



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

**Development of a Recommender System  
that addresses the diversity principle to  
improve user satisfaction.**

Ozan Pekmezci





DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

**Development of a Recommender System  
that addresses the diversity principle to  
improve user satisfaction.**

**Entwicklung eines Empfehlungsdienst unter  
Berücksichtigung der  
Benutzerzufriedenheit zur Diversität der  
Empfehlungen.**

Author:	Ozan Pekmezci
Supervisor:	Prof. Dr-Ing. Klaus Diepold
Advisor:	Julian Wörmann, M.Sc.
Submission Date:	15.05.2019

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.05.2019

Ozan Pekmezci

## Acknowledgments

# Abstract

# Contents

<b>Acknowledgments</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background of Classical Recommender Systems . . . . .	1
1.1.1 Why Recommender Systems? . . . . .	1
1.1.2 Functions . . . . .	2
1.1.3 Applications . . . . .	3
1.1.4 Objects . . . . .	3
1.1.5 Types . . . . .	4
1.1.6 Evaluation Metrics . . . . .	4
1.1.7 User Satisfaction . . . . .	5
1.2 Interest in Recommender Systems . . . . .	5
1.3 Motivation . . . . .	6
1.4 Solution . . . . .	6
<b>2 Review of Literature and Research</b>	<b>7</b>
2.1 Types of Recommender Systems . . . . .	7
2.2 Neighborhood-Based Approach . . . . .	8
2.2.1 Introduction . . . . .	8
2.2.2 User-Based Rating Prediction . . . . .	10
2.2.3 User-Based Classification . . . . .	10
2.2.4 Regression vs Classification . . . . .	10
2.2.5 Item-Based Recommendation . . . . .	11
2.2.6 Item-Based vs User-Based Recommendation . . . . .	11
2.2.7 Steps of Neighborhood Methods . . . . .	11
2.2.8 Conclusion . . . . .	14
2.3 Collaborative Filtering . . . . .	15
2.3.1 Introduction . . . . .	15
2.3.2 Baseline . . . . .	17
2.4 Content-Based Filtering . . . . .	21
2.4.1 Introduction . . . . .	21

2.4.2	Conclusion . . . . .	27
2.5	Knowledge-Based Recommender Systems . . . . .	27
2.6	Evaluation Metrics . . . . .	27
2.6.1	Introduction . . . . .	27
2.6.2	Experimental Settings . . . . .	29
2.6.3	Recommender System Properties . . . . .	37
2.6.4	User Preference . . . . .	38
2.6.5	Notation . . . . .	39
2.6.6	Conclusion . . . . .	44
2.7	Novelty and Diversity . . . . .	45
2.7.1	Introduction . . . . .	45
2.7.2	Novelty and Diversity in Recommender Systems . . . . .	46
2.7.3	Novelty and Diversity Evaluation . . . . .	51
2.7.4	Novelty and Diversity Enhancement Approaches . . . . .	53
2.7.5	Unified View . . . . .	55
2.7.6	Empirical Metric Comparison . . . . .	56
2.7.7	Conclusion . . . . .	56
<b>3</b>	<b>Implementation</b>	<b>57</b>
3.1	Datasets . . . . .	57
3.1.1	Freelancer Dataset . . . . .	58
3.1.2	Company Dataset . . . . .	60
3.2	Unsupervised Individual Recommender . . . . .	61
3.2.1	Recommendation by Similarity . . . . .	61
3.2.2	Recommendation by Popularity . . . . .	61
3.2.3	Hybrid Recommendation . . . . .	62
3.3	Supervised Individual Recommender . . . . .	62
3.3.1	Using Sparse Input . . . . .	62
3.3.2	Using Embeddings . . . . .	66
3.4	Unsupervised Group Recommender . . . . .	69
3.4.1	Baseline . . . . .	69
3.4.2	Diverse . . . . .	70
3.5	Supervised Group Recommender . . . . .	70
3.6	Group Recommendation using clustering . . . . .	71
3.7	Dashboard to show data and enter Feedback . . . . .	75
3.7.1	General Dashboard . . . . .	75
3.7.2	Individual Recommendations . . . . .	76
3.7.3	Group Recommendations . . . . .	77
3.8	Improvement of Recommendations via Feedback Learning . . . . .	79

3.9	Summary . . . . .	80
<b>4</b>	<b>Evaluation</b>	<b>82</b>
4.1	Unsupervised Individual Recommendation . . . . .	83
4.1.1	Existing Company Recommender . . . . .	84
4.1.2	Recommendation by Similarity . . . . .	84
4.1.3	Recommendation by Popularity . . . . .	86
4.1.4	Hybrid Recommendation . . . . .	87
4.2	Supervised Individual Recommender . . . . .	88
4.2.1	Using Sparse Input . . . . .	88
4.2.2	Using Embeddings . . . . .	89
4.3	Unsupervised Group Recommender . . . . .	90
4.3.1	Baseline Recommender . . . . .	90
4.3.2	Diverse Recommender . . . . .	91
4.4	Supervised Group Recommender . . . . .	92
4.4.1	Baseline Recommender . . . . .	94
4.4.2	Diverse Recommender . . . . .	94
4.5	Feedback Loop . . . . .	95
4.5.1	Online Evaluation and User Study . . . . .	97
4.6	Summary . . . . .	97
<b>5</b>	<b>Discussion</b>	<b>98</b>
5.1	Problems about datasets . . . . .	98
5.1.1	Problems about Motius dataset . . . . .	98
5.1.2	Problems about Freelancer dataset . . . . .	98
5.2	Comparison of Individual Recommenders . . . . .	98
5.3	Comparison of Group Recommenders . . . . .	98
5.4	Conclusion . . . . .	98
<b>6</b>	<b>Conclusion</b>	<b>99</b>
6.1	Introduction . . . . .	99
6.2	Conclusion . . . . .	99
<b>7</b>	<b>Appendix</b>	<b>101</b>
	<b>List of Figures</b>	<b>103</b>
	<b>List of Tables</b>	<b>105</b>
	<b>Bibliography</b>	<b>106</b>



# 1 Introduction

Recommender Systems (RSs) are software tools and techniques that provide suggestions for items that are most likely of interest to a particular user [RRS15].

Suggestions and items depend on the field that recommender system is applied. For example, for the topic of news article recommenders, the aim will most likely be suggesting news to the readers. In the field of job recommenders though, these suggestions can be bidirectional. Meaning that, job postings can be suggested to applicants or resumes can be recommended to the human resources team of a company.

This chapter of the thesis will focus on explaining the basic terminology of recommender systems so that readers who are new to the topic can understand the rest easily.

## 1.1 Background of Classical Recommender Systems

Although the history of the recommender systems go back to mid-1990s [Par+12], the real boom happened after e-commerce services became mainstream [1]. Since there were too many items to choose from for users, such service was needed. Users of websites were becoming overloaded with the information and the developers of recommender systems had aim of reducing the information to be only relevant to users.

This section contains brief information about classical recommender systems.

### 1.1.1 Why Recommender Systems?

As mentioned in the last paragraph, recommender systems have the general function of suggesting items to users. However, why do recommender systems get developed? What kind of benefit do they have for both companies and users?

First of all, recommender systems increase the number of items sold [RRS15]. Also, most recommenders suggest personalized results, which means that users will see content that fits their desires. This will also increase users buying more items.

Recommenders also increase the coverage of items that user see. Coverage denotes number of recommended unique items divided by the number of all items. Therefore, users can interact with items that they wouldn't even see without recommenders, that improves chances of buying more items.

Another important point is definitely increasing the user satisfaction. This function of recommenders is the foundation of the thesis at hand. Unfortunately, most of the researchers don't take into account that the user satisfaction does not solely depend on simple evaluation metrics like accuracy, precision or recall but it can also depend on privacy, data security, diversity, serendipity, labeling, and presentation [Bee+16]. When recommender systems take those factors into account, they clearly increase user satisfaction. [Gerekirse iki madde daha var]

### 1.1.2 Functions

To achieve the goals in the previous subsection, recommender systems need to fulfill some functions. Common functions include but not limited to [RRS15]:

[kopi - peyst]

- Find Some Good Items: Recommend to a user some items as a ranked list along with predictions of how much the user would like them (e.g., on a scale of one-to-five stars). This is the main recommendation task that many commercial systems address (see, for instance, Chap. 11). Some systems do not show the predicted rating.
- Find all good items: Recommend all the items that can satisfy some user needs. In such cases it is insufficient to just find some good items. This is especially true when the number of items is relatively small or when the RS is mission-critical, such as in medical or financial applications. In these situations, in addition to the benefit derived from carefully examining all the possibilities, the user may also benefit from the RS ranking of these items or from additional explanations that the RS generates.
- Recommend a sequence: Instead of focusing on the generation of a single recommendation, the idea is to recommend a sequence of items that is pleasing as a whole. Typical examples include recommending a TV series, a book on RSs after having recommended a book on data mining, or a compilation of musical tracks
- Recommend a bundle: Suggest a group of items that fits well together. For instance, a travel plan may be composed of various attractions, destinations, and accommodation services that are located in a delimited area. From the point of view of the user, these various alternatives can be considered and selected as a single travel destination

[kopi - peyst]

### 1.1.3 Applications

[kopi - peyst]

- Entertainment—recommendations for movies, music, games, and IPTV.
- Content—personalized newspapers, recommendation for documents, recommendations of webpages, e-learning applications, and e-mail filters.
- E-commerce—recommendations of products to buy such as books, cameras, PCs etc. for consumers.
- Services—recommendations of travel services, recommendation of experts for consultation, recommendation of houses to rent, or matchmaking services.
- Social—recommendation of people in social networks, and recommendations of content social media content such as tweets, Facebook feeds, LinkedIn updates, and others.

### 1.1.4 Objects

Data used by typical recommender systems refer to three types of objects [RRS15]: users, items and interactions.

[kopi - peyst] Items Items are the objects that are recommended. Items may be characterized by their complexity and their value or utility. The value of an item may be positive if the item is useful to the user, or negative if the item is not appropriate and the user made the wrong decision when selecting it. We note that when a user is acquiring an item, one will always incur in a cost which includes the cognitive cost of searching for the item and the real monetary cost eventually paid for the item.

Users Users of an RS, as mentioned above, may have very diverse goals and characteristics. In order to personalize the recommendations and the human-computer interaction, RSs exploit a range of information about the users. This information can be structured in various ways, and again, the selection of what information to model depends on the recommendation technique.

Transactions We generically refer to a transaction as a recorded interaction between a user and the RS. Transactions are log-like data that store important information generated during the human-computer interaction and which are useful for the recommendation generation algorithm that the system is using. For instance, a transaction log may contain a reference to the item selected by the user and a description of the context (e.g., the user goal/query) for that particular recommendation. If available, that transaction may also include explicit feedback that the user has provided, such as the rating for the selected item. Implicit, explicit

[kopi - peyst]

### 1.1.5 Types

[gelistir, buyut]

- Collaborative-Filtering: Recommendations based on how other users rated items. Similar users are identified and the items that they rated well are recommended. Has cold-start, sparsity and scalability issues. First research paper released in the mid-1990s, still used widely [8].
- Content-based filtering: The requirements for this approach are features and the ratings of the items by users. A classifier for users' 'profile' is built and similar items are recommended based on it. It has the problem of recommending only very similar results that the user already is aware of. It was first mentioned on an academic paper on 1998 [8].
- Knowledge-based filtering: This approach also requires features of the items and explicit description of what the user needs or wants. Then, items that match those needs are recommended. The advantage of this approach is the fact that the system doesn't need data from different users. Unfortunately, the suggestion ability is rather static [4]. Research about this topic was first released on 1999.
- Hybrid Recommender Systems: Since all of the methods have some drawbacks, applications have shifted to combining more than one of the previous approaches. Hybrid recommender systems are still widely used by big companies like Amazon [1], Spotify [2] and Netflix [3].

### 1.1.6 Evaluation Metrics

Evaluation metrics are bla

#### Offline Evaluation

- Accuracy
- Precision
- Recall

## Online Evaluation

Off-line experiments can measure the quality of the chosen algorithm in fulfilling its recommendation task. However, such evaluation cannot provide any insight about the user satisfaction, acceptance or experience with the system. The algorithms might be very accurate in solving the core recommendation problem, i.e., predicting user ratings, but for some other reason the system may not be accepted by users, for example, because the performance of the system was not as expected. Therefore, a user-centric evaluation is also required. It can be performed online after the system has been launched, or as a focused user study. During on-line evaluation, real users interact with the system without being aware of the full nature of the experiment running in the background. It is possible to run various versions of the algorithms on different groups of users for comparison and analysis of the system logs in order to enhance system performance. In addition, most of the algorithms include parameters, such as weight thresholds, the number of neighbors, etc., requiring constant adjustment and calibration.

[kopi - peyst]

More information in research part

### 1.1.7 User Satisfaction

[proposaldan]

- The recommendation performance is mainly evaluated in terms of accuracy (i.e. difference between true and predicted rating).
- This might not be a desired goal of the talent/project recommendations (e.g. simply recommending a 'top-N' list of talents that best match the requirements might result in similar talents whose skills only vary in a small extent).
- Besides the desired diversity, there might be other properties necessary for adequate recommendations (e.g. privacy, data security, diversity, serendipity, labeling, and presentation, and group recommendation).

## 1.2 Interest in Recommender Systems

In recent years, the interest in recommender systems has dramatically increased, as the following facts indicate: 1. Recommender systems play an important role in highly-rated Internet sites such as Amazon.com, YouTube, Netflix, Spotify, LinkedIn, Facebook, Tripadvisor, Last.fm, and IMDb. Moreover many media companies are now developing and deploying RSs as part of the services they provide to their subscribers. For example, Netflix, the online provider of on-demand streaming media, awarded a million dollar prize to the team that first succeeded in substantially improving the

performance of its recommender system [31]. 2. There are conferences and workshops dedicated specifically to the field, namely the Association of Computing Machinery's (ACM) Conference Series on Recommender Systems (RecSys), established in 2007. This conference stands as the premier annual event in recommender technology research and applications. In addition, sessions dedicated to RSs are frequently included in more traditional conferences in the area of databases, information systems and adaptive systems. Additional noteworthy conferences within this scope include: ACM's Special Interest Group on Information Retrieval (SIGIR); User Modeling, Adaptation and Personalization (UMAP); Intelligent User Interfaces (IUI); World Wide Web (WWW); and ACM's Special Interest Group on Management Of Data (SIGMOD). 3. At institutions of higher education around the world, undergraduate and graduate courses are now dedicated entirely to RSs, tutorials on RSs are very popular at computer science conferences, and a book introducing RSs techniques has been published as well [27]. Springer is publishing several books on specific topics in recommender systems in its series: Springer Briefs in Electrical and Computer Engineering. A large, new collection of articles dedicated to recommender systems applications to software engineering has also recently been published [46]. 4. There have been several special issues in academic journals which cover research and developments in the RSs field. Among the journals that have dedicated issues to RSs are: AI Communications (2008); IEEE Intelligent Systems (2007); International Journal of Electronic Commerce (2006); International Journal of Computer Science and Applications (2006); ACM Transactions on Computer Human Interaction (2005); ACM Transactions on Information Systems (2004); User Modeling and User-Adapted Interaction (2014, 2012); ACM Transactions on Interactive Intelligent Systems (2013); and ACM Transactions on Intelligent Systems and Technology (2015). [RRS15]

### 1.3 Motivation

- Thus, the aim of this thesis is to investigate how these properties transfer to the talent/project matching (i.e. are they necessary or not) and how they can be algorithmically achieved.

[Also explain job recommender if we do that]

### 1.4 Solution

How I solve the problem

## 2 Review of Literature and Research

### 2.1 Types of Recommender Systems

[kopi peyst] Item recommendation approaches can be divided in two broad categories: personalized and non-personalized. Among the personalized approaches are content-based and collaborative filtering methods, as well as hybrid techniques combining these two types of methods. The general principle of content-based (or cognitive) methods [4, 8, 42, 54] is to identify the common characteristics of items that have received a favorable rating from a user, and then recommend to this user new items that share these characteristics. Recommender systems based purely on content generally suffer from the problems of limited content analysis and over-specialization [63]. Limited content analysis occurs when the system has a limited amount of information on its users or the content of its items. For instance, privacy issues might refrain a user from providing personal information, or the precise content of items may be difficult or costly to obtain for some types of items, such as music or images. Another problem is that the content of an item is often insufficient to determine its quality. Over-specialization, on the other hand, is a side effect of the way in which content-based systems recommend new items, where the predicted rating of a user for an item is high if this item is similar to the ones liked by this user. For example, in a movie recommendation application, the system may recommend to a user a movie of the same genre or having the same actors as movies already seen by this user. Because of this, the system may fail to recommend items that are different but still interesting to the user.

Instead of depending on content information, collaborative (or social) filtering approaches use the rating information of other users and items in the system. The key idea is that the rating of a target user for a new item is likely to be similar to that of another user, if both users have rated other items in a similar way. Likewise, the target user is likely to rate two items in a similar fashion, if other users have given similar ratings to these two items. Collaborative approaches overcome some of the limitations of content-based ones. For instance, items for which the content is not available or difficult to obtain can still be recommended to users through the feedback of other users. Furthermore, collaborative recommendations are based on the quality of items as evaluated by peers, instead of relying on content that may be a bad indicator of quality. Finally, unlike content-based systems, collaborative filtering ones can recommend items

with very different content, as long as other users have already shown interest for these different items.

Collaborative filtering approaches can be grouped in the two general classes of neighborhood and model-based methods. In neighborhood-based (memory-based [10] or heuristic-based [2]) collaborative filtering [14, 15, 27, 39, 44, 48, 57, 59, 63], the user-item ratings stored in the system are directly used to predict ratings for new items. This can be done in two ways known as user-based or item-based recommendation. User-based systems, such as GroupLens [39], Bellcore video [27], and Ringo [63], evaluate the interest of a target user for an item using the ratings for this item by other users, called neighbors, that have similar rating patterns. The neighbors of the target user are typically the users whose ratings are most correlated to the target user's ratings. Item-based approaches [15, 44, 59], on the other hand, predict the rating of a user for an item based on the ratings of the user for similar items. In such approaches, two items are similar if several users of the system have rated these items in a similar fashion.

In contrast to neighborhood-based systems, which use the stored ratings directly in the prediction, model-based approaches use these ratings to learn a predictive model. Salient characteristics of users and items are captured by a set of model parameters, which are learned from training data and later used to predict new ratings. Model-based approaches for the task of recommending items are numerous and include Bayesian Clustering [10], Latent Semantic Analysis [28], Latent Dirichlet Allocation [9], Maximum Entropy [72], Boltzmann Machines [58], Support Vector Machines [23], and Singular Value Decomposition [6, 40, 53, 68, 69]. A survey of state-of-the-art model-based methods can be found in Chap. 3 of this book.

Finally, to overcome certain limitations of content-based and collaborative filtering methods, hybrid recommendation approaches combine characteristics of both types of methods. Content-based and collaborative filtering methods can be combined in various ways, for instance, by merging their individual predictions into a single, more robust prediction [8, 55], or by adding content information into a collaborative filtering model [1, 3, 51, 65, 71]. Several studies have shown hybrid recommendation approaches to provide more accurate recommendations than pure content-based or collaborative methods, especially when few ratings are available [2].

## **2.2 Neighborhood-Based Approach**

### **2.2.1 Introduction**

[kopi peyst] While recent investigations show state-of-the-art model-based approaches superior to neighborhood ones in the task of predicting ratings [40, 67], there is also an emerging understanding that good prediction accuracy alone does not guarantee users



an effective and satisfying experience.

Model-based approaches excel at characterizing the preferences of a user with latent factors. For example, in a movie recommender system, such methods may determine that a given user is a fan of movies that are both funny and romantic, without having to actually define the notions “funny” and “romantic”. This system would be able to recommend to the user a romantic comedy that may not have been known to this user. However, it may be difficult for this system to recommend a movie that does not quite fit this high-level genre, for instance, a funny parody of horror movies. Neighborhood approaches, on the other hand, capture local associations in the data. Consequently, it is possible for a movie recommender system based on this type of approach to recommend the user a movie very different from his usual taste or a movie that is not well known (e.g. repertoire film), if one of his closest neighbors has given it a strong rating. This recommendation may not be a guaranteed success, as would be a romantic comedy, but it may help the user discover a whole new genre or a new favorite actor/director. -> Model based suck at serendipity

- **Simplicity:** Neighborhood-based methods are intuitive and relatively simple to implement. In their simplest form, only one parameter (the number of neighbors used in the prediction) requires tuning.
- **Justifiability:** Such methods also provide a concise and intuitive justification for the computed predictions. For example, in item-based recommendation, the list of neighbor items, as well as the ratings given by the user to these items, can be presented to the user as a justification for the recommendation. This can help the user better understand the recommendation and its relevance, and could serve as basis for an interactive system where users can select the neighbors for which a greater importance should be given in the recommendation [6].
- **Efficiency:** One of the strong points of neighborhood-based systems are their efficiency. Unlike most model-based systems, they require no costly training phases, which need to be carried at frequent intervals in large commercial applications. These systems may require pre-computing nearest neighbors in an offline step, which is typically much cheaper than model training, providing near instantaneous recommendations. Moreover, storing these nearest neighbors requires very little memory, making such approaches scalable to applications having millions of users and items.
- **Stability:** Another useful property of recommender systems based on this approach is that they are little affected by the constant addition of users, items and ratings, which are typically observed in large commercial applications. For

instance, once item similarities have been computed, an item-based system can readily make recommendations to new users, without having to re-train the system. Moreover, once a few ratings have been entered for a new item, only the similarities between this item and the ones already in the system need to be computed.

While neighborhood-based methods have gained popularity due to these advantages, they are also known to suffer from the problem of limited coverage, which causes some items to be never recommended. Also, traditional methods of this category are known to be more sensitive to the sparseness of ratings and the cold-start problem, where the system has only a few ratings, or no rating at all, for new users and items. Section 2.5 presents more advanced neighborhood-based techniques that can overcome these problems.

### **2.2.2 User-Based Rating Prediction**

[ Kitapta 2.3.1] 2 Sayfa kadar

### **2.2.3 User-Based Classification**

[ Kitapta 2.3.2] 0.5 Sayfa kadar

### **2.2.4 Regression vs Classification**

[kopi -peys, genel daha uzun bahsedilebilir ve belki resim eklenebilir] The choice between implementing a neighborhood-based regression or classification method largely depends on the system's rating scale. Thus, if the rating scale is continuous, e.g. ratings in the Jester joke recommender system [20] can take any value between -10 and 10, then a regression method is more appropriate. On the contrary, if the rating scale has only a few discrete values, e.g. "good" or "bad", or if the values cannot be ordered in an obvious fashion, then a classification method might be preferable. Furthermore, since normalization tends to map ratings to a continuous scale, it may be harder to handle in a classification approach. Another way to compare these two approaches is by considering the situation where all neighbors have the same similarity weight. As the number of neighbors used in the prediction increases, the rating  $r_{ui}$  predicted by the regression approach will tend toward the mean rating of item  $i$ . Suppose item  $i$  has only ratings at either end of the rating range, i.e. it is either loved or hated, then the regression approach will make the safe decision that the item's worth is average. This is also justified from a statistical point of view since the expected rating (estimated in this case) is the one that minimizes the RMSE. On the other hand, the classification

approach will predict the rating as the most frequent one given to  $i$ . This is more risky as the item will be labeled as either “good” or “bad”. However, as mentioned before, taking risks may be desirable if it leads to serendipitous recommendations.

### 2.2.5 Item-Based Recommendation

[ Kitapta 2.3.2] 1 Sayfa kadar

### 2.2.6 Item-Based vs User-Based Recommendation

[ Kitapta 2.3.3] 1 Sayfa kadar

### 2.2.7 Steps of Neighborhood Methods

three very important considerations in the implementation of a neighborhood-based recommender system are (1) the normalization of ratings, (2) the computation of the similarity weights, and (3) the selection of neighbors. This section reviews some of the most common approaches for these three components, describes the main advantages and disadvantages of using each one of them, and gives indications on how to implement them.

#### Normalization of Ratings

[2.4.1.1 ve 2.4.1.2’ i de ekle] In some cases, rating normalization can have undesirable effects. For instance, imagine the case of a user that gave only the highest ratings to the items he has purchased. Mean-centering would consider this user as “easy to please” and any rating below this highest rating (whether it is a positive or negative rating) would be considered as negative. However, it is possible that this user is in fact “hard to please” and carefully selects only items that he will like for sure. Furthermore, normalizing on a few ratings can produce unexpected results. For example, if a user has entered a single rating or a few identical ratings, his rating standard deviation will be 0, leading to undefined prediction values. Nevertheless, if the rating data is not overly sparse, normalizing ratings has been found to consistently improve the predictions [25, 29].

Comparing mean-centering with Z-score, as mentioned, the second one has the additional benefit of considering the variance in the ratings of individual users or items. This is particularly useful if the rating scale has a wide range of discrete values or if it is continuous. On the other hand, because the ratings are divided and multiplied by possibly very different standard deviation values, Z-score can be more sensitive than mean-centering and, more often, predict ratings that are outside the rating scale. Lastly,

while an initial investigation found mean-centering and Z-score to give comparable results [25], a more recent one showed Z-score to have more significant benefits [29].

Finally, if rating normalization is not possible or does not improve the results, another possible approach to remove the problems caused by the rating scale variance is preference-based filtering. The particularity of this approach is that it focuses on predicting the relative preferences of users instead of absolute rating values. Since an item preferred to another one remains so regardless of the rating scale, predicting relative preferences removes the need to normalize the ratings. More information on this approach can be found in [12, 18, 32, 33].

### **Computation of Similarity Weights**

The similarity weights play a double role in neighborhood-based recommendation methods: (1) they allow to select trusted neighbors whose ratings are used in the prediction, and (2) they provide the means to give more or less importance to these neighbors in the prediction. The computation of the similarity weights is one of the most critical aspects of building a neighborhood-based recommender system, as it can have a significant impact on both its accuracy and its performance.

[2.4.2.1, 2.4.2.2] cosine vector, person correlation, adjusted cosine, mean squared prediction

There are also some considerations: [en iyisi bu kısmi ayri chapter yapip bunu subsubsection yap]

Significance of weights: 1 paragraph Variance of Ratings: 1 paragraph, mesela godfather'i herkes seviyor => IDF yap

### **Neighborhood Selection**

The number of nearest-neighbors to select and the criteria used for this selection can also have a serious impact on the quality of the recommender system. The selection of the neighbors used in the recommendation of items is normally done in two steps: (1) a global filtering step where only the most likely candidates are kept, and (2) a per prediction step which chooses the best candidates for this prediction.

Prefiltering[aslinda subsubsec]: In large recommender systems that can have millions of users and items, it is usually not possible to store the (non-zero) similarities between each pair of users or items, due to memory limitations. Moreover, doing so would be extremely wasteful as only the most significant of these values are used in the predictions. The pre-filtering of neighbors is an essential step that makes neighborhood-based approaches practicable by reducing the amount of similarity weights to store,

and limiting the number of candidate neighbors to consider in the predictions. There are several ways in which this can be accomplished:

- Top-n filtering: For each user or item, only a list of the  $N$  nearest-neighbors and their respective similarity weight is kept. To avoid problems with efficiency or accuracy,  $N$  should be chosen carefully. Thus, if  $N$  is too large, an excessive amount of memory will be required to store the neighborhood lists and predicting ratings will be slow. On the other hand, selecting a too small value for  $N$  may reduce the coverage of the recommendation method, which causes some items to be never recommended.
- Threshold filtering: Instead of keeping a fixed number of nearest-neighbors, this approach keeps all the neighbors whose similarity weight's magnitude is greater than a given threshold  $w_{min}$ . While this is more flexible than the previous filtering technique, as only the most significant neighbors are kept, the right value of  $w_{min}$  may be difficult to determine.
- Negative filtering: In general, negative rating correlations are less reliable than positive ones. Intuitively, this is because strong positive correlation between two users is a good indicator of their belonging to a common group (e.g., teenagers, science-fiction fans, etc.). However, although negative correlation may indicate membership to different groups, it does not tell how different are these groups, or whether these groups are compatible for some other categories of items.

