# TLTI

# 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: | 15.05.2019 |

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

Munich, 15.05.2019                                                Ozan Pekmezci

# Acknowledgments

# Abstract

# Contents

# 1 Introduction

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

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

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

## 1.1 Background of Classical Recommender Systems

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

This section contains brief information about classical recommender systems.

### 1.1.1 Why Recommender Systems?

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

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

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

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

### 1.1.2 Functions

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

[kopi - peyst]

- Find Some Good Items: Recommend to a user some items as a ranked list along with predictions of how much the user would like them (e.g., on a scale of one-to-five stars). This is the main recommendation task that many commercial systems address (see, for instance, Chap. 11). Some systems do not show the predicted rating.

- Find all good items: Recommend all the items that can satisfy some user needs. In such cases it is insufficient to just find some good items. This is especially true when the number of items is relatively small or when the RS is mission-critical, such as in medical or financial applications. In these situations, in addition to the benefit derived from carefully examining all the possibilities, the user may also benefit from the RS ranking of these items or from additional explanations that the RS generates.

- Recommend a sequence: Instead of focusing on the generation of a single recommendation, the idea is to recommend a sequence of items that is pleasing as a whole. Typical examples include recommending a TV series, a book on RSs after having recommended a book on data mining, or a compilation of musical tracks

- Recommend a bundle: Suggest a group of items that fits well together. For instance, a travel plan may be composed of various attractions, destinations, and accommodation services that are located in a delimited area. From the point of view of the user, these various alternatives can be considered and selected as a single travel destination

[kopi - peyst]

### 1.1.3 Applications

[kopi - peyst]

- Entertainment—recommendations for movies, music, games, and IPTV.

- Content—personalized newspapers, recommendation for documents, recommendations of webpages, e-learning applications, and e-mail filters.

- E-commerce—recommendations of products to buy such as books, cameras, PCs etc. for consumers.

- Services—recommendations of travel services, recommendation of experts for consultation, recommendation of houses to rent, or matchmaking services.

- Social—recommendation of people in social networks, and recommendations of content social media content such as tweets, Facebook feeds, LinkedIn updates, and others.

### 1.1.4 Objects

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

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

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

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

[kopi - peyst]

### 1.1.5 Types

[gelistir, buyut]

- Collaborative-Filtering: Recommendations based on how other users rated items. Similar users are identified and the items that they rated well are recommended. Has cold-start, sparsity and scalibility issues. First research paper released in the mid-1990s, still used widely [8].

- Content-based filtering: The requirements for this approach are features and the ratings of the items by users. A classifier for users' 'profile' is built and similar items are recommended based on it. It has the problem of recommending only very similar results that the user already is aware of. It was first mentioned on an academic paper on 1998 [8].

- Knowledge-based filtering: This approach also requires features of the items and explicit description of what the user needs or wants. Then, items that match those needs are recommended. The advantage of this approach is the fact that the system doesn't need data from different users. Unfortunately, the suggestion ability is rather static [4]. Research about this topic was first released on 1999.

- Hybrid Recommender Systems: Since all of the methods have some drawbacks, applications have shifted to combining more than one of the previous approaches. Hybrid recommender systems are still widely used by big companies like Amazon [1],Spotify [2] and Netflix [3].

### 1.1.6 Evaluation Metrics

Evaluation metrics are bla

**Offline Evaluation**

- Accuracy

- Precision

- Recall

**Online Evaluation**

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

[kopi - peyst]
More information in research part

### 1.1.7 User Satisfaction

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

## 1.2 Interest in Recommender Systems

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

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

## 1.3 Motivation

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

[Also explain job recommender if we do that]

## 1.4 Solution

How I solve the problem

# 2 Review of Literature and Research

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 in 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. The general principle of content-based methods is to identify the common characteristics of items that have received a favorable rating from a user, and then suggest to this user new items 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 occurs when the system has a limited amount of information on its users or the content of its items. This 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].

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 limitations of content-based methods. With collaborative filtering, the recommender system can still suggest items to users, even though the system doesn't 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].

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, 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, which is called the cold-start problem [DK11].

## 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 could be a star rating or a thumbs up/down action. The second is the implicit feedback, which is harder to collect. It represents clicks, browsing history, mouse movements and other ways of input that the users don't send explicitly.

## 2.3 Collaborative Filtering

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

Collorative filtering recommenders need the interactions between users and items. Using these information, 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.

[TODO maybe matrix image]

Since we haven't 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].

