Informatics

A Recommender System for the Matchmaking of Event Participants

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Daniel Bugl, BSc
Matrikelnummer 01425285

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Allan Hanbury
Mitwirkung: Univ.Ass. Dipl.-Ing. Mete Sertkan

Wien, 2. Mai 2023

Daniel Bugl	Allan Hanbury

Informatics

A Recommender System for the Matchmaking of Event Participants

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Daniel Bugl, BSc
Registration Number 01425285

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Allan Hanbury
Assistance: Univ.Ass. Dipl.-Ing. Mete Sertkan

Vienna, 2nd May, 2023

Daniel Bugl

Allan Hanbury

Erklärung zur Verfassung der Arbeit

Daniel Bugl, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 2. Mai 2023

Daniel Bugl

Acknowledgements

I want to thank Mete Sertkan and Allan Hanbury for their support and reviews during the writing of this thesis. Their ideas and knowledge really helped me shape my thesis and the project resulting from it. Additionally, I would like to thank everyone from b2match, but especially Mirza Ceric, Goran Popovic and Fran Hrabar for their valuable input in the project and helping with improving the data exports from their platform. Also, thank you for letting me evaluate the recommender system during live events. I also want to thank my family and friends for supporting me during my study. This thesis would not have been possible without all these people - thank you so much!

Kurzfassung

Viele Teile unseres Lebens werden in letzter Zeit digitalisiert. Insbesondere Veranstaltungen, die traditionell persönlich abgehalten wurden, wechseln nun zu digitalen und hybriden Formaten. Solche Formate erschweren es, die richtigen Personen zu treffen, insbesondere bei großen Veranstaltungen. Ziel dieser Arbeit ist es, ein Empfehlungssystem ("Recommender System") für Business-to-Business (B2B)-Veranstaltungen zu entwerfen, zu implementieren und zu evaluieren, das jedem Teilnehmer eine personalisierte Liste von Teilnehmern zur Verfügung stellt, die für ihn oder sie von Interesse sein könnten. Dabei wird aus vergangenen Interaktionen gelernt und ausschließlich auf implizites Feedback wie Besuche, Lesezeichen, Nachrichten und gebuchte Meetings zurückgegriffen. Wir versuchen auch Empfehlungen für Teilnehmer zu machen, die keine vergangenen Interaktionen haben, indem wir ähnliche Benutzer in den Empfehlungsprozess einbeziehen. Die Daten werden von der Firma b2match bereitgestellt, die eine Online-Plattform zur Verwaltung von Veranstaltungen anbietet. Das Empfehlungssystem wird iterativ mit dem CRISP-DM-Prozess entwickelt, als eigenständiger Service implementiert und in die b2match-Plattform integriert. Um zu überprüfen, ob das Empfehlungssystem gut funktioniert, führen wir eine Offline-Evaluierung gegen Baselines (18 Veranstaltungen für die Entwicklung, 6 Veranstaltungen für die Evaluierung) und eine Online-Evaluierung auf 27 Veranstaltungen durch, die das Empfehlungssystem in der Produktion verwenden. Für die Bewertung mitteln wir die nDCG@10-Ranking-Metrik auf den personalisierten Listen von Teilnehmern aus, wobei jeder Teilnehmer eine Liste erhält. Die Ergebnisse sind vielversprechend. Das entwickelte Empfehlungssystem schneidet in einer Offline-Evaluierung signifikant besser ab als alle Baselines mit einem nDCG@10-Score von 0,1967. Die Ergebnisse der Baselines waren wie folgt: 0,0361 für eine Liste, die aus der Normalverteilung gezogen wurde ($p = 0{,}0044$), 0,0716 für ein Popularitätsranking ($p = 0{,}0073$), 0,0277 für zufällige Listen ($p = 0{,}0045$), 0,0452 für ein Ähnlichkeitsranking ($p = 0{,}0051$). Durch Hinzufügen eines hybriden Empfehlungssystems zur Lösung des Cold-Start-Problems konnten wir eine Verbesserung in Bezug auf nDCG@10 von 0,1967 auf 0,2227 ($p = 0{,}0051$) erreichen. Das entwickelte Empfehlungssystem erhöht auch die relative Anzahl erfolgreicher Meetings in einer Online-Evaluierung von 0,18% auf 0,31% ($p = 0{,}0005$). Unsere Studie kommt zu dem Schluss, dass Matrixfaktorisierungsalgorithmen auf unserem B2B-Event-Datensatz am besten abschneiden. Wenn ein Cold-Start-Szenario auftritt und keine Vorhersage für einen Teilnehmer getroffen werden kann, liefern personalisierte Empfehlungen auf der Grundlage von Interaktionsdaten ähnlicher Teilnehmer vielversprechende Ergebnisse.

Abstract

Recently, many parts of our lives are becoming digitized. Especially events, which were traditionally held in-person, are now migrating to digital and hybrid formats. Such formats make it harder to find the right people to meet, particularly at large events. The goal of this thesis is to design, implement and evaluate a recommender system for business-to-business (B2B) events that provides to each participant a personalized list of participants that might be of interest to them, by learning from past interactions, relying exclusively on implicit feedback, such as visits, bookmarks, messages and meetings booked. We also attempt to make recommendations for participants that have no past interactions, by incorporating similar users into the recommendation process. Data is provided by the company b2match[1], which provides an online platform for managing events. The recommender system is developed iteratively using the CRISP-DM process, implemented as a standalone service, and integrated into the b2match platform. To verify that the recommender system works well, we do an offline evaluation against baselines (18 events for development, 6 events for evaluation) and an online evaluation on 27 events that use the recommender system in production. For the evaluation we average the nDCG@10 ranking metric on the personalized lists of participants, one list provided for each participant. The results are promising. The developed recommender system performs significantly better than all baselines in an offline evaluation, with an nDCG@10 score of 0.1967. The results of the baselines were as follows: 0.0361 for a list sampled from the normal distribution ($p = 0.0044$), 0.0716 for a popularity ranking ($p = 0.0073$), 0.0277 for random lists ($p = 0.0045$), 0.0452 for a similarity ranking ($p = 0.0051$). Adding a hybrid recommender to solve the cold start problem, we were able to achieve an improvement in terms of nDCG@10 from 0.1967 to 0.2227 ($p = 0.0051$). The developed recommender also increases the relative number of successful meetings in an online evaluation from 0.18% to 0.31% ($p = 0.0005$). Our study concludes that matrix factorization-based algorithms perform best on our B2B event data set. When a cold-start scenario arises and a prediction cannot be made for a participant, providing personalized recommendations based on interaction data from similar participants yields promising results.

[1]`https://www.b2match.com`

Contents

CHAPTER 1

Introduction

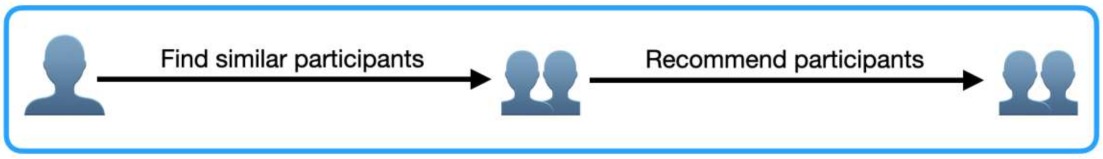
Recent advances in computer science and the pandemic have resulted in large parts of our lives becoming increasingly digitized. This digitization especially affected events, traditionally held in-person, which are now migrating to digital formats, such as online-only events and hybrid events. Business-to-Business (B2B) events present a unique opportunity for participants to promote their interests and network with others. For example, investors may be trying to find potential startups to invest in, or companies may be trying to find potential clients at an event. In such a context, digital formats have a considerable disadvantage: It is harder to find the right people to meet from a long list of participants, particularly for large events where participants have only limited availability.

To support people in finding the right people to meet, we can introduce Artificial Intelligence (AI) to the events, which can recommend participants of interest and provide a personalized list to each participant. Nowadays, AI has permeated most parts of our lives, with machines performing and supporting various human tasks [Pom97]. One application of AI is supporting humans in decision-making, such as selecting relevant entities from a large set. AI systems that support humans in selecting relevant entities are called recommender systems, as they recommend relevant entities to the user [RV97]. Recommender systems learn patterns from existing information, for example, by analyzing interactions from previous events, such as which participants someone met with, bookmarked, messaged, or visited. Then, recommender systems can use these learned patterns to recommend potential participants to meet that will result in more interactions. More interactions means that the participants thought they were interesting to each other, so the recommender system is implicitly learning to recommend interesting participants.

This thesis aims to design a recommender system for B2B events that provides each participant a personalized list of participants that might be of interest to them, solving the aforementioned problem by reducing the long list of participants to a shorter list, including only participants of interest to them. The system should learn to recommend relevant participants to meet by analyzing past interactions, such as visits, bookmarks, messages sent, meeting requests and accepted meetings. In existing people-to-people recommender systems, such as online dating recommenders [PRC+10a], online classrooms [PSZ17] or recommender systems for job applicants [WYJY15], profile information is vastly available, as people have an interest to enter as much information about themselves as possible. However, in our B2B context, participants often do not want to disclose information about what they are looking for on a platform, especially not publicly, as it may leak internal strategic information. As a result, we do not have extensive information about the profiles of participants, and have to almost exclusively rely on interactions between them. This is a unique challenge for recommender systems. Online dating recommenders also often focus on recommending people who have something in common. In our context, however, this would be a downside. For example, an investor is most likely not interested in meeting other investors, but rather looking for startup founders. As a result, the basic recommender system has to focus on finding implicit patterns of interest, without relying on explicit profile information.

An additional problem with our B2B context, which has not been traditionally a problem with recommender systems, is that there are various kinds of events, such as startup founders looking for investors, or businesses trying to find partners and customers. These different kinds of events require a recommender system that can detect heterogeneous interest patterns instead of relying on common patterns. When we cannot rely on common patterns from previous events, the cold start problem [LKH14] becomes even more pronounced: When event participants have no signals yet (new users), we cannot tell who they would be interested in meeting. If we have only unstructured profile information, and no signals, we have no information to make a recommendation from. To solve this issue, we introduce a hybrid recommender system (Figure 1.1), which uses the sparse information we have on profiles (a list of similar participants). These participants already have signals, which makes it possible to detect implicit interest patterns between them and other participants. For the selected similar participants, we now make a recommendation. This recommendation is used to provide a semi-personalized list of participants that might be of interest to the new user.

Figure 1.1: Diagram of the Hybrid Recommender System for B2B Events



1.1 Research Questions

The digitization of B2B events presents unique challenges, such as the difficulty of finding the right people to meet from a long list of participants. Additionally, the B2B context provides unique challenges for recommender systems, such as having to deal with various kinds of events, which do not have common patterns of interest, and a lack of profile information. In this thesis, we aim to design a recommender system for B2B events that provides personalized recommendations for participants. Our goal is to investigate the effectiveness of different recommender algorithms, as well as a hybrid approach that uses recommendations from similar users to improve the quality of recommendations for new users.

Research has shown that people-to-people recommender systems can provide good recommendations for domains with homogeneous data sets, such as online dating [PRC+10a]. However, in our case, we do not have homogeneous data, as each event has vastly different kinds of participants and thus also different patterns of interest. For example, when recommending investors to startups, different qualities are important than when recommending potential clients to a company. Our aim is to find out which recommender algorithms work well for a heterogeneous data set of various different kinds of B2B events. We compare Co-Clustering, Collaborative Filtering (k-NN) and Matrix Factorization algorithms to the following baseline models: Random sorting, a random predictor based on the normal distribution of ratings, recommending the most popular, and recommending the most similar users. Thus, we investigate:

- **RQ1:** Which recommender algorithms provide an appropriate quality of results for a data set with heterogeneous events, compared to baseline models?

Many implementations of people-to-people recommender systems take into account both directions of preference [KY22], making recommendations reciprocal. Reciprocal recommender systems are symmetrical, if user A gets recommended user B, user B should also get recommended user A, with the same score [PRC+10a]. Instead of only taking into account the one-directional interest score of A to B, we can also compute the interest score of B to A and average the scores, resulting in a bi-directional prediction. We explore making bi-directional predictions for our recommender system:

- **RQ2:** To what extent does the use of a reciprocal bi-directional prediction change the overall quality of recommendations?

Existing approaches to the cold start problem either focus on improving the quality of recommendations for users with few signals [BOHB12], or attempt to classify users into demographic groups [LVLD08]. In our B2B event context, however, we have users with no signals (right after they register), and demographic data is not provided in our data set. Generally, existing solutions make use of well structured profile data, while in our context, we only have a large amount of signals, but almost no profile data. When we have no signals for a participant, we will use a similarity measure based on the unstructured profile description, which is the only profile information we have, to find similar participants to them. Then, we make a recommendation for those similar participants, for which we do have signals. We argue that such a hybrid recommender system that uses recommendations from similar users could improve recommendation quality for cold start cases. Thus, we investigate:

- **RQ3:** To what extent does the hybrid recommender change the overall quality of recommendations?

For RQ1, RQ2 and RQ3, we perform an offline evaluation using existing data from previous events and use Normalized Discounted Cumulative Gain (nDCG@10) as the relevance metric [ZBS+16]. We recommend a list of 10 participants to each participant and compare it to the perfectly ranked list of 10 participants according to the ground truth.

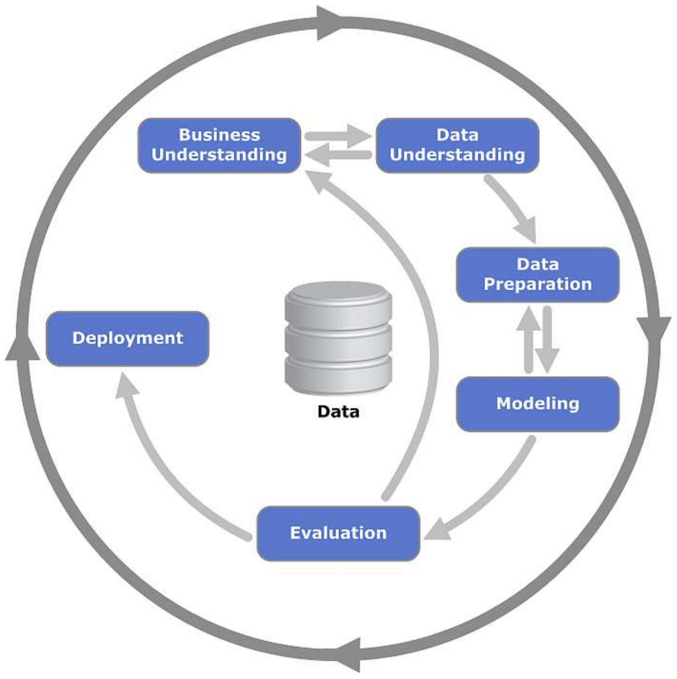
As the recommender system developed in this thesis is used in the real world, it is necessary to also test its performance at events where the recommender system is used. The company b2match offered us an opportunity to deploy the developed recommender system on their platform and do an online evaluation. The recommender system used for this online evaluation is the version optimized according to the results of the offline evaluation. The business success criteria for b2match is increasing the number of successful meetings booked on their platform. As we have no explicit ratings for the meeting quality, a meeting is considered successful when it was successfully held and neither canceled nor missed. Thus, we investigate:

- **RQ4:** How does the developed recommender system change the number of successful meetings in an online evaluation?

1.2 Methodological Approach

The methodological approach follows a typical data science project, consisting of analysis of the data, implementation of the recommender system, and finally evaluation of the results [Cao17]. Specifically, we follow the CRoss Industry Standard Process for Data Mining (CRISP-DM) process [WH00], which is an iterative approach that allows for multiple opportunities to re-evaluate the direction of the project and if it is still achieving its original objectives. This iterative approach is especially important when working together with a company, where first evaluations can lead to changes in the exported data set, which lead to further evaluations. The CRISP-DM process consists of the following phases:

Figure 1.2: Diagram of the CRISP-DM process, by Kenneth Jensen [Jen12]



1.2.1 Business Understanding

We set objectives and the business success criteria, which is increasing the number of successful meetings booked on the platform. A meeting is considered successful if it was accepted by both participants involved, and did not end up being canceled or missed. As we are dealing with implicit ratings, we also define the importance of certain signals in the business context, and assign a score to each of them. For example, visits are less of an indicator of interest than bookmarks. The highest indicator of interest, and thus the highest score, is our business success criteria, a successful meeting. The definition of this interest score directly affects the performance of the recommender, so we need to work with domain experts from the company to define reasonable scores for the different signals. As we are not including any explicit feedback from participants in our evaluation, the usage of implicit signals may result in a bias towards business goals rather than recommending participants that are actually interesting.

1.2.2 Data Understanding

We analyze the data set using descriptive statistics and report potential problems with data quality. Data for 24 events is provided, of which we use 18 events for training and optimizing the recommender system, and 6 events for testing the performance of the optimized recommender system. The provided data set consists of participant pairs with *visits*, *bookmarks*, *messages sent*, *meeting requested* and *meeting accepted* signals.

Furthermore, for each participant, lists of similar participants in the same event are provided.

1.2.3 Data Preparation

We clean the data, and derive an *interest score*, which defines how interesting a participant is to another participant. This interest score is based on signals between the two participants, and will be used as the target value to train and evaluate the models on. Specifically, the interest score uses the importance of signals defined in the *Business Understanding* phase. The less important a signal is, the lower the score will be.

1.2.4 Modeling

In the modeling phase, we create a plan for training, testing, and evaluating the models. Then, we implement the recommender system.

Implementation of the Basic Recommender

We start by implementing a basic recommender system, based only on the signals between participants, including information such as profile visits, bookmarks, messages sent and meetings booked. We explore different algorithms and their performance on our data set.

Implementation of the Reciprocal Recommender

We explore using bi-directional predictions in a reciprocal recommender system, using various aggregation methods to combine the predictions. We evaluate how bi-directional predictions affect the overall quality of recommendations in comparison to simple one-directional predictions.

Implementation of the Hybrid Recommender

We extend the recommender system by providing a list of similar participants for each participant. When a participant has few or no signals, we use this list to find similar participants, recommend matches to them, and then provide this list of recommendations to the participant with few signals. We evaluate how the hybrid recommender affects the overall quality of recommendations in comparison to making direct predictions for participants with few or no signals.

1.2.5 Evaluation

The evaluation is split into four parts, to answer each of the four research questions. We use a combination of offline and online evaluation [RRS15] to ensure that the business success criteria has been met. In the evaluation, we use the nDCG@10 [ZBS$^+$16] ranking metric to compare the recommended lists to optimal lists. Optimal lists are created by analyzing signals and giving them a score (lowest score being no interaction between participants, highest score being a meeting was successfully held, so neither canceled nor missed), then sorting the list based on the score.

Offline Evaluation of the Basic Recommender

We compare our model using Co-Clustering, Collaborative Filtering (k-NN) and Matrix Factorization algorithms with various parameters to the following baselines: Random sorting, a random predictor based on the normal distribution of ratings, recommending the most popular, and recommending the most similar users.

Offline Evaluation of the Reciprocal Recommender

We compare a reciprocal version of the recommender system, making bi-directional predictions and aggregating them, to the basic recommender and the baselines.

Offline Evaluation of the Hybrid Recommender

We evaluate the hybrid recommender, comparing it to the basic recommender and the baselines. We also optimize the hybrid recommender to find out how many signals are needed to be able to make a direct prediction using the signals from the participants themselves. When less than this amount of signals are available, we use the hybrid recommender.

Online Evaluation

Since the recommender system is used in the real world, we evaluate the performance during events managed by b2match. The online evaluation compares the relative change in successful meetings for participants found through the recommender systems, versus participants selected from the long list of all participants.

1.2.6 Deployment

We provide b2match with information on integrating and deploying the recommender system, assist with deployment, and produce a final report.

1.3 Contributions

Throughout this thesis, a prototype of a recommender system is developed and evaluated based on historic event data provided by the company b2match, which provides a platform for managing events. We optimize the recommender system based on 18 events and validate the resulting model on 6 events. After evaluation, the prototype is deployed and integrated into their platform, providing personalized recommendations at 27 events. The signals collected from these events is then evaluated in this thesis, analyzing whether participants that were found through the personalized list provided by the recommender system resulted in more successful meetings in comparison to those found by manually looking through the long list of all participants.

The results of this thesis show how recommender algorithms perform in a B2B context, with events catering to various different groups of people, without using profile information or explicit ratings, and relying exclusively on implicit feedback. We also show that for participants with no signals, incorporating the little amount of unstructured profile information we have to find similar participants, and using their recommendations to make a recommendation for the participant with no signals, significantly improves the overall performance of the recommender system. In production, the recommender system also met the business success criteria, increasing the number of successful meetings.

1.4 Structure

The thesis consists of 7 chapters:

- **Chapter 2: State of the Art** covers existing recommender systems and already solved problems.
- **Chapter 3: Methodology** describes in detail the approach on how the project was iteratively implemented and evaluated.
- **Chapter 4: The CRISP-DM Process** gives a retrospective view on the iterations that lead to the final recommender system and lessons learned along the way.
- **Chapter 5: Implementation** gives an overview of the technologies involved in the final recommender system and why they were chosen.
- **Chapter 6: Offline Evaluation Results** contains the results of the final evaluation on the offline data sets.
- **Chapter 7: Online Evaluation Results** contains the results of the online evaluation of the recommender system being used in the real world.
- **Chapter 8: Conclusion** discusses the results and describes further work.

CHAPTER 2

State of the Art

In everyday life, it is often necessary to make choices without having extensive information about the various alternatives. For example, we often rely on recommendations from our family or friends to make decisions, such as which restaurant we visit or which products we buy. Recommender systems attempt to automate this process on a larger scale, where implicit and/or explicit feedback of other persons is used to make automated recommendations [RV97].

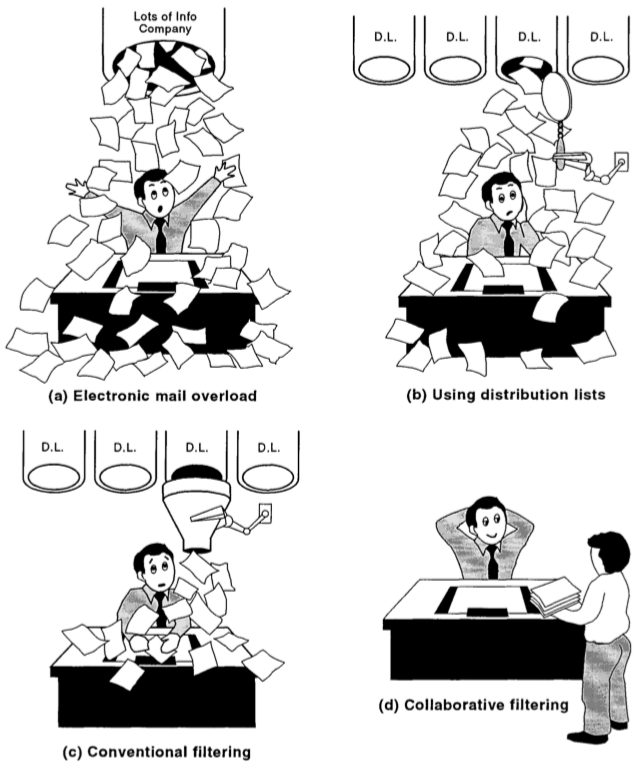
2.1 Recommender Systems

There are already various existing models for recommender systems, such as content-based approaches [IDGL$^+$08], item-item [LSY03] and user-user collaborative filtering [ERK11], which started to appear as far back as 1992, with the Tapestry system [GNOT92]. Most existing recommender systems, such as MovieLens [MAL$^+$03] and PolyLens [OCKR01] focus on recommending items with similar content. For example, if you watch a lot of action movies, you are likely going to enjoy other action movies (content-based), or at least movies that other action-movie lovers enjoyed (user-user collaborative filtering).

2.1.1 Collaborative Filtering (CF)

The first appearance of collaborative filtering systems was in the Tapestry system, which was an experimental mail system developed at the Xerox Palo Alto Research Center. Back then, the motivation was that e-mail usage started increasing and companies were overloaded by the large amount of information flowing in. The only way to handle this situation back then was using mailing lists, such that users can subscribe to the lists that they are interested in. However, it is sometimes hard to map e-mails of interest to a certain mailing list, as the interests of each user are unique. An alternative to mailing lists are filters, which would scan all lists and select based on predefined criteria. The idea of Tapestry was to involve humans in the filtering process by collecting information on which e-mails were interesting to them in the past [GNOT92].

Figure 2.1: Visualization of the different organization systems [GNOT92]


Lots of Info Company
(a) Electronic mail overload
D.L.
D.L.
D.L.
D.L.
(b) Using distribution lists
D.L.
D.L.
D.L.
D.L.
(c) Conventional filtering
(d) Collaborative filtering

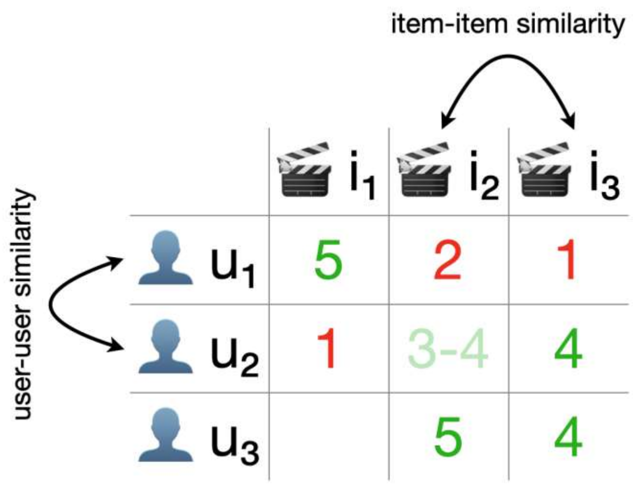
All collaborative filtering systems consist of *users*, which express interests in *items* with a *rating* score. The rating score may be explicit, for example, rating movies on a scale of 1-5 stars. Sometimes, the rating score is implicit, and we have to compute a score based on signals from the user, such as how long a movie was watched for, or how often movies were clicked on [ERK11].

The user-item pairs can then be represented in a rating matrix $\mathbf{R}$, where the values are either known ratings or unknown. The unknown ratings are then predicted using an algorithm, which we will go into detail later in this chapter. After predicting all missing ratings in the matrix, we can select a ranked list of n items that would be most relevant to a given user based on the predicted ratings.

Fundamentally, the task of recommender systems is to compute $\hat{R}(u,i)$, where for each user u and item i an estimated rating is computed. This is in contrast to the true function R, which can only provide ratings for user-item pairs defined in the rating matrix $\mathbf{R}$ [RRS15].

In Figure 2.2 we can see an example of a sparse rating matrix $\mathbf{R}$, with movie ratings on

Figure 2.2: Visualization of the rating matrix **R**



a scale of 1 to 5. We can see that items i_2 and i_3 are similar to each other, because a user who gives one of them a bad rating, also gives the other one a bad rating, while a user who gives one of them a good rating also gives the other a good rating. We can also see that users u_1 and u_2 are very dissimilar to each other, as they give the same movies opposite ratings. These relationships between users and items allow us to compute predicted ratings $\hat{R}(u, i)$ for the missing user-item combinations. For example, we could predict that since i_2 and i_3 are similar, and u_1 and u_2 are dissimilar, the rating $r_{2,2}$ for u_2 on i_2 might be around 3-4.

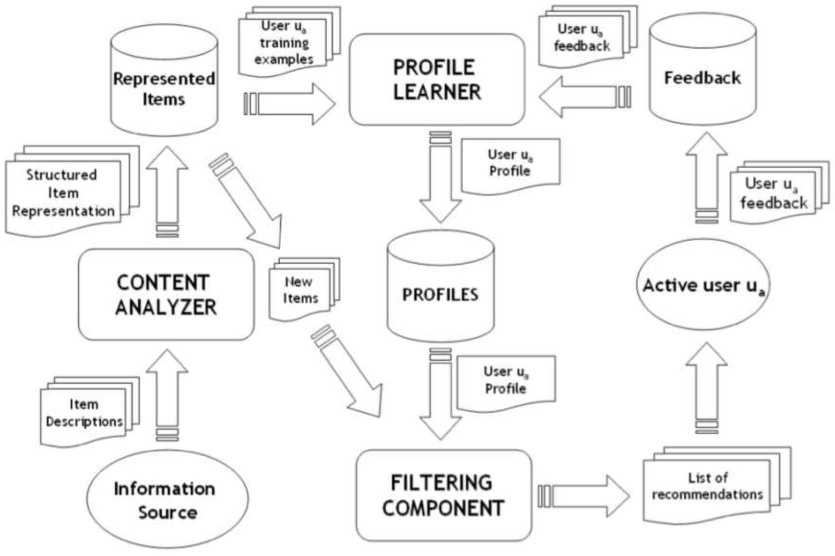
There are two main variants of collaborative filtering systems: User-user collaborative filtering and item-item collaborative filtering. The main idea of user-user collaborative filtering is to, for a given user, find other users whose behavior is similar in terms of ratings. Then, we can use their ratings to predict what items the given user might like. As a simple example, if user u_2 and user u_3 both agree on their ratings for item i_3 (4 stars), user u_3 rated item i_2 with 5 stars, but user u_2 has not rated item i_2 yet, we can assume that user u_2 will likely rate item i_2 highly too. In user-user CF, the predicted ratings are also weighted by the degree of agreement with other users. In contrast to user-user CF, item-item CF uses item similarity instead of user similarity. This solves a performance issue with user-user CF, where a growing number of users increases the computation complexity of user similarity, which is at best a $O(|U|)$ operation. In situations where there is a high user to item ratio, an item-item CF system significantly increases performance, as item similarity is unlikely to change through the rating of a single user [LSY03].

