



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

**Development of Recommender Systems  
with a Focus on Improving User  
Satisfaction**

Ozan Pekmezci





DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

**Development of Recommender Systems  
with a Focus on Improving User  
Satisfaction**

**Entwicklung von Empfehlungssystemen  
mit dem Schwerpunkt auf der  
Verbesserung der Benutzerzufriedenheit**

Author:	Ozan Pekmezci
Supervisor:	Prof. Dr-Ing. Klaus Diepold
Advisor:	Julian Wörmann, M.Sc.
Submission Date:	15.06.2019

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

Munich, 15.06.2019

Ozan Pekmezci

## Acknowledgments

Throughout the writing of this thesis, I have received a great deal of support and assistance. I would first like to thank the company Motius for supporting me in the evaluation and providing me the dataset.

I want to acknowledge my supervisor, Professor Diepold, for accepting my thesis topic.

I would also like to thank Julian Wörmann, who guided me, brainstormed with me, showed interest, and gave feedback about my thesis. I wish him the best with his career as a doctor and also on his new adventure with his family.

Besides, I would like to thank my parents, Zübeyde and Necmi Pekmezci. I would not be here without your emotional and financial support from the day that I was born.

I want to thank my girlfriend, Cosku Özdemirci, who reminded me how to feel emotions and made life more bearable.

I also want to thank my Munich friends for making life more enjoyable, especially Berkay Soykan, Haydar Sahin, Helin Senbayram, Ulas Özdemirci, Zeynep Sasi and Zeynep Karagoz, who gave me the idea of using this section more productively.

My long-distance friends Ada Gencoglu and Ugur Kocak also supported me emotionally, and I am thankful for having them in my life.

I want to offer special thanks to Akin Akkan, who, although no longer with us, our memories will be in my mind until I die.

I also want to thank some people that I do not know personally: thanks to the organizers of the Fusion Festival for teaching how to resist against the authority when needed and I want to thank the elected mayor of Istanbul, Ekrem Imamoglu, for giving me motivation, hope and showing me how to win hearts without being exclusive or hostile.

# Abstract

Recommendation systems have an essential aim of increasing user satisfaction. However, many developers of these systems do not take into account that user satisfaction does not only depend on simple evaluation metrics like accuracy, but it can also depend on privacy, data security, diversity, serendipity, labeling, and presentation.

This work aimed to find answers to three main questions. The first question is whether high accuracy guarantees high user satisfaction. The second is an extension of the first question and asks if diversity has a positive impact on user satisfaction. The last question asks if a feedback loop increases user satisfaction.

To answer these questions, we have developed models and algorithms to generate predictions. Then we implemented these models and algorithms in a dashboard. Most recently, we conducted offline and online evaluation rounds and user studies to validate our hypothesized questions. First, we have seen that high accuracy does not always result in high user satisfaction. This is because other properties can also affect user satisfaction and a slight increase in accuracy can lead to a reduction in other properties that can also reduce user satisfaction. These properties can vary from setting to setting and from application to application. We have found that diversity is a critical factor in our use-case, which is the recommendation of a group of talents for a project. Although accuracy and diversity are inversely proportional, a slight increase in diversity led to slightly higher user satisfaction. However, a significant increase in diversity led to a decline in user satisfaction. Last, we checked if a feedback loop increases user satisfaction. Although initial implementation and training with a feedback loop does not increase user satisfaction, subjects in the user study reported more positive feedback after more and more feedback. After inserting more than a thousand feedbacks and re-training with the resulting data, users were happier with the results. We have concluded that user satisfaction in recommender systems can be increased if developers take user feedback into account and not solely focus on algorithmic accuracy.

It is hoped this thesis will inform researchers and developers of recommender systems about increasing user satisfaction.

Keywords: recommendation systems, accuracy, diversity, evaluation properties, group recommendations, job recommender system

# Contents

<b>Acknowledgments</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>1. Introduction</b>	<b>1</b>
1.1. Motivation . . . . .	2
1.2. Contribution . . . . .	2
<b>2. Review of Literature and Research</b>	<b>4</b>
2.1. Types of Recommender Systems . . . . .	4
2.1.1. Content-Based Filtering . . . . .	4
2.1.2. Collaborative Filtering . . . . .	7
2.1.3. Knowledge-Based Recommender Systems . . . . .	8
2.1.4. Hybrid Recommender Systems . . . . .	8
2.2. Types of input in recommender systems . . . . .	8
2.3. Recommender System Evaluation Properties . . . . .	9
2.3.1. Summary . . . . .	12
2.4. Diversity . . . . .	12
2.4.1. Diversity Evaluation . . . . .	13
2.4.2. Diversity Enhancement Approaches . . . . .	15
2.5. Artificial Neural Networks . . . . .	15
2.5.1. Embeddings . . . . .	16
2.6. Summary . . . . .	16
<b>3. Implementation</b>	<b>18</b>
3.1. Datasets . . . . .	18
3.1.1. Freelancer.com Dataset . . . . .	19
3.1.2. Motius Dataset . . . . .	22
3.2. Unsupervised Individual Recommender . . . . .	22
3.2.1. Recommendation by Similarity . . . . .	22
3.2.2. Recommendation by Popularity . . . . .	23
3.2.3. Hybrid Recommendation . . . . .	23

3.3. Supervised Individual Recommender . . . . .	23
3.3.1. Using Sparse Input . . . . .	24
3.3.2. Using Embeddings . . . . .	28
3.4. Unsupervised Group Recommender . . . . .	31
3.4.1. Baseline Recommender . . . . .	31
3.4.2. Diverse Recommender . . . . .	31
3.5. Supervised Group Recommender . . . . .	32
3.6. Group Recommendation using Clustering . . . . .	33
3.7. Dashboard to show data and enter feedback . . . . .	35
3.7.1. General Dashboard . . . . .	35
3.7.2. Individual Recommendations . . . . .	37
3.7.3. Group Recommendations . . . . .	38
3.8. Improvement of Recommendations via Feedback Learning . . . . .	40
3.9. Summary . . . . .	41
<b>4. Evaluation . . . . .</b>	<b>43</b>
4.1. Unsupervised Individual Recommendation . . . . .	44
4.1.1. Existing Company Recommender . . . . .	44
4.1.2. Recommendation by Similarity . . . . .	45
4.1.3. Recommendation by Popularity . . . . .	47
4.1.4. Hybrid Recommendation . . . . .	48
4.2. Supervised Individual Recommender . . . . .	49
4.2.1. Using Sparse Input . . . . .	49
4.2.2. Using Embeddings . . . . .	50
4.3. Unsupervised Group Recommender . . . . .	52
4.3.1. Baseline Recommender . . . . .	52
4.3.2. Diverse Recommender . . . . .	53
4.4. Supervised Group Recommender . . . . .	56
4.4.1. Baseline Recommender . . . . .	56
4.4.2. Diverse Recommender . . . . .	57
4.5. Feedback Loop . . . . .	58
4.5.1. Online Evaluation and User Study . . . . .	60
4.6. Summary . . . . .	61
<b>5. Discussion . . . . .</b>	<b>62</b>
5.1. Comparison of Individual Recommenders . . . . .	62
5.1.1. Offline Evaluation . . . . .	62
5.1.2. User Study . . . . .	63

## *Contents*

---

5.2. Comparison of Group Recommenders . . . . .	64
5.2.1. Offline Evaluation . . . . .	64
5.2.2. User Study . . . . .	65
5.3. Feedback Loop . . . . .	66
5.4. Other Topics to Discuss . . . . .	66
5.4.1. Problems about datasets . . . . .	66
5.4.2. Problem about the recommender type . . . . .	67
5.4.3. Reasons to use hybrid recommender . . . . .	68
5.5. Summary . . . . .	68
<b>6. Conclusion</b>	<b>69</b>
<b>A. Appendix</b>	<b>71</b>
<b>List of Figures</b>	<b>73</b>
<b>List of Tables</b>	<b>75</b>
<b>Bibliography</b>	<b>76</b>



# 1. Introduction

The introduction chapter of the thesis will focus on explaining the basic terminology of recommender systems. In addition to that, we will also present the motivation and the hypotheses that we want to validate.

"Recommender Systems 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 the 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.

Although the history of the recommender systems goes back to mid-1990s [Par+12], the real boom happened after e-commerce services became mainstream [SL17]. 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 the aim of reducing the information to be only relevant to users.

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. These recommendations will also increase users buying more items.

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

Another essential point is increasing user satisfaction. This function of recommenders is the foundation of the thesis at hand. Unfortunately, most of the researchers do not 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 increase user satisfaction.

The recommendation performance is mainly evaluated in terms of accuracy (i.e., the difference between actual and predicted rating). Concentrating on the accuracy might not be a desired goal of the talent/project recommendations (e.g., merely 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 accuracy, there might be other properties necessary for adequate recommendations (e.g., privacy, data security, diversity, serendipity, labeling, and presentation, and group recommendation).

Recommender systems can be applied to a various set of fields like entertainment, content creation, e-commerce, service sector, and social. However, we chose a specific use-case, which is the suggestion of talents to projects to be managed by the recruiters of the companies.

To be able to develop a solution on this topic, we used two datasets: one from the website *Freelancer.com* and another one from the company *Motius GmbH*. These datasets are presented in the section 3.1.

### 1.1. Motivation

This thesis aims to find answers to three main questions. The first question is if high accuracy guarantees high user satisfaction. The second one is an extension to the first question and asks if diversity affects user satisfaction positively. The last question asks if a feedback loop increases user satisfaction eventually.

### 1.2. Contribution

To answer the questions that we asked in section 1.1, we first developed models and algorithms that generate predictions. Then, we implemented these models and algorithms into a dashboard. Lastly, we executed offline, online evaluation rounds, and user studies to validate our hypotheses that were formulated as questions. After performing the previous steps, we can finally answer the questions that we asked in section 1.1.

First of all, we saw that high accuracy does not always result in high user satisfaction. This case is because other properties also affect user satisfaction, and having a marginal increase in the accuracy may result in a reduction of other properties, that also may decrease user satisfaction. These properties may vary from setting to setting and use-case to use-case.

We found out that diversity is an essential factor for our case, which is recommending a group of talents to a project. Although accuracy and diversity are inversely propor-

tional, a small increase in diversity led to a slightly higher user satisfaction. However, a further marginal increase in diversity resulted in a decrease in user satisfaction.

Lastly, we validated if a feedback loop increases user satisfaction. Although the initial implementation and training with a feedback loop do not increase user satisfaction, the subjects of the user study saw a positive effect, after giving more and more feedback. After inserting more than a thousand feedback and retraining with the resulting data, users were more satisfied.

We conclude that user satisfaction in recommender systems can be increased if the developers take user feedback into account and not only focus on algorithmic accuracy.

## 2. Review of Literature and Research

This chapter of the thesis aims to create a theoretical background of the topics that were demonstrated later in this thesis.

### 2.1. Types of Recommender Systems

Item recommendation approaches can be divided into two main 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.

#### 2.1.1. Content-Based Filtering

Content-based recommender systems employ features of both items and users to build item and user profiles that recommend items that are similar to the other items that the target user liked in the past. The necessary process of producing content-based recommendations consists of matching up the attributes of the target user profile, in which preferences and interests are stored, with the attributes of the items. After this process, we acquire a relevancy score that is an indicator for user-item relevancy. Generally, attributes that describe an item are features that are extracted from the item description. The content extracted from metadata is often too short and not sufficient to correctly define the user interests, that is why textual features are also added [De +15].

The general principle of content-based methods is to recognize the common characteristics of items that have received a positive rating from a user and then suggest new items to this user that share these characteristics. Recommender systems based only on content generally suffer from the problems of limited content analysis and over-specialization [SM95]. Limited content analysis happens when the system has a limited amount of information on its users or the content of its items. This limitation means the content of an item is often insufficient to determine its quality. Over-specialization, on the other hand, is the process of recommending similar items to other items that the user rated positively. "Because of this, the system may fail to recommend items that are different but still interesting to the user" [DK11].

### Overview of Content-Based Recommender Systems

This section discloses an overview and the process of building a general content-based recommender system and the relevant techniques [De +15].

The high level architecture of a content-based recommender system is shown in figure 2.1. "The recommendation process is performed in three steps, each of which is handled by a separate component" [De +15]:

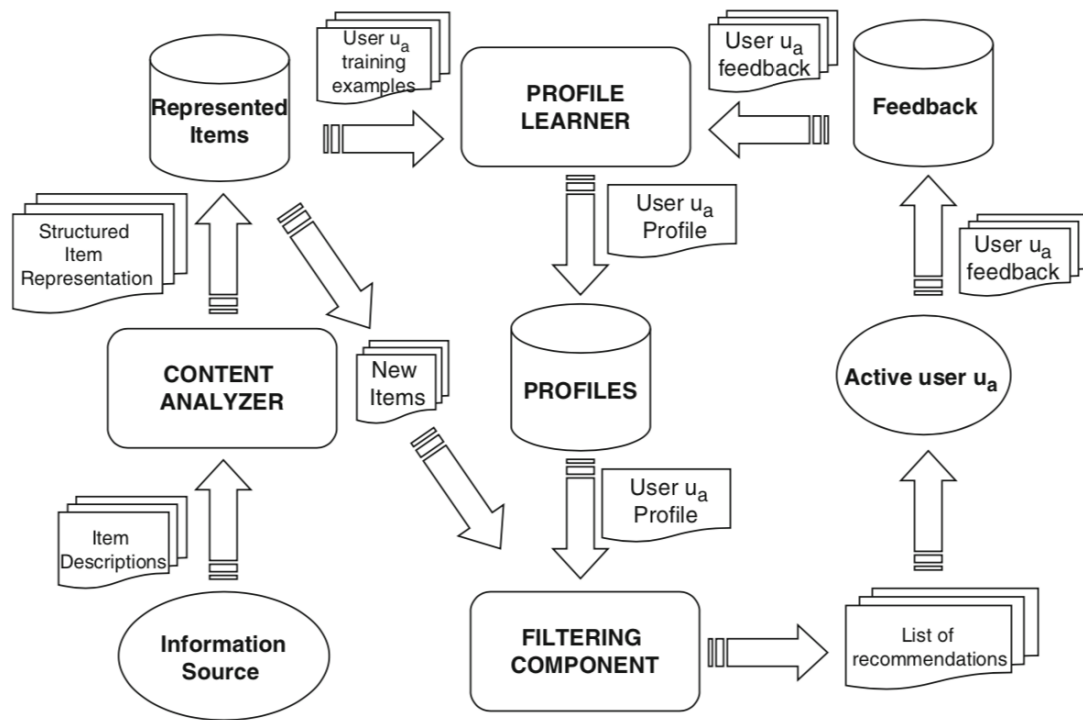


Figure 2.1.: High level architecture of a content based recommender [De +15].