## 2.4 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 basic process of producing content-based

recommendations consists in matching up the attributes of the target user profile, in which preferences and interests are stored, with the attributes of the items. 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 from the item description. The content extracted from metadata is often too short and not sufficient to correctly define the user interests, that's why textual features are also added [De +15].

### 2.4.1 Overview of Content-Based Recommender Systems

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

The high level architecture of a content-based recommender system is shown in2.1. The recommendation process is performed in three steps, each of which is handled by a separate component:
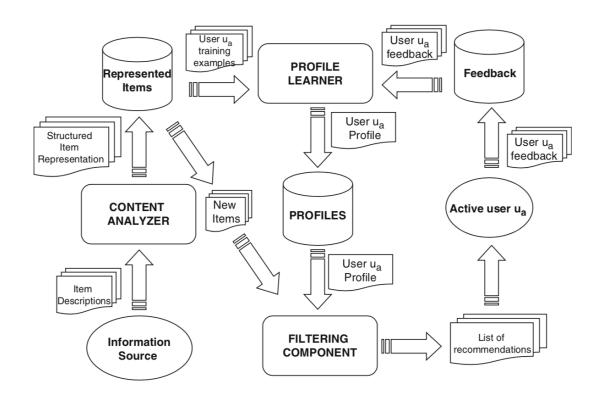


Figure 2.1: High level architecture of a content based recommender [De +15].

- CONTENT ANALYZER—The first part in 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 the advantage on enhancing the models with the human 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 constantly 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 a content to feature vector loses information or some misunderstatings 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 a content.

- OVER-SPECIALIZATION — Since the the content-based recommenders suggest items based on the past behaviour, 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 means that the users need some history first, so that the content-based recommenders work.

[Maybe summary]

## 2.5 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. These information are compared with the

items and a recommendation list is returned. More information can be found in the resources [Bur00] and [Fel+15].

## 2.6 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 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 shoud 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 shouldn't be measured, but whether the change in property has increased their satisfaction. Like other user studies, it is advantageous, when the participants don't 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.

### 2.6.1 Accuracy

In this section some equations to calculate the are presented. To understand them better, the notation used is also explained briefly.

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, $R$ depicsts 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 a element of the set like, dislike or other values depend on the implementation. Also, we suppose that no more than one rating can be made by any user $u \in U$ for a particular item $i \in I$ and set $r_u i$ this rating. To identify the subset of users that have rated an 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 [SG11].

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 with the goal of learning 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 and the author of this thesis used them as the cost function of the neural networks[TODO imp supervised] [SG11].

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

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

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:

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

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

A drawback of 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):

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

, which is used to evaluate the success of the company recommender[SG11].[TODO if necessary add other source + ref to implementation]

### 2.6.2 Coverage

Coverage can also be an important evaluation property for some recommender systems. The aim of maximizing the coverage would be making sure that a big portion of items are recommended by the recommender system. This might be especially important for recommender systems of e-commerce systems, so that different range of products get sold [SG11].

### 2.6.3 Confidence

Confidence in context of recommender systems refers to the system's trust into it's own suggestions. As we don't use this property in the implementation part, we also go into detail [HKR00].

### 2.6.4 Trust

Opposite to the confidence that was explained in the subsection 2.6.3, trust refers to user's trust into the recommender system. More information on this can be found in other sources [SG11].

### 2.6.5 Novelty

Novel recommendations are recommendations for the items that the user didn't know about before [SG11]. A simple solution to increase this would be filtering out the items for every user, which they interacted before.

### 2.6.6 Serendipity

Serendipity also known as unexpectedness depicts how surprising the recommendation results are. Although serendipty and novelty may sound similar in the beginning, items that are the similar to what the user already saw wouldn't be surprising at all. That's why, filtering out the items that the user already saw woudn't be a solution [SG11].

### 2.6.7 Diversity

Diversity is a property that ensures the recommended items are not similar to each other. As this is the main metric that we focus on, there is a special section just for this topic [See section 2.7].

### 2.6.8 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 their revenue. Generally, the utility touches on any system or user gain with the help of the recommender system [SG11].

### 2.6.9 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 big role [SG11].

### 2.6.10 Robustness

Robustness in recommender systems explain how strong the system is to unwanted mofications on the system. For example. if an hotel owner can inject fake positive reviews to the system to make their hotel get recommended, then the system is not robust enough [SG11].

### 2.6.11 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 everone can interact with the system and get suggestions without worrying about their feedback being public [SG11].

