remain unexplored.

1.1 Recommender systems

The role of recommender systems is to infer the rating of a user on an item that the user has not seen. “Item” is the term commonly used to describe the entities recommended to users, be it movies or news articles[2]. When recommender systems provide users with recommendations, it is usually in the form of ranked lists of items a user is likely to be interested in[2].

Most recommender systems use *explicit feedback*, *implicit feedback*, or a combination of both[3], from users as a basis for predictions. Explicit feedback refers to feedback from users in which their preference for an item is explicitly stated. For example, the user might rate a movie on a scale from 1 to 5, or indicate whether liked an item by pressing a thumbs-up or thumbs-down button[3]. This may also be in the form of a questionnaire in which a user makes clear their preferences. Implicit feedback, on the other hand, refers to feedback for which the user preference is not explicitly stated. Examples of this include clicking an article, playing a video or scrolling the description page of a product on an e-commerce website. While implicit feedback is often considered positive only, methods exist for inferring when feedback is negative and weighting different feedback types[4].

Recommender systems can take different approaches for making predictions about user ratings. The most notable categories of recommender systems are those that use *content-based filtering* or *collaborative filtering*, and *hybrid recommender systems*, which combine several approaches to improve performance. A brief introduction to each of these categories is provided in the next sections.

1.1.1 Content-based filtering

Content-based filtering is an approach in which high ratings are predicted for items similar to those the user has interacted with in the past. Item similarity may for example be computed using different item attributes[2]. A basic content-based filtering recommender system may use a *profile learner* to generate a profile for a user, which is then compared to unseen items using a *filtering component*[2]. The profile learner can make use of the past interactions of users to infer which properties of items the user finds interesting.

When comparing different attributes of items, it is important to note that often users do not consider all attributes equally important. In this scenario, it may be beneficial to assign different weights to different properties[5].

Content-based filtering has several advantages: first, it is able to recommend items, which any user has yet to interact with. Secondly, the basis for recommendation is very transparent: profiles of users are compared with item attributes, making the reason for any given recommendation easy to understand. However, this is also one of the drawbacks of this approach. As the recommendations are all based on the past interactions of a user, there is no surprise in the recommendations, a problem known as the *serendipity problem*[2]. Furthermore, a profile of users is the foundation for recommendations, meaning for a new user ratings cannot be accurately predicted.

1.1.2 Collaborative filtering

Collaborative filtering "analyzes relationships between users and interdependencies among products to identify new user-item associations"[3, p. 43]. Contrary to content-based filtering, collaborative filtering approaches do not consider the attributes of items and users, and instead predict ratings through patterns in the feedback users provide for items. One popular collaborative-filtering method is Matrix Factorization (MF). A simple approach, MF describes users and items as vectors of latent factors inferred using feedback. Items whose latent factors align best with the latent factors of a users are those recommended to a user [3]. In Section 4.3, MF is described in detail.

While collaborative filtering recommenders are able to provide serendipitous recommendations, a drawback of collaborative filtering algorithms is the cold-start problem. The cold-start problem is when no ratings exists for a user or an item, making it difficult to provide accurate recommendations. We make a distinction between the user cold-start problem, where a user has not yet rated any items, and the item cold-start problem, where an item has yet to be rated by any user. Unless explicitly stated, in this report cold start or cold-start problem refers to the item cold-start problem.

1.1.3 Hybrid recommender systems

Hybrid recommender systems combine several approaches to improve recommendation quality. Some recommender systems extend collaborative filtering models with item information to overcome the problem of item cold start. Numerous works suggest the usefulness of alleviating the cold-start problem using item information for different datasets[6][7]. Hybrid recommender systems may also leverage user information, such as demographics,

to overcome the user cold-start problem[3][8].

Recommender systems have been developed that combine different sources of information[3]. In [9], a recommender system is proposed to deal with the item cold-start problem and the user cold-start problem. The recommender system is shown to outperform baselines when providing recommendations for new users, recommending new items, and recommending new items to new users.

1.2 Evaluating recommender systems

A number of different approaches exist for evaluating recommender systems. Traditionally recommender systems are evaluated using holdout methods, in which a subset of the set of known ratings is hidden, and the model is trained on the remaining ratings. The model then scores based on its ability to predict the items that were hidden [10].

In most real world applications, however, recommender systems are required to handle data that is continuously generated in order to provide recommendations[7]. Holdout methods are not well suited for evaluating such online recommender systems. In online recommendation setting, it is also important that models are able to process documents at the rate they appear from the data stream. This means that completely retraining a model at each step is infeasible for platforms with much activity. The *online setting* is a less studied aspect of recommender systems. It is, however, relevant to the 01Intern dataset, as the dataset been collected from the 01Intern website, which exhibits the property of continuously generating data. Different measures for the evaluation of recommendation in an online setting is described in [11].

1.3 Data streams and concept drift

The task of providing recommendations in an online setting is a type of data stream mining. An important concept in data stream mining is the notion of *concept drift*, which means that the distribution of the modelled data changes over time. The strategies for adapting to concept drift can be split into the categories of *passive* methods and *active* methods. Passive methods are those that retrain the method each time an observation is received, which may lead to signi□cant overhead for large datasets. Active methods, on the other hand, rely on a drift detection component to detect when a concept drift has occurred. Only when a drift is detected is the model retrained.

1.4 Initial problem

During an earlier semester, the author developed a recommender system for the 01Intern dataset. The developed recommender system took a simple content-based approach to rank items. Since collaborative □ltering models generally outperform content-based □ltering models[3], taking this aspect of the data into account is likely to lead to improved performance. Furthermore, the previously developed model was created for a setting where all ratings are known

from the start - this does not reflect the online nature of the 01Intern dataset. The initial problem thus is to develop a suitable recommender system for the 01Intern dataset, that better realises the potential of the dataset. This recommender system should leverage the relevant aspects of the dataset and be applicable to an online setting. To get a more general idea of the performance of the developed recommender system, we apply it to the widely used MovieLens 100K (ML-100K) dataset.

In Part II, the 01Intern and ML-100K datasets are analysed in order to facilitate a better understanding of the characteristics of the datasets. Through this understanding the subject is delimited with the purpose of being able to clearly define the problem that characterises the 01Intern dataset in the context of item recommendation.

Part II

Analysis

2. Datasets

In this section we will describe the datasets used in the evaluation of the proposed recommender system. The two datasets are 01Intern and ML-100K. With regards to the 01Intern dataset, which has previously been described in an earlier work[12], we focus on describing the quantitative properties of the dataset. We summarise the attributes for jobs, corporations and users in Section 2.1.4. Meanwhile, we describe both the qualitative and quantitative aspects of the ML-100K dataset in detail.

2.1 01Intern

The 01Intern dataset consists of 1,878 jobs, 30,722 users and the 213,612 unique interactions between them, collected from the Japanese internship recruitment website 01Intern. In truth, there are approximately 40,000 users and 2,000 jobs, however, the ones for which no interactions exists have been filtered out. The user-interaction matrix[1] of the 01Intern dataset is very sparse, with a density of $\frac{213{,}612}{30{,}722 \cdot 1{,}878} = 0.00037\%$.

2.1.1 Interactions

In the 01Intern dataset, interactions are received through three distinct channels. Jobs are offered by corporations, and users may view jobs, add jobs to their favourites and apply for jobs. Jobs, corporations and users each have a number of unstructured and structured attributes associated with them (e.g. job description, salary, corporation address or student grade). In a previous work[12], the attributes of jobs, corporations and users have been described in detail. In this work the focus is on the qualitative properties of the 01Intern dataset.

When describing the data we generalise views, favourites and applications as interactions, but we will also investigate the distribution of the different types of interactions. Figure 2.1 shows the number of jobs that have more than a certain number of interactions. For example, consider the bin labelled 150. This bin tells us that just over 500 jobs have at least 150 interactions. Figure 2.2 is a histogram of interactions by job. The figures show that the dataset holds characteristics that can result in item cold-start problems for recommender systems, as a significant number of jobs have been interacted with only few times: approximately 25 jobs lie in the bin for 0-9 interactions and more than 60 jobs in the bin for 10-19 interactions. Generally, the fewer ratings an item has, the more difficult it is to recommend that item using collaborative filtering methods, as the data does not provide a good basis for recommendation[6]. In Figure 2.1 the number of jobs above the threshold drops quickly before 100 interactions - approximately 100 less jobs with each bin - and then starts to decay. The majority of jobs, specifically 955 jobs, thus have less than 100 interactions (and many much fewer). In Figure 2.2 we see that nearly 100 jobs have less than 20 interactions. In settings where many items have only few ratings, leveraging content-based information has been shown to improve performance [6][9].

[1] A matrix where each row represents a user, each column represents and item, and each entry has a value of 1 if the user has interacted with the item in the past, and 0 otherwise.

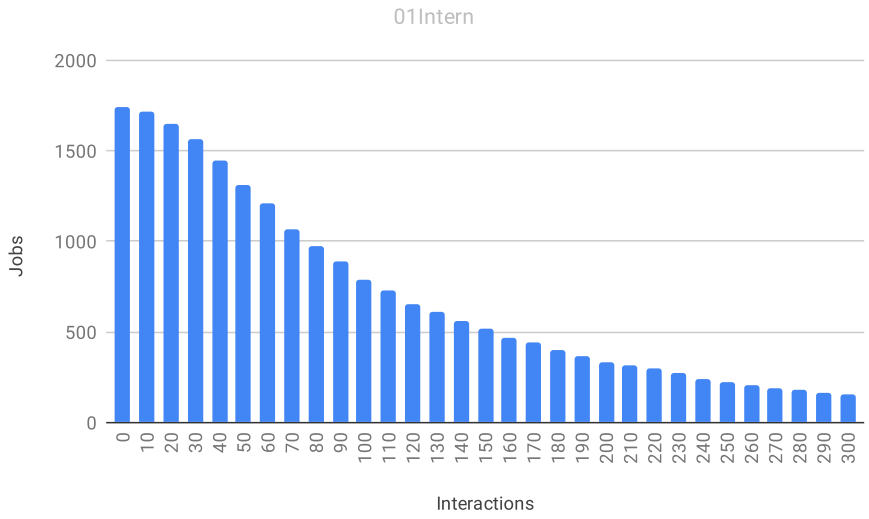


Figure 2.1: Column chart of frequency of jobs that have at least a certain number of interactions

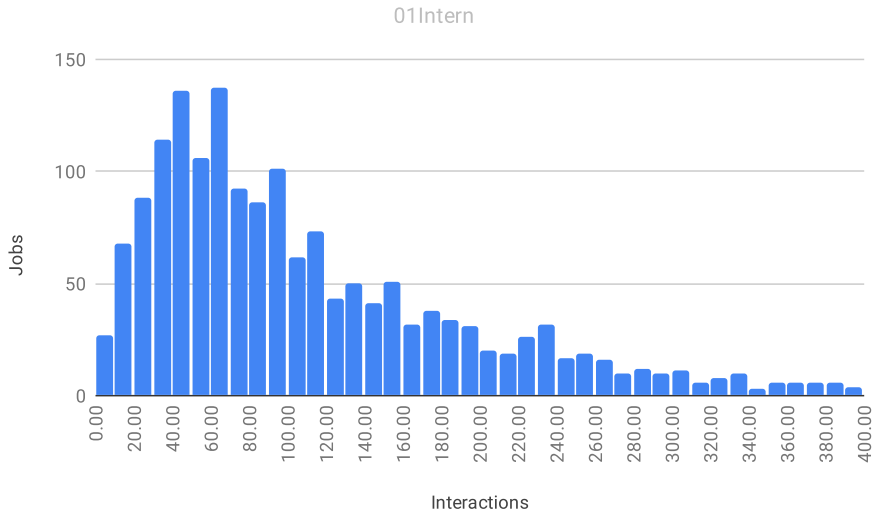


Figure 2.2: Histogram of interactions by job

In Figure 2.3 we see the number of users that have above a certain number of interactions. There is a large decrease in users every time we move up a bin: approximately 9,000 as we move from the 0 bin to the 1 bin, then 7,000 as we move to the next bin and 4,000 for the following. By three interactions, less

than half the users remain. We also notice how approximately 9,000 users have no interactions with jobs at all. The interaction bins continue far beyond the cutoff point. Hence, we can conclude that a small number of users are responsible for a signi□cant number of the ratings. However, the many users with few interactions show that the user cold-start problem may indeed also be an issue for recommender systems for the dataset, as there are not enough ratings to infer the preferences for the users. Usually, the user cold-start problem is alleviated by leveraging additional information[8][13].

In fact, if we consider only the users that have had 20 or more interactions with jobs, we are left with only 3,087 users, but 82,681 of the total 213,612 unique ratings remain. 7 of the jobs have been □ltered out, leaving 1871 jobs. If we compute the density of the user-interaction matrix with the majority of users □ltered out, the result is $\frac{82;681}{3;087\ 1;871} = 1{:}4315\%$. This is an increase of density by several orders of magnitude. Figure 2.4 shows the interactions by user, similarly to Figure 2.3, but with users with less than 20 ratings cut off.

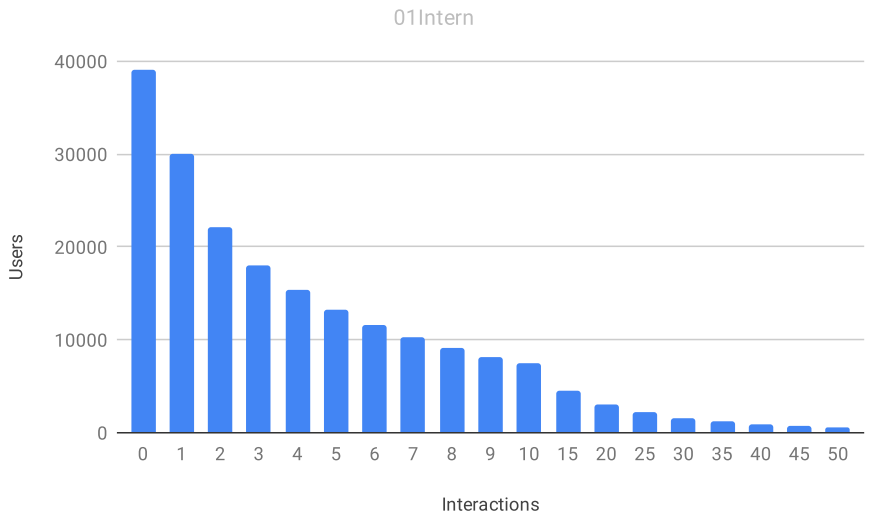


Figure 2.3: Column chart of frequency of users that have at least a certain number of interactions

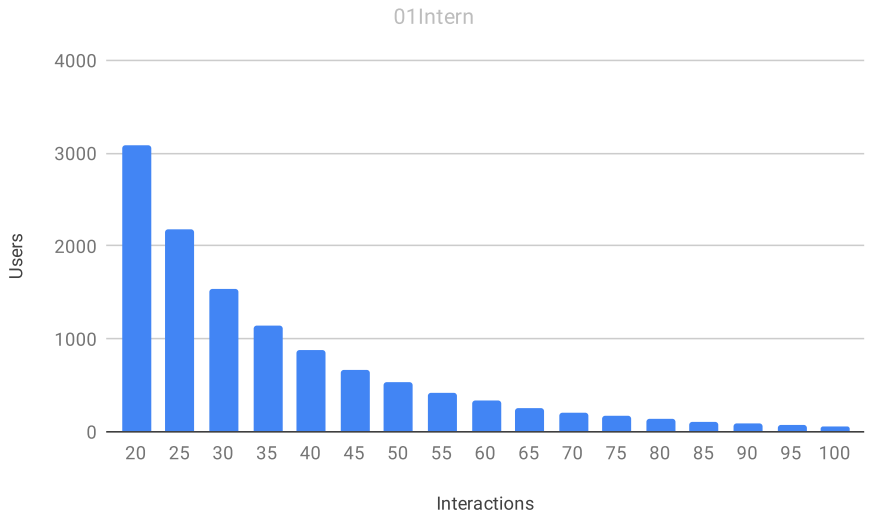


Figure 2.4: Column chart of frequency of users that have at least a certain number of interactions, zoomed in on the 20-100 range

2.1.2 Interaction channel distribution

As mentioned previously, interactions come from three distinct channels: views, favourites and applications. Thus far, we have considered these as a single type of feedback, namely interactions. In Figure 2.5 we see the interaction type distribution across the three channels. Like previously, each bin contains the distribution of interaction types for users with more than a certain number of interactions. Similarly, in Figure 2.6 we see the interaction type distribution but with the aforementioned cut-off. Something that immediately draws attention is how users can add more items to favourites than they view, as one would expect users to view an item before making the decision to add it to their favourites. This is, however, explained by the fact that the view tracking channel was added much later than the other two channels[12].

In the two □gures we see that for users with lower numbers of interactions, favourites is the more common interaction channel. As the number of interactions increases, the distribution shifts towards one dominated by the views channel. One possible explanation for this, is that many users browse the website anonymously and only register when they want to apply for a job or add it to their favourites. When users browse the website anonymously, their view actions are not tied to a user account and thus the views are not registered. For these users, the views make up less of the total interactions than for users that browse the site while logged in. Recurring users of the service are more likely to browse jobs while logged in. The percentage of interactions that come from the applications channel remains more or less the same over time.

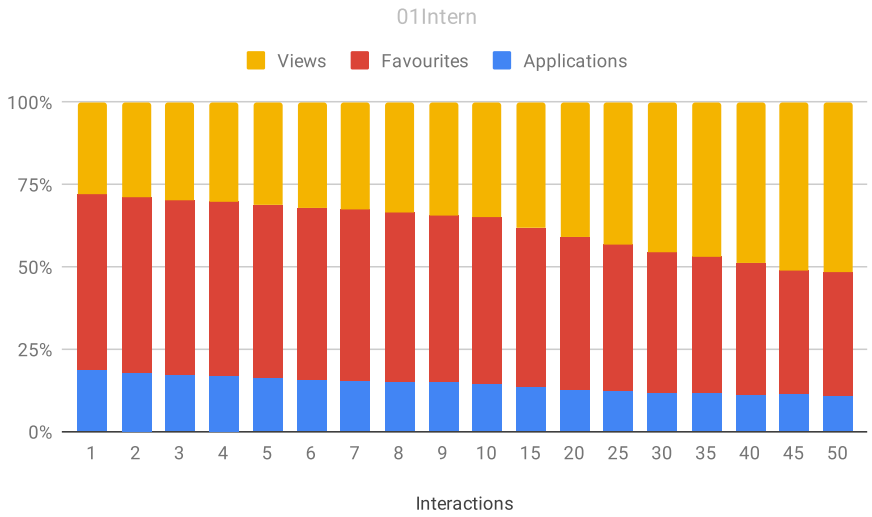


Figure 2.5: The distribution of interaction types for users with at least a certain number of interactions

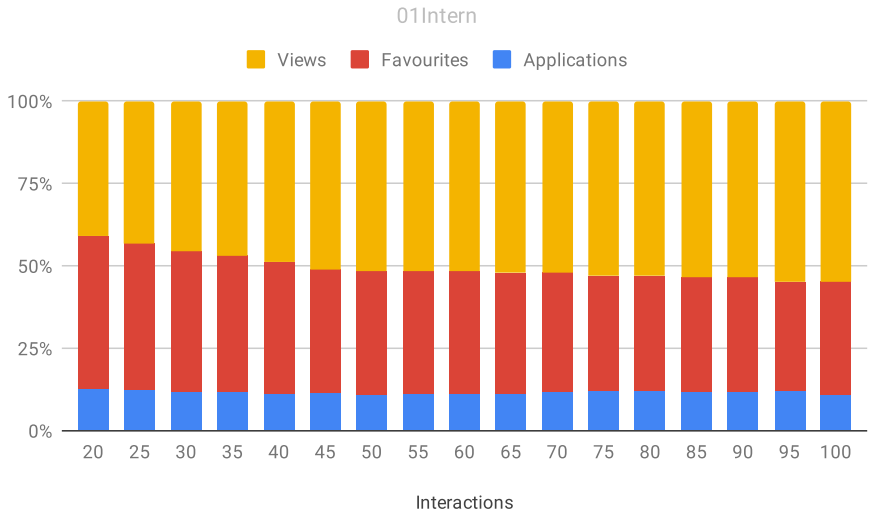


Figure 2.6: The distribution of interaction types for users after the cutoff

2.1.3 Temporality of jobs

Jobs from the 01Intern dataset are characterised by being highly temporal. Jobs come and go, they are published and become inactive again after they

are taken. Some jobs are published only once in their lifetime, whereas others are published again and again as more openings become available. Generally, jobs initially see a burst of interactions, but popularity soon decreases when they become occupied.