Actual Prediction[aslinda subsubsec]: Once a list of candidate neighbors has been computed for each user or item, the prediction of new ratings is normally made with the  $k$ -nearest-neighbors, that is, the  $k$  neighbors whose similarity weight has the greatest magnitude. The choice of  $k$  can also have a significant impact on the accuracy and performance of the system.

As shown in Table 2.3, the prediction accuracy observed for increasing values of  $k$  typically follows a concave function. Thus, when the number of neighbors is restricted by using a small  $k$  (e.g.,  $k < 20$ ), the prediction accuracy is normally low. As  $k$  increases, more neighbors contribute to the prediction and the variance introduced by individual neighbors is averaged out. As a result, the prediction accuracy improves. Finally, the accuracy usually drops when too many neighbors are used in the prediction (e.g.,  $k > 50$ ), due to the fact that the few strong local relations are "diluted" by the many weak ones. Although a number of neighbors between 20 to 50 is most often described in the literature, see e.g. [24, 26], the optimal value of  $k$  should be determined by cross-validation.

On a final note, more serendipitous recommendations may be obtained at the cost of a decrease in accuracy, by basing these recommendations on a few very similar

users. For example, the system could find the user most similar to the active one and recommend the new item that has received the highest rated from this user.

### **2.2.8 Conclusion**

One of the earliest approaches proposed for the task of item recommendation is neighborhood-based recommendation, which ranks among the most popular methods for this problem. Although quite simple to describe and implement, this recommendation approach has several important advantages, including its ability to explain a recommendation with the list of the neighbors used, its computational and space efficiency which allows it to scale to large recommender systems, and its marked stability in an online setting where new users and items are constantly added. Another of its strengths is its potential to make serendipitous recommendations that can lead users to the discovery of unexpected, yet very interesting items.

In the implementation of a neighborhood-based approach, one has to make several important decisions. Perhaps the one having the greatest impact on the accuracy and efficiency of the recommender system is choosing between a user-based and an item-based neighborhood method. In typical commercial recommender systems, where the number of users far exceeds the number of available items, item-based approaches are typically preferred since they provide more accurate recommendations, while being more computationally efficient and requiring less frequent updates. On the other hand, user-based methods usually provide more original recommendations, which may lead users to a more satisfying experience. Moreover, the different components of a neighborhood-based method, which include the normalization of ratings, the computation of the similarity weights and the selection of the nearest-neighbors, can also have a significant influence on the quality of the recommender system. For each of these components, several different alternatives are available. Although the merit of each of these has been described in this document and in the literature, it is important to remember that the “best” approach may differ from one recommendation setting to the next. Thus, it is important to evaluate them on data collected from the actual system, and in light of the particular needs of the application.

Finally, when the performance of a neighborhood-based approach suffers from the problems of limited coverage and sparsity, one may explore techniques based on dimensionality reduction or graphs. Dimensionality reduction provides a compact representation of users and items that captures their most significant features. An advantage of such approach is that it allows to obtain meaningful relations between pairs of users or items, even though these users have rated different items, or these items were rated by different users. On the other hand, graph-based techniques exploit the transitive relations in the data. These techniques also avoid the problems of sparsity

and limited coverage by evaluating the relationship between users or items that are not “directly connected”. However, unlike dimensionality reduction, graph-based methods also preserve some of the “local” relations in the data, which are useful in making serendipitous recommendations.

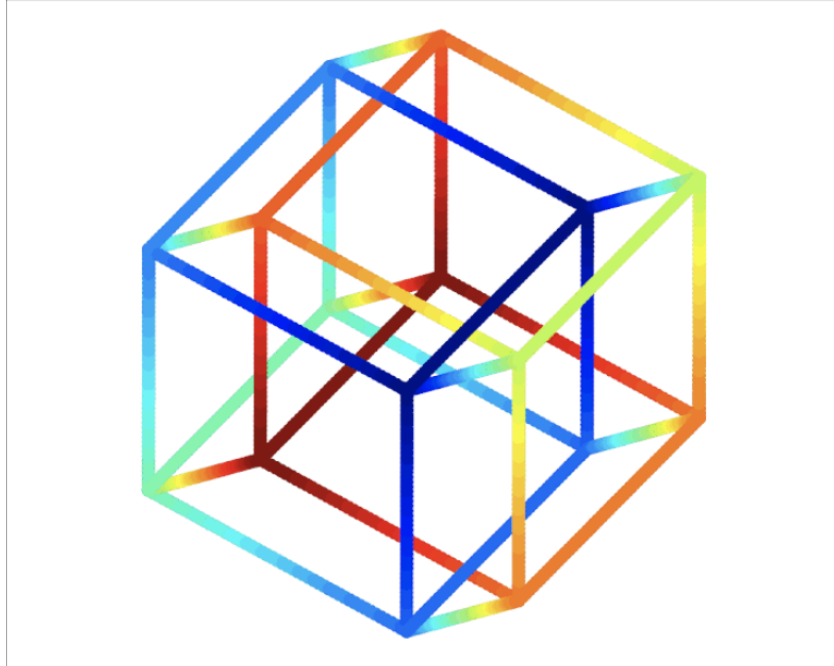


Figure 2.1: 2D representation of a 4D cube. The colors indicate the depth in fourth dimension [LV07].

[TODO: burke2002hybrid -> table 3]

## 2.3 Collaborative Filtering

### 2.3.1 Introduction

[kopi peyst] Collaborative filtering recommender system (CF) methods produce user specific recommendations of items based on patterns of ratings or usage (e.g., purchases) without need for exogenous information about either items or users. While well established methods work adequately for many purposes, we present several recent extensions available to analysts who are looking for the best possible recommendations.

The Netflix Prize competition that began in October 2006 has fueled much recent progress in the field of collaborative filtering. For the first time, the researchcommunity

gained access to a large-scale, industrial strength data set of 100 million movie ratings—attracting thousands of scientists, students, engineers and enthusiasts to the field. The nature of the competition has encouraged rapid development, where innovators built on each generation of techniques to improve prediction accuracy. Because all methods are judged by the same rigid yardstick on common data, the evolution of more powerful models has been especially efficient.

Recommender systems rely on various types of input. Most convenient is high quality explicit feedback, where users directly report on their interest in products. For example, Netflix collects star ratings for movies and TiVo users indicate their preferences for TV shows by hitting thumbs-up/down buttons.

Because explicit feedback is not always available, some recommenders infer user preferences from the more abundant implicit feedback, which indirectly reflects opinion through observing user behavior [20]. Types of implicit feedback include purchase history, browsing history, search patterns, or even mouse movements. For example, a user who purchased many books by the same author probably likes that author. This chapter focuses on models suitable for explicit feedback. Nonetheless, we recognize the importance of implicit feedback, an especially valuable information source for users who do not provide much explicit feedback. Hence, we show how to address implicit feedback within the models as a secondary source of information.

In order to establish recommendations, CF systems need to relate two fundamentally different entities: items and users. There are two primary approaches to facilitate such a comparison, which constitute the two main techniques of CF: the neighborhood approach and latent factor models. Neighborhood methods focus on relationships between items or, alternatively, between users. An item-item approach models the preference of a user to an item based on ratings of similar items by the same user. Latent factor models, such as matrix factorization (aka, SVD), comprise an alternative approach by transforming both items and users to the same latent factor space. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback.

Producing more accurate prediction methods requires deepening their foundations and reducing reliance on arbitrary decisions. In this chapter, we describe a variety of recent improvements to the primary CF modeling techniques. Yet, the quest for more accurate models goes beyond this. At least as important is the identification of all the signals, or features, available in the data. Conventional techniques address the sparse data of user-item ratings. Accuracy significantly improves by also utilising other sources of information. One prime example includes all kinds of temporal effects reflecting the dynamic, time-drifting nature of user-item interactions. No less important is listening to hidden feedback such as which items users chose to rate (regardless of rating values). Rated items are not selected at random, but rather reveal interesting aspects of user



preferences, going beyond the numerical values of the ratings. Section 3.3 surveys matrix factorization techniques, which combine implementation convenience with a relatively high accuracy. This has made them the preferred technique for addressing the largest publicly available dataset—the Netflix data.

### 2.3.2 Baseline

We are given ratings for  $m$  users (aka customers) and  $n$  items (aka products). We reserve special indexing letters to distinguish users from items: for users  $u; v$ , and for items  $i; j; l$ . A rating  $r_{ui}$  indicates the preference by user  $u$  of item  $i$ , where high values mean stronger preference. For example, values can be integers ranging from 1 (star) indicating no interest to 5 (stars) indicating a strong interest. We distinguish predicted ratings from known ones, by using the notation  $\hat{r}_{ui}$  for the predicted value of  $r_{ui}$ .

The scalar  $t_{ui}$  denotes the time of rating  $r_{ui}$ . One can use different time units, based on what is appropriate for the application at hand. For example, when time is measured in days, then  $t_{ui}$  counts the number of days elapsed since some early time point. Usually the vast majority of ratings are unknown. For example, in the Netflix data 99% of the possible ratings are missing because a user typically rates only a small portion of the movies. The  $(u, i)$  pairs for which  $r_{ui}$  is known are stored in the set  $\mathcal{K} = \{(u, i) | r_{ui} \text{ is known}\}$ . Each user  $u$  is associated with a set of items denoted by  $R.u/$ , which contains all the items for which ratings by  $u$  are available. Likewise,  $R.i/$  denotes the set of users who rated item  $i$ . Sometimes, we also use a set denoted by  $\lambda$ , which contains all items for which  $u$  provided an implicit preference (items that he rented/purchased/watched, etc.).

### Baseline Predictors

CF models try to capture the interactions between users and items that produce the different rating values. However, much of the observed rating values are due to effects associated with either users or items, independently of their interaction. A principal example is that typical CF data exhibit large user and item biases—i.e., systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others. We will encapsulate those effects, which do not involve user-item interaction, within the baseline predictors (also known as biases). Because these predictors tend to capture much of the observed signal, it is vital to model them accurately. Such modeling enables isolating the part of the signal that truly represents user-item interaction, and subjecting it to more appropriate user preference models. Denote by  $\mu$  the overall average rating. A baseline prediction for an unknown

rating  $r_{ui}$  is denoted by  $b_{ui}$  and accounts for the user and item effects:

$$b_{ui} = \mu + b_u + b_i$$

The parameters  $b_u$  and  $b_i$  indicate the observed deviations of user  $u$  and item  $i$ , respectively, from the average. For example, suppose that we want a baseline predictor for the rating of the movie Titanic by user Joe. Now, say that the average rating over all movies,  $\mu$ , is 3.7 stars. Furthermore, Titanic is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the baseline predictor for Titanic's rating by Joe would be 3.9 stars by calculating  $3.7 - 0.3 + 0.5$ . In order to estimate  $b_u$  and  $b_i$  one can solve the least squares problem

$$\min_{b_*} \sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 \left( \sum_u b_u^2 + \sum_i b_i^2 \right)$$

Here, the first term  $\sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2$  is the given ratings. The regularizing term  $\lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$  avoids overfitting by penalizing the magnitudes of the parameters. This least square problem can be solved fairly efficiently by the method of stochastic gradient descent.

For the Netflix data the mean rating  $\mu$  is 3.6. As for the learned user biases ( $b_u$ ), their average is 0.044 with standard deviation of 0.41. The average of their absolute values ( $|b_u|$ ) is: 0.32. The learned item biases ( $b_i$ ) average to -0.26 with a standard deviation of 0.48. The average of their absolute values ( $|b_i|$ ) is 0.43.

An easier, yet somewhat less accurate way to estimate the parameters is by decoupling the calculation of the  $b_i$ 's from the calculation of the  $b_u$ 's. First, for each item  $i$  we set

$$b_i = \frac{\sum_{u \in \mathcal{R}(i)} (r_{ui} - \mu)}{\lambda_2 + |\mathcal{R}(i)|}$$

Then, for each user  $u$  we set:

$$b_u = \frac{\sum_{i \in \mathcal{R}(u)} (r_{ui} - \mu - b_i)}{\lambda_3 + |\mathcal{R}(u)|}$$

For Netflix data, the accuracy can be easily evaluated using RMSE:

$$\sqrt{\sum_{(u,i) \in \text{Testset}} (r_{ui} - \hat{r}_{ui})^2 / |\text{TestSet}|}$$

## Matrix Factorization

Latent factor models approach collaborative filtering with the holistic goal to uncover latent features that explain observed ratings; examples include pLSA [14], neural networks [22], Latent Dirichlet Allocation [7], and models that are induced by factorization of the user-item ratings matrix (also known as SVD-based models). Recently, matrix factorization models have gained popularity, thanks to their attractive accuracy and scalability.

In information retrieval, SVD is well established for identifying latent semantic factors [9]. However, applying SVD to explicit ratings in the CF domain raises difficulties due to the high portion of missing values. Conventional SVD is undefined when knowledge about the matrix is incomplete. Moreover, carelessly addressing only the relatively few known entries is highly prone to overfitting. Earlier works relied on imputation [15, 24], which fills in missing ratings and makes the rating matrix dense. However, imputation can be very expensive as it significantly increases the amount of data. In addition, the data may be considerably distorted due to inaccurate imputation. Hence, more recent works [4, 6, 10, 16, 21, 22, 26] suggested modeling directly only the observed ratings, while avoiding overfitting through an adequate regularized model.

In this section we describe several matrix factorization techniques, with increasing complexity and accuracy. We start with the basic model—“SVD”. Then, we show how to integrate other sources of user feedback in order to increase prediction accuracy, through the “SVD++ model”. Finally we deal with the fact that customer preferences for products may drift over time. Product perception and popularity are constantly changing as new selection emerges. Similarly, customer inclinations are evolving, leading them to ever redefine their taste. This leads to a factor model that addresses temporal dynamics for better tracking user behavior.

**SVD** [kitapta 3.3.1, 1.5 sayfa ve gerekirse SVD++(with implicit feedback)]

**Summary** In its basic form, matrix factorization characterizes both items and users by vectors of factors inferred from patterns of item ratings. High correspondence between item and user factors leads to recommendation of an item to a user. These methods deliver prediction accuracy superior to other published collaborative filtering techniques. At the same time, they offer a memory efficient compact model, which can be trained relatively easy. Those advantages, together with the implementation ease of gradient based matrix factorization model (SVD), made this the method of choice within the Netflix Prize competition.

What makes these techniques even more convenient is their ability to address several crucial aspects of the data. First, is the ability to integrate multiple forms of user

feedback. One can better predict user ratings by also observing other related actions by the same user, such as purchase and browsing history. The proposed SVD++ model leverages multiple sorts of user feedback for improving user profiling.

Another important aspect is the temporal dynamics that make users' tastes evolve over time. Each user and product potentially goes through a distinct series of changes in their characteristics. A mere decay of older instances cannot adequately identify communal patterns of behavior in time changing data. The solution we adopted is to model the temporal dynamics along the whole time period, allowing us to intelligently separate transient factors from lasting ones. The inclusion of temporal dynamics proved very useful in improving quality of predictions, more than various algorithmic enhancements.

### **Neighborhood Models**

The most common approach to CF is based on neighborhood models. Chapter 2 provides an extensive survey on this approach. Its original form, which was shared by virtually all earlier CF systems, is user-user based; see [13] for a good analysis. User-user methods estimate unknown ratings based on recorded ratings of like-minded users.

Later, an analogous item-item approach [18, 25] became popular. In those methods, a rating is estimated using known ratings made by the same user on similar items. Better scalability and improved accuracy make the item-item approach more favorable in many cases [2, 25, 26]. In addition, item-item methods are more amenable to explaining the reasoning behind predictions. This is because users are familiar with items previously preferred by them, but do not know those allegedly like-minded users. We focus mostly on item-item approaches, but the same techniques can be directly applied within a user-user approach; see also Sect. 3.5.2.2.

In general, latent factor models offer high expressive ability to describe various aspects of the data. Thus, they tend to provide more accurate results than neighborhood models. However, most literature and commercial systems (e.g., those of Amazon [18] and TiVo [1]) are based on the neighborhood models. The prevalence of neighborhood models is partly due to their relative simplicity. However, there are more important reasons for real life systems to stick with those models. First, they naturally provide intuitive explanations of the reasoning behind recommendations, which often enhance user experience beyond what improved accuracy may achieve. Second, they can provide immediate recommendations based on newly entered user feedback.

The structure of this section is as follows. First, we describe how to estimate the similarity between two items, which is a basic building block of most neighborhood techniques. Then, we move on to the widely used similarity-based neighborhood

method, which constitutes a straightforward application of the similarity weights. We identify certain limitations of this similarity based approach. As a consequence, in Sect. 3.4.3 we suggest a way to solve these issues, thereby improving prediction accuracy at the cost of a slight increase in computation time.

[3.4.1 ve 3.4.2 banko] [3.4.3'te yeni bir neighborhood modeli anlatiyor, belki o eger kullanilirsan]

**Summary** Collaborative filtering through neighborhood-based interpolation is probably the most popular way to create a recommender system. Three major components characterize the neighborhood approach: (1) data normalization, (2) neighbor selection, and (3) determination of interpolation weights.

Normalization is essential to collaborative filtering in general, and in particular to the more local neighborhood methods. Otherwise, even more sophisticated methods are bound to fail, as they mix incompatible ratings pertaining to different unnormalized users or items. We described a suitable approach to data normalization, based around baseline predictors.

Neighborhood selection is another important component. It is directly related to the employed similarity measure. Here, we emphasized the importance of shrinking unreliable similarities, in order to avoid detection of neighbors with a low rating support.

Finally, the success of neighborhood methods depends on the choice of the interpolation weights, which are used to estimate unknown ratings from neighboring known ones. Nevertheless, most known methods lack a rigorous way to derive these weights. We showed how the interpolation weights can be computed as a global solution to an optimization problem that precisely reflects their role.

[kullanirsan ve yer kalirsan 3.5]

[yer kalirsan ve matrix factorization kullanirsan 3.6]

[taxonomy vs olayina girersen 4]

## 2.4 Content-Based Filtering

### 2.4.1 Introduction

[kopi peyst from semantics aware cbf, 4]

Content-based recommender systems (CBRSs) rely on item and user descriptions (content) to build item representations and user profiles to suggest items similar to those a target user already liked in the past. The basic process of producing content-based recommendations consists in matching up the attributes of the target user profile, in which preferences and interests are stored, with the attributes of the items. The result is

a relevance score that predicts the target user's level of interest in those items. Usually, attributes for describing an item are features extracted from metadata associated to that item, or textual features extracted directly from the item description. The content extracted from metadata is often too short and not sufficient to correctly define the user interests, while the use of textual features involves a number of complications when learning a user profile due to natural language ambiguity. Polysemy, synonymy, multi-word expressions, named entity recognition and disambiguation are inherent problems of traditional keyword-based profiles, which are not able to go beyond the usage of lexical/syntactic structures to infer the user interest in topics.

The ever increasing interest in semantic technologies and the availability of several open knowledge sources, such as Wikipedia, DBpedia, Freebase, and BabelNet have fueled recent progress in the field of CBRs. Novel research works have introduced semantic techniques that shift from a keyword-based to a concept-based representation of items and user profiles. These observations make very relevant the integration of proper techniques for deep content analytics borrowed from Natural Language Processing (NLP) and Semantic Technologies, which is one of the most innovative lines of research in semantic recommender systems [61].

We roughly classify semantic techniques into top-down and bottom-up approaches. Top-down approaches rely on the integration of external knowledge, such as machine readable dictionaries, taxonomies (or IS-A hierarchies), thesauri or ontologies (with or without value restrictions and logical constraints), for annotating items and representing user profiles in order to capture the semantics of the target user information needs. The main motivation behind top-down approaches is the challenge of providing recommender systems with the linguistic knowledge and common sense knowledge, as well as the cultural background which characterize the human ability of interpreting documents expressed in natural language and reasoning on their meaning.

On the other side, bottom-up approaches exploit the so-called geometric metaphor of meaning to represent complex syntagmatic and paradigmatic relations between words in high-dimensional vector spaces. According to this metaphor, each word (and each document as well) can be represented as a point in a vector space. The peculiarity of these models is that the representation is learned by analyzing the context in which the word is used, in a way that terms (or documents) similar to each other are close in the space. For this reason bottom-up approaches are also called distributional models. One of the great virtues of these approaches is that they are able to induce the semantics of terms by analyzing their use in large corpora of textual documents using unsupervised mechanisms, as evidenced by the recent advances of machine translation techniques [52, 83].

This chapter describes a variety of semantic approaches, both top-down and bottom-up, and shows how to leverage them to build a new generation of semantic CBRs that

we call semantics-aware content-based recommender systems.

xd

This section reports an overview of the basic principles for building CBRs, the main techniques for representing items, learning user profiles and providing recommendations. The most important limitations of CBRs are also discussed, while the semantic techniques useful to tackle those limitations are introduced in the next sections.

The high level architecture of a content-based recommender system is depicted in Fig. 4.1. The recommendation process is performed in three steps, each of which is handled by a separate component:

[Figure 4.1 ya da benzeri ekle]

- **CONTENT ANALYZER**—When information has no structure (e.g. text), some kind of pre-processing step is needed to extract structured relevant information. The main responsibility of the component is to represent the content of items (e.g. documents, Web pages, news, product descriptions, etc.) coming from information sources in a form suitable for the next processing steps. Data items are analyzed by feature extraction techniques in order to shift item representation from the original information space to the target one (e.g. Web pages represented as keyword vectors). This representation is the input to the PROFILE LEARNER and FILTERING COMPONENT;
- **PROFILE LEARNER**—This module collects data representative of the user preferences and tries to generalize this data, in order to construct the user profile. Usually, the generalization strategy is realized through machine learning techniques [86], which are able to infer a model of user interests starting from items liked or disliked in the past. For instance, the PROFILE LEARNER of a Web page recommender can implement a relevance feedback method [113] in which the learning technique combines vectors of positive and negative examples into a prototype vector representing the user profile. Training examples are Web pages on which a positive or negative feedback has been provided by the user;
- **FILTERING COMPONENT**—This module exploits the user profile to suggest relevant items by matching the profile representation against that of items to be recommended. The result is a binary or continuous relevance judgment (computed using some similarity metrics [57]), the latter case resulting in a ranked list of potentially interesting items. In the above mentioned example, the matching is realized by computing the cosine similarity between the prototype vector and the item vectors.

The first step of the recommendation process is the one performed by the CONTENT ANALYZER, that usually borrows techniques from Information Retrieval systems

[6, 118]. Item descriptions coming from Information Source are processed by the CONTENT ANALYZER, that extracts features (keywords, n-grams, concepts, . . . ) from unstructured text to produce a structured item representation, stored in the repository Represented Items.

In order to construct and update the profile of the active user  $ua$  (user for which recommendations must be provided) her reactions to items are collected in some way and recorded in the repository Feedback. These reactions, called annotations [51] or feedback, together with the related item descriptions, are exploited during the process of learning a model useful to predict the actual relevance of newly presented items. Users can also explicitly define their areas of interest as an initial profile without providing any feedback. Typically, it is possible to distinguish between two kinds of relevance feedback: positive information (inferring features liked by the user) and negative information (i.e., inferring features the user is not interested in [58]). Two different techniques can be adopted for recording user's feedback. When a system requires the user to explicitly evaluate items, this technique is usually referred to as "explicit feedback"; the other technique, called "implicit feedback", does not require any active user involvement, in the sense that feedback is derived from monitoring and analyzing user's activities. Explicit evaluations indicate how relevant or interesting an item is to the user [111]. Explicit feedback has the advantage of simplicity, albeit the adoption of numeric/symbolic scales increases the cognitive load on the user, and may not be adequate for catching user's feeling about items. Implicit feedback methods are based on assigning a relevance score to specific user actions on an item, such as saving, discarding, printing, bookmarking, etc. The main advantage is that they do not require a direct user involvement, even though biasing is likely to occur, e.g. interruption of phone calls while reading.

In order to build the profile of the active user  $ua$ , the training set  $TRa$  for  $ua$  must be defined.  $TRa$  is a set of pairs  $\langle I_k, rk \rangle$ , where  $rk$  is the rating provided by  $ua$  on the item representation  $I_k$ . Given a set of item representation labeled with ratings, the PROFILE LEARNER applies supervised learning algorithms to generate a predictive model—the user profile—which is usually stored in a profile repository for later use by the FILTERING COMPONENT. After the user profile has been learned, the FILTERING COMPONENT predicts whether a new item is likely to be of interest for the active user, by comparing features in the item representation to those in the representation of user preferences (stored in the user profile).

User tastes usually change in time, therefore up-to-date information must be maintained and provided to the PROFILE LEARNER in order to automatically update the user profile. Further feedback is gathered on generated recommendations by letting users state their satisfaction or dissatisfaction with items in  $La$ . After gathering that feedback, the learning process is performed again on the new training set, and the



resulting profile is adapted to the updated user interests. The iteration of the feedback-learning cycle over time enables the system to take into account the dynamic nature of user preferences.

### **Text to Vector Space Model**

[4.2.1 tf-idf, cosine similarity falan]

### **Methods for Learning User Profile**

[4.2.2.1 eger naive bayes kullanırsan 4.2.2.1, 4.2.2.2 bilmiyorum, 4.2.2.3 kesin]

### **Advantages and Drawbacks of Content-Based Filtering**

The adoption of the content-based recommendation paradigm has several advantages when compared to the collaborative one:

- **USER INDEPENDENCE**—Content-based recommenders exploit solely ratings provided by the active user to build her own profile. Instead, collaborative filtering methods need ratings from other users in order to find the “nearest neighbors” of the active user, i.e., users that have similar tastes since they rated the same items similarly. Then, only the items that are most liked by the neighbors of the active user will be recommended;
- **TRANSPARENCY**—Explanations on how the recommender system works can be provided by explicitly listing content features or descriptions that caused an item to occur in the list of recommendations. Those features are indicators to consult in order to decide whether to trust a recommendation. Conversely, collaborative systems are black boxes since the only explanation for an item recommendation is that unknown users with similar tastes liked that item;
- **NEW ITEM**—Content-based recommenders are capable of recommending items not yet rated by any user. As a consequence, they do not suffer from the first-rater problem, which affects collaborative recommenders which rely solely on users’ preferences to make recommendations. Therefore, until the new item is rated by a substantial number of users, the system would not be able to recommend it.