In our recommender system, we do not recommend items to users, but instead the items themselves are users again. This is a special case, called people-to-people recommender systems. However, the same principles still apply. In our case, we can consider users to be "hosts" (people who request meetings, the active party of the interaction), and items to be "guests" (people who respond to the request, the passive party of the interaction). Nevertheless, collaborative filtering systems have issues that make them non-suitable for our use case. Specifically, they do not deal well with sparse data. Large, sparse rating matrices result in very poor runtime performance and the cold start problem. Usually, these issues could be solved by using either user-user CF or item-item CF, depending on whether there are many items or many users. However, in our case, items and users are the same set, so for both CF systems we have the same runtime performance and cold start problems. Existing research of people-to-people recommenders that use collaborative filtering have found that these work especially well in finding previously known contacts, for example, in social networks [CGD+09]. However, this is also a downside in our case, as in a B2B matchmaking context, we want to find new people to meet rather than already known people.

2.1.2 Content-Based Recommender Systems

Content-based recommender systems focus on analyzing descriptions of previously rated items. Then, a model is built based on the features of the objects that were previously rated by the user [LDGS11]. For example, for a movie recommender system, a content-based recommender might find out that a user likes action-adventure movies, or movies with a certain kind of story, as analyzed from the movie genres and descriptions.

Figure 2.3: Diagram of a content-based recommender system [LDGS11]



Content-based systems consist of:

- A *content analyzer*: to convert item descriptions into a structured representation

- A *profile learner*: to build user profiles based on previous ratings and the structured representations
- A *filtering component*: which uses the user profiles to generate a list of recommendations for a given user

To learn the user profiles, we use feedback from the users to figure out which items are interesting to them. There are various kinds of feedback, mostly grouped into two groups: explicit and implicit feedback. Explicit feedback are explicit ratings given by the user, such as liking/disliking an item or rating it on a scale of 1-5. Implicit feedback does not require active user involvement, and can be derived from the user's activities, such as visiting an item's page, or searching for certain terms.

Content-based recommenders have certain advantages in comparison to collaborative filtering approaches. Content-based approaches do not require other users, as each user can build their own profile. Furthermore, it is possible to give explanations on why an item was recommended, as we can look at the features that matched with the user profile. Finally, content-based approaches can recommend new items that do not have any ratings of any users yet.

However, there are also disadvantages. There needs to be enough and well structured information in order to build item representations and user profiles. Content-based approaches also have low serendipity, as they tend to recommend the same kinds of items again and again. Finally, it is hard to make recommendations for new users, as we do not have enough feedback from them in order to build a user profile [LDGS11].

Existing research has shown that content-based approaches can provide good recommendations for contacts that were not previously known yet [CGD$^+$09]. Finding potential new contacts sounds promising in our B2B context. However, in our data set, we do not have extensive information on user profiles. We only have for each participant a list of similar participants, computed by the b2match platform based on free-text entered in profile descriptions. As we do not have access to the profile description itself, content-based recommender systems cannot be applied.

2.2 Implicit vs Explicit Feedback

We have already established that there are two main types of feedback: implicit and explicit feedback. Explicit feedback allows users to directly express their interest in certain items by adding them to their favorites, "liking"/"disliking" them, or rating them on a given scale. Implicit feedback instead is generated by the recommender system by analyzing user behavior, such as viewing items or watch duration for movies. Recommender systems can use either type of feedback to learn more about users' preferences, or even combine them.

Each feedback type has different characteristics. Implicit feedback has low accuracy, high abundance, usually expresses only positive user preferences and is measured in relative numbers. Explicit feedback has high accuracy, low abundance, can express positive and negative preferences and is measured in absolute numbers [JSK10].

Explicit feedback is often used in traditional recommenders, where items are recommended to users. An example for such a system is recommending movies [MAL$^+$03]. Platforms that use such systems generally already collect explicit feedback for other purposes, for example, to list the overall most popular movies. As explicit feedback is already available on such platforms, it can be made use of in recommender systems. Generally, if it is available, explicit feedback is the better option to train a recommender system, as it has high accuracy and can express both positive and negative preferences of users. However, in many cases, explicit feedback is only sparsely available, requiring recommender systems to focus on the more abundantly available implicit feedback. In our case, explicit feedback is not available at all, so we need to entirely rely on implicit feedback.

Implicit feedback is often used in people-to-people recommender systems, as platforms that connect people to people often do not ask people to rate each other. For example, online dating platforms do not ask people to rate other people, as it would not be socially acceptable to do so. Instead, recommender systems for online dating [PRC$^+$10b] rely on implicit feedback (positive interactions) between people. In our data set, implicit feedback consists of information about who visited someones profile, who sent a message to whom, who bookmarked whom and who booked a meeting with whom. Relying only on positive interactions has the downside of being potentially inaccurate, as a positive interaction does not mean that the person was interesting. However, positive interactions have the upside of being vastly available. The use case of online dating is similar to our data set, where we only have implicit feedback. In our case, the signals are scored by domain experts from the company that provided the data set.

2.3 Reciprocal Recommender Systems

Traditional recommender systems consider users and items as separate entities, and attempt to recommend items to users, which is a one-sided form of recommendation. In a matchmaking context, however, the items are other users. Such recommender systems are called people-to-people recommender systems. For such recommender systems, we can directly use existing methods and just apply the same set of users for both, users and items. However, this simple approach may have the downside of recommending person A to person B because person A is very much interested in meeting B, but person B has no interest at all to meet person A. In the end, a meeting between these persons will not happen, because there is no mutual interest between them.

Reciprocal recommender systems take into account preferences of both sides to recommend users to other users [NP19]. An existing recommender system can be extended to a reciprocal recommender system by adding information about mutual compatibility [Pal20]. There are various existing applications of reciprocal recommender systems, where the focus is more on reciprocal matchmaking instead of a one-sided recommendation. For example, RECON [PRC$^+$10a] is a reciprocal recommender system used for online dating.

There have also been applications of reciprocal recommender systems for matching job applicants to companies, such as iHR+ [WYJY15], and for matching students of Massive Open Online Courses (MOOCs) with each other [PSZ17]. Further applications include matchmaking friends through TV characters that best represent them [BHHM12], and patient-doctor matchmaking in primary care [HJdT+18]. A generalized framework for reciprocal recommender systems has also been developed [LL12].

A reciprocal recommendation approach seems interesting for our B2B context, as it is very similar to applications such as iHR+ and patient-doctor matchmaking. However, there is no research yet on using reciprocal recommendations in a B2B matchmaking context. The advantage of making a reciprocal recommendation would be that the other person is more likely to accept a meeting request if they are both interested in meeting each other. A downside of reciprocal recommendations is that we may not have enough data from both users to make a meaningful prediction in both directions, resulting in an overall worse quality of recommendations. Additionally, making bi-directional predictions for each user-user pair results in worse runtime performance of the recommender system.

2.4 Hybrid Recommender Systems

All previously mentioned recommender systems suffer from the cold start problem, where it is hard to recommend items to users who do not have many signals yet. Hybrid recommender systems combine content-based approaches with collaborative filtering [GM09]. Some approaches to solve this issue focus on increasing the quality of recommendations for users with few signals [BOHB12]. However, these approaches do not offer a solution for when a user has no signals at all yet, which is often the case in our B2B context. Others attempt to classify users into groups to do a recommendation based on the group [LKH14]. In the case of heterogeneous B2B events, however, it is hard to find explicit groups that will work well for the various kinds of events. Another approach is to group users based on demographic data [LVLD08], which is only possible if structured profile information is available. In our data set, we do not have structured profile information available, so there is no possibility to create demographic groups.

In our B2B context, participants, after creating their account, do not have any signals at all yet, and only unstructured text as profile information, which is a unique challenge to recommender systems. We attempt to find implicit groups and meeting patterns by looking at similar participants from the same event using our hybrid recommender system, which makes recommendations for similar participants that already have signals and then serves those recommendations to participants with no signals.

2.5 Summary

Recommender systems are a widely researched subject, with collaborative filtering systems learning about user and item groups based on ratings, and content-based systems using structured information to make a recommendation. However, collaborative filtering systems have bad runtime performance if both the set of users and items is large, and content-based systems require structured information on users and items. In our case, users and items are the same, large set, and structured information on users is not available. As a result, matrix factorization based models seem to be the best fit. People-to-people recommenders have previously used collaborative filtering to find already known contacts, and content-based approaches to find new contacts. However, in our B2B context, we do not want to find already known contacts, and do not have the structured information required for a content-based approach. In our research, we thus want to compare the performance of matrix factorization, collaborative filtering (k-NN), and co-clustering models for recommendation **(RQ1)**.

Reciprocal recommender systems that take into account preferences of both sides have been widely used for people-to-people recommender systems. However, using such a system in a B2B context has not been approached yet. The B2B context provides unique challenges, such as matching people with various different interests, which has not been the case for existing people-to-people recommenders, such as online dating. In our research, we want to explore how adding reciprocity in our prediction changes the quality of recommendations **(RQ2)**.

Existing hybrid recommender systems either make use of structured profile data to create groups of users, or attempt to improve the performance for users with few signals. However, a situation where only unstructured profile data and no signals exist for some users has not been tackled yet. We thus explore to what extent such a hybrid recommender system increases the overall quality of recommendations **(RQ3)**.

In general, all these approaches have not been used in the context of matchmaking between representatives of companies yet, as such a use case is missing from a survey of recommender system use-cases [LWM$^+$15]. Furthermore, information provided by the users of online dating and job applications is usually vastly more extensive and more well structured than the descriptions provided by participants of events. As such, a recommender system for the matchmaking of event participants provides unique challenges that have not been tackled before. Finally, b2match offers us a unique opportunity to deploy and evaluate the recommender system in production during events, allowing us to collect data from real world usage of the recommender system. We then analyze this real world usage data to find out how the developed recommender system changed the number of successful meetings **(RQ4)**.

CHAPTER 3

Methodology

3.1 Domain Overview

We are working with an existing platform for event management, where users can book meetings with other participants. The following user interactions are possible on the platform and relevant for the recommender system:

- Browsing a list of participants to find other users that may be of interest to them, clicking on their profiles to visit them.
- Bookmarking users if they are particularly interesting to them.
- Messaging users if they are interested and would like to get to know them more.
- Requesting to book a meeting with them if they would like to meet them during the event.

3.2 Data

The data sets for the recommender system are provided by the company b2match, which specializes in the organization of B2B events by providing a platform for event management. Their platform includes a matchmaking process, where participants of an event can find other people to meet and get to know. They can then book meetings on the platform. The provided data set contains a set of JSON files, one for each event, which consist of the following information:

- *Event* information, such as the id, start/end of the booking phase, start/end of the registration phase, start/end of the meetings phase, and rules about which participant groups can meet each other.

- *Profiles*, containing an id for each user, a flag telling us if the participant wants to be included in matchmaking, a list of similar profiles, and a registration date.
- *Signals*, containing a type, timestamp, and the host and guest participant ids for each signal.

Each event has 3 phases, which can overlap:

- A *registration phase*, where participants can register for the event
- A *booking phase*, where meetings can be booked
- A *meeting phase*, during which the meetings are held

Each signal has a *host*, which is the person requesting the meeting with someone, and a *guest*, which is the person accepting the invitation.

Three data sets were provided in separate folders:

- A *development data set* containing 18 events, used for optimization of the parameters of the recommender system.
- An *evaluation data set* containing 6 events, used for testing the final recommender system performance on a separate set of events.
- A *usage data set* containing 27 events that used the recommender system, for an online evaluation of the recommender system (where signals are tagged with info if the users used the recommender system to find a particular person), including explicit feedback for 9 events.

Each event JSON file contains:

- An *id* of the event
- An *euid* of the event (usually the name of the event, not published for data protection reasons)
- A *registration_begin* timestamp that tells us when the registration phase started
- A *registration_end* timestamp that tells us when the registration phase ended
- A *booking_begin* timestamp that tells us when the booking phase started
- A *booking_end* timestamp that tells us when the booking phase ended
- A *meetings_begin* timestamp that tells us when the meeting phase started

- A *meetings_end* timestamp that tells us when the meeting phase ended
- A *signals* array, which describes interactions between two participants
  - *host_id*: the id of the host participant, the active part, taking the action
  - *guest_id*: the id of the guest participant, the passive part, receiving the action
  - *type*: the type of signal (1 = visit, 2 = bookmark, 3 = message, 4 = meeting requested, 5 = meeting accepted, 6 = user blocked, 7 = meeting feedback, 8 = meeting canceled)
  - *created_at*: the timestamp when the signal was created
- A *profiles* array, which contains information about the participants
  - *id*: unique identifier of the participant
  - *participation_type_id*: the participation type id of the participant, describing which kind of participant they are (e.g. investor, startup founder)
  - *matchmaking*: set to true by default, false if they want to be excluded from matchmaking altogether
  - *similar_profiles*: an array of tuples with (id, similarity score) exported from Apache Solr, used for the hybrid recommender, sorted by similarity descending
- A *rules* array, which describes which participant types can book meetings with which other types
  - *ptype_id*_1: the participation type id of the host
  - *ptype_id*_2: the participation type id of the guest

Figure 3.1: Distribution of signal counts across all events per data set.

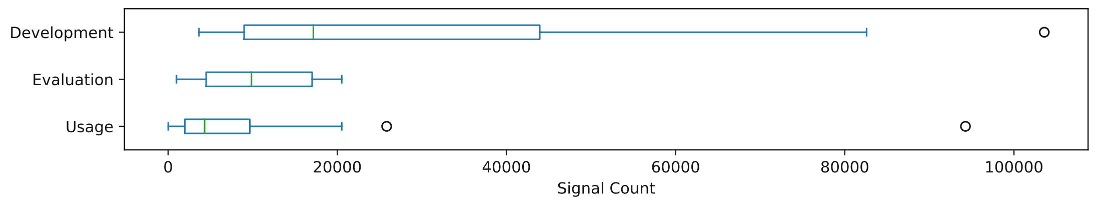


Figure 3.1 shows the amount of signals across all events per data set. The development data set has 3649 signals for the smallest event, 103586 signals for the largest event, and on average 30205 signals per event. The evaluation data set consists of, on average, smaller events, with the smallest event consisting of 989 signals, the largest of 20528, and the mean being 10558 signals. The usage data set has the largest range of events, with a

distribution similar to the evaluation set events, but some outliers with a large amount of signals. The smallest event in the usage data set has only 14 signals, the largest 94268 signals, and the mean number of signals per event is 7779.

Figure 3.2: Distribution of participant counts across all events per data set.

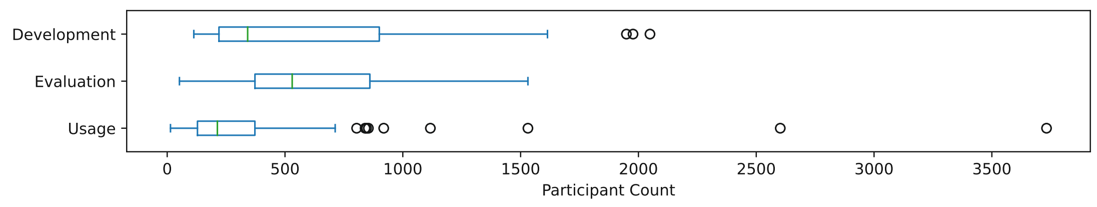


Figure 3.2 shows the number of participants across all events per data set. The development data set has 113 participants for the smallest event, 2049 participants for the largest event, and on average 707 participants per event. The evaluation data set has a similar distribution of participants, with the smallest event having 52 participants, the largest 1531 participants and on average 655 participants. The usage data set has the largest range of events again, with a lot of outliers with many participants. The smallest event in the usage data set has only 14 participants, the largest event has 3732 participants, and the mean is 396 participants.

Figure 3.3: Distribution of event durations in days across all events per data set.

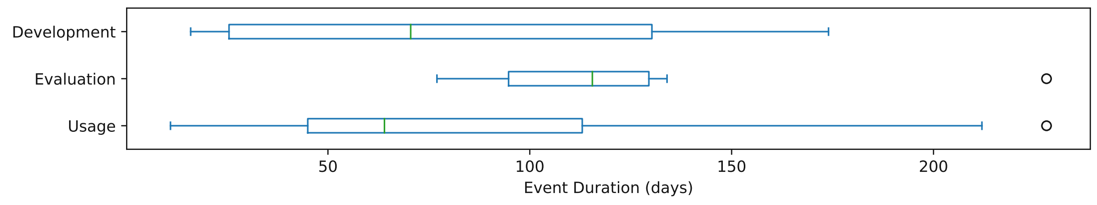


Figure 3.3 shows the duration of events per data set in days. The event durations are computed by taking the earliest (`booking_begin`, `registration_begin` and `meetings_begin`) and latest dates (`booking_end`, `registration_end` and `meetings_end`) from the phases. As some event organizers have set the phases to start many days earlier than the actual events, and sometimes even end many days after the actual ending time of the event, the event durations should be considered with caution. The shortest event of the development data set lasted 16 days, the longest 174 days, and on average events in that set lasted 80 days. The evaluation data set has a smaller range of event durations, with the shortest event lasting 77 days, the largest 228 days, and on average 126 days. The usage data set has a similar range of event durations to the

development data set, with the shortest event lasting 11 days, the longest 228 days and on average 82 days.

Figure 3.4: Average signal counts per event for various signal types across all events per data set.

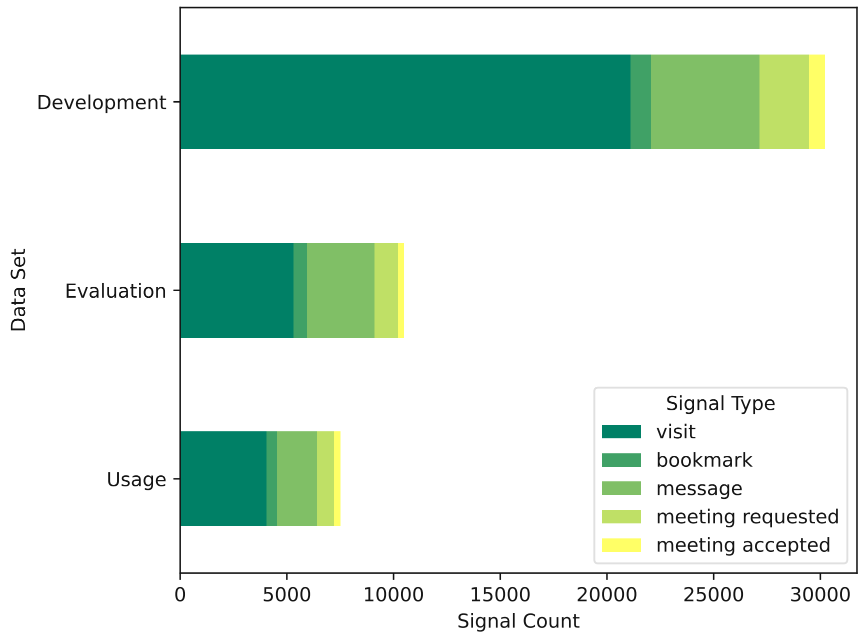


Figure 3.4 shows the average number of signals per signal type across all events per data set. As can be seen, the development set events have, on average, the largest amount of signals, but specifically they have a lot of "visit" signals. In the evaluation and usage data sets, there are on average less signals, but relatively more of the other signal types. The usage set events have, on average, the lowest amount of signals.

Figure 3.5: Relative distribution of signal types across all events per data set.

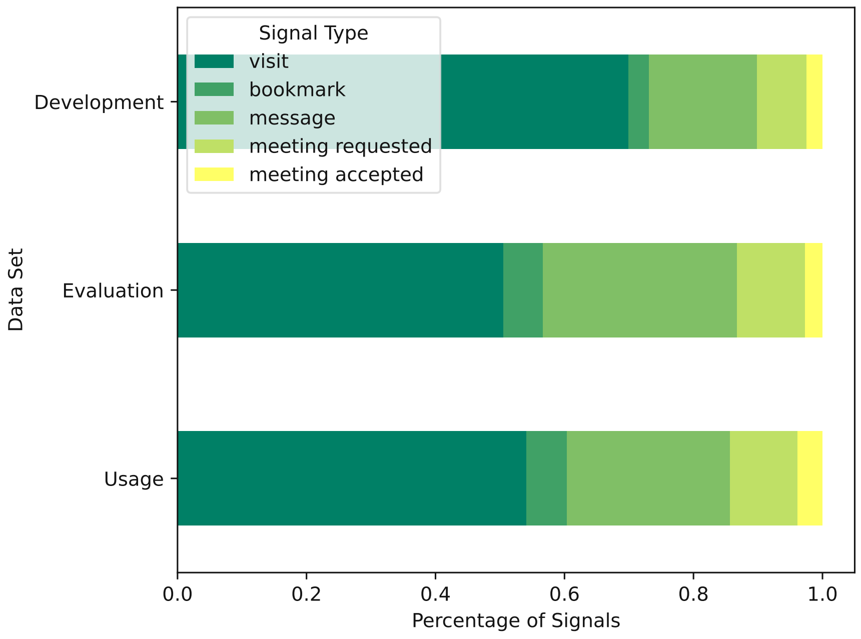


Figure 3.5 highlights the average distribution of different signal types across the different data sets. As can be seen, the evaluation and usage data sets have relatively fewer "visit" signals, but relatively more "message" signals and also slightly more "meeting requested" signals. In the usage data set, a larger percentage of "meeting accepted" signals can also be seen.

3.2.1 Preprocessing

In initial experiments, we used direct exports from the b2match database tables and preprocessed them into the aforementioned signals format. However, in later iterations, to improve performance and increase flexibility of the recommender system, we moved this preprocessing to the b2match backend, which now directly sends a request to the recommender API with the correct signals format.

We still do preprocessing on the signals, though, to convert them into an internal format used by the recommender engine:

- We create a set for each host, when they interacted with a guest (one directional relationship).
- We create a set for each participant, when they interacted with or were interacted with by any participant (bi-directional relationship).

Signal	**Rating**
visit	1
bookmark	2
message	3
meeting requested	4
meeting accepted	5

Table 3.1: Conversion table for converting signal types to a numeric rating

- We create a list of combinations to be skipped for the prediction, which excludes users that have already messaged, blocked, bookmarked each other or already had a meeting booked.

After this preprocessing step, we convert the signals to ratings. Each signal gets a certain score, according to Table 3.1. Then, the highest rating between a certain host and guest is selected as the score for that pairing of users. We then use these ratings to train the algorithms and evaluate their performance.

We developed the mapping from signals to ratings together with 3 experts from the company b2match. We started by defining the business success criteria (successful meetings) as the highest rated signal. Then, we went backwards through their customer journey to rate the other signals. The typical customer journey on their platform is as follows: A participant *visits* the profile of someone and *bookmarks* them if they are interested in meeting them later. Then, they send a *message* to them, asking them follow up questions to see if they are interesting to them. Finally, they *request a meeting*, which is then *accepted* and thus considered a successful meeting by the platform.

3.3 Baseline Models

In this section we describe the baseline models used to evaluate our models against. The first two baseline models are random lists and normal distributions. At the moment, the platform just shows the full list of participants, so a model would be considered suitable if it at least outperforms these baselines. The popularity baseline model shows the most popular participants first, and a good model should not just learn to recommend the most popular participants, but instead recommend personalized lists to each participant. The similarity baseline model is a simple version of a personalized recommendation, returning the participants most similar to the user. We expect this baseline not to perform particularly well on our data set, as in a B2B context, we would rather meet someone who complements us, versus someone who is very similar to us. For example, investors do not want to get recommendations to meet other investors, but instead want to meet startup founders. Nevertheless, our evaluated models should outperform this baseline to be considered suitable, otherwise we could simply return the most similar users instead of making a more personalized recommendation based on implicit signals.

3.3.1 Random

The random baseline generates for each participant-participant pair a random number from the uniform distribution between the lowest interest score and the highest interest score. As a result, it will return to each participant a randomly sorted list of participants.

We set r_{min} to be the lowest possible rating (interest score), and r_{max} to be the highest possible rating, the predicted rating is then computed as follows:

$$\hat{R}(u,i) \sim \mathcal{U}(r_{min}, r_{max}) \tag{3.1}$$

In our case, $r_{min} = 0$ and $r_{max} = 5$. For simplicity and consistency with existing literature, we use u for hosts and i for guests, even though guests are technically not items, but other users.

3.3.2 Normal

The normal baseline assumes that the distribution of the ratings in the development data set is normal, estimates the mean and variance based on the development data set, and then returns a sample from the normal distribution with the estimated mean and variance.

We define R_{train} as the ratings from the development data set. Then, we can estimate the mean and variance as follows:

$$\begin{gathered}\hat{\mu} = \frac{1}{|R_{train}|} \sum_{r_{ui} \in R_{train}} r_{ui} \\ \hat{\sigma} = \sqrt{\sum_{r_{ui} \in R_{train}} \frac{(r_{ui} - \hat{\mu})^2}{|R_{train}|}}\end{gathered} \tag{3.2}$$

Finally, we can compute a predicted rating for each participant-participant pair as follows:

$$\hat{R}(u,i) \sim \mathcal{N}(\hat{\mu}, \hat{\sigma}) \tag{3.3}$$

3.3.3 Popularity

The popularity baseline first computes for each user their average rating in all participant-participant pairs. Then, when we make a prediction, we return their average rating. As a result, each participant will get recommended the same list of the most popular participants.

3.3.4 Similarity

The similarity baseline uses the list of similar profiles provided in the data set. The list contains tuples of user ids and similarity scores. We take the list of similarity scores and scale it using min/max scaling to be between $r_{min} + 1$ (we add one to avoid predicting 0 for users that are at least somewhat similar) and r_{max}. These user ids and ratings are then returned as recommendations for the given participant. As a result, each participant will get recommended a list of similar participants, with the most similar participants being first in the list.

3.4 Evaluated Models

We evaluated three different recommender system algorithms, with various combinations of parameters. The algorithms evaluated are k-NN based models, Matrix Factorization based models and Co-Clustering. The evaluated variants and parameters of the models are described in detail in the *Offline Evaluation Results* chapter. In this section, we give a general overview of the models.

3.4.1 k-NN

k-NN based models are simple implementations of the user-user and item-item collaborative filtering approaches [Kor10] mentioned in the *State of the Art* chapter.

Baseline Predictors

A very simple algorithm to predict the unknown ratings are baseline predictors, which use statistical means to compute a rating. Baseline predictors are very similar to imputation for filling in missing values. Given a user u, an item i and a baseline prediction $b_{u,i}$, the simplest baseline predictor is $b_{u,i} = \mu$ (statistical mean of the ratings). To make it more specific to the user and item, we can use the mean of users or items: $b_{u,i} = \bar{r}_u$ or $b_{u,i} = \bar{r}_i$, respectively. These simple baseline predictors could be extended by adding baseline predictors for the user and item to the statistical mean of all ratings, as follows:

$$b_{u,i} = \mu + b_u + b_i \tag{3.4}$$

The user and item baseline predictors b_u and b_i can be computed using average offsets by computing effects within the residuals of previous effects [ERK11]:

$$b_u = \frac{1}{|I_u|} \sum_{i \in I_u} (r_{u,i} - \mu) \tag{3.5}$$

$$b_i = \frac{1}{|U_i|} \sum_{u \in U_i} (r_{u,i} - b_u - \mu) \tag{3.6}$$

As can be seen from the formulas, baseline predictors take into account various effects on the general rating mean μ, such as user bias b_u and item bias b_i, to get an estimate for the potential rating of a given user on the given item.

User-User Collaborative Filtering

The main idea of user-user collaborative filtering is to, for a given user, find other users whose behavior is similar in terms of ratings. Then, we can use their ratings to predict what items the given user might like. As a simple example, if user u_2 and user u_3 both agree on their ratings for item i_3 (4 stars), user u_3 rated item i_2 with 5 stars, but user u_2 has not rated item i_2 yet, we can assume that user u_2 will likely rate item i_2 highly too. In user-user CF, the predicted ratings are also weighted by the degree of agreement with other users.

For user-user CF, in addition to the ratings matrix $\mathbf{R}$, we need a similarity function $s : U \times U \to \mathbb{R}$ to compute the similarity between two users. We then use s to choose a neighborhood $N \subseteq U$ for user u. This neighborhood is then used to compute a weighted average rating using the similarity as weights:

$$p_{u,i} = \bar{r}_u + \frac{\sum_{u' \in N} s(u,u')(r_{u',i} - \bar{r}_{u'})}{\sum_{u' \in N} |s(u,u')|} \tag{3.7}$$

By subtracting each users' average rating $\bar{r}_{u'}$ from their rating of a given item, we take into account the user-specific rating bias. Some people might generally rate items highly, while others will give average or low ratings. By using the difference to their average rating, we make sure to take into account only when their rating differs from the given user mean. For example, if someone generally rates items with 4 stars, and then gives an item 5 stars, it does not have as much significance as when someone who generally rates very low (average of 2 stars) gives an item 5 stars.

