Ing. Fabian Henger, BSc

Content-Based Recommendation for Real Estates

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Helic, Denis, Assoc.Prof. Dipl.-Ing. Dr.techn.

Institute of Interactive Systems and Data Science
Head: Lindstaedt, Stefanie, Univ.-Prof. Dipl.-Inf. Dr.

Graz, January 2019

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Danksagung

Mit dem Abschluss dieser Arbeit und der darauffolgenden Masterprüfung beginnt ein neuer Lebensabschnitt für mich. Ich kann es kaum glauben, dass diese Zeit nun zu Ende geht. Im Verlauf des Studiums durfte ich viele Menschen kennen lernen, die mich auf diesem Weg begleitet haben. Dafür möchte ich mich bedanken.

Ein besonderer Dank gilt meinen Eltern und meinem Bruder, die mir dieses Studium ermöglicht und mich dabei stets unterstützt haben. Es ist nicht selbstverständlich eine Möglichkeit wie diese im Leben wahrnehmen zu dürfen. Vielen Dank dafür! Von ganzem Herzen danke ich meiner Freundin, Carina, die mir immer Mut machte, mich in den langen Nächten motivierte und stets für mich da war. Ich kann dir nicht genug dafür danken!

Zu guter Letzt danke ich meinem Betreuer Herrn Assoc.Prof. Dipl.-Ing. Dr.techn. Helic und dem Auftraggeber dieser Arbeit Herrn Zangerl. Vielen Dank für euer Engagement und die gute Zusammenarbeit im Laufe dieser Arbeit!

Abstract

Searching for a real estate on broker applications especially on mobile applications and websites is challenging and time-consuming. This is due to the fact that the demand for real estates is higher than the supply and users are not able to specify their preferences on real estate broker applications. In this work we focus on a content-based recommender system for broker applications as filtering and ordering mechanism to ease the search for real estates. The system constructs a user profile which reflects the user's preferences with the use of real estate attributes (content) such as the price or footage of a flat. On the basis of this profile, the system calculates the most appropriate real estates and presents the results to the user in descending order – the first hit meets the user's preferences most while the least one at least. Furthermore, the system tracks the interactions of the user to improve its parameters and the user profile. Interactions in this case above all refer to liking and disliking real estates. A like indicates that the user likes a real estate and a dislike means the opposite. We applied our approach to manually searched real estates marked with likes and dislikes from a participant and achieved an accuracy of 80 % in the recommended set. This means that eight of ten recommended real estates were marked as liked by the participant. We were able to surpass our initially set goal of 70 % accuracy and thus made a significant contribution to the research on recommender systems for real estates.

Contents

List of Abbreviations

APH Analytical Hierarchy Process

API Application Programming Interface

CBF Content-Based Filtering

CF Collaborative Filtering

CS Cosine Similarity

ED Euclidean Distance

GCF Google Cloud Functions

GCP Google Cloud Platform

HR Hybrid Recommendations

IBCF Item-Based Collaborative Filtering

JS Jaccard Similarity

KBR Knowledge-Based Recommendation

LIB Less Is Better

MIB More Is Better

MSD Mean Squared Difference

NIB Nearer Is Better

NN Nearest Neighbours

PCC Pearson Correlation Coefficient

REST Representational State Transfer

RS Recommender System

UBCF User-Based Collaborative Filtering

List of Figures

Listings

List of Tables

1 Introduction

A statistic on the European Union shows a sustained upward trend of people moving from the country towards the city during the last few years (Statista, 2018). This trend is accompanied by the need for more real estates in cities and thus leads to an imbalance between supply and demand. A study about regional price indices in Austria describes that in the years from 2010 to the mid of 2017 the need for real estates was always higher than the available offerings (Mundt and Karin, 2017). Due to this fact, prices for real estates have kept on rising continuously ever since. Hence, purchasing or renting a real estate is getting expensive and problematic. Either there is no appropriate real estate available, or the prices are too high to afford the property (Geymüller and Christl, 2014). Especially students studying at university who want to live close to their university have a limited budget (Horstmann, 2017). These conditions make the search for a real estate challenging and time-consuming.

Since the Digital Age has widely spread across the world, many possibilities for the real estate market relating to the web arose. People are sharing flats via Facebook or Instagram, using websites such as willhaben.at or immobilienscout24.at, etc. instead of publishing their offerings in the newspaper. However, all of these platforms only offer filtering and sorting options to refine the search results of their users. This means the user defines a search criterion and sorting order for the application by selecting her or his preferences from a list of predefined attributes. Depending on the property, the user can also define specific values such as the minimal and maximal price for the rental fee seen in 1.1.