- **CONTENT ANALYZER**—The first part of the architecture is responsible for scraping or getting the data from a source and saving them in a structured representation as items. If the data is a view data on a website, a scraper is needed to download and save them. However, if the desired input is metadata, then more advanced methods are needed to conduct an analysis. The output of this step is the input for the *profile learner*.
- **PROFILE LEARNER**—The profile learner module gets the learning data and trains the model or creates the user profiles, which generalizes from the learning

data. Using the generalization strategy, the profile learner creates a profile that improves on user's positive and negative interactions with the items. For the case of this thesis, we both create profiles for users and items(talents). The profile learner also takes advantage of enhancing the models with personal feedback after running on production mode.

- **FILTERING COMPONENT**—The last part of the architecture is the filtering component. It takes the user and item profiles, runs an operation, which generates a relevancy score for each user-item pair. Lastly, the component returns a recommendation list. Generating the relevancy score is a different process for different kind of models.

All of these components are crucial to make a functioning content-based recommender system. As the system always waits for new items, the feedback process is critical, and the retraining should happen regularly.

**Advantages and Drawbacks of Content-Based Filtering** The advantages of content-based filtering against collaborative filtering are listed below [De +15]:

- **USER INDEPENDENCE**—Content-based recommenders only need user profile for the relevant user. However, collaborative filtering recommenders require user profiles of all the other users so that it can suggest items based on nearest neighbors principle.
- **TRANSPARENCY**—Since the user profiles contain clear feature vectors, it is easy to directly understand why an item is recommended to a user by looking at their feature vectors. Conversely, collaborative systems are black boxes since the only explanation for an item recommendation is that unknown users with similar tastes rated that item positively.
- **NEW ITEM**—When a new item is added to the database, content-based recommendation systems can directly inject it to the system, because the feature vector of the new item is known. However, collaborative recommenders should wait until a substantial number of users rate this new item before the new item is recommended to anyone.

Nonetheless, content-based systems have drawbacks compared to collaborative filtering [De +15]:

- **LIMITED CONTENT ANALYSIS**—To run content-based recommenders, feature vectors for items and users are created. However, the process of limiting content

to feature vector loses information, or some misunderstandings can happen. For example, a popular machine learning library has two names: *scikit-learn* and *sklearn* or a popular frontend library is sometimes written as *Vue* or *Vue.js* or another popular programming language is called *Python*, so is an animal and a TV show *Monty Python*. Therefore, it is sometimes hard to create a *perfect* feature vector of content.

- **OVER-SPECIALIZATION** — Since the content-based recommenders suggest items based on past behavior, they recommend items that are expected by the user. This drawback is also called lack of serendipity problem, which highlights the tendency of the content-based systems to produce recommendations with a limited degree of novelty. If a recruiter has only engaged with people who know Python, the recommender will also suggest people who know Python. A *perfect* content-based technique would rarely find anything novel, limiting the range of applications for which it would be useful.
- **NEW USER**—Although it is easy to add new items to content-based recommender systems, it is not as easy to generate meaningful predictions for a new user. New users need to rate many items before we can recommend some items to them. This requirement means that the users need some history first so that the content-based recommenders work.

### 2.1.2. Collaborative Filtering

Rather than looking for content information, collaborative filtering approaches use the rating information of other users and items in the system. The main idea is that the rating of a target user for a new item is similar to that of another user. Similarly, the target user probably evaluates two elements in a similar way if other users have given similar ratings to these two elements. Collaborative approaches overcome some of the weaknesses of content-based methods. With collaborative filtering, the recommender system can still suggest items to users, even though the system does not know about the new user or item. Furthermore, collaborative recommendations are based on the ratings of items as evaluated by peers instead of relying on content. "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" [DK11].

Collaborative filtering recommender system methods produce user-specific recommendations of items based on patterns of ratings or hiring data without the need for content information about either items or users [DK11].

Collaborative filtering recommenders need interactions between users and items. Then, these recommenders can establish recommendations with two different approaches: the neighborhood approach and latent factor models. Neighborhood methods focus on relationships between items and users and can function by checking the similarity between users or items. Latent factor models, such as matrix factorization transfer the user-item matrix into a different space. By doing this operation, it is possible to fill the gaps in the matrix, which correspond to the predictions.

Since we have not used collaborative filtering in the implementation part of this thesis, we keep this section short. Readers who are interested in this topic are advised to check the resources [KB15] and [WWY15].

### **2.1.3. Knowledge-Based Recommender Systems**

Knowledge-based recommenders are the third most popular type of recommender systems [Bur00]. In this type of recommender, users are explicitly asked to enter criteria for what kind of items they want to see. This information is compared with the items, and a recommendation list is returned. More information can be found in the resources [Bur00] and [Fel+15].

### **2.1.4. Hybrid Recommender Systems**

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" [DK11], or by adding content information into a collaborative filtering model. "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" [DK11], which is called the cold-start problem.

## **2.2. Types of input in recommender systems**

Recommender systems can receive different types of input. The straightforward one would be the explicit feedback that represents the actual ratings that users give to the items. This rating could be a star rating or a thumbs up/down action. The second is implicit feedback, which is harder to collect. It represents clicks, browsing history, mouse movements, and other ways of input that the users do not send explicitly.



### 2.3. Recommender System Evaluation Properties

When recommender systems are evaluated, the relevant metrics should be chosen according to the needs. Some of the properties consist of trade-offs, accuracy declines when the diversity and other properties increase, or other properties like diversity and novelty can be directly proportional. The developers of the recommender systems should evaluate important properties using offline and online evaluation [SG11].

Although offline evaluation may be enough for properties like accuracy, user studies and online evaluation are required to draw reliable conclusions on properties like diversity. Such an online experiment/user study uses a recommendation method with a tunable parameter that affects the property being considered. Test subjects should be presented with the list of recommendations that are affected by diverse values for tunable parameters, whether the user noticed the change in the property should not be measured, but whether the change in property has increased their satisfaction. Like other user studies, it is advantageous when the participants do not know about the aim of the experiment. We can only measure properties like diversity with user study or online evaluation because we need user response that gets affected by this parameter [SG11].

When the developers conduct experiments to select the relevant properties and evaluate their performance, the most suitable recommenders can be selected. According to [SG11], there are several related properties outlined in the following.

**Accuracy** In this section, some equations to calculate the accuracy are presented. To understand these equations better, we briefly explain the notation.

To understand the equations, we define some notations; the set of users in the recommender system will be denoted by  $U$  and the set of items by  $I$ . Additionally,  $R$  depicts the set of ratings recorded in the system, and the letter  $S$  is for the set of possible values for a rating.  $S$  can be a discrete number from 1 to 5 or an element of the set like, dislike or other values depending on the implementation. Also, we suppose that a user  $u \in U$  can perform no more than one rating for a particular item  $i \in I$  and set  $r_{ui}$  this rating. To identify the subset of users that have rated item  $i$ , we use the notation  $U_i$ . Similar to that,  $I_u$  represents the subset of items that have been rated by a user  $u$ .

Recommender systems have two main tasks, which are predicting the ratings and recommending a list to the users. Prediction of ratings is self-explanatory, and the aim is predicting the ratings of an unrated item  $i$  from the user  $u$ . This task is defined as a regression or a classification problem to learn the function  $f : U \times I \rightarrow S$ . There are two popular measures of accuracy to for this task: Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE). These are presented below [See (2.1) and (2.2)]

respectively]

$$\text{MAE}(f) = \frac{1}{|\mathcal{R}_{test}|} \sum_{r_{ui} \in \mathcal{R}_{test}} |f(u, i) - r_{ui}|, \quad (2.1)$$

$$\text{RMSE}(f) = \sqrt{\frac{1}{|\mathcal{R}_{test}|} \sum_{r_{ui} \in \mathcal{R}_{test}} (f(u, i) - r_{ui})^2}. \quad (2.2)$$

The author of this thesis used them as the cost function of the neural networks [See section 3.3].

The other task that was mentioned is presenting a top-n recommendation list. This list that is shown to a user is represented with  $L(u)$ . We define  $T(u)$  as the subset of test items that the user  $u$  found relevant. The performance of the method is then calculated using the measures of precision and recall, which are defined as

$$\text{Precision}(L) = \frac{1}{|u|} \sum_{u \in \mathcal{U}} |L(u) \cap T(u)| / |L(u)|, \quad (2.3)$$

$$\text{Recall}(L) = \frac{1}{|u|} \sum_{u \in \mathcal{U}} |L(u) \cap T(u)| / |T(u)|. \quad (2.4)$$

A drawback of the previous methods is that all items of a recommendation list  $L(u)$  are considered equally interesting to user  $u$ . An alternative setting would be calculating the success of average hit ranks with the method Average Reciprocal Hit-Rank (ARHR), which is defined as

$$\text{ARHR}(L) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{\text{rank}(i_u, L(u))}, \quad (2.5)$$

which is used to evaluate the success of the company recommender [See section 4.1.1].

**Coverage** Coverage can also be a vital evaluation property for some recommender systems. The aim of maximizing the coverage would be making sure that the recommender system recommends a significant portion of items. Coverage might be especially crucial for recommender systems of e-commerce systems so that different range of products get sold.

**Confidence** Confidence in the context of recommender systems refers to the system's trust into its suggestions. As we do not use this property in the implementation part, we do not go into detail [HKR00].

**Trust** Opposite to the confidence that was explained in subsection 2.3, trust refers to user's trust into the recommender system. More information on this can be found in other sources.

**Novelty** Novel recommendations are recommendations for the items that the user did not know about before. A simple solution to increase this would be filtering out the items for every user, which they interacted before.

**Serendipity** Serendipity also is known as unexpectedness, depicts how surprising the recommendation results are. Although serendipity and novelty may sound similar in the beginning, items that are similar to what the user already saw would not be surprising at all. That is why filtering out the items that the user already saw would not be a solution.

**Diversity** Diversity is a property that ensures the recommended items are not similar to each other. As this is the primary metric that we focus on, there is a special section just for this topic [See section 2.4].

**Utility** Utility as a property hints the optimization of predefined property that brings any advantage to the recommendation system. For example, the utility function of an e-commerce website may be improving revenue. Generally, the utility touches on any system or user gain with the help of the recommender system.

**Risk** Risk is another property that is important for some specific use-cases. For example, for a recommender system that suggests stocks to buy, then the risk would play a significant role.

**Robustness** Robustness in recommender systems explains how secure the system is to unwanted modifications on the system. For example, If a hotel owner can inject fake positive reviews to the system to make their hotel get recommended, then the system is not robust enough.

**Privacy** For recommender systems that want to keep the preferences of users secret from others, privacy is an important metric, so that they can make sure that everyone can interact with the system and get suggestions without worrying about their feedback being public.

**Adaptivity** Adaptivity in recommender systems tries to give relevant recommendation even if the trends change. For example, for a news recommender system, the system should be able to recommend news about the earthquake, in case of one happens, even though the user does not have a previous interest in earthquakes.

**Scalability** It may also be necessary for recommender systems to be performant. For production systems with millions of items, the recommender should still be able to suggest new items without any latency.

### 2.3.1. Summary

In this section, we discussed how recommendation algorithms could be evaluated in order to increase user satisfaction. There are different properties for different needs, and the developers should understand what they need.

As this thesis focuses mostly on diversity, the next section gives detailed information about diversity and its evaluation techniques.

## 2.4. Diversity

In the first decades of recommender systems, the main concern was running predictions with a high accuracy rate. Since the beginning of 2000s other properties like utility, novelty, diversity are also perceived as essential metrics in addition to the accuracy [HZ11]. In this section, we explain the motivation and techniques that are about diversity.

### Why Diversity in Recommendation

Adding diversity as a target property of the desired outcome brings the recommendation problem to a broader perspective instead of only focusing on accuracy [MRK06]. There are also other properties that should be considered according to the needs, which are described in the previous section.

Recommender systems get the information of user clicks, reviews, purchases that are driven by the user interest and try to have a perfect guess for the users. However, user interests are complex, dynamic, context-dependent, heterogeneous, and contradictory. That is why the prediction of user needs just by looking at accuracy is a difficult task and may lead to decreased user satisfaction. Diversity can be an excellent strategy to optimize the chances that at least some item pleases the user, by widening the range of different item types and characteristics, rather than suggesting in a too narrow and risky range [CHV15].

On the other hand, from the user perspective, diversity is generally desirable, as it increases user satisfaction. According to consumer behaviorists, humans seek variety. They get satisfied from novelty, unexpectedness, change, and also humans have a genuine desire for the unfamiliar [CHV15]. Not employing diversity may also lead to too much bias, which causes a filter bubble. A filter bubble is a terminology that explains the state that the person only receives personalized recommendations, which leads to seeing similar content all the time. This situation means that there is a healthy level of diversity required [Ngu+14]. The increase in user satisfaction also leads to increased activity, revenues, and customer loyalty.

In the context of this thesis, the main task is recommending talents to roles of the projects. If the system suggests people with very similar or overlapping skills, it would not be too beneficial for the recruiters, since they would like to have a team that can tackle a wide range of topics.

### 2.4.1. Diversity Evaluation

In this subsection, we show different options to evaluate the diversity of recommender systems. We first present the notation and go on with the methods.

Similar to the notation that we used before,  $i$  and  $j$  denote items,  $u$  and  $v$  are used for users, and  $I$  and  $U$  symbolize the set of all items and users.  $I_u$  and  $U_i$  are shorthand symbols for items that the user  $u$  has interacted with and users that have interacted with the item  $i$  respectively.  $r_{u,i}$  is the set of ratings for the ratings from user  $u$  to the item  $i$ . The letter  $R$  is used to indicate the recommendations to the target user  $u$  [CHV15].

#### Average Intra-List Distance

Perhaps the most frequently used diversity metric is the average intra-list distance [CHV15]

$$ILD = \frac{1}{|R|(|R| - 1)} \sum_{i \in R} \sum_{j \in R} d(i, j). \quad (2.6)$$

As the name suggests, it has the aim to calculate the average diversity inside a list. A distance function is picked, and the distance of each item in the list to other items in the same list is calculated. The results give us diversity. This method is also employed in the evaluation part of the implementation, and cosine similarity is picked for the distance function. The similarity of the list items can be easily calculated with the formula *Inter – List – Similarity* = 1 – ILD [See chapter 4].

### User-Specific Unexpectedness

Unexpectedness is a property that can lead to user satisfaction. The formula below is the way to calculate it for a user

$$\text{Unexp} = \frac{1}{|R| |\mathcal{J}_u|} \sum_{i \in R} \sum_{j \in \mathcal{J}_u} d(i, j), \quad (2.7)$$

where

$$\mathcal{J}_u \stackrel{\text{def}}{=} \{i \in \mathcal{J} | r(u, i) \neq \emptyset\}. \quad (2.8)$$

In words, the distance between recommended items and the items that the user has already interacted with are averaged. This method is also used in the evaluation part of the thesis [See chapter 4]. Another way to calculate the unexpectedness is with this formula

$$\text{Unexp} = |R - \text{EX}| / |R|. \quad (2.9)$$

In the above formula, EX is the set of items that the user expects [CHV15].