### 2.6.12 Adaptivity

Adaptivity in recommender systems try to give relevant recommendation even if the the trends change. For example, for a news recommender system, the system should be able to recommend news about earthquake, in case of one happens, even though the user doesn't have previous interest in earthquakes [SG11].

### 2.6.13 Scalability

It may also be important for recommender systems to be performant. For production systems with millions of items, the recommender shoud still be able to suggest new items without any latency [SG11].

### 2.6.14 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.

## 2.7 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 percieved as important 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 wider perspective instead of only focusing on accuracy [MRK06]. The 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 perfect guess for the users. However, user interests are complex, dynamic, context-dependent, heterogeneous and contradictory. That's why prediction of user needs just by looking at accuracy is a difficult task and may lead to decreased user satisfaction. Diversity can be a good 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 generalle desirable, as it increases the user satisfaction. According to the 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 persinalization which causes a filter buble, meaning that there is a healty level of diversity required [Ngu+14]. The increase of user satisfaction also leads to increased activity, revenues and customer loyalty.

In 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 wouldn't be too beneficial for the recruiters, since they would like to have a team that can tackle a wide range of topics.

### 2.7.1 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 has 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]:

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

As the name suggests, it's aim is 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 are calculated. The results gives us the diversity. This method is also employed in the evaluation part of the implementation and cosine similiarity 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 property that can lead to user satisfaction. The formula below is the way to calculate it for a user:

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

where

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

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:

$$Unexp = |R - EX| / |R|$$

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 programatically used in the scope of this thesis, they are not presented because of space reasons.

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

$$Aggdiv = \left| \bigcup_{u \in U} R_u \right|$$

**Gini Index**

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

, where

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

To calculate the gini index, the interacted items are first sorted in an ascending order by the chosen probability, which is the second equation above. In that equation, $k$ is the number of the least recommended item. In the end, a gini index 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**

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

Shannon Entropy is a logically similar to the *gini index* and they are calculated similarly. In the end, if the entropy is zero, a single item was always chosen and if the entropy is $log2(\#items)$, then each item is equally chosen [CHV15].

[TODO if some place left, add IUD, ISD, embeddings and NN maybe]

### 2.7.2 Diversity Enhancement Approaches

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

### 2.7.3 Summary

In scope of this chapter, we gave detailed information about the types of recommender systems with focus on the content-based recommendation. Then, we listed the evaluation properties that can lead to user satisfaction. Lastly, we disclosed the property that we chose, which is the diversity and illustrated how diversity can be calculated.

# 3 Implementation

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

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

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

## 3.1 Datasets

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

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

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

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

### 3.1.1 Freelancer Dataset
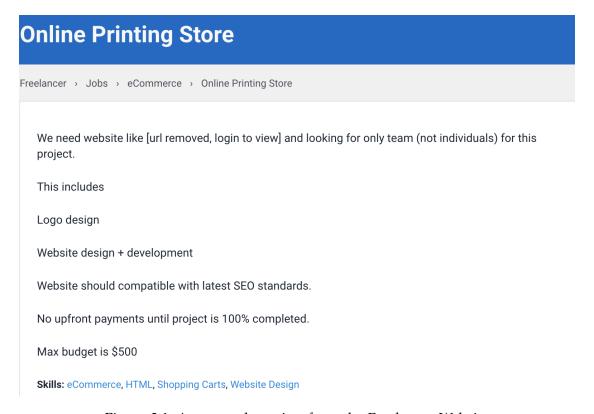


Figure 3.1: An example project from the Freelancer Website

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

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

🏅 **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** 🇮🇳

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 ]

$515 USD in 10 days

4.5 ★★★★⯨ (205 Reviews)

8.2 💲▮▮▮▮▮▮▮▯

**Bidder 2** 🇮🇳

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

$500 USD in 15 days

4.7 ★★★★★ (138 Reviews)

7.8 💲▮▮▮▮▮▮▮▯

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

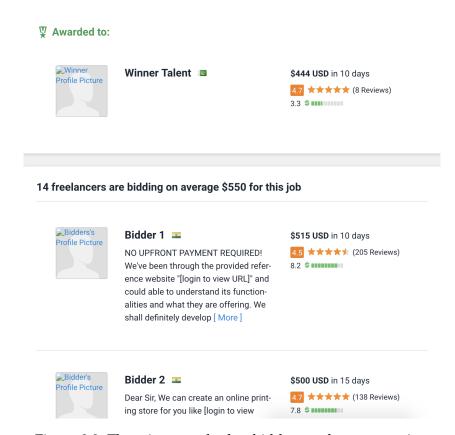their star rating until the time of bidding, amount of reviews they received and their total earnings until that date. In this thesis, we only consider the money they demand, their star rating and the number of reviews. For the sake of simplicity, we don't use the motivation text.