Nonetheless, content-based systems have several shortcomings:

- **LIMITED CONTENT ANALYSIS**—Content-based techniques have a natural limit in the number and type of features that are associated, whether automatically

or manually, with the objects they recommend. Domain knowledge is often needed, e.g., for movie recommendations the system needs to know the actors and directors, and sometimes, domain ontologies are also needed. No content-based recommendation system can provide suitable suggestions if the analyzed content does not contain enough information to discriminate items the user likes from items the user does not like. Some representations capture only certain aspects of the content, but there are many others that would influence a user's experience. For instance, often there is not enough information in the word frequency to model the user interests in jokes or poems, while techniques for affective computing would be most appropriate. Again, for Web pages, feature extraction techniques from text completely ignore aesthetic qualities and additional multimedia information. Furthermore, CBRs based on a string matching approach suffer from problems of:

- POLYSEMY, the presence of multiple meanings for one word;
  - SYNONYMY, multiple words with the same meaning;
  - MULTI-WORD EXPRESSIONS, the difficulty to assign the correct properties to a sequence of two or more words whose properties are not predictable from the properties of the individual words;
  - ENTITY IDENTIFICATION or NAMED ENTITY RECOGNITION, the difficulty to locate and classify elements in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, etc.
  - ENTITY LINKING or NAMED ENTITY DISAMBIGUATION, the difficulty of determining the identity (often called the reference) of entities mentioned in text.
- OVER-SPECIALIZATION — Content-based recommenders have no inherent method for finding something unexpected. The system suggests items whose scores are high when matched against the user profile, hence the user is going to be recommended items similar to those already rated. This drawback is also called lack of serendipity problem to highlight the tendency of the content-based systems to produce recommendations with a limited degree of novelty. To give an example, when a user has only rated movies directed by Stanley Kubrick, she will be recommended just that kind of movies. A “perfect” content-based technique would rarely find anything novel, limiting the range of applications for which it would be useful.
  - NEW USER—Enough ratings have to be collected before a content-based rec-

ommender system can really understand user preferences and provide accurate recommendations. Therefore, when few ratings are available, as for a new user, the system will not be able to provide reliable recommendations.

[4.3 Eger textten anlam cikarma falan yaparsan eklersin] [4.4 Eger textten anlam cikarma falan yaparsan eklersin, bir oncekinin daha basiti ama iste insan olmadan halletme gibi] [4.5 ayni sekilde]

## 2.4.2 Conclusion

[Biraz 4.6 dan alabilirsin ama hepsi alakali degil]

## 2.5 Knowledge-Based Recommender Systems

[5. bolum ve baska kaynaklar eger kullanirsan]

## 2.6 Evaluation Metrics

### 2.6.1 Introduction

[kopi peyst, 8.1]

Recommender systems can now be found in many modern applications that expose the user to a huge collections of items. Such systems typically provide the user with a list of recommended items they might prefer, or predict how much they might prefer each item. These systems help users to decide on appropriate items, and ease the task of finding preferred items in the collection.

For example, the DVD rental provider Netflix<sup>1</sup> displays predicted ratings for every displayed movie in order to help the user decide which movie to rent. The online book retailer Amazon<sup>2</sup> provides average user ratings for displayed books, and a list of other books that are bought by users who buy a specific book. Microsoft provides many free downloads for users, such as bug fixes, products and so forth. When a user downloads some software, the system presents a list of additional items that are downloaded together. All these systems are typically categorized as recommender systems, even though they provide diverse services.

In the past decade, there has been a vast amount of research in the field of recommender systems, mostly focusing on designing new algorithms for recommendations. An application designer who wishes to add a recommender system to her application has a large variety of algorithms at her disposal, and must make a decision about the most appropriate algorithm for her goals. Typically, such decisions are based on

experiments, comparing the performance of a number of candidate recommenders. The designer can then select the best performing algorithm, given structural constraints such as the type, timeliness and reliability of availability data, allowable memory and CPU footprints. Furthermore, most researchers who suggest new recommendation algorithms also compare the performance of their new algorithm to a set of existing approaches. Such evaluations are typically performed by applying some evaluation metric that provides a ranking of the candidate algorithms (usually using numeric scores).

Initially most recommenders have been evaluated and ranked on their prediction power—their ability to accurately predict the user’s choices. However, it is now widely agreed that accurate predictions are crucial but insufficient to deploy a good recommendation engine. In many applications people use a recommender system for more than an exact anticipation of their tastes. Users may also be interested in discovering new items, in rapidly exploring diverse items, in preserving their privacy, in the fast responses of the system, and many more properties of the interaction with the recommendation engine. We must hence identify the set of properties that may influence the success of a recommender system in the context of a specific application. Then, we can evaluate how the system preforms on these relevant properties.

In this chapter we review the process of evaluating a recommendation system. We discuss three different types of experiments; offline, user studies and online experiments. Often it is easiest to perform offline experiments using existing data sets and a protocol that models user behavior to estimate recommender performance measures such as prediction accuracy. A more expensive option is a user study, where a small set of users is asked to perform a set of tasks using the system, typically answering questions afterwards about their experience. Finally, we can run large scale experiments on a deployed system, which we call online experiments. Such experiments evaluate the performance of the recommenders on real users which are oblivious to the conducted experiment. We discuss what can and cannot be evaluated for each of these types of experiments.

We can sometimes evaluate how well the recommender achieves its overall goals. For example, we can check an e-commerce website revenue with and without the recommender system and make an estimation of the value of the system to the website. In other cases, it can also be useful to evaluate how recommenders perform in terms of some specific properties, allowing us to focus on improving properties where they fall short. First, one must show that a property is indeed relevant to users and affect their experience. Then, we can design algorithms that improve upon these properties. In improving one property we may reduce the quality of another property, creating a trade-off between a set of properties. In many cases it is also difficult to say how these trade-offs affect the overall performance of the system, and we have to either

run additional experiments to understand this aspect, or use the opinions of domain experts.

This chapter focuses on property-directed evaluation of recommender algorithms. We provide an overview of a large set of properties that can be relevant for system success, explaining how candidate recommenders can be ranked with respect to these properties. For each property we discuss the relevant experiment types—offline, user study, and online experiments—and explain how an evaluation can be conducted in each case. We explain the difficulties and outline the pitfalls in evaluating each property. For all these properties we focus on ranking recommenders on that property, assuming that better handling the property will improve user experience.

We also review a set of previous suggestions for evaluating recommender systems, describing a large set of popular methods and placing them in the context of the properties that they measure. We especially focus on the widely researched accuracy and ranking measurements, describing a large set of evaluation metrics for these properties. For other, less studied properties, we suggest guidelines from which specific measures can be derived. We provide examples of such specific implementations where appropriate.

The rest of the chapter is structured as follows. In Sect. 8.2 we discuss the different experimental settings in which recommender systems can be evaluated, discussing the appropriate use of offline experiments, user studies, and online trials. We also outline considerations that go into making reliable decisions based on these experiments, including generalization and statistical significance of results. In Sect. 8.3 we describe a large variety of properties of recommender systems that may impact their performance, as well as metrics for measuring these properties. Finally, we conclude in Sect. 8.4.

### **2.6.2 Experimental Settings**

[8.2 kitap] In this section we describe three levels of experiments that can be used in order to compare several recommenders. The discussion below is motivated by evaluation protocols in related areas such as machine learning and information retrieval, highlighting practices relevant to evaluating recommender systems. The reader is referred to publications in these fields for more detailed discussions [17, 61, 75].

We begin with offline experiments, which are typically the easiest to conduct, as they require no interaction with real users. We then describe user studies, where we ask a small group of subjects to use the system in a controlled environment, and then report on their experience. In such experiments we can collect both quantitative and qualitative information about the systems, but care must be taken to consider various biases in the experimental design. Finally, perhaps the most trustworthy experiment is when the system is used by a pool of real users, typically unaware of the experiment.

While in such an experiment we are able to collect only certain types of data, this experimental design is closest to reality.

In all experimental scenarios, it is important to follow a few basic guidelines in general experimental studies:

- **Hypothesis:** before running the experiment we must form an hypothesis. It is important to be concise and restrictive about this hypothesis, and design an experiment that tests the hypothesis. For example, an hypothesis can be that algorithm A better predicts user ratings than algorithm B. In that case, the experiment should test the prediction accuracy, and not other factors. Other popular hypothesis in recommender system research can be that algorithm A scales better to larger datasets than algorithm B, that system A gains more user trust than system B, or that recommendation user interface A is preferred by users to interface B.
- **Controlling variables:** when comparing a few candidate algorithms on a certain hypothesis, it is important that all variables that are not tested will stay fixed. For example, suppose that in a movie recommendation system, we switch from using algorithm A to algorithm B, and notice that the number of movies that users watch increases. In this situation, we cannot tell whether the change is due to the change in algorithm, or whether something else changed at about the same time. If instead, we randomly assign users to algorithms A and B, and notice that users assigned to algorithm A watch more movies than those who are assigned to algorithm B, we can be confident that this is due to algorithm A.
- **Generalization power:** when drawing conclusions from experiments, we may desire that our conclusions generalize beyond the immediate context of the experiments. When choosing an algorithm for a real application, we may want our conclusions to hold on the deployed system, and generalize beyond our experimental data set. Similarly, when developing new algorithms, we want our conclusions to hold beyond the scope of the specific application or data set that we experimented with. To increase the probability of generalization of the results we must typically experiment with several data sets or applications. It is important to understand the properties of the various data sets that are used. Generally speaking, the more diverse the data used, the more we can generalize the results.

### **Offline Experiments**

An offline experiment is performed by using a pre-collected data set of users choosing or rating items. Using this data set we can try to simulate the behavior of users that

interact with a recommendation system. In doing so, we assume that the user behavior when the data was collected will be similar enough to the user behavior when the recommender system is deployed, so that we can make reliable decisions based on the simulation. Offline experiments are attractive because they require no interaction with real users, and thus allow us to compare a wide range of candidate algorithms at a low cost. The downside of offline experiments is that they can answer a very narrow set of questions, typically questions about the prediction power of an algorithm. In particular, we must assume that users' behavior when interacting with a system including the recommender system chosen will be modeled well by the users' behavior prior to that system's deployment. Thus we cannot directly measure the recommender's influence on user behavior in this setting.

Therefore, the goal of the offline experiments is to filter out inappropriate approaches, leaving a relatively small set of candidate algorithms to be tested by the more costly user studies or online experiments. A typical example of this process is when the parameters of the algorithms are tuned in an offline experiment, and then the algorithm with the best tuned parameters continues to the next phase.

**Data Sets for Offline Experiments** As the goal of the offline evaluation is to filter algorithms, the data used for the offline evaluation should match as closely as possible the data the designer expects the recommender system to face when deployed online. Care must be exercised to ensure that there is no bias in the distributions of users, items and ratings selected. For example, in cases where data from an existing system (perhaps a system without a recommender) is available, the experimenter may be tempted to pre-filter the data by excluding items or users with low counts, in order to reduce the costs of experimentation. In doing so, the experimenter should be mindful that this involves a trade-off, since this introduces a systematic bias in the data. If necessary, randomly sampling users and items may be a preferable method for reducing data, although this can also introduce other biases into the experiment (e.g. this could tend to favor algorithms that work better with more sparse data). Sometimes, known biases in the data can be corrected for by techniques such as reweighing data, but correcting biases in the data is often difficult.

Another source of bias may be the data collection itself. For example, users may be more likely to rate items that they have strong opinions on, and some users may provide many more ratings than others. Furthermore, users tend to rate items that they like, and avoid exploring, and hence rating, items that they will not like. For example, a person who doesn't like horror movies will tend not to watch them, would not explore the list of available horror movies for rental, and would not rate them. Thus, the set of items on which explicit ratings are available may be biased by the ratings

themselves. This is often known as the not missing at random assumption [47]. Once again, techniques such as resampling or reweighting the test data [70, 71] may be used to attempt to correct such biases.

**Simulating User Behavior** In order to evaluate algorithms offline, it is necessary to simulate the online process where the system makes predictions or recommendations, and the user corrects the predictions or uses the recommendations. This is usually done by recording historical user data, and then hiding some of these interactions in order to simulate the knowledge of how a user will rate an item, or which recommendations a user will act upon. There are a number of ways to choose the ratings/selected items to be hidden. Once again, it is preferable that this choice be done in a manner that simulates the target application as closely as possible. In many cases, though, we are restricted by the computational cost of an evaluation protocol, and must make compromises in order to execute the experiment over large data sets.

Ideally, if we have access to time-stamps for user selections, we can simulate what the systems predictions would have been, had it been running at the time the data set was collected [11]. We can begin with no available prior data for computing predictions, and step through user selections in temporal order, attempting to predict each selection and then making that selection available for use in future predictions. For large data sets, a simpler approach is to randomly sample test users, randomly sample a time just prior to a user action, hide all selections (of all users) after that instant, and then attempt to recommend items to that user. This protocol requires changing the set of given information prior to each recommendation, which can still be computationally quite expensive.

An even cheaper alternative is to sample a set of test users, then sample a single test time, and hide all items after the sampled test time for each test user. This simulates a situation where the recommender system is built as of the test time, and then makes recommendations without taking into account any new data that arrives after the test time. Another alternative is to sample a test time for each test user, and hide the test user's items after that time, without maintaining time consistency across users. This effectively assumes that the sequence in which items are selected is important, not the absolute times when the selections are made. A final alternative is to ignore time. We would first sample a set of test users, then sample the number  $n_a$  of items to hide for each user  $a$ , and finally sample  $n_a$  items to hide. This assumes that the temporal aspects of user selections are unimportant. We may be forced to make this assumption if the timestamps of user actions are not known. All three of the latter alternatives partition the data into a single training set and single test set. It is important to select an alternative that is most appropriate for the domain and task of interest, given the



constraints, rather than the most convenient one.

A common protocol used in many research papers is to use a fixed number of known items or a fixed number of hidden items per test user (so called “given  $n$ ” or “all but  $n$ ” protocols). This protocol may be useful for diagnosing algorithms and identifying in which cases they work best. However, when we wish to make decisions on the algorithm that we will use in our application, we must ask ourselves whether we are truly interested in presenting recommendations only for users who have rated exactly  $n$  items, or are expected to rate exactly  $n$  items more. If that is not the case, then results computed using these protocols have biases that make them unreliable in predicting the performance of the algorithms online, and these protocols should be avoided.

### **User Studies**

Many recommendation approaches rely on the interaction of users with the system (see, e.g., Chaps. 24, 5, 10, and 18). It is very difficult to create a reliable simulation of users interactions with the system, and thus, offline testing are difficult to conduct. In order to properly evaluate such systems, real user interactions with the system must be collected. Even when offline testing is possible, interactions with real users can still provide additional information about the system performance. In these cases we typically conduct user studies.

We provide here a summarized discussion of the principles of user studies for the evaluation of recommender systems. The interested reader can find an in depth discussion in Chap. 9. A user study is conducted by recruiting a set of test subjects, and asking them to perform several tasks requiring an interaction with the recommender system. While the subjects perform the tasks, we observe and record their behavior, collecting any number of quantitative measurements, such as what portion of the task was completed, the accuracy of the task results, or the time taken to perform the task. In many cases we can ask qualitative questions, before, during, and after the task is completed. Such questions can collect data that is not directly observable, such as whether the subject enjoyed the user interface, or whether the user perceived the task as easy to complete.

A typical example of such an experiment is to test the influence of a recommendation algorithm on the browsing behavior of news stories. In this example, the subjects are asked to read a set of stories that are interesting to them, in some cases including related story recommendations and in some cases without recommendations. We can then check whether the recommendations are used, and whether people read different stories with and without recommendations.

We can collect data such as how many times a recommendation was clicked, and even, in certain cases, track eye movement to see whether a subject looked at a rec-

ommendation. Finally, we can ask qualitative questions such as whether the subject thought the recommendations were relevant [30, 32].

Of course, in many other research areas user studies are a central tool, and thus there is much literature on the proper design of user studies. This section only overviews the basic considerations that should be taken when evaluating a recommender system through a user study, and the interested reader can find much deeper discussions elsewhere (see. e.g. [7]).

**Advantages and Disadvantages** User studies can perhaps answer the widest set of questions of all three experimental settings that we survey here. Unlike offline experiments this setting allows us to test the behavior of users when interacting with the recommender system, and the influence of the recommendations on user behavior. In the offline case we typically make assumptions such as “given a relevant recommendation the user is likely to use it” which are tested in the user study. Second, this is the only setting that allows us to collect qualitative data that is often crucial for interpreting the quantitative results. Also, we can typically collect in this setting a large set of quantitative measurements because the users can be closely monitored while performing the tasks.

User studies however have some disadvantages. Primarily, user studies are very expensive to conduct[39]; collecting a large set of subjects and asking them to perform a large enough set of tasks is costly in terms of either user time, if the subjects are volunteers, or in terms of compensation if paid subjects are employed. Therefore, we must typically restrict ourselves to a small set of subjects and a relatively small set of tasks, and cannot test all possible scenarios. Furthermore, each scenario has to be repeated several times in order to make reliable conclusions, further limiting the range of distinct tasks that can be tested.

As these experiments are expensive to conduct we should collect as much data about the user interactions, in the lowest possible granularity. This will allow us later to study the results of the experiment in detail, analyzing considerations that were not obvious prior to the trial. This guideline can help us to reduce the need for successive trials to collect overlooked measurements.

Furthermore, in order to avoid failed experiments, such as applications that malfunction under certain user actions, researchers often execute pilot user studies. These are small scale experiments, designed not to collect statistical data, but to test the systems for bugs and malfunctions. In some cases, the results of these pilot studies are then used to improve the recommender. If this is the case, then the results of the pilot become “tainted”, and should not be used when computing measurements in the final user study.

Another important consideration is that the test subjects must represent as closely as possible the population of users of the real system. For example, if the system is designed to recommend movies, the results of a user study over avid movie fans may not carry to the entire population. This problem is most persistent when the participants of the study are volunteers, as in this case people who are originally more interested in the application may tend to volunteer more readily.

However, even when the subjects represent properly the true population of users, the results can still be biased because they are aware that they are participating in an experiment. For example, it is well known that paid subjects tend to try and satisfy the person or company conducting the experiment [60]. If the subjects are aware of the hypothesis that is tested they may unconsciously provide evidence that supports it. To accommodate that, it is typically better not to disclose the goal of the experiment prior to collecting data. Another, more subtle effect occurs when the payment to subjects takes the form of a complete or partial subsidy of items they select. This may bias the data in cases where final users of the system are not similarly subsidized, as users' choices and preferences may be different when they pay full price. Unfortunately, avoiding this particular bias is difficult.

**Between vs. Within Subjects** As typically a user study compares a few candidate approaches, each candidate must be tested over the same tasks. To test all candidates we can either compare the candidates between subjects, where each subject is assigned to a candidate method and experiments with it, or within subjects, where each subject tests a set of candidates on different tasks [24].

Typically, within subjects experiments are more informative, as the superiority of one method cannot be explained by a biased split of users between candidate methods. It is also possible in this setting to ask comparative questions about the different candidates, such as which candidate the subject preferred. However, in these types of tests users are more conscious of the experiment, and hiding the distinctions between candidates is more difficult.

Between subjects experiments, also known as A-B testing (All Between), provide a setting that is closer to the real system, as each user experiments with a single treatment. Such experiments can also test long term effects of using the system, because the user is not required to switch systems. Thus we can test how the user becomes accustomed to the system, and estimate a learning curve of expertise. On the downside, when running between subjects experiments, typically more data is needed to achieve significant results. As such, between subjects experiments may require more users, or more interaction time for each user, and are thus more costly than within subjects experiments.

[ If needed 8.2.2.3]

**Questionnaires** User studies allow us to use the powerful questionnaire tool (e.g. [58]). Before, during, and after subjects perform their tasks we can ask them questions about their experience. These questions can provide information about properties that are difficult to measure, such as the subject's state of mind, or whether the subject enjoyed the system.

While these questions can provide valuable information, they can also provide misleading information. It is important to ask neutral questions, that do not suggest a "correct" answer. People may also answer untruthfully, for example when they perceive the answer as private, or if they think the true answer may put them in an unflattering position.

Indeed, vast amount of research was conducted in other areas about the art of questionnaire writing, and we refer the readers to that literature (e.g. [56]) for more details.

### **Online Evaluation**

In many realistic recommendation applications the designer of the system wishes to influence the behavior of users. We are therefore interested in measuring the change in user behavior when interacting with different recommender systems. For example, if users of one system follow the recommendations more often, or if some utility gathered from users of one system exceeds utility gathered from users of the other system, then we can conclude that one system is superior to the other, all else being equal.

The real effect of the recommender system depends on a variety of factors such as the user's intent (e.g. how specific their information needs are), the user's personality (Chap. 21), such as how much novelty vs. how much risk they are seeking, the user's context, e.g., what items they are already familiar with, how much they trust the system (Chap. 6), and the interface through which the recommendations are presented.

Thus, the experiment that provides the strongest evidence as to the true value of the system is an online evaluation, where the system is used by real users that perform real tasks. It is most trustworthy to compare a few systems online, obtaining a ranking of alternatives, rather than absolute numbers that are more difficult to interpret.

For this reason, many real world systems employ an online testing system [40], where multiple algorithms can be compared. Typically, such systems redirect a small percentage of the traffic to different alternative recommendation engine, and record the users interactions with the different systems.

There are a few considerations that must be made when running such tests. For example, it is important to sample (redirect) users randomly, so that the comparisons

between alternatives are fair. It is also important to single out the different aspects of the recommenders. For example, if we care about algorithmic accuracy, it is important to keep the user interface fixed. On the other hand, if we wish to focus on a better user interface, it is best to keep the underlying algorithm fixed.

In some cases, such experiments are risky. For example, a test system that provides irrelevant recommendations, may discourage the test users from using the real system ever again. Thus, the experiment can have a negative effect on the system, which may be unacceptable in commercial applications.

For these reasons, it is best to run an online evaluation last, after an extensive offline study provides evidence that the candidate approaches are reasonable, and perhaps after a user study that measures the user's attitude towards the system. This gradual process reduces the risk in causing significant user dissatisfaction.

Online evaluations are unique in that they allow direct measurement of overall system goals, such as long-term profit or user retention. As such, they can be used to understand how these overall goals are affected by system properties such as recommendation accuracy and diversity of recommendations, and to understand the trade-offs between these properties. However, since varying such properties independently is difficult, and comparing many algorithms through online trials is expensive, it can be difficult to gain a complete understanding of these relationships.

### **Drawing Reliable Conclusion**

In any type of experiment it is important that we can be confident that the candidate recommender that we choose will also be a good choice for the yet unseen data the system will be faced with in the future. As we explain above, we should exercise caution in choosing the data in an offline experiments, and the subjects in a user study, to best resemble the online application. Still, there is a possibility that the algorithm that performed best on this test set did so because the experiment was fortuitously suitable for that algorithm. To reduce the possibility of such statistical mishaps, we must perform significance testing on the results.

[Also rest of 8.2.4 on handbook, couple of pages]

### **2.6.3 Recommender System Properties**

In this section we survey a range of properties that are commonly considered when deciding which recommendation approach to select. As different applications have different needs, the designer of the system must decide on the important properties to measure for the concrete application at hand. Some of the properties can be traded-off, the most obvious example perhaps is the decline in accuracy when other properties (e.g.

diversity) are improved. It is important to understand and evaluate these trade-offs and their effect on the overall performance. However, the proper way of gaining such understanding without intensive online testing or deferring to the opinions of domain experts is still an open question.

Furthermore, the effect of many of these properties on the user experience is unclear, and depends on the application. While we can certainly speculate that users would like diverse recommendations or reported confidence bounds, it is essential to show that this is indeed important in practice. Therefore, when suggesting a method that improves one of these properties, one should also evaluate how changes in this property affect the user experience, either through a user study or through online experimentation.

Such an experiment typically uses a single recommendation method with a tunable parameter that affects the property being considered. For example, we can envision a parameter that controls the diversity of the list of recommendations. Then, subjects should be presented with recommendations based on a variety of values for this parameter, and we should measure the effect of the parameter on the user experience. We should measure here not whether the user noticed the change in the property, but whether the change in property has affected their interaction with the system. As is always the case in user studies, it is preferable that the subjects in a user study and users in an online experiment will not know the goal of the experiment. It is difficult to envision how this procedure could be performed in an offline setting because we need to understand the user response to this parameter.

Once the effects of the specific system properties in affecting the user experience of the application at hand are understood, we can use differences in these properties to select a recommender.

#### **2.6.4 User Preference**

As in this chapter we are interested in the selection problem, where we need to choose one out of a set of candidate algorithms, an obvious option is to run a user study (within subjects) and ask the participants to choose one of the systems [29]. This evaluation does not restrict the subjects to specific properties, and it is generally easier for humans to make such judgments than to give scores for the experience. Then, we can select the system that had the largest number of votes.

However, aside from the biases in user studies discussed earlier, there are additional concerns that we must be aware of. First, the above scheme assumes that all users are equal, which may not always be true. For example, an e-commerce website may prefer the opinion of users who buy many items to the opinion of users who only buy a single item. We therefore need to further weight the vote by the importance of the user, when applicable. Assigning the right importance weights in a user study may not be easy.

It may also be the case that users who preferred system A, only slightly preferred it, while users who preferred B, had a very low opinion on A. In this case, even if slightly more users preferred A we may still wish to choose B. To measure this we need non-binary answers for the preference question in the user study. Then, the problem of calibrating scores across users arises.

Finally, when we wish to improve a system, it is important to know why people favor one system over the other. Typically, it is easier to understand that when comparing specific properties. Therefore, while user satisfaction is important to measure, breaking satisfaction into smaller components is helpful to understand the system and improve it.

### 2.6.5 Notation

[kopi peyst kitap first chapters] [8.3.2 de daha detayli anlatiyor prediction accuracy konusunu sayfalarca, bunu onlarla birlestir ya da bunu sal belki de]

In order to give a formal definition of the item recommendation task, we introduce the following notation. The set of users in the recommender system will be denoted by  $U$ , and the set of items by  $I$ . Moreover, we denote by  $R$  the set of ratings recorded in the system, and write  $S$  the set of possible values for a rating (e.g.,  $S = [1..5]$  or  $S = \text{like; dislike}$ ). Also, we suppose that no more than one rating can be made by any user  $u \in U$  for a particular item  $i \in I$  and write  $r_{ui}$  this rating. To identify the subset of users that have rated an item  $i$ , we use the notation  $U_i$ . Likewise,  $I_u$  represents the subset of items that have been rated by a user  $u$ . Finally, the items that have been rated by two users  $u$  and  $v$ , i.e.  $I_u \cap I_v$ , is an important concept in our presentation, and we use  $I_{uv}$  to denote this concept. In a similar fashion,  $U_{ij}$  is used to denote the set of users that have rated both items  $i$  and  $j$ .

Two of the most important problems associated with recommender systems are the rating prediction and top-N recommendation problems. The first problem is to predict the rating that a user  $u$  will give his or her unrated item  $i$ . When ratings are available, this task is most often defined as a regression or (multi-class) classification problem where the goal is to learn a function  $f : U \times I \rightarrow S$  that predicts the rating  $f(u, i)$  of a user  $u$  for a new item  $i$ . Accuracy is commonly used to evaluate the performance of the recommendation method. Typically, the ratings  $R$  are divided into a training set  $R_{train}$  used to learn  $f$ , and a test set  $R_{test}$  used to evaluate the prediction accuracy. Two popular measures of accuracy are the Mean Absolute Error (MAE):

$$MAE(f) = \frac{1}{|R_{test}|} \sum_{r_{ui} \in R_{test}} |f(u, i) - r_{ui}|$$

and the Root Mean Squared Error (RMSE):

$$\text{RMSE}(f) = \sqrt{\frac{1}{|\mathcal{R}_{test}|} \sum_{r_{ui}} (f(u, i) - r_{ui})^2}$$

When ratings are not available, for instance, if only the list of items purchased by each user is known, measuring the rating prediction accuracy is not possible. In such cases, the problem of finding the best item is usually transformed into the task of recommending to an active user  $u_a$  a list  $L(u_a)$  containing  $N$  items likely to interest him or her [15, 59]. The quality of such method can be evaluated by splitting the items of  $I$  into a set  $I_{train}$ , used to learn  $L$ , and a test set  $I_{test}$ . Let  $T(u) \in I_u \cap I_{test}$  be the subset of test items that a user  $u$  found relevant. If the user responses are binary, these can be the items that  $u$  has rated positively. Otherwise, if only a list of purchased or accessed items is given for each user  $u$ , then these items can be used as  $T(u)$ . The performance of the method is then computed using the measures of precision and recall:

$$\text{Precision}(L) = \frac{1}{|u|} \sum_{u \in \mathcal{U}} |L(u) \cap T(u)| / |L(u)|$$

$$\text{Recall}(L) = \frac{1}{|u|} \sum_{u \in \mathcal{U}} |L(u) \cap T(u)| / |T(u)|$$

A drawback of this task is that all items of a recommendation list  $L.u/$  are considered equally interesting to user  $u$ . An alternative setting, described in [15], consists in learning a function  $L$  that maps each user  $u$  to a list  $L.u/$  where items are ordered by their “interestingness” to  $u$ . If the test set is built by randomly selecting, for each user  $u$ , a single item  $i_u$  of  $I_u$ , the performance of  $L$  can be evaluated with the Average Reciprocal Hit-Rank (ARHR):

$$\text{ARHR}(L) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{\text{rank}(i_u, L(u))}$$

where  $\text{rank}(i_u ; L(u))$  is the rank of item  $i_u$  in  $L(u)$ . A more extensive description of evaluation measures for recommender systems can be found in Chap. 8 of this book.