For the similarity function, various statistical correlations can be used, such as Pearson correlation, Constrained Pearson correlation, Spearman rank correlation, or Cosine similarity [ERK11].

Item-Item Collaborative Filtering

In contrast to user-user CF, item-item CF uses item similarity instead of user similarity. This solves a performance issue with user-user CF, where a growing number of users increases the computation complexity of user similarity, which is at best a $O(|U|)$ operation.

In situations where there is a high user to item ratio, an item-item CF system significantly increases performance, as item similarity is unlikely to change through the rating of a single user [LSY03].

The similarity function used in item-item CF is $s : I \times I \to \mathbb{R}$. After using the similarity function to get a set S of the k most similar items to a given item i, we can predict the rating as follows:

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j) r_{u,j}}{\sum_{j \in S} |s(i,j)|} \tag{3.8}$$

For the MovieLens data set, $k = 30$ resulted in a good performance of the recommender system [SKKR01].

However, equation 3.8 might result in negative ratings, which means that after ordering by rating we still get the most relevant items, but the ratings themselves might not make sense. This is due to the fact that the similarity function might return a negative result, which then gets multiplied with the ratings. To solve this issue, we can include the baseline in our equation:

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j)(r_{u,j} - b_{u,i})}{\sum_{j \in S} |s(i,j)|} + b_{u,i} \tag{3.9}$$

In equation 3.9 we now subtract the baseline for a given user u on the item i from the ratings of similar items, and add it again at the end.

3.4.2 Matrix Factorization

Matrix factorization models map users and items to a joint latent factor space of dimensionality f, where user-item interactions are modeled as inner products within the space. First, we compute two vectors: $q_i \in \mathbb{R}^f$ for each item representation in the latent vector space, and $p_u \in \mathbb{R}^f$ for each user representation. The vectors represent the values of each factor, which can be positive or negative. As such, we can measure properties of items using the given factors, as well as user interest in those properties, using the same vector space. The dot product between user and item vectors $q_i^T p_u$ then represents the predicted rating of user u on item i:

$$\hat{r}_{ui} = q_i^T p_u \tag{3.10}$$

We can also add biases to the matrix factorization model, as follows:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \tag{3.11}$$

While predictions are easily computed in this model, mapping each item and user to their respective factor vectors is the biggest challenge of this model. Such a problem is very similar to singular value decomposition (SVD), an already well researched technique to identify latent semantic factors in information retrieval. However, a major difference between matrix factorization and SVD is that conventional SVD requires a rather complete matrix, and thus might lead to overfitting to the few values that are available. An initial solution to this problem was to impute the missing ratings and thus turn the ratings matrix into a dense matrix, which might distort the data and result in higher computational complexity due to the large amount of data. Recently, matrix factorization models have shifted towards fitting a regularized model using the regularized squared error on the set of known ratings [KBV09]:

$$\min_{q*,p*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2) \tag{3.12}$$

Or, alternatively, we can fit a regularized model with biases:

$$\min_{q*,p*,b*} \sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_u - b_i - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2 + b_u^2 + b_i^2) \tag{3.13}$$

Equations 3.12 and 3.13 show the optimization problem of training such a regularized model, where κ is the development data set containing user-item pairs (u, i) with known ratings r_{ui}. By regularizing the learned parameters, their magnitudes get penalized through the regularized term, controlled by the constant λ. As we now have an optimization problem, learning the vectors q_i and p_u for each item and user can be done using existing approaches to optimization problems, such as stochastic gradient descent (SGD) and alternating least squares (ALS).

Stochastic Gradient Descent (SGD)

In the Netflix prize competition, Simon Funk used a SGD optimization of the regularized model from Equation 3.12. The algorithm iterates through all ratings from the development data set κ as follows:

- Predict a rating $\hat{r}_{ui}$ with the current model and compare it to the true rating r_{ui}: $e_{ui} \overset{def}{=} r_{ui} - q_i^T p_u$

- Modify the model parameters q_i and p_u by a factor of γ in the opposite direction of the gradient:
  - $q_i \leftarrow q_i + \gamma * (e_{ui} * p_u - \lambda * q_i)$
  - $p_u \leftarrow p_u + \gamma * (e_{ui} * q_i - \lambda * p_u)$

Alternating Least Squares (ALS)

ALS builds upon the idea that if we have one of the solutions of the optimization problem (either p_u or q_i), the optimization problem becomes quadratic and can be optimally solved. ALS fixes one of the parameters and computes the other, then keeps alternating between them until Equation 3.12 converges.

Generally, SGD is easier and faster than ALS, but ALS can be parallelized more easily and performs better when the training data is dense [KBV09]. As our data set is sparse and we parallelize by recommending for each event on a separate instance of the recommender system, we use SGD as our method of computing the model.

3.4.3 Co-Clustering

While Matrix Factorization models provide high accuracy, one of their main downsides is that they are computationally expensive. Co-Clustering aims to provide good accuracy while being much faster. Like k-NN, Co-Clustering uses neighborhoods, but it instead creates multiple clusters, C_u for users, C_i for items, and co-clusters C_{ui} for each user-item cluster combination. The average ratings $\overline{C_u}$, $\overline{C_i}$ and $\overline{C_{ui}}$ of the clusters are then computed. Finally, the rating is predicted as follows [GM05]:

$$\hat{r}_{ui} = \overline{C_{ui}} + (\mu_u - \overline{C_u}) + (\mu_i - \overline{C_i}) \tag{3.14}$$

3.4.4 Reciprocal Recommender Systems

To introduce reciprocity in a people-to-people recommender, we combine the preferences of both users involved (from user A to user B and vice versa). To do this, we first need to compute the individual scores from user A to user B and from user B to user A. Then, aggregation functions are used to combine the results. Various aggregation strategies can be used, such as the arithmetic, geometric and harmonic means, as well as the uninorm. In the literature, the harmonic mean is generally used for reciprocal recommender systems [NP19].

The *arithmetic* mean of two scores $r_1, r_2 \in [0, 1]$ is defined as:

$$M(r_1, r_2) = \frac{r_1 + r_2}{2} \tag{3.15}$$

When talking about averages, it is usually the arithmetic mean that was used. It has the downside of being greatly affected by outliers, making it not very robust. The advantage of such a property, however, is that in terms of reciprocal recommenders, both sides are considered equally by the aggregation.

The *geometric* mean of two scores is defined as:

$$G(r_1, r_2) = \sqrt{r_1 * r_2} \tag{3.16}$$

In business and finance context, this aggregation is usually used, as it works well for percentages that accumulate over time. Due to the multiplication, the smaller value has a greater influence on the end result than the larger value. This might be a desirable property for some reciprocal recommenders, as it means that if user B is not interested in the user A, even if the interest of user A to user B is large, they will not get recommended to each other.

The *harmonic* mean of two scores is defined as:

$$H(r_1, r_2) = \frac{2 * r_1 * r_2}{r_1 + r_2} \tag{3.17}$$

This aggregation also makes use of a multiplication, giving it similar properties to the geometric mean, where the smaller value affects the result more. In the past, this aggregation has been frequently used in reciprocal recommender systems [NP19].

The *uninorm* of two scores is defined as:

$$U(r_1, r_2) = \frac{r_1 * r_2}{r_1 * r_2 + (1 - r_1) * (1 - r_2)} \tag{3.18}$$

The advantage of uninorms is the self-reinforcement property: Two low values will aggregate into an even lower result, while two large values will aggregate into an even larger result. This property might be useful for reciprocal recommender systems, as it boosts recommendations where both users have a moderate preference to each other rather than strong unidirectional preferences.

3.4.5 Hybrid Recommender System

To attempt solving the cold start problem, we use information about similar users to make an indirect recommendation through them for a given target user. To find similar users at other events, we use a similarity function $s : U_e \times U_{\not e} \to \mathbb{R}$ to compute similarity between two users, where U_e is the set of users at event e and $U_{\not e}$ the set of users across all other events in the database. We then:

- Take the top k similar users to a given user $u_e \in U_e$.
- Make predictions for each user $u_{\not{e}} \in U_{\not{e}}$ on their respective events to:
  - Get a set of predicted users from other events $U_{\not{e}}^{pred}$, and
  - do a reverse match by using the similarity function s to find similar users to $U_{\not{e}}^{pred}$ in U_e.
  - This set of similar users then gets the respective predictions from the user $u_{\not{e}}$.
  - If predictions were not possible, exclude them from the list.
  - If we have no users left in the list of potential similar users, prediction is overall impossible for the given user u_e (return NaN).
  - If we have more than a defined number of similar users, select only the top n similar users, where n is called the neighborhood size.

Our hybrid recommender system is essentially a variation of a Content-Collaborative Recommender system, as introduced by [AKY+11].

3.5 Metrics

In order to evaluate the quality of our recommendations we need to define metrics that matter most for the use case. Machine learning models are usually compared by evaluating the set of items of interest against items that are not of interest. We distinguish items that are defined as of interest by the machine learning algorithm (predicted label) from the ground truth (actual label). Let us consider a simple case where labels are either positive or negative, we can then define the possible combinations in a confusion matrix [Alv02]. In a recommender system context, predicted labels are recommended items, and actual labels are relevant items.

		Actual Label	
		Positive	Negative
Predicted Label	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Table 3.2: Confusion Matrix of the possible combinations of labels.

Table 3.2 shows those combinations, where True Positive (TP) and True Negative (TN) are correct predictions of a model, while False Negative (FN) and False Positive (FP) are wrong predictions of a model. We can then use these combinations to define the following metrics:

$$Precision = \frac{\sum TP}{\sum TP + FP} \tag{3.19}$$

$$Recall = \frac{\sum TP}{\sum TP + FN} \tag{3.20}$$

$$Accuracy = \frac{\sum TP + TN}{\sum TP + FP + FN + TN} \tag{3.21}$$

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.22}$$

These simple metrics, however, do not take into account the ordering of items, and only care if they are part of a set or not. To make these simple metrics more useful for recommender systems, we can instead take the N most relevant items and only calculate precision, recall and accuracy based on those. Adding such a cutoff turns all metrics into their $@N$ variants. For example, taking only the top 10 items into account when calculating Precision, is called the $Precision@10$ metric.

More advanced metrics for recommender systems are derived from these simple metrics, and often used in information retrieval, such as evaluating search engines. A popular advanced metric is Mean Average Precision (MAP). To calculate MAP, we first need to define Average Precision (AP). We set a cutoff k to take the most relevant items, then compute the following:

$$AP@N = \frac{\sum_{k=1}^{N} P(k) * rel(k)}{total\ number\ of\ relevant\ documents} \tag{3.23}$$

Where $rel(k)$ indicates if the item at position k is relevant, as follows:

- $rel(k) = 1$ if the item at position k is relevant
- $rel(k) = 0$ if the item at position k is not relevant

For recommender systems with ratings, this $rel(k)$ function can be extended to return the true rating of an item.

This computation of average precision can be repeated for each user, recommending the most relevant items to them, and then computing $AP@N$. Taking the arithmetic mean over all users then results in the MAP [STL11].

All these metrics take into account the relevancy of an item, but do not take into account the ranking. A widely used ranking measure is the Normalized Discounted Cumulative

Gain (NDCG) [WWL+13]. To define NDCG, we first have to define Cumulative Gain (CG):

$$CG(N) = \sum_{k=1}^{N} rel(k) \tag{3.24}$$

Discounted Cumulative Gain (DCG) is then defined by multiplying the $rel(k)$ function with a discounting function. In standard NDCG, this discounting function is defined as the inverse logarithm decay:

$$D(k) = \frac{1}{log(1+k)} \tag{3.25}$$

We can then define DCG as follows:

$$DCG(N) = \sum_{k=1}^{N} rel(k) * D(k) \tag{3.26}$$

To compute NDCG, we compare the DCG of our predicted set to the DCG of the ideal order (IDCG). IDCG is the DCG of the ground truth set. Then, NDCG can be computed as follows:

$$NDCG(N) = \frac{DCG(N)}{IDCG(N)} \tag{3.27}$$

NDCG has the downside of not taking the error into account as much as metrics such as MAP [AR22]. In our case, however, true relevancy is hard to estimate. We do not have explicit ratings during the development of the recommender system. To avoid bias from our estimated implicit ratings, we focus only on NDCG, which mostly takes into account the relative ranking of items to measure the performance of a recommender system.

3.6 Splitting Strategy

For offline evaluation, using historical data and splitting it into multiple sets is most common. For recommender systems, there are the following main data splitting strategies:

- **Leave One Last:** Include all data except the last user-item pair in the development data set and test the model on the last pair. Instead of the last item, the last user session could also be used.

- **Random Split:** Randomly select user-item pairs from the full set for training and testing
- **User Split:** Split the data set by user rather than by signals.
- **User Temporal Split:** Split the data set by selecting a certain time frame for each user for training and another one for testing.
- **Global Temporal Split:** Split the data set by selecting a certain time frame for all users for training and another one for testing.

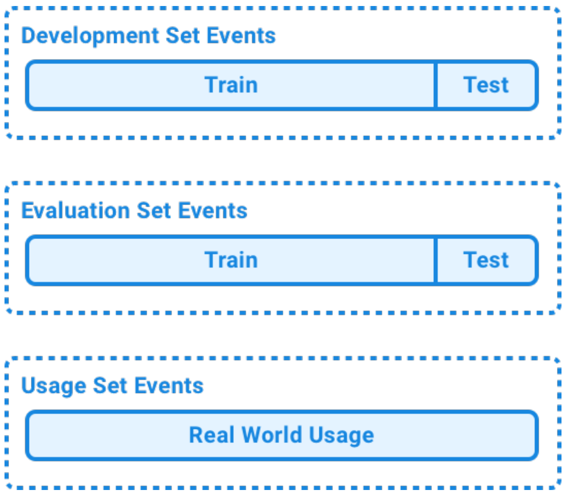
The "Leave One Last" strategy has the issue that it might not reflect the true performance of the recommender system, as the test set is very small. The random split and user split do not work well for recommender systems that deal with temporal data. The user temporal split is not a very realistic scenario, especially when users are recommended to users, because the boundary varies between users, and thus future signals for some users could leak into the model during training of other users. The global temporal split defines a fixed point in time which is shared across all users, which makes it the most realistic scenario, as it is basically simulating the actual historic state of a recommender at some point in time in the past [MMMO20].

For our evaluation, we selected the global temporal split strategy, splitting the signal data from the development data set of events further into two splits: a *train split* and a *test split*. The split time is decided by first defining a beginning and an end of an event, as this is not defined in the data set:

- Calculate an *event_begin* timestamp by taking the earliest date from *booking_begin*, *registration_begin* and *meetings_begin*.
- Calculate an *event_end* timestamp by taking the earliest date from *booking_end* and *meetings_begin*.
- An event begins either when the first signal happens, or when the *event_begin* timestamp is set, whichever is later.
- An event ends either when the last signal happens, or when the *event_end* timestamp is set, whichever is earlier.

This calculation of beginning and ends attempts to model different structures of events. The idea is that we should never take into account anything after the booking phase for training, as making a prediction at this point would be too late (it is not possible to book any more meetings, so the recommender would be useless at this point). However, some events have the timestamps set way in the future, which is why we also take into account the signals, if the last signal happens earlier than the calculated end time. Taking signals into account makes sure we do not select an empty test split.

Figure 3.6: Visualization of the data sets and splits.



After calculating a beginning and an end, we make a split after 80% of the time has passed between them. Everything before the split point is included in the train split, and everything after in the test split. The models are trained on the train split and then their performance is evaluated on the test set. After doing optimizations on the development data set of events, we evaluate the final performance of the optimized models on a new, evaluation set of events. Using a separate set of events for this final evaluation ensures that our results do not report an overfitting of our optimized models to the development data set. For the online evaluation, the recommenders use the full data available (at the time of making a recommendation) during live events.

CHAPTER 4

The CRISP-DM Process

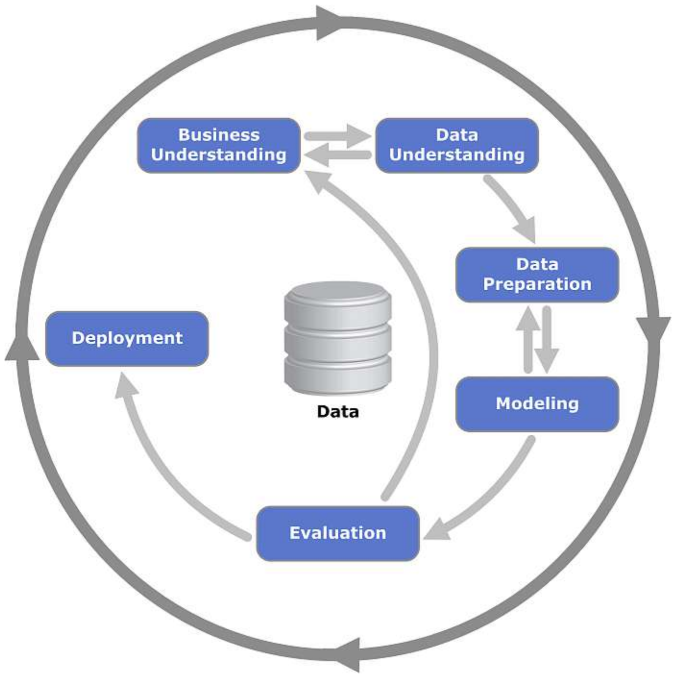
The methodological approach follows the CRoss Industry Standard Process for Data Mining (CRISP-DM) process [WH00], which is an industry and technology independent model for data mining projects. CRISP-DM is an iterative process where the first step is attempting to gain business understanding, through initial meetings with the relevant stakeholders. The next step is attempting to gain data understanding for the data that is already available at this point. Then, this data is prepared and a model is created and evaluated. From this evaluation, we gain more understanding of the business and in turn the data. This process repeats until a well performing model is found with respect to the defined business success criteria. Finally, the well performing model is deployed and used in the real world. In this section, we describe in detail the iterations throughout this project following the CRISP-DM process, and how we gained business and data understanding in each step.

4.1 1st Iteration: Initial Experiments

The project started out with an initial meeting with the company b2match to learn more about their business and which problems they would like to solve using the recommender system. We found out that one of their main business success criteria is increasing the number of successful meetings on their platforms. This increase would be beneficial to all stakeholders on the b2match platform: the participants would be able to meet more and better quality people, which would in turn benefit the organizers as their events will be visited more if the meetings were good quality, which finally benefits b2match, as the organizers will happily use their platform again.

The initial data export was a `meetings.csv` file exported from their database, with the attributes specified in Table 4.1.

Figure 4.1: Diagram of the CRISP-DM process, by Kenneth Jensen [Jen12]



Attribute	Description
`id`	unique identifier of a meeting
`event_id`	unique identifier of an event
`host_participant_id`	unique identifier of a participant initiating the request
`guest_participant_id`	unique identifier of a participant receiving the request

Table 4.1: Attributes of the initial `meetings.csv` export

After getting the initial data export, which contained only a list of meetings between participants, it was quickly clear that this would not be enough data to make a decent quality recommender system. As the data set did not include any granularity, we could only generate pairs that did not meet and combine it with the existing data set of pairs that did have a meeting. As expected, the models built upon this data set did not perform very well, as a meeting not happening between two participants does not mean that they were not interested in meeting each other. This was a key learning from this first cycle, which also lead to more business understanding: If there was no meeting between two participants, we cannot say that they were not interested in meeting in the first place.

4.2 2nd Iteration: Using Meeting Status

In the second cycle, armed with this new knowledge, we decided to add more information about the meetings to the export, such as if a participant requested a meeting with someone else, but it was never answered, or if the other person declined the meeting. Unfortunately, this also turned out not to help the model, as even declining a meeting should not be treated as negative feedback. For example: maybe they were both interested in meeting, but just did not have time for it. As a result, we decided that going forward we should not be treating such cases as negative feedback, but rather focus on the signals of interest that we do have, specifically:

- If someone requested a meeting, that means that they do have interest in meeting such a person.
- If someone accepted the meeting request, that means there is interest in meeting such a person in both directions.
- If someone declined a meeting request, we do not know if the person who declined has interest, but we do know that the requester had interest.

From these assumptions, we built a new model, which still did not perform very well, as there were very few meetings in general, so we decided we needed some more information, leading us to experiment with content-based recommender systems and adding profiles for all participants.

4.3 3rd Iteration: Content-Based Recommender

In the third iteration, we used a new dataset, a `participants.csv` export containing information about participants, with the attributes specified in Table 4.2. For a content-based recommender it is very important to find a similarity metric between users. To compute a similarity measure, we need to transform the text into a numeric format. Most of the attributes in the dataset were free text fields, so we first needed to transform them into a numeric format. To facilitate this, we used Bidirectional Encoder Representations from Transformers (BERT) [DCLT18], a transformer-based natural language processing technique.

However, this again did not improve the performance. Additionally, building this similarity model was extremely slow, especially for larger events. The main problem with recommendation quality seemed to be the amount of user feedback, as we still only had few meeting requests. Additionally, during discussions with b2match, we found out that there is a pre-booking and a booking phase at each event, which means that in the beginning of the event, there would be no feedback at all until booking starts. However, when booking starts is already the time when we need the recommendations the most, so we had to figure out a way to include more signals with implicit user feedback to figure out who participants would be interested in meeting.

Attribute	**Description**
`id`	unique identifier of a participant
`event_id`	unique identifier of an event
`job_title`	job title specified in the profile
`organization_name`	organization name specified in the profile
`organization_description`	organization description specified in the profile
`organization_type`	organization type specified in the profile
`country`	country specified in the profile
`business_fields`	business fields specified in the profile

Table 4.2: Attributes of the `participants.csv` export

4.4 4th Iteration: More Signals

After many discussions with b2match, we found out ways to export more signals from the platform, specifically:

- profile visits
- bookmarks
- messages sent
- meeting requested
- meeting accepted

We came up with a rating system to combine these signals into a single "interest score", discussed with the business stakeholders to order them by relevancy to the business success criteria:

- If there is a visit, the score is 1
- If there is a bookmark, it is 2
- If there is a message, it is 3
- If there is a meeting request, it is 4
- If the meeting request was accepted, it is 5, the highest score

We trained a matrix factorization model on this new signals table, and finally managed to get meaningful predictions out of it. However, there was still a problem with predicting participants with few or even no signals - the cold start problem. As, at the time of writing, there was no way to get more signals than the ones we already have, we decided to try something differently. We came up with the idea of a hybrid recommender system, which uses similarity to find similar participants in previous events and do the prediction there.

4.5 5th Iteration: Cold Start Problem

Unfortunately, the hybrid recommender system meant that we needed to compute content-based similarity again. As we already know, this method was previously very slow in the 3rd iteration when we used BERT to compute it. After discussions with developers from the company, we found out that the platform already implemented a search based on Apache Solr. Apache Solr is an open-source enterprise search platform. The search problem also requires similarity measures to be computed between the keywords entered and the items found. As such, this seemed like the perfect system to use for our content-based recommendation.

We decided to implement an export from Apache Solr to give a list of profiles with arrays of ids to similar profiles, ordered by similarity descending. This system is used to make a recommendation for users with few signals. To do this, we use the following hybrid approach:

- If a user has less than a configurable amount of minimum interactions:
  - Get the top N most similar users, which have more than the minimum amount of interactions (if there are no such similar users, return NaN)
  - Make direct predictions using matrix factorization for these users
  - Aggregate the results of these predictions (configurable: weighted by the amount of interactions)
- Otherwise:
  - Make a direct prediction using matrix factorization for the given user

This hybrid recommender system seemed to improve the performance of the recommender on users with few signals, at least for some events.

4.6 6th Iteration: Optimizations

Now that we had a well performing model, we focused on further improving its runtime speed. We found some small bottlenecks in how data was handled internally, and used numpy and plain python lists instead of pandas where feasible. Furthermore, we created indices from profile ids to profiles, and from profile ids to number of interactions, to further speed up some checks. However, the biggest improvement was reducing the number of combinations to be predicted, as these grow exponentially with the number of users at an event. First, we excluded the users that have already bookmarked, messaged or requested a meeting with each other. There was still an enormous amount of potential pairs.

After some discussions with the company, we found out that the platform actually does not allow anyone to just meet with anyone, there are constraints, so-called meeting rules. Depending on the event, only certain user groups can meet with other user groups, such as startup founders meeting with investors, but not investors meeting other investors. Integrating these rules into the recommender system greatly reduced the number of combinations, significantly speeding up the prediction.

Now that our model works and runs fast, we conducted our offline evaluation, the results of which are reported in Chapter 6: Evaluation.

4.7 7th Iteration: Online Evaluation

After our model was evaluated and optimized, we deployed it in production. As we started collecting data on the recommender system, we figured that it would be useful to also ask the users for explicit ratings in terms of stars between 1 and 5 after meetings, to get actual feedback from the users whether the recommendations were good or not. Unfortunately, this was done too late after the deployment of the recommender system, which means that we were not able to collect a lot of data on explicit feedback. Also, we later found out that a lot of participants were not even using the recommender system, meaning that it might need to be used more prominently, such as sending out an email to users with their recommendations.

CHAPTER 5

Implementation

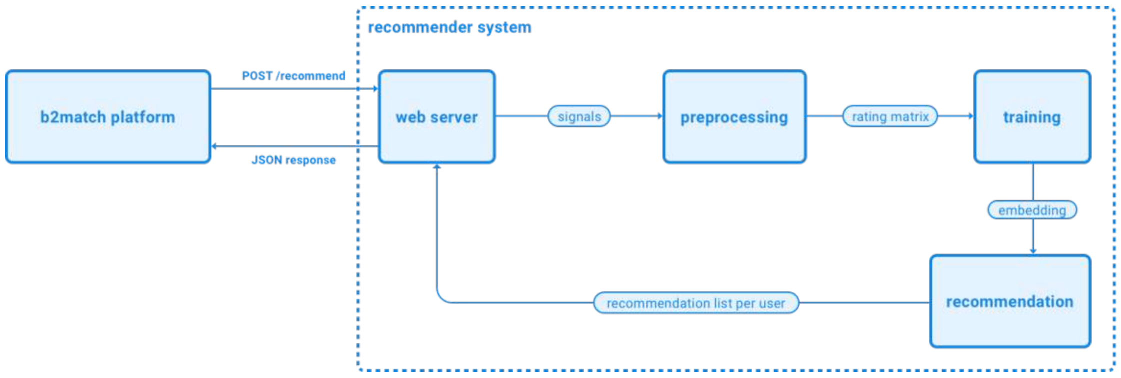
We implemented a recommender system for the matchmaking of event participants, usable across a variety of different kinds of events. The architecture of the recommender system has been designed in a way that is simple, flexible and scalable. The recommender system provides an API accessible via HTTP on a web server. The web server exposes routes for making predictions and evaluating the recommender system performance.

All instances of the recommender systems are stateless, with an optional cache that stores the preprocessed rating tables for speeding up the evaluation process. In production, however, each prediction is made by sending the latest state of the event including all signals to the recommender system via an HTTP request. The recommender system is then trained on the fly and makes predictions for all users. This architectural decision was made in cooperation with the b2match team, because signal data changes frequently and keeping it in sync with the recommender system would be more effort than simply sending all signals every time in a request. Training the recommender system with the matrix factorization algorithm used is also comparatively fast, with most of the time spent doing predictions for all possible user/user pairs.

An initial version of the recommender system was working by direct access to the database, however, we decided to turn it into a stateless service to allow scaling the service by running as many instances of the recommender system as needed. For example, we could run one instance per event, such that prediction can be done for all events in parallel, with no limit in how many instances and thus events we can run in parallel.

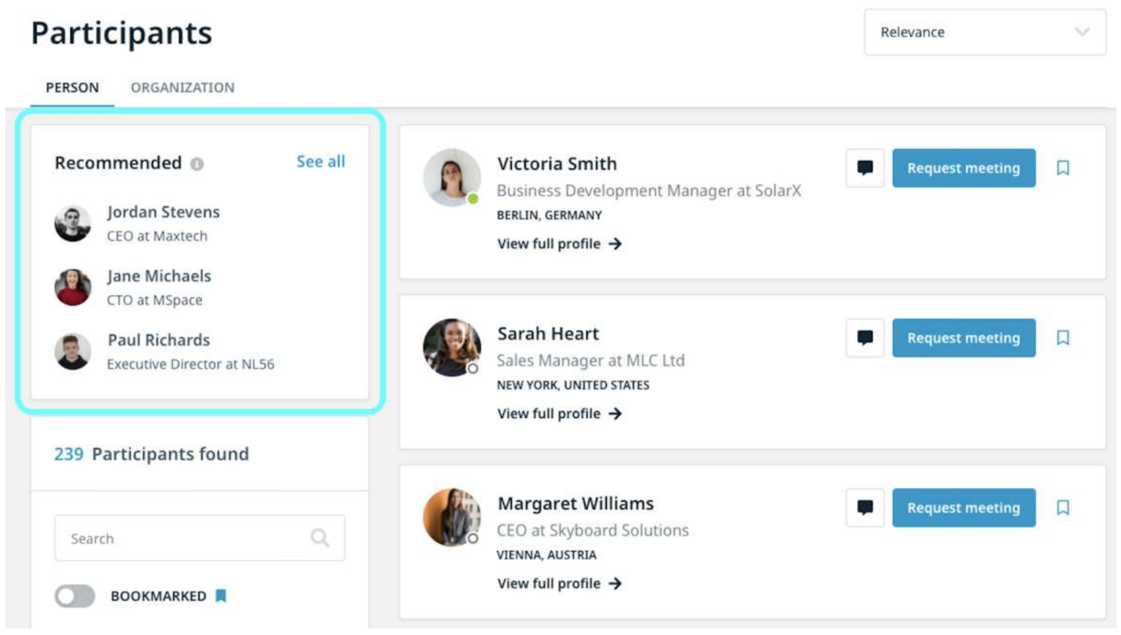
As can be seen in Figure 5.1, the b2match platform directly sends all information of an event, including event info, signals, user profiles and matching rules to the web server of the recommender system, via an HTTP POST request. The recommender system then converts the signals into a rating matrix and stores the matching rules for later use. The training step converts the ratings matrix into user and item embeddings using matrix factorization. Next, all possible user/user pairs (taking into account the matching

Figure 5.1: Visualization of the recommender system implementation



rules) have their scores predicted using matrix factorization. The recommender returns a ranked list of recommendations for each user. Finally, the web server returns these ranked recommendation lists in a JSON response to the b2match platform, which stores the results in its database and uses them to show recommendations in the user interface [1].