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

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

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

**My Top Skills**

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

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

### 3.1.2 Company Dataset

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

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

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

## 3.2 Unsupervised Individual Recommender

### 3.2.1 Recommendation by Similarity

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

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

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

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

### 3.2.2 Recommendation by Popularity

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

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

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

### 3.2.3 Hybrid Recommendation

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

## 3.3 Supervised Individual Recommender

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

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

### 3.3.1 Using Sparse Input

| user_url | .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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Talent 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Talent 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Talent 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Talent 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Talent 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Talent 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Talent 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Talent 8 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Talent 9 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Talent 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

10 rows × 780 columns

Figure 3.4: The talent skill matrix from freelancer.com

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

$$z = (x - u)/s \tag{3.2}$$

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

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

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

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

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

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

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

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

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

For each of the inputs, a dense layer exists with the number of neurons equal to the number of features. Making this decision means that project and talent layers contain 780 neurons and profile information layer contains four neurons. For these layers, we use *relu* activation, *l1* regularization with the value *0.0001* and we initialize the
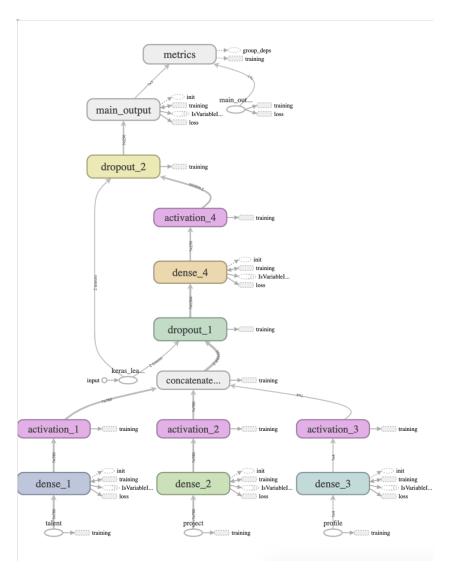
Figure 3.6: The graph that explains the sparse input model

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

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

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

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

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

### 3.3.2 Using Embeddings

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

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

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

**Preprocessing**

For both Freelancer and Motius data, we know the skill levels of projects and talents for various skills. Instead of having some positive and hundreds of zero skill values for each project/talent, we can set a skill threshold. This threshold implies skills above or equals to the limit are positive, and the rest are zero. The idea is converting the talent-skill and project-skill matrices so that, each talent/skill holds a list of skills they know/require. However, another constraint that needs to be addressed is the maximum length of the padded skill matrix, because neural networks require a fixed input shape. Therefore, the talent/project maximum amount of positive skills is determined. For
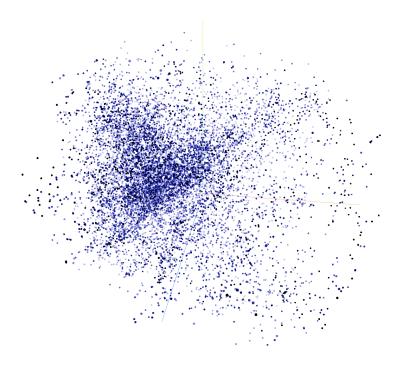
Figure 3.7: 3D version of the embedding space that is created for projects of this thesis

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

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

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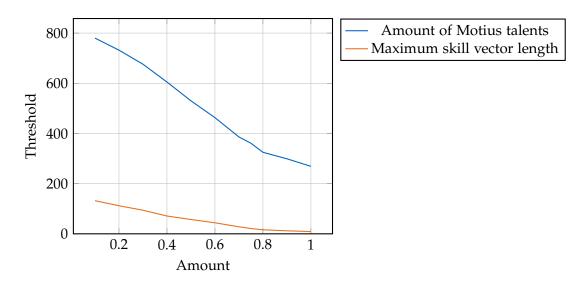
Figure 3.8: Effect of threshold selection on talents and maximum skill vector length