### Inter-Recommendation Diversity Metrics

In the following paragraphs, we show methods to calculate the diversity of the whole system. These results can be used to compare diversity between recommendation methods. Although these methods are used in the implementation chapter of this thesis [See chapter 3], they are not presented in the evaluation chapter [See chapter 4] because of space reasons.

**Aggregate diversity** We can calculate the aggregate diversity for all users in the system to compare the results with the other recommender systems. This way, we compare the diversity of different systems [CHV15]. The aggregate diversity is defined as

$$\text{Aggdiv} = \left| \bigcup_{u \in U} R_u \right|. \quad (2.10)$$

In the equation (2.10),  $\bigcup$  is the union sign. Therefore, we calculate the set of recommended items for each user and return the size of it.

**Gini Index** Calculating the Gini index means that the interacted items are first sorted in ascending order by the chosen probability, which is the equation (2.12) below. The equations to calculate the gini coefficient are

$$\text{Gini} = \frac{1}{|J| - 1} \sum_{k=1}^{|J|} (2k - |J| - 1) p(i_k | s), \quad (2.11)$$

where

$$p(i_k | s) = \frac{|\{u \in U | i \in R_u\}|}{\sum_{j \in J} |\{u \in U | j \in R_u\}|}. \quad (2.12)$$

In that equation,  $k$  is the number of the least recommended item. In the end, a Gini index from the equation (2.12) close to zero means items are chosen equally often, and a value of one would mean a single item is always chosen [CHV15].

**Shannon Entropy** Shannon Entropy is logically similar to the *gini index* and they are calculated similarly. The equation to calculate the Shannon Entropy reads

$$H = - \sum_{i \in \mathcal{J}} p(i | s) \log_2 p(i | s). \quad (2.13)$$

In the equation (2.13), if the entropy is zero, a single item was always chosen, and if the entropy is  $\log_2(\#items)$ , then each item is equally chosen [CHV15].

#### 2.4.2. Diversity Enhancement Approaches

There are also some methods not just to evaluate but also increase diversity. The methods that we employed are reranking and clustering. These methods are explained in chapter 3.

### 2.5. Artificial Neural Networks

Neural Networks are frameworks to process complex data inputs, and the version of neural networks that are used in this thesis are feed-forward neural networks. These type of networks have input, hidden and output layers, and useful for learning non-linear patterns with the help of various activation functions [See figure 2.2 ].

The goal of a feedforward network is to approximate some function  $f^*$ . For example, for a classifier,  $y = f^*(x)$  maps an input  $x$  to a category  $y$ . A feedforward network defines a mapping  $y = f(x; \theta)$  and learns the value of the parameters  $\theta$  that result in

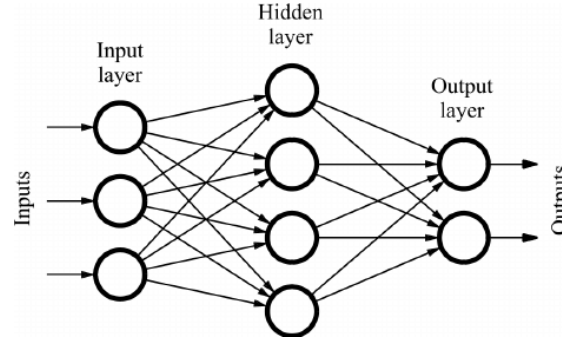


Figure 2.2.: High level architecture of a feedforward neural network [QD11].

the best function approximation" [GBC16]. For a linear model,  $\theta$  would consist of  $w$  and  $b$ . The equation for one layer would be

$$f(x; w, b) = \sigma(x^\top w + b). \quad (2.14)$$

In the equation (2.14),  $\sigma$  refers to the activation function,  $w$  represents the weighting matrix that is learned and  $b$  is some bias. These layers can be connected to form the deep neural network, where  $x$  denotes the output of the previous layer. Finally, a loss function, that measures the deviation between the predicted label and the true label is minimized by the network.

Neural networks are already used to recommend videos on YouTube [CAS16], recommend movies [CVS07] and recommend Android applications on Google Play store [Che+16]. The advantage of neural networks compared to other supervised learning methods is the fact that they can learn and generalize from labeled data without much of adjustments [MW+14].

### 2.5.1. Embeddings

Embeddings can be used to reduce dimensionality with neural networks. Embeddings layer of a network only accepts the indices of the desired skills and outputs a much smaller dimension. This technique is also used in this thesis [GG16].

## 2.6. Summary

In the scope of this chapter, we gave detailed information about the types of recommender systems with a focus on the content-based recommendation. Then, we listed the evaluation properties that can lead to user satisfaction. Next, we disclosed the



## *2. Review of Literature and Research*

---

property that we chose, which is the diversity and illustrated how diversity could be calculated. Lastly, a short introduction to the neural networks is given.

## 3. Implementation

This chapter explains different solutions for the problem at hand. These solutions include applications using datasets from the website Freelancer.com and the company Motius running different methods.

As we mentioned in the Introduction part [See section 1.1], 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.

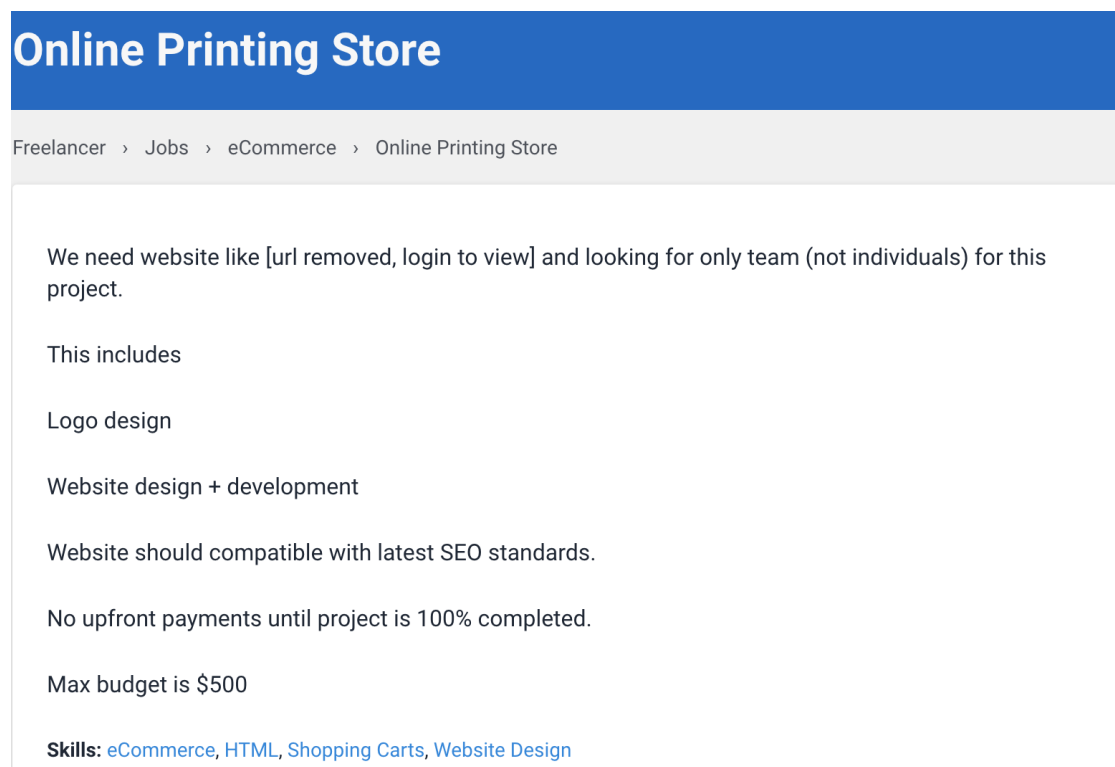
### 3.1. Datasets

The author of the thesis received two separate but similar datasets. Both of the datasets contain information about projects and talents. The company dataset [See subsection 3.1.2] is the internal database of the company Motius and includes skill vectors of 795 talents and 375 roles. These roles are parts of bigger projects, which is not the case in Freelancer.com dataset. In Freelancer.com dataset, we have projects as opposed to positions in the Motius dataset. However, we treat the projects in the Freelancer.com data and the roles in the Motius 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.com dataset is much bigger and detailed compared to the Motius dataset. The Freelancer.com dataset contains 30606 roles that are comparable to the positions in the Motius 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.com data carries approximately 14-15 applicants per projects, and only one of the applicants get selected as the person to implement the project. Differently, Motius 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.com Dataset



The screenshot shows a project listing on the Freelancer.com website. At the top, there is a blue header with the text '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.' followed by 'This includes' and a list of requirements: 'Logo design', 'Website design + development', 'Website should compatible with latest SEO standards.', 'No upfront payments until project is 100% completed.', and 'Max budget is \$500'. At the bottom, there is a section labeled 'Skills:' followed by the tags 'eCommerce, HTML, Shopping Carts, Website Design'.

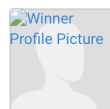
Figure 3.1.: An example project from the Freelancer.com Website

As it can be seen in figure 3.1, a typical Freelancer.com project posting consists of a title, the description, and the relevant skills. For simplicity, the thesis at hand only concentrates on the skills and does not take the project description into account. Including other information would be the topic of another paper/thesis, as it would require natural language processing and other techniques [BKL09].

### 3. Implementation

---

🏆 Awarded to:



**Winner Talent** 🇧🇪

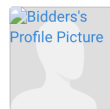
**\$444 USD** in 10 days

4.7 ★★★★★ (8 Reviews)  
3.3 \$ ██████████

---

**14 freelancers are bidding on average \$550 for this job**

---



**Bidder 1** 🇮🇳

**\$515 USD** in 10 days

NO UPFRONT PAYMENT REQUIRED!  
We've been through the provided reference website "[login to view URL]" and could able to understand its functionalities and what they are offering. We shall definitely develop [ [More](#) ]

4.5 ★★★★★ (205 Reviews)  
8.2 \$ ██████████



**Bidder 2** 🇮🇳

**\$500 USD** in 15 days

Dear Sir, We can create an online printing store for you like [login to view

4.7 ★★★★★ (138 Reviews)  
7.8 \$ ██████████

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

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, 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 do not 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 is why the amounts can range from one to more than hundreds.

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.com web page

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 were not used in notable amounts, and the data can be expressed without using those skills. This dataset was scraped from Freelancer.com.

### 3.1.2. Motius Dataset

The company dataset at hand is acquired from Motius GmbH. This dataset exported from their internal database and contains information about 795 talents and 375 roles. Each role includes the required skills for them, and each person has their skills listed. One difference to the Freelancer.com 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 Motius 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.com and Motius have used different names for skills, there are exists only a set of 59 common skills. Therefore, training both datasets together does not improve model like it is expected.

When all of Freelancer.com and Motius 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 may create a big problem for the developers. When an operation like normalization is being done, the library *Pandas* applies many copying operations, which doubles the memory usage and may crash, if the physical memory is below 32 GB. That is why the author employed embeddings as a dimensionality reduction mechanism [See 3.3.2].

## 3.2. Unsupervised Individual Recommender

This section demonstrates how unsupervised individual recommenders got implemented.

### 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. The similarity measure we use for this part of the thesis is the cosine similarity [Ama+11], which reads

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

In this formula for the cosine similarity, the big black dot denotes the inner product, and the double pipes are the Euclidian standard vector norm. 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  $n$ -dimensional space, and the formula calculates the similarity as the cosine of the angle between two vectors. The equation 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 talents 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 an algorithm that is easy to implement, but it is also not easy to come up with new algorithms that perform better than the popularity recommender [AB15]. The logic comes from the fact that the *items* that are in demand approved by many *talents*. That is why it is likely that selecting these items will increase user satisfaction [AB15].

For this specific project, the popular items that are in demand are the talents that finished the maximum amount of projects at Freelancer.com or Motius. Although the results will not 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

### 3. Implementation

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[See section 2.5.1]. Both of the methods employ neural networks[See section 2.5].

#### 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 x 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.

It is crucial to normalize input data before training neural networks [SS97]. 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 of the data points. The equation reads

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

As it was mentioned before [See section 3.1], the Freelancer.com dataset contains 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 does not exist in Motius dataset, this subsection focuses only on the implementation with Freelancer.com dataset. The extra information that is mentioned is also scaled and input into the neural network.



### 3. Implementation

---

After normalizing data, we prepare the matrix that is fed into the neural network row by row. For each bid in the Freelancer.com 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 did not 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 the 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, in every case [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 serves

to prevent overfitting. In our case, it compares the validation loss of current batch with the previous one. If the validation loss does not 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 weights with *he normal* [HR18].

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. 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. The weight initializer of the last layer is *glorot uniform* [Ped18].

Each neural network has the aim of minimizing its cost function [GBC16]. The cost function that we chose is *mean squared error*[See (3.3)]. In the equation.  $C$  is the cost function,  $n$  is the total number of data points,  $y(x)$  are the predicted results, and  $a$  is the correct result. 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. Mean squared error is defined as

### 3. Implementation

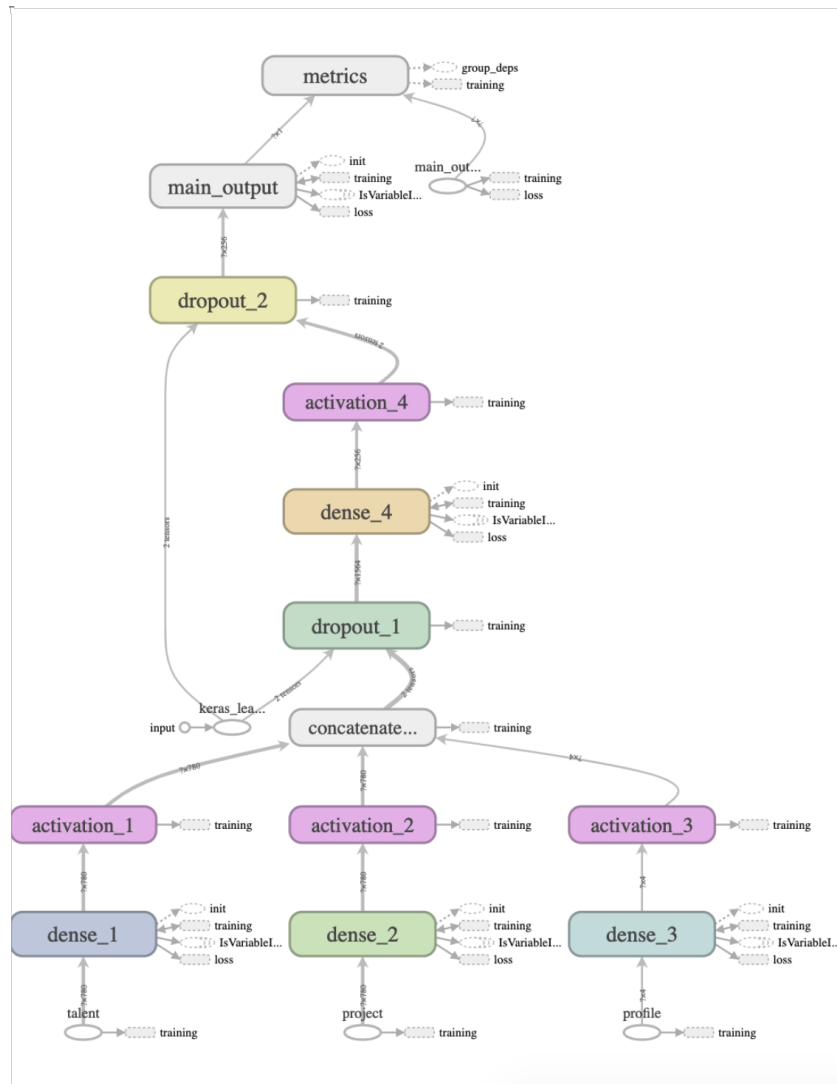


Figure 3.6.: The graph that explains the sparse input model

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

### 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.com 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 zeros so that we can reduce the dimensionality [See section 2.5.1].