Figure 5.2: Recommendations on a b2match managed event page (participants list)


Participants

Relevance

PERSON ORGANIZATION

Recommended See all

Jordan Stevens
CEO at Maxtech

Jane Michaels
CTO at MSpace

Paul Richards
Executive Director at NL56

239 Participants found

Search

BOOKMARKED

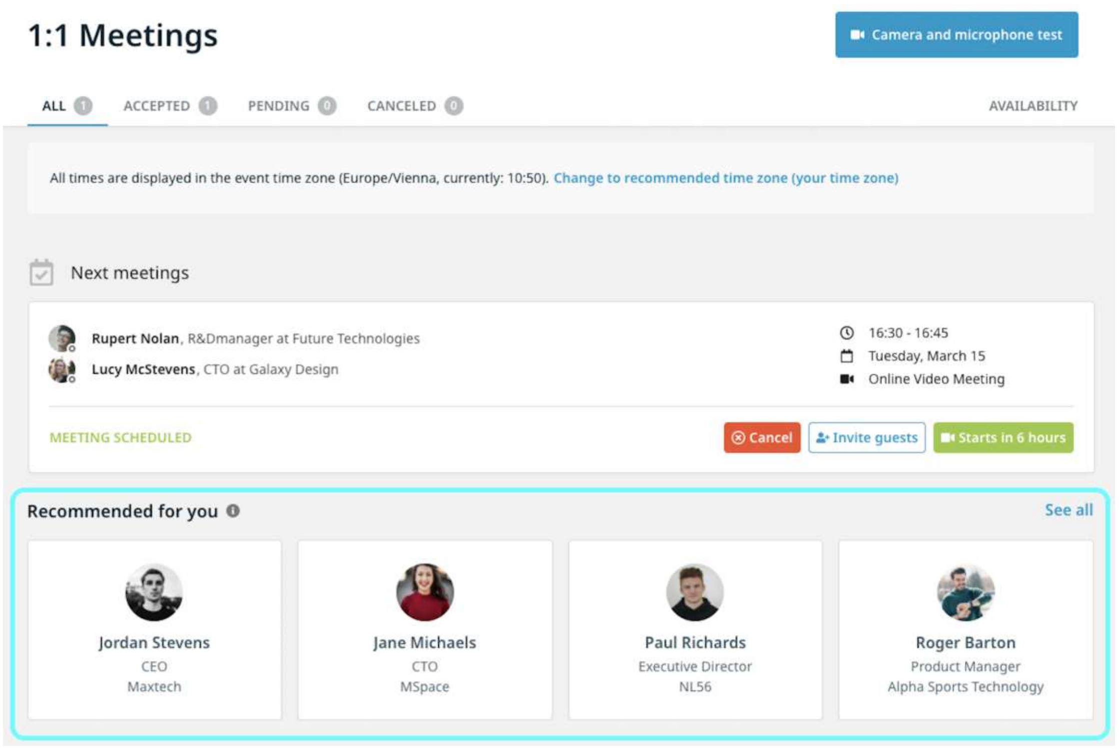
Victoria Smith
Business Development Manager at SolarX
BERLIN, GERMANY
View full profile →
Request meeting

Sarah Heart
Sales Manager at MLC Ltd
NEW YORK, UNITED STATES
View full profile →
Request meeting

Margaret Williams
CEO at Skyboard Solutions
VIENNA, AUSTRIA
View full profile →
Request meeting

[1] https://support.b2match.com/en/b2match-recommender

Figure 5.3: Recommendations on a b2match managed event page (meetings list)



All times are displayed in the event time zone (Europe/Vienna, currently: 10:50). Change to recommended time zone (your time zone)

Rupert Nolan, R&Dmanager at Future Technologies
Lucy McStevens, CTO at Galaxy Design

16:30 - 16:45
Tuesday, March 15
Online Video Meeting

Jordan Stevens
CEO
Maxtech

Jane Michaels
CTO
MSpace

Paul Richards
Executive Director
NL56

Roger Barton
Product Manager
Alpha Sports Technology

5.1 Architectural Design

Signals from events are somewhat small data sets that change all the time. Multiple events may be running at a time, requiring the recommender system to be able to scale with an increasingly higher number of active events. As a result, traditional recommender system implementations, where models are stored and re-trained with new data, would require a high synchronization effort. As such, scaling would mean that more recommender system instances require replication and scaling of a database of models as well. Furthermore, the overhead of storing and loading models is often not worth the effort when data changes all the time, resulting in wildly different models in a short time frame.

To accommodate for this specific use-case, we decided to design the recommender system as a stateless microservice. The recommender system accepts all signals and profiles on each request, builds a new model, and then makes predictions based on it. Such a system can be easily scaled by spinning up multiple instances. The most fine grained split being one instance for each active event, since the recommender can independently process recommendation tasks on a per-event basis.

5.2 Technologies Used

The recommender system was implemented and evaluated using:

- Python 3.10.8 (Interpreter)
- scikit-learn 1.1.2 [PVG+11] (Machine learning and evaluation)
- scikit-surprise 1.1.1 [Hug20] (Recommender system algorithms)
- numpy 1.21.2 (Utility functions for optimized data handling)
- bottleneck 1.3.2 (Utility functions for optimized data handling)
- pandas 1.3.2 (Utility functions for handling data sets)
- scipy 1.7.1 (Utility functions)
- matplotlib 3.4.2 (Graphing library)
- Flask 2.0.1 (Serving the HTTP REST API)
- waitress 2.0.0 (Web Server Gateway Interface [WSGI] for scaling the recommender system)

5.3 API

The recommender system provides endpoints for the prediction of a single event, and an evaluation endpoint, which can evaluate multiple events at once and compute the nDCG metric as well as return information about runtime performance of training and recommendation phases of the recommender system.

As the recommender system is stateless, settings need to be passed to it to configure the recommender system on the fly. The settings object has the following properties:

- `ratings.algo`: the algorithm to use for computing the ratings matrix from the signals, only `simple` is implemented at the time of writing, which contains the mapping of signals to ratings as described in Table 3.1
- `combinator`: the settings for how results from the recommender ensemble are combined
  - `algo`: the algorithm to use for combining the recommender results, `fallback` will fall back to hybrid recommendation if not enough signals are available, `mean` will always compute both a direct recommendation and a hybrid recommendation and take the mean of both

  - `min_interactions`: the number of minimum interactions required for a user to make a direct recommendation, if `fallback` is configured and the user has less than this number of signals, the hybrid recommender system will be used

- `recommender`: the settings for the recommender algorithm to be used
  - `algo`: recommender algorithm to be used,
  - `variant` (optional, required for `mf` and `knn` algos): some recommender algorithms have multiple variants, for example, `mf` has a `svd` variant
  - `args` (optional): arguments for the recommender algorithm, as documented in the scikit-surprise [Hug20] documentation [2] (default: `{}`)
- `enabled` (optional): enabling parts of the recommender system ensemble
  - `recommender` (optional): direct recommendation algorithm (default: `true`)
  - `similarity` (optional): hybrid recommender looking at similar users to make predictions (default: `true`)
- `seed` (optional): to use as a seed for all random numbers generated in the recommender system, makes results reproducible (default: a random seed)
- `reciprocal` (optional): turn on reciprocal recommendation by specifying an aggregation strategies, `off` means no reciprocal recommendation, `arithmetic`, `geometric`, `harmonic`, `uninorm` are possible aggregation strategies for the bi-directional prediction (default: `"off"`)
- `neighborhood_similarity` (optional): how many similar users to take into account for the hybrid recommender (default: `3`)
- `weighted_similarity` (optional): turn on to weigh predictions from the hybrid recommender based on the number of signals (default: `true`)

The evaluation settings can also be changed on the fly, by passing an object with the following properties:

- `split`: strategy used to split the signals into train and test sets, none turns of splitting and trains and evaluates on the full set, `before_booking` uses all signals before the booking phase starts for training and the rest for evaluation, `time` makes a global temporal split based on a percentage of time, `data` makes a global split based on a percentage of signals
- `split_value` (optional, only used for splitting strategies `time` and `data`): the percentage used for training, a number between 0 and 1 (default: `0.8`)

[2] `https://surprise.readthedocs.io/en/stable/`

- use_signals (optional, only used for splitting strategy time): when enabled, takes into account when the first and last signal happened to compute the true start/end time of the whole event (default: false)
- top_n (optional): the number of results to take into account for the evaluation (default: 10)

5.3.1 GET /version

This endpoint returns the current version of the recommender system. At the time of writing, this is v1.0.0.

Response

- version: the current version of the recommender system

Example Response

```
{
    "version": "v1.0.0"
}
```

5.3.2 POST /recommend

This endpoint takes a single event JSON, trains a new instance of the recommender system, makes recommendations for all possible users and returns those recommendations.

Request

- settings: a settings object for the recommender system, as explained earlier in this section
- event: an event object, as explained in the Introduction chapter in the "Data Set" section 3.2

Response

- data: contains all recommendation results from the recommender system in an array of:
  - id: the user id of the host
  - hits: a sorted array of guest-rating pairs for all recommendations made by the system to the host

- `total`: total number of recommendations (host-guest-rating triples) made by the recommender system
- `direct`: number of recommendations made directly for a given host
- `similar`: number of recommendations made by using similar hosts through the hybrid recommender system
- `time_spent`: statistics about the runtime performance of the recommender system
  - `fit`: number of seconds spent to fit the recommender system model
  - `pred`: number of seconds spent to make recommendations for all possible combinations of hosts and guests

Example Request

```
{
    "settings": {
        "ratings": {
            "algo": "simple"
        },
        "combinator": {
            "algo": "fallback",
            "min_interactions": 1
        },
        "recommender": {
            "algo": "mf",
            "variant": "svd",
            "args": {
                "n_factors": 15,
                "n_epochs": 50,
                "biased": false,
                "reg_all": 0.02,
                "lr_all": 0.007,
                "verbose": false
            }
        },
        "enabled": {
            "recommender": true,
            "similarity": false
        },
        "reciprocal": "off",
        "neighborhood_similarity": 1,
        "weighted_similarity": false,
        "seed": 42
        },
        "event": {
            "id": 42,
            "euid": "event-name",
            "signals": [ ... ],
            "profiles": [ ... ],
            "booking_begin": "2021-06-09T12:45:00Z",
            "booking_end": "2021-10-19T16:00:00Z",
            "registration_begin": "2021-06-07T12:58:04Z",
            "registration_end": "2021-10-19T16:00:00Z",
            "meetings_begin": "2021-07-07T08:00:00Z",
            "meetings_end": "2021-10-19T15:00:00Z",
            "rules": [
                {
                    "ptype_id_1": 17544,
                    "ptype_id_2": 17545
                },
                {
                    "ptype_id_1": 17545,
                    "ptype_id_2": 17544
                }
            ]
        }
}
```

Example Response

```
{
    "data": [
        {
            "hits": [
                [
                    1702,
                    3.821179226063678
                ],
                [
                    1721,
                    3.689105788754919
                ],
                ...
            ],
            "id": 1
        },
        {
            "hits": [ ... ],
            "id": 2
        },
        ...
    ],
    "direct": 2839729,
    "similar": 0,
    "time_spent": {
        "fit": 0.6890418529510498,
        "pred": 43.88013005256653
    },
    "total": 4177035
}
```

5.3.3 POST /evaluate

This endpoint takes multiple event objects, makes a split of the signals into a train and test set for each event, then trains a new instance of the recommender for each event based on the train set, makes recommendations for each user, and evaluates the performance of the recommendations based on signals from the train and test set combined.

Request

- `settings`: a settings object for the recommender system, as explained earlier in this section
- `evaluation`: an evaluation settings object to configure the splitting and evaluation, as explained earlier in this section
- `events`: an array of event objects, as explained in the Introduction chapter in the "Data Set" section 3.2

Response

- `events`: array of event objects that were actually evaluated (not skipped)
- `metrics.ndcg`: array of nDCG values per event
- `metrics_avg.ndcg`: average nDCG value of all evaluated events
- `metrics_list.ndcg`: array of arrays of nDCG values for each host per event
- `time_spent`: statistics about the runtime performance of the recommender system
  - `fit`: time spent to train the recommender models per event

- `fit_avg`: average time spent to train the recommender models across all events
- `fit_total`: total time spent to train the recommender models across all events
- `eval`: time spent to evaluate the recommender models' performances per event
- `eval_avg`: average time spent to evaluate the recommender models' performances across all events
- `eval_total`: total time spent to evaluate the recommender models' performances across all events

Example Request

```
{
    "settings": { ... },
    "evaluation": {
        "split": "time",
        "split_value": 0.8,
        "use_signals": true
    },
    "events": [
        {
            "id": 42,
            "euid": "event-name",
            ...
        },
        {
            "id": 43,
            "euid": "event-2-name",
            ...
        },
        ...
    ]
}
```

Example Response

```
{
    "events": [
        {
            "id": 42,
            "euid": "event-name",
            ...
            }
        ],
        "metrics": {
            "ndcg": [
                0.005213288717167683,
                0.07253450307588923,
                0.0,
                ...
            ]
        },
        "metrics_avg": {
            "ndcg": 0.08096421483937885
        },
        "metrics_list": {
            "ndcg_list": [
                [
                    0.005213288717167683,
                    ...
                ]
            ]
        },
        "time_spent": {
            "eval": [
                44.78465795516968,
                ...
            ],
            "eval_avg": 44.78465795516968,
            "eval_total": 44.78465795516968,
            "fit": [
                0.22023415565490723,
                ...
            ],
            "fit_avg": 0.22023415565490723,
            "fit_total": 0.22023415565490723
        }
    }
```

CHAPTER 6

Offline Evaluation Results

6.1 Analyzing Signal Distributions

Before we evaluate the recommender system, we should verify that our split is indeed a realistic representation of a point "in the middle of the event", where the recommender would be most useful. The graphs show the distribution of signals over time, with the registration, booking and meeting phases marked at the top, and a dashed line showing where the train/test global temporal split is made. Everything before the dashed line is included for training, and everything after for evaluating the performance of the recommender system.

In the signal distribution patterns, we detected three distinct patterns:

- **Group A:** Few signals at the beginning of the event, with signals increasing towards the end of the event. Usually, events in this group have a registration and booking phase, and then meetings and the end, when the event actually happens.
- **Group B:** Many signals at the beginning of the event, with signals decreasing towards the end of the event. Events in this group have a lot of users that book meetings at the beginning, before the meeting phase of the events start.
- **Group C:** Events from this group are similar to group A in terms of their pattern, but are much more dragged out over time, having more than 100 days in total, where signals still happen across the whole span of time. Events in this group are long and large-scale events, usually conferences that span over a long time.

Additionally, we have the signals from the evaluation data set, which are called Group D events. Events from the evaluation data set are meant to cover a little bit of every kind of event that we also had in the development data set. Group E events are events where the recommender system was used in the real world during the online evaluation.

Figure 6.1 shows events A1-A8, Figure 6.2 shows events A9-A12, B1-B3 and C1, and Figure 6.3 shows events C2-C4. Figure 6.4 shows events D1-D6. Group E event signals are not analyzed.

Figure 6.1: Signal distribution of events A1-A8 over time with Registration, Booking and Meeting phases marked and a dashed line showing the train/test data split.

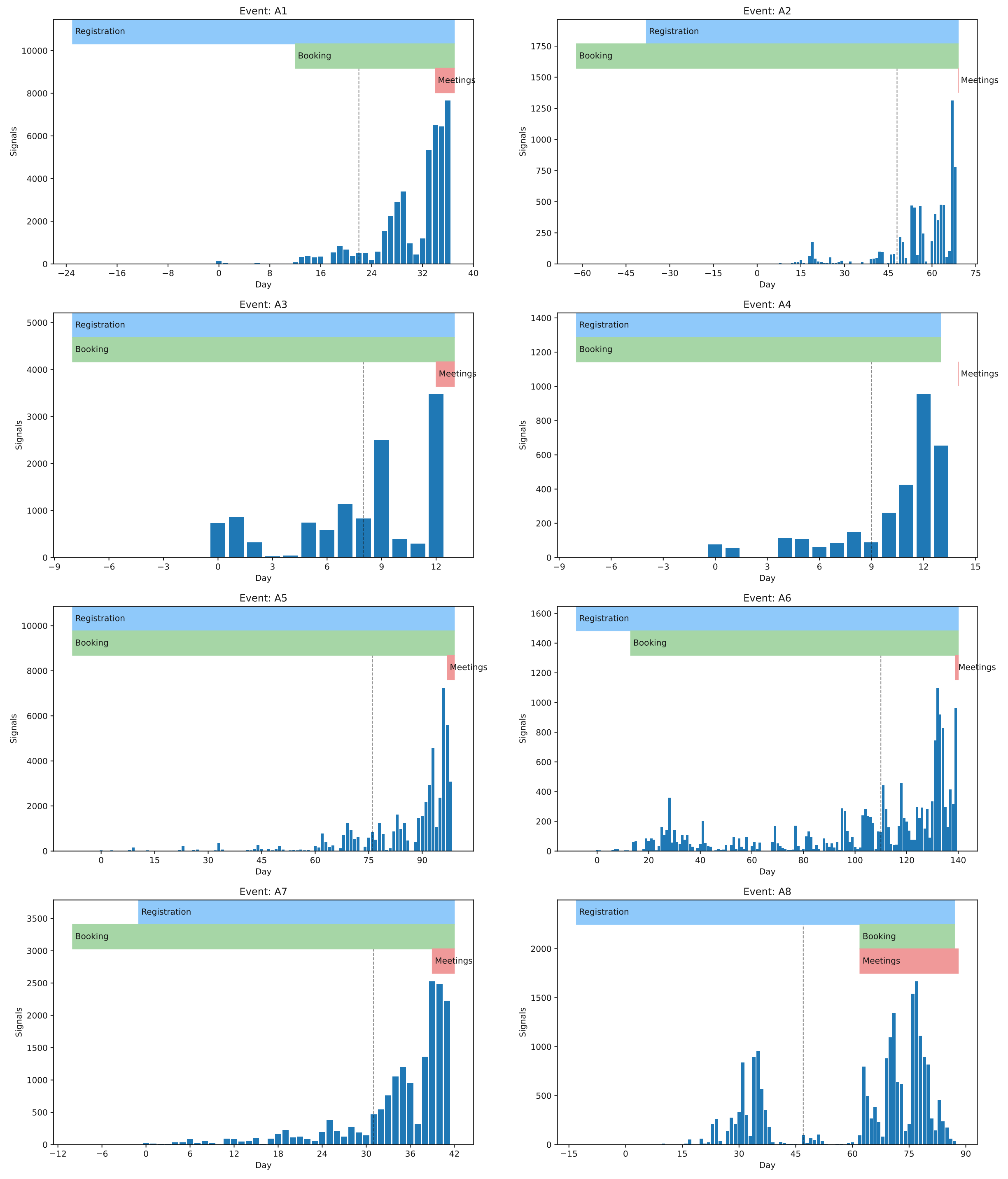


Figure 6.2: Signal distribution of events A9-A12, B1-B3, C1 over time with Registration, Booking and Meeting phases marked and a dashed line showing the train/test data split.

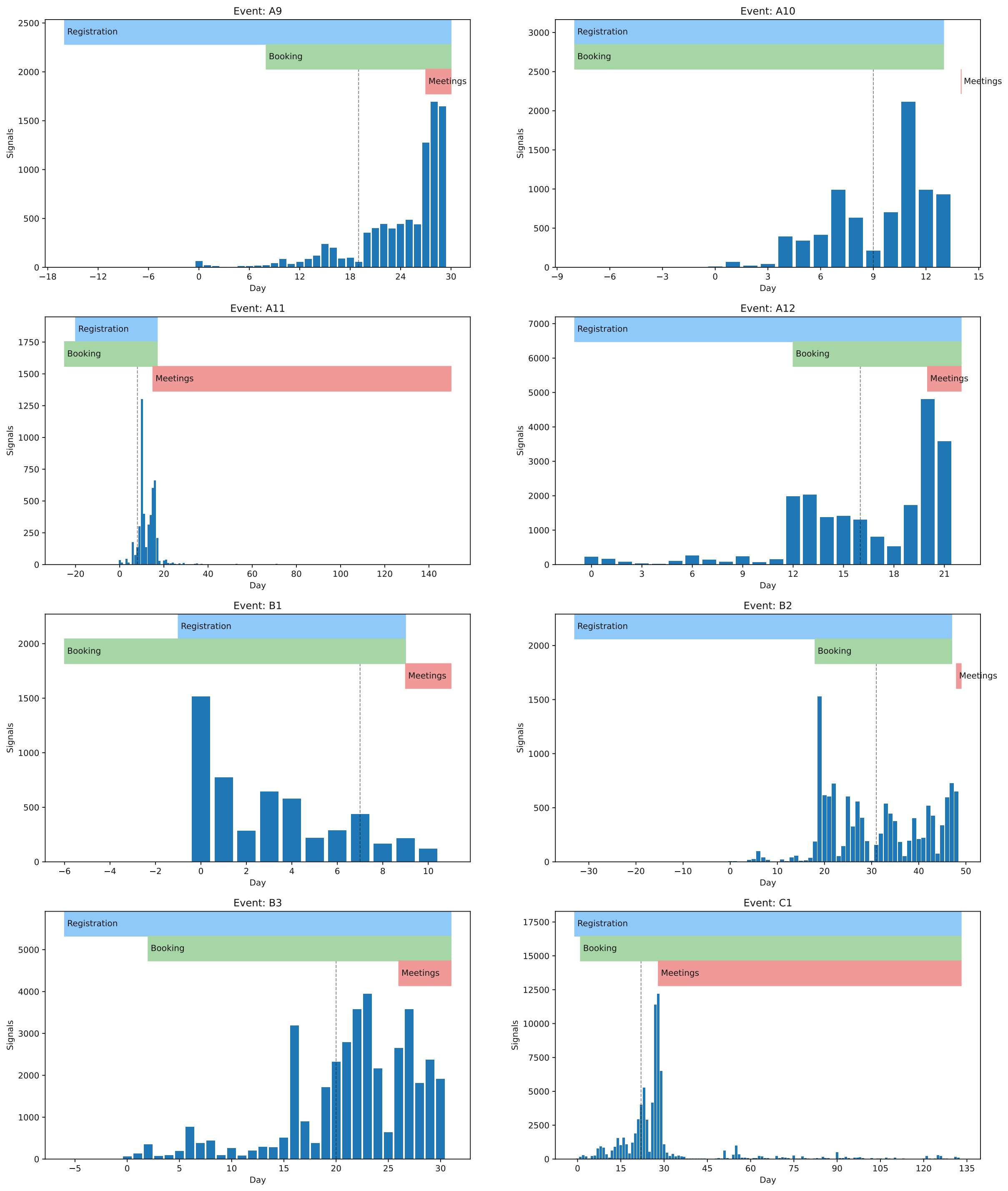
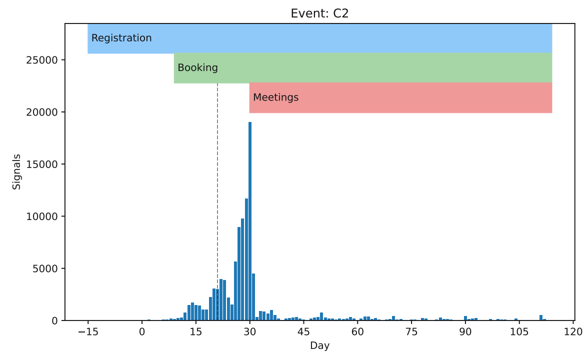


Figure 6.3: Signal distribution of events C2-C4 over time with Registration, Booking and Meeting phases marked and a dashed line showing the train/test data split.



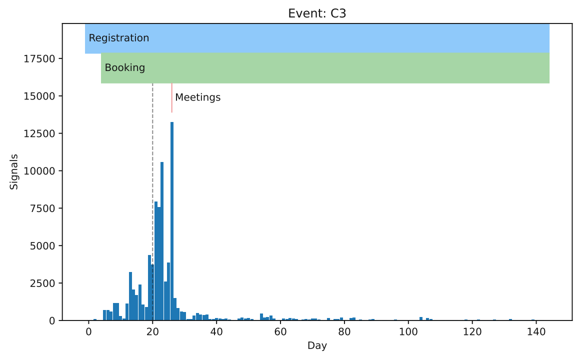
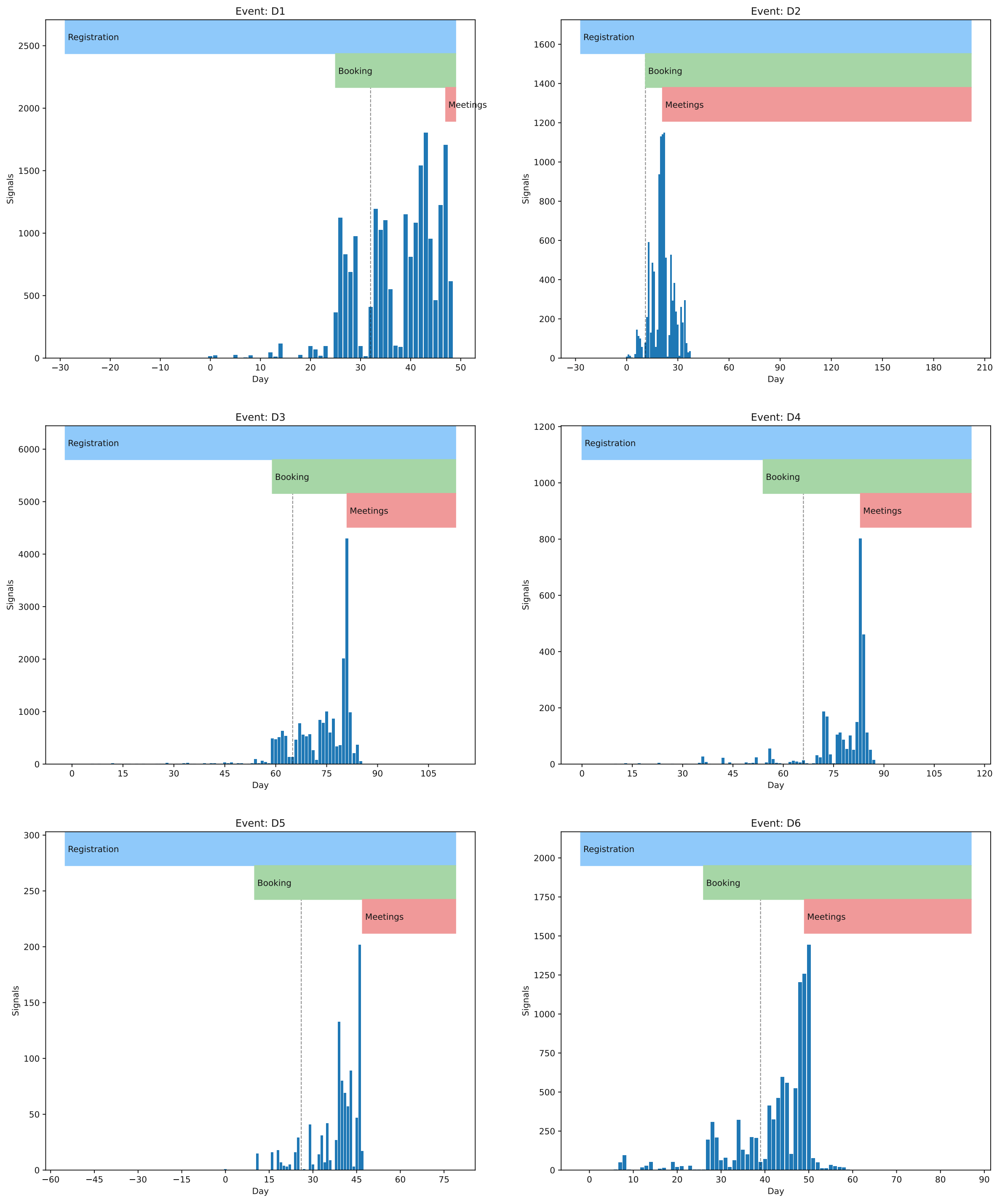


Figure 6.4: Signal distribution of events D1-D6 over time with Registration, Booking and Meeting phases marked and a dashed line showing the train/test data split.