### Converge

As the prediction accuracy of a recommender system, especially in collaborative filtering systems, in many cases grows with the amount of data, some algorithms may provide recommendations with high quality, but only for a small portion of the items where they have huge amounts of data. This is often referred to as the long tail or heavy tail problem, where the vast majority of the items were selected or rated only by a handful of users, yet the total amount of evidence over these unpopular items is much more



than the evidence over the few popular items. The term coverage can refer to several distinct properties of the system that we discuss below.

[8.3.3 birkac sayfa var]

### **Confidence**

[8.3.4]

### **Trust**

While confidence is the system trust in its ratings (Chap.20), in trust we refer here to the user's trust in the system recommendation.<sup>4</sup> For example, it may be beneficial for the system to recommend a few items that the user already knows and likes. This way, even though the user gains no value from this recommendation, she observes that the system provides reasonable recommendations, which may increase her trust in the system recommendations for unknown items. Another common way of enhancing trust in the system is to explain the recommendations that the system provides (Chap. 10). Trust in the system is also called the credibility of the system.

If we do not restrict ourselves to a single method of gaining trust, such as the one suggested above, the obvious method for evaluating user trust is by asking users whether the system recommendations are reasonable in a user study [5, 14, 26, 57]. In an online test one could associate the number of recommendations that were followed with the trust in the recommender, assuming that higher trust in the recommender would lead to more recommendations being used. Alternatively, we could also assume that trust in the system is correlated with repeated users, as users who trust the system will return to it when performing future tasks. However, such measurements may not separate well other factors of user satisfaction, and may not be accurate. It is unclear how to measure trust in an offline experiment, because trust is built through an interaction between the system and a user.

### **Novelty**

[8.3.6]

### **Serendipity**

[8.3.7]

## Diversity

Diversity is generally defined as the opposite of similarity (Chap. 26). In some cases suggesting a set of similar items may not be as useful for the user, because it may take longer to explore the range of items. Consider for example a recommendation for a vacation [68], where the system should recommend vacation packages. Presenting a list with five recommendations, all for the same location, varying only on the choice of hotel, or the selection of attraction, may not be as useful as suggesting five different locations. The user can view the various recommended locations and request more details on a subset of the locations that are appropriate to her.

The most explored method for measuring diversity uses item-item similarity, typically based on item content, as in Sect.8.3.7. Then, we could measure the diversity of a list based on the sum, average, min, or max distance between item pairs, or measure the value of adding each item to the recommendation list as the new item's diversity from the items already in the list [8, 80]. The item-item similarity measurement used in evaluation can be different from the similarity measurement used by the algorithm that computes the recommendation lists. For example, we can use for evaluation a costly metric that produces more accurate results than fast approximate methods that are more suitable for online computations.

As diversity may come at the expense of other properties, such as accuracy [78], we can compute curves to evaluate the decrease in accuracy vs. the increase in diversity.

Example 8.4. In a book recommendation application, we are interested in presenting the user with a diverse set of recommendations, with minimal impact to accuracy. We use  $d_b; B/$  from Example 8.3 as the distance metric. Given candidate recommenders, each with a tunable parameter that controls the diversity of the recommendations, we train each algorithm over a range of values for the diversity parameters. For each trained model, we now compute a precision score, and a diversity score as follows; we take each recommendation list that an algorithm produces, and compute the distance of each item from the rest of the list, averaging the result to obtain a diversity score. We now plot the precision-diversity curves of the recommenders in a graph, and select the algorithm with the dominating curve.

In recommenders that assist in information search, we can assume that more diverse recommendations will result in shorter search interactions [68]. We could use this in an online experiment measuring interaction sequence length as a proxy for diversification. As is always the case in online testing, shorter sessions may be due to other factors of the system, and to validate this claim it is useful to experiment with different diversity thresholds using the same prediction engine before comparing different recommenders.

## Utility

Many e-commerce websites employ a recommender system in order to improve their revenue by, e.g., enhancing cross-sell. In such cases the recommendation engine can be judged by the revenue that it generates for the website [66]. In general, we can define various types of utility functions that the recommender tries to optimize. For such recommenders, measuring the utility, or the expected utility of the recommendations may be more significant than measuring the accuracy of recommendations. It is also possible to view many of the other properties, such as diversity or serendipity, as different types of utility functions, over single items or over lists. In this chapter, however, we define utility as the value that either the system or the user gains from a recommendation.

Utility can be measured cleanly from the perspective of the recommendation engine or the recommender system owner. Care must be taken, though, when measuring the utility that the user receives from the recommendations. First, user utilities or preferences are difficult to capture and model, and considerable research has focused on this problem [9, 25, 59]. Second, it is unclear how to aggregate user utilities across users for computing a score for a recommender. For example, it is tempting to use money as a utility thus selecting a recommender that minimizes user cost. However, under the diminishing returns assumption [69], the same amount of money does not have the same utility for people with different income levels. Therefore, the average cost per purchase, for example, is not a reasonable aggregation across users.

In an application where users rate items, it is also possible to use the ratings as a utility measurement [10]. For example, in movie ratings, where a five star movie is considered an excellent movie, we can assume that a recommending a five star movie has a higher utility for the user than recommending a movie that the user will rate with four stars. As users may interpret ratings differently, user ratings should be normalized before aggregating across users.

While we typically only assign positive utilities to successful recommendations, we can also assign negative utilities to unsuccessful recommendations. For example, if some recommended item offends the user, then we should punish the system for recommending it by assigning a negative utility. We can also add a cost to each recommendation, perhaps based on the position of the recommended item in the list, and subtract it from the utility of the item.

For any utility function, the standard evaluation of the recommender is to compute the expected utility of a recommendation. In the case where the recommender is trying to predict only a single item, such as when we evaluate the system on time-based splits and try to predict only the next item in the sequence, the value of a correct recommendation should simply be the utility of the item. In the task where

the recommender predicts  $n$  items we can use the sum of the utilities of the correct recommendations in the list. When negative utilities for failed recommendations are used, then the sum is over all recommendations, successful or failed. We can also integrate utilities into ranking measurements, as discussed in Sect. 8.3.2.3. Finally, we can normalize the resulting score using the maximal possible utility given the optimal recommendation list.

Evaluating utility in user studies and online is easy in the case of recommender utility. If the utility we optimize for is the revenue of the website, measuring the change in revenue between users of various recommenders is simple. When we try to optimize user utilities the online evaluation becomes harder, because users typically find it challenging to assign utilities to outcomes. In many cases, however, users can say whether they prefer one outcome to another. Therefore, we can try to elicit the user preferences [31] in order to rank the candidate methods.

### **Risk**

8.3.10

### **Robustness**

8.3.11

### **Privacy**

8.3.12

### **Adaptivity**

8.3.13

### **Scalability**

8.3.14

## **2.6.6 Conclusion**

In this chapter we discussed how recommendation algorithms could be evaluated in order to select the best algorithm from a set of candidates. This is an important step in the research attempt to find better algorithms, as well as in application design where a designer chooses an existing algorithm for their application. As such, many evaluation metrics have been used for algorithm selection in the past.

We describe the concerns that need to be addressed when designing offline and online experiments and user studies. We outline a few important measurements that one must take in addition to the score that the metric provides, as well as other considerations that should be taken into account when designing experiments for recommendation algorithms.

We specify a set of properties that are sometimes discussed as important for the recommender system. For each such property we suggest an experiment that can be used to rank recommenders with regards to that property. For less explored properties, we restrict ourselves to generic descriptions that could be applied to various manifestations of that property. Specific procedures that can be practically implemented can then be developed for the specific property manifestation based on our generic guidelines.

[If needed chapters 9 and 10 of handbook]

## **2.7 Novelty and Diversity**

### **2.7.1 Introduction**

[kopi peyst. handbook chapter 26] Accurately predicting the users' interests was the main direct or implicit drive of the recommender systems field in roughly the first decade and a half of the field's development. A wider perspective towards recommendation utility, including but beyond prediction accuracy, started to appear in the literature by the beginning of the 2000s [36, 70], taking views that began to realize the importance of novelty and diversity, among other properties, in the added value of recommendation [53, 90]. This realization grew progressively, reaching an upswing of activity by the turn of the past decade [1, 3, 20, 39, 75]. Today we might say that novelty and diversity are becoming an increasingly frequent part of evaluation practice. They are being included increasingly often among the reported effectiveness metrics of new recommendation approaches, and are explicitly targeted by algorithmic innovations time and again. And it seems difficult to conceive progress in the recommender systems field without considering these dimensions and further developing our understanding thereof. Even though dealing with novelty and diversity remains an active area of research and development, considerable progress has been achieved in these years in terms of the development of enhancement techniques, evaluation metrics, methodologies, and theory, and we deem the area is therefore ripe for a broad overview as we undertake in this chapter.

In this chapter we analyze the different motivations, notions and perspectives under which novelty and diversity can be understood and defined (Sect.26.2). We revise the evaluation procedures and metrics which have been developed in this area (Sect. 26.3),

as well as the algorithms and solutions to enhance novelty and/or diversity (Sect. 26.4). We analyze the relationship with the recent and prolific stream of work on diversity in Information Retrieval, as a confluent area with recommender systems, and discuss a unifying framework that aims to provide a common basis as comprehensive as possible to explain and interrelate different novelty and diversity perspectives (Sect. 26.5). We show some empirical results that illustrate the behavior of metrics and algorithms (Sect. 26.6), and close the chapter with a summary and discussion of the progress and perspectives in this area, and directions for future research (Sect. 26.7).

### **2.7.2 Novelty and Diversity in Recommender Systems**

Novelty can be generally understood as the difference between present and past experience, whereas diversity relates to the internal differences within parts of an experience. The difference between the two concepts is subtle and close connections can in fact be established, depending on the point of view one may take, as we shall discuss. The general notions of novelty and diversity can be particularized in different ways. For instance, if a music streaming service recommends us a song we have never heard before, we would say this recommendation brings some novelty. Yet if the song is, say, a very canonical music type by some very well known singer, the involved novelty is considerably less than we would get if the author and style of the music were also original for us. We might also consider that the song is even more novel if, for instance, few of our friends know about it. On the other hand, a music recommendation is diverse if it includes songs of different styles rather than different songs of very similar styles, regardless of whether the songs are original or not for us. Novelty and diversity are thus to some extent complementary dimensions, though we shall seek and discuss in this chapter the relationships between them. The motivations for enhancing the novelty and diversity of recommendations are manifold, as are the different angles one may take when seeking these qualities. This is also the case in other fields outside information systems, where novelty and diversity are recurrent topics as well, and considerable efforts have been devoted to casting clear definitions, equivalences and distinctions. We therefore start this chapter by overviewing the reasons for and the possible meanings of novelty and diversity in recommender systems, with a brief glance at related perspectives in other disciplines.

#### **Why Novelty and Diversity in Recommendation**

Bringing novelty and diversity into play as target properties of the desired outcome means taking a wider perspective on the recommendation problem concerned with final actual recommendation utility, rather than a single quality side such as accuracy

[53]. Novelty and diversity are not the only dimensions of recommendation utility one should consider aside from accuracy (see e.g. Chap. 8 for a comprehensive survey), but they are fundamental ones. The motivations for enhancing novelty and diversity in recommendations are themselves diverse, and can be founded in the system, user and business perspectives.

From the system point of view, user actions as implicit evidence of user needs involve a great extent of uncertainty as to what the actual user preferences really are. User clicks and purchases are certainly driven by user interests, but identifying what exactly in an item attracted the user, and generalizing to other items, involves considerable ambiguity. On top of that, system observations are a very limited sample of user activity, whereby recommendation algorithms operate on significantly incomplete knowledge. Furthermore, user interests are complex, highly dynamic, context-dependent, heterogeneous and even contradictory. Predicting the user needs is therefore an inherently difficult task, unavoidably subject to a non-negligible error rate. Diversity can be a good strategy to cope with this uncertainty and optimize the chances that at least some item pleases the user, by widening the range of possible item types and characteristics at which recommendations aim, rather than bet for a too narrow and risky interpretation of user actions. For instance, a user who has rated the movie “Rango” with the highest value may like it because—in addition to more specific virtues—it is a cartoon, a western, or because it is a comedy. Given the uncertainty about which of the three characteristics may account for the user preference, recommending a movie of each genre generally pays off more than recommending, say three cartoons, as far as three hits do not necessarily bring three times the gain of one hit—e.g. the user might rent just one recommended movie anyway—whereas the loss involved in zero hits is considerably worse than achieving a single hit. From this viewpoint we might say that diversity is not necessarily an opposing goal to accuracy, but in fact a strategy to optimize the gain drawn from accuracy in matching true user needs in an uncertain environment.

On the other hand, from the user perspective, novelty and diversity are generally desirable per se, as a direct source of user satisfaction. Consumer behaviorists have long studied the natural variety-seeking drive in human behavior [51]. The explanation of this drive is commonly divided into direct and derived motivations. The former refer to the inherent satisfaction obtained from “novelty, unexpectedness, change and complexity” [50], and a genuine “desire for the unfamiliar, for alternation among the familiar, and for information” [64], linking to the existence of an ideal level of stimulation, dependent on the individual. Satiation and decreased satisfaction results from the repeated consumption of a product or product characteristic in a decreasing marginal value pattern [25]. As preferences towards discovered products are developed, consumer behavior converges towards a balance between alternating choices and

favoring preferred products [16]. Derived motivations include the existence of multiple needs in people, multiple situations, or changes in people's tastes [51]. Some authors also explain diversity-seeking as a strategy to cope with the uncertainty about one's own future preference when one will actually consume the choices [44], as e.g. when we choose books and music for a trip. Moreover, novel and diverse recommendations enrich the user experience over time, helping expand the user's horizon. It is in fact often the case that we approach a recommender system with the explicit intent of discovering something new, developing new interests, and learning. The potential problems of the lack of diversity which may result from too much personalization has recently come to the spotlight with the well-known debate on the so-called filter bubble [60]. This controversy adds to the motivation for reconciling personalization with a healthy degree of diversity.

Diversity and novelty also find motivation in the underlying businesses in which recommendation technologies are deployed. Customer satisfaction indirectly benefits the business in the form of increased activity, revenues, and customer loyalty. Beyond this, product diversification is a well-known strategy to mitigate risk and expand businesses [49]. Moreover, selling in the long tail is a strategy to draw profit from market niches by selling less of more and getting higher profit margins on cheaper products [9].

All the above general considerations can be of course superseded by particular characteristics of the specific domain, the situation, and the goal of the recommendations, for some of which novelty and diversity are indeed not always needed. For instance, getting a list of similar products (e.g. photo cameras) to one we are currently inspecting may help us refine our choice among a large set of very similar options. Recommendations can serve as a navigational aid in this type of situation. In other domains, it makes sense to consume the same or very similar items again and again, such as grocery shopping, clothes, etc. The added value of recommendation is probably more limited in such scenarios though, where other kinds of tools may solve our needs (catalog browsers, shopping list assistants, search engines, etc.), and even in these cases we may appreciate some degree of variation in the mix every now and then.

### **Defining Novelty and Diversity**

Novelty and diversity are different though related notions, and one finds a rich variety of angles and perspectives on these concepts in the recommender system literature, as well as other fields such as sociology, economy, or ecology. As pointed out at the beginning of this section, novelty generally refers, broadly, to the difference between present and past experience, whereas diversity relates to the internal differences within parts of an experience. Diversity generally applies to a set of items or "pieces", and has



to do with how different the items or pieces are with respect to each other. Variants have been defined by considering different pieces and sets of items. In the basic case, diversity is assessed in the set of items recommended to each user separately (and typically averaged over all users afterwards) [90]. But global diversity across sets of sets of items has also been considered, such as the recommendations delivered to all users [3, 4, 89], recommendations by different systems to the same user [11], or recommendations to a user by the same system over time [46].

The novelty of a set of items can be generally defined as a set function (average, minimum, maximum) on the novelty of the items it contains. We may therefore consider novelty as primarily a property of individual items. The novelty of a piece of information generally refers to how different it is with respect to “what has been previously seen” or experienced. This is related to novelty in that when a set is diverse, each item is “novel” with respect to the rest of the set. Moreover, a system that promotes novel results tends to generate global diversity over time in the user experience; and also enhances the global “diversity of sales” from the system perspective. Multiple variants of novelty arise by considering the fact that novelty is relative to a context of experience, as we shall discuss.

Different nuances have been considered in the concept of novelty. A simple definition of novelty can consist of the (binary) absence of an item in the context of reference (prior experience). We may use adjectives such as unknown or unseen for this notion of identity-based novelty [75]. Long tail notions of novelty are elaborations of this concept, as they are defined in terms of the number of users who would specifically know an item [20, 61, 89]. But we may also consider how different or similar an unseen item is with respect to known items, generally— but not necessarily—on a graded scale. Adjectives such as unexpected, surprising and unfamiliar have been used to refer to this variant of novelty. Unfamiliarity and identity novelty can be related by trivially defining similarity as equality, i.e. two items are “similar” if and only if they are the same item. Finally, the notion of serendipity is used to mean novelty plus a positive emotional response— in other words, an item is serendipitous if it is novel—unknown or unfamiliar—and relevant [57, 88].

The present chapter is concerned with the diversity and novelty involved in recommendations, but one might also study the diversity (in tastes, behavior, demographics, etc.) of the end-user population, or the product stock, the sellers, or in general the environment in which recommenders operate. While some works in the field have addressed the diversity in user behavior [31, 72], we will mostly focus on those aspects a recommender system has a direct hold on, namely the properties of its own output.

### **Diversity in Other Fields**

Diversity is a recurrent theme in several fields, such as sociology, psychology, economy, ecology, genetics or telecommunications. One can establish connections and analogies from some—though not all—of them to recommender systems, and some equivalences in certain metrics, as we will discuss.

Diversity is a common keyword in sociology referring to cultural, ethnic or demographic diversity [47]. Analogies to recommender system settings would apply to the user population, which is mainly a given to the system, and therefore not within our main focus here. In economy, diversity is extensively studied in relation to different issues such as the players in a market (diversity vs. oligopolies), the number of different industries in which a firm operates, the variety of products commercialized by a firm, or investment diversity as a means to mitigate the risk involved in the volatility of investment value [49]. Of all such concepts, product and portfolio diversity most closely relate to recommendation, as mentioned in Sect. 26.2.1, as a general risk-mitigating principle and/or business growth strategy.

Behaviorist psychology has also paid extensive attention to the human drive for novelty and diversity [51]. Such studies, especially the ones focusing on consumer behavior, provide formal support to the intuition that recommender system users may prefer to find some degree of variety and surprise in the recommendations they receive, as discussed in Sect. 26.2.1.

An extensive strand of literature is devoted to diversity in ecology as well, where researchers have worked to considerable depth on formalizing the problem, defining and comparing a wide array of diversity metrics, such as the number of species (richness), Gini-Simpson and related indices, or entropy [62]. Such developments connect to aggregate recommendation diversity perspectives that deal with sets of recommendations as a whole, as we shall discuss in Sects. 26.2.3 and 26.5.3.3.

Finally, the issue of diversity has also attracted a great deal of attention in the Information Retrieval (IR) field. A solid body of theory, metrics, evaluation methodologies and algorithms has been developed in this scope in the last decade [6, 17, 21, 22, 24, 67, 84], including a dedicated search diversity task in four consecutive TREC editions starting in 2009 [23]. Search and recommendation are different problems, but have much in common: both tasks are about ranking a set of items to maximize the satisfaction of a user need, which may or may not have been expressed explicitly. It has in fact been found that the diversity theories and techniques in IR and recommender systems can be connected [77, 78], as we will discuss in Sect. 26.5.4. Given these connections, and the significant developments on diversity in IR, we find it relevant to include an overview of this work here, as we will do in Sects. 26.3 (metrics) and 26.4 (algorithms).

### 2.7.3 Novelty and Diversity Evaluation

The definitions discussed in the previous sections can only get a full, precise and practical meaning when one has given a specific definition of the metrics and methodologies by which novelty and diversity are to be measured and evaluated. We review next the approaches and metrics that have been developed to assess novelty and diversity, after which we will turn to the methods and algorithms proposed in the field to enhance them.

#### Notation

As is common in the literature, we will use the symbols  $i$  and  $j$  to denote items,  $u$  and  $v$  for users,  $I$  and  $U$  for the set of all items and users respectively. By  $I_u$  and  $U_i$  we shall denote, respectively, the set of all items  $u$  has interacted with, and the set of users who have interacted with  $i$ . In general we shall take the case where the interaction consists of rating assignment (i.e. at most one time per user-item pair), except where the distinction between single and multiple interaction makes a relevant difference (namely Sect. 26.5.2.1. We denote ratings assigned by users to items as  $r(u, i)$ , and use the notation  $r(u, i) = \emptyset$  to indicate missing ratings, as in [5]. We shall use  $R$  to denote a recommendation to some user, and  $R_u$  whenever we wish or need to explicitly indicate the target user  $u$  to whom  $R$  is delivered—in other words,  $R$  will be a shorthand for  $R_u$ . By default, the definition of a metric will be given on a single recommendation for a specific target user. For notational simplicity, we omit as understood that the metric should be averaged over all users. Certain global metrics (such as aggregate diversity, defined in Sect. 26.3.5) are the exception to this rule: they directly take in the recommendations to all users in their definition, and they therefore do not require averaging. In some cases where a metric is the average of a certain named function (e.g. IUF for inverse user frequency, SI for self-information) on the items it contains, we will compose the name of the metric by prepending an “M” for “mean” (e.g. MIUF, MSI) in order to distinguish it from the item-level function.

#### Average Intra-List Distance

Perhaps the most frequently considered diversity metric and the first to be proposed in the area is the so-called average intra-list distance—or just intra-list diversity, ILD (e.g. [70, 85, 90]). The intra-list diversity of a set of recommended items is defined as the average pairwise distance of the items in the set:

$$\text{ILD} = \frac{1}{|R|(|R| - 1)} \sum_{i \in R} \sum_{j \in R} d(i, j)$$

The computation of ILD requires defining a distance measure  $d(i, j)$ , which is thus a configurable element of the metric. Given the profuse work on the development of similarity functions in the recommender systems field, it is common, handy and sensible to define the distance as the complement of well-understood similarity measures, but nothing prevents the consideration of other particular options. The distance between items is generally a function of item features [90], though the distance in terms of interaction patterns by users has also been considered sometimes [79].

The ILD scheme in the context of recommendation was first suggested, as far as we are aware of, by Smyth and McClave [70], and has been used in numerous subsequent works (e.g. [75, 79, 85, 90]). Some authors have defined this dimension by its equivalent complement intra-list similarity ILS [90], which has the same relation to ILD as the distance function has to similarity, e.g.  $ILD = 1 - ILS$  if  $d = 1 - \text{sim}$ .

### **Global Long-Tail Novelty**

[26.3.3]

### **User-Specific Unexpectedness**

[26.3.4]

### **Inter-Recommendation Diversity Metrics**

[26.3.5]

### **Specific Methodologies**

As an alternative to the definition of special-purpose metrics, some authors have evaluated the novelty or diversity of recommendations by accuracy metrics on a diversity-oriented experimental design. For instance, Hurley and Zhang [39] evaluate the diversity of a system by its ability to produce accurate recommendations of difficult items, “difficult” meaning unusual or infrequent for a user’s typical observed habits. Specifically, a data splitting procedure is set up by which the test ratings are selected among a ratio of the top most different items rated by each user, “different” being measured as the average distance of the item to all other items in the user profile. The precision of recommendations in such a setting thus reflects the ability of the system to produce good recommendations made up of novel items. A similar idea is to select the test ratings among cold, non-popular long tail items. For instance, Zhou et al. [89] evaluate accuracy on the set of items with less than a given number of ratings. Shani and Gunawardana also discuss this idea in Chap. 8.

## **Diversity vs. Novelty vs. Serendipity**

[26.3.7]

### **2.7.4 Novelty and Diversity Enhancement Approaches**

Methods to enhance the novelty and diversity of recommendations are reviewed in this section. It is noteworthy that research in this area has accelerated over the last number of years. The work can be categorized into methods that re-rank an initial list to enhance the diversity/novelty of the top items; methods based on clustering; hybrid or fusion methods; and methods that consider diversity in the context of optimization of learning to rank objectives.

#### **Result Diversification/Re-ranking**

[26.4.1, onemli]

#### **Using Clustering for Diversification**

A method proposed in [86] clusters the items in an active user's profile, in order to group similar items together. Then, rather than recommend a set of items that are similar to the entire user profile, each cluster is treated separately and a set of items most similar to the items in each cluster is retrieved.

A different approach is presented in [14], where the candidate set is again clustered. The goal now is to identify and recommend a set of representative items, one for each cluster, so that the average distance of each item to its representative is minimized.

A nearest-neighbor algorithm is proposed in [48] that uses multi-dimensional clustering to cluster items in an attribute space and select clusters of items as candidates to recommend to the active user. This method is shown to improve aggregate diversity.

A graph-based recommendation approach is described in [69] where the recommendation problem is formulated as a cost flow problem over a graph whose nodes are the users and items of the recommendation. Weights in the graph are computed by a biclustering of the user-item matrix using non-negative matrix factorization. This method can be tuned to increase the diversity of the resulting set, or increase the probability of recommending long-tail items.

#### **Fusion-Based Methods**

Since the early days of recommender systems, researchers have been aware that no single recommendation algorithm will work best in all scenarios. Hybrid systems

have been studied to offset the strengths of one algorithm against the weaknesses of another (see [68] for example). It may be expected that the combined outputs of multiple recommendation algorithms that have different selection mechanisms, may also exhibit greater diversity than a single algorithm. For example, in [66, 79], recommendation is treated as a multi-objective optimization problem. The outputs of multiple recommendation algorithms that differ in their levels of accuracy, diversity and novelty are ensemble using evolutionary algorithms. As another example, in a music recommendation system called Auralist [88], a basic item-based recommender system is combined with two additional algorithms, in order to promote serendipity (see section below).

### **Learning to Rank with Diversity**

In the last few years, there is an increasing interest in learning to rank algorithms for recommender systems. These algorithms directly optimize an objective related to the ranking rather than forming the ranking in a post-processing step from a set of predicted ratings. To date, most such techniques do not take into account the dependencies between the items in the ranking. However, a small number of works have appeared in the literature that optimize a combined objective of ranking accuracy and set diversity. In [71], the concept of diversity is integrated into a matrix factorization model, in order to directly recommend item sets that are both relevant and diversified. A matrix factorization model is again used in [40] to optimize a ranking objective that is explicitly modified to account for the diversity of the ranking.