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

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

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

Figure 3.9: Training data that contains padded embedding skill vectors

**Simpler Architecture for Company Dataset**

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

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

## 3.4 Unsupervised Group Recommender

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

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

### 3.4.1 Baseline

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

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

### 3.4.2 Diverse

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

$$
\begin{aligned}
&R \leftarrow \varnothing \\
&while\ |R| < k\ : \\
&\quad i* \leftarrow \arg\max_{i \in C-R} g(R \cup \{i\}, \lambda) \\
&\quad R \leftarrow R \cup \{i*\} \\
&end\ while \\
&return\ R
\end{aligned}
\tag{3.4}
$$

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

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

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

## 3.5 Supervised Group Recommender

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

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

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

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

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

## 3.6 Group Recommendation using clustering

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

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

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

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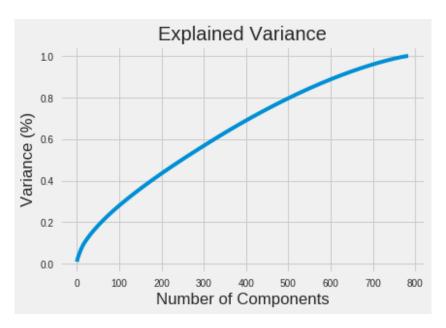
Figure 3.10: Explained variance ratio for difference number of PCs is shown.

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

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

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

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

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

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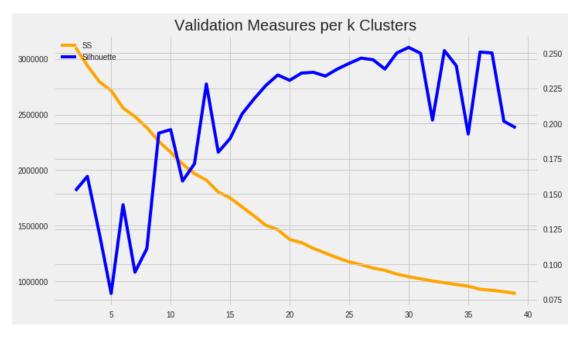


Figure 3.11: Silhouette scores of many different *k* values of k-means

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

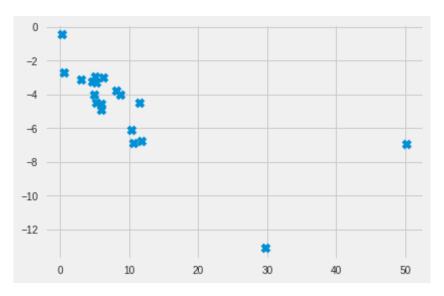After we exercise clustering, we can start with the group recommendation process.

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

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

| | .NET | 2D Animation | 360-degree video | 3D Animation | 3D Design | 3D Model Maker | 3D Modelling | 3D Printing | 3D Rendering | 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 3.13: Examples of some centers of clusters that are projected on a 2D space

## 3.7 Dashboard to show data and enter Feedback

### 3.7.1 General Dashboard

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

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

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

Figure 3.14: Main screen of the dashboard

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

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



| a |
| --- |
| a.i. & software engineer |
| aerodynamics engineer |
| ai dev |
| alexa engineer |
| algorithm dev |
| android & nfc developer |
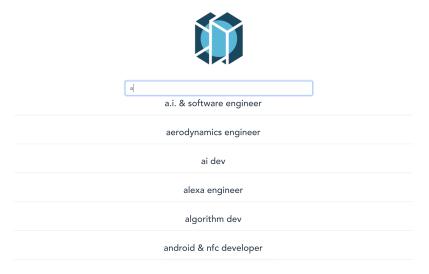
Figure 3.15: A snippet from the list of all projects that start with the letter *a*

### 3.7.2 Individual Recommendations

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

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

similarity to come up with new predictions.



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

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

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

### 3.7.3 Group Recommendations

The dashboard also can perform group recommendations. For the case of group recommenders, the scores can be from baseline neural networks, baseline cosine
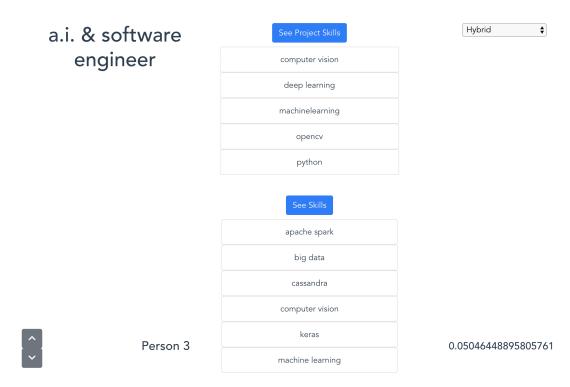
### a.i. & software engineer