There are some special events in each group, for example event A8 has two "waves" of signals, with the first wave peaking around days 30-40, then very low amounts of signals for a while. Around day 60, the signals start increasing again. Generally, the pattern of A8 is still increasing signals over time, just with a dip in-between. The split here is made after the first wave and long before the booking phase, so there are no "meeting requested" or "meeting accepted" signals in the development data set.

Event A11 is also very long (more than 100 days), making it similar to C events in that way, but there are no signals happening anymore at the end, it seems like the meetings end time was just accidentally set very far in the future. It also has relatively few signals in the development data set, with most signals being part of the test set.

Event B2 is very interesting, because most signals happen right at the start of the booking phase. Event B3 is a bit of a mixture of A events and B events, where signals generally decrease towards the end, but they also increase at the beginning. It might be interesting to see if B3 is similar to A events or B events in terms of recommendation performance.

In further evaluations, we will compare the different groups of events to each other to see if the signal distribution patterns affect the recommender system performance. We expect that the recommender system works well for some kinds of events, and not as well for others. We will also consider the events with special patterns: A8, A11, B2 and B3.

We now evaluate the recommender system with various settings and compare their performances to each other. For all evaluations, all random seeds are set to 42 to keep results reproducible.

6.2 Recommender Algorithms Versus Baselines

First, we focused on direct recommendation, meaning that we directly use the signals from a specific user (host) to make predictions about the potential interest scores of other users (guests). We evaluate different algorithms with their default parameters against baselines.

The following algorithms and default parameters (as defined by scikit-surprise 1.1.1 [Hug20]) are evaluated:

- Co-clustering (`coclustering`) with 3 user clusters (`n_cltr_u`), 3 item clusters (`n_cltr_i`) and 20 epochs (`n_epochs`).
- Basic k-NN (`knn/basic`) with 40 neighbors (`k`), and 1 minimum neighbors (`min_k`).
- SVD Matrix Factorization (`mf/svd`) with 100 factors (`n_factors`), 20 epochs (`n_epochs`), biases enabled (`biased`), learning rate 0.005 (`lr_all`) and regularization term 0.02 (`reg_all`).

	normal	**popularity**	**random**	**similarity**
coclustering	$0.1052 > 0.0822$ (p = 0.0268)	$0.1052 < 0.1178$ (p = 0.2819)	$0.1052 > 0.0817$ (p = 0.1224)	$0.1052 > 0.1051$ (p = 0.4981)
knn/basic	$0.1465 > 0.0822$ (p = 0.0023)	$0.1465 > 0.1178$ (p = 0.1705)	$0.1465 > 0.0817$ (p = 0.0129)	$0.1465 > 0.1051$ (p = 0.0433)
mf/svd	$0.2377 > 0.0822$ (p = 0.0000)	$0.2377 > 0.1178$ (p = 0.0001)	$0.2377 > 0.0817$ (p = 0.0000)	$0.2377 > 0.1051$ (p = 0.0000)

Table 6.1: nDCG@10 and p-values of algorithms with default parameters versus baselines

The following baselines are used for comparison of all evaluations:

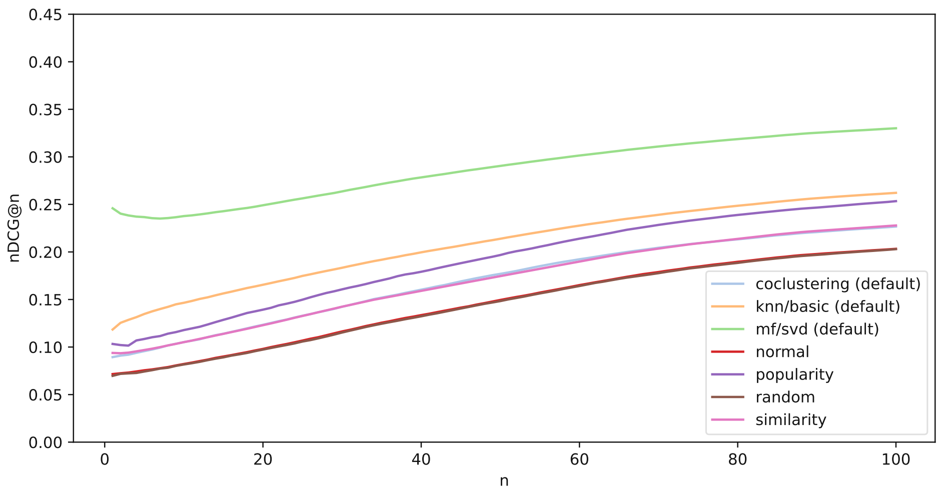
- Samples from a normal distribution of ratings (`normal`)
- Scores based on popularity of users (`popularity`)
- Samples from a uniform distribution between min/max values of ratings (`random`)
- Recommending similar users (`similarity`)

In the following, we present the results of our offline evaluation. We compare the performance of the evaluated methods using nDCG@10. The Tables provide the following information: Algorithms to be evaluated are shown on the left, each row displaying a different algorithm. Algorithms to be compared against (baselines) are shown on the top, each column displaying a different algorithm. The matrix formed by the table compares performance of algorithms in the given row and column. If an evaluated algorithm performs better than the baseline, a ">" is shown between the results, otherwise a "<" is shown. Significant differences in nDCG@10 are highlighted with a background color. If values are significantly better than the baseline, the highlight is green, if they are significantly worse, the highlight is red. p-values lower than 0.05 are considered significant. p-values lower than 0.01 are highlighted slightly stronger. p-values lower than 0.001 are highlighted even stronger.

Table 6.1 shows the results of the evaluation of algorithms with default parameters versus baselines. It can be seen that mf/svd is the only algorithm that, with the default parameters, performs significantly better on average than all baselines. knn/basic performs better than all baselines and significantly better than all baselines except popularity. coclustering performs better than all baselines except popularity and only significantly better than the normal baseline.

Figure 6.5 shows the relevance scores of the recommender algorithms with default parameters versus the baselines across different numbers of results returned from the recommender. The x-axis shows increasing numbers of recommendations and the y-axis shows the nDCG score at that number of recommendations. As can be seen from the figure, mf/svd performs better than all other algorithms including all baselines already.

Figure 6.5: nDCG@n relevance scores of the recommender algorithms with default parameters versus the baselines.



knn/basic also performs better than all baselines, but at high number of results gets close to the popularity baseline. Interestingly, coclustering seems to be clustering similar users together, as it performs almost exactly the same as the similarity baseline. random and normal baselines also perform similarly. Another interesting pattern to see here is that mf/svd seems to return best matches first, as even with just one result, the performance is quite good. It then decreases slightly, and increases again with larger number of results.

6.3 Parameter Optimization Results

After evaluating the algorithms themselves with default parameters, we now do a parameter optimization for all algorithms by employing a grid search [BBBK11], optimizing for nDCG@10. We do the parameter optimization on the development set. We came up with a set of parameters that covers all variants of the various algorithms and adjusting the parameter values by lowering and increasing them from the default value. In the grid search, we are trying out combinations of the following parameters:

- **coclustering:**
  - *n_cltr_u*: 2, 3, 5, 10 (default: 3)
  - *n_cltr_i*: 2, 3, 5, 10 (default: 3)
  - *n_epochs*: 10, 20, 50 (default: 20)
- **knn:**

  - *variant*: basic, means, zscore, baseline (default: basic)
  - *k*: 1, 5, 20, 40, 60, 100 (default: 40)
  - *min_k*: 1, 3, 5 (default: 1)

- **mf:**
  - *variant*: svd, svdpp, nmf (default: svd)
  - *n_factors*: 50, 100, 200 (default: 100)
  - *n_epochs*: 10, 20, 50 (default: 20)
  - *biased*: true, false (default: true)
  - *lr_all*: 0.001, 0.005, 0.01 (default: 0.005)
  - *reg_all*: 0.01, 0.02, 0.03 (default: 0.02)

The following optimal parameters were found:

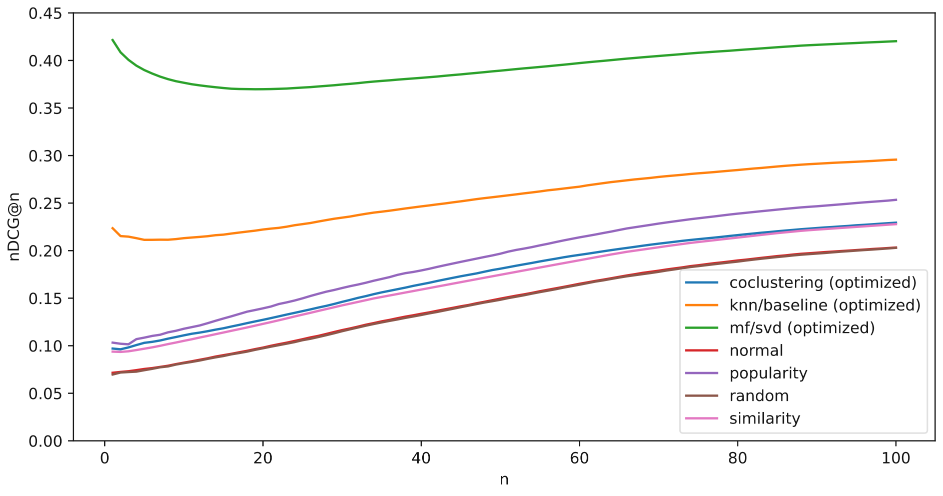
- **coclustering:**
  - *n_cltr_u*: 5
  - *n_cltr_i*: 10
  - *n_epochs*: 20
- **knn:**
  - *variant*: baseline
  - *k*: 5
  - *min_k*: 5
- **mf:**
  - *variant*: svd
  - *n_factors*: 200
  - *n_epochs*: 50
  - *biased*: false
  - *lr_all*: 0.005
  - *reg_all*: 0.01

Table 6.2 shows the performance of the algorithms with optimized parameters versus baselines. We can immediately see that a lot of differences in nDCG are significant now. coclustering still only significantly beats the normal baseline, but significantly

	normal	**popularity**	**random**	**similarity**
coclustering	0.1109 > 0.0822 (p = 0.0010)	0.1109 < 0.1178 (p = 0.3638)	0.1109 > 0.0817 (p = 0.0529)	0.1109 > 0.1051 (p = 0.3704)
knn/baseline	0.2131 > 0.0822 (p = 0.0000)	0.2131 > 0.1178 (p = 0.0007)	0.2131 > 0.0817 (p = 0.0001)	0.2131 > 0.1051 (p = 0.0000)
mf/svd	0.3766 > 0.0822 (p = 0.0000)	0.3766 > 0.1178 (p = 0.0000)	0.3766 > 0.0817 (p = 0.0000)	0.3766 > 0.1051 (p = 0.0000)

Table 6.2: nDCG@10 and p-values of algorithms with optimized parameters versus baselines

Figure 6.6: nDCG@n relevance scores of the recommender algorithms with the best optimized parameters versus the baselines.



now. However, it is still performing worse than the popularity baseline (not significant). knn/baseline and mf/svd both significantly outperform all baselines.

Figure 6.6 shows the nDCG relevance scores across different numbers of results again, but this time with the parameter optimized algorithms. The best algorithms and variants are still mf/svd and knn/baseline, and they both outperform all baselines across all numbers of results. Interestingly, the pattern where the first results performs very well is now visible for knn/baseline as well, and even more strongly visible for mf/svd. coclustering with optimized parameters still performs similarly to the similarity baseline. However, coclustering is generally better than the similarity baseline now, outperforming all baselines except the popularity baseline. Interestingly, the mf/svd and knn/baseline algorithms have a high nDCG@k for the first few top results, then it declines and towards the end there is a rising trend again. This effect might mean that those algorithms are particularly good at predicting the most interesting participants and listing those first.

When adding more users, they seem to add some less interesting participants (according to the ground truth).

Figure 6.7: nDCG@n relevance scores of the recommender algorithms with the best optimized parameters versus the default parameters.

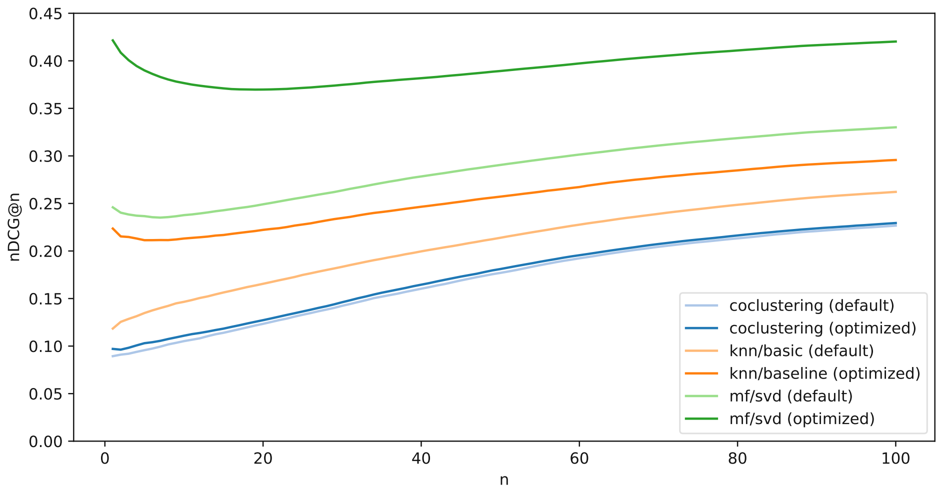
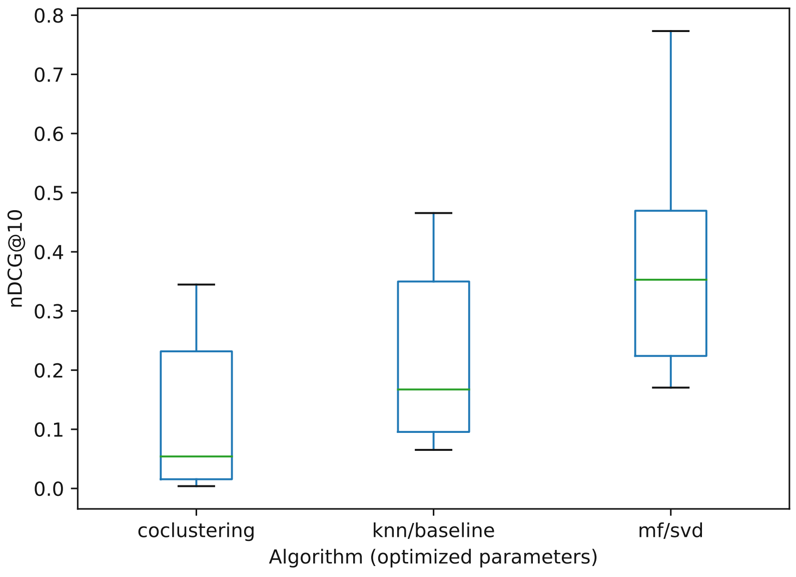


Figure 6.7 shows the nDCG relevance scores across different numbers of results, comparing the algorithms with default parameters to the parameter optimized ones. It can be seen that we managed to achieve an overall improvement in recommendation performance for all algorithms. The largest improvement was achieved for mf, especially with the first results. For knn we still managed to achieve a great improvement, especially by changing the variant from "basic" to "baseline". For coclustering, we only managed to achieve a slight improvement through parameter optimization. It is interesting to see that parameter optimization seems to especially have improved the recommender algorithms for the first results, where it matters most that we provide good recommendations. This makes sense, since nDCG values first results more than lower results through its discounting function. Another interesting fact is that all optimized algorithms still did not perform as well than mf/svd with default parameters, suggesting that matrix factorization algorithms may be best suited for our data set. Interestingly, the non-optimized variant of knn does not have this declining trend at low values of k for the nDCG@k score, while the optimized variant does. This might be because the first results for the non-optimized variants are particularly bad, as the nDCG@k score for these is much lower than the optimized variant.

Figure 6.8 shows the spread of performance of algorithms using the optimized parameters and all parameter combinations via boxplots. It can be clearly seen that even with optimized parameters, coclustering performs the worst on our data set. knn/baseline is slightly better on average and also tops out higher than coclustering, as seen by the

Figure 6.8: Boxplots of nDCG@10 relevance score distribution per algorithm and variant with the best parameter combinations.



whiskers at over 0.4 nDCG. This means that an nDCG of over 0.4 was achieved for at least one event. mf/svd clearly has the best average performance across all events, and also tops out higher than knn/baseline. The lowest performance of mf/svd is also still higher than the average performance of the other algorithms.

Figure 6.9: Boxplots of nDCG@10 relevance score distribution per event for the matrix factorization algorithm with default versus optimized parameters.

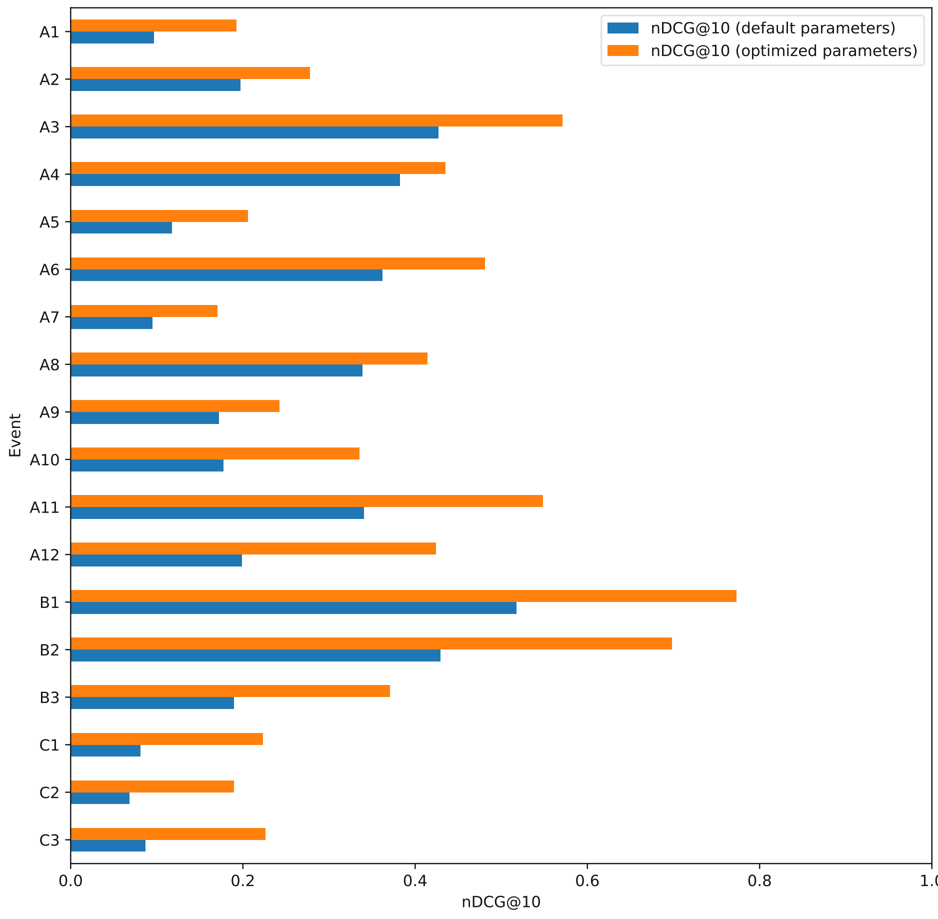


Figure 6.9 shows the performance of the matrix factorization algorithm with default parameters versus optimized parameters. It can be clearly seen that the parameter-optimized version performs much better for all events. This difference is especially large for events that performed particularly bad before optimization, namely all events in the C group (large scale events).

6.4 Comparing Performance Spreads of Events

After analyzing algorithms, variants and parameter combinations, we now look at the performance spreads of events. We attempt to exploratively analyze if there are some events that work particularly well with recommender systems, and others that may be hard to recommend for.

Figure 6.10: Boxplots of nDCG@10 relevance score distribution per event for all algorithms with all parameter combinations.

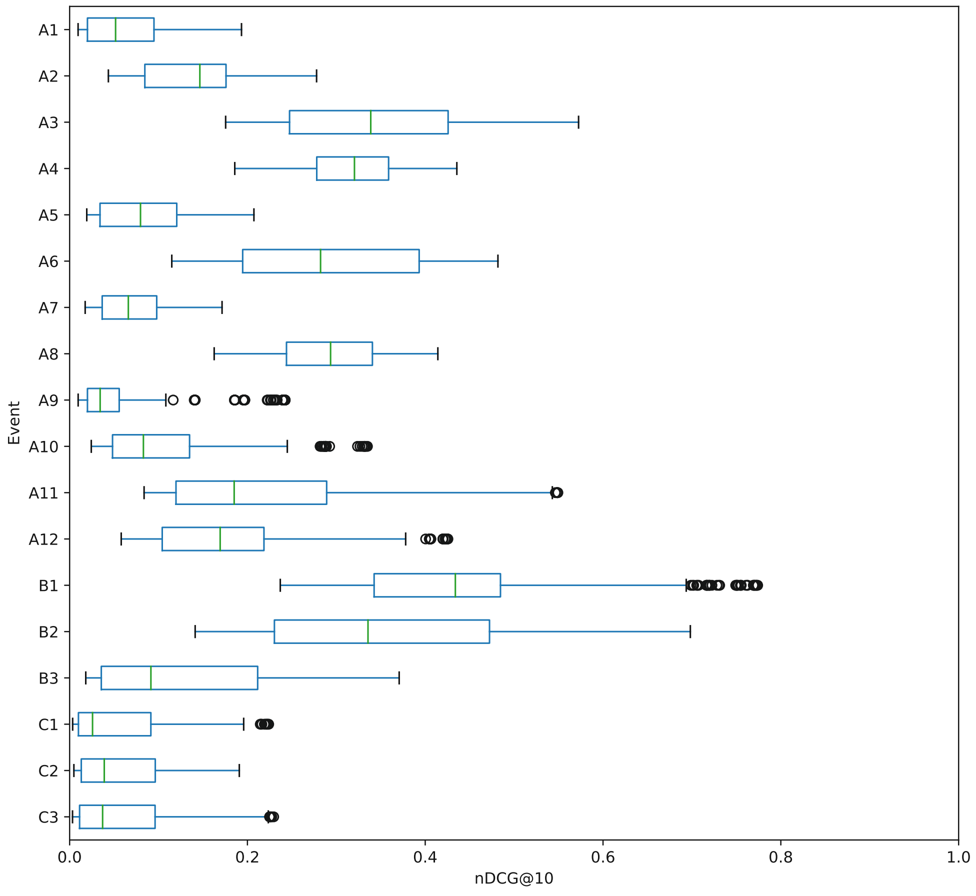


Figure 6.10 shows the nDCG@10 relevance score distribution per event for all algorithms with all parameter combinations. It is interesting to see that the algorithms perform particularly bad overall on events from the C group. This might be because those are long and large-scale events, so most of the data comes after the training split. The outlier A11 has a particularly wide spread of performance, suggesting that some algorithms and parameter combinations were able to deal with the low amount of training data, while others struggled. Overall, there is no consistent pattern for performance of A and B group events. Both A and B group events are smaller scale events. However, some events

seem generally hard to recommend for, such as A1, A5, A7, A9, A10 and B3, as well as all events from the C group (larger scale events). It seems that this effect has more to do with the kind of events, rather than their signal patterns. Events were grouped by their signal patterns, not by the type of events (which we do not have information on).

Figure 6.11: Boxplots of nDCG@10 relevance score distribution per event for all algorithms with the best parameter combinations.

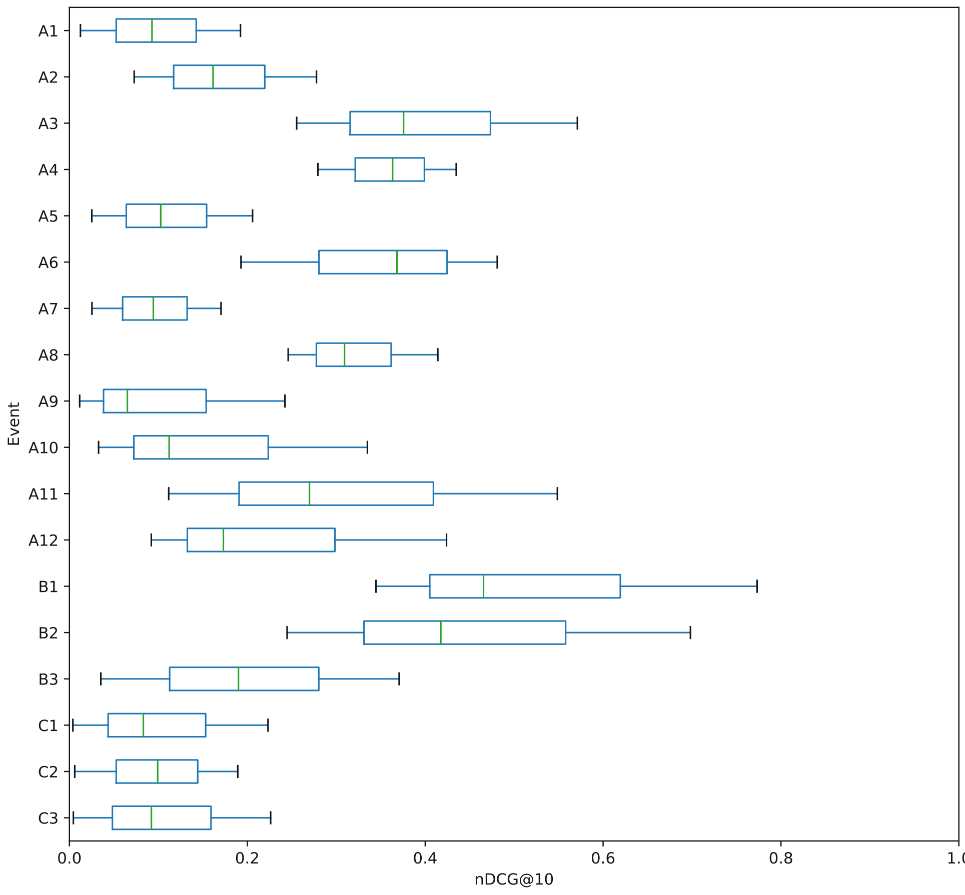


Figure 6.12: Boxplots of nDCG@10 relevance score distribution per event for the matrix factorization algorithm with all parameter combinations.

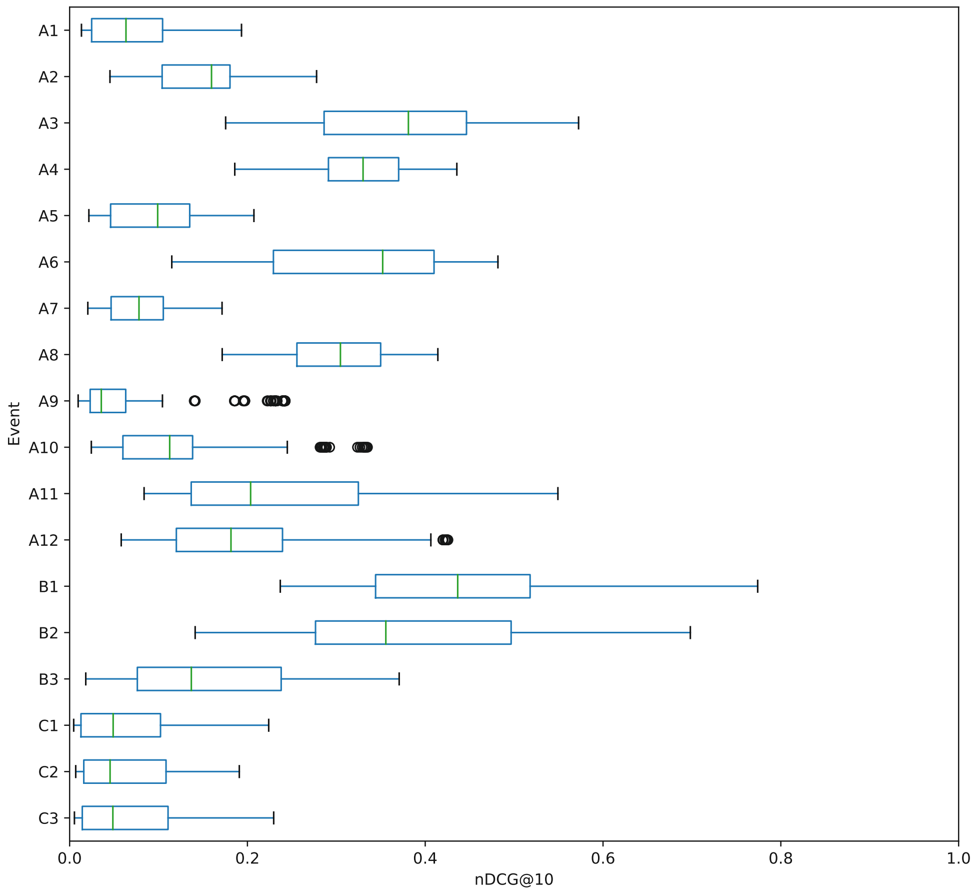


Figure 6.11 shows the nDCG@10 relevance score distribution per event for all algorithms with the optimized parameters, Figure 6.12 shows the distribution per event for the matrix factorization algorithm with all variants and parameter combinations. Both figures show a similar pattern to Figure 6.10, suggesting that the variance in performance of different events has nothing to do with algorithms or parameter optimization, and is rather a general pattern of some events being harder or easier to recommend for. It is interesting to note that the two events that performed particularly well (B1 and B2) have a lot of signals in the train split (as most signals there happen at the start of the event). Events that are particularly hard to recommend for, such as A1, A5, A7 and A9 have relatively few signals in the train split. Events from the C group always have a

spike with a lot of signals after the meeting phase starts. However, this spike is not part of the train split, which may again be why they are hard to predict for.