### **Other Approaches**

A nearest neighbor algorithm called usage-context based collaborative filtering (UCBCF) is presented in [58], which differs from standard item-based CF in the calculation of item-item similarities. Rather than the standard item representation as a vector of user ratings, an item profile is represented as a vector of the  $k$  other items with which the item significantly co-occurs in user profiles. UCBCF is shown to obtain greater aggregate diversity than standard kNN and Matrix Factorization algorithms. A system described in [8] maps items into a utility space and maps a user's preferences to a preferred utility vector. In order to make a diverse recommendation, the utility space is split into  $m$  layers in increasing distance from the preferred utility and non-dominated items are chosen from each layer so as to maximize one dimension of the utility vector.

The works discussed so far have considered diversity in terms of the dissimilarity of items in a single recommendation set, or, in the case of aggregate diversity, the coverage of items in a batch of recommendations. Another approach is to consider

diversity in the context of the behavior of the system over time. Temporal diversity (Eq.(26.7) in Sect.26.3.4) is investigated by Lathia et al. [46] in a number of standard CF algorithms, and methods for increasing diversity through re-ranking or hybrid fusion are discussed. In a related vein, Mourao et al. [56] explore the “oblivion problem”, that is, the possibility that in a dynamic system, items can be forgotten over time in such a way that they recover some degree of the original novelty value they had when they were discovered.

### User Studies

It is one thing to develop algorithms to diversify top k lists, but what impact do these algorithms have on user satisfaction? A number of user studies have explored the impact of diversification on users. Topic diversification is evaluated in [90] by carrying out a user survey to assess user satisfaction with a diversified recommendation. In the case of their item-based algorithm, they find that satisfaction peaks around a relevance/diversity determined by  $\lambda = 0.6$  in Eq. (26.9) suggesting that users like a certain degree of diversification in their lists.

While much of the work in diversifying top k lists does not consider the ordering of items in the recommended list, provided an overall relevance is attained, Ge et al. [32, 33] look at how this ordering affects the user’s perception of diversity. In a user study, they experiment with placing diverse items—ones with low similarity to the other items in the list—either in a block or dispersed throughout the list and found that blocking the items in the middle of the list reduces perceived diversity.

The work of Hu and Pu [37] addresses user-interface issues related to augmenting users’ perception of diversity. In a user study that tracks eye movements, they find that an organizational interface where items are grouped into categories is better than a list interface in supporting perception of diversity. In [19], 250 users are surveyed and presented with 5 recommendation approaches, with varying degrees of diversity. They find that users perceive diversity and that it improves their satisfaction but that diverse recommendations may require additional explanations to users who cannot link them back to their preferences.

### Diversification Approaches in Information Retrieval

[26.4.8, eger kullanirsan]

#### 2.7.5 Unified View

As the overview through this chapter shows, a wide variety of metrics and perspectives have been developed around the same concepts under different variants and angles. It is

natural to wonder whether it is possible to relate them together under a common ground or theory, establishing equivalences, and identifying fundamental differences. We summarize next a formal foundation for defining, explaining, relating and generalizing many different state of the art metrics, and defining new ones. We also examine the connections between diversity as researched and developed in the Information Retrieval field, and the corresponding work in recommender systems. [26.5, information retrieval'daki diverisity yi recommendation diversity'y bagliyor, kullanirsan kullan]

### **2.7.6 Empirical Metric Comparison**

[26.6, Kendileri deney yapmislar ve sonuclari metriclere gore karsilastirmislar, kullanilabilir en azindan tablo]

### **2.7.7 Conclusion**

The consensus is clear in the community on the importance of novelty and diversity as fundamental qualities of recommendations, and it seems difficult to make progress in the field without considering these dimensions. Considerable progress has been achieved in the area in defining novelty and diversity from several points of view, devising methodologies and metrics to evaluate them, and developing different methods to enhance them. This chapter aims to provide a wide overview on the work so far, as well as a unifying perspective linking them together as developments from a few basic common root principles.

It is our perception that work in this area is far from being finished. There is still room for further understanding the role of novelty and diversity, as well as theoretical, methodological and algorithmic developments around them. For instance, modeling feature-based novelty in probabilistic terms in order to unify discovery and familiarity models would be an interesting line for future work.

Aspects such as the time dimension, along which items may recover part of their novelty value [43, 56], or the variability among users regarding their degree of novelty-seeking trend, are examples of issues that require further research. Last but not least, user studies would bring considerable light as to whether the described metrics match the actual user perception, as well as the precise extent and conditions in which users appreciate novelty and diversity versus accuracy and other potential dimensions of recommendation effectiveness.



## 3 Implementation

This chapter explains different solutions for the problem at hand. These solutions include applications using different datasets and different methods.

As we mentioned in the Introduction part [See section 1.3], the implementation of this thesis focuses on recommending the best talents to projects. While serving this aim, we use different datasets and different methods.

The methods used can be categorized as individual and group recommenders. Individual recommenders have the aim of suggesting only one person to a project. As oppose to that, group recommenders combine multiple subprojects as a super project and recommend various talents to this super project. Another differentiation of these recommenders is the type of learning algorithms. Both group recommenders and the individual recommenders are implemented via supervised and unsupervised learning approaches. The supervised learning approach trains neural networks with the help of ground-truth labels. On the other hand, the unsupervised approach employs training just with the feature vectors [SA13]. Detailed information about these methods can be found in the upcoming sections.

### 3.1 Datasets

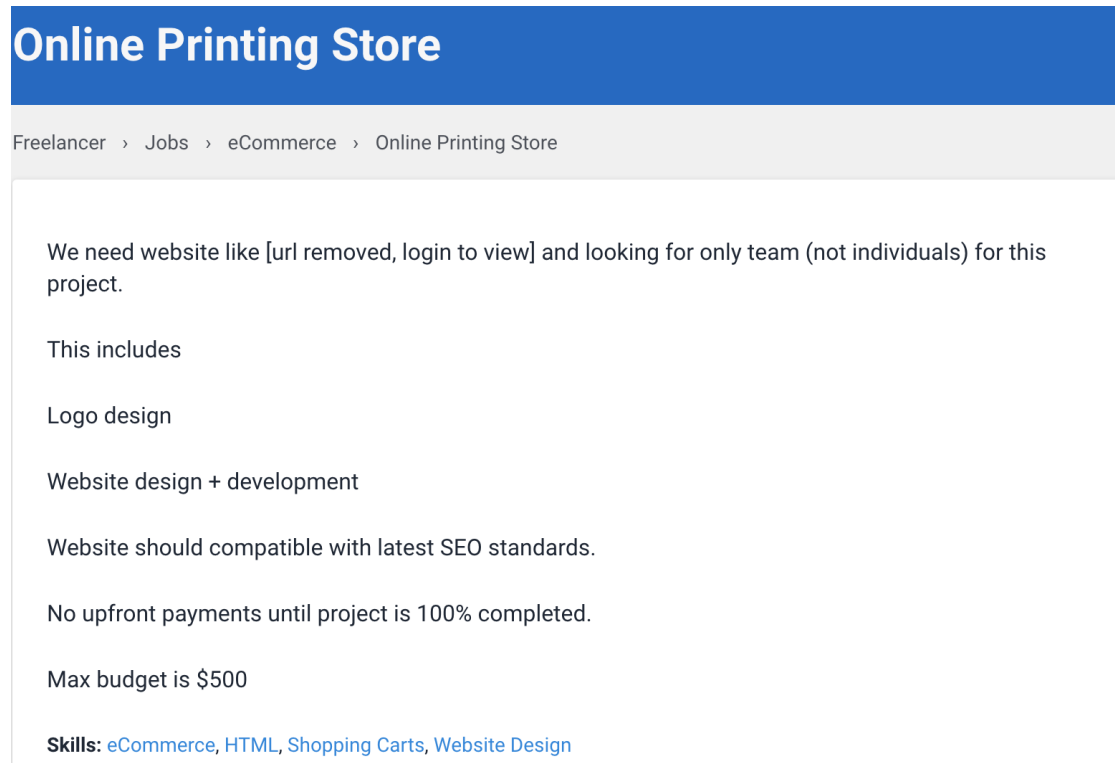
The author of the thesis received two separate but similar datasets before starting the thesis. Both of the datasets contain information about projects and people. The company dataset [See subsection 3.1.2] is the internal database of the company Motius and includes skill vectors of 795 people and 375 roles. These roles come together and form projects, which is not the case in freelancer dataset. In freelancer dataset, we have projects as opposed to positions in the company dataset. However, we treat the projects in the freelancer data and the roles in the company data the same. The reason for that is that we want to combine both data and we also want to compare them.

A huge difference is a fact that the Freelancer dataset is much bigger and detailed compared to the company dataset. The freelancer dataset contains 30606 roles that are comparable to the positions in the company dataset. It has 32922 unique talents and 463536 bids by talents to the projects that represent the project-talent pairs.

Another significant contrast between the two datasets is the distribution of their positive and negative labels. Freelancer data carries approximately 14-15 applicants per

projects, and only one of the applicants get selected as the person to implement the project. Differently, the company dataset includes multiple talents that advance to the next steps of the interviews. Therefore, we marked all of these talents that got invited with a positive label, and we marked the rest with negative tags. We will give detailed information about both datasets in the upcoming subsections.

#### 3.1.1 Freelancer Dataset



The screenshot shows a project listing on the Freelancer website. At the top is a blue header with the title 'Online Printing Store'. Below this is a breadcrumb trail: 'Freelancer > Jobs > eCommerce > Online Printing Store'. The main content area contains the following text:

We need website like [url removed, login to view] and looking for only team (not individuals) for this project.

This includes

- Logo design
- Website design + development

Website should compatible with latest SEO standards.

No upfront payments until project is 100% completed.

Max budget is \$500

**Skills:** [eCommerce](#), [HTML](#), [Shopping Carts](#), [Website Design](#)

Figure 3.1: An example project from the Freelancer Website

As you can see in the figure 3.1, a typical freelancer project posting consists of a title, the description and the relevant skills. For simplicity, the thesis at hand only concentrates on the skills and doesn't take the project description into account. This would be topic of another paper/thesis, as it would require natural language processing and other techniques [BKL09].

The figure 3.2 shows the bidders of the same project as above. The first one of the bidders has won the bidding race, which is decided by the creator of the project. The bidders include information such as a motivation text, the demanded monetary amount,

### 3 Implementation

---

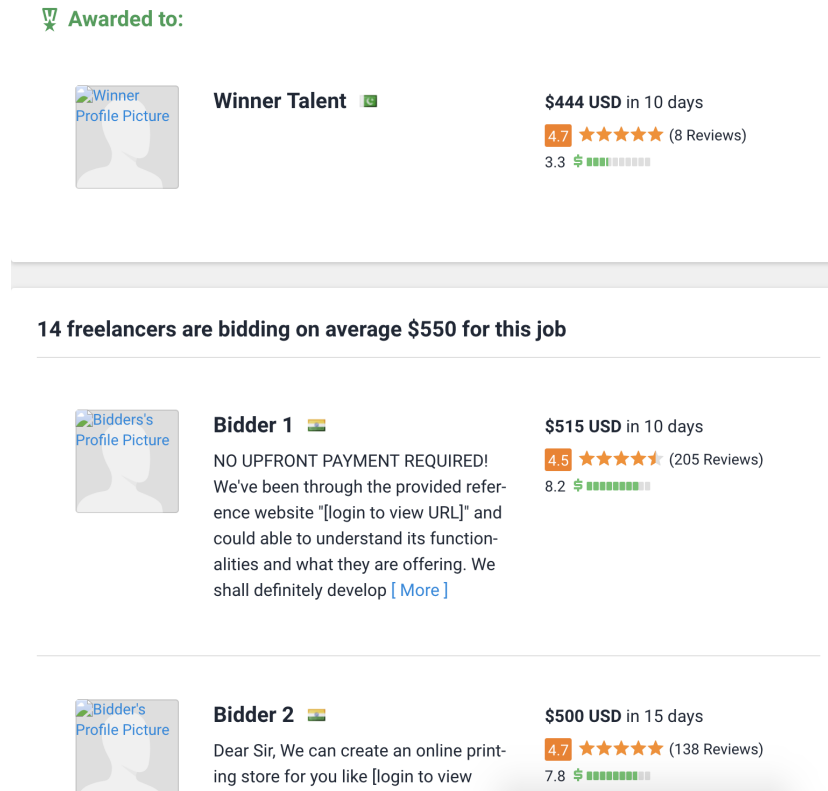


Figure 3.2: The winner and other bidders to the same project

their star rating until the time of bidding, amount of reviews they received and their total earnings until that date. In this thesis, we only consider the money they demand, their star rating and the number of reviews. For the sake of simplicity, we don't use the motivation text.

Each bidder lists their skills on their profile page, and the employers may check their profiles before hiring talents. The figure depicts the top skills of an arbitrary talent, which are listed in descending order. The number near each skill shows how many related projects the talent completed. That's why the amounts can range from one to more than hundreds.

The dataset encloses 941 unique skills, which are both technical and non-technical. However, the author of the thesis chose to limit these skills to 780, since some of them weren't used in notable amounts and the data can be expressed without using those skills. [TODO see dimensionality reduction add to research]

This dataset was scraped from freelancer.com by Philip Offtermatt, who also took part in this very project.

My Top Skills	
PHP	17
Website Design	13
HTML	13
Graphic Design	11
Javascript	3
Mobile App Development	2
WordPress	2
CSS	2
Script Install	1
Web Scraping	1

Figure 3.3: The list of tops skills by a talent on Freelancer web page

#### 3.1.2 Company Dataset

The company dataset at hand is acquired from the sponsor of this thesis, which is Motius GmbH. This dataset exported from their internal database and contains information of 795 people and 375 roles. Each role includes the required skills for them, and each person has their skills listed. One difference to the freelancer dataset would be that the talents also include skills that are associated with their original skill set. In theory, this is called association rules, and the skills that are mostly used together are considered to have a correlation score of 1. Because of these correlations, each talent has many skills listed, some of them highly correlated and others are not correlated at all.

The amount of unique skills in the company data equals to 1768. Nonetheless, more than 85% of the skills are used rarely, so the author reduced the unique set of skills to 202. Both of the datasets combined, 923 unique skills were given at least five times. A problem we have with the datasets is the naming. Since both Freelancer and Motius have used different names for skills, there are exists only a set of 59 common skills. Therefore, training both datasets together doesn't improve model like it's expected[TODO: evaluation neural network 10 000 data training, etc.].

When all of freelancer and company dataset are put together in their raw form, the matrix that contains all talent and project data reaches the size of 8 GB. Such a significant memory usage created a big problem for the author of the thesis. Since we only had available physical memory of 16 GB, working with an 8 GB matrix wasn't possible. When an operation like normalization is being done, the library *Pandas* applies many copying operations, which doubles the memory usage and crashes. That's why the author employed embeddings as a dimensionality reduction mechanism[See 3.3.2].

## 3.2 Unsupervised Individual Recommender

### 3.2.1 Recommendation by Similarity

As it was mentioned before, the unsupervised learning techniques focus on learning without the use of labels. Therefore, in the context of this thesis, we find similarities between projects and talents by using their feature vectors.

$$\cos(x, y) = \frac{(x \bullet y)}{\|x\| \|y\|} \quad (3.1)$$

The similarity measure we use for this part of the thesis is the cosine similarity [Ama+11]. The formula for the cosine similarity is shown above. The inputs  $x$  and  $y$  in the equation can correspond to a project-talent or a talent-talent pair. The types of input data are document vectors of an  $n$ -dimensional space, and the formula calculates the similarity as the cosine of the angle between two vectors. The recipe first calculates the dot product of the vectors and then divides it by the multiplication of the normal vectors.

To get the most important talents for the project, we calculate the cosine similarity between a selected project and every other talent. Then the algorithm sorts the people by similarity and returns *top n* bidders.

### 3.2.2 Recommendation by Popularity

Another unsupervised recommendation mechanism that is used as a baseline is the popularity recommender. The popularity recommender is hard to beat algorithm, that increases user satisfaction. [TODO add pages 394 and 395 to research section(from book)]. The logic comes from the fact that the *items* that are in demand approved by many *people*. That's why it's likely that selecting these items will increase user satisfaction [AB15].

For this specific project, the popular items that are in demand are the people that finished the maximum amount of projects at Freelancer or Motius. Although the results

won't be personal, recommending the same successful talents is a helpful strategy to acquire proven talents. The proof of this approach is shown in subsection 4.1.3.

### 3.2.3 Hybrid Recommendation

As a last submethod of unsupervised individual recommenders, we can name the hybrid recommender. Hybrid methods are also called as *ensemble learning* methods. This technique combines the results from multiple processes and outputs a new result [BCJ15]. For our use case, the author implemented different versions that combine the similarity recommender and the popularity recommender. We can merge both of the recommenders by adding or multiplying the results. It is also possible to give different weights to these sources.

## 3.3 Supervised Individual Recommender

Supervised learning means creating a model that learns with the help of labels. In our project, the author conceptualized labels as 0 or 1. 1 is for the case of the person got accepted for the project at freelancer.com or Motius. 0 is for the case that the person got rejected. The models try to predict if the talent should be hired for the project or not(1 or 0).

For this task, we use two different versions; one version that takes all the skills as-is, the other one creates embeddings[TODO: embeddings in research chapter]. Both of the methods employ neural networks[TODO: neural networks in the research chapter].

### 3.3.1 Using Sparse Input

	.net	2d animation	360- degree video	3d animation	3d design	3d model maker	3d modelling	3d printing	3d rendering	3ds max	...	xero	xml	xmpp	xslt	yii	youtube	zbrush	zen cart	zend
user_url																				
Talent 1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
Talent 2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
Talent 3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
Talent 4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
Talent 5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
Talent 6	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
Talent 7	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0
Talent 8	0	0	0	1	0	0	1	0	1	0	...	0	0	0	0	0	0	0	0	0
Talent 9	0	0	0	1	0	0	1	0	1	1	...	0	0	0	0	0	0	0	0	0
Talent 10	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

10 rows × 780 columns

Figure 3.4: The talent skill matrix from freelancer.com

The first option that comes to mind is using the data as it is and training the model with them. The format of the talent data is shown in figure 3.4 and the projects skills matrix also have the same form with project names as keys.

$$z = (x - u)/s \quad (3.2)$$

According to experts [SS97], it is crucial to normalize input data before training neural networks. It has two significant benefits; it reduces the estimation errors, and it cuts down the training time. That's why we normalize all of the inputs using *StandardScaler* module of *scikit-learn*. It uses equation 3.2 to scale the data. In the equation,  $u$  is the mean and  $s$  is the standard deviation.

As it was mentioned before [See section 3.1], the freelancer.com dataset accommodates some extra information like experience level, star rating, number of reviews and hourly rate. These extra information of 10 example talents are shown in the figure 3.5. Since this information doesn't exist in Motius dataset, this subsection focuses only on the implementation with freelancer datafset. The extra information that is mentioned is also scaled and input into the neural network.

After normalizing data, we prepare the matrix that is fed into the neural network row by row. For each bid in the freelancer data, we create a vector of length 1565. Seven hundred eighty of these values correspond to talent skills, the next 780 correspond to project skills, 4 of them are the extra information that is mentioned above and the last of them is for the outcome. The outcome is 1 for the case that the person received the project and 0 for the example that the person didn't receive the project.

	experience_level	star_rating	number_of_reviews	hourly_rate
bidder_url				
Talent 1	5	4.8	385	12
Talent 2	17	4.9	162	25
Talent 3	17	5.0	5	15
Talent 4	6	4.9	116	30
Talent 5	6	0.0	0	2
Talent 6	6	0.0	0	3
Talent 7	6	5.0	24	40
Talent 8	6	5.0	2	5
Talent 9	6	5.0	16	20
Talent 10	3	5.0	67	20

Figure 3.5: The talent extra information matrix from freelancer.com

As one would expect, the model tries to guess if the person should be employed or not. Out of the 321225 data points in total, we use 60% for training, 20% for test and the rest for validation. We split the data into those sets randomly using *train\_test\_split* function of *scikit-learn*. An important parameter not to miss is *stratify*; since our dataset has 6% positive and 94% negative samples, we need to make sure that this ratio also remains in the sets. Not using this feature could result in the model always predicting the same negative results. Doing that means that the model would learn the same result, no matter what [SP15].

After splitting the data, we can start with training. There also exists some important features that we need to use; these optional features all have different objectives, but they all serve to improve the results. These features are all supported by the packages *Keras* and *TensorFlow*, which are open-source neural network libraries. Keras is an abstraction layer for TensorFlow that lets the users train neural networks with a minimal number of lines [Cho18]. While fitting the model with training data, Keras gives the option to add callbacks. The callbacks that we adopt are *EarlyStopping*, *ModelCheckpoint*, *ReduceLROnPlateau* and *TensorBoard*. As the name suggests, early stopping [TODO: add neural networks to research part; also validation loss, overfitting, graph vs] serves to prevent overfitting. In our case, it compares the validation loss of current batch with the previous one. If the validation loss doesn't drop for ten times, the training stops. The next callback model checkpoint is used complementary to early stopping. Model checkpoint saves the model weights of the batch with the minimum validation loss. After the training is complete, we load those model weights that achieved the best accuracy. ReduceLROnPlateau reduces learning rate when the validation loss has stopped improving. Lastly, TensorBoard is a visualization tool for TensorFlow. It produces model visions and graphs that show the evolution of the accuracy, loss and learning rate.

When we are training the model, we should also set the training weights for both labels manually. The dataset encompasses 6% positive and 94% negative samples, so we need to penalize the errors according to this rate. After training, the class weights are ignored and not used in testing/predicting.

The figure 3.6 depicts the model that is used to predict if a talent should be employed or not. The direction of the graph starts at the bottom of the image and goes up. Like it was mentioned before, the model expects three feature vectors. These vectors are defined as the skill vectors of the project, the skill vector of the talent and extra information of the talent(e.g., hourly rate, total experience).

For each of the inputs, a dense layer exists with the number of neurons equal to the number of features. Making this decision means that project and talent layers contain 780 neurons and profile information layer contains four neurons. For these layers, we use *relu* activation, *l1* regularization with the value *0.0001* and we initialize the



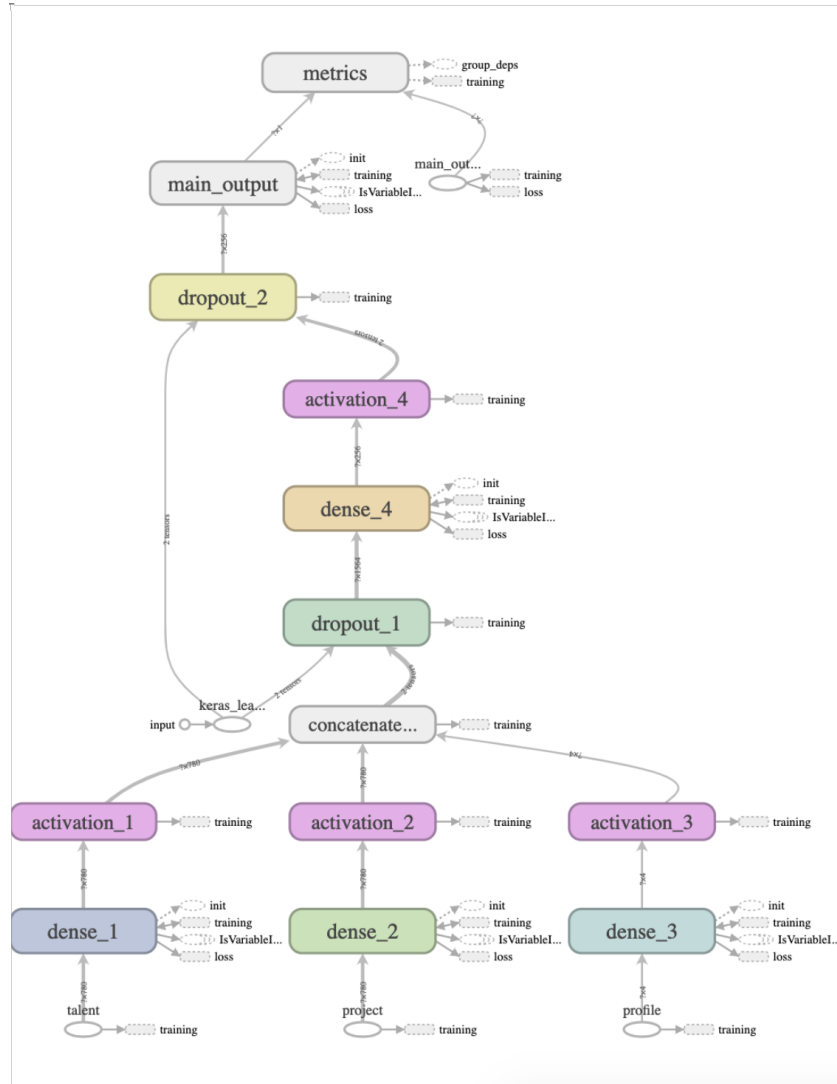


Figure 3.6: The graph that explains the sparse input model

weights with *he normal*. Detailed knowledge about activation, regularization, and weight initialization can be found in the chapter 2[TODO: add activation, regularization, and weight init to research chapter].

After the activation functions, all layers get concatenated horizontally. Concatenation layer is followed by a dropout layer with half of the neurons are disabled randomly[TODO: add dropout to neural networks section of chapter 2]. Next one in the model is a dense layer with 256 nodes, which possesses the same activation

function, regularization and weight initialization methods as the previous dense layers. The model accommodates the last dropout layer and ends with the main output. The output is only a one node layer and involves a *sigmoid* activation function that squeezes the output value to be between 0 and 1[TODO: research -> nn -> activation functions -> sigmoid]. The weight initializer of the last layer is *glorot uniform*. [TODO: research -> nn -> weight initializers -> glorot uniform].

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 \quad (3.3)$$

Each neural network has the aim of minimizing its cost function [GBC16]. The cost function that we chose is *mean squared error*[See Equation 3.3]. This example of the cost function is used mostly for regression tasks and calculates the mean squared difference of the actual value and the predicted output value. The metric we use is accuracy, and more information about it can be found in chapter 4.

#### 3.3.2 Using Embeddings

High-dimensional spaces and distributions prove to be unexpected and completely differ from low-dimensional spaces. The empty space phenomenon and other ones are examples of the *curse of dimensionality*. With the help of embeddings layers, we can represent high-dimensional data in low-dimensions [LV07].

Although deep neural networks can avoid the curse of dimensionality [Pog+17], we still need to use embeddings layers for spatial reasons. In the previous sections, we mentioned that there are 780 unique skills for freelancer data and 923 skills if we also add Motius data. Having all of this data means that the data has 923 dimensions and we know that the information is sparse, most of the data matrices consist of zeroes so that we can reduce the dimensionality.