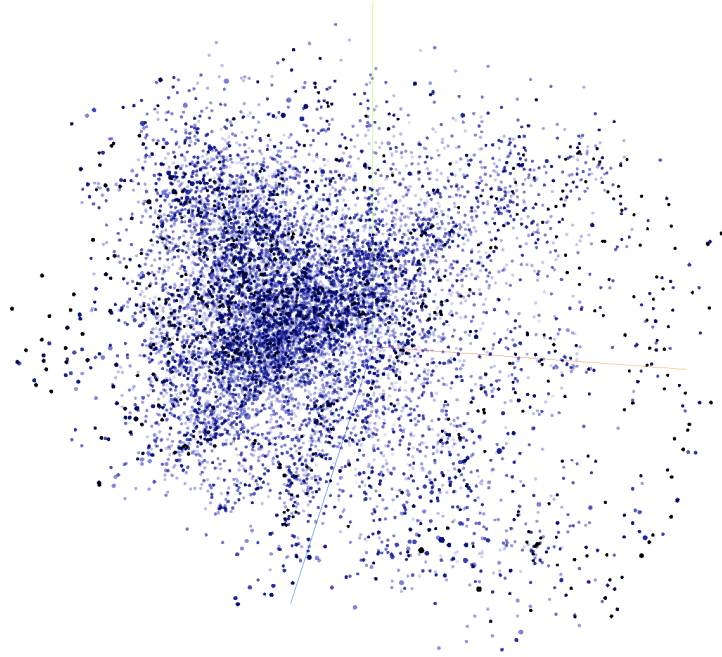


Figure 3.7.: 3D version of the embedding space that is created for projects of this thesis

#### Preprocessing

For both Freelancer.com 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 equal to the limit are positive, and the remaining entries are set to zero. In this way, 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 Freelancer.com 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 is 2, and the smallest is 0.

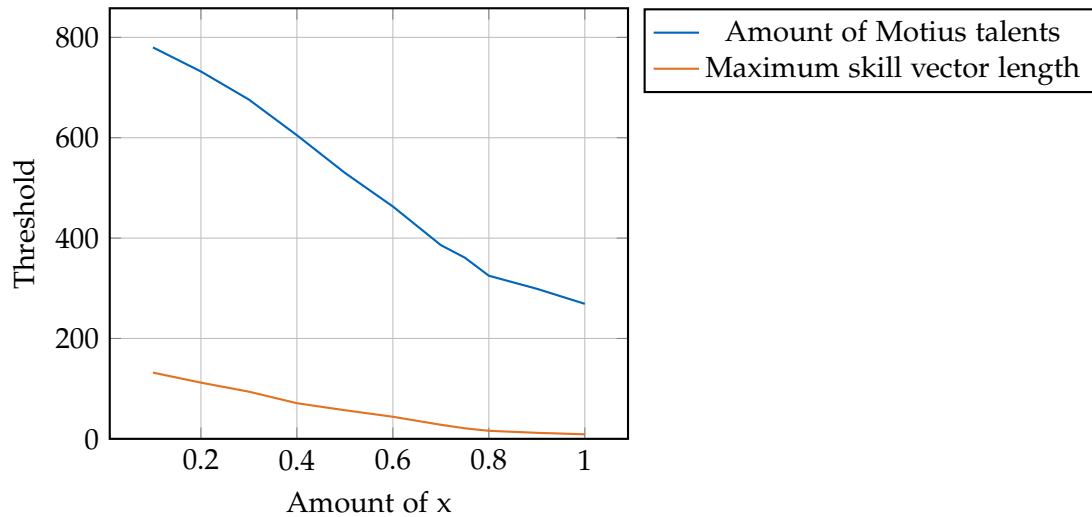


Figure 3.8.: Effect of threshold selection on talents and maximum skill vector length

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 do not specify skill levels, so these are taken as they are. That is 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 would not want to implement this version

### 3. Implementation

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 would significantly decrease because the Freelancer.com data also has a maximum length of 18. Therefore, the optimum threshold value we reached is 0.75. As a consequence, 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.com data combined. In the variant with only Freelancer.com 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.com dataset has a massive 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 A.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 [Mur12]. Lastly, *sigmoid* activation function squeezes the output value to be between 0 and 1 [Ped18].

## 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.

In order to perform group recommendations, project-role, or project-project information are needed. The website 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.com data. In terms of simplicity and shortness, we only take Freelancer.com dataset with groups of size five into account.

### 3.4.1. Baseline Recommender

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 will not be diverse, and we can pick talents that have similar skills to each other. We want to avoid that [See section 4.3.1] and have diverse recommendations for each project.

### 3.4.2. Diverse Recommender

Because of the reasons above, we want to create the recommendation list diversely from the beginning. The topic of diversity is already explained before [See section 2.4] and the pseudocode to enhance diversity is shown below.

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 has not 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 (3.4). 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

---

**Algorithm 1** Diversity Enhancement Algorithm

---

```

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

```

---

gives the balanced result.  $f_{rel}$  in the equation stands for the relevancy score of the item, and  $div(R)$  tells the diversity between the items in the list. When we receive the optimal candidate from the equation, we add the talent to the recommendation list and iterate until we have enough talents for the whole group. The equation for the diversity is

$$g(R, \lambda) = (1 - \lambda) \frac{1}{|R|} \sum_{i \in R} f_{rel}(i) + \lambda div(R). \quad (3.4)$$

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 [See section 4.3].

### 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 3.4 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 idea was also the first aim for supervised group recommender approach. However, the data for such knowledge does not exist. To apply it, we would need data on hiring decisions for the groups, not just projects. Since we do not have such information, we would have to generate it with a separate algorithm. In the end, it would not bring much, because the model would learn the data generation algorithm and would not have an effect on the real-life hiring prediction.



Due to the reason above, step by step learning process is applied. 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 6 is applied. The supervised learning only optimizes the relevancy score, and the diversity still gets calculated via the unsupervised approach.

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], 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. Here, it makes sense to note that a higher number will affect the clustering model negatively, and a lower number will not be able to capture everything in the dataset. The author of the thesis experimented with different values.

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 3.10 shows silhouette scores for different cluster amounts. Silhouette score calculates the similarity of a data point to its cluster compared to other clusters [Rou87]. For this task, we use the equations (3.5), (3.6), (3.7). In the equations, different distance metrics can be employed. The choice of the author is the Euclidian distance. The equation (3.5) calculates the mean intra-cluster distance for each sample and the next (3.6) 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.  $d(i, j)$  in the (3.5) stands for the chosen distance function and the  $C_i$  refers to the number of clusters. The last function (3.7)

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 3.10, it makes sense to select a value like 30. To visualize results, we project the centers of the clusters on a 2D space[See 3.11]. 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. The relevant equations are defined to be

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j), \quad (3.5)$$

$$b(i) = \min_{i \neq j} \frac{1}{|C_j|} \sum_{j \in C_j} d(i, j), \quad (3.6)$$

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}. \quad (3.7)$$

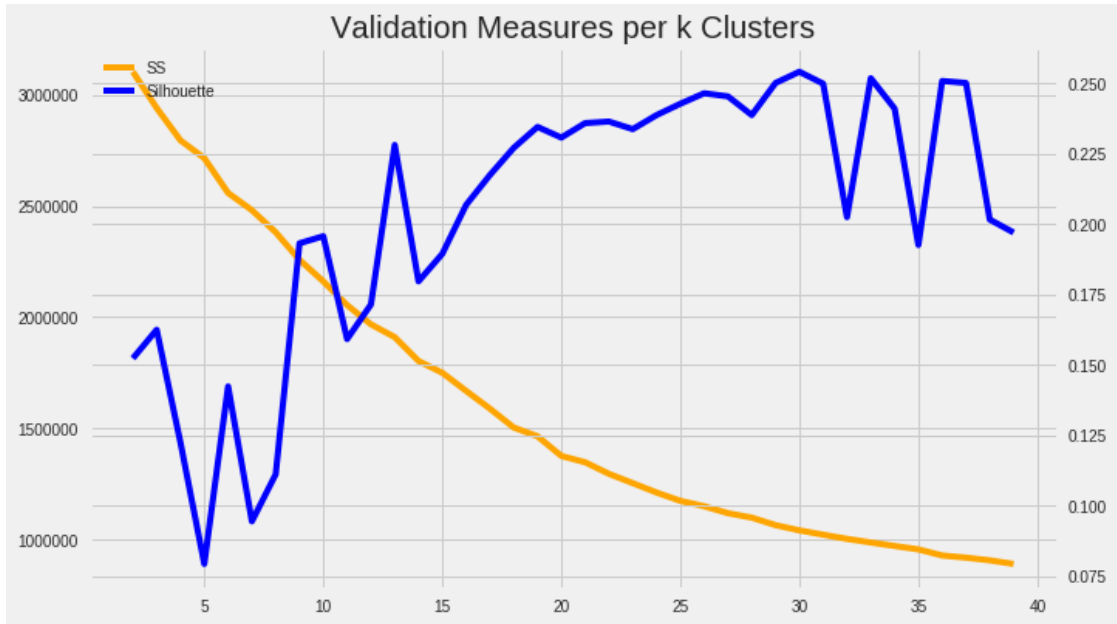


Figure 3.10.: Silhouette scores of many different  $k$  values of k-means

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 3.12]. 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 that these values are calculated after standard scaling [Gru19].

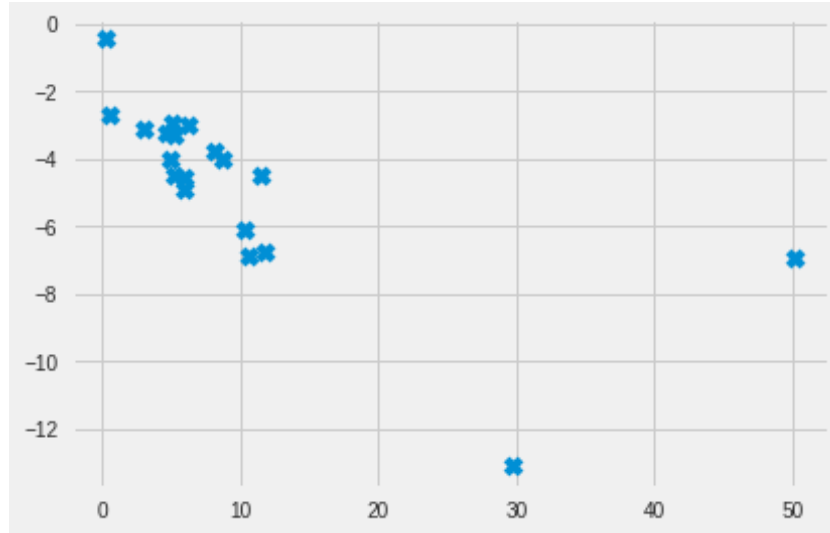


Figure 3.11.: Centers of clusters that are projected on a 2D space

After we execute clustering, we can start with the group recommendation process. 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 3.12]. This way, we determine the ideal cluster for the project. When a project got recommended 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.

## 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 recom-

### 3. Implementation

	.NET	2D Animation	360- degree video	3D Animation	3D Design	3D Model Maker	3D Modelling	3D Printing	3D Rendering	3ds 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 3.12.: Examples of some centers of clusters that are projected on a 2D space

mendations for Motius and Freelancer.com 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*. *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 personal reasons; it is a reactive, modern framework that is easy to develop [You18]. On the other hand, *Flask* is chosen because it is a popular lightweight 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 is 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 [And15].

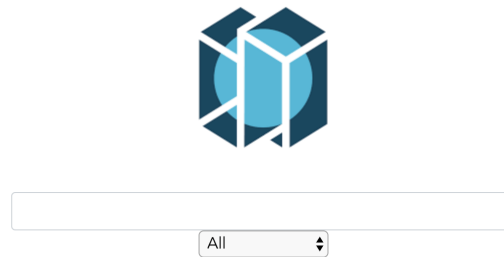


Figure 3.13.: Main screen of the dashboard

The main screen of the dashboard is shown in figure 3.13, and it contains a search

box and a dropdown. When users type anything on the search box, the front-end sends a *get request* [Mas11]. 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 related 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.com 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 3.14].