| | |
|---|---|
| See Project Skills | Hybrid ⬍ |

computer vision

deep learning

machinelearning

opencv

python

See Skills

apache spark

big data

cassandra

computer vision

keras

machine learning

⌃
⌄

Person 3

0.05046448895805761

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

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

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

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

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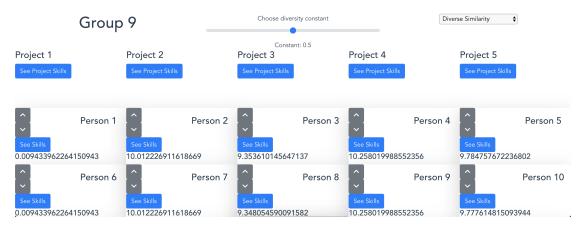


Figure 3.18: A screenshot from the list of all recommendations from diverse cosine similarity for the group 9

## 3.8 Improvement of Recommendations via Feedback Learning

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

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

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

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

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

## 3.9 Summary

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

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

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

# 4 Evaluation

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

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

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

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

While we are working on our hypothesis, we use all three types of experiments; offline, online and user studies [SG11]. Offline evaluation serves the purpose of calculating the accuracy and diversity. This type of experiments don't involve human feedback and are calculated via algorithms. When we aim for more advanced metrics, like user satisfaction, we have to rely on other types of experiments like user studies; we ask show users the results and ask their opinion about overall list value. Lastly, we also employ online experiments by allowing them to intereact 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 succesful experiment; a *hypothesis*, *controlling variables* and *generalization power*. Our hypothesis is that the diversity increases user satisfaction and serves as an important factor near accuracy. The controlling variables are the factors that stay the same during different experiments. We need those variables to make sure that different results are comparable. In our case, the trained dataset is the same. Also,

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

## 4.1 Unsupervised Individual Recommendation

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

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

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

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

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

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

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

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

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

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

### 4.1.1 Existing Company Recommender

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

**Offline Evaluation**

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

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

### 4.1.2 Recommendation by Similarity

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

**Offline Evaluation**

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

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

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

| A | B | C | D |
|------|------|------|------|
| 0.28 | 0.36 | 0.31 | 0.35 |

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

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

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

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

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

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

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

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

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

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

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

**Online Evaluation**

[TODO]

### 4.1.3 Recommendation by Popularity

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

**Offline Evaluation**

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

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

**Online Evaluation**

### 4.1.4 Hybrid Recommendation

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

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

| Similarity | Popularity | Hybrid |
|---|---|---|
| 4987 | 20433 | 14152 |

**Offline Evaluation**

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

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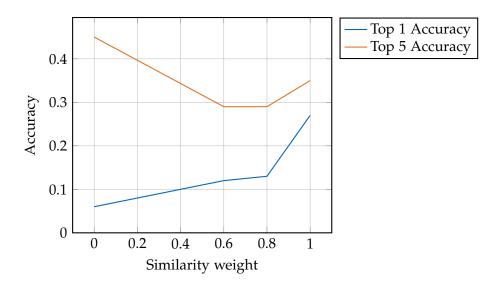
Figure 4.1: Effect of weight selection on hybrid recommenders to the accuracy

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

**Online Evaluation**

## 4.2  Supervised Individual Recommender

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

### 4.2.1  Using Sparse Input

Since the sparse data is too big for computers to handle, this one only contains offline evaluation results. The performance of this subtype of recommender is measured for different amount of layers, different cost functions, other activation functions and various optimizations. In the end, it wouldn't make sens to visualize all of the distinct results. However, we must note the maximum accuracy that is measured with the test set is 79%. Here the accuracy doesn't reveal top 1 or top 5 prediction that we used before. This prediction with the test set demonstrates that the model is able to predict
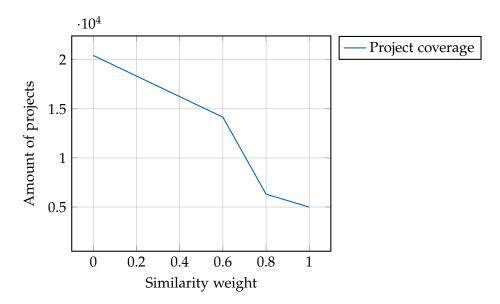
Figure 4.2: Effect of weight selection on the amount of projects that we can generate predictions.