[TODO explain embeddings in research. also explain embeddings math. instead]

#### Preprocessing

For both Freelancer and Motius data, we know the skill levels of projects and talents for various skills. Instead of having some positive and hundreds of zero skill values for each project/talent, we can set a skill threshold. This threshold implies skills above or equals to the limit are positive, and the rest are zero. The idea is converting the talent-skill and project-skill matrices so that, each talent/skill holds a list of skills they know/require. However, another constraint that needs to be addressed is the maximum length of the padded skill matrix, because neural networks require a fixed input shape. Therefore, the talent/project maximum amount of positive skills is determined. For

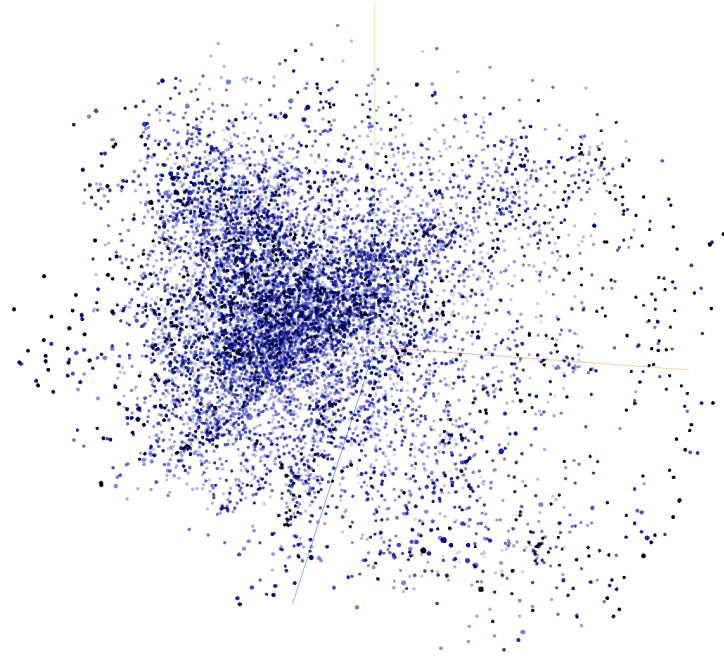


Figure 3.7: 3D version of the embedding space that is created for projects of this thesis

Freelancer data, this is 18, which means all skills vectors are padded with zeroes to have the length of 18.

In the case of Motius data, the topic is more complicated. The subsection 3.1.2 explained how the correlation mechanism of the company data works. To describe it briefly, Motius stores user skills and other skills that are correlated for each user. Including this data affects that there exist many skills for each user, but most of these skill levels are low. Here the highest skill level would be 2, and the smallest would be 0.

The effect of different threshold values on Motius data is shown on Figure 3.8. Without any threshold, there is a couple of Motius projects with the maximum skill length of 21. Projects don't specify skill levels, so these are taken as they are. That's why we also wanted to have a similar maximum length for Motius talents. When there is no threshold, there are 780 Motius talents with at least one skill value, but the maximum skill vector length is 132. We wouldn't want to implement this version because the maximum range of 132 will create millions of zeroes in the dataset, which we tried to avoid in the first place. Setting the threshold to a high value (like 1 or more) is also not optimal since it limits the maximum skill vector length to 9 and number of Motius talents to 269. Having such a high value would decrease the amount of information we have significantly because the Freelancer data also has a maximum

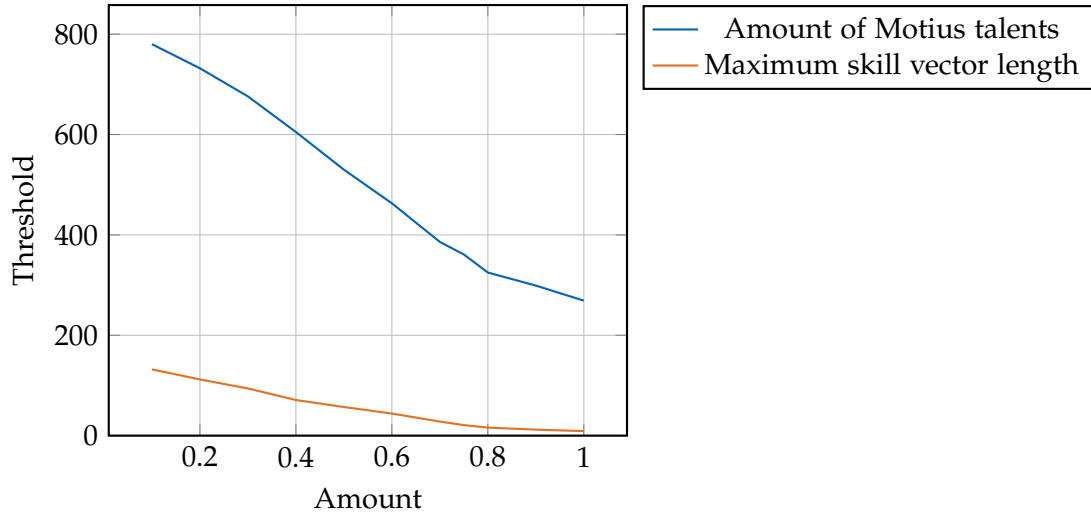


Figure 3.8: Effect of threshold selection on talents and maximum skill vector length

length of 18. Therefore, the optimum threshold value we reached is  $0.75$ . Doing this limits the number of Motius talents to 361 and limits the maximum skill vector length to 21, just like the project with the most skills.

Figure 3.9 depicts the training data with full skill matrix. The columns with the numbers in range 0 to 20 are the indices of the talent skills and the columns with the names 21 to 41 are project skills indices. The version of the image is the one with the Motius and Freelancer data combined. In the variant with only Freelancer data, we have skill vector lengths of 18 and the extra information of talents included.

	project_url	bidder_url	outcome	0	1	2	3	4	5	6	...	32	33	34	35	36	37	38	39	40	41
0	a.i. & software engineer	Talent 1	0.0	69.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	a.i. & software engineer	Talent 2	0.0	577.0	778.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	a.i. & software engineer	Talent 3	0.0	350.0	396.0	487.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	a.i. & software engineer	Talent 4	0.0	69.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	a.i. & software engineer	Talent 5	0.0	222.0	460.0	776.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 3.9: Training data that contains padded embedding skill vectors

### Simpler Architecture for Company Dataset

The Freelancer dataset has a huge advantage over Motius data, which is the extra information that we know about the applicants. When we use both datasets together to train a model, we can limit the inputs to talent and project skills.

Figure 7.1 illustrates the simplified version of the *Python* code that constructs the model to predict the hiring result. The *features* parameter of the model building function corresponds to the length of the padded skill vector and the next parameter *dimensions* represents the total amount of unique skills. Embedding layers in Keras expect the arguments *input dimension*, *output dimension*, *input length* and the optional flag *mask zero*. As embeddings only accept positive integers, input dimension should be the size of the vocabulary, which is the number of total skills in the recommender system. The value of the output dimension can be decided by the developer and explains the size of the desired output dimension. Although no scientific document states an ideal output dimension, the trial-and-error method showed that the best result is achieved with the fourth root of the number of dimensions. The additions of one in multiple places in the code are due to using the mask zero operation. Zeros in rows get filtered out, which increases the performance and speeds up the training process. The cost function that we use is *binary\_crossentropy* because we want to optimize the process of hiring or not hiring talent to the project. [TODO: research -> cost functions -> binary\_crossentropy]. Lastly, *sigmoid* activation function squeezes the output value to be between 0 and 1[TODO: research -> nn -> activation functions -> sigmoid].

## 3.4 Unsupervised Group Recommender

In this section, we explain the process of recommending multiple talents to supergroups. The methods that are used for this part are derivations of the ones that are used in individual recommenders. Therefore, the basic concepts that are employed before also apply here.

To perform group recommendations, project-role or project-project information are needed. The website [freelancer.com](https://www.freelancer.com) shows other projects from the same supervisor, which can be combined. Then these projects will form a super project and projects can be treated as roles of a more significant project. For Motius data, we already possess this information as project-role data. Roles of Motius correspond to the projects in the Freelancer data. In terms of simplicity and shortness, we only take Freelancer dataset with groups of size five into account.

### 3.4.1 Baseline

The basic approach to unsupervised group recommendations would be calculating the cosine similarity between each project and talents. Then pick the best talents and listing them. However, the results won't be diverse, and we can pick talents that have similar skills to each other. We want to avoid that [TODO add different evaluations to both

theory and their results to the evaluation part!] and have diverse recommendations for each project.

### 3.4.2 Diverse

Because of the reasons above, we want to create the recommendation list in a diverse way from the beginning. The topic of diversity is already explained before[TODO: research -> diversity enhancement] and the pseudocode to enhance diversity is shown below.

```

 $R \leftarrow \emptyset$ 
while  $|R| < k$  :
     $i^* \leftarrow \arg \max_{i \in C-R} g(R \cup \{i\}, \lambda)$ 
     $R \leftarrow R \cup \{i^*\}$ 
end while
return  $R$ 

```

(3.4)

In the algorithm above, we first create an empty recommendation list  $R$  and set a recommendation length  $k$ . In our example, we only consider the groups with project amount of 5, so  $k$  is five as well. After that, we find the optimal candidate that hasn't been selected yet, is relevant to the project at hand and is also diverse to the other selected candidates. Finding the optimal candidate can be tuned with the help of Equation 3.5. The  $\lambda$  parameter in the equation can be optimized to value the relevancy or diversity more;  $\lambda$  of 1 means to only consider variety, 0 factors to consider relevancy and 0.5 gives the balanced result. When we receive the optimal candidate from the equation, we add them to the recommendation list and iterate until we have enough talents for the whole group.

$$g(R, \lambda) = (1 - \lambda) \frac{1}{|R|} \sum_{i \in R} f_{rel}(i) + \lambda div(R) \quad (3.5)$$

When we compare the diversity of the baseline approach to the diverse group recommendation, it is obvious that the diversity of talents recommended has increased. The evaluation algorithm for diversity and other relevant measures can be found in chapter 4. [TODO group-rec evaluation]

## 3.5 Supervised Group Recommender

The previous section was about performing group recommendations with unsupervised learning. This section will do the same job using a supervised learning model that we used in section 3.3.

Equation 3.5 includes a  $f_{rel}$ , which is a relevancy score and a diversity rate that can be computed via cosine similarity, neural networks or other methods. In contrast to the unsupervised method, we calculate the relevancy score using the neural network that we used in section 3.3.

What we do in the individual supervised learning part is, training all parameters jointly, which is called end-to-end learning. This ideal was also the first aim for supervised group recommender approach. However, the data for such knowledge doesn't exist. To apply it, we would need data of hiring decisions for the groups not just projects. Since we don't have such information, we would have to generate it with a separate algorithm. In the end, it wouldn't bring much, because the model would learn the data generation algorithm and wouldn't have an effect on the real-life hiring prediction.

Due to the reason above, step by step learning process practiced. The first step of the process is training the model to optimize the individual hiring of talents. Then, we predict the relevancy score for each project-talent pair. For diversity score, we use the cosine similarity between the talents. After set those functions, the algorithm 3.4 is applied.

In the end, this method increases diversity according to the evaluation methods that are listed in chapter 4.

## 3.6 Group Recommendation using clustering

Clustering is the process of dividing data into different groups. To perform different multi-project recommendations, we can pick talents from clusters. Therefore, we can be sure that they are dissimilar.

Since k-means clustering can suffer from the curse of dimensionality [SEK04] [TODO research: the curse of dim, clustering ve k-means, PCA ekle], To prevent it, it is logical to reduce the dimensionality first. The choice of the author to reduce dimensionality is *Principal Component Analysis*.

The central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the data set. This idea is achieved by transforming to a new set of variables, the principal components (PCs), which are uncorrelated, and which are ordered so that the first few retain most of the variation present in all of the original variables [Jol11].

To determine the number of PCs, we can check the explained variance ratio for the different amount of PCs. Figure 3.10 shows how much variance information we lose if we set the number of components to a specific value. Here, it makes sense to note that

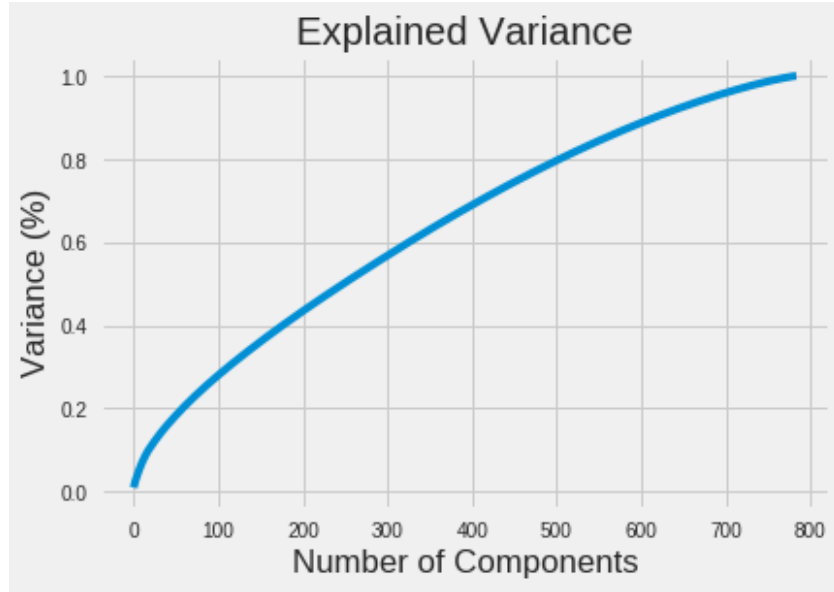


Figure 3.10: Explained variance ratio for difference number of PCs is shown.

a higher number will affect the clustering model negatively and a lower number won't be able to capture everything in the dataset. The author of the thesis experimented with different values.

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j) \quad (3.6)$$

$$b(i) = \min_{i \neq j} \frac{1}{|C_j|} \sum_{j \in C_j} d(i, j) \quad (3.7)$$

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \quad (3.8)$$

K-means clustering requires a  $k$  value that determines the number of clusters the model is going to create. The ideal number of clusters can be verified by calculating the silhouette scores for a different number of clusters. The figure 3.11 shows silhouette scores for different cluster amounts. Silhouette score calculates how similar a data point to its cluster compared to other clusters [Rou87]. For this task, we use the equation in the equations 3.6, 3.7, 3.8. In the equations, different distance metrics can be employed. The choice of the author is euclidian distance[TODO: distance metrics research]. The equation 3.6 calculates mean intra-cluster distance for each sample and the next 3.7 computes mean nearest-cluster distance for each sample, which means the distance



between a sample and the nearest cluster that the sample is not a part of. The last function 3.8 converts the results of the first two equations into silhouette coefficients. The mean of all silhouette coefficients from every sample gives the silhouette score for that  $k$  value. Higher silhouette scores suggest that the samples well matched to its cluster and poorly matched to neighboring clusters. In the example of 3.11, it makes sense to select a value like 30. To visualize results, we project the centers of the clusters on a 2D space[See 3.12]. The X-axis of the graph is the maximum value in each cluster center coordinate, and the Y-axis of the graph is the maximum value in each cluster center coordinate.

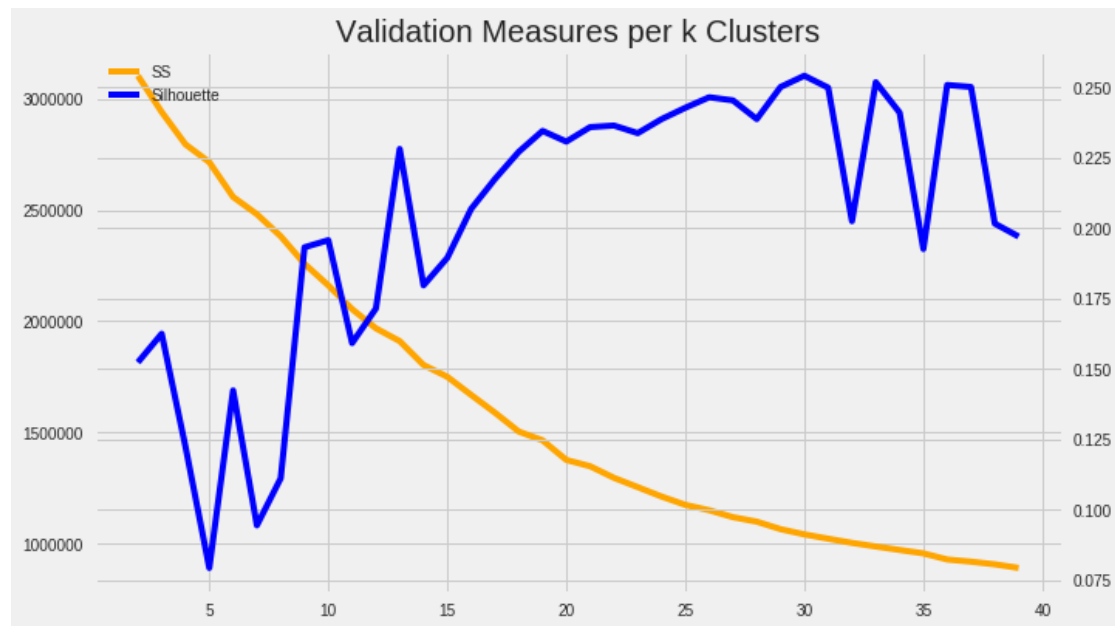


Figure 3.11: Silhouette scores of many different  $k$  values of k-means

Next, we benefit from another function of the PCA; *inverse\_transform*. Inverse transform takes the cluster centers as an input and converts them to full talent values. This conversion promises that we treat each cluster center like talent and transform their values to skill values and extra information. In end, we possess an average skill vector for every cluster [See figure 3.13]. The figure contains some part of the skill vectors of the first three average talents. For example the cluster(segment) 0 in the figure has exceptional *Adobe Illustrator* skills. Segment 1, on the other hand, is an all-rounder. Lastly, segment 3 is a *c#* developer. It must be noted these values are calculated after standard scaling[TODO research -> standard scaling].

After we exercise clustering, we can start with the group recommendation process.

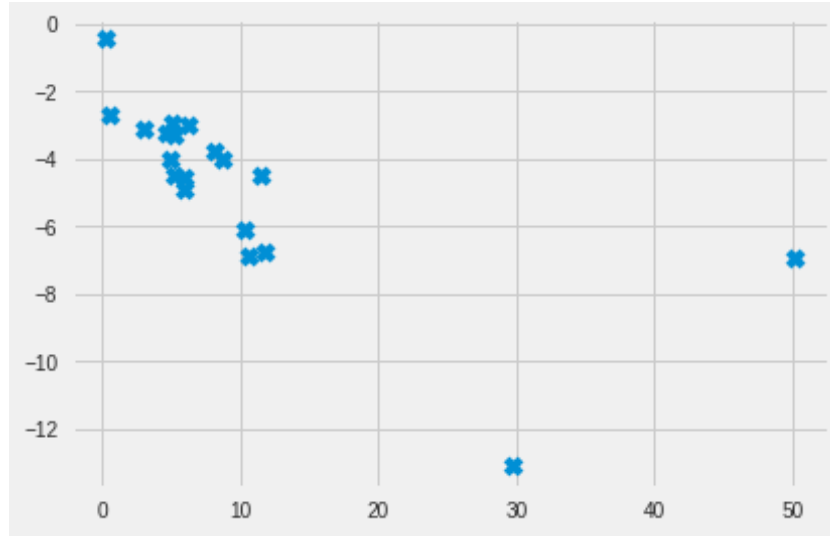


Figure 3.12: Centers of clusters that are projected on a 2D space

The recommendation can be operated both supervised[See 3.5] or unsupervised[See figure 3.4]. The same principles apply, and we calculate the relevancy score with cosine similarity or neural networks. In contrast to the other methods, the algorithm computes the relevancy score of the project and average cluster skills [See figure 3.13]. This way, we determine the ideal cluster for the project. When a project got recommended a talent from a specific cluster, that cluster is excluded from the next projects in the group. Therefore, diversity in a group is guaranteed. After the selection of the optimal cluster, the best candidate in that cluster is chosen via neural networks or cosine similarity.

	.NET	2D Animation	360-degree video	3D Animation	3D Design	3D Model Maker	3D Modelling	3D Printing	3D Rendering	3D Max	...	iPhone	jQuery / Prototype	node.js	phpMyAdmin
Segment 0	-0.212465	-0.004102	-0.019304	-0.095342	0.063749	-0.031560	-0.123809	-0.053663	-0.089939	-0.127223	...	-0.225476	-0.279852	-0.131371	-0.027035
Segment 1	-0.081847	-0.055546	-0.015600	-0.196805	-0.166759	-0.046560	-0.244780	-0.051745	-0.240299	-0.162773	...	-0.077620	0.500239	0.035455	0.019867
Segment 2	2.467993	-0.061241	-0.038406	-0.222762	-0.197445	-0.049275	-0.256107	-0.066796	-0.251231	-0.177760	...	-0.119369	0.142341	-0.028433	0.005529

Figure 3.13: Examples of some centers of clusters that are projected on a 2D space

## 3.7 Dashboard to show data and enter Feedback

### 3.7.1 General Dashboard

Another big part of the thesis is the dashboard that was built for various purposes; these purposes are showing individual unsupervised, supervised and hybrid recommendations for Motius and Freelancer datasets, showing group recommendations using unsupervised, supervised and hybrid methods and allowing to enter feedback, that has direct and indirect effects on the results.

The dashboard adopts the front-end that is programmed with *Vue.js* and a back-end that employs *Flask*[TODO: research -> frontend, backend, docker]. *Vue.js* is a front-end development framework that can be programmed with JavaScript. *Flask* is a back-end development framework that can be called with Python. The reason to use *Vue.js* is because of subjective reasons; it a reactive, modern framework that is easy to develop [You18]. *Flask* is, on the other hand, is chosen, because it is a popular light-weight Python framework [Gri18]. Since the rest of the machine learning training/prediction was done on Python, the author seized the opportunity to reuse/adapt the same codebase.

Docker is a container virtualization technology, which like a very lightweight virtual machine. Adding Docker to our software stack gives the advantage of portability. This is important for various reasons; first of all the operating system choice of the author is *MacOS* but most of the servers run different flavors of *Linux*. All of the different operating systems have different installation methods, different pre-installed libraries, and different dependencies. Docker solves this problem by standardizing the building and running operations of virtual machines. This way, the author was able to run everything on own computer and can be sure that it will also run perfectly on the servers of Motius if they choose to implement the solution on their internal system. [And15].

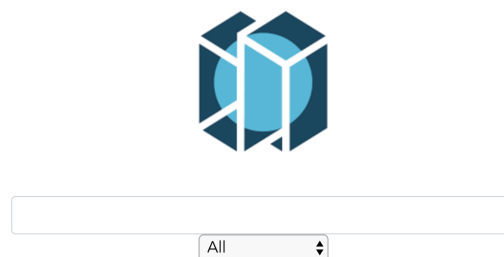


Figure 3.14: Main screen of the dashboard

The main screen of the dashboard is shown in figure 3.14 and it contains a search box and a dropdown. When users type anything on the search box, the front-end sends a *get request*[TODO: research -> get, post requests]. As the back-end receives the request, it checks the database for the text that is given by the user. Then, the back-end returns relevant projects as a list.

The dropdown on that screen has three options; *All*, *Motius* and *Group*. The *all* mode searches the input text in all projects database that includes Freelancer and Motius projects. As the name suggests, *Motius* mode only returns the company projects, and the *group* mode returns group numbers, which are a combination of various projects[See figure 3.15].

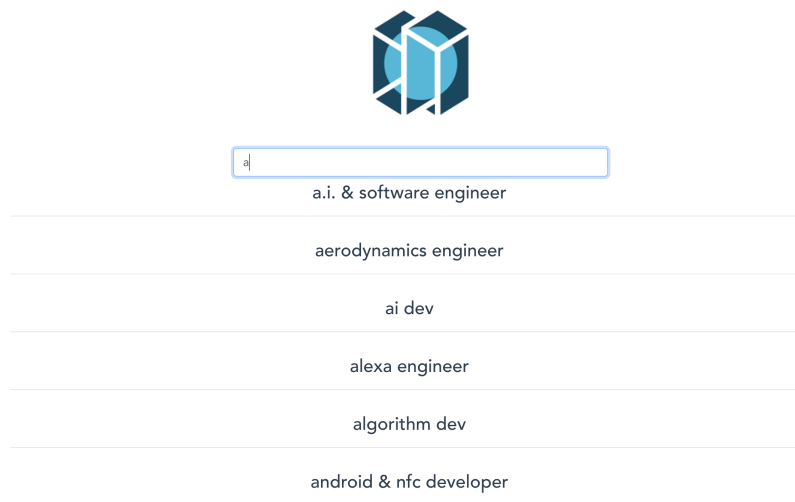


Figure 3.15: A snippet from the list of all projects that start with the letter *a*

#### 3.7.2 Individual Recommendations

When users click on any of the projects that are listed in figure 3.15, they receive recommendations with respective scores. For the case of individual recommenders, these scores can be from the neural network model, cosine similarity or hybrid.

The back-end of these different recommenders is all explained in the previous subsections. The figure 3.16 shows individual recommendations for an artificial intelligence project. There is a dropdown that has choices such as *neural networks*, *cosine similarity* and *hybrid*. Baseline neural networks give individual recommendations that come from the machine learning model. Cosine similarity method checks the angle between talent and project vectors. Lastly, hybrid mode combines neural networks and unsupervised

similarity to come up with new predictions.

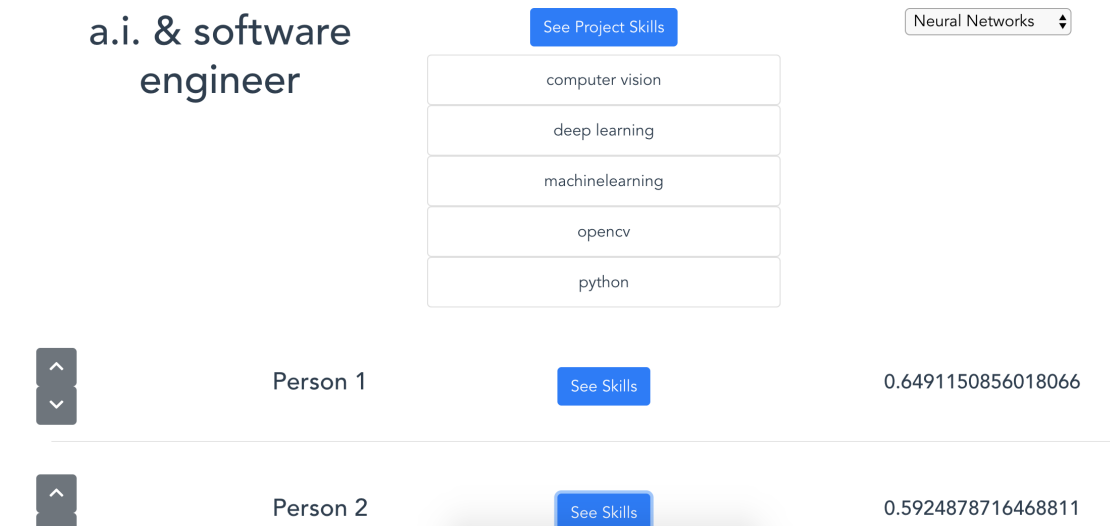


Figure 3.16: A screenshot from the list of all recommendations from neural networks for the project *a.i. & software engineer*

Figure 3.16 shows the results for the project *a.i. & software engineer* on the dashboard that is built by the author. Names of the people that were recommended are anonymized, but the rest of the data are real-life. The example project at hand requires the skills *computer vision*, *deep learning*, *machinelearning*, *opencv* and *python*. Although the skills of the person one is not shown (because that person retains 21 skills, which are too long for the figure), person 1 is not the talent with the most overlapping skills. Instead, person 1 is a talent that was recommended the most by the Motius internal recommender. Next, person 2 knows no skills that the project requires. This lack of knowledge aligns with the fact that it is hard to conclude the neural networks and they are mostly referred to as *black-boxes* [BCR97] [ref TODO evaluation].