The temporality of jobs is a complicated problem due to the lack of history of jobs. While we can see whether a job was available or unavailable at the time the dataset was exported, we do not know how the availability of jobs has changed over time. We do, however, know when a job was first created. Consequently, when evaluating a recommender system based on the 01Intern dataset, we have to reason about jobs that may not have been available to users at that time. To give a more concrete example, consider a job that is created and published in January 2018. In March, the position becomes occupied and the job can no longer be applied for. In October the job again is opened for applications and at the end of November the dataset is exported. In the dataset, the only information available about the job is that it was published in January 2018 and open for applications at the end of November. The history of the job is not tracked and there is no information that the job was unavailable from March to October (other than what can be inferred from the set of applications). Thus when tasked with predicting which jobs a user will apply for in July, it is assumed that the job was open for applications at this time. However, in the case of predicting whether a user will interact with the target, this is less of a problem, as the job can still be viewed and added to favourites.

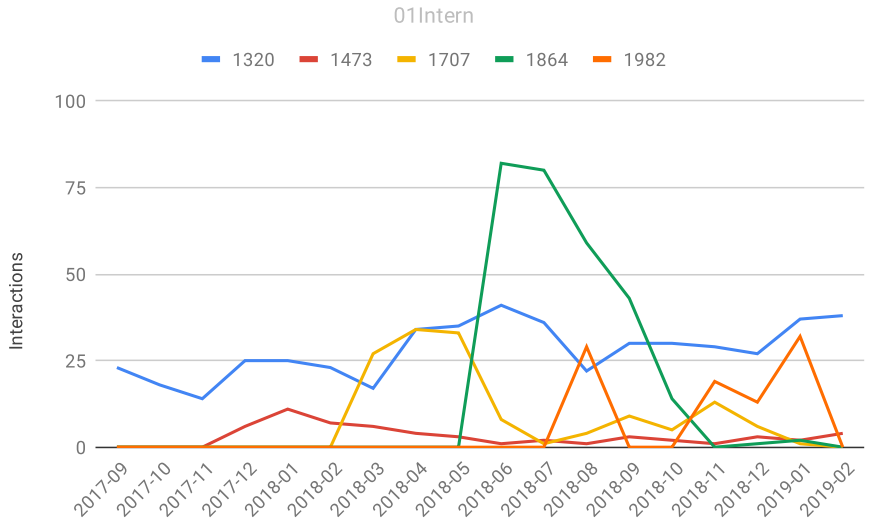


Figure 2.7: Number of interactions over time

Figure 2.7 shows the number of interactions over jobs received each month over an 18 month period. The jobs have been selected to be representative of several categories of job histories at different degrees of popularity.

Three basic categories can be established:

- **singly-published**: as mentioned previously, jobs generally are popular when they are first published. This popularity then decreases over time. Jobs 1473, 1707, and 1864 show this tendency. While the degree of popularity varies, they all show high popularity in their first months and then relatively few interactions afterwards.
- **multiply-published**: Job 1982 is an example of a job that has been republished. In August, 2018 the job is published but the position gets taken soon after. Later, in November the job is published again and once more receives a lot of interactions.
- **long-term**: Job 1320, which receives a relatively stable number of ratings each month, has remained published over the entire 18 month period.

The fact that many jobs receive applications and then die out further emphasises the importance of a content-based filtering component to alleviate cold start. Once a job has a sufficient number of ratings for collaborative filtering to be effective, they usually soon become occupied and thus less relevant.

2.1.4 Job, corporation, and user attributes

In this section the attributes of jobs, corporations, and users of the 01Intern dataset are described. As 01Intern is a website targeting a Japanese audience, the content and item attributes are all in Japanese. The tables containing a description of each attribute are from the author's earlier work [12]. Some attributes are omitted as they are considered too obscure to be meaningful, e.g. the images displayed to users when viewing a job.

In Table 2.1, the attributes of jobs are described. For each attribute a brief description is provided along with the type of the attribute. Generally, the attributes are either unstructured text attributes, categorical attributes or sets of properties. The categorical attributes can also be viewed as sets of exactly one property. It is also clear from the data description that not all attributes contribute equally to the description of a job. For example, the *job type* or *entrusted job detail* provide more insight into contents of a job than the *required entry comment* attribute.

In Table 2.2, a description of the attributes of corporations is provided. As in the case of jobs, this set of attributes primarily consists of text attributes and sets of properties. Again, not all attributes are equally descriptive of the content. As each job is associated with a corporation, the attributes for corporations can be seen as an extension to the job attributes.

In addition to the attributes for jobs and corporation, the dataset also contains demographic information for users. In Table 2.3, the attributes of users are described. The attributes primarily provide a description of the educational background of the user. Such a description may prove useful in determining what jobs the user is interested in. The majority of the attributes are in a text format.

2.2 MovieLens 100K

The ML-100K dataset, discussed in [14], is a commonly used machine-learning dataset, consisting of 100,000 movie ratings by 943 users. Other versions of

this dataset exists with the number of interactions ranging from 1-20 million. The ratings have been collected through the MovieLens website over the course of seven months. 1,682 movies have been rated in the dataset, with each rating expressing a user's preference for a movie. Users with less than 20 ratings have been filtered out. In addition to the ratings, the dataset also contains information about the movies and demographic information about the users. This dataset is similar to the 01Intern dataset as there are unstructured text attributes and structured attributes available for the items, as well as demographic information for the users. While some of this information is not available in the dataset itself, in Section 2.2.2 a description of how to collect the information is presented.

2.2.1 Ratings

As opposed the 01Intern dataset, feedback in the ML-100K dataset is received from a single channel: ratings. Ratings in the dataset are tuples consisting of a user ID, a movie ID, a 1-5 star rating and a timestamp. The stars express the user's preference for a movie with 1 indicating the lowest possible preference and 5 the highest.

The density of the ML-100K dataset is $\frac{100;000}{943\ 1;682}$ = 6:3046%, meaning that there is no rating for 93.6954% of the entries of the ratings matrix. This means that the dataset is significantly denser than other datasets[15][16]. There is a varying number of ratings for each movie, ranging from as low as a single rating to 583 ratings. This means that for some items, cold start may pose a problem for recommender systems.

Figure 2.8 shows the number of movies that have above a certain number of ratings. One thing that draws attention is how approximately two thirds of the movies are shown to have at least 10 ratings. Conversely, a third of the movies have less than 10 ratings. Item cold start can be a problem for recommender systems modelling these items. For the first bin, the number of movies with more than 0 ratings is 1,682, the total number of movies. For the next bin, the number of movies is 1,349, a reduction 333. To the next bin there is a decrease in jobs by 197. The decrease in jobs per bin gradually slows down as we move up the bins.

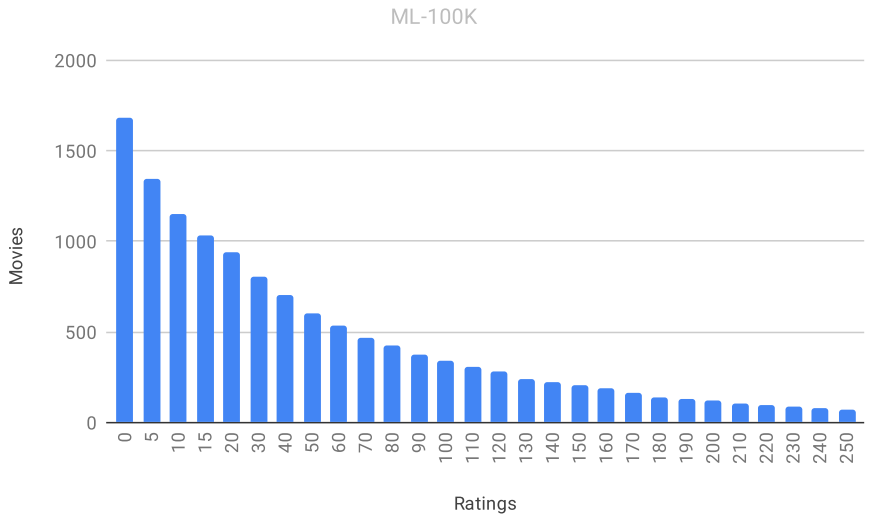


Figure 2.8: Column chart of frequency of movies that have at least a certain number of ratings

Figure 2.9 shows the number of ratings by user. As mentioned previously, none of the 943 users have less than 20 ratings. This means that user cold start is not likely to pose a problem for the ML-100K dataset. On the other hand, there is great variation in the number of ratings per user. The majority of users have less than 70 ratings, while about 5% of users have more than 300 ratings.

Furthermore, note that the distribution of stars given to movies is not linear. Figure 2.10 shows a histogram of the stars by rating. As seen in the figure, movies receive much fewer 1-star ratings than any other rating category. 5-star ratings are given approximately twice as often as 1-star ratings, but are still nowhere near as common as ratings of 2-4 stars. Users are more likely to rate movies positively (4-5 stars) than negatively (1-2 stars), with a ratio of 1.3695:1.

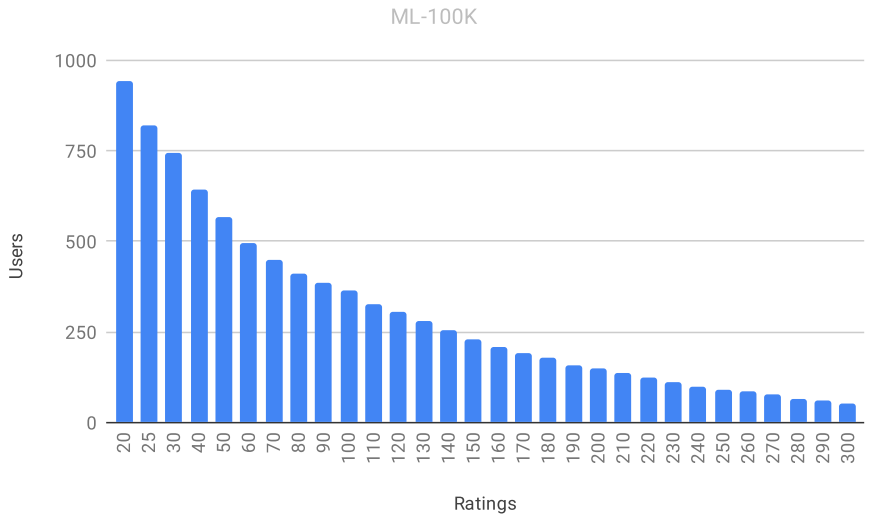


Figure 2.9: Column chart of frequency of users that have at least a certain number of ratings

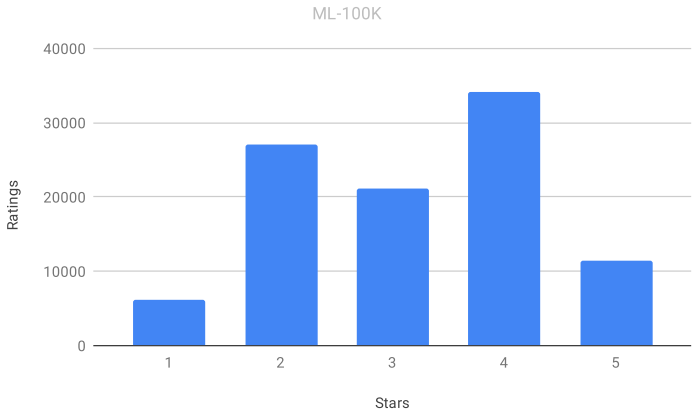


Figure 2.10: Histogram of stars by rating

We finally consider the temporality of movies in the dataset. The ratings were collected over a seven month period from 1997-09-19 through 1998-04-22[17]. Of the 1682 movies, 1603 had been released before collection of the ratings commenced. However, 273 movies were released in 1997 before collection started, some of which were definitely still being shown in theatres.

Figure 2.11 shows the ratings received by five arbitrarily picked movies per month over the seven month collection period. The movies were all released prior to the start of collection. With the exception of “The Truth About Cats &

Dogs" (movie 111), there does not seem to be a strong correlation between time and the amount of ratings given. Rather, the number of ratings remains stable over time. Whereas movie 111 may indicate that recently released movies may be subject to a burst of popularity, the vast majority of the dataset consists of older movies to which this does not apply.

However, it should be noted that there is a signi□cant difference in the popularity of a more recently released movie such as movie 111 and the other movies, which have been released earlier. Note also that this analysis does not take into account how the popularity of the MovieLens website may have changed over time.

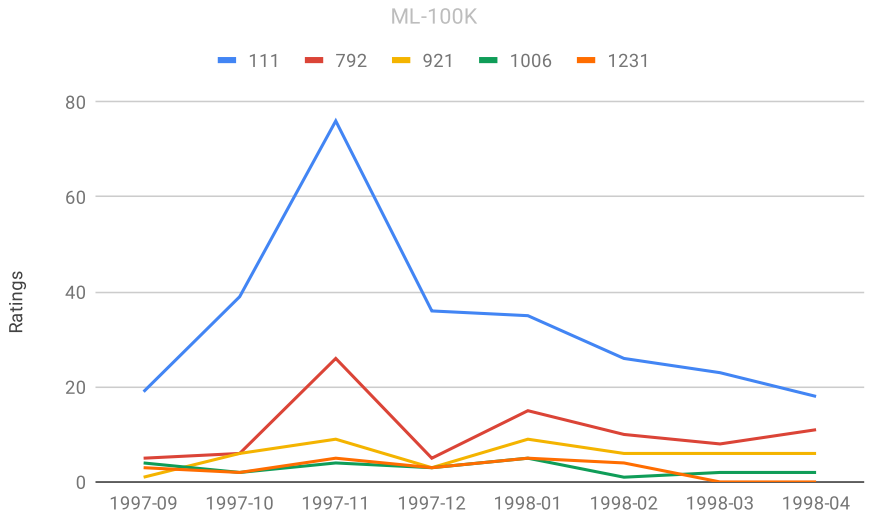


Figure 2.11: Number of movie ratings over time

2.2.2 Movie information

The ML-100K dataset contains very limited information about the rated movies. The information for each movie is limited to a movie title, release date, video release date, IMDb URL and a set of genres (e.g. action, drama, thriller). To further limit the available information, the IMDb URLs are no longer valid as of 2019-03-27 and most movies do not have a video release date assigned.

While the ML-100K dataset in itself contains very limited information, additional information is available elsewhere. In [7], the authors use English abstracts of the movies collected from a publicly available set of abstracts from DBpedia[2]. These abstracts are unstructured text descriptions of the movies, that are able to describe the contents of movies in much more detail than, say, a genre tag.

Internet Movie Database (IMDb) provides data about movies for personal and non-commercial use[18]. The data contains plenty of useful attributes

[2]https://wiki.dbpedia.org/

such as cast, directors and language. The full set of attributes can be seen in Table 2.4, which is based on the dataset descriptions in [18]. Version specific information and information with limited value has been filtered out. Table 2.5 describes the available information about people referenced in Table 2.4.

2.2.3 Demographic information

In addition to information about movies, the ML-100K dataset also contains basic demographic information about the users. Such information may be useful in providing recommendations based on a profile of the user. The available information about users is summarised in Table 2.6.

2.3 Dataset comparison

In this section we compare the 01Intern and ML-100K datasets. Table 2.7 summarises the similarities and differences between the two datasets.

Whereas the ML-100K user-item interaction matrix is considered sparse, that of 01Intern is very sparse in comparison. If we consider only the users with 20 or more ratings in the 01Intern dataset, the density of the matrix becomes 1.43%, which is a lot closer to the 6.30% of the ML-100K dataset.

The 01Intern dataset has three channels of interactions: views, favourites and applications, while ML-100K has only a single channel: ratings. Where all 01Intern channels can be considered implicit feedback, the feedback of ML-100K is explicit. This means that models that require feedback are suitable for the 01Intern dataset, however, methods such as Matrix Factorisation are applicable if feedback from users is considered a rating of 1[19]. Methods designed to train on implicit feedback have been proposed, such as Bayesian Personalised Ranking (BPR)[20], where items a user has interacted with are considered preferable to the items the user has not interacted with, but the lack of feedback for an item does not indicate that the user dislikes the item.

There is an abundance of content information available for both datasets, however, for ML-100K the majority is not included in the dataset itself. The data can, however, be acquired elsewhere with little effort.

Ttem cold start might pose a problem in both datasets. User cold start may be problematic for recommender systems working with the 01Intern dataset, as for many users there are few ratings to infer a user preference from. In the ML-100K dataset this is not a problem, as each user has at least 20 ratings. However, in the setting of online recommendation both the item and user cold-start problems are problematics, as all users and items not involved in the initial training of the model will be new at some step in the data stream. Thus in the online recommendation setting, methods for alleviating cold start should be considered.

The temporality of items is much more prevalent in the 01Intern dataset. Most jobs receive the majority of their interactions in the first few months, and jobs often lose their relevancy with time. While one could argue that new movies are often more relevant than older movies, only very few movies in the ML-100K dataset can be considered new. Thus, temporality is not of as high relevance.

Field	Description	Field type
Corporation id	The id of the corporation offering the internship. The relevant fields of corporations are listed in a different table.	Id
Job type	The job type of the internship. There are 10 different types of jobs in the dataset. Job types include *sales*, *engineer* and *designer*.	Enum
Catch copy	The catchphrase associated with the internship. This is a text field with the role of catching the attention of users.	Text
Appeal	A more detailed description of the internship and what makes it interesting.	Text
Entrusted job detail	A description of the tasks interns will be assigned during the internship.	Text
Grow up skill	The skills that interns will develop over the course of the internship.	Text
Grow up skill ids	Ids associated with the skills described in the `Grow up skill` field.	List of ids
Salary detail	Information about the salary interns will receive.	Text
Qualification	The qualifications required for the internship.	Text
Working condition	Information about the working hours for the internship.	Text
Sticking condition	This field enumerates the conditions that might make the internship attractive to interns. For example, interns may be able to use English as a part of the internship or work during the weekend.	List of ids
Work location	The location where the internship takes place. Includes prefecture, city, address and nearest train station.	Text
Required entry comment	Whether the users are required to write a comment when applying for the internship.	Boolean
Priority category	The priority category to which the internship belongs. Internships with higher priority are shown earlier in job listings.	Enum
Status	The status of the internship, e.g. whether it is open for application, temporarily closed or deleted.	Enum
Published	The date the job was published.	Date

Table 2.1: The relevant fields of jobs in the 01Intern dataset

Field	Description	Field type
Business type	The business type id of the corporation	Id
Secondary business type	The secondary business type id, if applicable	Id
Corporation detail	An introduction to the corporation.	Text
Business detail	A detailed description of the business the corporation conducts.	Text
Employee count id	The id of the employee count category to which the corporation belongs. “Between 21 and 50 employees” is an example of such a category.	Id
President name	The name of the president of the corporation.	Text
Address	The address of corporation main office.	Text
Established	The date when the corporation was established.	Date

Table 2.2: The relevant fields of corporations in the 01Intern dataset

Field	Description	Field type
Gender	This field describes whether the user is male or female.	Enum
Address	The prefecture and city the user resides in.	Text
University	The university the user is currently attending.	Text
Faculty	The faculty at the university to which the user belongs, e.g. *the Faculty of International Relations*.	Text
Department	The department at the university to which the user belongs, e.g. *the Department of Intercultural Communication*.	Text
Status	Status of the user, of which there are two: *active* and *delete*. Users with the active status are users that have been created, can log on and may apply for jobs. These accounts may or may not be in use. Users with the delete status have been deleted and may no longer be used.	Enum
Created	The date for the creation of the user.	Date

Table 2.3: The relevant fields of users in the 01Intern dataset

Field	Description	Field type
title	The localised title of the movie	string
language	The language of the title	string
titleType	The type of the title (e.g. movie or short)	string
startYear	The release data of a title or in the case of a TV series, the series start year	string
runtimeMinutes	Primary runtime of the title in minutes	integer
genres	Up to three genres associated with the title	array of strings
directors	Director(s) of the given title	array of ids
writers	Writer(s) of the given title	array of ids
castAndCrew	IDs for cast and crewmembers including job category, specific job title and character names	arrays of composites

Table 2.4: Collective attributes from the IMDb datasets

Field	Description	Field type
primaryName	Name by which the person is most often credited	string
birthYear	The person's year of birth	string
deathYear	The person's year of death, if applicable	string
primaryProfession	The top-3 professions of the person	array of strings

Table 2.5: IMDb person data

Field	Description
age	Integer age of the user
gender	'M' or 'F'
occupation	String representing the occupation of the user (e.g. salesman, student or doctor)
zipcode	ZIP code associated with the address of the user

Table 2.6: The demographic information in the ML-100K dataset

	01Intern	ML-100K
Interaction matrix	very Sparse	sparse
Interaction channels	views, favourites and applications	ratings
Content information	abundant in dataset	can be collected
Item cold start	potential	potential
User cold start	potential	if online
temporality	very relevant	less relevant

Table 2.7: Differences between the 01Intern and ML-100K datasets

3. Problem statement

Through the analysis of the 01Intern and ML-100K datasets we have gained insights into the characteristics they exhibit. Notably the two datasets are rich on both structured and unstructured attributes. As both datasets show characteristics that might cause cold-start problems for recommender systems working with the datasets, the item attributes should be leveraged in the extent that it is useful in overcoming these problems.

Furthermore, due to the online setting of 01Intern, items should be available for recommendation soon after they are received in the data stream. This means both the item and user cold-start problems are present to some extent, as all items and users will be new at some point in time. Although the user cold start can be a problem in the case of the 01Intern dataset, the problem will focus on item cold start. The conclusion of the analysis is the following problem statement:

How can information about internships be exploited jointly with the user-internship interactions to alleviate the problem of cold start in an online recommendation setting?