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

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

### 4.2.2 Using Embeddings

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

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

**Offline Evaluation**

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

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

**Online Evaluation**

## 4.3 Unsupervised Group Recommender

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

### 4.3.1 Baseline Recommender

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

**Offline Evaluation**

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

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

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

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

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

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

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

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

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

**Online Evaluation**

### 4.3.2 Diverse Recommender

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

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

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

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

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

**Offline Evaluation**

When we evaluate a diverse recommender, we need to show variations of results for different $\lambda$ values. Because, every small or big modification of this value will have an effect on the accuracy, diversity or overall list value.
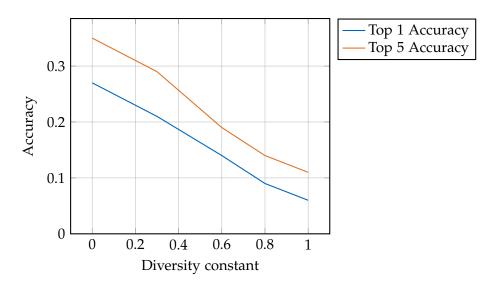
Figure 4.3: Effect of diversity constant on unsupervised group recommender to the accuracy

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

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

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

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

**Online Evaluation**

## 4.4 Supervised Group Recommender

The supervised group recommender takes advantage of neural networks to ascertain the relevancy of talents to the project and may use cosine similarity to ensure the variance of
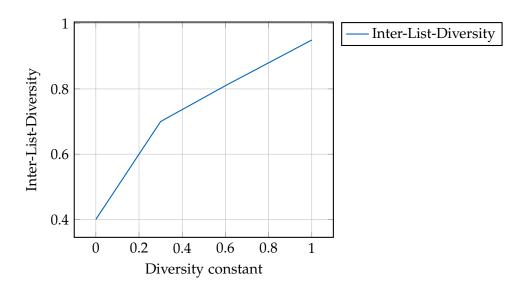
Figure 4.4: Effect of diversity constant on unsupervised group recommender to the diversity
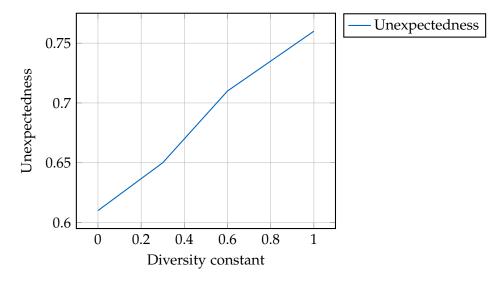


Figure 4.5: Effect of diversity constant on unsupervised group recommender to the unexpectedness

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

### 4.4.1 Baseline Recommender

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

**Offline Evaluation**

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

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

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

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

**Online Evaluation**

### 4.4.2 Diverse Recommender

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

**Offline Evaluation**

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

**Accuracy**   The figure 4.6 depicts the impact of diversity constant change to the accuracy, which are calculated on freelancer.com dataset.
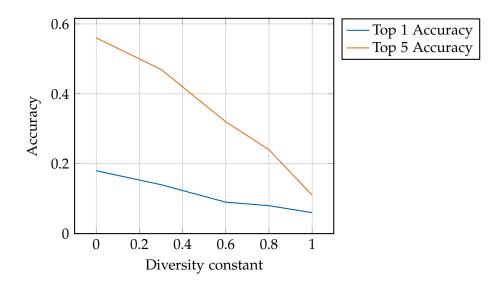
Figure 4.6: Effect of diversity constant on supervised group recommender to the accuracy

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

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

**Online Evaluation**

## 4.5  Feedback Loop

Improving the outcomes via a feedback loop is an essential part of this thesis. Like it was told before [See 3.8], the loop has a direct and an indirect effect. When the recruiters that use the dashboard, post feedback to talents, the recommendation list is instantly changed accordingly. This is called the direct effect. Posting feedback, also modifies the existing dataset that is used to train the model. Sending a positive feedback modifies the datapoint to a positive label, which is one. Also, sending a negative feedback modifies the datapoint to a negative label, which is zero. After that, the model will produce *better* results, when we retrain with the modified data.
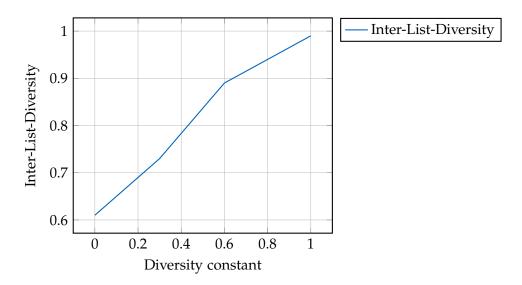
Figure 4.7: Effect of diversity constant on supervised group recommender to the diversity
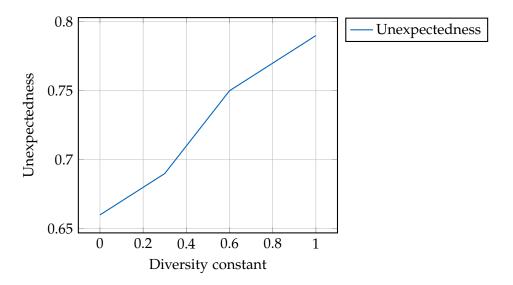