6.5 Reciprocal Recommender

After evaluating recommender algorithms and parameters that provide an appropriate quality of results for our data set, we now introduce a reciprocal bi-directional prediction and evaluate how it changes the overall quality of recommendations. We evaluate the reciprocal recommender on its own, before we add the hybrid recommender system.

	coclustering reciprocal=off	**knn/basic reciprocal=off**	**mf/svd reciprocal=off**
reciprocal=arithmetic	0.1021 < 0.1052 (p = 0.4049)	0.1438 < 0.1465 (p = 0.4314)	0.2070 < 0.2377 (p = 0.0343)
reciprocal=geometric	0.1034 < 0.1052 (p = 0.4426)	0.1449 < 0.1465 (p = 0.4725)	0.2027 < 0.2377 (p = 0.0195)
reciprocal=harmonic	0.1034 < 0.1052 (p = 0.4234)	0.1451 < 0.1465 (p = 0.4634)	0.1971 < 0.2377 (p = 0.0196)
reciprocal=uninorm	0.0798 < 0.1052 (p = 0.0447)	0.0746 < 0.1465 (p = 0.0023)	0.0877 < 0.2377 (p = 0.0000)

Table 6.3: nDCG@10 and p-values of reciprocal versus non-reciprocal recommenders using default parameters for the algorithms

Table 6.3 shows the nDCG@10 relevance scores and p-values of the three evaluated algorithms with reciprocal off versus different reciprocal aggregation strategies. The different aggregation strategies combine the bi-directional predictions in different ways. Interestingly, on our data set, turning reciprocal recommendation on made the performance worse for all algorithms and reciprocal aggregation strategy combinations. This difference is significant for mf/svd across all aggregations and all algorithms when using the "uninorm" aggregation. This is interesting, because others studies have shown that for people-to-people recommenders, such as online dating, reciprocal recommenders increase the performance [AKY+11]. This may suggest that in a B2B context reciprocity is not necessary to increase the number of successful meetings (our business success criteria). For example, an investor may not have a particular type of company that they invest in, so they may still accept a meeting with a company that did not initially seem interesting to them. Such a use case is special, as even in other business contexts, such as job recommender systems [WYJY15], the interest of both sides definitely matter.

Figure 6.13: nDCG@10 relevance scores of the reciprocal recommender using different strategies for aggregation, per event and overall.

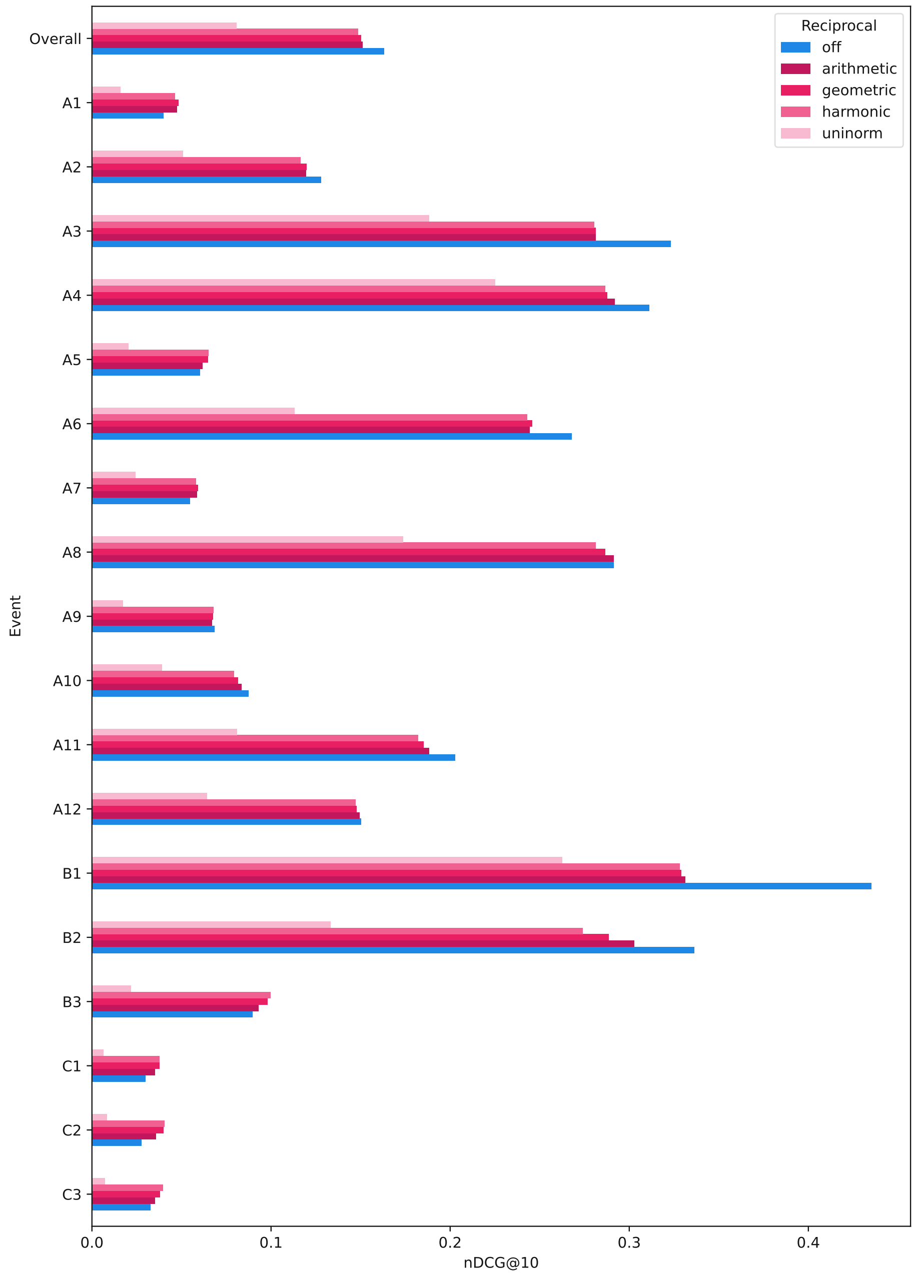


Figure 6.13 shows nDCG@10 relevance scores of the reciprocal recommender using different strategies for aggregation, per event and overall. The reciprocal recommender seems to perform worse than the standard recommender for almost all events. However, there are some exceptions: For the large-scale C group events reciprocal recommenders

work better, as well as for events A1, A5, A7 and B3, all of which have few signals as part of the train split. However, this cannot be generalized to all events with low amount of signals in the train set: A2, A3, A4, A8, A10 and A11 also have relatively few signals in the train split, with most signals part of the test split (this is a general pattern of A-group events, but particularly so for these events). Yet, for those events the recommender system performs better without reciprocal predictions. Out of all reciprocal recommenders, the arithmetic aggregation seems to generally perform best. Interestingly, for events from the group C, the harmonic reciprocal recommender performs well.

Figure 6.14: Distribution of signals per user in the reciprocal recommender.

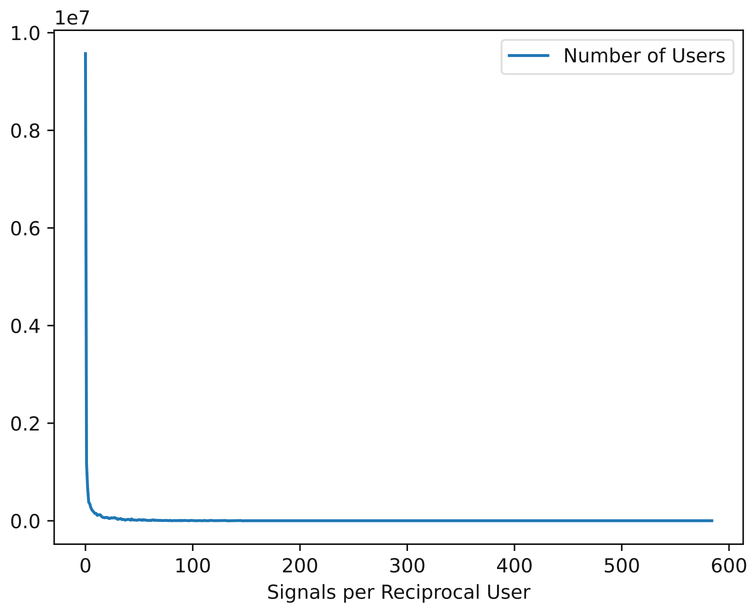


Figure 6.14 shows the distribution of signals for the corresponding users where we are doing the reciprocal recommendation. It seems like most reciprocal users (guests) have very low numbers of signals, with most having 0 signals, making a prediction impossible. This pattern might explain why the reciprocal recommender performed particularly bad on our data set.

Figure 6.15: Overview of possible reciprocal predictions per event based on mf/svd.

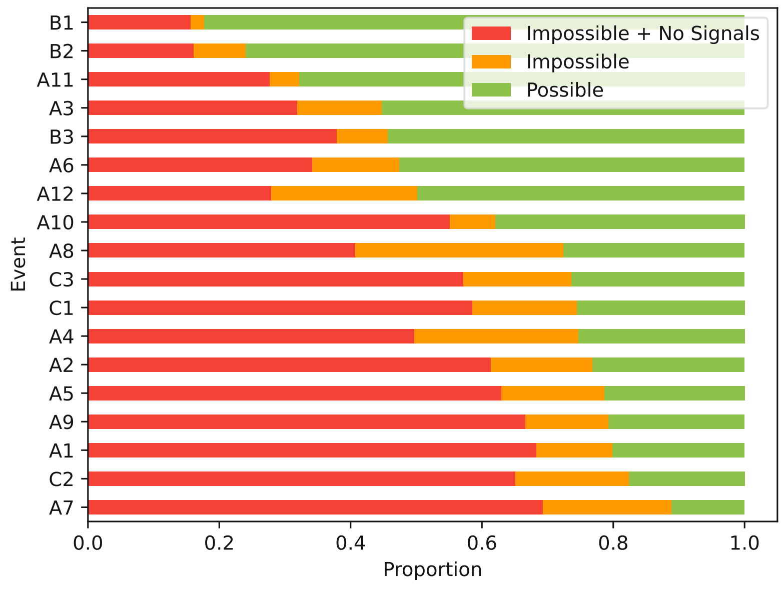
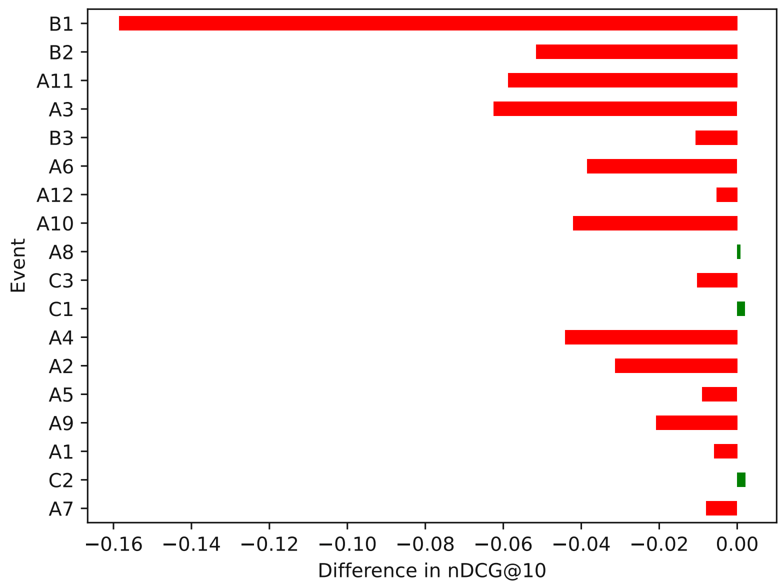


Figure 6.15 shows an overview of possible and impossible predictions, sorted in a way that shows events with the most possible reciprocal predictions first. A prediction was impossible if the recommender algorithm returned `NaN` for any reason (not enough data, not even for making a baseline estimate), a prediction was impossible + no signals when it was impossible for the specific reason of having no signals to make a recommendation. For most of the events, a reciprocal prediction was impossible in most cases, due to not having enough signals. Interestingly, events from the B group have a relatively large percentage of possible reciprocal predictions, yet the reciprocal recommender still performs worse than the normal recommender on those. Reversely, the C group events have a low percentage of possible reciprocal predictions (20%), but perform better with the reciprocal recommender.

Figure 6.16 highlights the differences in nDCG@10 of using a reciprocal recommender versus standard recommender per event. We can see that only for A8, C1 and C2 a positive difference in nDCG was achieved. For all other events, the reciprocal recommender performs worse. For event B1, the reciprocal recommender seems to perform especially worse. Events are sorted in the same way as Figure 6.15, making it possible to compare those two figures. Interestingly, B1 has the most possible reciprocal predictions, but performs much worse with the reciprocal recommender. Generally, the events that have the most possible reciprocal recommendations seem to perform worse with it, such as B1, B2, A11, A3, A6 and A10. Interestingly, those events where only few reciprocal predictions are even possible, seem to perform either on par or slightly better with the reciprocal recommender.

Figure 6.16: Differences in nDCG@10 when using the reciprocal recommender with arithmetic aggregation based on mf/svd.



6.6 Hybrid Recommender Versus Direct Recommender

After evaluating direct recommendations with our recommender system, we now evaluate our solution to the cold start problem: Adding a hybrid recommender system that, if we do not have enough signals on our target user, attempts to make recommendations for users similar to the target user, and then uses those as recommendation for our target user.

The hybrid recommender system has the following parameters:

- `min_interactions`: Minimum number of interactions/signals needed before a direct recommendation is made. If less than those interactions are available, the hybrid recommender is used.
- `neighborhood_similarity`: Decides how many similar users are taken into account to make the hybrid recommendation.
- `weighted_similarity`: Takes into account the number of interactions/signals of similar users and the target user to make a combined weighted prediction.

We now do a parameter optimization for the hybrid recommender, trying out combinations of the following parameters:

- *min_interactions*: 0, 1, 2, 3, 4, 5
- *neighborhood_similarity*: 1, 2, 3, 4, 5 (default: 3)
- *weighted_similarity*: true, false (default: true)

After parameter optimization, the following optimal values were found:

- *min_interactions*: 1
- *neighborhood_similarity*: 5
- *weighted_similarity*: false

With these optimal values, we do an evaluation of the hybrid recommender using default algorithm parameters and optimized parameters.

6.6.1 Using Default Parameters

We start by looking at the results of the hybrid recommender using default parameters for the recommender algorithms.

	Direct (default)		**Direct (default)**
Hybrid (default) Event: A1	0.0782 > 0.0659 (p = 0.0003)	**Hybrid (default)** Event: A2	0.1822 > 0.1354 (p = 0.0009)
Hybrid (default) Event: A3	0.4032 > 0.3749 (p = 0.0004)	**Hybrid (default)** Event: A4	0.4019 > 0.3526 (p = 0.0003)
Hybrid (default) Event: A5	0.0935 > 0.0777 (p = 0.0001)	**Hybrid (default)** Event: A6	0.3105 > 0.2869 (p = 0.0003)
Hybrid (default) Event: A7	0.1044 > 0.0844 (p = 0.0003)	**Hybrid (default)** Event: A8	0.2955 > 0.2675 (p = 0.0007)
Hybrid (default) Event: A9	0.1227 > 0.1036 (p = 0.0000)	**Hybrid (default)** Event: A10	0.1590 > 0.1350 (p = 0.0017)
Hybrid (default) Event: A11	0.2169 > 0.2097 (p = 0.0217)	**Hybrid (default)** Event: A12	0.1951 > 0.1613 (p = 0.0006)
Hybrid (default) Event: B1	0.4835 > 0.4790 (p = 0.0097)	**Hybrid (default)** Event: B2	0.3495 > 0.3222 (p = 0.0105)
Hybrid (default) Event: B3	0.1347 > 0.1223 (p = 0.0004)	**Hybrid (default)** Event: C1	0.0609 > 0.0546 (p = 0.0010)
Hybrid (default) Event: C2	0.0529 > 0.0448 (p = 0.0003)	**Hybrid (default)** Event: C3	0.0726 > 0.0616 (p = 0.0029)

Table 6.4: nDCG@10 and p-values of hybrid recommender versus direct recommender using default parameters per event

Figure 6.17: nDCG@10 relevance scores of the hybrid recommender versus the direct recommender per event with default parameters.

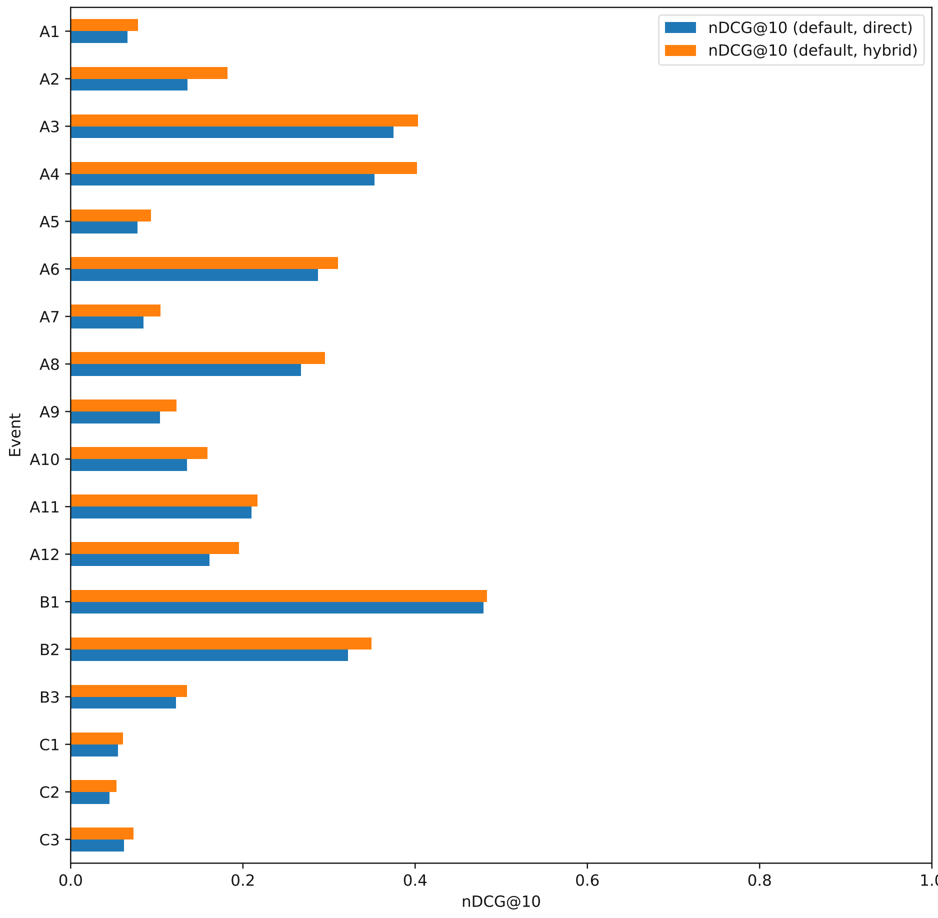


Table 6.4 lists the nDCG@10 and p-values of the hybrid recommender versus direct recommender per event, both use default parameters for the algorithms. As can be seen, the hybrid recommender results in a significant overall improvement for all events, and a significant improvement for all events from the A group except for A10 and A11. The A group events have relatively few signals in the train split, with most of the signals happening only in the test set. This hints at a potential cold start problem for some participants in the train split, which was improved by adding the hybrid recommender.

Figure 6.17 visualizes the nDCG@10 relevance scores of the hybrid recommender versus the direct recommender per event with default parameters. As can be seen, for most events from the A group a quite large improvement was achieved. This might be due to the fact that the signal pattern of those events includes only a small amount of the signals as training data, with most signals coming afterwards in the test split. For other events, such as B1, where most signals are already included in the training data, we can see that the hybrid recommender does not improve the performance much.

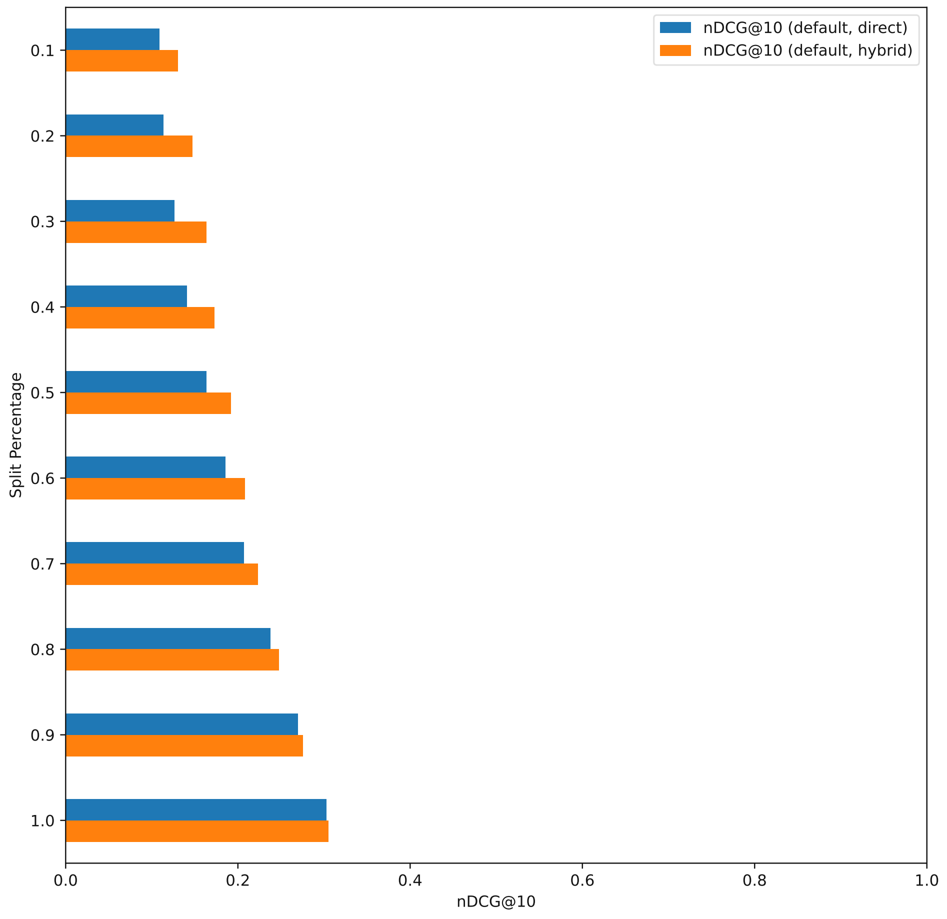
	Direct (default)		**Direct (default)**
Hybrid (default) (10% split)	0.1301 > 0.1089 (p = 0.1300)	**Hybrid (default)** (20% split)	0.1471 > 0.1137 (p = 0.0200)
Hybrid (default) (30% split)	0.1637 > 0.1260 (p = 0.0296)	**Hybrid (default)** (40% split)	0.1726 > 0.1408 (p = 0.0794)
Hybrid (default) (50% split)	0.1919 > 0.1632 (p = 0.1209)	**Hybrid (default)** (60% split)	0.2083 > 0.1853 (p = 0.1632)
Hybrid (default) (70% split)	0.2233 > 0.2069 (p = 0.2048)	**Hybrid (default)** (80% split)	0.2474 > 0.2377 (p = 0.3054)
Hybrid (default) (90% split)	0.2756 > 0.2699 (p = 0.3859)	**Hybrid (default)** (100% split)	0.3051 > 0.3027 (p = 0.4207)

Table 6.5: nDCG@10 and p-values of hybrid recommender versus direct recommender using default parameters per split

Table 6.5 shows the nDCG@10 and p-values of the hybrid recommender versus the direct recommender per split with default parameters. We analyze the different splits to find out how the hybrid recommender performs in comparison to the direct recommender at different points in time during events. Splits are made using the global temporal split strategy, so they are based on the duration of the events. The hybrid recommender seems to perform better for all splits, but only significantly for 20% and 30% splits of data. This suggests that the hybrid recommender performs especially well when only 20-30% of the signals are available yet, but generally improves performance across all splits of data.

Figure 6.18 visualizes the nDCG@10 relevance scores of the hybrid recommender versus the direct recommender per split with default parameters. As can be seen, the improvement is largest for lower splits, and decreases for the larger splits. For 100% of the data, the hybrid recommender makes almost no improvement anymore, which is fine, as it is only supposed to solve the cold start problem where we do not have enough data yet.

Figure 6.18: nDCG@10 relevance scores of the hybrid recommender versus the direct recommender per split with default parameters.



6.6.2 Using Optimized Parameters

We now look at the results of the hybrid recommender versus the direct recommender using the optimized parameters from the previous section for both direct and hybrid algorithms.

	Direct (optim.)		**Direct (optim.)**
Hybrid (optim.) Event: A1	0.1400 > 0.1261 (p = 0.0000)	**Hybrid (optim.)** Event: A2	0.2421 > 0.1956 (p = 0.0004)
Hybrid (optim.) Event: A3	0.4754 > 0.4467 (p = 0.0002)	**Hybrid (optim.)** Event: A4	0.4376 > 0.3880 (p = 0.0003)
Hybrid (optim.) Event: A5	0.1641 > 0.1468 (p = 0.0000)	**Hybrid (optim.)** Event: A6	0.3982 > 0.3683 (p = 0.0002)
Hybrid (optim.) Event: A7	0.1683 > 0.1461 (p = 0.0000)	**Hybrid (optim.)** Event: A8	0.3413 > 0.3103 (p = 0.0001)
Hybrid (optim.) Event: A9	0.1652 > 0.1445 (p = 0.0000)	**Hybrid (optim.)** Event: A10	0.2702 > 0.2424 (p = 0.0005)
Hybrid (optim.) Event: A11	0.3185 > 0.3132 (p = 0.0326)	**Hybrid (optim.)** Event: A12	0.3035 > 0.2641 (p = 0.0004)
Hybrid (optim.) Event: B1	0.6948 > 0.6922 (p = 0.0096)	**Hybrid (optim.)** Event: B2	0.5049 > 0.4794 (p = 0.0112)
Hybrid (optim.) Event: B3	0.2498 > 0.2319 (p = 0.0001)	**Hybrid (optim.)** Event: C1	0.1533 > 0.1438 (p = 0.0000)
Hybrid (optim.) Event: C2	0.1357 > 0.1238 (p = 0.0002)	**Hybrid (optim.)** Event: C3	0.1727 > 0.1572 (p = 0.0002)

Table 6.6: nDCG@10 and p-values of hybrid recommender versus direct recommender using optimized parameters per event

Table 6.6 shows the nDCG@10 and p-values of the hybrid recommender versus the direct recommender per event using the parameters from the hyper-parameter optimization. Interestingly, the hybrid recommender performs even better than the direct recommender now, significantly better across all events. For all events except A11, B1 and B2, the difference is statistically significant. B1 and B2 are events where a large portion of the signals are already part of the development data set, meaning that there is not as much of a cold start problem there than for other events.

Figure 6.19: nDCG@10 relevance scores of the hybrid recommender versus the direct recommender per event with optimized parameters.

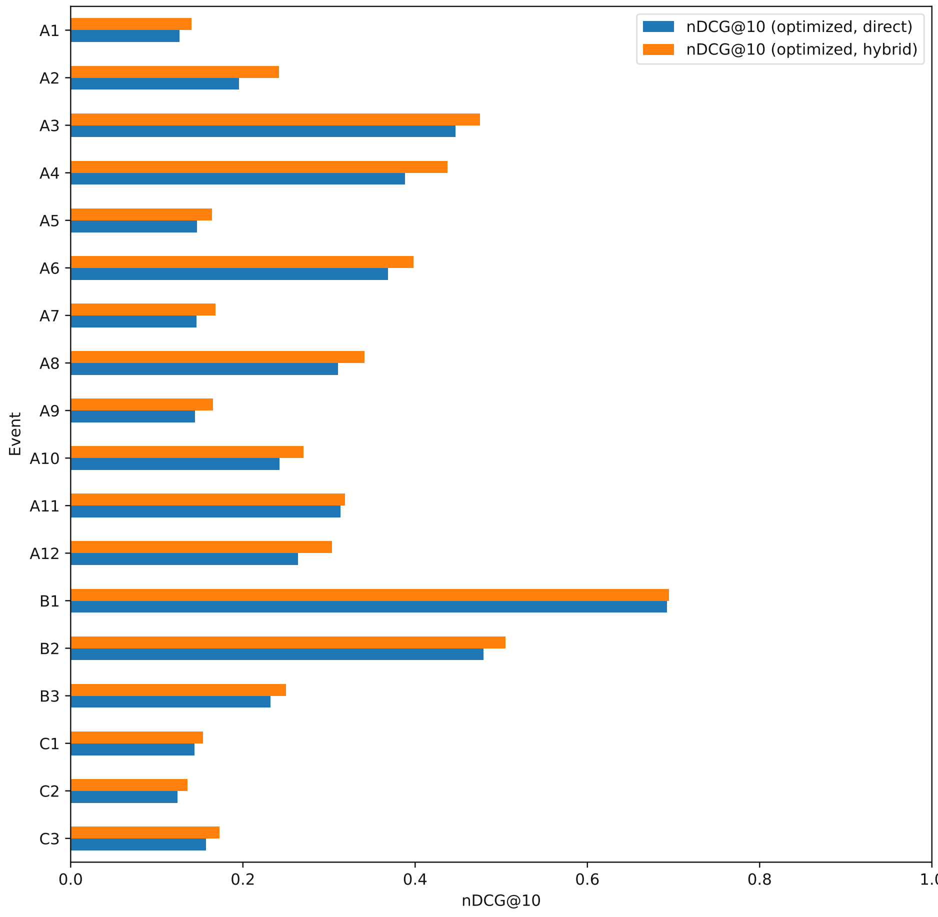


Figure 6.19 visualizes the differences between the hybrid recommender and the direct recommender per event using optimized parameters. Here we can see an increase in nDCG@10 when using the hybrid recommender for all events, but especially for most events from the A group. Events from the C group do not improve that much more with the hybrid recommender when using optimized parameters, and B1 is almost the same.