1. How can the union of structured and unstructured attributes of internships be leveraged so that rare internships may be recommended to users?
2. Which internship attributes are most relevant to the users of 01Intern, and which attributes introduce noise into the distinction between internships? How do we appropriately balance these attributes?
3. How can collaborative filtering and content-based filtering components be balanced to maximise accuracy of the recommender system?
4. What are the necessary measures that must be taken for the recommender system to be suited for an online setting?

4. Preliminaries

In this chapter, a number of preliminary methods are described. Notably, methods for describing different types of attributes, as well as using them for the task of recommendation, are presented.

4.1 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA)[21] is a generative probabilistic model for collections of discrete data such as text corpora. Among its uses is its ability to characterise the documents in a corpus as a mixture of a number of topics. In this section, a description is provided of the generative model of LDA and a variational inference algorithm that approximates the posterior probability of the observed data. A variational Bayes algorithm that processes a corpus in batch is presented, as well as an online variational inference algorithm, that is able to iteratively process mini-batches of documents as they are received from a data stream.

LDA is a topic model, which means it characterises documents as a mixture of the topics they exhibit. Topic models are useful because they allows us to describe documents more concisely. In a field such as Natural Language Processing (NLP), where we often have to deal with very large amounts of unstructured text, short, yet representative descriptions are sometimes a crucial factor for the time it takes to run the algorithm. Examples of such tasks may be clustering documents or measuring the similarity of documents. Topic models are not limited to text documents; LDA can be used in many domains such as collaborative filtering, content-based image retrieval and bioinformatics[21], in which large amounts of data are also processed. When describing LDA we consider the concrete domain of text documents in an effort to improve clarity.

LDA is a Bayesian model operating at three levels, namely the *corpus*, *documents* and *words*. A corpus consists of M documents, each with its own mixture of topics, and each document consists of N words, each of which is generated from a specific topic. A graphical representation of LDA can be seen in Figure 4.1. In the figure, the plates indicate that the random variables inside are actually multiple random variables, as the plate is repeated a number of times. For example, the outer plate is repeated M times, once for each document, and thus the topic distribution a document is actually M random variables in the Bayesian network. is the corpus-level distribution of topics. Each word w is generated from a specific topic z. In the figure, the node for w is coloured to indicate that it is an observable variable. Lastly, is the per-topic term distribution, that is, the probabilities of each term in the corpus vocabulary being drawn when the topic is known.

In LDA, when generating each document w the following generative process from [21] is assumed:

1. Choose $Dir(\ )$
2. For each word w_n in w:
   (a) Select a topic z_n $Multinomial(\ )$
   (b) Select w_n from $p(w_n|z_n;\ )$

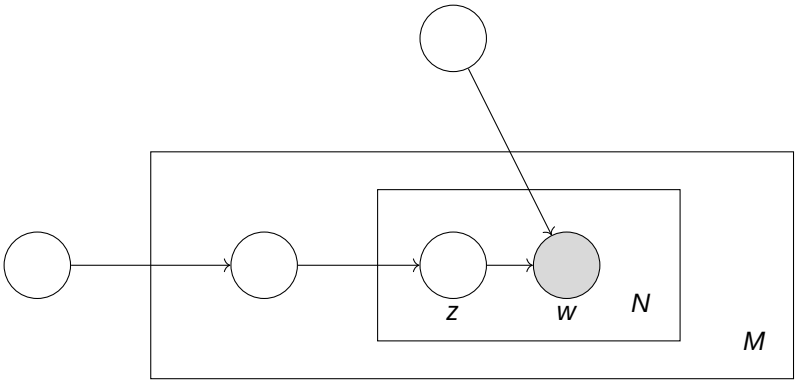
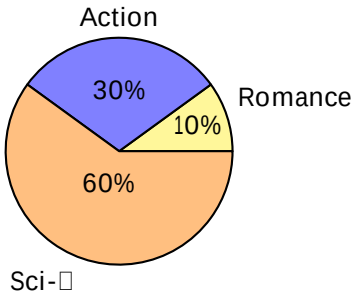


Figure 4.1: Graphical representation of the Latent Dirichlet Allocation model

To give a more concrete example of this generative process, let us consider an LDA model that generates descriptions of movies. For simplicity, our model only has three topics: *action*, *romance* and *sci-* . When generating a new description of a movie, we rst draw from the Dirichlet distribution parameterised by the corpus topic distribution . After drawing, the topic mixture may be the following:



Next, we generate the N words by rst selecting z_n and then selecting w_n the multinomial distribution conditioned on the selected z_n and the per-topic term distribution . Evidently, given topic z_n has a higher probability of being sci- than action, and the romance topic has the lowest probability of being selected. Let us assume we select the action topic for z_n. When selecting w_n, there is a relatively high probability of selecting the terms “explosion” and “ght” given z_n = *action*, while the terms “spaceship” and “kiss” have a comparatively low probability of being selected.

As one might notice, in LDA no effort is made to ensure that words appear in any meaningful order. Rather, the model generates a collection of terms in which the order plays no role. Such a representation of text documents is often referred to as a *bag of words*.

In Figure 4.2 we show that LDA generates a corpus. As mentioned earlier the strength of topic models lies in being able to provide compact descriptions of the documents they model. As we see at the bottom, however, when given a corpus we may use Bayesian inference to compute the posterior of the hidden variables and z. However, due to the complexity of the model, the

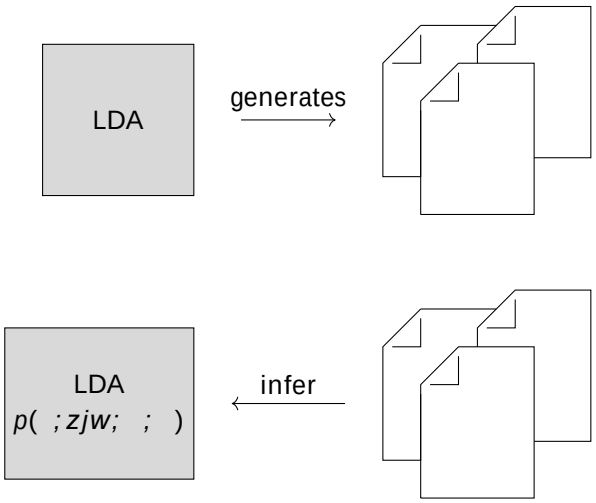


Figure 4.2: (Top) LDA generates a corpus using the model parameters and , where is the topic distribution across the entire corpus and is the per-topic term distribution. (Bottom) Given a corpus, the posterior distribution of the hidden variables and z is inferred through Bayesian inference.

posterior distribution is intractable for exact inference. Fortunately, a number of approximate inference algorithms can be considered for LDA. In the next Section, we will see how variational Bayesian inference can be used to approximate the hidden variables.

4.1.1 Variational inference for Latent Dirichlet Allocation

For the variational inference algorithm the goal is to nd an adjustable lower bound on the log likelihood. In this section, we summarise key ideas of the variational inference presented in [21], and later [22], with the purpose of obtaining a set of parameter update rules that allow us to maximise the so-called Evidence Lower Bound (ELBO), a lower bound of the log probability of the observed variables[23], and thus t the model.

In LDA, it is intractable to compute the posterior distribution of the hidden variables due to the coupling between and in the summation over latent topics. This coupling arises due to the edges between the , z, and w nodes. One can obtain a family of distributions on the latent variables by dropping these edges and introducing the free variational parameters, Dirichlet parameter and multinomial parameters , resulting in the model represented in Figure 4.3. Maximum likelihood estimates of the multinomial parameters assign a probability of 0 to words that did not appear in any of the training document, meaning unseen documents will have no probability of being generated. To cope with this, a solution is to perform Dirichlet smoothing of the multinomial parameters by assuming that each row of is drawn from an exchangeable Dirichlet distribution parameterized by the scalar parameter . Therefore, an additional variational parameter is introduced into the variational model.

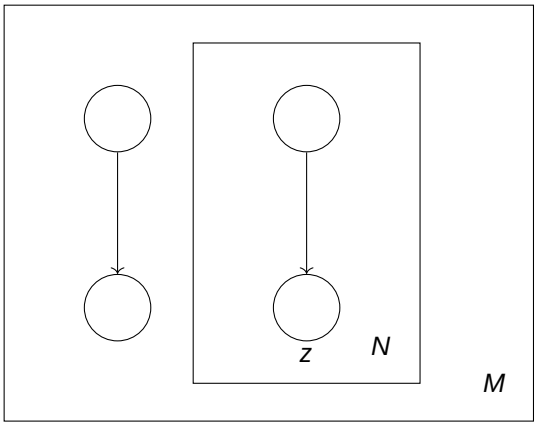


Figure 4.3: Graphical representation of the variational distribution used for inference of the posterior

The problem of finding a tight lower bound on the log likelihood is defined as follows:

$$\log p(w|\,;\,) \quad L(w;\;;\;;\;) \;,\; \mathrm{E}_q[\log p(w,z;\;;\;|\;;\;)] - \mathrm{E}_q[\log q(z;\;;\;)] \tag{4.1}$$

The Kullback-Leibler (KL) divergence is a function that is often "used as a measure of the similarity between two probability densities"[24, p. 1]. By minimising the KL divergence between the variational distribution $q(z;\;;\;)$ and the true posterior $p(z;\;;\;|w;\;;\;)$, one obtains the optimising values of the variational parameters. Then, computing the derivatives of the KL divergence and setting them equal to 0, the following update rules for the variational parameters are obtained:

$$_{dwk} \;/\; \exp\{\mathrm{E}_q[\log \;_{dk}] + \mathrm{E}_q[\log \;_{kw}]\} \tag{4.2}$$

$$_{dk} = \; + \sum_{w} \;_{dwk} n_{dw} \tag{4.3}$$

$$_{kw} = \; + \sum_{d} n_{dw} \;_{dwk} \tag{4.4}$$

Here, the expectations of log and log under q are given as

$$\mathrm{E}_q[\log \;_{dk}] = (\;_{dk}) \quad (\sum_{i=1}^{K} \;_{di})$$

and

$$\mathrm{E}_q[\log \;_{kw}] = (\;_{kw}) \quad (\sum_{i=1}^{W} \;_{ki}),$$

where is the digamma function, the logarithmic derivative of the gamma function .

The variational objective relies only on n_{dw}, that is, the number of times word w appears in document d. This means that documents can be represented as the number of times each word appears.

4.1.2 Batch variational Bayes algorithm

In this section we present the batch variational Bayes inference algorithm presented in [22]. The algorithm takes as input a training corpus of D documents as well as hyperparameters and . The algorithm alternates between an expectation step (E step), in which we optimise the variational parameters (indicates that the variational parameters are a function of w, which is held xed) for each $d \in D$, and a maximisation step (M step), where we maximise L with respect to the model parameters and , until L converges.

```
1  Initialise      randomly.
2  while relative improvement in L(w; ; ; ) > 0:00001 do
3      E step:
4      for d = 1 to D do
5          Initialize   dk = 1. (The constant 1 is arbitrary.)
6          repeat
7              Set  dwk / exp{E_q[log  dk] + E_q[log  kw]}
8              Set  dk =  + P_w  dwk n_dw
9          until (1/K) P_k |change in  dk| < 0:00001
10     end for
11     M step:
12     Set  kw =  + P_d n_dw  dwk
13 end while
```

Listing 4.1: The batch variational inference algorithm for LDA, as presented in [22]

In Listing 4.1, the variational Bayes inference algorithm for LDA is shown. As seen on lines 4-10, we repeatedly update the variational parameters until convergence for each document in the training corpus. The updates performed are those shown in Equation 4.2 and Equation 4.3. After the expectation step, the posterior over the per-topic terms, , is updated as in Equation 4.4. The algorithm alternates between the two steps until it converges, measured by the relative improvement in L.

4.1.3 Online variational inference algorithm

While the batch inference algorithm presented in Section 4.1.2 processes a text corpus in batch, the online variational Bayes algorithm presented in [22] has been adapted to a setting, where discrete data documents are received one at a time.

Indeed, this algorithm is similar to the batch variational Bayes algorithm, but differs in the following ways:

- Instead of working on a complete training corpus, the algorithm operates on mini-batches of documents of an arbitrary size, each of which is processed at a time t.
- At each iteration, instead of performing a full update in accordance with the update rule in Equation 4.4, the model de nes the new value for as a weighted average of the current value and the optimal value if corpus only consisted of the mini-batch currently being processed, $\tilde{}$. The weighted average is computed as $(1 \quad _t) + _t \tilde{}$, where $_t$ is de ned as , $(_0 + t)$.

In the definition for ρ_t, τ_0 and κ are constants that control the rate of learning. τ_0 slows down learning for early iterations of the algorithm. κ controls the rate at which the values for $\tilde{}$ in earlier iterations are forgotten. Here, a lower value for κ leads to faster forgetting of older values. When κ is set to 0, the online inference algorithm for LDA becomes equivalent to the batch variational Bayes algorithm, as older values are completely forgotten with each iteration. κ has to satisfy the condition $0.5 < \kappa \leq 1$ in order to guarantee convergence. The algorithm for online variational Bayes for LDA is shown in Listing 4.2.

```
1  Define ρ_t , (τ_0 + t)^−κ
2  Initialise λ randomly.
3  for t = 0 to ∞ do
4      E step:
5      Initialize γ_tk = 1. (The constant 1 is arbitrary.)
6      repeat
7          Set φ_twk ∝ exp{E_q[log θ_tk] + E_q[log β_kw]}
8          Set θ_tk = α + Σ_w φ_twk n_tw
9      until (1/K) Σ_k |change in θ_tk| < 0.00001
10     M step:
11     Compute λ̃_kw = η + D n_tw φ_twk
12     Set λ = (1 − ρ_t) λ + ρ_t λ̃.
13 end for
```

Listing 4.2: The online variational inference algorithm for LDA, as presented in [22]

In [22], experiments are performed to test the effectiveness and efficiency of online LDA. The effectiveness is measured by the perplexity of a held-out test set of documents, where perplexity is defined as "the geometric mean of the inverse marginal probability of each word in the held-out set of documents"[22, p. 7]. A lower perplexity score indicates a higher effectiveness of the model, through being able to generalise documents better. The experiments show that the model achieves the best performance with a batch size of 4,096 for both of the tested text corpora. However, for batches of 256 to 16,384 documents, there is little difference in terms of document perplexity. Batches of more than a single document are, however, not a possibility in the online recommendation setting we assume.

4.2 Adaptive Windowing

Adaptive Windowing (ADWIN) [25] is an active drift detection algorithm that has gained a lot of interest due to its simplicity and the theoretical performance guarantees for the rate of false positives and false negatives it provides [7]. In ADWIN, instead of using sliding windows with a fixed size, the size is dynamic and adjusted based on the rate of change in the data being observed. This saves the user having to guess a time-scale for change [25].

The algorithm keeps a window W of variable size containing bits or real numbers. This window automatically grows when no change in the distribution of the observed data is detected and shrinks when data changes. In Section 4.2.1 a naive version of the ADWIN algorithm is presented, and in Section 4.2.2 a more efficient version, called ADWIN2, is presented. Both variants take as input a confidence value $\delta \in (0, 1)$, which controls the sensitivity of the algorithm, and a stream of real values x, the values of which are generated independently according to some distribution D_t, which may change at each time t. Stream x consists of values $x_1, x_2, \ldots, x_t, \ldots$ of which only x_t is available at time t.

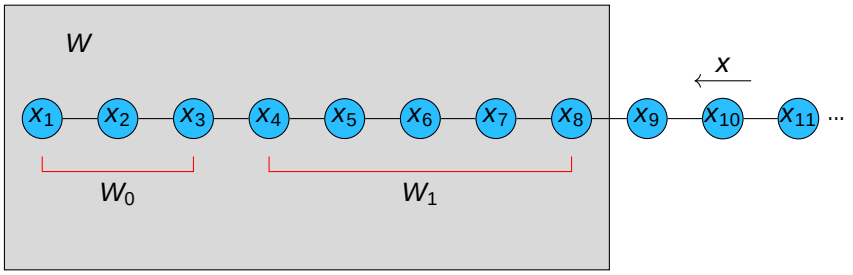


Figure 4.4: Example partitioning of sliding window W into subwindows W_0 and W_1 in ADWIN.

4.2.1 Naive variant

In [25], a naive variant of ADWIN is presented. The algorithm detects drifts by partitioning W into subwindows W_0 and W_1 and checking if the difference of the averages of the two subwindows is above a threshold $_{cut}$. The intuition is that when two large enough subwindows have average values that are suf-ciently different, it indicates that a data drift has occurred. The notion "large enough" and "suf-ciently different" is de-ned through threshold $_{cut}$.

The algorithm tries every possible con-guration of subwindows W_0 and W_1. Figure 4.4 illustrates one possible partioning into subwindows W_0 and W_1. In this setting, W contains values $x_1; x_2; :::; x_8$. W_0 contains values x_1, x_2 and x_3, while W_1 contains the remaining -ve values currently in W. Let $\hat{}_{W_0}$ and $\hat{}_{W_1}$ be the average value of W_0 and W_1, respectively. Then, a drift is detected if the following holds:

$$j\hat{}_{W_0} \quad \hat{}_{W_1} j \quad _{cut}$$

The treshold $_{cut}$ is de-ned as follows:

$$_{cut} = \sqrt{\frac{1}{2m} \ln \frac{4}{\ }} \tag{4.5}$$

In Equation 4.5, m is the harmonic mean of the size of W_0, n_0, and the size of W_1, n_1, de-ned as $m = \frac{1}{1=n_0 + 1=n_1}$. $\ $ is de-ned as $\ = \frac{}{n}$.

```
1  Initialize  Window W
2  for each t > 0
3      do W     W [ x_t (i.e., add x_t to the head of W)
4          repeat Drop elements from the tail of W
5              until j^_W0   ^_W1 j <  _cut holds for every split of W into W0  W1 = W
6          output ^_W
```

Listing 4.3: Naive variant of the ADWIN algorithm, as presented in [25]

In Listing 4.3, the naive variant of the ADWIN algorithm is presented. At each step t, W -rst grows as a new value is added to its head. If there exists a partitioning of W into W_0 and W_1 such that $j\hat{}_{W_0} \quad \hat{}_{W_1} j \quad _{cut}$, W will shrink by dropping elements from its tail until such a partitioning no longer exists, as seen on line 5. After the repeat-until loop on lines 4-5, the algorithm outputs the mean of W, $\hat{}_W$.

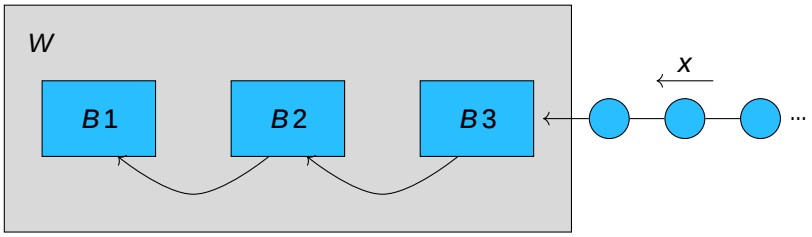


Figure 4.5: The ADWIN2 algorithm stores values in a variant of an exponential histogram to reduce memory and time complexity

4.2.2 Improving performance with exponential histograms

The naive variant described in Section 4.2.1 partitions W into every possible con□guration of subwindows W_0 and W_1, and thus is computationally expensive. A more ef□cient variant, named ADWIN2, is proposed in [25], which differs from the naive variant in that it uses an exponential histogram as its underlying data structure.

The ADWIN2 algorithm is examined in [26]. In an exponential histogram, the values being inserted are not stored individually, but are assigned to buckets. A bucket contains the sum and the variance of the elements it represents. Figure 4.5 illustrates the idea of ADWIN2, where new values from x are inserted into the newest bucket, in this case $B3$. Whenever a value is inserted, buckets may be compressed and smaller buckets may be combined to form larger buckets. Consequently, newer buckets contain only few elements, whereas older buckets contain an exponentially growing number of elements. When an element has been inserted and the buckets updated, drift detection takes place as in the naive algorithm. In ADWIN2, the subwindows are created using the buckets of the exponential histogram. This signi□cantly reduces the complexity to $O(log(n))$.

Experiments in [25] show that ADWIN2 performs only slightly worse than the best window for each rate of change and performs far better than any □xed-size window W, when the rate of change is very different from W.

4.3 Matrix Factorisation

In Section 4.3-4.6, we will take a closer look at MF models. MF models are induced by factorisation of the user-item ratings (or interactions) matrix, and are popular due to their attractive accuracy and scalability [27]. Furthermore, MF models allow for the incorporation of additional information sources, such as implicit feedback or item attributes. In Section 4.5, we explain two approaches for enhancing a basic MF model with item information. This section and Section 4.4 are heavily based on [3].

MF models are latent factor models that decompose the ratings matrix into two lower-rank matrices P and Q, for users and items respectively. By representing users and items as vectors of latent factors, we can make predictions about the missing values of the ratings matrix based on these factors. Each factor in the latent space measures some characteristic of items in the domain.