In contrast to the figure 3.16, figure 3.17 demonstrates the adequate talents for the same project as before. However, people that are listed are different. As it was explained in 3.2.3, hybrid practice multiplies the results of from the neural networks and skill vector similarity. This way, the results that are recorded, always have some common skills. In the example, these skills are *computer vision* and *machine learning*.

### 3.7.3 Group Recommendations

The dashboard also can perform group recommendations. For the case of group recommenders, the scores can be from baseline neural networks, baseline cosine

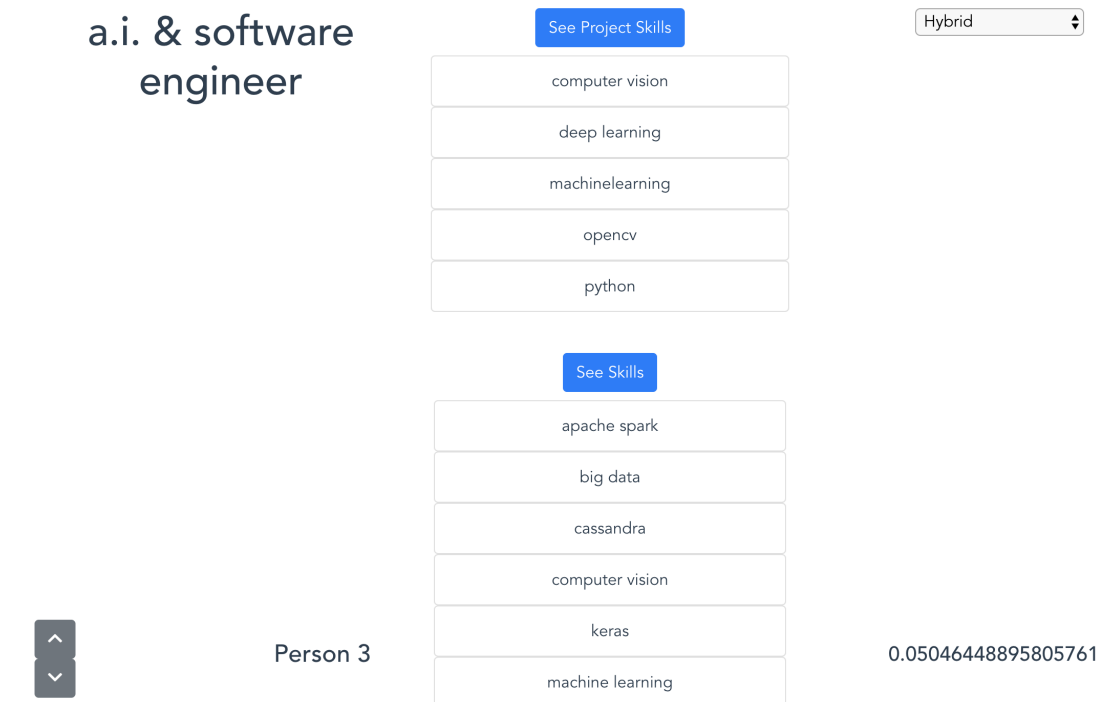


Figure 3.17: A screenshot from the list of all recommendations from neural networks for the project *a.i. & software engineer*

similarity, diverse neural networks or diverse cosine similarity. [TODO: in evaluation-> show screenshots of diverse nn, nn, constant 0 and constant 1 for comparison.]

Baseline cosine similarity, reveal the list of best talents for each project using skill vector similarity and baseline neural networks do the same with neural networks. Diverse cosine similarity and diverse neural networks operate, but they include the likeness of talents for each project to other talents in other projects in the same group.

Figure 3.18 shows the recommendation results for a real-life group with anonymized names. On the top-right part of the figure, a dropdown can be seen. This dropdown reads *Diverse Similarity*. Other options in the dropdown are *neural networks*, *similarity* and *diverse neural networks*. Meaning of these options is already explained in this section. This selection option makes sure that the chosen talents for each project have similar skills and diverse to each other. Between the group name and recommendation mode, a slider is located. This slider determines the constant for the diversity enhancement formula [See 3.5]. This diversity constant can be tuned to value the relevancy or diversity more; a value of 1 means to only consider diversity, 0 means to only consider relevancy and 0.5 gives the balanced result. There are also options to check project and

talent skills. Last but not least, the dashboard provides the option to rate all talents positively or negatively. This part of the thesis is explained in the next section [See section 3.8].

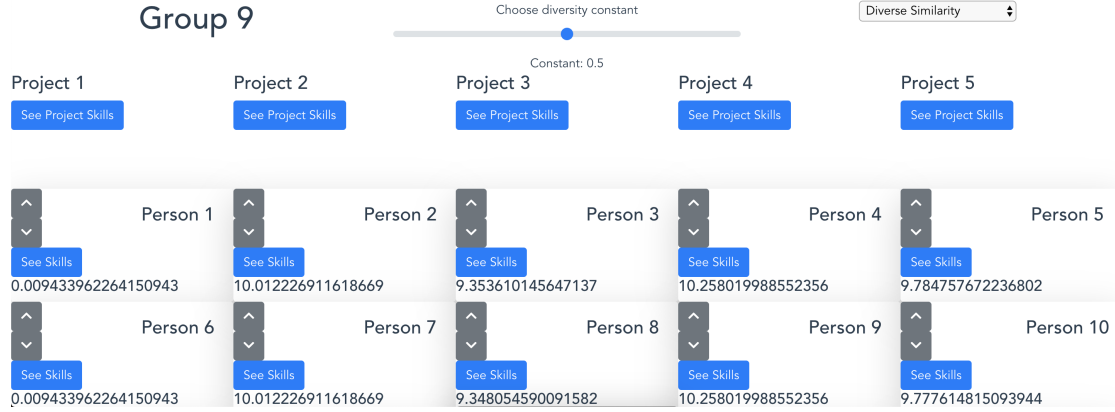


Figure 3.18: A screenshot from the list of all recommendations from diverse cosine similarity for the group 9

### 3.8 Improvement of Recommendations via Feedback Learning

Readers of this thesis may wonder what do the up and down arrows on the figures 3.18, 3.17 and 3.16 mean; these arrows correspond to the feedback learning. Recruiters or Human Resources employees that use the dashboard may send positive or negative feedback to every recommendation.

The feedback that are provided by real people may have direct or indirect effects. To understand the process, it may make sense to check the figure 3.19 first. Feedback data is given to the model as a separate input, and this input is followed by a dense layer with one node with *tanh* activation function. Tanh activation is chosen, because the bias can be negative or positive. [TODO research: tanh]. The output of the bias activation functions is *added* to the result that comes from project-talent information. In this context, adding means actually the mathematical addition operation [See 7.2]. Due to this addition, the outcome from the human feedback has a direct effect, even without retraining.

The default value for any talent bias is 0, and they may be increased up to 1 with positive feedbacks and decrease to -1 with negative feedbacks. These values are added to the result that comes from the dense layer, which gets data from talent and project skills. Since human feedback is stored in the database immediately, they can directly

be used and have a direct effect on the total results.

Feedback learning also has an indirect effect; entering feedback changes the labels of previous training data. This change signifies that positive feedback changes the label of a result to 1(positive) and negative feedback changes it to a 0(negative). That's why, when the developer retrains the model with the new data, it also modifies how the model learns and may also have an impact on other projects.

## 3.9 Summary

This chapter analyzed the practical portion of this thesis exhaustively. We explained how we programmed different recommenders with diverse approaches. We also revealed the dashboard that glues everything together and the proposal to improve recommendations with the help of feedback learning.

The next chapter focuses on the evaluation of the results that were programmed in the scope of this section.



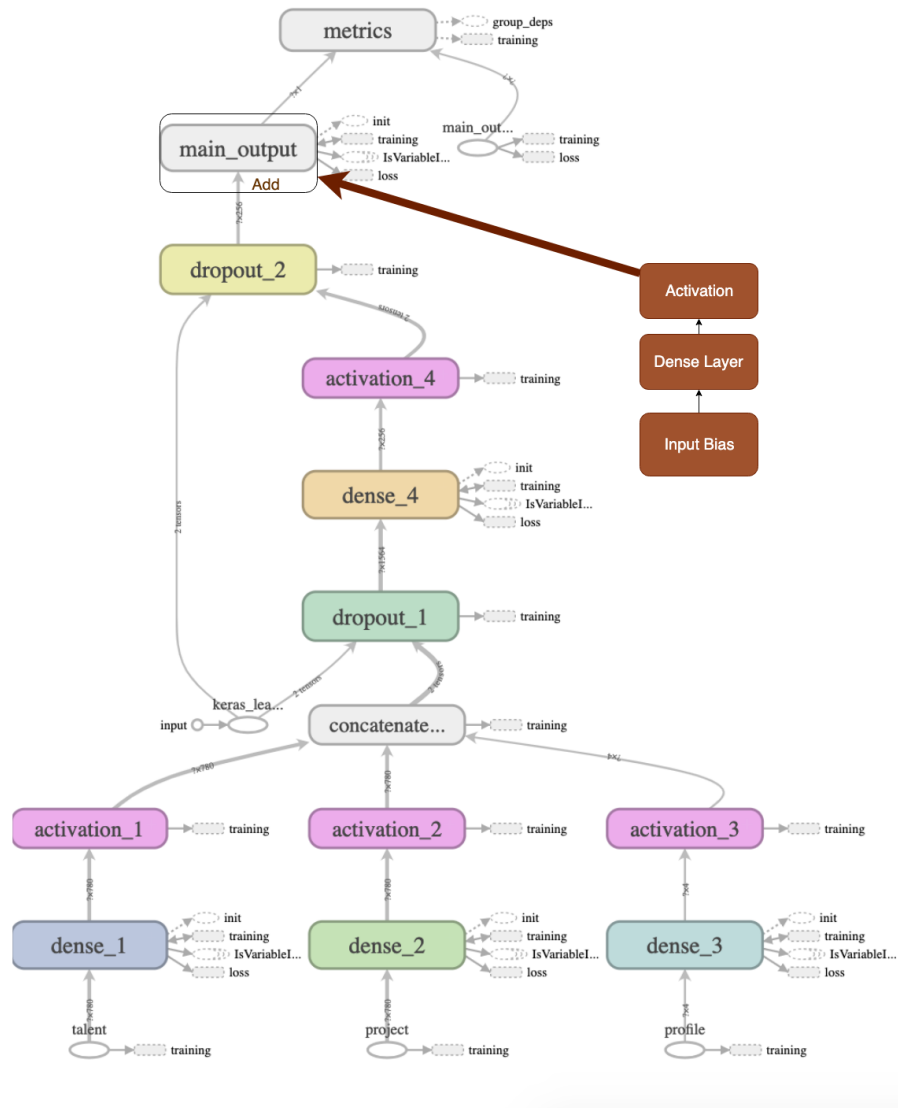


Figure 3.19: Neural networks model with the addition of feedback loop bias

## 4 Evaluation

This chapter focuses on assessing the results of the recommender systems that were created in scope of the chapter 3. Recommendation systems have a variety of properties that may affect user experience, such as accuracy, robustness, scalability, and so forth [SG11]. In this thesis, we mostly focus on characteristics such as accuracy, diversity and others that are related to these two.

In the first years of recommender systems, the developers of recommendation systems only focused on accuracy. However, this has started to change and now we know that accuracy is not always equals user satisfaction. Before starting this thesis, we analyzed what properties are important to improve user satisfaction in job recommender systems. The research led us to the diversity as a metric [CHV15].

When we recommend a talent to a project, the only important evaluation properties are accuracy of the first item in the list, accuracy of top  $n$  items in the list and the overall list value. We can find out the first two via evaluation algorithms. However, the overall list value can only be acquired from human observers.

When we recommend multiple talents to a group of projects, we have the hypothesis that the extra evaluation metric diversity is also important to increase the user satisfaction. That's why, we need evaluation results for diversity, accuracy and user satisfaction, so that prove or disprove the hypothesis.

While we are working on our hypothesis, we use all three types of experiments; offline, online and user studies [SG11]. Offline evaluation serves the purpose of calculating the accuracy and diversity. This type of experiments don't involve human feedback and are calculated via algorithms. When we aim for more advanced metrics, like user satisfaction, we have to rely on other types of experiments like user studies; we ask show users the results and ask their opinion about overall list value. Lastly, we also employ online experiments by allowing them to interact with the system and reranking the talents using the feedback loop [See section 4.5].

According to the research [SG11], there are some guidelines that we need to follow to conduct a successful experiment; a *hypothesis*, *controlling variables* and *generalization power*. Our hypothesis is that the diversity increases user satisfaction and serves as an important factor near accuracy. The controlling variables are the factors that stay the same during different experiments. We need those variables to make sure that different results are comparable. In our case, the trained dataset is the same. Also,

when we conduct user studies, we ask about the results of the same groups and projects. This way, the user study results are directly comparable. The last guideline is the generalization power; to make sure that our model generalizes well, we combine and adapt *freelancer.com* and Motius datasets together. That's why, the generalization power for different job recommender datasets is secured.

## 4.1 Unsupervised Individual Recommendation

As it was explained in the section 3.2, unsupervised individual recommendation addresses proposal of single talents to single projects using different methods. What these methods have in common is that, they all ignore labels of data and only consider features. The first approach is recommendation via feature similarity, the second one suggests the most popular candidates and the last one is the combination of first two.

When we calculate the accuracy for these methods, we use performance measures such as *precision* and *recall* [BOH15]. Precision refers to the percentage of your results which are relevant and recall refers to the percentage of total relevant results correctly classified by your algorithm [DG06].

$$recall \equiv sensitivity = \frac{\#truepositives}{\#truepositives + \#falsenegatives} \quad (4.1)$$

$$precision = \frac{\#truepositives}{\#truepositives + \#falsepositives} \quad (4.2)$$

The above equations show how we calculated recall and precision. True positives in our case explain selected and relevant talents. False positives are selected but irrelevant talents. Lastly, false negatives are not selected but relevant talents.

To be more specific, true positive is the case, when the first person in recommendation list is the person that won the project. False positives disclose all non-awarded bidders that are first in list and false negatives demonstrate all bidders that are not first in the list but won the project in real life.

When we start to calculate results, we see that precision and recall always have the same results for the freelancer dataset. For each project, the number of true positives are either one or zero. The number of false negatives or false positives are also one or zero for each project. If the prediction was correct, the number of true positives are one. In this case, both the number of false negatives and false positives are zero.

The takeaway from the previous paragraph does not hold true for every dataset. For Motius case, there are more than one talent that got positive feedback for each project. That's the number of true positives can vary from zero to the number of talents that received invitation. Next, the number of false negatives can have values from zero to

the number of people that were accepted. Last of all, the number of false positives also range from zero to number of people with positive feedback for each project. Therefore, it is safe to say that those values vary for each project.

In the next subsections, we demonstrate the evaluation results for various approaches.

#### 4.1.1 Existing Company Recommender

The sponsor of this thesis, Motius, runs an internal recommendation system to suggest talents to roles. The internal mechanism that submits recommendations has stored the logs of the past recommendations in an internal database. This database has included all important details, such as the name of the roles, projects, talents, recommendation strength and if the person got an invitation to the next stage. We used the logs to conduct an offline evaluation.

##### Offline Evaluation

First of all, the logs contained 140914 recommendations for 375 roles. There are only 961 positive labels, which means the people who got accepted on the recommendation list. Therefore, we can conclude that there are 2-3 accepted talents for every role. We can also confirm that the most of the logs consist of talents that got rejected.

As part of the evaluation, we checked the rate of an accepted talent being the first in the recommendation list and being in the top 5 of the recommendation list, so that the results are directly comparable with the recommender systems that we implemented. The first talents in the list only get accepted in 7% of the roles and the accepted talents were only a part of 21% of the projects. Lastly, the average rank of accepted people was 161.

#### 4.1.2 Recommendation by Similarity

Subsection 3.2.1 revealed the implementation details of the approach recommendation by feature similarity. Now, we show the evaluation results.

##### Offline Evaluation

For this type of recommender, we used the same evaluation metric for different purposes. First of all, we assessed different flavors of implementation mechanisms to select the best one. Secondly, we calculate, final accuracy score for each individual unsupervised recommender, so that the recommenders are comparable.

As figure 4.1 depicts top 5 accuracy from different flavors of similarity recommendation systems. Top 5 accuracy explains the case that the recommender suggests a bidder

Table 4.1: A table that shows results of different implementation settings of recommendation by similarity.

A	B	C	D
0.28	0.36	0.31	0.35

list and the correct result is searched in top 5 elements of this lists, which contains bidders with a descending recommendation strength.

The depths of freelancer.com dataset are already defined in 3.1.1. To sum it up, each project lists their required skills. These required skill don't have any any value, so they are used as zero or one. Each talent accommodates a skill vector, which encloses the number of projects that the person finished. For example, if an example talent 1 participated in 10 projects that only required *Python* and 2 projects that only required *JavaScript*, this person would have a 10 for Python, a 2 for JavaScript and zero for rest of the skills.

When we do talk about recommendation by similarity, it suggests that each row is normalized first, so that the greatest values correspond to one the smallest zero. Then the cosine similarity of the feature vector of the picked project and the feature vector of all talents are calculated. This cosine similarity is used as the recommendation strength of each talent for a particular project. Then, we sort these values in a descending order.

**A** In this first setting, we just use the plain mode like it was told in the previous paragraph. This means normalizing rows and calculating the cosine similarity. The accuracy we reached is 28%.

**B** After carefully checking and debugging the setting A, we found out that there are many cases of winners with no skill vector. This indicates that, there are many projects that hired talents with zero experience according to the freelancer.com dataset. Since, this method picks talent just by looking at their features, it is impossible to predict the winners for those projects. That's why we removed the talents that have no skill vector present. Doing this increased the accuracy to 36%.

**C** When the developer of this thesis checked the data, it seemed like some further improvements could be made. An idea was normalizing columns additional to the rows. Normalizing rows was what we were doing in other setups, which is scaling for each talent. Normalizing columns means setting the highest value for each skill to one and the lowest value to zero. Doing this has the advantage that the talents,

who finished a lot of projects with particular skills won't lose that advantage against others. However, this method decreases the accuracy to 31%. This implicates that when the project owners choose talents, the information of how many projects the talents completed with relevant skills doesn't play a big role. However, it's hard to draw conclusions, since every employer is different.

**D** To test the suspicion that we have grown, we start losing some information on purpose. This version changes the talent skills to zeroes and ones just like the project feature vectors. If a person has completed any project that involved the relevant skill, that is a one. If this person didn't work with a skill before, that value becomes zero. In the end, we reach 35% accuracy, which is not the best among others but also not the worst, considering we slimmed down the data and lost a lot of information about talent skills.

In the end, we decide to use the last version. Because that version has the smallest dataframe size, which is beneficial according to what we explained before [See subsection 3.1.2]. Also the accuracy is the second best among others and its structure is similar to what we programmed in subsection 3.3.2.

When to try to guess the project winners with the chosen version, we reach the accuracy of 0.27. When we check if the winner is in the top 5 of the score list, then the accuracy is 0.35, like we mentioned before.

Lastly, it must be noted that during preprocessing, we remove the projects that has less than 5 bidders. Therefore, the minimum number of applicants is 5. We continue with the online evaluation of this recommender.

## Online Evaluation

[TODO]

### 4.1.3 Recommendation by Popularity

Popularity recommender is a recommender that is easy to explain: it checks the profile information of every bidder from a specific project. Then it sorts them by their experience level and recommends people that have the most experience.

## Offline Evaluation

Offline evaluation of this method is straightforward. Like we did before, we check if the winner is in the top one and top 5 in the recommendation list that holds the top bidders with descending recommendation score. Top one accuracy is 0.06 and

top 5 accuracy is 0.45. This shows that the popularity recommender is worse than the similarity recommender at guessing if it is given just one chance. However, it runs better than the similarity recommender if it is given 5 chances.

## Online Evaluation

### 4.1.4 Hybrid Recommendation

According to [Bur02], hybridization is a valid technique to combine several recommendation methods to produce a single recommendation. There are different methods to achieve this, but we combine weighted scores of both of other recommenders. In our version, we have decided to add results from both recommender with some weights. The reason for that is to be able to predict results for as many projects as possible. The 4.2 depicts for how many projects is it possible to predict the results. The slimmed down version that we developed to recommend by similarity, can't predict for all projects, because a big percent of the winners don't have a skill vector. When we combine both results, we also increase our coverage, since we get to use the results from two different recommenders.

Table 4.2: A table that shows the amount of projects that we can generate predictions.

Similarity	Popularity	Hybrid
4987	20433	14152

## Offline Evaluation

Since we decided for a weighted addition of two recommenders, we also have the flexibility to decide for the weights. We tried different weights for similarity and popularity recommenders and calculated the accuracy for these different settings.

Figures 4.1 and 4.2 depict the results of what happens with different similarity weights. Although, the figures only show the similarity weights, popularity weights can also be calculated by subtracting the similarity weight from one. The figure 4.1 displays top 1 and top 5 accuracy for different weights. When the similarity weight is zero, top 1 accuracy is the lowest and top 5 accuracy is highest. When the similarity weight is one, both accuracies stay in the middle. The figure 4.2 show the amount of projects that is possible to predict results with the recommender at hand. That graph includes a clear function that is inversely proportional to the similarity weight. The highest project coverage can be seen on highest popularity weight and the lowest project coverage can be seen on the highest similarity weight.

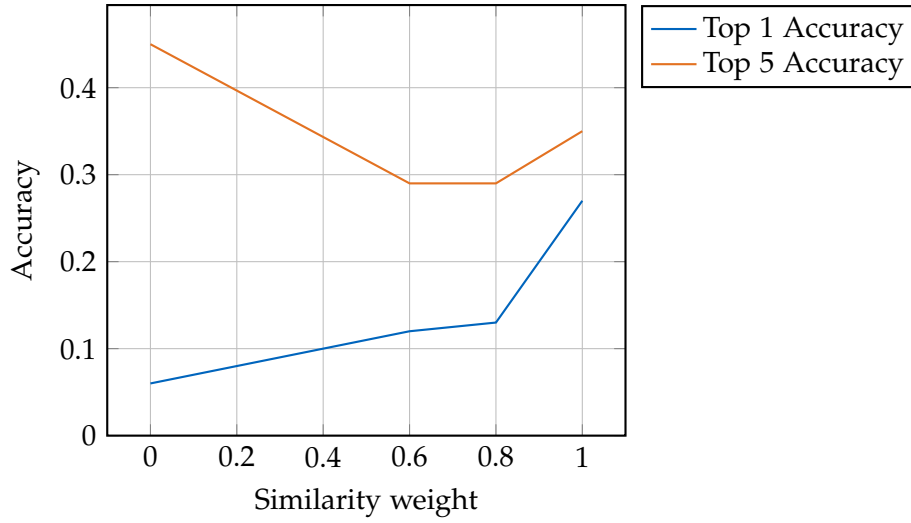


Figure 4.1: Effect of weight selection on hybrid recommenders to the accuracy

It is clear that there is no obvious best weight choice, since we both want to increase project coverage and accuracies. We chose the similarity weight 0.6 and popularity weight 0.4 to increase these results.

### Online Evaluation

## 4.2 Supervised Individual Recommender

The details of supervised individual recommender is already clarified in the section 3.3. This type of recommender suggests individual talents to individual projects using neural networks. There are two different settings that we apply supervised individual recommender; using sparse input and using embeddings.

### 4.2.1 Using Sparse Input

Since the sparse data is too big for computers to handle, this one only contains offline evaluation results. The performance of this subtype of recommender is measured for different amount of layers, different cost functions, other activation functions and various optimizations. In the end, it wouldn't make sense to visualize all of the distinct results. However, we must note the maximum accuracy that is measured with the test set is 79%. Here the accuracy doesn't reveal top 1 or top 5 prediction that we used before. This prediction with the test set demonstrates that the model is able to predict



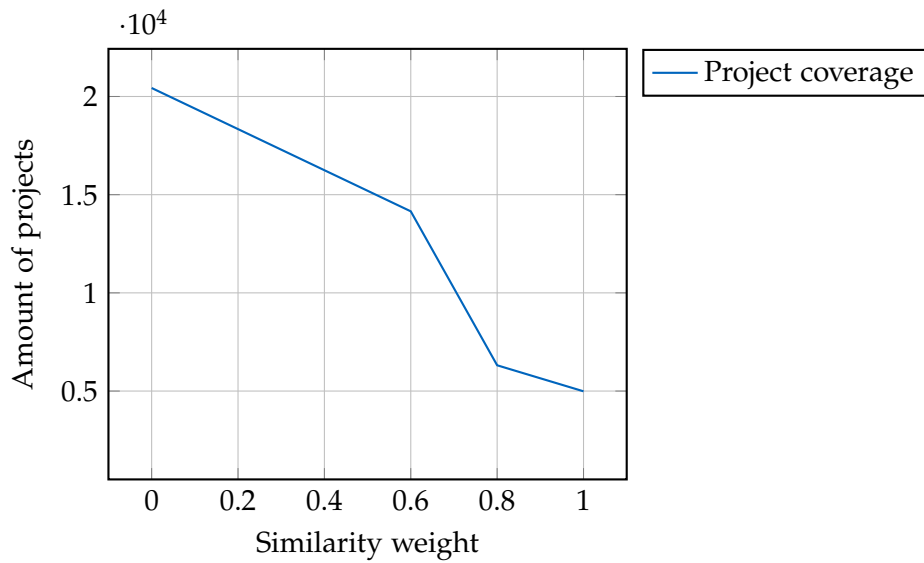


Figure 4.2: Effect of weight selection on the amount of projects that we can generate predictions.

the right results for 79% of the talent-project pairs. The neural network generates a value between 0 and 1 for each pair and values that are below 0.5 interpreted as the person should be rejected and above 0.5 gets treated as the talent should be hired. That's why the model successfully predicts 79% of those pairs.

Predicting the winners of projects is another story. Through different parameters and settings, the most that we have reached is 21% for the first-place accuracy and 59% for the top 5 accuracy. It is fair to note that these maximum values are reached with the help of extra profile information like experience level on top of skills [TODO: if more text needed add evaluation information from SparseFreelancerBias.ipynb].

#### 4.2.2 Using Embeddings

To be able to use embeddings, we reduce the dimensionality of the data first. This makes the dataframe smaller in size but it also decreases the performance, since some information is lost during the reduction.

This version is the actual approach that is used in the dashboard. That's why we gained evaluation results for both offline and online cases.

### Offline Evaluation

The maximum accuracy that is reached to predict the test set results is 72%, which is lower than the sparse version. When the top 1 and top 5 talents for each project is calculated, the best that is achieved are 18.5% and 56% respectively.

This is calculated with the extra profile information included. When we don't add them, the biggest value for guessing the winner is 9%. [TODO: if more info needed, Embedding notebooks]

### Online Evaluation

## 4.3 Unsupervised Group Recommender

This is another type of recommender that we implemented in the dashboard is the unsupervised group recommender and contains different modes such as baseline and diverse.

### 4.3.1 Baseline Recommender

The baseline unsupervised group recommender recommends a group of talents by maximizing the relevancy of project-talent pairs. However, this method doesn't check the relation of talents that are chosen.

### Offline Evaluation

There are many evaluation methods that count under offline evaluation. Since they are employed first time in this thesis, we explain them in a detailed way.

**Accuracy** The first detail that we check is the top 1 and top 5 accuracy, like we did before. This didn't change from results of the section 4.1.2. Which means that the top 1 accuracy is around 20% and top 5 is around 30%.