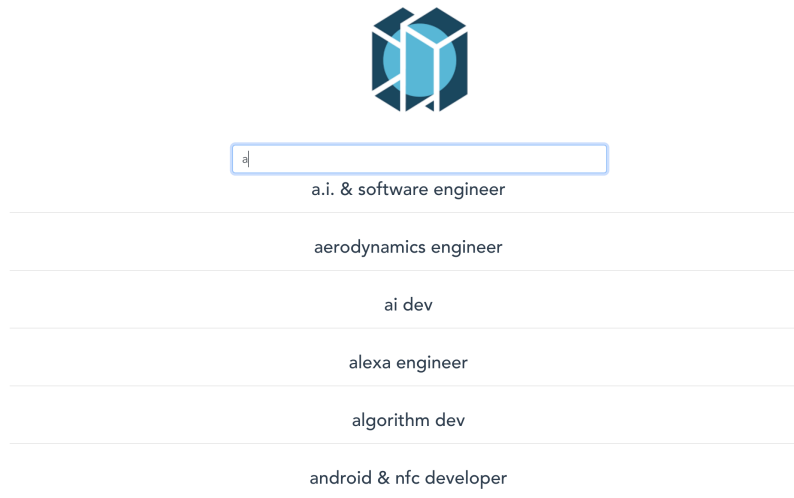


Figure 3.14.: A snippet from the list of all projects that start with the letter *a*

#### 3.7.2. Individual Recommendations

When users click on any of the projects that are listed in figure 3.14, 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 3.15 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 3.15 shows the results for the project *a.i. & software engineer* on the dashboard

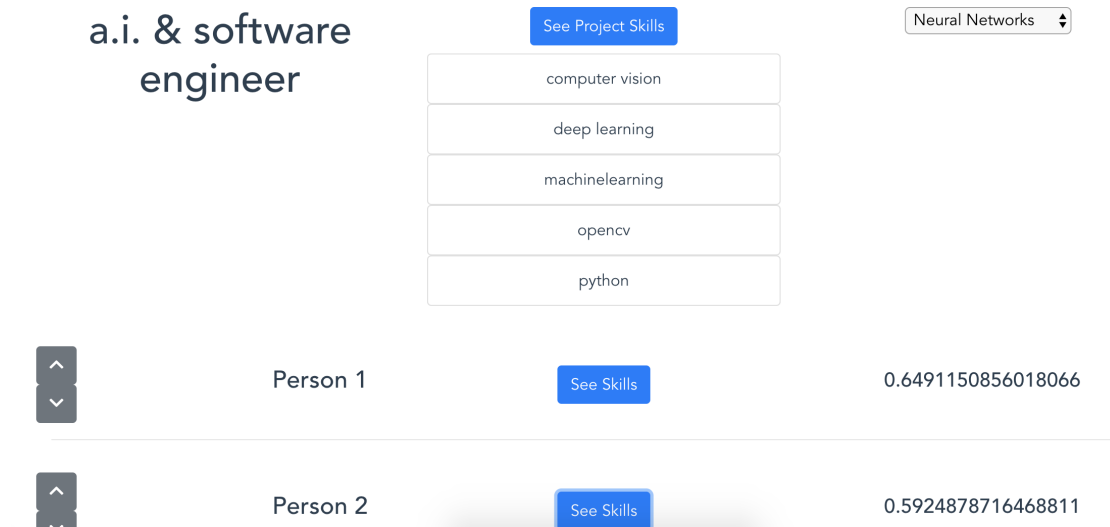


Figure 3.15.: A screenshot from the list of all recommendations from neural networks for the project *a.i. & software engineer*

that is built by the author. Names of the talents that were recommended are anonymized, but the rest of the data are untouched. 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 debug neural networks, and they are mostly referred to as *black-boxes* [See chapter 4] [BCR97].

In contrast to the figure 3.15, figure 3.16 demonstrates the adequate talents for the same project as before. However, talents 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 can also perform group recommendations. For the case of group recommenders, the scores can be from baseline neural networks, baseline cosine similarity, diverse neural networks, or diverse cosine similarity [See chapter 4].

Baseline cosine similarity, reveal the list of best talents for each project using skill

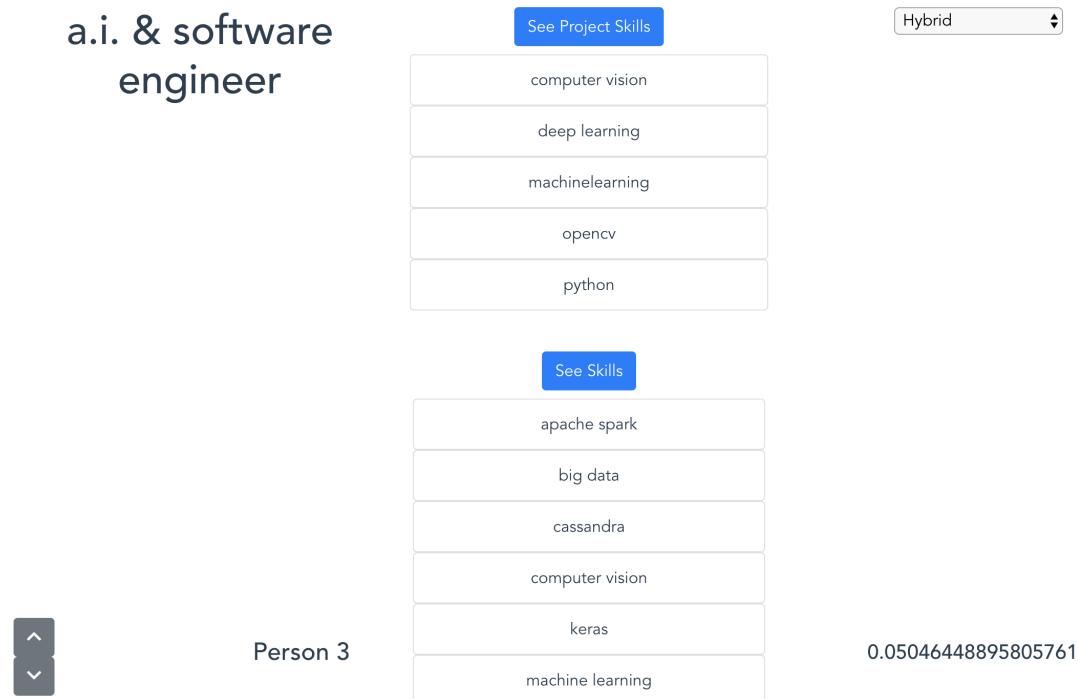


Figure 3.16.: A screenshot from the list of all recommendations from neural networks for the project *a.i. & software engineer*

vector similarity and baseline neural networks do the same with neural networks. Diverse cosine similarity and diverse neural networks show a list of talents for each project of a group, and these talents are sorted by their relevancy to projects and the diversity between other talents in the group.

Figure 3.17 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 are diverse to each other simultaneously. Between the group name and recommendation mode, a slider is located. This slider determines the constant for the diversity enhancement formula [See (3.4)]. 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].

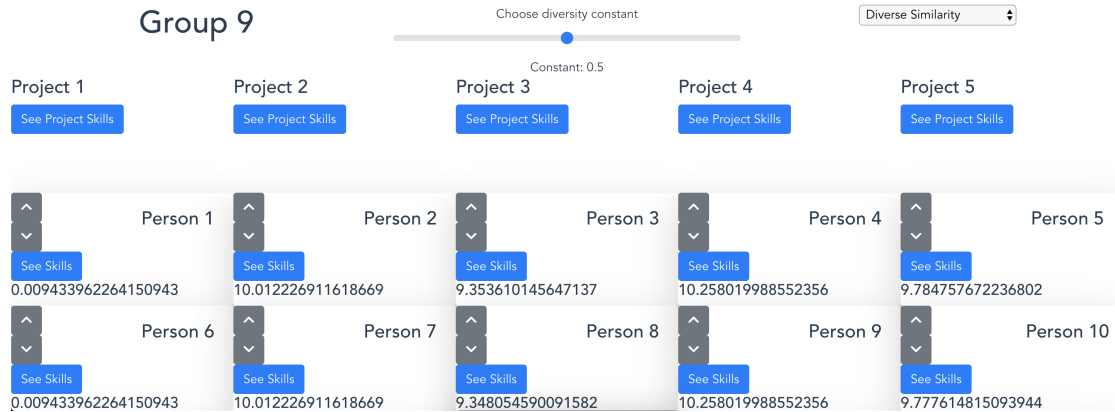


Figure 3.17.: A screenshot from the list of all recommendations from diverse cosine similarity for the group 9

### 3.8. Improvement of Recommendations via Feedback Learning

The arrows in the figures 3.17, 3.16 and 3.15 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 is provided by real talents may have direct or indirect effects. To understand the process, it may make sense to check the figure 3.18 first. Feedback data is given to the model as a separate input, and this input is followed by a layer with one node with *tanh* activation function. Tanh activation is chosen because the bias can be negative or positive [Ped18]. 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 A.2]. Due to this addition, the outcome of personal 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 personal 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 is 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 realization of this thesis exhaustively. We explained how we programmed different recommenders with diverse approaches. We also revealed the dashboard that puts 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.

### 3. Implementation

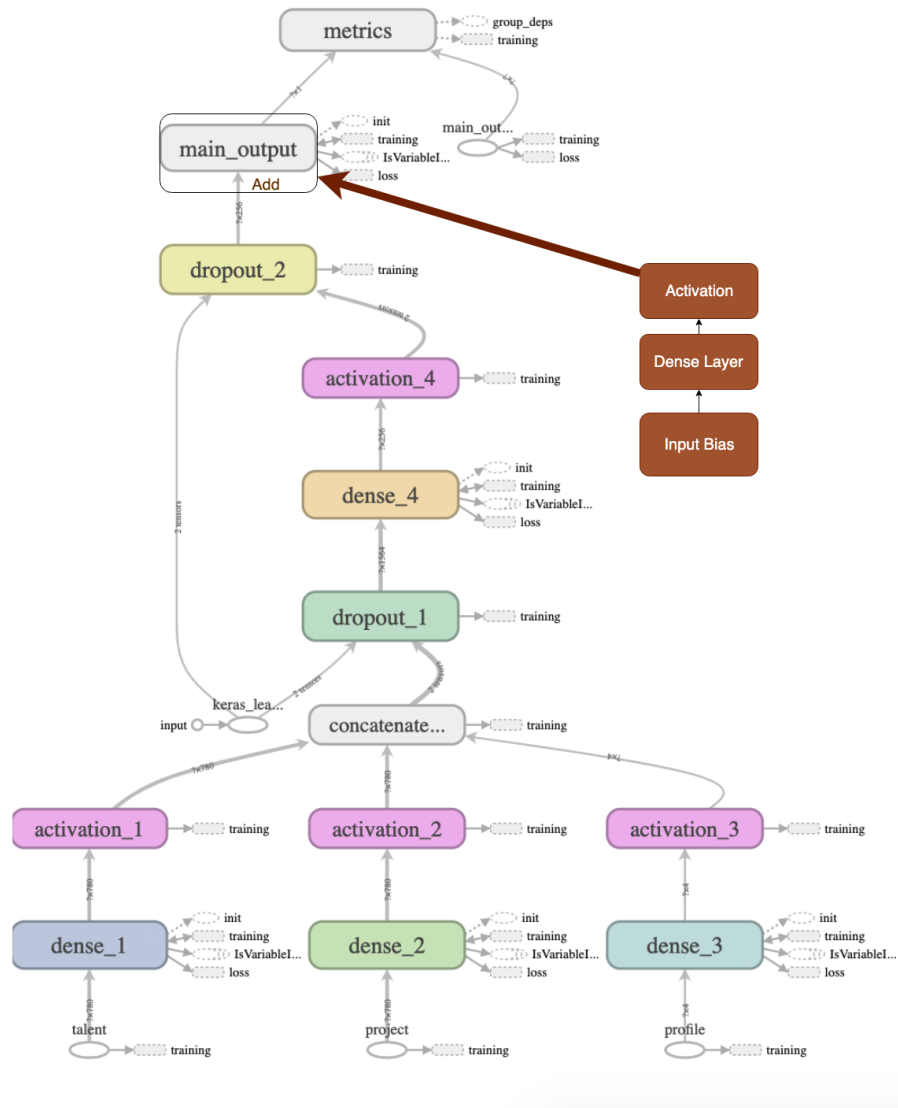


Figure 3.18.: Neural networks model with the addition of feedback loop bias

## 4. Evaluation

This chapter focuses on assessing the results of the recommender systems that were created in the 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 equal to the user satisfaction. Before starting this thesis, we analyzed what properties are essential to improve user satisfaction in job recommender systems. The research led us to diversity as a metric [CHV15].

When we recommend talent to a project, the only essential evaluation properties are the accuracy of the first item in the list, the 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 vital to increase user satisfaction. That is 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, which include offline, online, and user studies [SG11]. Offline evaluation serves the purpose of calculating accuracy and diversity. This type of experiments do not involve personal 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 to 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*. We hypothesize that 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 is why the generalization power for different job recommender datasets is secured.

### 4.1. Unsupervised Individual Recommendation

As it was explained in section 3.2, unsupervised individual recommendation addresses the 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, and the second one suggests the most popular candidates, and the last one is the combination of the first two.

When we calculate the accuracy for these methods, we use the top one and top five accuracies. For the individual recommendation methods, top one refers to what percent of winners for roles were correctly predicted on the first try. Top five is the other case where we do the same if the recommender system has five tries. It can also be explained that the evaluation method checks if the winner of the role exists in the first five positions of the ranked recommendation list.

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

#### 4.1.1. Existing Company Recommender

The company 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 crucial 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 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, the rate of acceptance for either one of first five talents in the list were only 21%. 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, which is a 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 unsupervised recommender, so that the recommenders are comparable.

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

As table 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 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 skills do not have 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 two 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 the recommendation by similarity, it suggests that each row is normalized first, so that the highest values correspond to one and the smallest to 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 descending order.

**A** In this first set, we use the plain mode like it was told in the previous paragraph. This setting 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 outcome indicates that many projects 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 is why we removed the talents that have no skill vector present. Doing this increased the accuracy to 36%.

**C** The manual control of the data showed that further improvements could be made to the algorithm. An idea was normalizing columns added 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 many projects with particular skills, will not lose that advantage against others. However, this method decreases the accuracy of 31%. This implicates that when the project owners choose talents, the information of how many projects the talents completed with relevant skills do not play a significant role. However, it is hard to conclude since every employer is different.

**D** To verify this observation, 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 did not 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 much information about talent skills.

In the end, we decide to use the last version. Because that version has the smallest data frame 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 it is a structure 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, as we mentioned before.

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

### User Study

As this is the first user study we conduct, some background information is necessary. We asked eight recruiters about what they think about the first recommended item of the list and what they think about the overall list value for individual roles. We also asked them about user satisfaction, but the answers were always the same as the overall list value, so we put them together as an overall list value. The author gave them as little information as possible to avoid any bias on the recruiters. The table 4.2 shows the averaged results. Recruiters were allowed to give values between zero and five, with zero the worst and five the best.

Table 4.2.: A table that shows results of user study about individual recommendation by similarity

First Item Value	Overall List Value
4.375	3.8125

#### 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. This type of recommendation system is only evaluated via offline evaluation because the experience level property is only available for the Freelancer dataset. That is why we did not implement it on the dashboard.

### 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 of 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 five chances.

#### 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 recommenders with some weights. The reason for that is to be able to predict results for as many projects as possible. The table 4.3 depicts how many projects is it possible to predict the results. In many projects, it is not possible to predict the results, because there are no skill vectors for the project or the bidders. The slimmed-down version that we developed to recommend by similarity cannot predict for all projects, because a significant percentage of the winners do not 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. Since the popularity information for the Motius dataset does not exist, we did not implement this feature into the dashboard. That is why we also did not conduct a corresponding user study for this algorithm.