For example, in the case of movie recommendations factors might measure whether a movie is a comedy or more serious, or whether it is for children or targets an older audience. Factors might measure less well-defined characteristics, or even something that is incomprehensible to humans.

As stated previously, MF decomposes the ratings matrix into lower-rank matrices P and Q. Let the ratings matrix be an $M \times N$ matrix, where M is the number of items and N is the number of users. Matrix P then has dimensions $K \times N$, where K is the number of latent factors. Similarly, Q has dimensions $K \times M$. Each column p_u of P and q_i of Q, where $q_i, p_u \in \mathbb{R}^f$, corresponds to the latent factors for a particular user or item. Users and items are mapped to a joint latent factor space, but the factors can be interpreted differently. An item factor vector q_i represents to which extent item i possesses those factors. A user factor vector p_u indicates the preference of user u for items that have a high value for each of those factors. With this intuition it is easy to see how the dot product of vectors p_u and q_i approximates the rating of user u for item i. This approximate rating, denoted $\hat{r}_{ui}$, is as follows:

$$\hat{r}_{ui} = q_i^\intercal p_u \tag{4.6}$$

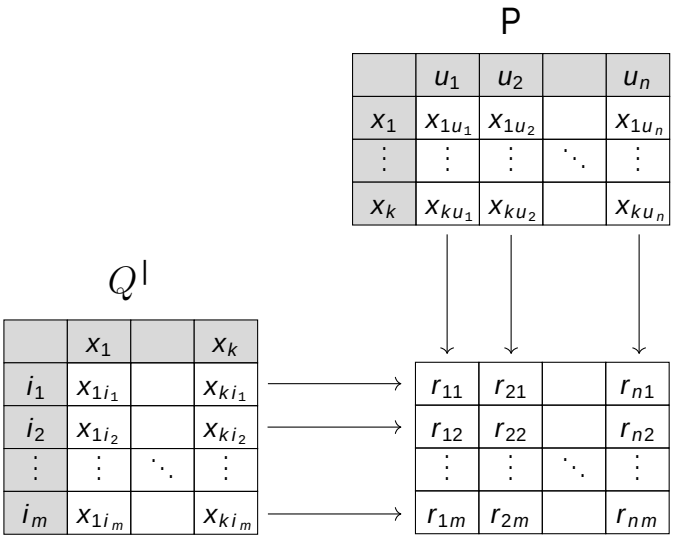


Figure 4.6: Decomposition of the ratings matrix into lower-rank matrices P and Q. The product, $Q^\intercal P$, yields the ratings matrix with every entry filled in.

Figure 4.6 shows how after the decomposition of the ratings matrix into P and Q, the product $Q^\intercal P$ yields a matrix where each row corresponds to an item, each column corresponds to a user, and each entry contains an estimate of the rating for the item by that user. Each column in P and Q corresponds to a user or an item, while each row corresponds to one of the latent factors $x_1, \ldots, x_k$. Since the dimensions of $Q^\intercal$ are $M \times K$ and P are $K \times N$, the product has dimensions $M \times N$. Thus, once we have obtained matrices P and Q, approximating the rating for any user-item pair can be trivially done by computing the dot product of the columns corresponding to that user and item.

The main problem of MF thus is estimating the lower-rank matrices.

One approach for estimating the lower-rank matrices is Singular Value Decomposition (SVD)[28], which decomposes a rectangular matrix A into a product of three matrices:

$$A = U \quad V \; ; \tag{4.7}$$

where U and V are unitary matrices, whose columns are called the left-singular and right-singular vectors of A, respectively, and is a rectangular diagonal matrix, whose diagonal elements are known as the singular values of A. The problem with conventional SVD is that it requires the matrix to be complete: in a recommendation setting the ratings matrix is usually very sparse. In [29], the missing entries of the ratings matrix are filled in with the average rating for the user or item in an effort to solve this problem. However, not only is this computationally expensive; it might also distort the data and is prone to overfitting.

Another popular approach is to model directly the observed ratings with a learning objective that minimises the so-called regularised squared error on the set of known ratings:

$$\min_{q\;;p} \sum_{(u;i)2} (r_{ui} \quad q_i^{|} p_u)^2 + \quad (jjq_ijj^2 + jjp_ujj^2); \tag{4.8}$$

Here, is the set of user-item pairs for which a rating is known, is a hyper-parameter that controls the degree of regularisation, and jjq_ijj denotes the L^2-norm of vector q_i. The regularisation term $(jjq_ijj^2 + jjp_ujj^2)$ imposes a penalty on the magnitude of the factor vectors in order to prevent overfitting. Overfitting the training data improves prediction accuracy on the training data, but affects the generalisation power of the model, impairing the prediction accuracy on future data.

In some MF models, the the user and item matrices are regularised by different amounts. In this case the learning objective can be written as:

$$\min_{q\;;p} \sum_{(u;i)2} (r_{ui} \quad q_i^{|} p_u)^2 + \quad _ijjq_ijj^2 + \quad _ujjp_ujj^2 \tag{4.9}$$

The number of latent factors K is a parameter that can be adjusted to optimise the performance of the model. With a low value for K, the factors are only able to capture the most important aspects of the data. As we increase the value for K, we expect to see increasingly obscure characteristics represented by the factors. In [30], experiments on the Netflix test set show that prediction accuracy improves as the number of latent factors is increased across several MF variants.

4.4 Learning algorithms for Matrix Factorisation

In the next sections we present two approaches for minimising the learning objective in Equation 4.8. The approaches presented are Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS), which are the most successful methods to solve this optimisation problem[19].

4.4.1 Stochastic Gradient Descent

A popular algorithm for optimising the learning objective using SGD is to iterate over the set of available ratings and modifying the parameters with each rating. The approach is popular due to being fast and easy to implement. Additionally, an incremental variant[19] has been proposed, which is suited for recommendation in an online setting.

In this algorithm, with each iteration the parameters are modified in the opposite direction of the gradient of the learning objective by a magnitude proportional to the hyperparameter , which controls the learning rate. Since we are modifying the model parameters with every iteration, we use the gradient of the learning objective for a single rating. In the case of the learning objective from Equation 4.8, we obtain the following gradient for model parameter p_u:

$$\begin{aligned} \frac{@}{@p_u}(r_{ui} \quad q_i^{|} p_u)^2 + \quad (jjq_ijj^2 + jjp_ujj^2) \\ = \quad 2q_i(r_{ui} \quad q_i^{|} p_u) + 2 \ p_u \end{aligned} \tag{4.10}$$

Let the prediction error e_{ui} be defined as follows:

$$e_{ui} \ , \ r_{ui} \quad q_i^{|} p_u \tag{4.11}$$

Then, the gradient can be written as

$$2q_ie_{ui} + 2 \ p_u: \tag{4.12}$$

Since the scalars are absorbed by the hyperparameters, we can write this simply as

$$q_ie_{ui} + \quad p_u: \tag{4.13}$$

Modifying p_u in the opposite direction of the gradient proportionally to , we obtain the following update rule:

$$p_u \quad p_u + \quad (e_{ui}q_i \quad p_u) \tag{4.14}$$

Because the original learning objective is symmetric, we can replace p_u with q_i and q_i with p_u to obtain the update rule for q_i:

$$q_i \quad q_i + \quad (e_{ui}p_u \quad q_i) \tag{4.15}$$

One obvious advantage of SGD is that by updating the parameters at each step, rather than with each full iteration, we take advantage of the high sparsity of the ratings set. This means that the complexity grows linearly with the number of known ratings[19].

The pseudocode for this algorithm can be seen in Listing 4.4. Here, R is the set of known ratings. The outer loop controls the number of iterations over R. On line 2, R is shuffled in order to avoid cycles when further passes are made over R[19]. On lines 4-6 we see the parameters being updated for a single rating $< u; i; r > 2 \ R$ using the equations that were just presented.

```
for i = 0 to iterations do
    shuf□e set of ratings R
    for < u; i; r >2 R do
        e_ui = r_ui   q_i^| p_u
        q_i   q_i +  (e_ui p_u   q_i)
        p_u   p_u +  (e_ui q_i   p_u)
    end for
end for
```

Listing 4.4: Pseudocode for the Stochastic Gradient Descent algorithm

4.4.2 Alternating Least Squares

The learning objective in Equation 4.8 is a biconvex optimisation problem, which are in general global optimisation problems and may have a large number of local minima[31]. However, by □xing one of the unknowns, the problem becomes quadratic and can be solved ef□ciently as a least-squares problem. This is the approach taken in ALS. We □x P and recompute Q by optimising a least-squares problem. Then, we □x Q and recompute P in the same manner. Alternating between □xing each unknown eventually leads to convergence.

Although it has been shown that SGD-based optimisation generally converges faster and has higher accuracy ALS on sparse datasets, ALS is preferable to SGD in two cases [19]. When the system can utilise parallelisation, we can optimise the learning process by computing the factor vectors in parallel. This is possible because each factor vector is recomputed independently of the other vectors. The other case where ALS outperforms SGD is when the dataset is not sparse. One practical example of this is a system centered on implicit feedback. Iterating over the entire dataset is not as ef□cient in this case, hence why ALS - which does not suffer from this problem - has an advantage.

4.5 Leveraging item information

Pure collaborative-□ltering-based recommender systems suffer from the cold start problem when certain users or items have only few interactions. In this case additional information may be included in the model to leverage the problem. In this section we present two approaches for enhancing the previously presented MF algorithm with information about items.

4.5.1 Representing attributes with latent factors

One approach proposed in [3] is to represent item attributes by vectors of latent factors, so that an attribute a is associated with the factor vector $y_a \; 2 \; \mathrm{R}^f$. The item information for an item i can thus be represented as the sum of the factor vectors associated with the attributes that i has:

$$\sum_{a2A(i)} y_a \tag{4.16}$$

Here, $A(i)$ is the function that returns the set of attributes for an item i. For example, in the case of movie recommendations, we may associate each genre

a movie can belong to with a factor vector. If a movie belongs to the “comedy” and “action” genres, the factor vector representing that item is the sum of the factor vectors for comedy and action. More attributes may be included, such as directors, actors and writers.

By integrating the item information representations into the rating prediction function from Equation 4.6, we get the following:

$$\hat{r}_{ui} = [q_i + \sum_{a \in A(i)} y_a]^\intercal p_u \tag{4.17}$$

This approach has the advantage that weights for different item attributes are integrated in the model, and the weights users assign to different attributes do not have to be modelled explicitly. However, the approach is limited in that we are unable to use existing item similarity measures directly. Furthermore, non-discrete attributes are difficult to model. For example, a continuous attribute such as release year would have to be partitioned into a finite number of groups, e.g. ’80s, ’90s and so on. This, in turn, introduces the problem of selecting the right granularity. To distinguish this method from the method presented in the next section, we refer to it as Attribute-Enhancement (AE).

4.5.2 Regularising latent factors of similar items to be closer

A different approach for enhancing the matrix factorisation model is proposed in [32]. The idea of the method to regularise the factor vectors of items with similar content to be more similar. The work builds on [33], in which the same idea is applied to users: the factor vectors of users with similar tagging histories are regularised to be more similar. To distinguish this method from the AE method described in the previous section, we refer to it as Factor Vector Alignment (FVA).

In the method, an arbitrary function $w(i; i')$ is used, which models the similarity between items i and i'. The intuition of the method is simple: we want to move the factor vector of items i and i', which w assigns a high score, "closer" to each other. On the other hand, items j and j', which are not similar according to w should not be moved closer to each other. A regularisation term is added that enforces this:

$$\sum_{i=1}^{|I|} \sum_{i'=1}^{|I|} \|q_i - q_{i'}\|^2 w(i; i') \tag{4.18}$$

Here, β controls the extent to which the factor vectors of similar items are made more similar. With this regularisation term, the Equation 4.9 becomes

$$\begin{aligned} \min_{q,p} \sum_{(u;i) \in \kappa} (r_{ui} - q_i^\intercal p_u)^2 + \lambda_i \|q_i\|^2 + \lambda_u \|p_u\|^2 \\ + \beta \sum_{i=1}^{|I|} \sum_{i'=1}^{|I|} \|q_i - q_{i'}\|^2 w(i; i') \end{aligned} \tag{4.19}$$

The modified learning objective from Equation 4.19 can be approximated using the SGD algorithm described in Section 4.4.1, but with new update rules

for p_u and q_i. The update rule for p_u is largely the same as the update rule in Equation 4.14, we use separate regularisation hyperparameters $_u$ and $_i$ for users and items, respectively:

$$p_u \quad p_u + (e_{ui}q_i \quad {}_u p_u) \tag{4.20}$$

To find the new update rule for q_i, we compute the gradient of the learning objective from Equation 4.19 for a single rating w.r.t. q_i:

$$\begin{aligned}
&\frac{@}{@q_i}(r_{ui} \quad q_i^{|} p_u) + {}_i jjq_i jj^2 + {}_u jjp_u jj^2 + \sum_{i^{\theta}=1}^{N} jjq_i \quad q_{i^{\theta}} jj^2 w(i; i^{\theta}) \\
&= \quad 2e_{ui}p_u + 2 \ {}_i q_i + \sum_{i^{\theta}=1}^{N} 2(q_i \quad q_{i^{\theta}}) w(i; i^{\theta}) \\
&= \quad 2e_{ui}p_u + 2 \ {}_i q_i + 2 \ (q_i \sum_{i^{\theta}=1}^{N} w(i; i^{\theta}) \quad \sum_{i^{\theta}=1}^{N} w(i; i^{\theta}) q_{i^{\theta}})
\end{aligned} \tag{4.21}$$

Again, the scalars are absorbed by the hyperparameters. Taking this into account, the new update rule for q_i becomes:

$$\begin{aligned}
q_i \quad q_i + [e_{ui}p_u \quad {}_i q_i \\
(q_i \sum_{i^{\theta}=1}^{N} w(i; i^{\theta}) \quad \sum_{i^{\theta}=1}^{N} w(i; i^{\theta}) q_{i^{\theta}})]
\end{aligned} \tag{4.22}$$

Aside from the modified update rules, the SGD algorithm from Section 4.4.1 remains unchanged.

An attractive property of FVA is that the similarity measure function w is independent of the matrix factorisation model. However, this also means that when w computes the similarity of items based on several different item attributes, the weights have to be modelled explicitly. Another obvious advantage of FVA is the ability to accurately model attributes with infinite possible values, e.g. continuous attributes such as the price of a ware. In the approach previously explained, where attributes are represented by factor vectors, the values of such an attribute have to be reduced to a finite set of values, e.g. "< $50", "$50-100" and "> $100" for a price attribute. This introduces the problem of determining appropriate intervals, which is done independently of the model. With this approach, we can compare two prices using an arbitrary function that takes two continuous numbers as input and outputs the degree of similarity.

A core difference between the AE approach and this approach is that AE deals with representations of solitary items, while this approach deals with comparing pairs of items. This has the additional benefit of making it simple to compare attributes such as geographical location, an attribute which may prove difficult to represent effectively when taking the AE approach, as the concept of distance between different locations will not necessarily be captured by the factor vectors.

4.6 Incremental Matrix Factorisation

Incremental Stochastic Gradient Descent (ISGD), proposed in [19], is very similar to the batch SGD model presented in section 4.4, but with a few important differences. First, only a single pass is made over the set of ratings R. Secondly, unlike batch SGD, R is not shuffled. This means that the temporal aspect of R is maintained. Lastly, the learning algorithm is adapted to positive-only feedback by assuming a rating of 1 for every interaction in R. Consequently, the prediction error is defined as e_{ui} , 1 $q_i^{\dagger} p_u$. Note that in order for the algorithm to able to correctly infer factors for users and items, it is initially trained until convergence on a set of interactions. Pseudocode for the algorithm for ISGD is shown in Listing 4.5

```
1  for < u; i >2 R do
2      if u 2 P then
3          p_u     R^K
4          p_u     N (0; 0:1)
5      end if
6      if i 2 Q then
7          q_i     R^K
8          q_i     N (0; 0:1)
9      end if
10     e_ui = 1    q_i^| p_u
11     q_i     q_i +  (e_ui p_u     q_i)
12     p_u     p_u +  (e_ui q_i     p_u)
13 end for
```

Listing 4.5: Pseudocode for the Incremental Stochastic Gradient Descent algorithm

In the algorithm, the outer for-loop iterates over every rating in the set of ratings R. As interactions for users which do not yet have factor vector representations are processed, the factor vectors are added with the initial values drawn from the multivariate normal distribution, as seen on lines 2-5. K is the number of latent factors used. Items are given a similar treatment when they first appear, as seen on line 6-9. The update rules for q_i and p_u remain unchanged with the exception of the new definition of the prediction error, seen on line 10.

By making only a single pass over the set of ratings after the initial training, ISGD is an algorithm that can quickly process a lot observation. This makes ISGD a suitable model for the task of online recommendation, in which the model must be able to process observations faster than they appear from the data stream. Another notable advantage is its being able to provide recommendations to new users and for new items, as the model is updated at each step in time. Models that only occasionally retrain the model in batch are not able to provide recommendations for items and users added in between two training phases.

5. Related Work

In this chapter, a number of works related to the current problem are presented. A recommender system developed for the 01Intern dataset previously developed by the author is described, as well as a more sophisticated approach that combines MF with topic modelling using LDA.

5.1 kNN regression content-based recommender system

In a previous work by the author, a purely content-based recommender system was proposed, in which each item is ranked based on its content-based similarity to the user's rated items [12]. In the model, the similarity between two items is given by the similarity between different attributes of the items. Attributes are compared using different functions, which are not part of the model and are chosen according to the domain and the type of the attributes. Furthermore, each attribute is assigned a weight to model the characteristic that users do not consider every attribute equally important.

5.1.1 Prediction model

The model considers the k most similar items K already rated by a user to an unknown item i. The score for i is given by the score for each item $j \in K$ multiplied by the similarity of i and j:

$$score_i = \sum_{j \in K} r_j w(i; j) \tag{5.1}$$

where r_j is the rating given to item j by the user and $w(i; j)$ is the similarity between items i and j. If the user has rated less than k items, K is the set of all items rated by the user. In the case where ratings originate from a single channel of implicit feedback, all items in K have the same rating and thus the score for i is given by

$$score_i = \sum_{j \in K} w(i; j): \tag{5.2}$$

The similarity between two items i and i', $w(i; i')$, is computed by first comparing each attribute of i and i', and then multiplying the resulting vector of similarity scores by a vector of weights. Let $x \in \mathbb{R}^n$ be the vector of weights and $a_{ii'} \in \mathbb{R}^n$ be the attribute-wise similarity of items i and i'. Then, the similarity between items i and i' is given by

$$w(i; i') = x a_{ii'} \tag{5.3}$$

5.1.2 Computing item similarity

As mentioned previously, the functions used to compare attributes are not part of the model, but instead depend on the domain, i.e. whether the system recommends movies or scientific articles, and the type of the attributes, i.e. whether the attributes are structured or unstructured, continuous or discrete. In [12], the model is applied to the 01Intern dataset, in which items are characterised mainly by unstructured text and categorical attributes, as previously explained in the Section 2.1.4. Two different similarity measures are used. The

categorical attributes are compared using the Jaccard similarity, which for two sets A and B is the ratio between the intersection of the sets to the union of the sets:

$$Jaccard(A; B) = \frac{jA \setminus Bj}{jA [Bj}$$

For example, let $A = fAction; Adventureg$ and $B = fAdventureg$ be sets of genres for two movies. The Jaccard similarity of A and B then is:

$$Jaccard(A; B) = \frac{jA \setminus Bj}{jA [Bj} = \frac{jfAdventuregj}{jfAction;Adventuregj} = \frac{1}{2} = 0:5 \text{ [12]}$$

The unstructured text attributes are represented as vectors in the vector space model and compared using the so-called cosine similarity measure. In the vector space model, a text is represented an m-dimensional vector, where each dimension corresponds to a distinct term from the vocabulary shared by all texts. The mth dimension characterises the extent to which mth term of the vocabulary is represented in the text. To model this, the *TF-IDF* scheme is applied. TF-IDF stands for Term Frequency-Inverse Document Frequency, which are the two characteristics of text documents the scheme is based on. The scheme assumes that the more frequently a term appears in a text document, the more relevant it is to the topic of the document, and that the more documents a term appears in across a corpus, the more poorly it discriminates between documents [34].

The term frequency of a term i in a document D is the number of times i appears in D. The document frequency of a term i across a corpus C is the number of documents $D\ 2\ C$ that contain i. Let TF_i be the term frequency of term i, DF_i the document frequency of term i, and N the total number of documents in the corpus. Then, $TF\quad IDF_i$ is given by the following equation:

$$TF\quad IDF_i = TF_i\quad log(\frac{N}{DF_i})$$

The TF-IDF scheme is known for being simple yet effective. Its main strength is its ability to return documents that "highly relevant to a particular query"[35].