**Diversity** There are many evaluation mechanism to measure diversity and the first equations were coined in the beginning of 2000's [SM01].

$$ILD = \frac{1}{|R|(|R| - 1)} \sum_{i \in R} \sum_{j \in R} d(i, j)$$

The above *inter-list diversity* is the first to calculate. This mechanism calculates the diversity inside a recommendation list. As a diversity measure, we use the cosine

distance, which is calculated as  $1 - \text{CosineSimilarity}$ , which is used many times in this thesis. With various sizes of groups from the freelancer.com dataset, we calculated the average *ILD* of 40% with this baseline suggestion engine.

**Unexpedtedness** Unexpectedness tells us how unforeseen the data is to the recruiters. In the formula below  $\mathcal{J}_u$  is a symbol for set of talents that the recruiter has interacted with. When we know about this past, we can compare the talents with the people in the set. We want to maximize the distance between already seen and not yet seen talents. The result that we reach in baseliner is 0.61.

$$Unexp = \frac{1}{|R| |\mathcal{J}_u|} \sum_{i \in R} \sum_{j \in \mathcal{J}_u} d(i, j)$$

$$\text{where } \mathcal{J}_u \stackrel{\text{def}}{=} \{i \in \mathcal{J} | r(u, i) \neq \emptyset\}$$

[TODO: if not enogh: unexp, Gini, shannonggroup-rec.ipynb]

## Online Evaluation

### 4.3.2 Diverse Recommender

Diverse unsupervised group recommender optimizes the relationship of the chosen talents in a group on top of maximizing the relevancy of project-talent pairs. Both project-talent and talent-talent relationships are optimized via cosine similarity. We continue with the evaluation results.

The way this type of recommender functions is already explained in the implementation chapter[See section 3.4.2]. We can sum it up using the formulas:

$$R_{opt}(\lambda) = \arg \max g(R, \lambda)$$

$$g(R, \lambda) = (1 - \lambda) \frac{1}{|R|} \sum_{i \in R} f_{rel}(i) + \lambda div(R)$$

In the above equations, the variable  $\lambda$  can be tuned according to our needs. A value closer to zero means, the recruiters want to see more relevant scores and a value closer to one shows more diverse results.

## Offline Evaluation

When we evaluate a diverse recommender, we need to show variations of results for different  $\lambda$  values. Because, every small or big modification of this value will have an effect on the accuracy, diversity or overall list value.

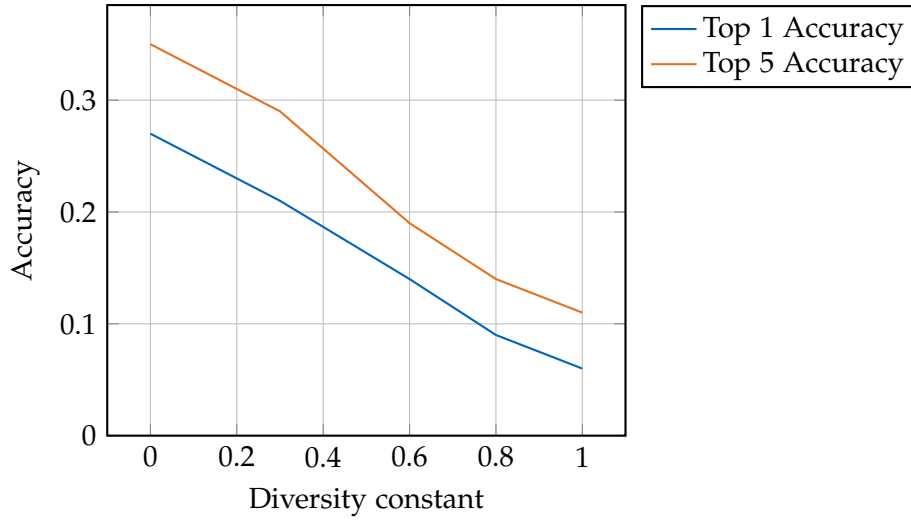


Figure 4.3: Effect of diversity constant on unsupervised group recommender to the accuracy

**Accuracy** The figure 4.3 depicts the impact of diversity constant change to the accuracy, which are calculated on freelancer.com dataset.

**Diversity** For the diversity, we use the same equation that we used in the paragraph 4.3.1. However, a major change is that we have to calculate the inter-list-diversity for different  $\lambda$  values.

The figure 4.4 portrays, how changing the diversity constant affects the actual diversity of the recommendation lists that are generated by the recommendation engine.

**Unexpectedness** We use the same formula, which was explained in the paragraph 4.3.1. Again, we calculate the values separately for different diversity rates and drawn on the figure 4.5. It is obvious that the diverse recommender didn't increase the unexpectedness as it improved the diversity.

#### Online Evaluation

### 4.4 Supervised Group Recommender

The supervised group recommender takes advantage of neural networks to ascertain the relevancy of talents to the project and may use cosine similarity to ensure the variance of

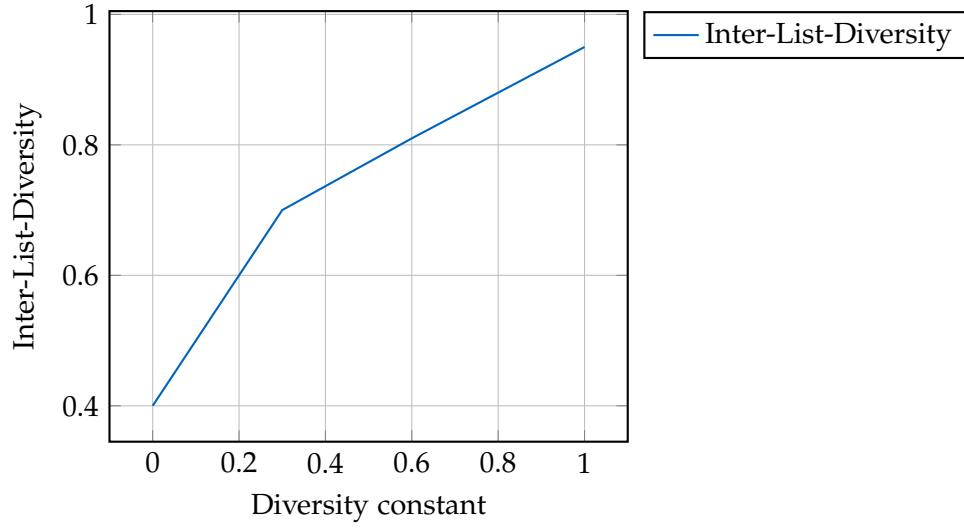


Figure 4.4: Effect of diversity constant on unsupervised group recommender to the diversity

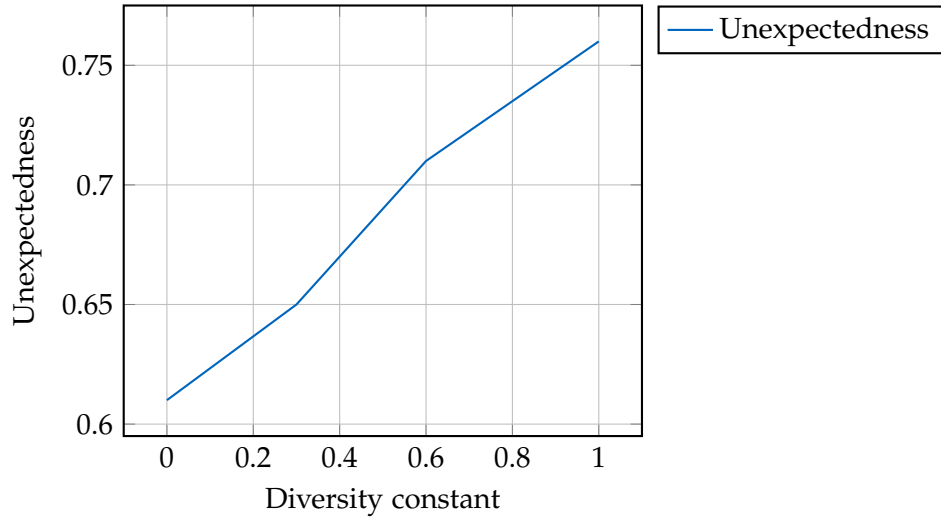


Figure 4.5: Effect of diversity constant on unsupervised group recommender to the unexpectedness

talent recommendation lists. It has two modes: baseline and diverse recommendations.

#### 4.4.1 Baseline Recommender

The baseline supervised group recommender maximizes the relevancy of each project-talent pair for a given group. It doesn't aim to optimize the diversity of talents.

##### Offline Evaluation

Again, we run algorithms to determine the accuracy, diversity and unexpectedness of the current recommender.

**Accuracy** When the top 1 and top 5 talents for each project of groups is calculated, the best that is achieved are 18% and 56% respectively.

**Diversity** We calculate the diversity according to the formula in 4.3.1; with various sizes of groups from the freelancer.com dataset, we calculated the average *ILD* of 44% with this baseline recommendation engine.

**Unexpectedness** The unexpectedness that is generated by this recommender was spelled out in the paragraph 4.3.1. When the average unexpectedness for all projects are measured, the value that we see is 65.5%.

##### Online Evaluation

#### 4.4.2 Diverse Recommender

With the help diverse supervised group recommender, we combine the forces of two different operations to boost the project-talent relevancy and also control the variance between selected talents.

##### Offline Evaluation

In the following paragraphs, we demonstrate the evaluation results using different mechanisms and for distinct diversity values.

**Accuracy** The figure 4.6 depicts the impact of diversity constant change to the accuracy, which are calculated on freelancer.com dataset.

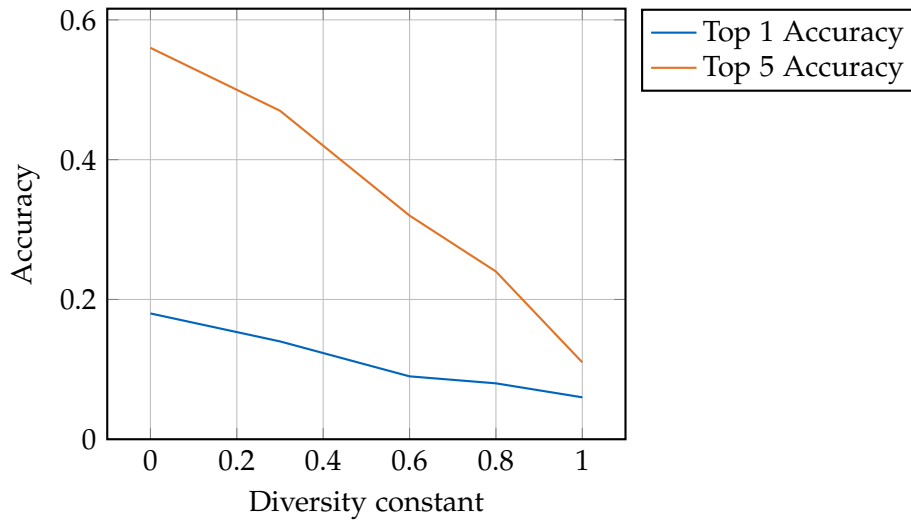


Figure 4.6: Effect of diversity constant on supervised group recommender to the accuracy

**Diversity** To calculate diversity, we use the same equation that we used in the paragraph 4.3.1 with different diversity constants. The figure 4.7 shows the diversity in recommendation lists under the effect of diversity constant.

**Unexpectedness** We use the same formula, which was explained in the paragraph 4.3.1 and the values are calculated separately for different diversity rates that are drawn on the figure 4.8.

#### Online Evaluation

### 4.5 Feedback Loop

Improving the outcomes via a feedback loop is an essential part of this thesis. Like it was told before [See 3.8], the loop has a direct and an indirect effect. When the recruiters that use the dashboard, post feedback to talents, the recommendation list is instantly changed accordingly. This is called the direct effect. Posting feedback, also modifies the existing dataset that is used to train the model. Sending a positive feedback modifies the datapoint to a positive label, which is one. Also, sending a negative feedback modifies the datapoint to a negative label, which is zero. After that, the model will produce *better* results, when we retrain with the modified data.

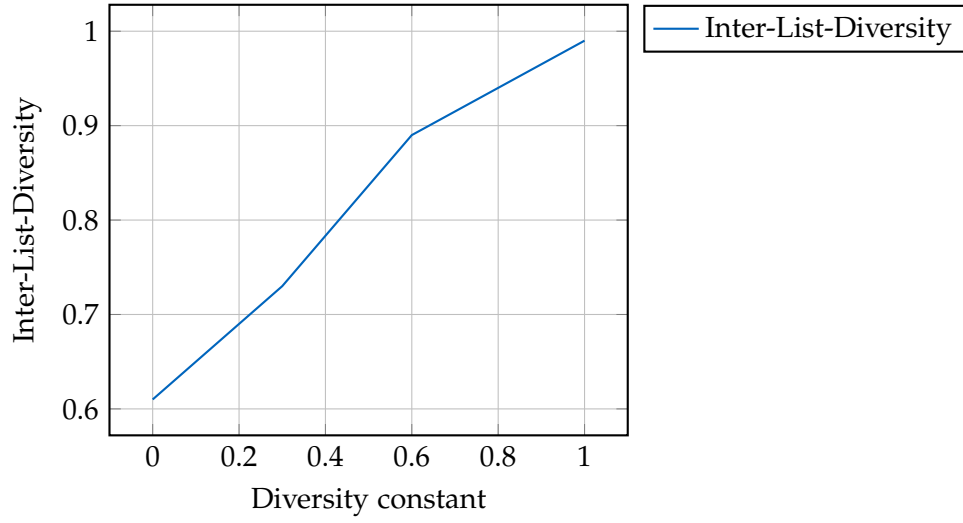


Figure 4.7: Effect of diversity constant on supervised group recommender to the diversity

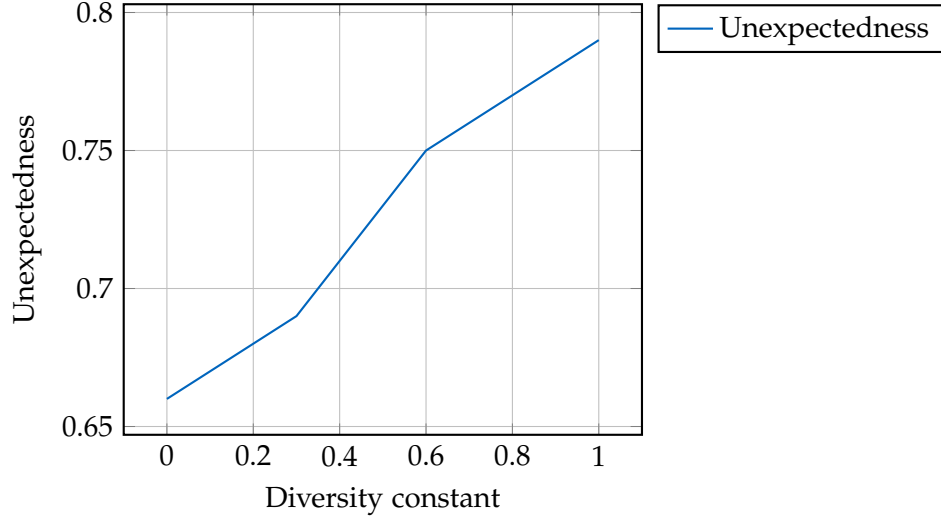


Figure 4.8: Effect of diversity constant on supervised group recommender to the unexpectedness



This section focuses on evaluation of this loop. It not feasible to conduct offline evaluation for the feedback loop, because the model that generates predictions tries to maximize the accuracy already. We already have a model and we want to improve it to satisfy the needs of recruiters. A logical evaluation method for the feedback loop would be a combination of online evaluation and user studies.

### 4.5.1 Online Evaluation and User Study

In this round of evaluation, we had eight participants that recruited talents before. The functional dashboard is given to each of the participants and the author of this thesis clearly gave instruction how the recommenders, the dashboard and the feedback loop work. Then, we asked the participants to choose some groups and arrange the loops according to their ideal scenario.

Participants gave more than 200 feedbacks in total. Considering the fact that the training data consists of more 350.000 data points, the amount of feedbacks are low. From the criteria that the recruiters stated, the author continued to add more feedbacks, making them 3500 in total. After this, the model is retrained with the modified data. Last of all, the results are shown again the recruiters and their opinions are asked again.

[TODO comparison of before and after]

## 4.6 Summary

## 5 Discussion

### 5.1 Problems about datasets

#### 5.1.1 Problems about Motius dataset

Another funny thing in Motius data: pos: android dev. 2 x lorenzo: 1 rejected 1 accepted

To sum up:

Only a small portion of Motius and Freelancer features overlap Since Motius data doesn't have many data points, those unique features stay untrained Motius recommender produces duplicates Many people were selected although they don't meet skill requirements(also on freelancer) There are some people with many skills entered(for example Lorenzo, over 20) and there are many people that has only 1-2 skills in system Selected people are not selected because their skills fit better than the others.

#### 5.1.2 Problems about Freelancer dataset

Problem: For example: the user <https://www.freelancer.com/u/crystalbernardo> has 0 experience, 0 skills, was not the cheapest she got the project <https://www.freelancer.com/projects/academic-writing/long-term-academic-writer-needed-11688789/>.

### 5.2 Comparison of Individual Recommenders

### 5.3 Comparison of Group Recommenders

### 5.4 Conclusion

bla

bla

bla

bla

# 6 Conclusion

## 6.1 Introduction

Recommender Systems (RSs) are software tools and techniques that provide suggestions for items that are most likely of interest to a particular user [RRS15].

Suggestions and items depend on the field that recommender system is applied. For example, for the topic of news article recommenders, the aim will most likely be suggesting news to the readers. In the field of job recommenders though, these suggestions can be bidirectional. Meaning that, job postings can be suggested to applicants or resumes can be recommended to the human resources team of a company.

See Table 6.1, Figure 6.1, Figure 6.2, Figure 6.3.

Table 6.1: An example for a simple table.

A	B	C	D
1	2	1	2
2	3	2	3

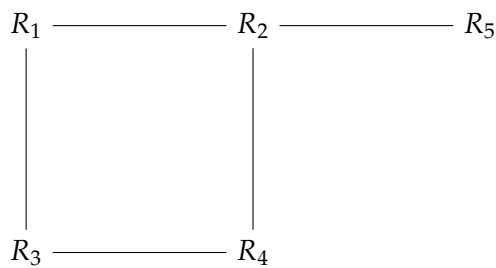


Figure 6.1: An example for a simple drawing.

## 6.2 Conclusion

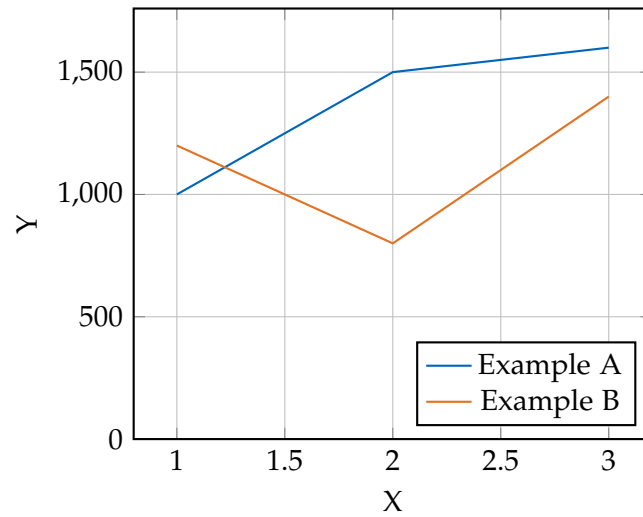


Figure 6.2: An example for a simple plot.

```
SELECT * FROM tbl WHERE tbl.str = "str"
```

Figure 6.3: An example for a source code listing.

## 7 Appendix

```
def nn_embedding(features, dimensions):
    talent = Input(shape = (features,))
    project = Input(shape = (features,))
    output_dim = int(dimensions ** 0.25) + 1

    talent_embedding = Embedding(dimensions + 1, output_dim,
    input_length=features, mask_zero=True )(talent)
    project_embedding = Embedding(dimensions+ 1, output_dim,
    input_length=features, mask_zero=True)(project)

    # these are required because of mask zero
    talent_embedding = Lambda(lambda x: x,
    output_shape=lambda s:s)(talent_embedding)
    project_embedding = Lambda(lambda x: x,
    output_shape=lambda s:s)(project_embedding)

    merged = Concatenate()([talent_embedding, project_embedding])
    merged = Flatten()(merged)
    merged = Dropout(0.5)(merged)
    main_output = Dense(1, activation='sigmoid')(merged)

    model = Model(inputs=[talent, project], outputs=[main_output])
    model.compile(optimizer='adam', loss='binary_crossentropy',
    metrics=["accuracy"])

    return model
```

Figure 7.1: The code of the model for both Motius and Freelancer data

```
bias = Input(name = 'bias', shape = (1,))
talent_bias = Dense(1, activation = 'linear')(bias)
talent_bias = Activation("tanh")(talent_bias)
main_output = Add()([output, talent_bias])
```

Figure 7.2: Simplified version of the extra code for enabling feedback

## List of Figures

2.1	2D representation of a 4D cube. The colors indicate the depth in fourth dimension [LV07]. . . . .	15
3.1	An example project from the Freelancer Website . . . . .	58
3.2	The winner and other bidders to the same project . . . . .	59
3.3	The list of tops skills by a talent on Freelancer web page . . . . .	60
3.4	The talent skill matrix from freelancer.com . . . . .	62
3.5	The talent extra information matrix from freelancer.com . . . . .	63
3.6	The graph that explains the sparse input model . . . . .	65
3.7	3D version of the embedding space that is created for projects of this thesis . . . . .	67
3.8	Threshold figure . . . . .	68
3.9	Training data that contains padded embedding skill vectors . . . . .	68
3.10	Explained variance ratio for difference number of PCs is shown. . . . .	72
3.11	Silhouette scores of many different $k$ values of k-means . . . . .	73
3.12	Centers of clusters that are projected on a 2D space . . . . .	74
3.13	Examples of some centers of clusters that are projected on a 2D space . . . . .	74
3.14	Main screen of the dashboard . . . . .	75
3.15	A snippet from the list of all projects that start with the letter $a$ . . . . .	76
3.16	A screenshot from the list of all recommendations from neural networks for the project <i>a.i. &amp; software engineer</i> . . . . .	77
3.17	A screenshot from the list of all recommendations from neural networks for the project <i>a.i. &amp; software engineer</i> . . . . .	78
3.18	A screenshot from the list of all recommendations from diverse cosine similarity for the group 9 . . . . .	79
3.19	Neural networks model with the addition of feedback loop bias . . . . .	81
4.1	Weight figure . . . . .	88
4.2	Coverage figure . . . . .	89
4.3	Unsupervised-diversity-accuracy Figure . . . . .	92
4.4	Unsupervised-diversity-diversity Figure . . . . .	93
4.5	Unsupervised Diverse Group Unexpectedness Figure . . . . .	93
4.6	Accuracy in Diverse Supervised Group Recommender . . . . .	95
4.7	Diversity in Diverse Supervised Group Recommender . . . . .	96

*List of Figures*

---

4.8	Supervised Diverse Group Unexpectedness Figure . . . . .	96
6.1	Example drawing . . . . .	99
6.2	Example plot . . . . .	100
6.3	Example listing . . . . .	100
7.1	Model Code . . . . .	101
7.2	Model Bias Code . . . . .	102



## List of Tables

4.1	Evaluation mid-results . . . . .	85
4.2	The number of predicted projects . . . . .	87
6.1	Example table . . . . .	99

# Bibliography

- [AB15] X. Amatriain and J. Basilico. "Recommender systems in industry: A netflix case study." In: *Recommender systems handbook*. Springer, 2015, pp. 385–419.
- [Ama+11] X. Amatriain, A. Jaimes, N. Oliver, and J. M. Pujol. "Data mining methods for recommender systems." In: *Recommender systems handbook*. Springer, 2011, pp. 39–71.
- [And15] C. Anderson. "Docker [software engineering]." In: *IEEE Software* 32.3 (2015), pp. 102–c3.
- [BCJ15] G. Beliakov, T. Calvo, and S. James. "Aggregation functions for recommender systems." In: *Recommender Systems Handbook*. Springer, 2015, pp. 777–808.
- [BCR97] J. M. Benitez, J. L. Castro, and I. Requena. "Are artificial neural networks black boxes?" In: *IEEE Transactions on neural networks* 8.5 (1997), pp. 1156–1164.
- [Bee+16] J. Beel, B. Gipp, S. Langer, and C. Breiting. "Research-paper recommender systems: a literature survey." In: *International Journal on Digital Libraries* 17.4 (Nov. 2016), pp. 305–338. ISSN: 1432-1300. DOI: 10.1007/s00799-015-0156-0.
- [BKL09] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.
- [BOH15] R. Burke, M. P. O'Mahony, and N. J. Hurley. "Robust collaborative recommendation." In: *Recommender systems handbook*. Springer, 2015, pp. 961–995.
- [Bur02] R. Burke. "Hybrid recommender systems: Survey and experiments." In: *User modeling and user-adapted interaction* 12.4 (2002), pp. 331–370.
- [Cho18] F. Chollet. *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.
- [CHV15] P. Castells, N. J. Hurley, and S. Vargas. "Novelty and diversity in recommender systems." In: *Recommender Systems Handbook*. Springer, 2015, pp. 881–918.

- [DG06] J. Davis and M. Goadrich. "The relationship between Precision-Recall and ROC curves." In: *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 233–240.
- [GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [Gri18] M. Grinberg. *Flask web development: developing web applications with python*. "O'Reilly Media, Inc.", 2018.
- [Jol11] I. Jolliffe. *Principal component analysis*. Springer, 2011.
- [LV07] J. A. Lee and M. Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.
- [Par+12] D. H. Park, H. K. Kim, I. Y. Choi, and J. K. Kim. "A literature review and classification of recommender systems research." In: *Expert Systems with Applications* 39.11 (2012), pp. 10059–10072. ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2012.02.038>.
- [Pog+17] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review." In: *International Journal of Automation and Computing* 14.5 (2017), pp. 503–519.
- [Rou87] P. J. Rousseeuw. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis." In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65.
- [RRS15] F. Ricci, L. Rokach, and B. Shapira. "Recommender Systems: Introduction and Challenges." In: *Recommender Systems Handbook*. Ed. by F. Ricci, L. Rokach, and B. Shapira. Boston, MA: Springer US, 2015, pp. 1–34. ISBN: 978-1-4899-7637-6. DOI: 10.1007/978-1-4899-7637-6\_1.
- [SA13] R. Sathya and A. Abraham. "Comparison of supervised and unsupervised learning algorithms for pattern classification." In: *International Journal of Advanced Research in Artificial Intelligence* 2.2 (2013), pp. 34–38.
- [SEK04] M. Steinbach, L. Ertöz, and V. Kumar. "The challenges of clustering high dimensional data." In: *New directions in statistical physics*. Springer, 2004, pp. 273–309.
- [SG11] G. Shani and A. Gunawardana. "Evaluating recommendation systems." In: *Recommender systems handbook*. Springer, 2011, pp. 257–297.
- [SM01] B. Smyth and P. McClave. "Similarity vs. diversity." In: *International conference on case-based reasoning*. Springer, 2001, pp. 347–361.

- [SP15] A. Singh and A. Purohit. "A survey on methods for solving data imbalance problem for classification." In: *International Journal of Computer Applications* 127.15 (2015), pp. 37–41.
- [SS97] J. Sola and J. Sevilla. "Importance of input data normalization for the application of neural networks to complex industrial problems." In: *IEEE Transactions on nuclear science* 44.3 (1997), pp. 1464–1468.
- [You18] E. You. "Vue. js." In: *Diakses dari <https://vuejs.org/>, pada tanggal 17* (2018).