Table 4.3.: A table that shows the amount of projects that we can generate predictions.

Similarity	Popularity	Hybrid
4987	20433	14152

#### Offline Evaluation

Since we decided on 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 number of projects that are 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 the highest popularity weight, and the lowest project coverage can be seen on the highest similarity weight.

It is clear that there is no obvious best weight choice since we both want to increase



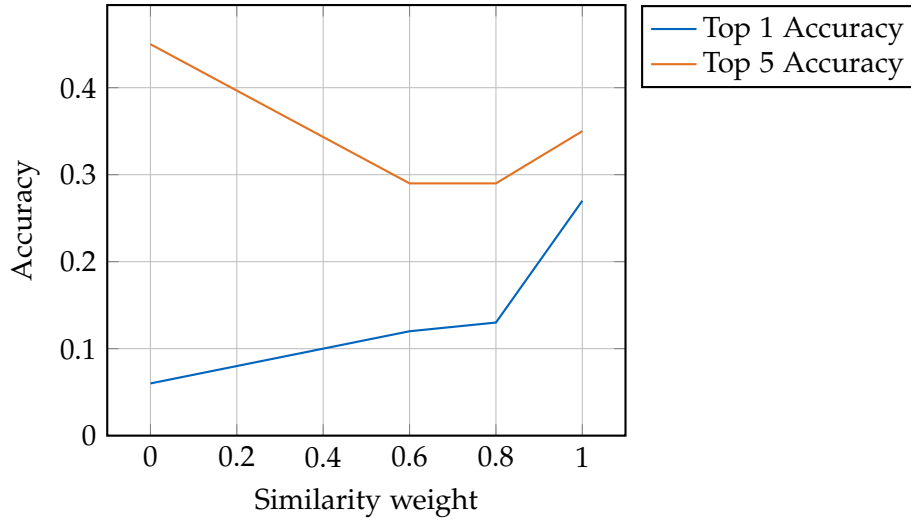


Figure 4.1.: Effect of weight selection on hybrid recommenders to the accuracy

project coverage and accuracies. We chose the similarity weight 0.6 and popularity weight 0.4 to increase these results.

## 4.2. Supervised Individual Recommender

The details of the supervised individual recommender are already clarified in 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 the recommender is measured for a different amount of layers, different cost functions, other activation functions, and various optimizations. In the end, it would not make sense to visualize all of the distinct results. However, we must note that the maximum accuracy that is measured with the test set is 79%. Here the accuracy does not reveal top 1 or top 5 predictions that we used before. This prediction with the test set demonstrates that the model can predict 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

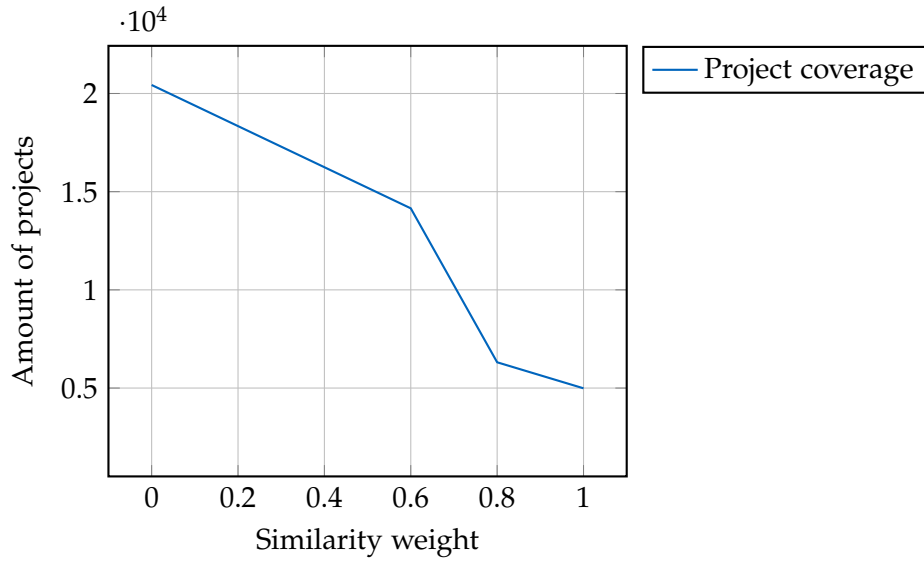


Figure 4.2.: Effect of weight selection on the amount of projects that we can generate predictions.

person should be rejected, and above 0.5 gets treated as the talent should be hired. That is 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 accuracies. 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.

#### 4.2.2. Using Embeddings

To be able to use embeddings, we reduce the dimensionality of the data first. The reduction makes the data frame 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 is why we gained evaluation results for both offline and online cases.

##### Offline Evaluation

In this round of offline evaluation, run evaluation tests for neural networks and the hybrid method. In this context, the term hybrid signifies the linear combination of neural networks and cosine similarity.

**Neural Networks** 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 do not add the extra profile information, the most significant value for guessing the winner is 9%.

**Hybrid** The outcome of this type of recommender come from the multiplication of results from the supervised and unsupervised recommendation systems. However, the offline evaluation results are worse than pure neural networks, which are 10% for the top 1 and 49% for the top 5.

### User Study

The rules that we set for the previous user study are still valid[See 4.1.2], but the topic has changed. In this user study, we ask the same questions for the results from neural networks and the hybrid results that come from the combination of neural networks and cosine similarity.

**Neural Networks** For this first user study, we present a list for a randomly selected consisted role, which is generated by neural networks. The results are shown in the table 4.4.

Table 4.4.: A table that shows results of user study about individual recommendation generated by a neural networks

First Item Value	Overall List Value
2.5625	2.8125

**Hybrid** For this user study, we listed talents for the same role that were suggested by the multiplication of neural networks and cosine similarity results that come from section 4.1.2. The results are shown in the table 4.5.

Table 4.5.: A table that shows results of the user study about hybrid individual recommendation

First Item Value	Overall List Value
4.5	4.0625

### 4.3. Unsupervised Group Recommender

Unsupervised group recommender is another type of recommender that we implemented in the dashboard. The unsupervised group recommender 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 does not check the relation of talents that are chosen.

#### Offline Evaluation

Many evaluation methods are part of the offline evaluation. Since they are employed the first time in this thesis, we explain them in more detail.

**Accuracy** The calculation of offline accuracy for group recommender is different from the individual one [See section 4.1]. For the case of group recommenders, we calculate the top one and top five accuracies for every project of each group and calculate their average.

The first detail that we check is the top one and top five accuracies as we did before. The performance did not change from the results of 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 mechanisms to measure diversity, and the first equations were coined at the beginning of 2000's [SM01]. The main equation we use is

$$ILD = \frac{1}{|R|(|R| - 1)} \sum_{i \in R} \sum_{j \in R} d(i, j) \quad (4.1)$$

The above *inter-list diversity* [See equation (4.1)] 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 [See chapter 3]. With various sizes of groups from the freelancer.com dataset, we calculated the average ILD value of 40% with this baseline suggestion engine.

**Unexpectedness** Unexpectedness tells us how unforeseen the data is to the recruiters. In the equation in equation (2.4.1)  $\mathcal{J}_u$  is a symbol for a 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. For the user studies, we also wanted to ask the recruiters their opinions about the unexpectedness[See section 4.3.2]. However, the subjects of the user study can not know about the past interactions of the system. That is why the unexpectedness results are only present for the offline evaluation.

### User Study

In this user study, we pick a random group and ask the opinions of the recruiters for the performance of the first item for each project in the group, the diversity of these first items in the group and the overall list value. The results are shown in the table 4.6.

Table 4.6.: A table that shows results of the user study about unsupervised group recommendation

First Item Value	Overall List Value	Diversity
4.125	3.4375	1.6875

### 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 formula

$$R_{opt}(\lambda) = \arg \max((1 - \lambda) \frac{1}{|R|} \sum_{i \in R} f_{rel}(i) + \lambda div(R)). \quad (4.2)$$

In the above equation, 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 different 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 affect the accuracy, diversity, or overall list value.

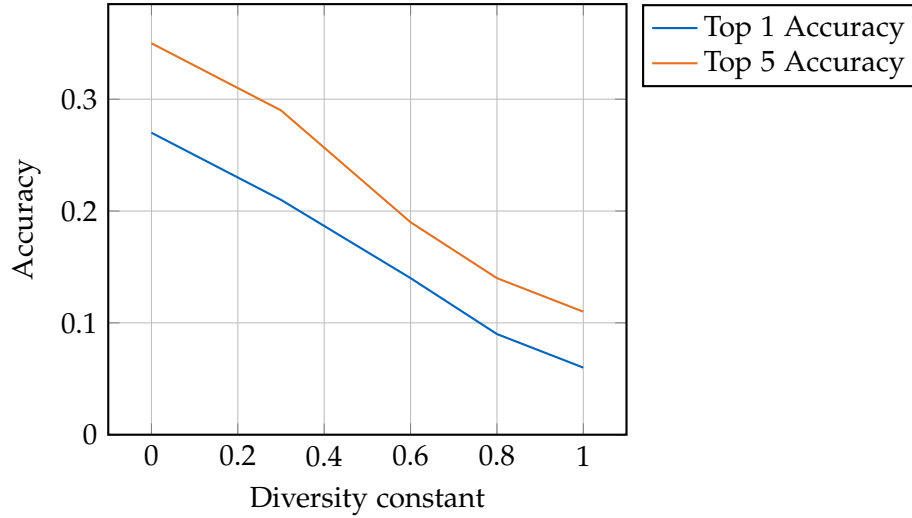


Figure 4.3.: Effect of diversity constant on unsupervised group recommender to the accuracy

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

**Diversity** For diversity, we use the same equation that we used in paragraph 4.3.1. However, a significant 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 paragraph 4.3.1. Again, we calculate the values separately for different diversity rates and drawn on the figure 4.5. It is evident that the diverse recommender did not increase the unexpectedness as steep as it improved diversity.

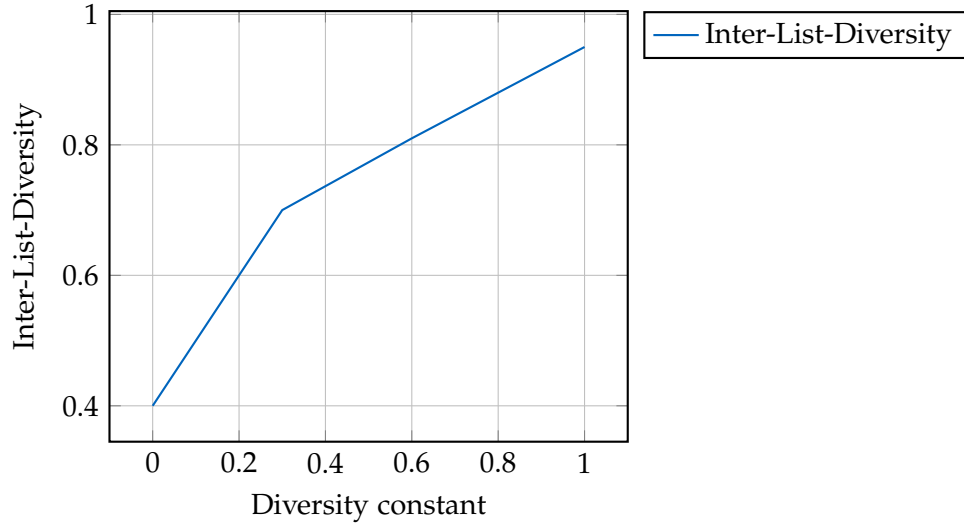


Figure 4.4.: Effect of diversity constant on unsupervised group recommender to the diversity

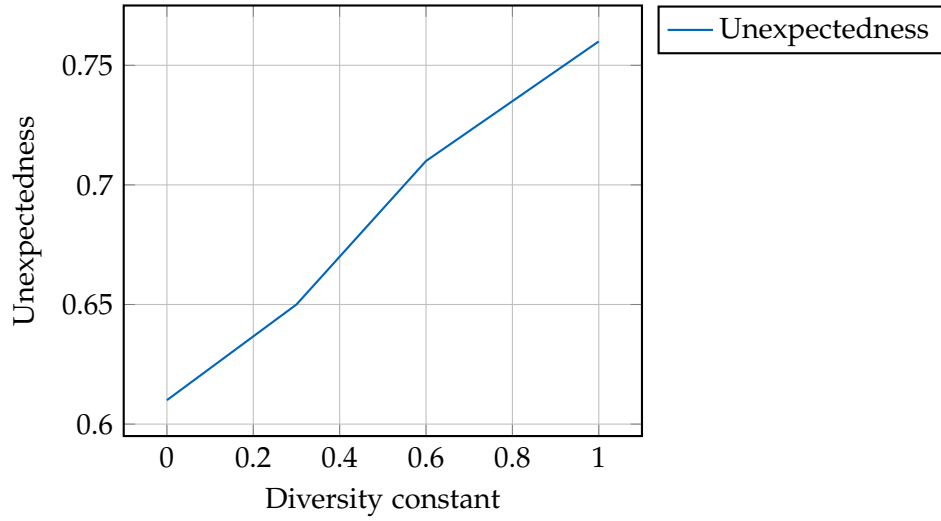


Figure 4.5.: Effect of diversity constant on unsupervised group recommender to the unexpectedness

### User Study

In this part of the user study, we gave the recruiters the control of the dashboard and received their opinion about a specific group. We asked them about the success of firsts items for each project, the diversity between them, and the overall list value. In comparison to the individual recommenders, we asked these questions for different  $\lambda$  values, which is the diversity constant. The results of this study are drawn in the graph 4.6.

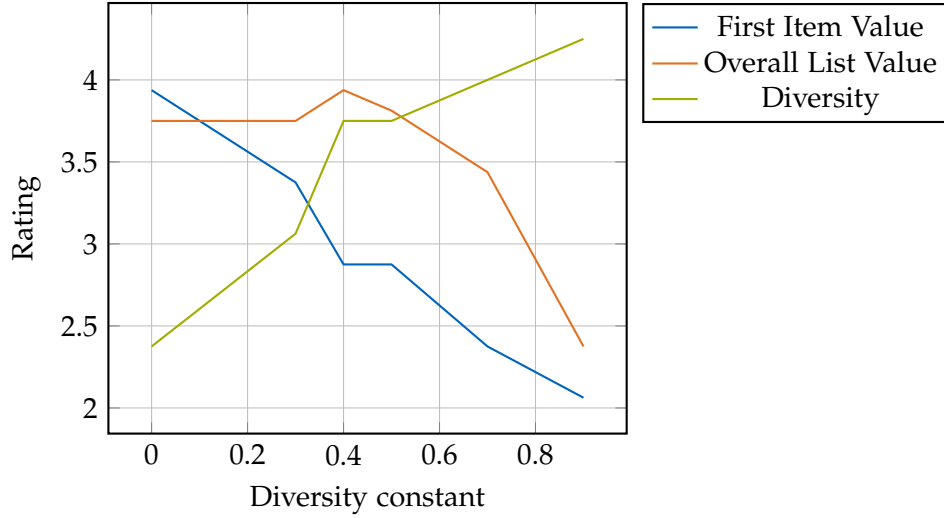


Figure 4.6.: Effect of diversity constant on unsupervised group recommender to the average user opinion

## 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 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 does not aim to optimize the diversity of talents.