Two unstructured text attributes in vector representation D and D^0 are compared using the cosine similarity measure. The cosine similarity is a measure of the angle between the two vectors, and thus how related the topics of the two documents are. This means that for two documents to be considered equal, the magnitudes need not be the same. The cosine similarity is given by the following equation:

$$cos\text{-}sim(D; D^0) = \frac{D\ D^\theta}{jjDjj\ jjD^\theta jj}$$

In [12], the model was evaluated in an offline context using holdout methods by selecting randomly a number of ratings to use for prediction, and evaluating the model using the remaining ratings. Experiments showed that the model performed better with a k of 10 or higher, but that setting the value higher than 10 made little difference. This is partially explained by the fact that only few users have rated more than 10 items.

5.2 Adaptive collaborative topic modelling

In [7] an approach is proposed for improving performance and alleviating cold start in the online recommendation setting, by combining ISGD with content information modelled using the so-called Adaptive Windowing based Incremental Latent Dirichlet Allocation (AWILDA) algorithm[36], and extension of LDA. The method, named CoAWILDA, is described in detail in this section.

Experiments in [7] with applying the method to the datasets ML-100K and plista, a large dataset consisting of approximately 84 million impressions of 70,000 news articles recorded over 4 weeks[15], clearly show the effectiveness of the model in improving recommendations. Interestingly, in the paper it is also argued that failing to adapt to concept drifts reduces the contribution of the topic modelling component to the point where it is outperformed by ISGD. While it is easy to see that CoAWILDA improves recommendation quality, it is not immediately clear how the model affects the cold-start problem. Furthermore, the model only leverages the text representation of the item, and does not consider additionally item attributes which may further improve recommendation quality.

5.2.1 Drift detection in topic modelling

AWILDA, proposed in [36], is a topic modelling approach that combines online LDA with ADWIN to detect and adapt to the drifts that may occur when mining a data stream. The key differences between AWILDA and other topic modelling algorithms that consider concept drift is that it does not require the user to determine the scale at which drifts occur and has been adapted to an online setting.

As argued in [7], a number of methods have been proposed for introducing a temporal aspect into the LDA model[37][38][39], but require users to arbitrarily choose a size for the time slices used, which is problematic, as drifts may occur in a smaller or larger window than that selected. [7] argues that there are methods which do not depend on a speci□c time slice size, such as [40], but that these require the entire set of data to be known for training, and thus are not suitable for the online stream setting.

The idea of AWILDA is to process documents with online LDA as they become available, while using ADWIN to whether a drift has occurred at any given step. When a drift occurs, the model is retrained on the second subwindow selected by ADWIN.

AWILDA splits the tasks of topic modelling and drift detection into two different tasks which use each their instance of LDA. For modelling the documents the instance LDA_m is used. After the initial training, this instance is updated with each new observation. For the task of drift detection, the instance LDA_d is used. The values monitored by ADWIN are the likelihood values of the observed documents given model LDA_d. This instance is trained initially, but unlike LDA_m is not updated with new observations; only when a drift is detected is the model retrained with the window selected by ADWIN.

```
1  For each received document w, compute likelihood L = p(wjLDA_d).
2  Process L with ADWIN. If ADWIN detects a drift from the W = W_0  W_1:
3      Retrain LDA_m based on the documents in W_1.
4      Retrain LDA_d based on the documents in W_1.
5  Update LDA_m with document w.
```

Listing 5.1: The AWILDA algorithm

Listing 5.1 shows the AWILDA algorithm. At each step, prior to updating LDA_m the likelihood of the observed document w is computed using LDA_d. If a drift occurs, both LDA_m and LDA_d are retrained on the second subwindow W_1 selected by ADWIN. To "retrain" a model, we reset the value for time t (of the online LDA algorithm presented in Listing 4.2) and process each document in window W_1[1]. Resetting t means that the contribution from the documents in W_1 is weighted higher. This leads to a better modelling of the post-drift distribution. Finally, LDA_m is updated with the observed document w.

5.2.2 The CoAWILDA algorithm

CoAWILDA[7] can be seen as an extension of the model proposed in [36] to the problem of providing recommendations. The approach combines ISGD with the content information inferred with AWILDA. Speci□cally, the topics inferred by AWILDA for a document i are embedded in an item factor vector $_i$ for that item, resulting in the vector q_i used for prediction. The algorithm takes as input a set of observations consisting of both interactions and new items.

Interactions in CoAWILDA algorithm take the form of $< u; i >$, where u is the ID of the user that interacted with an item with ID i. New items take the form $< i; doc_i >$, where doc_i is a document representing i processable by ADWIN. The set of observations is the data stream and hence is sorted according to the time of the observation: for items the time the item was added and for interactions the time the interaction happened. In the case of the ML-100K dataset, items are "added" according to their release data, while interactions happen each time an item is rated.

In [7] before applying the algorithm, documents from the datasets are preprocessed by removing stop words and words that only occur once. The words are □nally stemmed. The preprocessing results in a smaller vocabulary.

[1] While this is not immediately clear from the description of AWILDA in [36], a reply from the authors in a personal correspondance clarify this.

```
1  for observation o 2 O do
2      if o is a new item < i; doc_i > then
3          _i    AWILDA(doc_i)
4          _i    R^K
5          _i    N(0; _i)
6          q_i   _i + _i
7      end if
8      if o is an interaction < u; i >:
9          if this is the first interaction for user u then
10             p_u   R^K
11             p_u   N(0; _u)
12         end if
13         e_ui = 1   q_i^| p_u
14         p_u   p_u + (e_ui q_i   _u p_u)
15          _i   _i + (e_ui p_u   _i _i)
16         q_i   _i + _i
17     end if
18 end for
```

Listing 5.2: The CoAWILDA algorithm

In Figure 5.2, the algorithm for CoAWILDA can be seen. The algorithm iterates over each observation o from the sorted set of observations O. For each new item of the form $< i; doc_i >$, the doc_i is processed according to the AWILDA algorithm described in Section 5.2.1, and the resulting mixture of topics is stored in vector $_i$. The item factor vector $_i$ has its values drawn from the multivariate normal distribution. Contrary to the ISGD algorithm in Listing 4.5, the covariance matrix is the identity matrix scaled by $_i$, rather than the constant 0.1. The same applies when new user factor vectors are initialised, but with $_u$. Vector q_i is then assigned the sum of $_i$ and $_i$.

If o is a new interaction of the form $< u; i >$, the first algorithm checks if this is the first interaction for user u by checking if vector p_u exists. If not, p_u is initialised. Then, parameters p_u and $_i$ are updated in the gradient step on lines 13-15. Finally, q_i is reassigned with the sum of $_i$ and the updated $_i$. Like ISGD, CoAWILDA is initially trained until convergence on a set of observations.

Part III

Development

6. Attribute-enhanced collaborative topic modelling

In the previous part the 01Intern and ML-100K datasets were analysed. A number of methods were presented for providing recommendations and potentially alleviating some of the problems identified in the dataset analysis. We saw how in [7] a successful blend between content-based filtering and collaborative filtering is achieved with the proposed model CoAWILDA.

In this part, a recommender system is designed for the 01Intern dataset based on the findings from the previous part. We argue for the choice of methods and present the designed model along with the implementation details. Finally, we test how the the various components perform for different tasks: topic modelling, item recommendation and alleviating cold start.

6.1 Unstructured text modelling

In the Section 2.1.4, the attributes of 01Intern dataset were presented. The unstructured text attributes account for approximately half of the attributes for jobs. To leverage the available item information, a way of modelling the unstructured text attributes is needed. In Section 5.1.2, we saw how text attributes can be represented in the vector space model with each term weighted by the TF-IDF scheme. In Section 4.1, we saw how different types of documents, including text documents, can be represented as a mixture of topics using LDA, and in Section 5.2.1 how AWILDA improves LDA for data streams through the detection of and adaption to drifts. In this section the advantages and drawbacks of each approach are evaluated with the purpose of finding the approach most suitable for recommendation for the 01Intern dataset.

6.1.1 TF-IDF

TF-IDF is a simple, yet effective approach, the main strength of which is being able to characterise terms that are discriminative for documents. Through this, it is able to find the revelant documents given a query[35]. With regards to comparing documents, TF-IDF is expected to assign a high similarity score to pairs of documents that contain rare topics.

In [21], the main drawbacks of TF-IDF are described as “providing a relatively small amount of reduction in description length and revealing little in the way of inter- or intradocument statistical structure.”

In Section 5.1.2, a method is described for comparing two documents using the cosine similarity of the TF-IDF-weighted term vectors. This results in a similarity score between any two items. To use this similarity score in the recommendation setting, a model is needed that incorporates item information by pairwise similarity, such as FVA.

However, using an approach such as Latent Semantic Indexing (LSI)[41], in which the *term-document* matrix is decomposed using SVD, the compressed representations of documents can be embedded into the item factor vectors, as in the CoAWILDA algorithm from in [7]. This approach is not suited for the online stream setting, as the full term-document matrix is not known at the beginning of the stream.

6.1.2 Topic modelling

Latent Dirichlet Allocation is useful in that it better captures the generative probabilistic semantics[21] than for example TF-IDF and is adaptable for online settings. In [21], the modularity and extensibility of LDA is noted - as a probabilistic model it may be used directly as part of another model.

The dimensionality reduction is another notable advantage of LDA. The topics mixtures are a much smaller representation than the original documents, and as we saw in the CoAWILDA algorithm, the topic mixtures can be embedded directly into the item factor vectors in matrix factorisation.

As noted in [21], one of the disadvantages of LDA is that for n-gram terms, because of the bag-of-words approach, LDA might assign different topics to each element of the n-gram, when in reality they belong to the same topic.

6.1.3 Choice of model

The powerful representativeness of LDA combined with its adaptability to the online setting make a reasonable choice for modelling the textual attributes of items of the 01Intern dataset. On the other hand, using TF-IDF with the cosine similarity is less attractive due to it requiring a model that incorporates information by pairwise similarity. Futhermore, the effects on recommendation quality of using CoAWILDA for a number of datasets are documented in [7]. Since CoAWILDA is a simple addition to MF, it is also highly extensible: in [3] a number of extensions to the basic MF model shown in Section 4.3 are presented. Therefore, the approach from the CoAWILDA model is chosen as the component used to model the textual attributes.

6.2 Incorporating additional item attributes

In Section 4.5 two methods for incorporating item attributes into MF, AE and FVA were described. Some of the advantages and drawbacks of each method were presented.

AE has the attractive property of implicitly weighting different attributes. In the method, a higher magnitude for an item attribute factor vector means that the attribute is weighted higher by users. However, a disadvantage of AE is that the attributes modelled require a □nite set of values. Continuous values need to be assigned to different intervals, each of which is represented by an attribute factor vector. The size of the intervals are not re□ected in the model and thus need to be selected arbitrarily. FVA does not suffer from this problem, as the similarity of the continuous attributes for a pair of items can be computed using a suitable function. The main drawback of FVA is that attribute weighting is not an explicit part of the model, but depends on the similarity measure being used.

Due to the implicit weighting of attributes and the fact that there are few non-categorical attributes in the 01Intern dataset, the potential of the FVA method cannot be fully realised. In the light of this, AE is the more attractive approach for including information from additional item attributes. Because AE is a simple extension of MF, it can be combined directly with CoAWILDA.

6.3 A combined model

In this section a model is proposed that combines CoAWILDA and the acAE described in Section 4.5.1. The idea is that by combining and appropriately weighting the textual and structured attributes, the model has more information available about the content of an item than when using CoAWILDA. The method is named *CoAWILDA+* to reflect its roots in the CoAWILDA algorithm. First, by combining the two methods, ratings are predicted as follows:

$$\hat{r}_{ui} = [\ _i + \ _i + \sum_{a \in A(i)} y_a]^| p_u \tag{6.1}$$

In Equation 6.1, the rating is predicted as the product of the user factor vector and the sum of the topic mixture $_i$, the item factor vector $_i$ and attributes vectors y_a for each $a \in A(i)$. The following learning objective is proposed, using the updated rating prediction function and regularising the attribute factor vectors proportionally to $_i$:

$$\begin{aligned} \min_{;y_a;p} \sum_{(u;i)2} (1 \quad [\ _i + \ _i + \sum_{a2A(i)} y_a]^| p_u)^2 + \ _i jj \ _i jj^2 \\ + \ _i \sum_{a2A(i)} jjy_a jj^2 + \ _u jjp_u jj^2 \end{aligned} \tag{6.2}$$

Let us define the learning objective for a single observations as

$$\begin{aligned} L \ , \ (1 \quad [\ _i + \ _i + \sum_{a2A(i)} y_a]^| p_u)^2 + \ _i jj \ _i jj^2 \\ + \ _i \sum_{a2A(i)} jjy_a jj^2 + \ _u jjp_u jj^2 : \end{aligned} \tag{6.3}$$

For convenience, we define prediction error e_{ui} and the combined contribution from the item factor vector, attribute factor vectors, and the AWILDA topics for an item $_i$ as:

$$e_{ui} \ , \ 1 \quad \hat{r}_{ui} \tag{6.4}$$

$$_i \ , \ q_i + \sum_{a2A(i)} y_a \tag{6.5}$$

where $q_i = \ _i + \ _i$

As in Section 4.4.1, to obtain the SGD update rules for the parameters p_u, $_i$ and y_a, we compute the gradients of the learning objective with respect to each model parameter, and modify the parameters in opposite direction of the gradient, proportionally to :

$$\frac{L}{p_u} = \quad 2 \ _i e_{ui} + 2 \ _u p_u \tag{6.6}$$

As the scalars are absorbed, the following parameter update rule is obtained for p_u:

$$p_u \quad p_u + (e_{ui} \; _i \quad _u p_u) \tag{6.7}$$

We use the same approach to obtain the remaining parameter update rules:

$$\frac{L}{\;_i} = \quad 2p_u e_{ui} + 2 \; _i \; _i \tag{6.8}$$

$$_i \quad _i + (e_{ui} p_u \quad _i \; _i) \tag{6.9}$$

$$\frac{L}{y_a} = \quad 2p_u e_{ui} + 2 \; _i y_a; \text{for } y_a ja \; 2 \; A(i) \tag{6.10}$$

$$y_a \quad y_a + (e_{ui} p_u \quad _i y_a); \text{for } y_a ja \; 2 \; A(i) \tag{6.11}$$

Having obtained the parameter update rules for learning objective L, the CoAWILDA+ algorithm is proposed. Listing 6.1 shows the CoAWILDA+ algorithm in full. Similarly to CoAWILDA, the algorithm takes as input a stream of observations O. Like in CoAWILDA, the set of observations is sorted by the time of observation. The main differences are the updated prediction function $\hat{r}_{ui}$ and that the algorithm uses and maintains a set of factor vectors for attributes y. Items are processed as in CoAWILDA, however, on lines 6-11 factor vectors for the attributes in i are initialised, if it is the first time they are seen. When a new interaction is processed, p_u and $_i$ are updated according to the update rules in Equation 6.7 and Equation 6.9, respectively. Additionally, for each attribute a possessed by i, attribute factor vector y_a is updated according to the update rule in Equation 6.11, as seen on lines 22-24.

```
for observation o 2 O do
    if o is a new item < i; doc_i > then
        _i   AWILDA(doc_i)
        _i   R^K
        _i   N(0; _i)
        for a 2 A(i) do
            if this is the first item with attribute a then
                y_a   R^K
                y_a   N(0; _i)
            end if
        end for
        q_i   _i + _i
    end if
    if o is an interaction < u; i >:
        if this is the first interaction for user u then
            p_u   R^K
            p_u   N(0; _u)
        end if
        e_ui = 1   ^_ui
        p_u   p_u + (e_ui _i   _u p_u)
        _i   _i + (e_ui p_u   _i _i)
        for a 2 A(i) do
            y_a   y_a + (e_ui p_u   _i y_a)
        end for
        q_i   _i + _i
    end if
end for
```

Listing 6.1: The CoAWILDA+ algorithm

6.4 Implementation details

In the previous section, a model CoAWILDA+, which combines CoAWILDA with AE, was described. The author has implemented the model in Scala using a number of existing Java libraries. For the sake of subsequent evaluations of the model being reproducible, the implementation details of the implemented model are described in this section.

6.4.1 Interactions

In Section 2.1.1 it has been described that feedback in the 01Intern dataset are of three different types: views, favourites and applications. CoAWILDA+, like ISGD, used postive-only feedback and does not distinguish between different types of feedback. Hence, for the task of recommendation the three feedback channels are considered a single source of feedback, *interactions*.

In some cases users have interacted with a specific item in more than one way. For example, a user might first add an item to their list of favourites and later apply for that same job. In the case where an interaction $< u; i >$ is present in multiple channels, only the first instance of $< u; i >$ to appear in the data stream is considered. The interactions from the other channels are not taken into account.

6.4.2 Attributes used in the implementation

In this section the item attributes used for recommendation are described. A subset of the full set of attributes are used for recommendation. The attributes are characterised as either *textual descriptions* or *tags*.

doc_i, which is used for the topic modelling, is the concatenation of a number of the textual descriptions deemed most relevant. The unstructured text attributes used are *catch copy*, *appeal*, *entrusted job detail* and *quali cation*. The relevance of the selected textual descriptions is subject to tests.

For the tags, which are the attributes modelled by the AE component of the model, a number of structured attributes are used:

- Job type
- Business type
- Prefecture
- City
- Sticking conditions

While more of the attributes described in Section may prove useful in a recommendation setting, these are not considered due to time constraints.

6.4.3 Document preprocessing

As in [7], the documents of the 01Intern and ML-100K datasets are preprocessed, before they are used for topic modelling. The preprocessing consists of converting the text description for each document into a bag of words, consisting of useful terms. For example, punctuation marks are of little relevance to the topic of a document and are stripped stripped from the documents. For nding the appropriate terms for the documents, a different approach is used for each dataset. This is mainly prompted by the different languages of the datasets: Japanese for the 01Intern dataset and English for ML-100K.

For the *Japanese documents*, Kuromoji [42], a Japanese morphological analyser, is used to lemmatise words and perform *part-of-speech* tagging. Part-of-speech tagging here means inferring the word category to which each word in the document belongs. This is used to determine which words are useful and should be used as terms in the bag of words.

Upon processing a document using Kuromoji, it is represented list of tokens, where each token provides meaningful insights about the words, such as the base form and part-of-speech tagging. The base form of each word are the terms that may be used in a bag of words, although for some words, a base form is not available - in this case the original word is used instead. What determines if a word is used in the bag of words is the part-of-speech tagging. Words that are characterised by the following are *not* included in the bag of words for each document:

- â " (symbols)
- # U (auxiliary verbs)

- # U (verbs) labelled " r (not independent)
- U (particles)

For the *English documents*, words have been stemmed using the `PorterStemmer` from the Apache OpenNLP library[43].

For both datasets, when applicable, terms have been converted to lower case. Furthermore, stop words are removed from documents in each language by using an English list of stop words and a Japanese list of stop words. The complete lists of stop words can be found in Appendix B.

6.4.4 Existing implementations used

As mentioned in the previous section, the Apache OpenNLP and Kuromoji libraries are used for the tasks of lemmatisation and stemming. Additional libraries are used for the implementation of the AWILDA component. An implementation of ADWIN found in the MOA library[44] is used. It is an implementation of the more efficient ADWIN2 algorithm described in Section 4.2.2. For the topic modelling, a direct implementation of online LDA as described in [22], jolda[1], is used. The implementation has been modified slightly in order to:

- allow for t to be reset when a drift is detected by ADWIN
- be able to approximate the likelihood for a set of documents, without also updating the model

These modifications are necessary for the implementation to be compatible with the AWILDA algorithm, as the likelihood of a new document must be approximated *before* the model is updated, so that it is possible to adapt to potential drifts before updating the model with the document.

6.4.5 Initial training of the model

Before the model is able to meaningfully infer factors for the users and items that arrive in the stream of observations, it must initially be trained on a set of the observations. During this initial training phase, the new items in the set of observations are used by AWILDA to initially train models LDA_m and LDA_d, using the batch variational inference algorithm. Following this, the CoAWILDA+ model is trained on the interactions in the set of observations until it converges or reaches a max number of iterations. The max number of iterations is set to 10,000. The criteria for convergence is when the relative change in average error is less than 10^{-8}.

[1] GitHub repository: https://github.com/miberk/jolda

7. Evaluation

In this chapter the developed recommender system is evaluated. In particular, a separate evaluation of topic modelling component is carried out, before the recommender system is evaluated in an online stream setting using two different evaluation measures. Finally, the ability of the recommender system to alleviate cold start is assessed.

7.1 Topic modelling evaluation

In this section the implementation of AWILDA is evaluated with a focus on the 01Intern dataset. The purpose is to establish an overview of the effectiveness of the algorithm: how well it is able to model the data and whether the inferred topics bring out characteristics of the documents. Additionally, we analyse how drift detection affects the topic mixtures for documents and the relationship between different documents. The number of topics k is set to 10. Preliminary experiments have shown similar patterns appear when choosing different values for k such as 20 or 50. The hyperparameters of the models are determined using grid search over the parameter space and optimal values are listed where relevant.

7.1.1 Perplexity of topic models

In Section 4.1.3, perplexity is presented as a measure that can be used to evaluate the the ability of a model to generalise documents. Formally, the perplexity of a set of unseen documents D_{test} is de□ned as

$$perplexity(D_{test}) = \exp\{ -\frac{\sum_{d=1}^{M} \log p(w_d)}{\sum_{d=1}^{M} N_d} \} \tag{7.1}$$

where M is the number of documents in D_{test} and the probability of w_d is approximated according to the variational inference described in Section 4.1.1.