Figure 4.8: Effect of diversity constant on supervised group recommender to the unexpectedness

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

### 4.5.1 Online Evaluation and User Study

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

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

[TODO comparison of before and after]

## 4.6 Summary

# 5 Discussion

## 5.1 Problems about datasets

### 5.1.1 Problems about Motius dataset

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

To sum up:

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

### 5.1.2 Problems about Freelancer dataset

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

## 5.2 Problems about recommender type

Recommender systems based only on content generally suffer from the problems of limited content analysis and overspecialization [SM95]

## 5.3 Reasons to use hybrid recommender

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].

## 5.4 Comparison of Individual Recommenders

## 5.5 Comparison of Group Recommenders

## 5.6 Conclusion

bla
  bla
  bla
  bla

# 6 Conclusion

## 6.1 Introduction

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

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

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

Table 6.1: An example for a simple table.

| A | B | C | D |
|---|---|---|---|
| 1 | 2 | 1 | 2 |
| 2 | 3 | 2 | 3 |
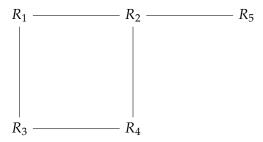


Figure 6.1: An example for a simple drawing.

## 6.2 Conclusion

Figure 6.2: An example for a simple plot.

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

Figure 6.3: An example for a source code listing.

# 7 Appendix

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

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

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

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

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

    return model
```

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

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

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

# List of Figures

# List of Tables

# 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 recom-
           mender 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 recom-
           mender 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. Breitinger. "Research-paper recommender
           systems: a literature survey." In: *International Journal on Digital Libraries*
           17.4 (Nov. 2016), pp. 305–338. ISSN: 1432-1300. DOI: 10.1007/s00799-015-
           0156-0.

[BKL09]    S. Bird, E. Klein, and E. Loper. *Natural language processing with Python:
           analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[BOH15]    R. Burke, M. P. O'Mahony, and N. J. Hurley. "Robust collaborative rec-
           ommendation." In: *Recommender systems handbook*. Springer, 2015, pp. 961–
           995.

[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.

[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.

[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.

[DG06]     J. Davis and M. Goadrich. "The relationship between Precision-Recall and ROC curves." In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 233–240.

[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.

[Gri18]    M. Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[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.

[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.

[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.

[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.

[Pog+17]    T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review." In: *International Journal of Automation and Computing* 14.5 (2017), pp. 503–519.

[Rou87]     P. J. Rousseeuw. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis." In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65.

[RRS15]     F. Ricci, L. Rokach, and B. Shapira. "Recommender Systems: Introduction and Challenges." In: *Recommender Systems Handbook*. Ed. by F. Ricci, L. Rokach, and B. Shapira. Boston, MA: Springer US, 2015, pp. 1–34. ISBN: 978-1-4899-7637-6. DOI: 10.1007/978-1-4899-7637-6_1.

[SA13]      R. Sathya and A. Abraham. "Comparison of supervised and unsupervised learning algorithms for pattern classification." In: *International Journal of Advanced Research in Artificial Intelligence* 2.2 (2013), pp. 34–38.

[SEK04]     M. Steinbach, L. Ertöz, and V. Kumar. "The challenges of clustering high dimensional data." In: *New directions in statistical physics*. Springer, 2004, pp. 273–309.

[SG11]      G. Shani and A. Gunawardana. "Evaluating recommendation systems." In: *Recommender systems handbook*. Springer, 2011, pp. 257–297.

[SM01]      B. Smyth and P. McClave. "Similarity vs. diversity." In: *International conference on case-based reasoning*. Springer. 2001, pp. 347–361.

[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).