	Direct (optim.)		**Direct (optim.)**
Hybrid (optim.) (10% split)	$0.1388 > 0.1161$ ($p = 0.2667$)	**Hybrid (optim.)** (20% split)	$0.1602 > 0.1277$ ($p = 0.1018$)
Hybrid (optim.) (30% split)	$0.1857 > 0.1464$ ($p = 0.1303$)	**Hybrid (optim.)** (40% split)	$0.2038 > 0.1695$ ($p = 0.1791$)
Hybrid (optim.) (50% split)	$0.2404 > 0.2112$ ($p = 0.2109$)	**Hybrid (optim.)** (60% split)	$0.2831 > 0.2571$ ($p = 0.2810$)
Hybrid (optim.) (70% split)	$0.3232 > 0.3029$ ($p = 0.3016$)	**Hybrid (optim.)** (80% split)	$0.3906 > 0.3766$ ($p = 0.3496$)
Hybrid (optim.) (90% split)	$0.4663 > 0.4578$ ($p = 0.4073$)	**Hybrid (optim.)** (100% split)	$0.5721 > 0.5683$ ($p = 0.4316$)

Table 6.7: nDCG@10 and p-values of hybrid recommender versus direct recommender using optimized parameters per split

Table 6.7 shows the nDCG@10 and p-values of the hybrid recommender versus the direct recommender per split using optimized parameters. The pattern is very similar to what we have already seen with default parameters, where the hybrid recommender outperforms the direct recommender everywhere, but the differences are not statistically significant.

Figure 6.20: nDCG@10 relevance scores of the hybrid recommender versus the direct recommender per split with optimized parameters.

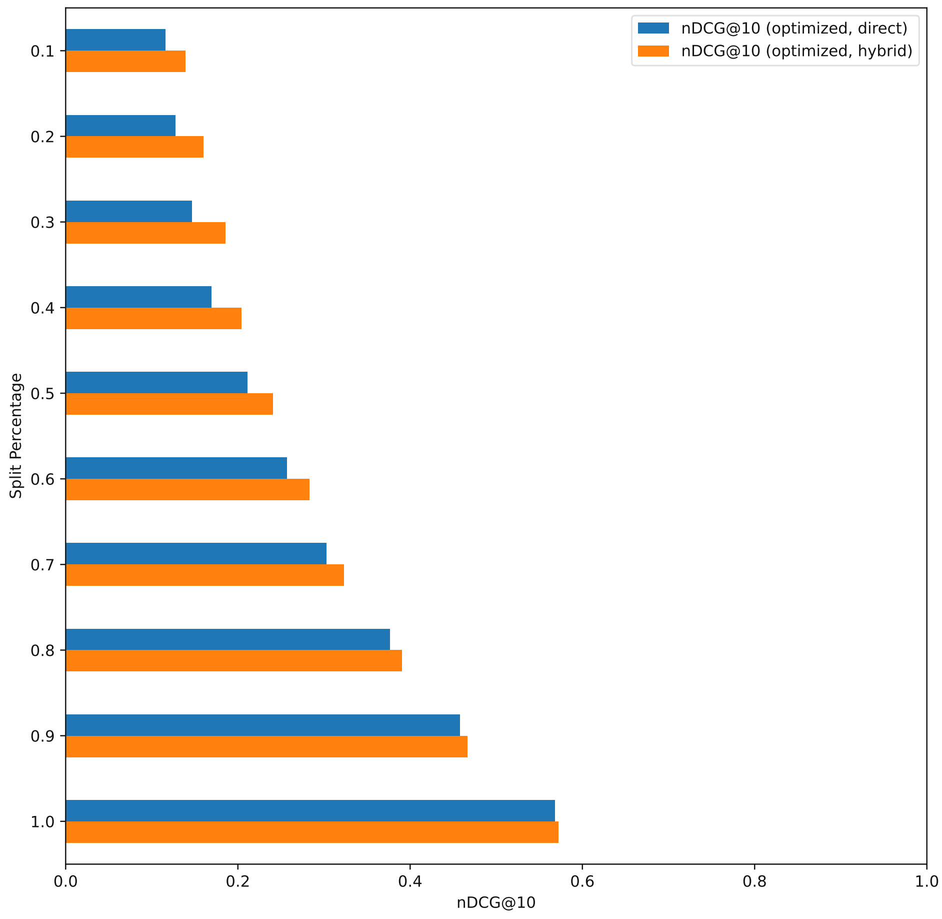


Figure 6.20 visualizes the differences between the hybrid recommender and the direct recommender per split using optimized parameters. It seems like even with optimized parameters, the hybrid recommender still performs better across all splits, and generally the performance of both is higher now. The pattern is similar to what we have seen when using default parameters.

Figure 6.21: nDCG@n relevance scores of the hybrid recommender versus the direct recommender.

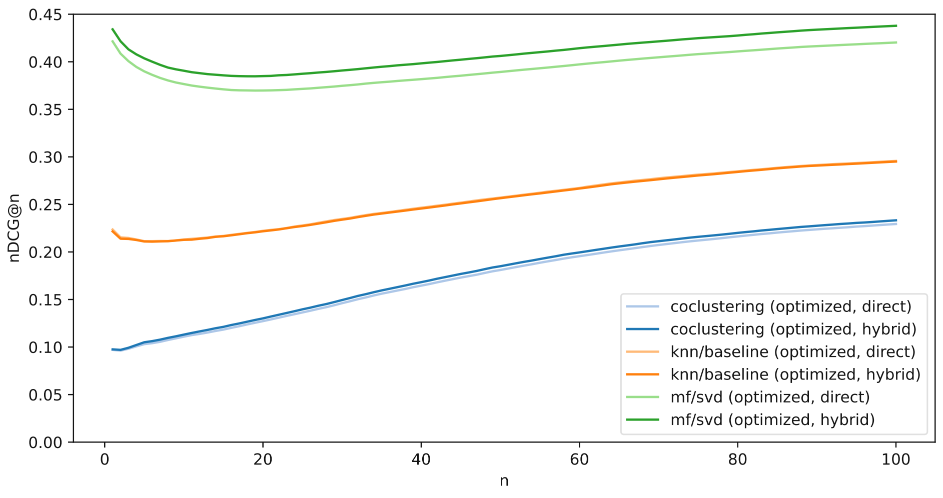


Figure 6.21 shows the nDCG@n relevance scores of the hybrid recommender versus the direct recommender. We can see that for mf/svd we managed to achieve an improvement across all numbers of results. Interestingly, for knn/baseline the hybrid recommender seems to have made no difference at all. For coclustering the improvement is only slight and only for larger numbers of results.

Figure 6.22: nDCG@n relevance scores of the hybrid recommender with the best optimized parameters versus the baselines.

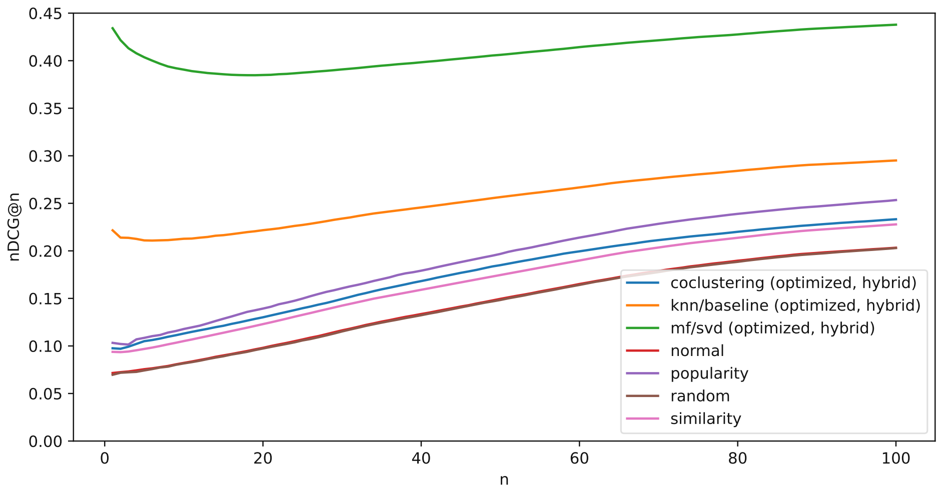


Figure 6.22 compares the nDCG@n relevance scores of the recommender algorithms with optimized parameters and the hybrid recommender enabled against baselines. As can be seen, mf and knn outperform all baselines, and coclustering outperforms all but the popularity baseline. The difference between coclustering and the similarity baseline is now larger, with the optimized and hybrid coclustering algorithm performing better than the baseline now.

6.6.3 Analysis of Hybrid Parameters

We now analyze the different parameters of the hybrid recommender separately to gain a better understanding of how they affect its performance.

Minimum Interactions The number of minimum interactions sets a cutoff for when the direct recommender is used instead of the hybrid recommender. If we have an amount of signals (like profile visits, bookmarks, messages or meeting requests) equal to or higher than the number of minimum interactions, we use the direct recommender, otherwise, the hybrid recommender is used.

Figure 6.23: nDCG@10 relevance scores of the hybrid recommender system with different numbers of minimum interactions across different global temporal splits.

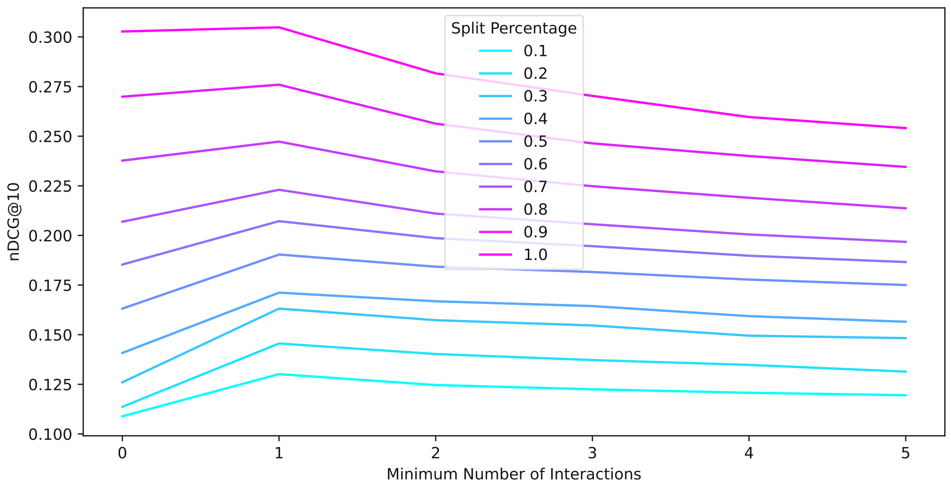


Figure 6.23 shows the nDCG@10 relevance scores of the hybrid recommender system with different numbers of minimum interactions across different global temporal splits. The lighter blue colors show splits with less data, the darker blue and violet colors show splits with more data. We can clearly see that when we have less data, the hybrid recommender especially works better, by looking at the slope of the line between 0 and 1 minimum interactions. Minimum number of interactions being 0 means direct recommendation

only, which can also be clearly seen to be performing worse for all splits here. Generally, however, once we have at least 1 interaction for a user (such as a profile visit), it seems to be better to use the direct recommendation. The performance of the hybrid recommender peaks when used only for users with less than 1 interaction (so no interactions). It seems like the hybrid recommender only improves the cold start case, where we have no signals at all yet.

Neighborhood Size The neighborhood size decides how many similar users are looked at to make a prediction with the hybrid recommender.

Figure 6.24: nDCG@10 relevance scores of the hybrid recommender system with different numbers of neighborhood size across different global temporal splits.

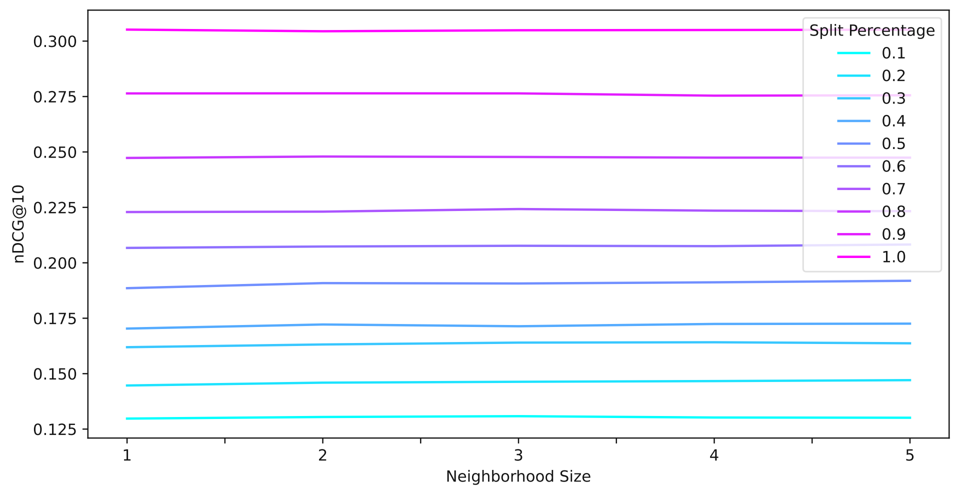


Figure 6.25: nDCG@10 relevance scores of the hybrid recommender system with different numbers of neighborhood size across different global temporal splits, with selected splits in separate plots.

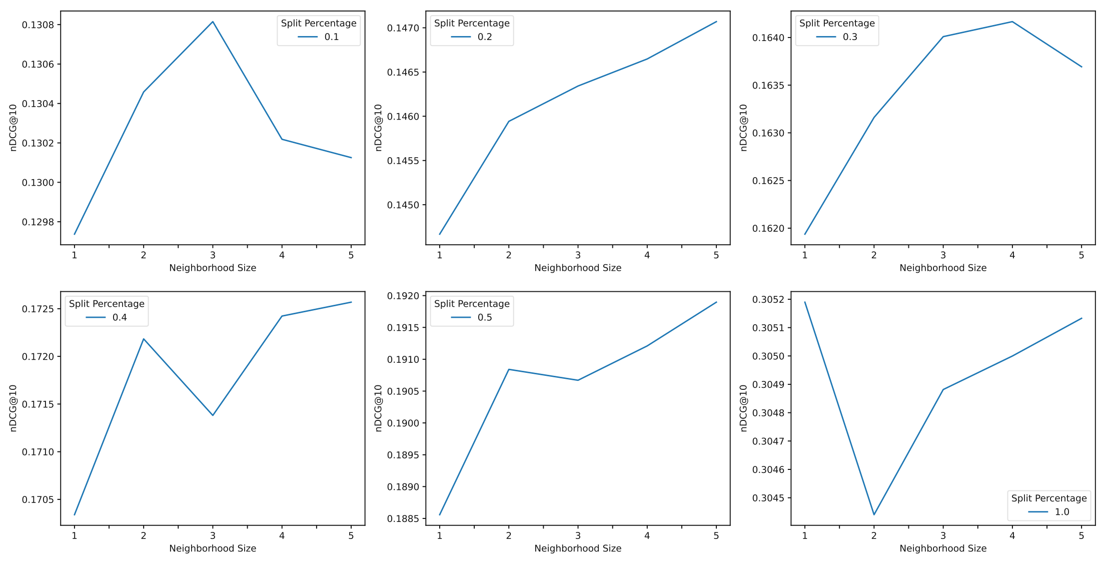


Figure 6.24 shows the nDCG@10 relevance scores of the hybrid recommender system with different numbers of neighborhood size across different global temporal splits. From this graphic it is quite hard to tell the effect the neighborhood size has on the performance.

Figure 6.25 shows the split percentages 10%, 20%, 30%, 40%, 50% and 100% separately. We can see that at only 10% of the data, a neighborhood size of 3 is optimal, using more or less decreases performance of the hybrid recommender slightly. For the 20% split, we can see that performance improves with larger neighborhood sizes, suggesting that it is better to use larger neighborhood sizes for this split. Interestingly, at 30% of the data, the optimal size is 4, with a slight decrease at 5. For the 40% split, the performance decreases after 2 neighbors, then increases again at around 4 and even more at 5. For the 50% split, the pattern is similar to the 40% split. When using all of the data, it seems like using 1 neighbor is optimal, then it sharply decreases at 2 neighbors and increases at 3, 4 and 5 neighbors again. However, this pattern should be considered with a grain of salt, as the difference looks quite large in the graph, but is actually not so large in reality. The highest point and lowest point of the graph only differ by 0.0007.

Overall, the neighborhood size makes not such a large difference, and even using only the most similar user for recommendation gives decent results. The decreases shown in the graphs here are negligible (on average a 0.001 decrease in nDCG). The hyper-parameter optimization earlier across all splits showed best results when using 1 minimum interaction and 5 neighbors, so these settings are probably best for the hybrid recommender on our given data set.

Weighting Enabling weighting in the hybrid recommender multiplies the prediction of each user (including the target user itself) with the relative number of signals we have on them. For example, if we have 1 signal on the target user, 4 signals on the most relevant user, and 5 signals on the second most relevant user, the target user prediction gets multiplied by 0.1, the most relevant user prediction by 0.4, and the second most relevant user prediction by 0.5.

Figure 6.26: nDCG@10 relevance scores of the hybrid recommender system with and without weighted results across different global temporal splits.

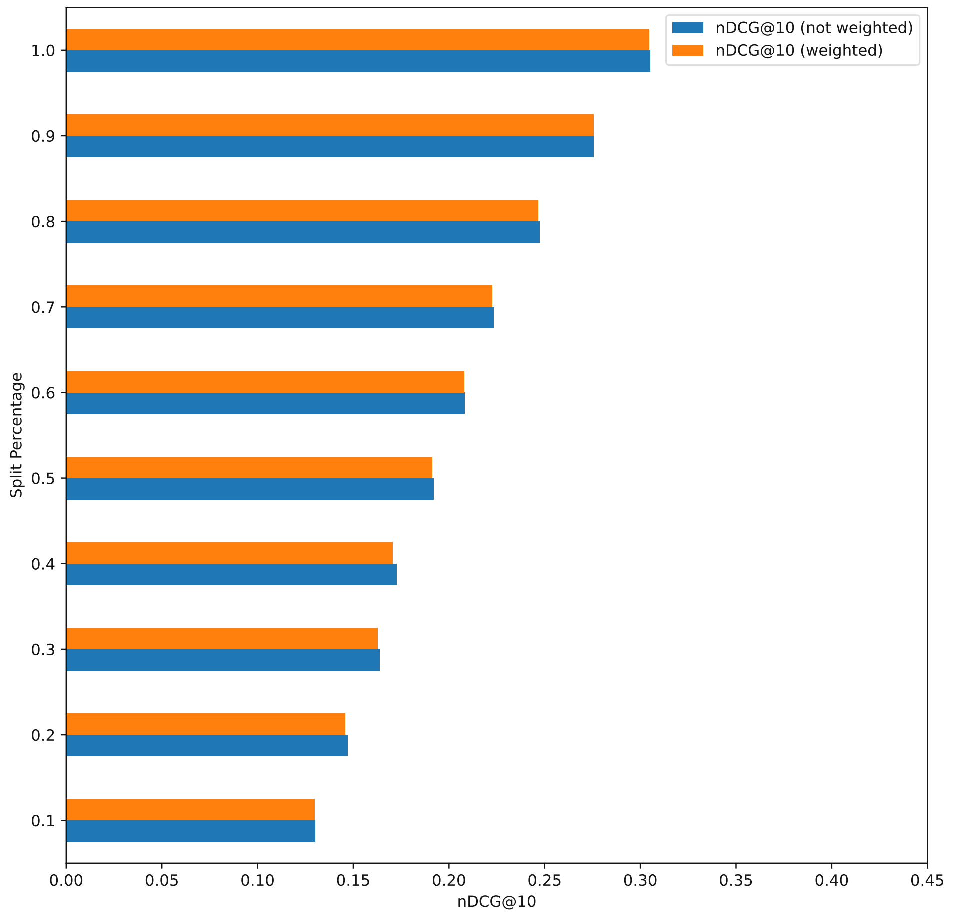


Figure 6.26 shows the nDCG@10 relevance scores of the hybrid recommender system with weighted results versus without weighted results. Generally, the effect of weighting by the number of signals is negligible, but overall it is slightly worse than not using weighted results. Additionally, weighting increases the prediction time.

6.7 Confirming Results With the Evaluation Data Set

	normal	**popularity**	**random**	**similarity**
coclustering	$0.0539 > 0.0361$ ($p = 0.0906$)	$0.0539 < 0.0716$ ($p = 0.0214$)	$0.0539 > 0.0277$ ($p = 0.0859$)	$0.0539 > 0.0452$ ($p = 0.2843$)
knn/basic	$0.0725 > 0.0361$ ($p = 0.0532$)	$0.0725 > 0.0716$ ($p = 0.4746$)	$0.0725 > 0.0277$ ($p = 0.0565$)	$0.0725 > 0.0452$ ($p = 0.1230$)
mf/svd	$0.1431 > 0.0361$ ($p = 0.0042$)	$0.1431 > 0.0716$ ($p = 0.0063$)	$0.1431 > 0.0277$ ($p = 0.0059$)	$0.1431 > 0.0452$ ($p = 0.0083$)

Table 6.8: nDCG@10 and p-values of algorithms with default parameters versus baselines on the evaluation data set

Table 6.8 shows the nDCG@10 and p-values of algorithms with default parameters versus baselines. The results are similar to the results from the development data set. mf/svd is still significantly better than all baselines. knn/basic is still better than all baselines, however, not significantly anymore. coclustering is now not significantly better than any baselines, and even significantly worse than the popularity baseline.

	normal	**popularity**	**random**	**similarity**
coclustering	$0.0599 > 0.0361$ ($p = 0.1022$)	$0.0599 < 0.0716$ ($p = 0.1405$)	$0.0599 > 0.0277$ ($p = 0.1006$)	$0.0599 > 0.0452$ ($p = 0.2498$)
knn/baseline	$0.1302 > 0.0361$ ($p = 0.0142$)	$0.1302 > 0.0716$ ($p = 0.0349$)	$0.1302 > 0.0277$ ($p = 0.0151$)	$0.1302 > 0.0452$ ($p = 0.0205$)
mf/svd	$0.1967 > 0.0361$ ($p = 0.0044$)	$0.1967 > 0.0716$ ($p = 0.0073$)	$0.1967 > 0.0277$ ($p = 0.0045$)	$0.1967 > 0.0452$ ($p = 0.0051$)

Table 6.9: nDCG@10 and p-values of algorithms with optimized parameters versus baselines on the evaluation data set

Table 6.9 shows the nDCG@10 and p-values of algorithms with optimized parameters versus baselines. The results are similar to the results from the development data set. mf/svd is still significantly better than all baselines, knn/baseline is also still significantly better than all baselines. coclustering is not significantly better than any baselines anymore, but still beats all baselines except popularity.

	coclustering reciprocal=off	**knn/basic reciprocal=off**	**mf/svd reciprocal=off**
reciprocal=arithmetic	$0.0505 < 0.0539$ ($p = 0.3242$)	$0.0815 > 0.0725$ ($p = 0.0421$)	$0.1149 < 0.1431$ ($p = 0.0044$)
reciprocal=geometric	$0.0510 < 0.0539$ ($p = 0.3355$)	$0.0836 > 0.0725$ ($p = 0.0264$)	$0.1125 < 0.1431$ ($p = 0.0017$)
reciprocal=harmonic	$0.0524 < 0.0539$ ($p = 0.4035$)	$0.0827 > 0.0725$ ($p = 0.3850$)	$0.1083 < 0.1431$ ($p = 0.0030$)
reciprocal=uninorm	$0.0388 < 0.0539$ ($p = 0.1128$)	$0.0387 < 0.0725$ ($p = 0.1069$)	$0.0482 < 0.1431$ ($p = 0.0050$)

Table 6.10: nDCG@10 and p-values of reciprocal versus non-reciprocal recommenders using default parameters for the algorithms on the evaluation data set

Table 6.10 shows the nDCG@10 and p-values of the reciprocal versus the non-reciprocal recommenders. The reciprocal recommender still performs significantly worse for mf/svd across all aggregations, and for coclustering still performs worse for all aggregations (although not significantly). Interestingly, knn/basic now performs significantly better with arithmetic and geometric reciprocal recommenders. However, the performance in terms of nDCG@10 of knn/basic is still worse than mf/svd with or without the reciprocal recommender, so it still would not make sense overall to use knn with the reciprocal recommender.

	Direct (optim.)		**Direct (optim.)**
Hybrid (optim.) Event: D1	$0.2689 > 0.2208$ ($p = 0.0000$)	**Hybrid (optim.)** Event: D2	$0.0764 > 0.0621$ ($p = 0.0000$)
Hybrid (optim.) Event: D3	$0.1455 > 0.1298$ ($p = 0.0017$)	**Hybrid (optim.)** Event: D4	$0.0617 > 0.0553$ ($p = 0.0014$)
Hybrid (optim.) Event: D5	$0.3226 > 0.2938$ ($p = 0.0027$)	**Hybrid (optim.)** Event: D6	$0.0987 > 0.0904$ ($p = 0.0002$)

Table 6.11: nDCG@10 and p-values of hybrid recommender versus direct recommender using optimized parameters per event on the evaluation data set

Table 6.11 shows the nDCG@10 and p-values of the hybrid recommender versus direct recommender using optimized parameters per event. The results are similar to the results from the development data set. The hybrid recommender significantly performs better than the direct recommender for all events on the evaluation data set.

	Direct (optim.)		**Direct (optim.)**
Hybrid (optim.) (10% split)	0.0443 > 0.0344 (p = 0.0796)	**Hybrid (optim.)** (20% split)	0.0500 > 0.0364 (p = 0.0807)
Hybrid (optim.) (30% split)	0.0652 > 0.0433 (p = 0.1480)	**Hybrid (optim.)** (40% split)	0.0840 > 0.0560 (p = 0.1187)
Hybrid (optim.) (50% split)	0.0952 > 0.0684 (p = 0.1403)	**Hybrid (optim.)** (60% split)	0.1296 > 0.1006 (p = 0.0140)
Hybrid (optim.) (70% split)	0.1652 > 0.1349 (p = 0.0081)	**Hybrid (optim.)** (80% split)	0.2227 > 0.1967 (p = 0.0051)
Hybrid (optim.) (90% split)	0.3084 > 0.2946 (p = 0.4487)	**Hybrid (optim.)** (100% split)	0.4585 > 0.4550 (p = 0.4862)

Table 6.12: nDCG@10 and p-values of hybrid recommender versus direct recommender using optimized parameters per split on the evaluation data set

Table 6.12 shows the nDCG@10 and p-values of the hybrid recommender versus direct recommender using optimized parameters per split. The results are similar to the results from the development data set, with the hybrid recommender being better than the direct recommender for all splits, but not significant for most. Interestingly, the result is now significant for the 60%, 70%, and 80% splits. Even more interesting, the 70% and 80% splits are significantly better than the direct recommender. It seems like for the evaluation data set, the hybrid recommender performs exceptionally well even for larger splits. However, for the 90% and 100% splits, the difference is negligible again.

Figure 6.27: nDCG@n relevance scores of the hybrid recommender with the best optimized parameters versus the baselines on the evaluation data set.