To measure how well AWILDA models documents, we measure its performance in terms of perplexity. To assess the contribution of the drift detection component that distinguishes AWILDA from online LDA, both topic models are evaluated. To obtain suitable values for the model hyperparameters, a grid search of the parameter space is performed. For the grid search, success is measured by a lower average perplexity. Note that due to time constraints, the grid search is not thorough and more optimal values may exist. Since AWILDA is an extension of LDA, we □rst □nd values for the hyperparameters of LDA. Performing the grid search, we obtain values = 1024 and = 1. Following this we adjust to further optimise the average perplexity, and obtain = 0:01.

Following the experimental protocol in [7], we measure the performance of the topic model by □rst approximating the likelihood and computing the perplexity of a document and then processing it with the model, repeating this process for each document in the stream of documents. In order to smooth the scores, they are reported as a moving average of a sliding window of 200 observations. To initialise the topic model, it is initially trained on the □rst 20% of the document stream. Following this, the documents in the remaining 80% of the document stream are evaluated one by one.

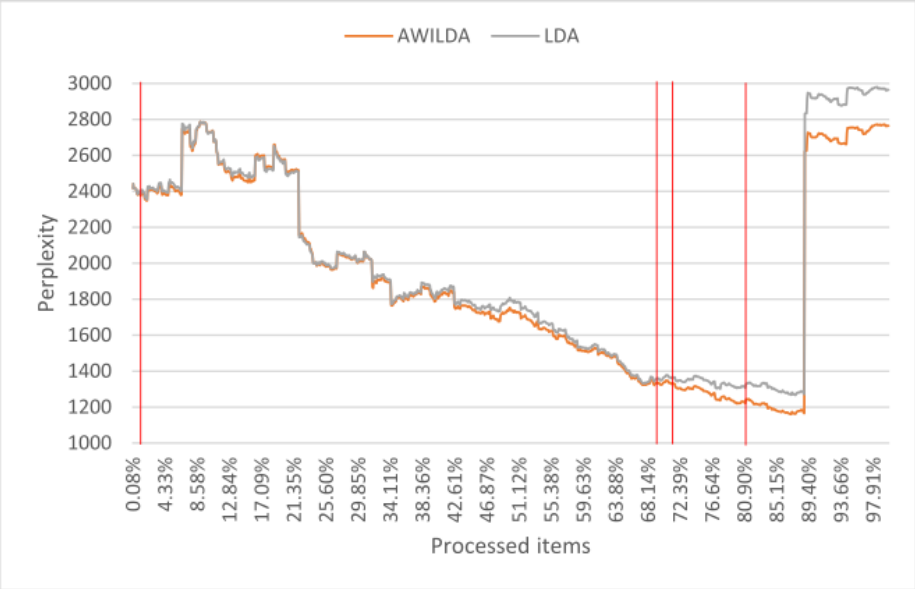


Figure 7.1: Comparison of the performance of AWILDA and online LDA on the 01Intern dataset, measured as the average perplexity of a sliding window. Each red line indicates a drift detected by AWILDA.

In Figure 7.1, a comparison of the reported average perplexity values of the sliding window for AWILDA and LDA on the 01Intern dataset is shown. In the □gure, a vertical red line indicates that a drift was detected by AWILDA at this time. On the y-axis is the reported perplexity of the window and on the x-axis is the position of the current value in the array of reported values in percentage . A drift is detected at approximately 200 documents into the stream (not zero, as the □rst reported value required 200 documents to have been processed). Following this, the reported values for AWILDA are lower than those for LDA, although not by much. It is not until around 45% into the values that the differences start to become more noticeable. At the 68% mark, another drift is detected, followed by two more. With each of these drifts the difference in performance between AWILDA and LDA grows bigger.

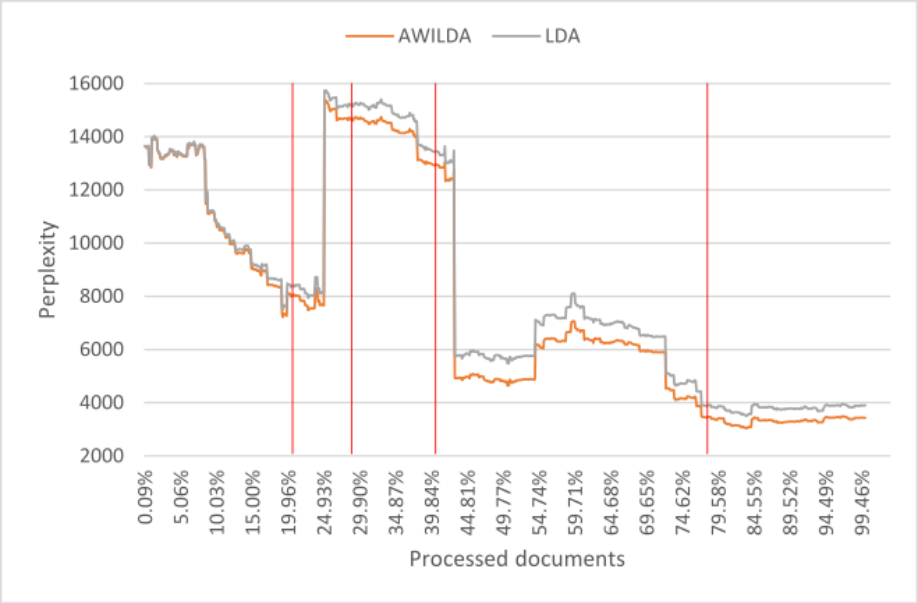


Figure 7.2: Comparison of the performance of AWILDA and online LDA on the ML dataset.

In Figure 7.2, AWILDA and LDA are compared for the ML-100K dataset. Here, no drift is detected until around the 20% mark, but here the performance of AWILDA is improved noticeably over that of LDA. Three more drifts are detected and with each, AWILDA continues to outperform LDA.

Comparing the performance of the topic models for the two datasets, the contribution of the drift detection from AWILDA becomes apparent at a much earlier time for the ML-100K dataset. From the time where the first drift is detected, AWILDA consistently performs better than LDA. For the 01Intern the difference is negligible at times. Additionally, the difference in performance between AWILDA and LDA is generally greater for the ML-100K dataset than for the 01Intern dataset. However, it is clear that drift detection improves ability to model the documents for both datasets.

7.1.2 Document topics

In the previous section the ability of AWILDA and LDA to model the documents was evaluated. While the evaluation revealed differences in the performance of the models, it did not provide any insights into the topics inferred. In this section, we take a closer look at the inferred topics and how documents are represented in terms of these topics.

To gain insights into the topics and documents, we follow the evaluation setting described in the previous section and capture the state of the topic model and the documents at certain interesting points in time. First, we assess the topics after the initial learning of the model. In an experiment in Section 7.1.4, we compare the state of the model right before and after a drift occurs.

The values used for the hyperparameters are similarly the same as in the previous experiment.

Topic 1	enterprise ! 0.0188, human ! 0.0146, work ! 0.0133, can ! 0.0133, life ! 0.012, growth ! 0.0103, of□ce ! 0.0096, and ! 0.0095, company ! 0.0085, experience ! 0.008,
Topic 2	design ! 0.0301, web* ! 0.0288, can ! 0.015, site ! 0.0132, service ! 0.0113, designer ! 0.0109, job ! 0.0092, creation ! 0.0089, student ! 0.0083, development ! 0.0079,
Topic 3	human ! 0.0124, life ! 0.0106, enterprise ! 0.0096, duties ! 0.009, service ! 0.0087, go ! 0.0086, can ! 0.0079, planning ! 0.0075, project ! 0.0074, job ! 0.0072, -like ! 0.007,
Topic 4	sales ! 0.04, -like ! 0.0123, can ! 0.0109, enterprise ! 0.0098, go ! 0.0094, experience ! 0.0085, service ! 0.0084, proposal ! 0.0082, shop ! 0.0081, management ! 0.0074,
Topic 5	human ! 0.0192, enterprise ! 0.0156, industry ! 0.0126, experience ! 0.0112, job ! 0.0099, can ! 0.0093, life ! 0.0091, company ! 0.0081, duties ! 0.0081, person ! 0.0073,
Topic 6	development ! 0.0371, experience ! 0.0212, engineer ! 0.0146, project ! 0.0146, service ! 0.0133, app ! 0.0086, life ! 0.0085, skill ! 0.0081, web* ! 0.0081, human ! 0.0076,
Topic 7	project ! 0.0155, enterprise ! 0.0121, human ! 0.0112, planning ! 0.0103, site ! 0.0097, can ! 0.0089, business ! 0.0087, administration ! 0.0079, marketing ! 0.0078, duties ! 0.0076,
Topic 8	job ! 0.0118, experience ! 0.0107, human ! 0.0103, creation ! 0.0099, can ! 0.0097, duties ! 0.0088, environment ! 0.0073, life ! 0.0073, -like ! 0.0069, game ! 0.0068,
Topic 9	service ! 0.0217, web* ! 0.0139, can ! 0.0139, skill ! 0.0109, job ! 0.0092, article ! 0.0088, user ! 0.0085, study ! 0.0079, experience ! 0.0079, company ! 0.0077,
Topic 10	service ! 0.0207, web* ! 0.0111, can ! 0.0091, human ! 0.0089, -like ! 0.0082, media ! 0.0077, education ! 0.0076, experience ! 0.0074, project ! 0.0066, duties ! 0.0065,

Table 7.1: The 10 most popular terms for each topic inferred using LDA

Table 7.1 shows the 10 most popular terms for each of the 10 topics inferred using LDA, sorted by their probability of being generated. The terms are listed in the format w ! p, where w is the term and p is the probability of generating w from that topic. As the documents in 01Intern dataset are in Japanese, the terms have been translated into English. A sample of the original Japanese terms for the topics can be found in Appendix A. If a term ends with a "*", it means that the term has not been translated from its original form in the Japanese document. This is the case with the term "web", which is used as it appears in many of the documents.

It is clear that the top 10 lists for the topics have a lot of terms in common - for example, the terms "job" and "web*" appear in □ve of the 10 topics. This is explained by the fact that due to the domain of the dataset, most doc-

Topic 4 "Business"	Topic 6 "Engineering"	Topic 7 "Global"	Topic 8 "Entertainment"	Topic 10 "Steering"
sales	development	project	reception	service
student	experience	enterprise	game	goal
growth	engineer	human	human	user
venture	project	planning	environment	vision
personnel	skill	site	sustenance	year
ability	app	Japan	more than	media
real estate	technology	business	support	education
proposal	university student	administration	cast	new
shop	make	marketing	mobile	woman
management	system	overseas	domain	day

Table 7.2: Some of the most popular terms for five of the LDA topics. Terms that poorly discriminate the topics have been removed.

uments, no matter their topic mixture, contain these terms. Aside from this, some topics have clear distinguishing features and are understandable to humans: Topic 2 is about graphic design[1] and Topic 6 is about engineering.

To see how documents can be represented as a mixture of different topics, we investigate how the different terms in a document are generated by a specific topic. That a term is "generated by a specific topic" here means that this topic has the highest probability of generating the topic.

We turn our attention to a specific document, titled "Recommended for science students! Internship where you can learn rapid big data processing skill", which we will from here on out refer to as Job A. In Table 7.2, a selection of terms from the five topics most represented in the topic mixture of Job A are shown. The terms shown are generally some of the most popular terms for each topic, but terms that poorly discriminate the topics have been left out. Each of the five topics is summarised with a title, that is common for most of the words in the topic. Additionally, each topic is assigned a colour: the purpose of this is to be able to clearly highlight using the colour by which topic each term in Job A is generated.

Documents in the 01Intern dataset are rather large, due to being composed of several text attributes. For this reason, only a segment of the document for Job A is provided. In Figure 7.3, a sample of the document for Job A is shown, with relevant terms highlighted by the colour of the topic they are generated by. In a few cases, multiple Japanese terms have been translated into a single term in English; in these cases, the term has the colour for both topics, as is the case with the term "gain". Here, the original terms are generated by both topic 4 and topic 10.

Consider another job, titled "2 days a week OK! Learn Java as a server-side engineer!" which we refer to as Job B. This job has the topic mixture shown in Figure 7.4. The figure shows that the most prominent topic in Job B is Topic

[1] If one ventures beyond the first 10 terms, "photoshop" and "illustrator" are some of the first terms to appear.

This is an engineering job where you can gain insights about server side development and databases. You will be developing a web service, *Zealup*, designed for companies. Zealup is a goal management tool which supervises if individual effort properly leads to the overall goal of the company. For example, when developing a web service, diverse roles such as development, marketing and support are necessary. The innovative service tries to achieve an overall goal of organisation, which is completing the service, by sharing how much achievements by individuals with different roles affect completion of the service among everyone.

Figure 7.3: Job description for Job A, with terms generated by each topic highlighted

6: Engineering. Topics 1, 2, 4, 8 and 9 make up the rest of the significant bit of the topic mixture, each with approximately the same degree of presence. When considering the sample of Job B in Figure 7.5, which corresponds to the qualification attribute (the required qualifications for the job), it becomes clear why Engineering is the most represented topic for Job B. Not all of the topics that make up the significant part of the topic mixture are represented in the sample; however, they can be identified in the rest of the document.

7.1.3 Similar documents

In this section, an analysis of how the topic mixtures for different documents relate to each other. Using Job A as a starting point, a comparison is made to the three jobs with the most similar topic mixtures. The similarity of topic mixtures is measured using the Euclidean distance.

In Table 7.3, the three jobs with the most similar topic mixtures are shown. For each job, the job title, distance to Job A, and a chart illustrating the topic mixture are shown. It is immediately clear from the job titles that these jobs are related to engineering and computer science.

For all of the jobs, Topic 6 is the most dominant topic. For none of the jobs the difference in Topic 6 differs by no more than 1%. For the remaining topics, the difference is greater, and some of the jobs consist of topics that Job A does not exhibit. However, it is easy to see that even disregarding Topic 6, the jobs in the list all show similarities to Job A.

Interestingly, Job A and the most similar job Job B are significantly represented by Topic 1 and Topic 4. Both of these jobs are, in addition to engineering, related to business and the growth of companies; both Topic 1 and Topic 4 relate to economics, which may explain why Job A is more similar to Job B with regards to Topic 1 and Topic 4 than to the 2nd and 3rd most similar jobs, which focus more on the development of games and services, respectively. In addition, Job A and Job B are both related to big data analysis. This reveals that the topic model is able to capture noteworthy aspects of the documents with as few as 10 topics.

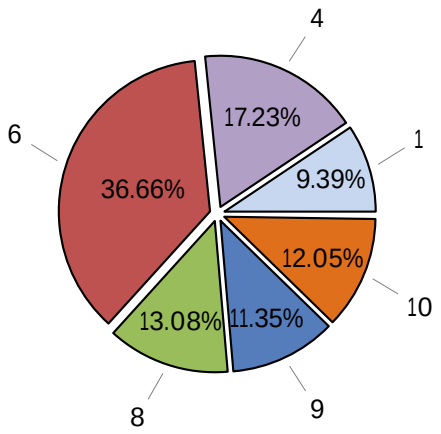
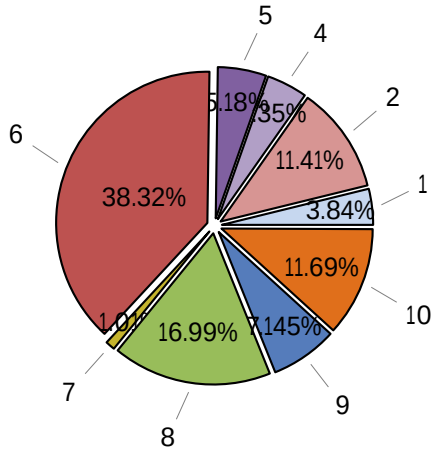
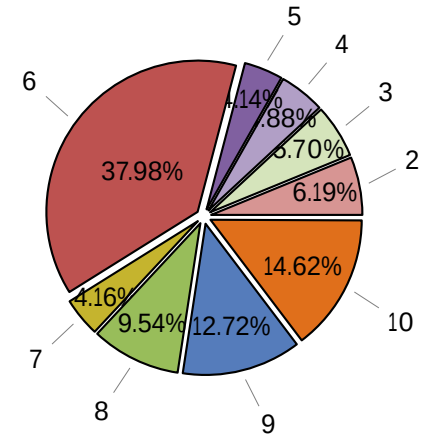
Distance	Job title	Topic mixture
0.1784	Recommended for science students! Internship where you can learn rapid big data processing skill	
0.1803	Development of various popular games!	
0.2166	Engage in the development of various popular services	

Table 7.3: The three jobs with topic mixtures most similar to Job A

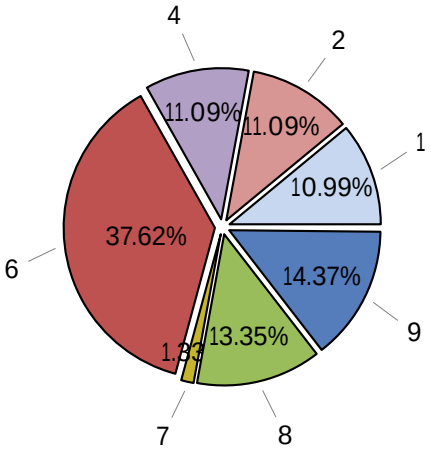


Figure 7.4: Topic mixture for Job A

Experience developing web applications (sample apps etc. also OK)
Experience using the Linux OS Basic understanding of algorithms and data structures
Development experience using the C language/Ruby on Rails

Figure 7.5: Required qualifications for Job B, with terms generated by each topic highlighted

7.1.4 Effects of drift adaption

In this final section of the evaluation of the topic modelling component of CoAWILDA+, the effects of drift adaption are analysed. The purpose is to establish an overview of how drift adaption affects which terms are generated by which topics and how the relationships between related documents are influenced. Specifically, we analyse the effects of the first drift that occurs when following the protocol described in Section 7.1.1.

In Figure 7.6 the 10 terms most likely to be generated by Topic 4 before and after the occurred as shown. The top of the figure shows the 10 terms before the drift occurs, while the bottom shows the terms after the drift has occurred.

The drift adaption has a noticeable effect on the terms generated by Topic 4. A change occurs in the probability the terms have of being generated by the topic, although not by much. This results in a reordering of some of the terms in the list, however, the 10 most popular terms remain the same, as no new terms appear in the list. This is as one would expect; the topics do not change radically from what was inferred during the initial training of the model, even when a drift occurs.

However, if we consider for example the top 500 terms, the differences become more noticeable, as the relative change in probability may be bigger for words with a lower probability of being generated. We see a similar pattern

sales ! 0.0378, -like ! 0.0119, can ! 0.0107, enterprise ! 0.0102, go ! 0.0096, experience ! 0.0091, management ! 0.0084, planning ! 0.0083, service ! 0.0078, shop ! 0.0076

#

sales ! 0.0367, -like ! 0.0117, can ! 0.0105, enterprise ! 0.0103, go ! 0.0097, experience ! 0.0096, management ! 0.0092, shop ! 0.008, planning ! 0.008, service ! 0.0076

Figure 7.6: Effect of drifts on the top 10 terms most likely to be generated from Topic 4. The top part of the figure are the terms just before the drift occurs, while the bottom part are the terms after retraining the model following the drift.

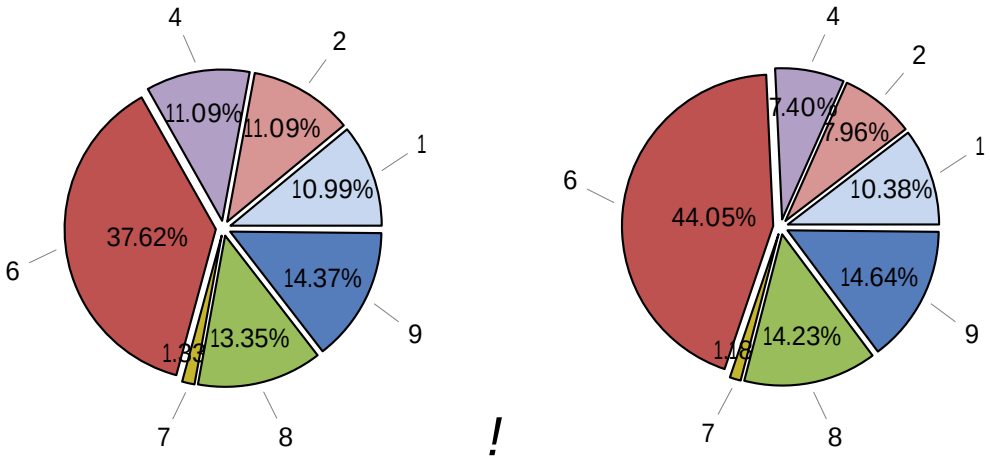


Figure 7.7: The effect of drift adaption on the topic distribution. On the left is the topic mixture for job A, on the right we see what the topic distribution would be, if we were to process the document using the post-drift LDA model.

for the other topics.