### Offline Evaluation

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

**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 paragraph 4.3.1. When the average unexpectedness for all projects is measured, the value that we see is 65.5%.

### User Study

The rules of the user study in section 4.3.1 still applies to this study. However, the recruiters give their opinions about the results of the group neural network approach, which are listed in table 4.7.

Table 4.7.: A table that shows results of the user study about supervised group recommendation

First Item Value	Overall List Value	Diversity
3.125	3	2.375

#### 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.

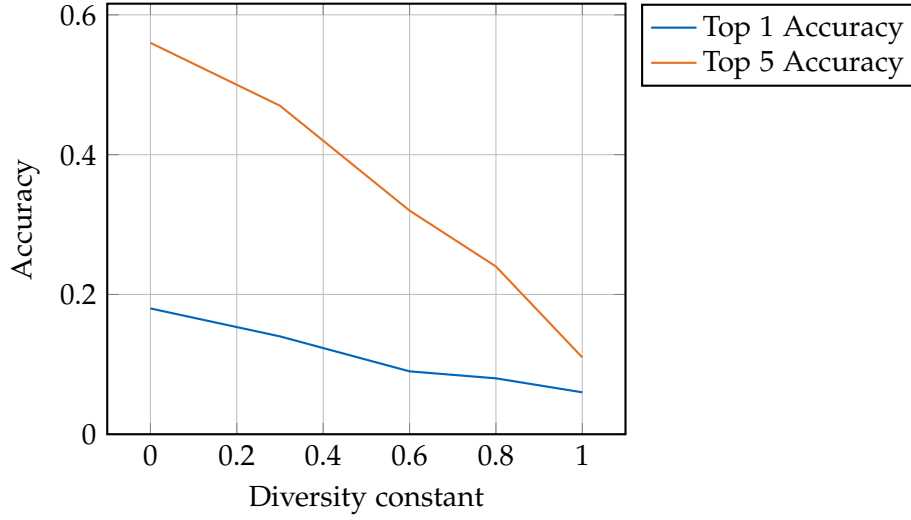


Figure 4.7.: Effect of diversity constant on supervised group recommender to the accuracy

**Accuracy** The figure 4.7 depicts the impact of the diversity constant change to the accuracy, which is calculated on freelancer.com dataset.

**Diversity** To calculate diversity, we use the same equation that we used in paragraph 4.3.1 with different diversity constants. The figure 4.8 shows the diversity in recommendation lists under the effect of diversity constant.

**Unexpectedness** We use the same formula, which was explained in paragraph 4.3.1 and the values are calculated separately for different diversity rates that are drawn on the figure 4.9.

### User Study

In the scope of this user study, the user gave ratings to different supervised group recommendations that were induced by varying  $\lambda$  values. The results of this study are drawn in the graph 4.10.

## 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

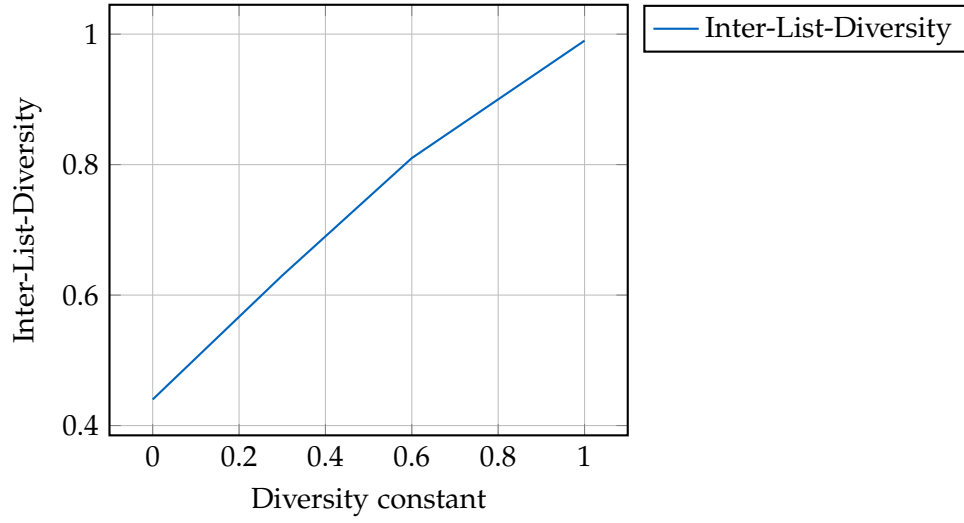


Figure 4.8.: Effect of diversity constant on supervised group recommender to the diversity

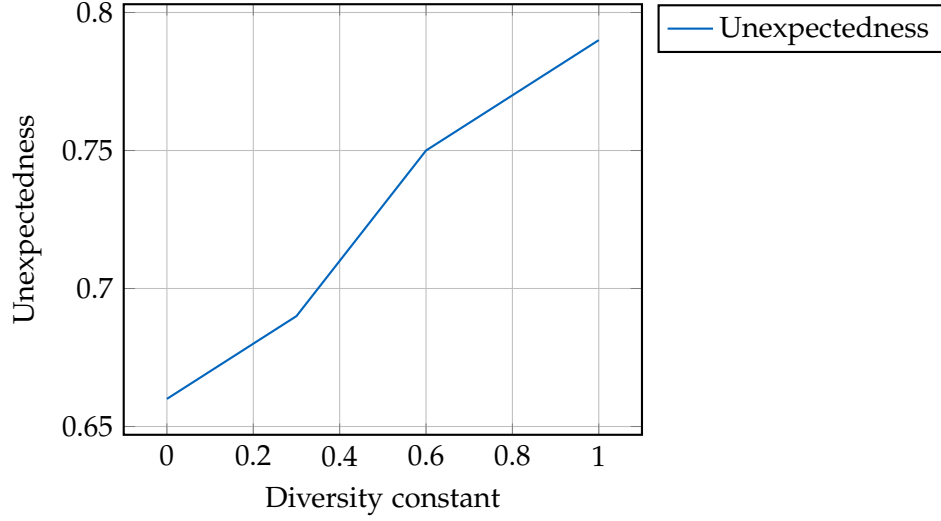


Figure 4.9.: Effect of diversity constant on supervised group recommender to the unexpectedness

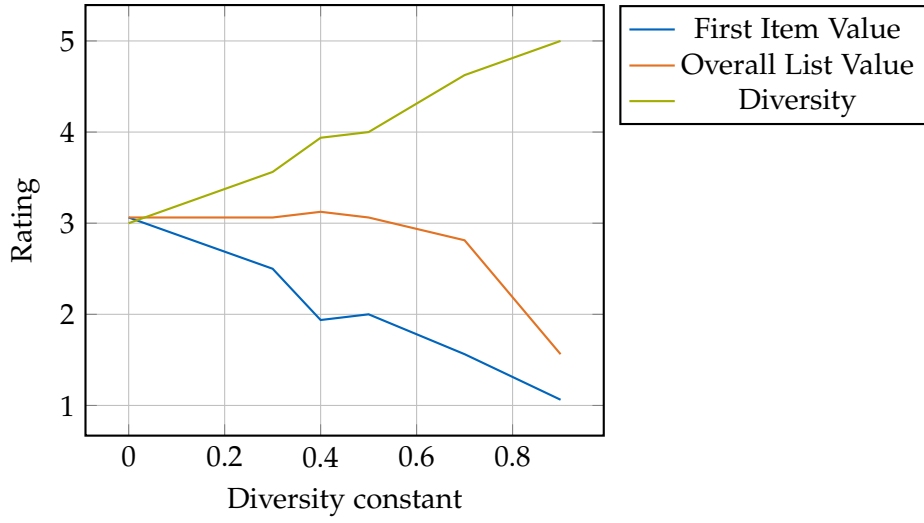


Figure 4.10.: Effect of diversity constant on supervised group recommender to the average user opinion

that use the dashboard, post feedback to talents, the recommendation list is instantly changed accordingly. This instant change is called the direct effect. Posting feedback also modifies the existing dataset that is used to train the model. Sending positive feedback modifies the data point to a positive label, which is one. Also, sending negative feedback modifies the data point to a negative label, which is zero. After that, the model will produce *better* results when we retrain with the modified data.

This section focuses on the evaluation of this loop. It not feasible to conduct an 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 instructed on 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. Because the training data consists of more 350.000 data points, the number 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, the results are shown to the recruiters again and their opinions are asked again.

The table 4.8 depicts the opinions of the recruiters before and after retraining. The first line is the data before, and the second line contains their ratings for after retraining for the same group recommendation.

Table 4.8.: A table that shows the user opinions before and after re-training.

First Item Value	Overall List Value	Diversity
3.125	3	2.375
3.75	3.6875	2.5625

The discussion about the results is included in chapter 5.

## 4.6. Summary

In this chapter, we showed the evaluation results for the unsupervised and supervised individual recommender, unsupervised, and supervised group recommender system and the feedback loop. These results come from offline, online evaluation, and user studies.

In the next chapter, we continue with understanding the results.

## 5. Discussion

This chapter of the thesis, comments on the results that we achieved in chapter 5 and speculates about possible reasons.

### 5.1. Comparison of Individual Recommenders

First, we start by comparing the results of individual recommenders for different kind of evaluation methods and properties.

#### 5.1.1. Offline Evaluation

The first evaluation method we discuss is offline evaluation, which includes the properties accuracy and coverage.

##### Accuracy

As always, we first check the accuracy levels of different individual recommendation systems. The table below shows the offline evaluation results for different recommenders.

Table 5.1.: Offline evaluation results for different recommenders are shown.

Type	Name	Top 1	Top 5
Unsupervised	Motius	0.07	0.21
Unsupervised	Similarity	0.28	0.36
Unsupervised	Popularity	0.07	0.45
Unsupervised	Hybrid	0.12	0.29
Supervised	Neural Network	0.19	0.56

The existing recommendation system at the company Motius provides the scores 0.07, 0.21 for the top one, and top five accuracies respectively. A pure similarity-based approach that we developed reached the values 0.28 and 0.36 for the same measures and the pure popularity algorithm achieves 0.07 and 0.45. Lastly, for unsupervised methods,

the combination of similarity and popularity algorithms did not reach better results than the similarity recommender. That is why we can conclude that the similarity based recommendation is the best unsupervised method for offline accuracy.

When we also include the supervised recommendation that we developed with neural networks, we achieve the top one and top five accuracies of 0.19 and 0.56. All of them together, unsupervised similarity recommender still had the best offline accuracy performance for guessing the first talent, but the neural networks suggest better lists in general.

### Coverage

Although we said the pure similarity-based approach is the best one for offline accuracy, it is also the worst one in terms of coverage. Since a significant proportion of talents did not get selected according to their skill match on the website *Freelancer.com*, it is only possible to make predictions for 4987 projects with the similarity-based recommendation system. However, the popularity recommender can make predictions for 20433 projects with a worse accuracy, because it always suggests the most experienced person among bidders and there are mostly at least one experienced talent inside the group of bidders for every project.

When we combine the popularity and the similarity recommendation systems with the weights 0.4 and 0.6 respectively, we can make predictions for 14152 projects. In addition to that, we achieve the accuracies of 0.12 for the top one and 0.29 for the top five. All in all, if we only want to have the maximum coverage, the popularity recommender is the most successful unsupervised method, but hybrid recommenders are more beneficial if we want to optimize both coverage and accuracy.

The supervised method has the same coverage as the popularity recommender, which is the highest among others. On top of that, the supervised method also has superior accuracy rates compared to unsupervised methods that make it favorable. However, the hybrid function of the combination of neural networks and cosine similarity performed worse in terms of accuracy and coverage. With all the information that we have, pure neural networks perform better than unsupervised processes, when we want to optimize both accuracy and coverage.

### 5.1.2. User Study

We also ran a second evaluation round that consisted of a user study with recruiters. The questions we asked were their opinions about the first item in the list and general satisfaction about the list.

### **First Item Value**

The recruiters rated the recommendation by similarity method a score of 4.375, which is an unsupervised recommendation method. Same recruiters graded pure neural networks the score of 2.5625 and the hybrid of the first two methods the score of 4.5. From this information, we can conclude that the hybrid model performed the best. However, we must note that the number of participants is only eight, and the score difference is also only around five percent.

### **Overall List Value**

Overall list value, also known as user satisfaction, had similar results to the first item value. The unsupervised model got the score of 3.815, neural networks got 2.8125, and the hybrid of the first two got 4.0625.

Here, it is interesting that the hybrid model outperformed other methods, which is not the case for offline accuracy. This situation can indicate that developing models that are only based on offline accuracy, does not always mean that the model will also increase personal satisfaction. However, further evaluation and research is necessary since our evaluation group only contains eight subjects.

## **5.2. Comparison of Group Recommenders**

Group recommenders of this thesis consist of two main categories: baseline and diverse. These categories also have unsupervised and supervised versions.

### **5.2.1. Offline Evaluation**

Offline evaluation of group recommenders comprises three different measures: accuracy, diversity, and unexpectedness.

#### **Accuracy**

The accuracy of the unsupervised baseline recommender is 0.28 for the top one and 0.36 for the top five. When we add the diversity constant to the equation, these results do not get any higher. When we employ the supervised baseline or diverse recommendation system, the maximum accuracy we see is 0.18 for the top one and 0.56 for the top five.

Just like in the individual recommender [See section 5.1.1], we conclude that the unsupervised similarity method predicts the winners better, but the supervised approach produces lists with better accuracy in general.



### Diversity

The diversity metric that we used *Inter-List-Diversity* resulted in being between 0.4 and 0.9 depending on the diversity constant. The only exception was that the diversity of the results from the neural networks were around 10% higher than the given results. These results are expected because projects of a group mostly require similar skills, which also produces similar talents for the group. Similar talents in a group create lower diversity suggested chosen talents. In contrary to that, the supervised learning model may produce results with more variety, because it also includes other factors such as extra profile information.

### Unexpectedness

The unexpectedness of the results from unsupervised recommenders range from 0.61 to 0.77, and the unexpectedness of the results from supervised recommenders range from 0.66 to 0.79. The reason is same as in the previous section [See 5.2.1].

#### 5.2.2. User Study

As part of the user study, we asked a group of recruiters about their opinions on the first item on the list, overall list value, and the diversity of the group.