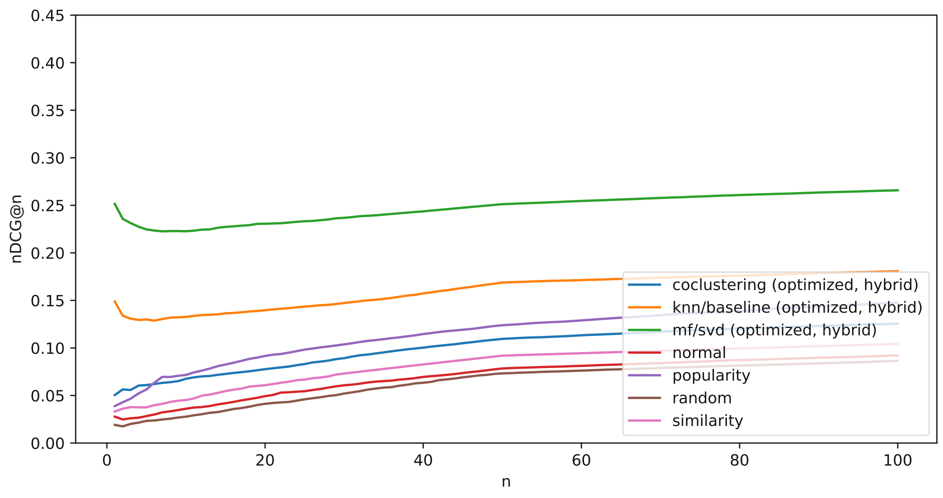


Figure 6.27 compares the nDCG@n relevance scores of the recommender algorithms with optimized parameters and the hybrid recommender enabled against baselines on the evaluation data set. The scale of the plot is the same as on the development data set. As can be seen, the performance overall on the evaluation data set is lower. Besides that, the pattern of mf/svd and knn/baseline performing best is similar to what we have seen on the development data set. Only coclustering is a little bit different now, beating the popularity baseline at the first numbers of results, but then falling behind and performing worse than the popularity data set.

CHAPTER 7

Online Evaluation Results

Being able to evaluate a recommender system in a production environment is a unique opportunity that was provided to us by the company b2match. Offline evaluations can only do so much to verify that the business success criteria has truly been achieved, as they are limited to the provided data sets and definition of ground truth. This is especially true for implicit feedback, as the scoring is defined by domain experts and not directly by users. In the online evaluation, however, we have the possibility to see if the relative number of successful meetings actually increased with the addition of a recommender system. The results of this evaluation tell us if the business success criteria has actually been achieved.

We now report the results from the recommender system being used in the real world, during real events, in an online evaluation. When a signal was made through the recommender system, we tagged it, so that we can distinguish between signals that came through the recommender system and normal signals. As the absolute numbers are skewed with the recommender system usage being generally lower than the usage of the existing participant list, we take a look at the relative proportions of signals. Overall, 767/2982 participants (hosts) were using the recommender system. This means that 25.7% of all meeting hosts used the recommender. For each host-guest participant pair, we only use the highest signal for this evaluation, just as we did before for the recommender scores. For example, if two users sent a message but then ended up booking a meeting successfully, we only count the "meeting accepted" signal. On the other hand, if the host only visited a profile but did not do anything else, we only count the "visit" signal.

	Not Recommended
Recommended Signal: Meeting Accepted	$0.0031 > 0.0018$ ($p = 0.0005$)
Recommended Signal: Meeting Requested	$0.0060 > 0.0037$ ($p = 0.0529$)
Recommended Signal: Message	$0.0029 < 0.0074$ ($p = 0.0526$)
Recommended Signal: Bookmark	$0.0005 < 0.0028$ ($p = 0.0104$)
Recommended Signal: Visit	$0.0244 > 0.0214$ ($p = 0.2976$)

Table 7.1: Signal proportions and p-values of recommender system usage versus usage of the existing participant list

Table 7.1 shows the signal proportions and p-values of recommender system usage versus usage of the existing participant list. It seems like the proportions of "Meeting Requested" and "Meeting Accepted" signals are higher for pairs that got matched through the recommender system, but also the proportion of "Visit" signals is slightly higher. Only the increase in "Meeting Accepted" signals is statistically significant. This suggests that our recommendations are either great (resulting in meeting requests and successful meetings) or bad (resulting in only visits and no further actions). However, it should be noted that the proportion of visits increased only slightly and the difference is not significant, so the recommender might not be the cause of the higher proportion of visits. Interestingly, the recommender has a lower proportion of messages and bookmarks, suggesting that those participants that were not just visited in the recommender, but also messaged and/or bookmarked, also booked a meeting in the end.

Figure 7.1: Proportions of signal types from the real world usage of the recommender system.

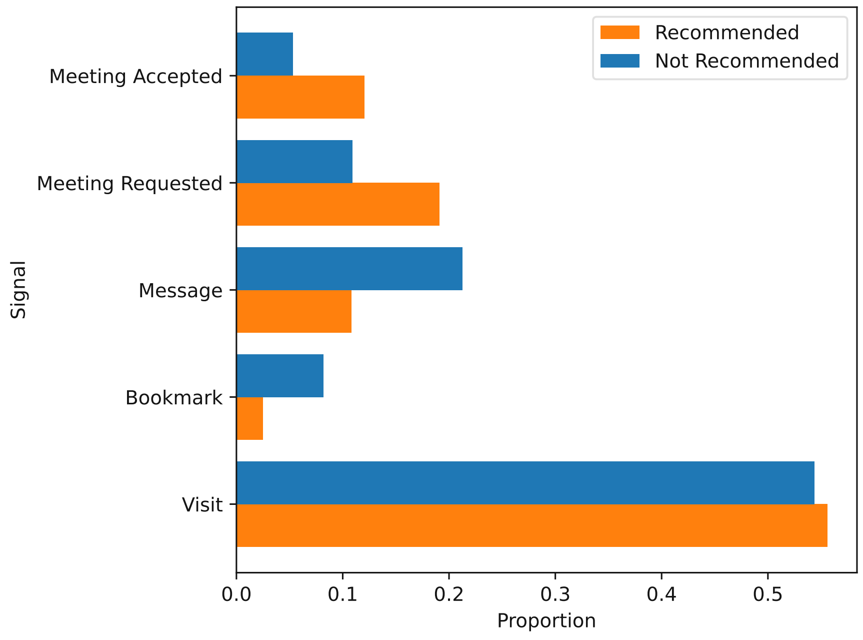


Figure 7.1 visualizes the differences in proportions of signal types. We can see that the proportions of host-guest pairs ending up with a meeting accepted and meeting requested are much higher with the recommender system. A lot less participants got stuck in the "message" and "bookmark" signals, so the overall conversion from finding a user to booking a meeting with them was significantly greater with the recommended participants.

	Not Recommended		**Not Recommended**
Recommended Event: E1	2.0000 < 2.3021 (p = 0.1613)	**Recommended** Event: E2	2.0481 < 2.5840 (p = 0.0001)
Recommended Event: E3	1.3333 < 2.5349 (p = 0.0174)	**Recommended** Event: E4	1.9091 > 1.7180 (p = 0.0130)
Recommended Event: E5	4.2381 > 3.2327 (p = 0.0067)	**Recommended** Event: E6	2.5238 > 1.8073 (p = 0.0001)
Recommended Event: E7	2.4894 > 1.5876 (p = 0.0000)	**Recommended** Event: E8	1.7103 < 2.6420 (p = 0.0000)
Recommended Event: E9	1.8559 < 1.9866 (p = 0.1639)	**Recommended** Event: E10	1.2667 < 4.0648 (p = 0.0000)
Recommended Event: E11	2.1400 > 1.8224 (p = 0.0369)	**Recommended** Event: E12	1.4242 < 1.5143 (p = 0.1811)
Recommended Event: E13	2.1758 < 2.3533 (p = 0.0162)	**Recommended** Event: E14	1.9022 < 2.5019 (p = 0.0000)
Recommended Event: E15	1.4000 < 2.1341 (p = 0.0000)	**Recommended** Event: E16	1.9818 > 1.8026 (p = 0.0524)
Recommended Event: E17	3.2812 > 1.9060 (p = 0.0000)	**Recommended** Event: E18	2.4885 > 1.7754 (p = 0.0000)
Recommended Event: E19	3.0000 > 2.2088 (p = 0.0013)	**Recommended** Event: E20	1.5103 < 2.2067 (p = 0.0000)
Recommended Event: E21	2.1000 > 1.8758 (p = 0.3217)	**Recommended** Event: E22	2.2778 > 1.7146 (p = 0.0000)
Recommended Event: E23	1.4906 < 2.2199 (p = 0.0000)	**Recommended** Event: E24	1.9730 > 1.8754 (p = 0.0358)
Recommended Event: E25	1.8400 > 1.7429 (p = 0.3490)	**Recommended** Event: E26	2.8947 > 2.6059 (p = 0.2059)
Recommended Event: E27	1.6413 > 1.4602 (p = 0.0386)		

Table 7.2: Average implicit rating and p-values of recommender system usage versus usage of the existing participant list

Table 7.2 shows the average implicit ratings and p-values per event when using the recommender system versus the existing participant list. Interestingly, for some events, the average implicit rating seems to have improved when using the recommender, for others it seems to have not worked so well. Overall, there were 11 events where the recommender worked significantly better and 9 events where it worked significantly worse. However, it should be noted that this analysis is done only using implicit feedback, which we can never be sure about.

Figure 7.2: Implicit ratings of users per event with the recommender and without the recommender.

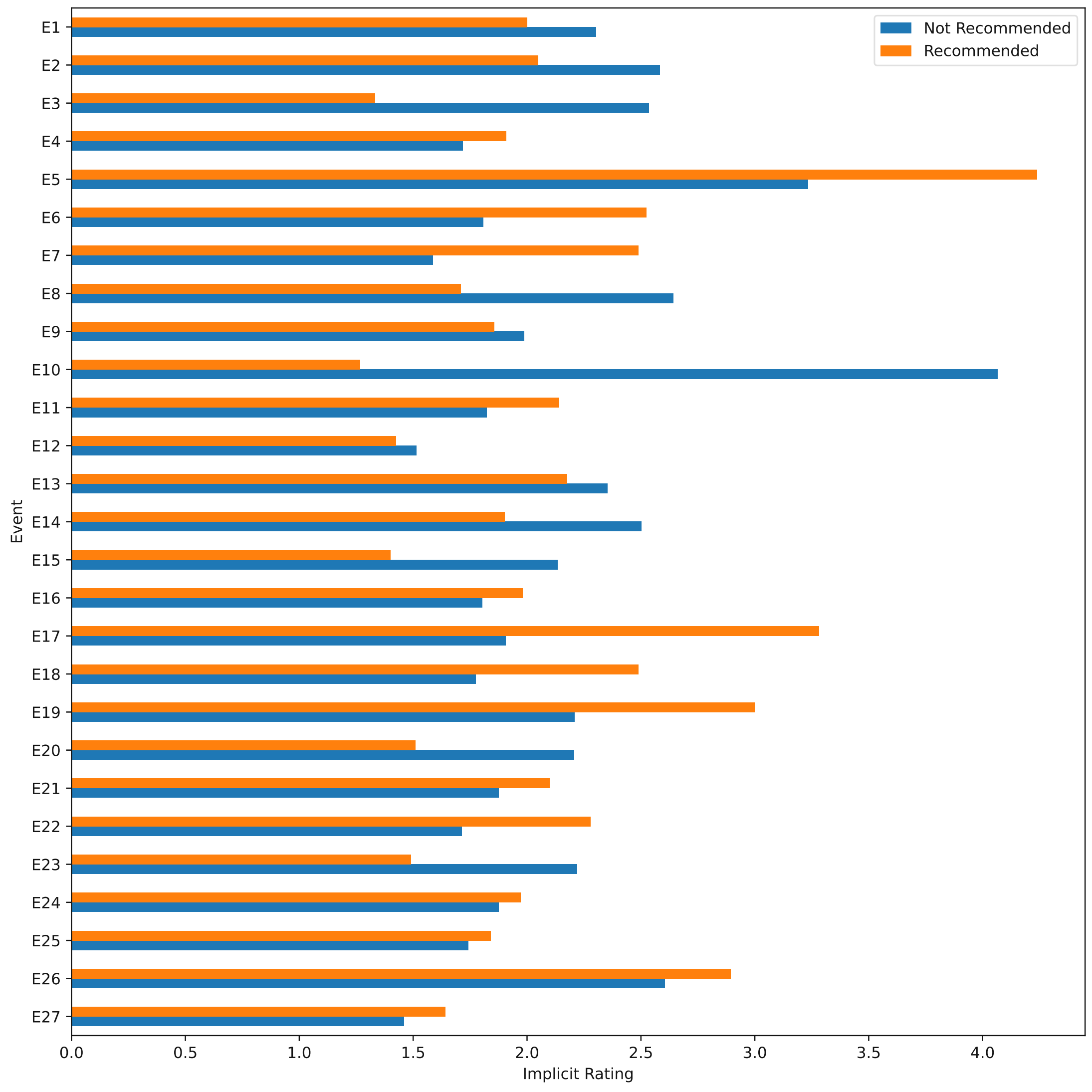


Figure 7.2 visualizes the differences in average implicit ratings. As can be seen, for some events the recommender resulted in a large increase in implicit rating, while for others it decreased the average implicit rating.

For some events during the real world usage (9 events total), we started introducing explicit ratings, where participants can rate the quality of a meeting afterwards on a scale of 1 to 5. All results from the analysis of explicit ratings should be carefully interpreted, though, because overall there were only 53 explicit ratings for results from the recommender system, and 850 for results from the existing participant list. Explicit ratings given to meetings were generally very high, with an average rating of 4.29 out of 5 stars.

	Not Recommended		**Not Recommended**
Recommended Event: E2	3.8000 < 4.2391 (p = 0.1877)	**Recommended** Event: E4	3.8824 > 3.7284 (p = 0.3080)
Recommended Event: E9	4.8333 > 4.4559 (p = 0.2174)	**Recommended** Event: E13	3.7500 < 4.3307 (p = 0.0245)
Recommended Event: E14	5.0000 > 4.6667 (p = nan)	**Recommended** Event: E15	5.0000 > 4.8947 (p = nan)
Recommended Event: E16	4.7500 < 4.8070 (p = 0.4097)	**Recommended** Event: E18	4.8333 > 4.1857 (p = 0.1093)
Recommended Event: E25	5.0000 > 4.7500 (p = nan)		

Table 7.3: Average explicit rating and p-values of recommender system usage versus usage of the existing participant list

Table 7.3 shows the average explicit ratings and p-values per event when using the recommender system versus the existing participant list. Due to the recent introduction of explicit ratings and thus very low amount of data, the results are inconclusive for most events.

Figure 7.3: Explicit ratings of users per event with the recommender and without the recommender.

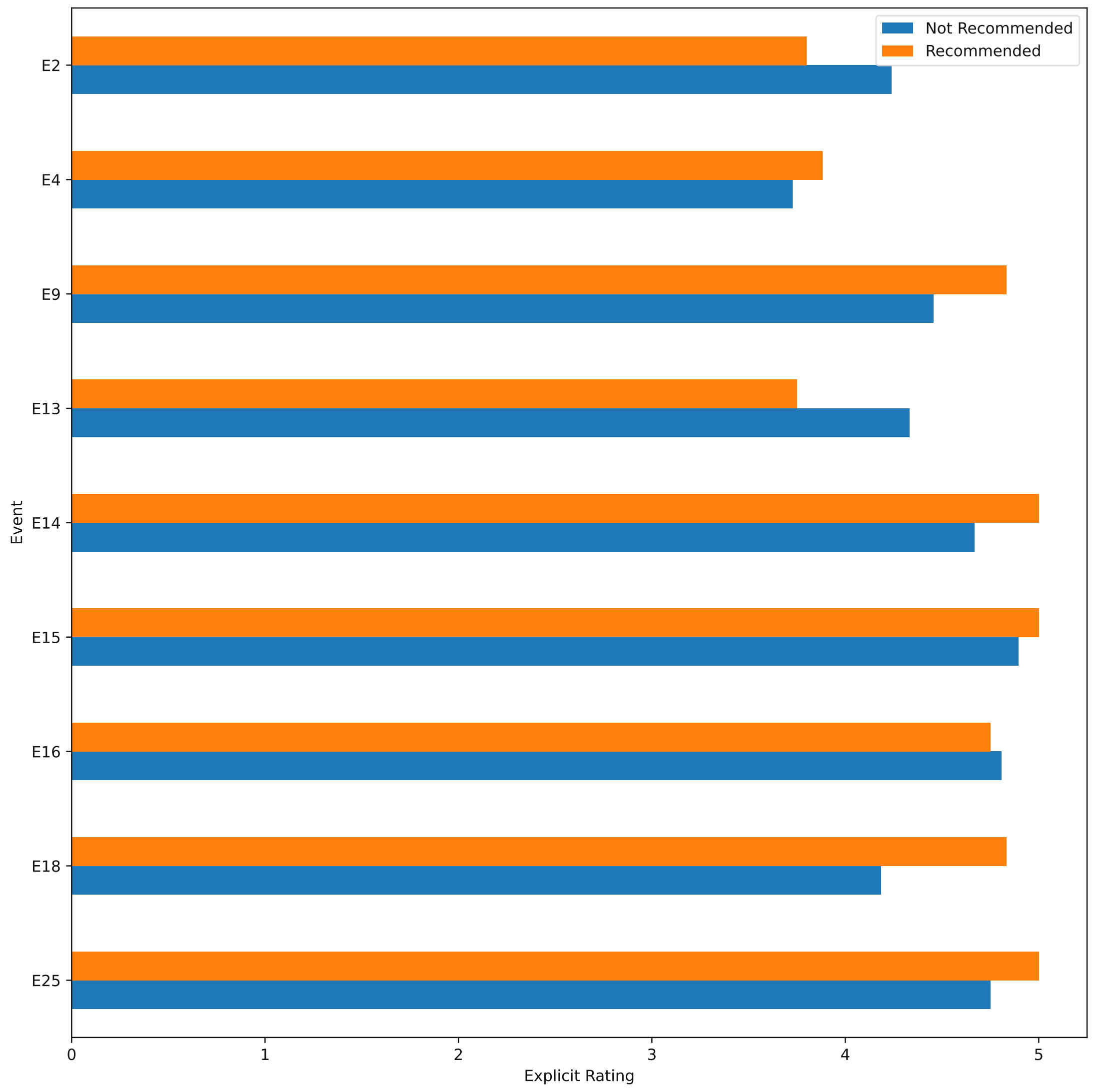


Figure 7.3 visualizes the difference in average explicit ratings. We can see that users were generally happy with the recommender, with ratings either on par with the non-recommended participants or much higher. For two events, however (E2 and E13), the rating was much lower with the recommended result.

CHAPTER 8

Conclusion

We set out to develop and evaluate a recommender system for matchmaking at B2B events. Initially, we attempted to find out which recommender algorithms provide an appropriate quality of results for a data set with heterogeneous events, compared to baseline models. Then, we explored how the use of a reciprocal bi-directional prediction changes the overall quality of recommendations. Next, we added a hybrid recommender to attempt solving the cold start problem. Finally, we evaluate how the developed recommender system performs in an online evaluation, by comparing the number of successful meetings.

Overall, matrix factorization models (with SVD) performed best on our given data set across the various kinds of events (nDCG@10 = 0.1431 for the evaluation events, outperforming all baselines at $p<0.01$). This is similar to results from [HJdT$^+$18], where it was found that matrix factorization models are scalable and provide domain-free flexibility. We optimized parameters for the data set (nDCG@10 = 0.1967 for evaluation events, outperforming all baselines at $p<0.01$) and introduced a hybrid recommender system that uses similar users to make indirect recommendations to a given participant (nDCG@10 = 0.2227 for evaluation events, outperforming the direct recommender at $p<0.01$). The hybrid recommender system improved the cold start problem by increasing the performance of the recommender system generally.

We came to a similar conclusion regarding the minimum number of signals needed to make a good prediction to [AKY$^+$11]: If we have at least one signal (like a profile visit) for the user, it is better to make a direct recommendation, but if we have no signals (cold start), it is better to make an indirect recommendation using the hybrid recommender system.

We also attempted introducing reciprocity into our recommender system, as existing research has shown that such kinds of systems work well for people-to-people recommenders [KY22]. Unfortunately, for our data set, this was not the case, and the performance of the recommender system decreased overall when using reciprocal recommenders. This might have to do with the fact that a lot of B2B meetings are one-sided, such as startups actively looking for investors, but investors not actively looking for startups to invest in. As a result, we often had the problem that if we had a lot of signals from person $A \to B$, we had 0 signals from $B \to A$, making a reciprocal recommendation hard. Even when reciprocal recommendations were possible, however, the reciprocal recommender performed worse. Maybe reciprocal recommenders matter more for use cases where both people need to be actively involved in the process, such as online dating. For B2B events, it seems like, for example, investors do not have a (known) pattern of startups that they invest in, but startups have a certain kind of investor they would be interested in. [PRC$^+$10a] explains that a reciprocal recommender system is symmetric, which might be a property that we do not want in our case. For example, it might make sense for someone to request a meeting and it might get accepted, even if the other side has not shown interest yet. This effect could be a result of having much more variety of users in our data set than, for example, an online dating system.

The platform b2match provided us with a unique opportunity to be able to evaluate the recommender system performance in the real world. Based on the analysis of implicit feedback, the recommender increases the proportion of successful meetings from 0.18% to 0.31% at $p<0.01$ and overall increases the implicit ratings for 55.56% of the events analyzed. This increase was statistically significant for 40.74% of the events analyzed (33.33% significantly decreased, 25.93% inconclusive). For the analysis of explicit feedback, we do not have enough data yet to make any conclusions.

Generally, it can be said that the developed recommender system overall significantly increases the proportion of "Meeting Accepted" signals, which we initially defined as our business success criteria. As such, the implementation of the recommender system can be considered successful by our measures. It seems like there are some events where the recommender performs especially well, while for others it does not perform well at all. In the future, those events where it does not perform well should be looked at in more detail to find out why the recommender does not work well on particular events.

8.1 Further Work

One idea to extend the hybrid recommender would be to attempt to use not only similar users from the same event, but also expand the user pool to other events that are similar to the current event. For example, some events happen yearly, so we could take a look at the series of events to make a recommendation based on similar users from previous events. Additionally, more profile information could be included in the recommendation. Unfortunately, the data set provided does not have a consistent structure as would be the case for, for example, an online dating platform. The events are just too different from each other. As such, it is hard to find meaningful data in the profile information.

Additionally, different metrics than nDCG could be used for evaluation. We found that nDCG had some problems, such as causing a very high nDCG value when predicting the same score for all results. It might make sense to include an error metric in addition to the ranking metric [AR22]. However, since such mixed metrics are still very new and not standard yet, we decided against doing so in this study. Additionally, we did not have enough explicit ratings yet to be able to compute a true error metric.

During a qualitative online evaluation by the company it was found out that the recommender tends to recommend popular users after a while. There have been some approaches like [LL12] to combat this issue by introducing vitality/passive users. The recommender system could take into account the inverse frequency of users to make less known users much more relevant. However, the effects of such a change would still have to be thoroughly evaluated.

In the future, when more explicit ratings have been collected from the events, the recommender system could be re-evaluated and further improved. Additionally, we suggested that the company could further improve the cold start problem by asking the users upon registration which kind of participants they would be interested in meeting. For example, the platform could first ask about different tags that participants put in (startup, investor, etc), and then show a list of participants, where the user selects interesting persons. This input could be used to make a good first recommendation using the recommender system, instead of having to rely on similar users. As we have found out, when we have at least one signal for a user, it is always better to make a direct recommendation rather than relying on similar users.

List of Figures

List of Tables

Bibliography

[AKY+11] Joshua Akehurst, Irena Koprinska, Kalina Yacef, Luiz Pizzato, Judy Kay, and Tomasz Rej. Ccr—a content-collaborative reciprocal recommender for online dating. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[Alv02] Sergio A Alvarez. An exact analytical relation among recall, precision, and classification accuracy in information retrieval. *Boston College, Boston, Technical Report BCCS-02-01*, pages 1–22, 2002.

[AR22] Sofia Aftab and Heri Ramampiaro. Evaluating top-n recommendations using ranked error approach: An empirical analysis. *IEEE Access*, 10:30832–30845, 2022.

[BBBK11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[BHHM12] Li Bian, Henry Holtzman, Tuan Huynh, and Marie-Jose Montpetit. Matchmaker: A friend recommendation system through tv character matching. In *2012 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 714–718. IEEE, 2012.

[BOHB12] JesúS Bobadilla, Fernando Ortega, Antonio Hernando, and Jesús Bernal. A collaborative filtering approach to mitigate the new user cold start problem. *Knowledge-based systems*, 26:225–238, 2012.

[Cao17] Longbing Cao. Data science: a comprehensive overview. *ACM Computing Surveys (CSUR)*, 50(3):1–42, 2017.

[CGD+09] Jilin Chen, Werner Geyer, Casey Dugan, Michael Muller, and Ido Guy. Make new friends, but keep the old: recommending people on social networking sites. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 201–210, 2009.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[ERK11] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. *Collaborative filtering recommender systems*. Now Publishers Inc, 2011.

[GM05] Thomas George and Srujana Merugu. A scalable collaborative filtering framework based on co-clustering. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 4–pp. IEEE, 2005.

[GM09] Asela Gunawardana and Christopher Meek. A unified approach to building hybrid recommender systems. In *Proceedings of the third ACM conference on Recommender systems*, pages 117–124, 2009.

[GNOT92] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[HJdT^{+}18] Qiwei Han, Mengxin Ji, Inigo Martinez de Rituerto de Troya, Manas Gaur, and Leid Zejnilovic. A hybrid recommender system for patient-doctor matchmaking in primary care. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 481–490. IEEE, 2018.

[Hug20] Nicolas Hug. Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174, 2020.

[IDGL^{+}08] Leo Iaquinta, Marco De Gemmis, Pasquale Lops, Giovanni Semeraro, Michele Filannino, and Piero Molino. Introducing serendipity in a content-based recommender system. In *2008 Eighth International Conference on Hybrid Intelligent Systems*, pages 168–173. IEEE, 2008.

[Jen12] Kenneth Jensen. Crisp-dm diagram, based on ibm spss modeler documentation. `https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining#/media/File:CRISP-DM_Process_Diagram.png`, 2012. [Online; accessed September 28, 2021].

[JSK10] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. Comparison of implicit and explicit feedback from an online music recommendation service. In *proceedings of the 1st international workshop on information heterogeneity and fusion in recommender systems*, pages 47–51, 2010.

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[Kor10] Yehuda Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):1–24, 2010.

[KY22] Irena Koprinska and Kalina Yacef. People-to-people reciprocal recommenders. In *Recommender Systems Handbook*, pages 421–446. Springer, 2022.

[LDGS11] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. *Recommender systems handbook*, pages 73–105, 2011.

[LKH14] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4):2065–2073, 2014.

[LL12] Lei Li and Tao Li. Meet: a generalized framework for reciprocal recommender systems. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 35–44, 2012.

[LSY03] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[LVLD08] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, pages 208–211, 2008.

[LWM$^+$15] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. Recommender system application developments: a survey. *Decision Support Systems*, 74:12–32, 2015.

[MAL$^+$03] Bradley N Miller, Istvan Albert, Shyong K Lam, Joseph A Konstan, and John Riedl. Movielens unplugged: experiences with an occasionally connected recommender system. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 263–266, 2003.

[MMMO20] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. Exploring data splitting strategies for the evaluation of recommendation models. In *Fourteenth ACM conference on recommender systems*, pages 681–686, 2020.

[NP19] James Neve and Ivan Palomares. Aggregation strategies in user-to-user reciprocal recommender systems. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 4031–4036. IEEE, 2019.

[OCKR01] Mark O'connor, Dan Cosley, Joseph A Konstan, and John Riedl. Polylens: A recommender system for groups of users. In *ECSCW 2001: Proceedings of the Seventh European conference on computer supported cooperative work 16–20 September 2001, Bonn, Germany*, pages 199–218. Springer, 2001.

[Pal20] Iván Palomares. Reciprocal recommendation: Matching users with the right users. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2429–2431, 2020.

[Pom97] Jean-Charles Pomerol. Artificial intelligence and human decision making. *European Journal of Operational Research*, 99(1):3–25, 1997.

[PRC+10a] Luiz Pizzato, Tomek Rej, Thomas Chung, Irena Koprinska, and Judy Kay. Recon: a reciprocal recommender for online dating. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 207–214, 2010.

[PRC+10b] Luiz Pizzato, Tomek Rej, Thomas Chung, Irena Koprinska, Kalina Yacef, and Judy Kay. Reciprocal recommender system for online dating. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 353–354, 2010.

[PSZ17] Sankalp Prabhakar, Gerasimos Spanakis, and Osmar Zaiane. Reciprocal recommender system for learners in massive open online courses (moocs). In *International Conference on Web-Based Learning*, pages 157–167. Springer, Cham, 2017.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RRS15] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems: Introduction and Challenges*, pages 265–308. Springer US, Boston, MA, 2015.

[RV97] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[SKKR01] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, 2001.

[STL11] Gunnar Schröder, Maik Thiele, and Wolfgang Lehner. Setting goals and choosing metrics for recommender system evaluations. In *UCERSTI2 workshop at the 5th ACM conference on recommender systems, Chicago, USA*, volume 23, page 53, 2011.

[WH00] Rüdiger Wirth and Jochen Hipp. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, volume 1. Springer-Verlag London, UK, 2000.

[WWL+13] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. A theoretical analysis of ndcg type ranking measures. In *Conference on learning theory*, pages 25–54. PMLR, 2013.

[WYJY15] Hong Wenxing, Chen Yiwei, Qi Jianwei, and Hui Yin. ihr+: A mobile reciprocal job recommender system. In *2015 10th International Conference on Computer Science & Education (ICCSE)*, pages 492–495. IEEE, 2015.

[ZBS$^+$16] Richong Zhang, Han Bao, Hailong Sun, Yanghao Wang, and Xudong Liu. Recommender systems based on ranking performance optimization. *Frontiers of Computer Science*, 10(2):270–280, 2016.