To see how drift adaption affects the topic mixtures of modelled documents and the relationship between similar documents, a simple experiment is set up in which the topic mixture of a document is inferred *before* a drift occurs, and then again for the same document *after* the drift has occurred. The purpose of the experiment is to see if the drift adaption has any significant effect on the balance between different topics and whether the most similar documents remain the same or change.

Figure 7.7 shows the topic mixture for Job A before the drift occurs (left) and after the drift has occurred (right). The drift has a noticeable effect on all of the topics: Topic 6 accounts for 6% more of the mixture after the drift. Meanwhile, Topic 2 and Topic 4 each account for approximately 3% less. The contribution of the remaining topics also shifts, but not by as much - generally by less than 1%.

To see how this shift in topic mixture affects the relationships between jobs, the jobs most similar to the topic mixture from before and after the drift are computed. Table 7.4 shows the 10 most similar jobs before the drift occurs, while Table 7.5 shows the 10 most similar jobs after drift adaption. The names in bold in Table 7.5 indicate that these jobs were also part of the 10 most similar jobs before the drift was detected. Note that only the topic mixture of Job A is recomputed after the drift occurs, not of any of the other jobs. This is because the purpose is to find out how jobs added after the drift relate to the jobs that have already had their topic mixtures inferred. This better reflects the CoAWILDA+ algorithm, where the topics for items are not recomputed even after a drift is detected; only the topics of jobs added after are affected by the drift adaption.

Of the 10 most popular jobs after the drift adaption, seven were also part of pre-drift 10 most similar jobs. However, the distance between Job A and the jobs in the list changes, causing a reordering of the jobs in the list. One of the jobs for which the similarity changes the most is the one titled "We support human resource education for big companies. System development engineers wanted!". The distance between this job and Job A changes from 0.2255 before the drift to 0.1828 after. In conclusion, the drift adaption has an effect on the relationship between the topic mixtures for jobs.

Distance	Job title
0.1784	Recommended for science students! Internship which you can learn rapid big data processing skill
0.1803	Development of various popular games!
0.2166	Engage in the development of various popular services
0.2255	We support human resource education for big companies. System development engineers wanted!
0.2384	Engineers wanted to make website with HTML and CSS
0.246	Average age of 24! Let's learn system development in a stylish office in Ebisu!
0.2483	No experience needed. Internship where you can learn the basics from an engineer who previously worked at Rakuten
0.2511	Learn PHP and Javascript in practice, if you have a basic knowledge!
0.2534	Statistic analysis/big data application. Engineers needed for software development!
0.2607	Spread popular products all over the world. Internship for creation of EC sites

Table 7.4: The 10 most similar jobs to job A before a drift is detected.

7.2 Recommender system evaluation

In this section, a number of recommender systems are evaluated. The stream of observations for each document is used for prequential evaluation, using the evaluation protocol described in [7]. Unfortunately, as the author has not been able to acquire suitable hyperparameters for AE and CoAWILDA+ due

Distance	Job title
0.1797	**Development of various popular games!**
0.1828	**We support human resource education for big companies. System development engineers wanted!**
0.1930	**Recommended for science students! Internship which you can learn rapid big data processing skill**
0.2097	**Average age of 24! Let's learn system development in a stylish office in Ebisu!**
0.212	**Engage in the development of various popular services**
0.2317	Why not try developing news apps or TV program apps?
0.2324	**Learn PHP and Javascript in practice, if you have a basic knowledge!**
0.2416	Experience programming in practice widely from HTML to server side!
0.26	**Statistic analysis/big data application. Engineers needed for software development!**
0.2796	Internship during where you develop portal sites run by our own company. You can learn from CTO who previously worked at Rakuten

Table 7.5: The 10 most similar jobs to job A after drift adaption. The job titles in bold indicate that the job was also in the top 10 before the drift was detected.

to time constraints, these models are left out. Instead the focus of the evaluation is on CoAWILDA, which is evaluated alongside several baselines. A set of different evaluation measures are used to highlight different aspects of the performance of the recommender systems.

7.2.1 Evaluation protocol

In [45], an evaluation protocol is proposed for evaluating recommender systems in an online setting. This is the protocol used for evaluating CoAWILDA in [7]. The protocol is designed for recommender systems that require an initial training phase. The protocol consists of three distinct phases:

1. **Batch train**: in this phase, the model is initialised using a subset consisting of a number of the □rst observations from the data stream
2. **Stream train**: this phase ensures a smooth transition between the initial training of the model and the online test phase
3. **Stream test and train**: in this □nal phase the model is evaluated using a prequential approach, in which each observation is □rst used to test the model, and subsequently used to train the model

Since the goal is to test the performance of the recommender system in the online setting, the online evaluation should not take place immediately after the batch training of the model. The second phase exists to avoid a temporal gap in the data, which is undesirable as temporal aspects are relevant to some models[45].

The datasets are partitioned into three subsets to be used by the different phases of the evaluation protocol. Following the setting in [7], both datasets are split in the following way: *batch train* uses the first 20% of observations, *stream train* uses the next 30%, and *stream test and train* uses the remaining 50% of observations.

In [45], the authors mention that a problem with recommendation using the evaluation protocol is that sometimes the model evaluated will be tasked with computing recommendations for a user for which no observations exist. In this case, the testing for this observation is skipped and the recommender system is trained on the model. The same approach is taken for this evaluation. We require at least one rating for a user before the recommender system can make predictions. An alternative approach is to provide recommendations using a just initialised user factor vector. For the task of evaluating recommender systems with an emphasis on the user cold-start problem, this is more relevant.

When evaluating a model on a dataset, prequential evaluation is used at each step in the stream test and train phase. An average of all the observations tested is kept throughout the evaluation of the model. To see how the models perform over time, the average value is reported each time another 10% of the subset has been processed.

7.2.2 Evaluation measures

The evaluation measures used are the *Discounted Cumulative Gain (DCG)* and *Recall@k*. In the prequential evaluation setting, where the set of rated items consists only of a single item, the Recall@k measure simply returns 1 if the item is present in the k first recommendations, and 0 otherwise[11]. DCG@Ni scores a list of recommendations based on the rank at which the observed item appears. If the item appears earlier in the list of recommendations, a higher score is received. DCG@Ni is given by the following formula:

$$DCG@Ni = \frac{1}{\log_2(rank(i) + 1)} \tag{7.2}$$

7.2.3 Models evaluated

For comparison, a number of models are evaluated alongside CoAWILDA. The purpose of this is to be able to establish an idea of how the model performs in general, and what its strengths and weaknesses are. The models evaluated are the following:

- **MF**: a basic matrix factorisation implementation as described in Section 4.3, with no added item or user information
- **CoAWILDA**: an implementation of CoAWILDA with the same implementation details described in Section 6.4
- **CoLDA**: Like CoAWILDA, but instead of AWILDA, pure online LDA is used
- **kNN regression**: the pure content-based filtering approach explained in Section 5.1

- **Random**: random ordering of the available items the user has not yet interacted with. The random recommender is evaluated 100 time to reduce noise in the results. The random recommender is not included in most evaluation results, as it signi□cantly distorts the charts

For the kNN regression recommender the same item attributes are used when computing the similarity of items as those used in CoAWILDA+. For the exact settings of the kNN regression recommender, refer to Appendix C.

7.2.4 01Intern

For the 01Intern, the evaluation is not based on the full dataset, but a variant that cuts off users with less than 20 ratings, as the focus is on the item cold-start problem. The cut-off is used to reduce the noise from users with only few ratings.

In the charts of the evaluation results, the values of the y-axis are the mean score for the tested observations at each point in time. The values of x-axis are what percentage of the stream test and train subset has been processed at that point.

Figure 7.8 shows the evaluation result for the 01Intern dataset measured using DCG@Ni. Surprisingly, the kNN regression recommender consistently scores highest out of all the recommender systems. The kNN recommender is closely followed by CoAWILDA and AWILDA whose performance are approximately the same. Towards the end of the stream, CoAWILDA and AWILDA have both caught up to the kNN regression recommender in terms of performance. The simple MF model performs signi□cantly worse than the models leveraging item information, consistently scoring about 0.01 less than CoAWILDA and LDA. MF, like CoAWILDA and AWILDA, improves in performance as more of the stream is processed. Worst of all is the random recommendations recommender, which scores 0.03 less than the other models. Like the kNN recommender, its performance does not improve as more of the stream is process. This is only natural, as the random recommendations model is not updated as new observations are encountered.

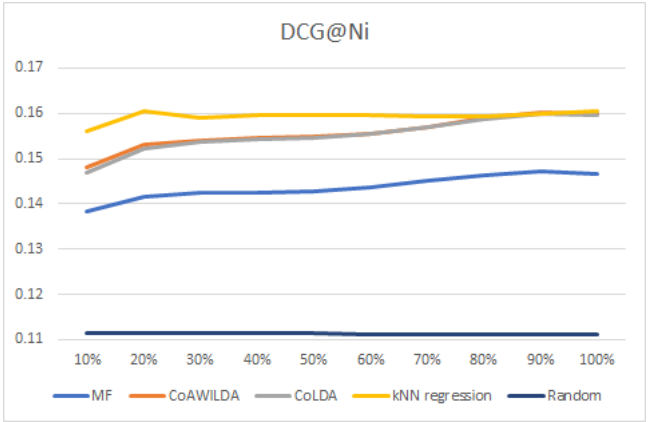


Figure 7.8: Evaluation of the recommender systems on the 01Intern dataset using Discounted Cumulative Gain @Ni.

In Figure 7.9 the Recall@k scores of the models for different values of k are shown. The values for k are 5, 10, 50 and 100. These are the same values used in [7]. The different sizes of the recommendation lists allows us to measure how the models perform when the item relevance is important in a very small set of recommendations, and when a larger set of items is considered. The performance of the models is generally the same as in Figure 7.8, with MF performing worse than the models that take into account item attributes. However, for Recall@50 and Recall@100, CoAWILDA and CoLDA start to outperform kNN regression about half way through the dataset. This highlights an interesting characteristic of the content-based kNN regression recommender: as it recommends the items most similar to the items already rated by a user, when this approach is successful the recommendations will often be ranked very high in the list. The more serendipitous hybrid recommender systems are less successful for this task, although they achieve a better overall score when k is more relaxed.

While it is possible to see that CoAWILDA outperforms AWILDA, it is not by much. This may be explained by the distribution being rather stable over time. After the initial training of the model two drifts are discovered, but since they are discovered soon after the initial training, the effect of retraining is not as strong.

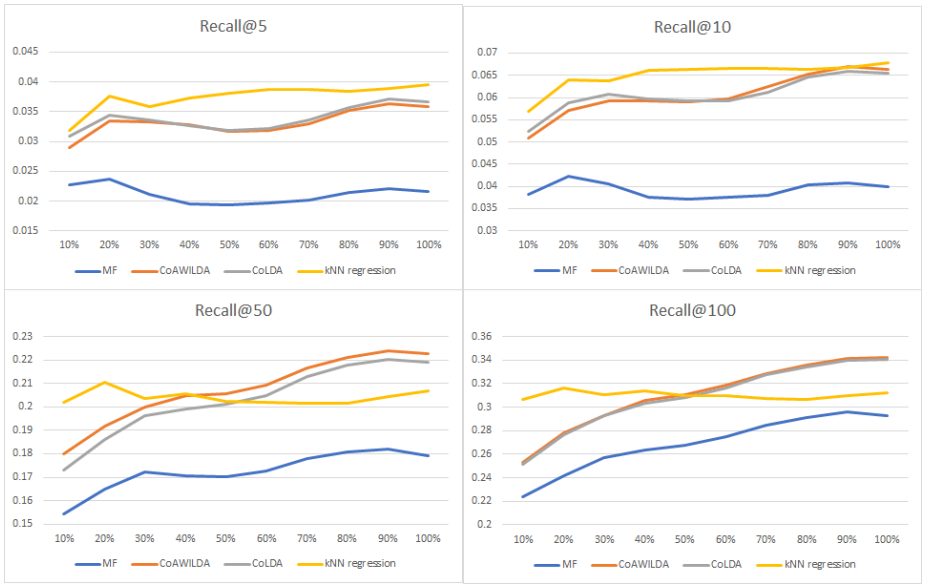


Figure 7.9: Evaluation of the recommender systems on the 01Intern dataset using Recall.

The optimal amount of regularisation used by MF, CoAWILDA and CoLDA for the 01Intern dataset is set rather low. For the three models, $_u$ and $_i$ is set 0.0005. This is very different from the regularisation parameters suggested for the ML-100K dataset in [7]. A possible explanation for this is that the ratings processed throughout the three phases are quite representative of the overall

dataset: it is capturing the important properties of the data without over□tting. If more regularisation is used the model becomes general, but this results in certain useful properties getting lost.

7.2.5 MovieLens 100K

Because 1,600 movies are in the batch train subset of the stream of observations, it unlikely that the drift detection mechanism of CoAWILDA will have an effect on recommendation quality. In a personal correspondence with the authors of [7], it was clari□ed that in the evaluation described in the paper the batch train subset accounts for approximately 1,200 of the items. It is unclear to the author what has resulted in this difference. However, to closer mimic the setup in [7], AWILDA is initialised with approximately 75% of the batch train subset (corresponding to 1200 items) and for the remaining 25% of items, AWILDA is updated with one item at a time. Using this setup, a contribution from the drift adaption may manifest itself.

For the ML-100K dataset, the kNN regression recommender is not evaluated. This is because its implementation is speci□c to the domain of 01Intern; adapting the model for the ML-100K dataset would require major refactoring.

Figure 7.10 shows the results of evaluating the models on the ML-100K dataset using DCG@Ni. The ordering of which models perform best is the same as for 01Intern dataset. The trends for CoAWILDA and AWILDA are more stable than for the 01Intern dataset, where performance increased over time. In Figure 7.11 the evaluation using the Recall measures are shown. These evaluations show similar characteristics, with CoAWILDA outperforming AWILDA. However, towards the end of the stream, their performance is approximately the same.

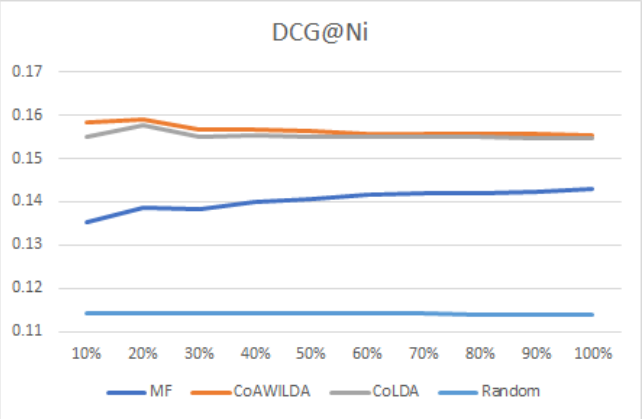


Figure 7.10: Evaluation of the recommender systems on the ML-100K dataset using Discounted Cumulative Gain @Ni.

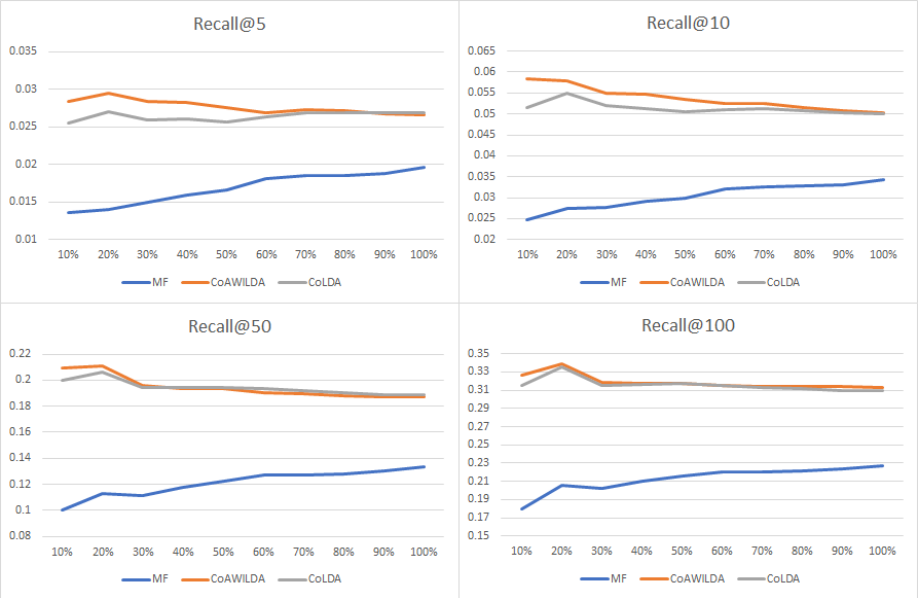


Figure 7.11: Evaluation of the recommender systems on the ML-100K dataset using Recall.

7.3 Effect on cold-start problem

For the task of highlighting the usefulness of the models in alleviating cold start, the Recall@k measure is used to evaluate speci□cally the observations in which the item has below a certain number of ratings r_{max}. For example, by setting $r_{max} = 0$, only entirely new items, for which no ratings exist, are considered. We refer to this evaluation measure as Recall(new)@k.

In Figure 7.12, the Recall(new)@10 is shown. The values of the y-axis is the ratio of total number of correct predictions across the entire stream to the maximum possible correct preditions; in other words it is the percentage of correct predictions. On the x-axis are the maximum number of ratings an item can have in order for it to be considered “new”. MF and CoAWILDA are evaluated based on the ability to provide correct predictions for items with less than or equal to 0, 5 and 10 ratings. It is clear from the Figure that CoAWILDA outperforms MF, however, it is not by much. When predicting completely new items, CoAWILDA has a success rate of about 0.9%.

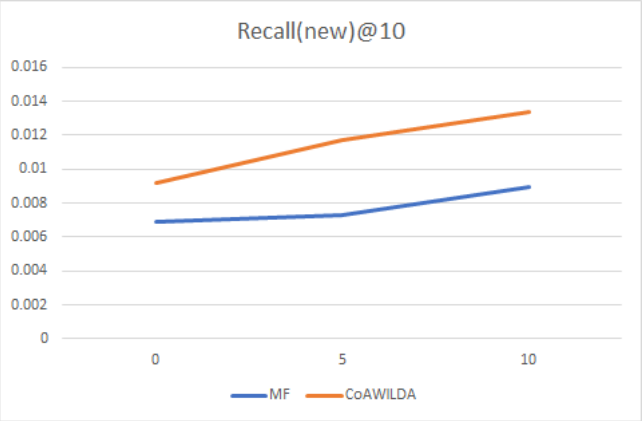


Figure 7.12: Evaluation of the recommender systems on the 01Intern dataset using Discounted Cumulative Gain @Ni.

In Figure 7.13, the results for Recall(new)@50 are shown. Again, the difference in performance between MF and CoAWILDA is not that large: about 1% for each value of r_{max}. This may indicate that for the 01Intern dataset some of the contribution from CoAWILDA lies not in the alleviation of cold start, but in the overall prediction of items.

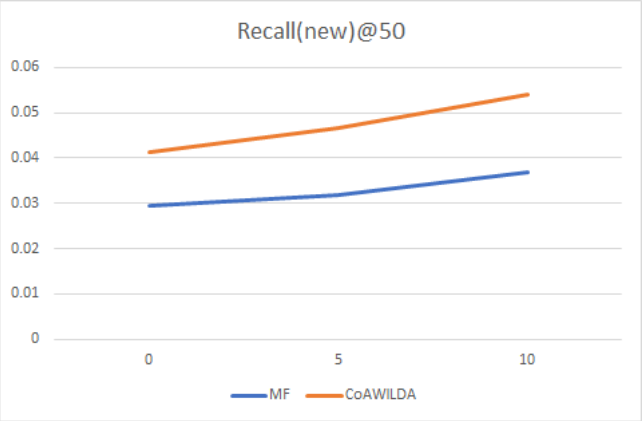


Figure 7.13: Evaluation of the recommender systems on the 01Intern dataset using Recall.

Part IV

Results

8. Discussion

In this chapter the findings from and problems encountered during the project are discussed. The questions that have not been answered over the course of the project are also presented.

8.1 Potential conflicts in recommendation model

One of the potential issues with the developed CoAWILDA+ model is the fact that factor vectors derived from different attributes are embedded in the same item factor vector. The factor vectors in question are the topics $_i$ inferred for an item i and attributes vectors of y. This potential issue may arise because $_i$ and the vectors y_a where $a \in A(i)$ are embedded into the same item factor vector q_i. While this is not necessarily problematic, in the case where $_i$ and the sum of the y_a vectors pull in opposite directions, the contribution from each approach may be reduced, and in the worst case they may each neutralise the contribution from the counterpart.

It remains unclear from the results whether this is the case. This question is one that merits further research, in future work with CoAWILDA+.

8.2 Effect of drift detection

Evaluations of the effects of drift adaption for topic modelling were performed in Section 7.1.1 and Section 7.1.4. Later, in Section 7.2.4, the effect of drift adaption for online recommendation was evaluated.

Experiments in Section 7.1.1 show that drift adaption can result in a lower average perplexity score, and thus a better fit of the modelled data. This supports the arguments in [7] that AWILDA results in a better fit of the modelled data, through providing empirical evidence of its effect on the 01Intern dataset. Furthermore, the online prequential evaluation of CoAWILDA on the 01Intern dataset show better results than for CoLDA, which does not adapt to drifts.