### First Item Value

The supervised baseline approach received a score of 3.125 from the subjects, and this is also the maximum score that we received for the supervised diverse recommender. The unsupervised versions went up to a score of 4.125. Just like the offline accuracy for group recommenders, we can conclude that the unsupervised similarity-based approach without diversity generated the best *winners* for the projects.

### Diversity

According to the user study, the average diversity score for the unsupervised baseline recommender is 1.6875. When we add the diversity constant, it goes up until 4.2. For the supervised approach, the baseline score is 2.375, and the diverse version goes up until having the diversity score of almost 5 out of 5. As we said before, neural networks produce more varied lists compared to the similarity-based approaches, and these scores are expected. From the scores that we acquired, supervised diverse recommender with a high diversity constant is the most logical approach, if the diversity is number one constraint.

### Overall List Value

Overall list value, also known as user satisfaction, of the unsupervised baseline recommender is 3.4375 on average. When we also add the diversity constant, we see the maximum value 3.9 for the diversity constant of 0.4. On the other hand, the overall list value is 3 for supervised baseline recommender, and we also see the maximum value of 3.1 for the diversity constant of 0.4.

The answers to this user study suggest that some level of diversity has a positive effect on user satisfaction because the highest overall list values were reached with the same diversity level for both supervised and unsupervised approaches. The number of subjects is only eight, and the peak values of user satisfaction do not provide a significant increase compared to other values. However, this proves that the hypothesis is worth further research and experimentations.

### 5.3. Feedback Loop

Since it is not possible to conduct an offline evaluation for the feedback loop, we conducted a combination of online evaluation and user study. Subjects were motivated to enter as many feedback as possible, and the author of this thesis continued adding more feedback according to the principles that the recruiters mentioned. Recruiters were asked before about results of a group, and we asked them the same questions afterward.

According to this user study, the first item values increased from 3.125 to 3.75, overall list value increased from 3 to 3.6875, and the diversity increased from 2.375 to 2.5625. From these results, improvements in first item value and user satisfaction can be seen. However, the number of subjects was only eight, and the author of the thesis added more feedback manually because of these reasons we can also say that the topic of feedback loop requires more experiments to be made. At least, we can be optimistic that the topic worths further experiments.

### 5.4. Other Topics to Discuss

This section of the thesis contains other topics that are required acknowledging.

#### 5.4.1. Problems about datasets

In the scope of the thesis, we use two databases: one from *Freelancer.com* and another one from the company *Motius*. Each of the datasets come with their problems, because

of that, it would make sense to run the same models, algorithms, and experiments with other datasets and settings.

### **Problems about Freelancer.com dataset**

This dataset contains information about projects, talents, and their interactions. However, there are many employers, and many of them have different selection criteria. That is why it is not a trivial task to find a scientific basis for all of the employers. When we checked projects individually, we found some selections by employers that are not logical. For example, there are multiple instances of people got hired, although there were other competitors with more relevant skills, higher experience level that is also cheaper. Due to this, the author thinks that the motivation text of the bidders and some other non-mathematical factors also play a role in the selection process.

### **Problems about Motius dataset**

The company Motius runs an internal recommender system that suggests talents to roles, and we acquired the logs from this recommender. We combined those logs with the other dataset and trained the model with that data.

After carefully debugging the model and digging deep into both datasets, we detected some of the problems. First of all, only a small portion of Motius and Freelancer.com features overlap. Since Motius dataset is much smaller, many weights that are unique for this dataset stay untrained. Another problem is that there are instances of duplicate results from the Motius recommender with different labels. For example, for the position of *Android dev*, the same person got recommended twice by the internal recommender, and that person got once accepted and once rejected. This situation was a problem that we solved by removing the duplicate elements. Also, just like in the Freelancer.com dataset, many talents were selected even though they do not meet skill requirements. Another problem in the Motius dataset is that many talents only have couple skills entered in the system, although some small number of others contain 20 skills. Because of this, these talents with 20 skills get favored by the system. Lastly, many of the selected talents are not selected because their skills fit better than the others, but they were selected due to reasons unknown to the system.

#### **5.4.2. Problem about the recommender type**

Recommender systems based only on content generally suffer from the problems of limited content analysis and overspecialization [SM95]. This case is problematic because we only use content-based recommendation systems. The reason for this is the fact that collaborative filtering does not fit to our use-case well. A use-case that allows the

combination of content-based and collaborative filtering recommender systems may perform better.

#### **5.4.3. Reasons to use hybrid recommender**

As we also saw in this chapter, "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" [AT05]. That is why hybrid recommenders should always be given a chance when it is possible.

### **5.5. Summary**

In this chapter, we discussed the outcome from different kind of experiments and commented on the reasons and consequences. Then, we presented the problems that arose with the use-case and datasets of choice. We have also added many concrete suggestions for people that want to research this topic further.

## 6. Conclusion

In the first chapter [See chapter 1], we asked some questions which were:

- Does high accuracy guarantee high user satisfaction?
- Does diversity affect user satisfaction positively?
- Would a feedback loop enhance user satisfaction?

From the discussion in section 5.1.2, we found out that an increase in accuracy does not guarantee higher user satisfaction; they do not correlate. In the offline evaluation, the unsupervised similarity-based method has the top one accuracy with 0.28, and the supervised neural networks method has the best top five accuracies with 0.56. However, the user study suggests that the combination of these two methods resulted in the best user satisfaction, with 4.0625 points out of 5. The research in the section 2.3, suggest that the reason for that is the fact that other properties also contribute to the user satisfaction and higher user satisfaction can only be guaranteed with a combination of different properties.

In section 5.2.2, we talked about the effect of diversity on user satisfaction. The unsupervised group recommender produced diversity scores of 1.6875 to 4.2 out of 5 depending on the diversity constant. However, user satisfaction reached the maximum value of 3.9 out of 5 with the diversity constant of 0.4. For the supervised group recommender, the diversity score ranged from 2.375 to 5 out of 5 depending on the diversity constant. The maximum value of 3.1 for user satisfaction was again reached with the diversity constant of 0.4 out of 1. From the relevant evaluation results, we found out that some level of diversity affects user satisfaction positively. However, this does not mean that there is a correlation between diversity and user satisfaction. From our findings, user satisfaction increases until some value of diversity, and then it drops significantly.

The last question is about feedback loops, and that topic was discussed in section 5.3. Only enabling a feedback loop did not affect the results positively or negatively. When performed an online evaluation round with around 200 entries, and then retrained the model, there was also a minimal effect on the outcome. However, we saw a noticeable difference with 3500 feedbacks. After having the last round of user study,

## 6. Conclusion

---

the experiment subjects agreed that user satisfaction increased from 3 to 3.6875. Here we can conclude that a feedback loop with enough constructive feedback can increase user satisfaction.

## A. 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 A.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 A.2.: Simplified version of the extra code for enabling feedback



## List of Figures

2.1. High level architecture of a content based recommender [De +15]. . . .	5
2.2. High level architecture of a feedforward neural network [QD11]. . . .	16
3.1. An example project from the Freelancer.com Website . . . . .	19
3.2. The winner and other bidders to the same project . . . . .	20
3.3. The list of tops skills by a talent on Freelancer.com web page . . . . .	21
3.4. The talent skill matrix from freelancer.com . . . . .	24
3.5. The talent extra information matrix from Freelancer.com . . . . .	25
3.6. The graph that explains the sparse input model . . . . .	27
3.7. 3D version of the embedding space that is created for projects of this thesis	28
3.8. Threshold figure . . . . .	29
3.9. Training data that contains padded embedding skill vectors . . . . .	30
3.10. Silhouette scores of many different $k$ values of k-means . . . . .	34
3.11. Centers of clusters that are projected on a 2D space . . . . .	35
3.12. Examples of some centers of clusters that are projected on a 2D space .	36
3.13. Main screen of the dashboard . . . . .	36
3.14. A snippet from the list of all projects that start with the letter $a$ . . . .	37
3.15. A screenshot from the list of all recommendations from neural networks for the project <i>a.i. &amp; software engineer</i> . . . . .	38
3.16. A screenshot from the list of all recommendations from neural networks for the project <i>a.i. &amp; software engineer</i> . . . . .	39
3.17. A screenshot from the list of all recommendations from diverse cosine similarity for the group 9 . . . . .	40
3.18. Neural networks model with the addition of feedback loop bias . . . .	42
4.1. Weight figure . . . . .	49
4.2. Coverage figure . . . . .	50
4.3. Unsupervised-diversity-accuracy Figure . . . . .	54
4.4. Unsupervised-diversity-diversity Figure . . . . .	55
4.5. Unsupervised Diverse Group Unexpectedness Figure . . . . .	55
4.6. Unsupervised diverse user study figure . . . . .	56
4.7. Accuracy in Diverse Supervised Group Recommender . . . . .	58
4.8. Diversity in Diverse Supervised Group Recommender . . . . .	59

*List of Figures*

---

4.9. Supervised Diverse Group Unexpectedness Figure . . . . .	59
4.10. Supervised diverse user study figure . . . . .	60
A.1. Model Code . . . . .	71
A.2. Model Bias Code . . . . .	72

## List of Tables

4.1. Evaluation mid-results . . . . .	45
4.2. User Study Individual Similarity . . . . .	47
4.3. The number of predicted projects . . . . .	48
4.4. User Study Individual Neural Networks . . . . .	51
4.5. User Study Individual Hybrid . . . . .	51
4.6. User Study Group Unsupervised . . . . .	53
4.7. User Study Group Supervised . . . . .	57
4.8. Online Evaluation Table . . . . .	61
5.1. Offline evaluation results for different recommenders are shown. . . . .	62

# 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.
- [AT05] G. Adomavicius and A. Tuzhilin. "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions." In: *IEEE Transactions on Knowledge & Data Engineering* 6 (2005), pp. 734–749.
- [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. Breiteringer. "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.
- [Bur00] R. Burke. "Knowledge-based recommender systems." In: *Encyclopedia of library and information systems* 69.Supplement 32 (2000), pp. 175–186.
- [Bur02] R. Burke. "Hybrid recommender systems: Survey and experiments." In: *User modeling and user-adapted interaction* 12.4 (2002), pp. 331–370.
- [CAS16] P. Covington, J. Adams, and E. Sargin. "Deep neural networks for youtube recommendations." In: *Proceedings of the 10th ACM conference on recommender systems*. ACM. 2016, pp. 191–198.

- [Che+16] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. "Wide & deep learning for recommender systems." In: *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM. 2016, pp. 7–10.
- [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.
- [CVS07] C. Christakou, S. Vrettos, and A. Stafylopatis. "A hybrid movie recommender system based on neural networks." In: *International Journal on Artificial Intelligence Tools* 16.05 (2007), pp. 771–792.
- [De +15] M. De Gemmis, P. Lops, C. Musto, F. Narducci, and G. Semeraro. "Semantics-aware content-based recommender systems." In: *Recommender Systems Handbook*. Springer, 2015, pp. 119–159.
- [DK11] C. Desrosiers and G. Karypis. "A comprehensive survey of neighborhood-based recommendation methods." In: *Recommender systems handbook*. Springer, 2011, pp. 107–144.
- [Fel+15] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. "Constraint-based recommender systems." In: *Recommender systems handbook*. Springer, 2015, pp. 161–190.
- [GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [GG16] Y. Gal and Z. Ghahramani. "A theoretically grounded application of dropout in recurrent neural networks." In: *Advances in neural information processing systems*. 2016, pp. 1019–1027.
- [Gri18] M. Grinberg. *Flask web development: developing web applications with python*. "O'Reilly Media, Inc.", 2018.
- [Gru19] J. Grus. *Data science from scratch: first principles with python*. O'Reilly Media, 2019.
- [HKR00] J. L. Herlocker, J. A. Konstan, and J. Riedl. "Explaining collaborative filtering recommendations." In: *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM. 2000, pp. 241–250.
- [HR18] B. Hanin and D. Rolnick. "How to start training: The effect of initialization and architecture." In: *Advances in Neural Information Processing Systems*. 2018, pp. 571–581.

- [HZ11] N. Hurley and M. Zhang. "Novelty and Diversity in Top-N Recommendation – Analysis and Evaluation." In: *ACM Trans. Internet Technol.* 10.4 (Mar. 2011), 14:1–14:30. ISSN: 1533-5399. DOI: 10.1145/1944339.1944341.
- [Jol11] I. Jolliffe. *Principal component analysis*. Springer, 2011.
- [KB15] Y. Koren and R. Bell. "Advances in collaborative filtering." In: *Recommender systems handbook*. Springer, 2015, pp. 77–118.
- [LV07] J. A. Lee and M. Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.
- [Mas11] M. Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. " O'Reilly Media, Inc.", 2011.
- [MRK06] S. M. McNee, J. Riedl, and J. A. Konstan. "Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems." In: *CHI '06 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '06. Montré#233;al, Qu#233;bec, Canada: ACM, 2006, pp. 1097–1101. ISBN: 1-59593-298-4. DOI: 10.1145/1125451.1125659.
- [Mur12] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- [MW+14] S. B. Maind, P. Wankar, et al. "Research paper on basic of artificial neural network." In: *International Journal on Recent and Innovation Trends in Computing and Communication* 2.1 (2014), pp. 96–100.
- [Ngu+14] T. T. Nguyen, P.-M. Hui, F. M. Harper, L. Terveen, and J. A. Konstan. "Exploring the Filter Bubble: The Effect of Using Recommender Systems on Content Diversity." In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW '14. Seoul, Korea: ACM, 2014, pp. 677–686. ISBN: 978-1-4503-2744-2. DOI: 10.1145/2566486.2568012.
- [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>.
- [Ped18] D. Pedamonti. "Comparison of non-linear activation functions for deep neural networks on MNIST classification task." In: *arXiv:1804.02763* (2018).
- [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.

- [QD11] R. Quiza and J. Davim. "Computational Methods and Optimization." In: Jan. 2011, pp. 177–208. ISBN: 978-1-84996-449-4. DOI: 10.1007/978-1-84996-450-0.
- [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.
- [SL17] B. Smith and G. Linden. "Two decades of recommender systems at Amazon.com." In: *Ieee internet computing* 21.3 (2017), pp. 12–18.
- [SM01] B. Smyth and P. McClave. "Similarity vs. diversity." In: *International conference on case-based reasoning*. Springer. 2001, pp. 347–361.
- [SM95] U. Shardanand and P. Maes. "Social information filtering: Algorithms for automating" word of mouth"." In: *Chi*. Vol. 95. Citeseer. 1995, pp. 210–217.
- [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.
- [WWY15] H. Wang, N. Wang, and D.-Y. Yeung. "Collaborative deep learning for recommender systems." In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2015, pp. 1235–1244.
- [You18] E. You. "Vue.js." In: *Diakses dari <https://vuejs.org/>, pada tanggal 17 (2018)*.