In the evaluation of the effects on topic modelling in Section 7.1.4, it was argued that drift adaption results in change in the relationships between the topic mixtures of documents, for which the topic mixtures are inferred prior to the drift, and those where inference takes place after. While it is likely that the topics inferred for post-drift documents are more accurate, as the model better fits the data, this may also result in in the topic mixtures of the post-drift documents being closer to or further from the pre-drift documents.

An interesting question that remains unanswered is how much of the increase in performance for CoAWILDA that can be attributed to the better modelling through drift adaption, and how much can be attributed to the shift in the topic mixtures, which can be seen as a form of “forgetting” older items. In [45] it is shown that forgetting older ratings when using MF can be beneficial. Forgetting in this context means to leave ratings out when retraining the model. It is likely that forgetting older items can similarly be beneficial to recommendation quality.

8.3 The user cold-start problem

In the 01Intern dataset, the vast majority of users have few interactions which means that user cold start might pose a serious problem for recommender systems working with the dataset. In this project, there has been no effort to alleviate this problem, as the focus has been on the item cold-start problem in accordance with the problem statement. However, as it is a signi□cant characteristic of 01Intern dataset it is a problem that is subject to further research.

In Section 2.1.4, it was shown that the 01Intern dataset contains a number of attributes for users in addition to those for items. Leveraging these attributes is a potential approach to alleviating the user cold-start problem. As described in [3], the AE approach used in the developed solution can be applied to user attributes as well. If the goal is to maximise performance using the 01Intern dataset in its original form, that is, without □ltering out users with few ratings, this approach can be considered.

8.4 Sources of error

In this section, the sources of error associated with the experiments in this project are discussed. The most obvious of these is incomplete optimisation of the hyperparameters. The fact that each model cannot be guaranteed to be in an optimal con□guration means that less con□dence can be put in the conclusions drawn during this project.

The hyperparameters and $_0$ used in online LDA have not been determined as part of the grid search for the optimal hyperparameters for each recommender system. Rather, the hyperparameters were optimised separately for the task of evaluating the effectiveness of the topic models. In the evaluation of the recommender systems, the set of documents to initially train on is much larger than in the evaluation of the topic models. As there is a difference between the stream setting in which the recommender systems are evaluated and the setting in which the topic models are evaluated, the recommender systems may have been evaluated with suboptimal hyperparameters for the topic model components.

The approach used to evaluate recommender systems on the ML-100K dataset was modi□ed in attempt to reproduce the evaluations described in ML-100K. The modi□ed approach trains the topic model incrementally on part of the batch train subset, as described in Section 7.2.5. While this contributes to the analysis the effect of drift detection (by signi□cantly increasing the potential of a drift occurring), it may have unwanted effects. Hence, this is listed as a potential source of error.

9. Conclusion

In this chapter, progress made and discoveries made during this project are summarised. The project as a whole is first summarised, after which the partial conclusions presented in the report are summarised. The conclusion answers, to some extent, the questions posed in the problem statement in Chapter 3.

Over the course of this project the model CoAWILDA+ has been developed. CoAWILDA+ is a model that stems from the idea of combining different types of item attributes with the advantages that collaborative filtering offers, in an effort to alleviate the cold start problem. CoAWILDA+ combines topic modelling of text documents using AWILDA with the AE model, which is a simple modification of the MF to include info for item attributes. The topic modelling component, AWILDA, is evaluated separately with purpose of identifying the characteristics of drift detection. A set of recommender systems is evaluated in online setting, with an emphasis on determining how well they perform on cold-start recommendation.

9.1 Alleviating cold start

The cold-start problem is usually handled by including in the model of a recommender system information about the items to be recommended. In Chapter 4, different methods for including item information for recommendation were examined. In Chapter 5, the method CoAWILDA was introduced, which elegantly combines MF with topic modelling to alleviate the problem of cold start. Following this, the CoAWILDA+ model was proposed, to combine the attractive properties of CoAWILDA and AE. Due to an incomplete hyperparameter optimisation for the CoAWILDA+ model, the performance of the model was not evaluated. The usefulness of CoAWILDA+ remains an unanswered question.

Findings from evaluation of the other recommender systems indicate that CoAWILDA is to some extent a successful approach to the alleviation of the cold start problem for the 01Intern dataset. CoAWILDA only makes use of the documents representing each item, and does not leverage additional item attributes. However, as concluded in Section 7.3, CoAWILDA still achieves performance gains over basic MF for the task of cold-start recommendation.

9.2 Attribute weighting

The question of which attributes are most relevant to users of 01Intern remains unanswered. As experiments with AE and CoAWILDA+ were unsuccessful, no attribute factor vectors have been obtained. As a consequence, there is no information about attributes that can be used to solve this sub-problem. The answer may be found by optimising the hyperparameters for CoAWILDA+ and seeing which attributes display great magnitudes and which do not.

9.3 Balancing contribution from different components

An approach that may be able to balance different attributes is AE presented in Section 4.5.1. As the attribute factor vectors are an integrated part of the MF,

the weights are implicitly inferred during training. However, as the method has not been evaluated for the 01Intern dataset, no points can be made about the effectiveness of the method.

9.4 Online recommendation

A necessary property of recommender systems for the online setting is being able to process observations at the same rate at which they appear. ISGD, described in Section 4.6, and the extension, CoAWILDA, are both models suited for the online recommendation setting. Both models perform a single-step model update with each observation received. Only occasionally, when a drift is detected, does CoAWILDA train on several observation at a single step in time. This makes it simple for the models to keep up with data streams.

In Section 7.1.1, arguments were made for the usefulness of drift detection when using topic model on a data stream. When adapting to drift, the model was able to better □t the data. As we saw in Section 7.2, the improved topic modelling carried over to recommendations, as the topic model was part of the evaluated recommender system.

Part V

Closing

Bibliography

[1] Wikipedia. (2015). Vancouver reference system. English, Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Vancouver_system (visited on Nov. 30, 2017).

[2] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems: Introduction and challenges", in *Recommender systems handbook*, Springer, 2015, pp. 1–34.

[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems", *Computer*, no. 8, pp. 30–37, 2009.

[4] L. Peska and P. Vojtas, "Negative implicit feedback in e-commerce recommender systems", in *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics*, ser. WIMS '13, Madrid, Spain: ACM, 2013, 45:1–45:4, ISBN: 978-1-4503-1850-1. DOI: 10.1145/2479787.2479800. [Online]. Available: http://doi.acm.org/10.1145/2479787.2479800.

[5] S. Debnath, N. Ganguly, and P. Mitra, "Feature weighting in content based recommendation system using social network analysis", in *Proceedings of the 17th international conference on World Wide Web*, ACM, 2008, pp. 1041–1042.

[6] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, "Methods and metrics for cold-start recommendations", in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, 2002, pp. 253–260.

[7] M. Al-Ghossein, P.-A. Murena, T. Abdessalem, A. Barré, and A. Cornuéjols, "Adaptive collaborative topic modeling for online recommendation", in *Proceedings of the 12th ACM Conference on Recommender Systems*, ACM, 2018, pp. 338–346.

[8] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong, "Addressing cold-start problem in recommendation systems", in *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, ACM, 2008, pp. 208–211.

[9] S.-T. Park and W. Chu, "Pairwise preference regression for cold-start recommendation", in *Proceedings of the third ACM conference on Recommender systems*, ACM, 2009, pp. 21–28.

[10] A. Gunawardana and G. Shani, "Evaluating recommender systems", in *Recommender Systems Handbook*, F. Ricci, L. Rokach, and B. Shapira, Eds. Boston, MA: Springer US, 2015, pp. 265–308, ISBN: 978-1-4899-7637-6. DOI: 10.1007/978-1-4899-7637-6_8. [Online]. Available: https://doi.org/10.1007/978-1-4899-7637-6_8.

[11] E. Frigó, R. Pálovics, D. Kelen, L. Kocsis, and A. Benczúr, "Online ranking prediction in non-stationary environments", 2017.

[12] V. H. Haugan, "Feature weighting in content-based recommender systems using bayesian personalized ranking", 2019.

[13] Y. Yu, C. Wang, and Y. Gao, "Attributes coupling based item enhanced matrix factorization technique for recommender systems", *arXiv preprint arXiv:1405.0770*, 2014.

[14] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context", *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, p. 19, 2016.

[15] B. Kille, F. Hopfgartner, T. Brodt, and T. Heintz, “The plista dataset”, 2013.

[16] J. Bennett, S. Lanning, *et al.*, “The net□ix prize”, in *Proceedings of KDD cup and workshop*, New York, NY, USA., vol. 2007, 2007, p. 35.

[17] GroupLens. (1998). Movielens 100k dataset, [Online]. Available: https://grouplens.org/datasets/movielens/100k/ (visited on Mar. 25, 2019).

[18] IMDb. (1990-2019). Imdb datasets, [Online]. Available: https://www.imdb.com/interfaces/ (visited on Apr. 8, 2019).

[19] J. Vinagre, A. Jorge, and J. Gama, “Fast incremental matrix factorization for recommendation with positive-only feedback”, Jul. 2014, pp. 459–470. DOI: 10.1007/978-3-319-08786-3_41.

[20] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, “Bpr: Bayesian personalized ranking from implicit feedback”, in *Proceedings of the twenty- fth conference on uncertainty in arti cial intelligence*, AUAI Press, 2009, pp. 452–461.

[21] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation”, *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[22] M. Hoffman, F. R. Bach, and D. M. Blei, “Online learning for latent dirichlet allocation”, in *advances in neural information processing systems*, 2010, pp. 856–864.

[23] X. Yang, “Understanding the variational lower bound”, 2017.

[24] J. R. Hershey and P. A. Olsen, “Approximating the kullback leibler divergence between gaussian mixture models”, in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP ’07*, vol. 4, Apr. 2007, pp. IV-317-IV-320. DOI: 10.1109/ICASSP.2007.366913.

[25] A. Bifet and R. Gavalda, “Learning from time-changing data with adaptive windowing”, in *Proceedings of the 2007 SIAM international conference on data mining*, SIAM, 2007, pp. 443–448.

[26] P. M. Grulich, R. Saitenmacher, J. Traub, S. Breß, T. Rabl, and V. Markl, “Scalable detection of concept drifts on data streams with parallel adaptive windowing.”, in *EDBT*, 2018, pp. 477–480.

[27] Y. Koren and R. Bell, “Advances in collaborative □ltering”, in *Recommender systems handbook*, Springer, 2015, pp. 77–118.

[28] G. Golub and W. Kahan, “Calculating the singular values and pseudo-inverse of a matrix”, *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965.

[29] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Application of dimensionality reduction in recommender system-a case study”, Minnesota Univ Minneapolis Dept of Computer Science, Tech. Rep., 2000.

[30] Y. Koren, “Factorization meets the neighborhood: A multifaceted collaborative □ltering model”, in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2008, pp. 426–434.

[31] J. Gorski, F. Pfeuffer, and K. Klamroth, “Biconvex sets and optimization with biconvex functions: A survey and extensions”, *Math. Meth. of OR*, vol. 66, pp. 373–407, 2007.

[32] J. Nguyen and M. Zhu, “Content-boosted matrix factorization techniques for recommender systems”, *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 6, no. 4, pp. 286–301, 2013.

[33] Y. Zhen, W.-J. Li, and D.-Y. Yeung, “Tagico□: Tag informed collaborative □ltering”, in *Proceedings of the third ACM conference on Recommender systems*, ACM, 2009, pp. 69–76.

[34] R. Van Meteren and M. Van Someren, “Using content-based □ltering for recommendation”, in *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*, 2000, pp. 47–56.

[35] J. Ramos *et al.*, “Using tf-idf to determine word relevance in document queries”, in *Proceedings of the rst instructional conference on machine learning*, Piscataway, NJ, vol. 242, 2003, pp. 133–142.

[36] P.-A. Murena, M. Al-Ghossein, T. Abdessalem, and A. Cornuéjols, “Adaptive window strategy for topic modeling in document streams”, in *2018 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2018, pp. 1–7.

[37] D. M. Blei and J. D. Lafferty, “Dynamic topic models”, in *Proceedings of the 23rd international conference on Machine learning*, ACM, 2006, pp. 113–120.

[38] L. Du, W. L. Buntine, and H. Jin, “Sequential latent dirichlet allocation: Discover underlying topic structures within a document”, in *2010 IEEE International Conference on Data Mining*, IEEE, 2010, pp. 148–157.

[39] T. L. Grif□ths and M. Steyvers, “Finding scienti□c topics”, *Proceedings of the National academy of Sciences*, vol. 101, no. suppl 1, pp. 5228–5235, 2004.

[40] X. Wang and A. McCallum, “Topics over time: A non-markov continuous-time model of topical trends”, in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2006, pp. 424–433.

[41] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, “Indexing by latent semantic analysis”, *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990.

[42] Atilika. (). About kuromoji, [Online]. Available: `http://www.atilika.org/` (visited on Dec. 10, 2018).

[43] T. A. S. Foundation. (2017). The apache opennlp library. English, [Online]. Available: `https://opennlp.apache.org/` (visited on Jun. 12, 2019).

[44] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “MOA: massive online analysis”, *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010. [Online]. Available: `http://portal.acm.org/citation.cfm?id=1859903`.

[45] P. Matuszyk and M. Spiliopoulou, “Selective forgetting for incremental matrix factorization in recommender systems”, in *International Conference on Discovery Science*, Springer, 2014, pp. 204–215.

List of Figures

List of Tables

List of Listings

Part VI

Appendix

A. Original terms and documents

This chapter includes the original terms and documents from the 01Intern dataset.

[Â [->0,0188] [® ->0,0146] [. _ ->0,0133] [K ->0,0133] [÷ ->0,012] [í 長->0,0103] [5 ->0,0096] [->0,0095] [™ ->0,0085] [Á ->0,008] [â ->0,0079] [it->0,0073] [A S 0 ->0,0071] [• [->0,0071] [€ ->0,007] [的->0,0066] [S ->0,0063] [™ 長->0,0062] [r — ->0,0061] [9 S ! C ->0,006] [› @ ->0,006] [™ V ->0,0059] [中->0,0058] [[->0,0057] [« ™ ->0,0056] [® ë ->0,0055] [7 J ->0,0054] [->0,0053] [a ->0,0052] [Ù 用->0,0048] [º ->0,0047] [R ->0,0046] [_ [->0,0045] [â Q ÷ ->0,0043] [? ->0,0042] [->0,0042] [P 味->0,0042] [h ->0,0041] [3 ->0,0039] [› ->0,0039] [g ->0,0038] [™ ->0,0037] [˜ á ->0,0036] [â _ ->0,0036] [S * ->0,0036] [o 本->0,0035] [K ->0,0035] [5 I S ->0,0035] [Ç ->0,0034] [(->0,0033] [名->0,0033] [D K ->0,0033] [A ' ->0,0032] [– Ï ->0,0032] [s Þ ->0,0031] [(->0,0031] [m w ™ ->0,0031] [Á • ->0,0031] [‰ ->0,003] [Â ->0,003] [= (->0,003] [2 ->0,003] [Q ->0,003] [! C L S ->0,003] [è ù ->0,003] [a # # ->0,003] [» Ò ->0,0029] [˜ ->0,0029] [[務->0,0028] [| • ->0,0028] [™ ->0,0028] [# 7 ->0,0028] [® ->0,0027] [K ->0,0027] [d 敵->0,0027] [U ->0,0027] [u ->0,0027] [o ->0,0026] [様->0,0026] [0 J ->0,0026] [知->0,0026] [æ " ->0,0025] [• ->0,0025] [S ->0,0025] [本 Ù ->0,0025] [長 Ó ->0,0025] [ã ->0,0024] [web->0,0024] [S + ->0,0024] [通->0,0024] [ê ->0,0024] [! C S ->0,0024] [ß ->0,0023] [B ! M S ->0,0023] [÷ a ->0,0023] [Q ÷ ->0,0023] [3 4 ->0,0023] [l [->0,0022] [) h ->0,0022] [知 x ->0,0022] [> # ! ->0,0022] [e ->0,0022] [# J ->0,0022] [->0,0022] [' ->0,0022] [ß Ž ->0,0022] [r ™ ->0,0021] [K ->0,0021] [+ (# & ->0,0021] [ð ö ->0,0021] [ ) ->0,0021] [u ->0,0021] [• à ->0,0021] [ù ->0,0021] [r 由->0,0021] [£ R ->0,0021] [優Í ->0,0021] [->0,002] [$ î ->0,002] [提 % ->0,002] [魅 2 ->0,002] [r ->0,002] [® ->0,002] [ã ->0,002] [/ ->0,002] [• ->0,002] [Ù 値->0,002] [¯ ->0,002] [Ò ->0,0019] [• ->0,0019] [中 • ->0,0019] [„ ->0,0019] [roots->0,0019] [通 • ->0,0019] [> & S ->0,0019] [Ý ->0,0019] [– 野->0,0019] [$ K ->0,0019] [Ø ->0,0019] [® ÷ ->0,0018] [4 x ->0,0018]

Figure A.1: A sample of the original Japanese terms for Topic 1.

0) DB + $ & . ú x R © + $ I L K S + [務 ' ? * S + / Â [M Web 3 6 Zealup 7 . $ Þ R u > Zealup / ® . 努 2 ™ Q É . 目 E + ! S (Ã # & K R K 目 E $ K F 0 Web 3 R $ Þ K h ' B $ Þ > & S = (*) 9 要 * 役] / 様 役] B ® . í ç 3 ‡ í +) L + ƒ T & K . R Q V ' 9 有 K (' e † Q É . 目 E ' K 3 ‡ í R ü í H (L > ' + * 3 '

Figure A.2: A sample of the original Japanese job description for Job A.

B. Lists of stop words

In this section the lists of stop words used for removal of stop words in documents are presented. The list of Japanese stop words can be seen in Figure B.1 and the list of English stop words can be seen in Figure B.2. The English list of stop words is from the Natural Language Toolkit[1], while the Japanese list is from the Stopwords ISO GitHub repository[2], with a few domain specific additions.

[1] https://www.nltk.org/
[2] https://github.com/stopwords-iso

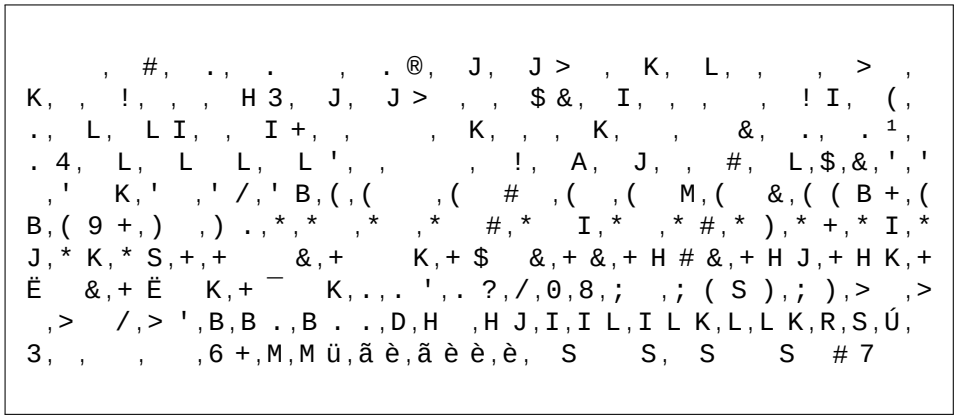


Figure B.1: The list of Japanese stop words

i,me,my,myself,we,our,ours,ourselves,you,your,yours,yourself,yourselves,he,
him,his,himself,she,her,hers,herself,it,its,itself,they,them,their,theirs,
themselves,what,which,who,whom,this,that,these,those,am,is,are,was,were,be,
been,being,have,has,had,having,do,does,did,doing,a,an,the,and,but,if,or,
because,as,until,while,of,at,by,for,with,about,against,between,into,through,
during,before,after,above,below,to,from,up,down,in,out,on,off,over,under,
again,further,then,once,here,there,when,where,why,how,all,any,both,each,few,
more,most,other,some,such,no,nor,not,only,own,same,so,than,too,very,s,t,can,
will,just,don,should,now

Figure B.2: The list of English stop words

C. kNN regression recommender settings

In this section the settings of the kNN regression recommender, which was used as a baseline in the evaluation of CoAWILDA+, are described. As a good trade-off between accuracy and performance, k is assigned the value 10. In Table C.1, the weights assigned to different attributes x, and the similarity measures used for comparing them $a_{ii'}$ are shown.

x	Attribute	$a_{ii'}$
0.1884	Job type	Jaccard similarity
0.0253	Business type	Jaccard similarity
1.0	Appeal	Cosine similarity/TF-IDF
0.4965	Entrusted job detail	Cosine similarity/TF-IDF
0.07361	Quali□cation	Cosine similarity/TF-IDF
0.0011	Sticking condition	Jaccard similarity
0.7841	Work prefecture	Jaccard similarity
0.0682	Work city	Jaccard similarity

Table C.1: The weight and similarity measure used for each attribute in the kNN regression recommender