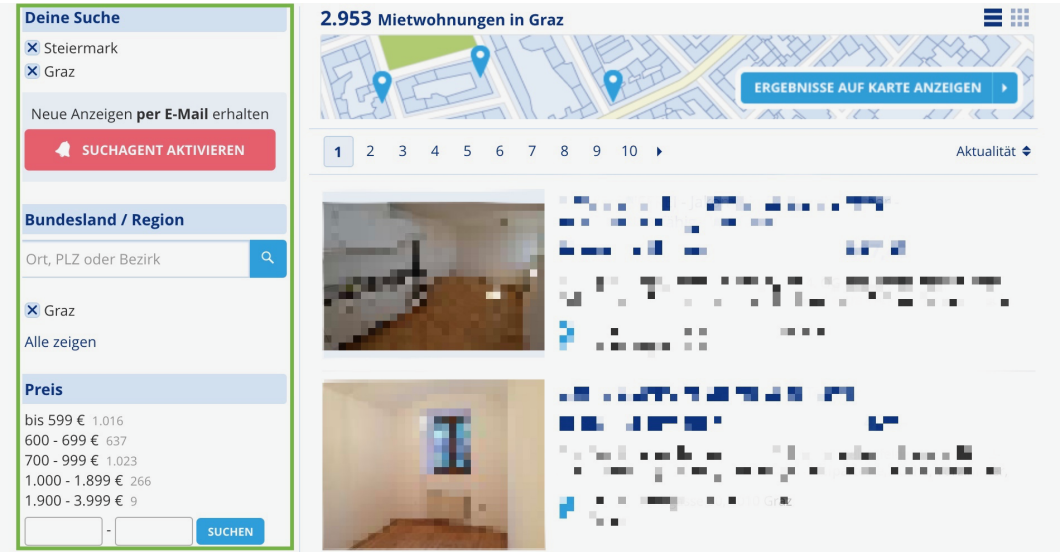

Figure 1.1: Example of filter options at willhaben.at
Source: Screenshot by author. (Willhaben, 2018)

In the course of this master's thesis we created a content-based recommender system for real estate to ease the time-consuming searching on broker applications. Therefore, the recommender system uses a user profile which reflects the user's preferences regarding real estate attributes for its similarity measurements. In the following chapters, we describe our approach and reached goals based on an analysis conducted on the application domain.

1.1 Goals

Three research questions formulate the goal of this work. The questions are as follows:

1. Is it possible to recommend real estates to a user based on real estate characteristics such as price or living space?
2. Is a content-based approach suitable to make real estate recommendations?

3. How can the user-behaviour represented by user likes and dislikes be used to adapt the parameters of the recommender system?

Furthermore, our goal is to implement a recommender system which achieves at least an accuracy of about 70 % for the real estates recommended to the user. Therefore, we consider likes as positive labels and dislikes as negative ones.

1.2 Customer

The Austrian start-up **Immoky**, headquartered in Graz and founded in the year 2017, wants to make the search for real estate easier and faster by providing an easy-to-use mobile application. According to Armin Zangerl, the CEO of Immoky, the key resources for a successful business model are good usability and an intelligent search mechanism for the application. Therefore, Immoky tries to keep the user interface as simple as possible. Armin Zangerl used the following words to describe the idea behind Immoky (Armin Zangerl, personal communication, October 17, 2018):

> *"Das Start-Up-Unternehmen „Immoky", welches im Frühjahr 2017 gegründet wurde, ist eine Immobilienplattform, welche die Suche nach der Traumimmobilie sowie die Kommunikation zwischen den Nutzergruppen durch einfache und innovative Features wesentlich erleichtert und mit einem benutzerfreundlichen User Interface überzeugt."*

The search for real estate in Immoky's mobile application is based on a tutorial in which the user has to answer a few questions. These answers represent the user's preferences and are further used for the filtering the real estate results. On the basis of these answers the system filters the available properties. In contrast to broker applications, Immoky uses swipeable cards for the presentation of a real estate as seen in figure 1.2.

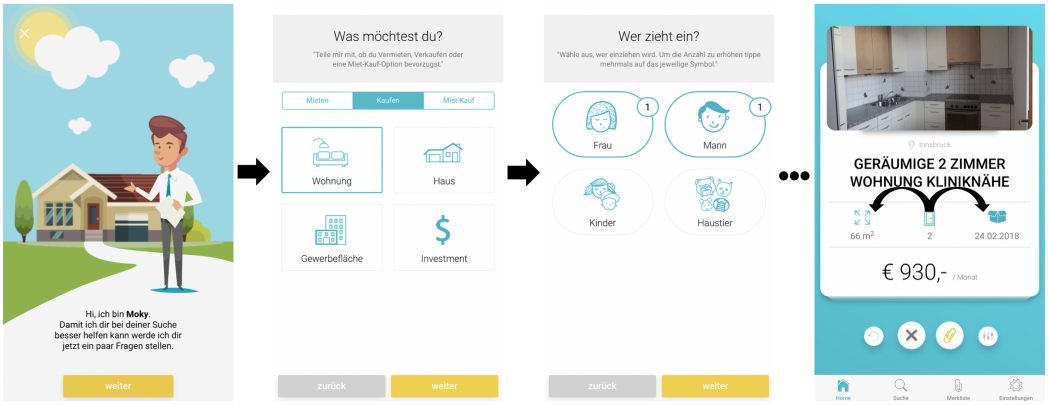

Figure 1.2: First questions of Immoky's tutorial
Source: Created by author based on screenshots from Immoky. (Zangerl, 2018)

A swipeable card is a screen which the user can swipe either to the right or to the left. Swiping the card to the right is a gesture that describes that the user likes a real estate while swiping it to the left indicates a dislike. The search mechanism of the application learns from these interactions and thus provides better search results. On the basis of this, Immoky identified two core requirements for its system. First, the search itself and second, the refinement of the search mechanism.

Immoky funded the project of this thesis to use the implemented recommender system as search engine for its application. Thus, when we write about the customer or requirements, we mean Immoky and the company's requirements regarding the recommender system.

2 Related Work

In this chapter, we conduct a research on the topics real estates, recommender systems (RS) and their knowledge discovery processes. On the basis of this analysis, we implement our approach. Furthermore, we examine whether there exist reasonable approaches for applying a RS to the real estate market.

2.1 Real Estates

Before approaching the technical part of this work, we analyse the term real estate, its categories and the real estate related broker websites. For the analysis regarding the websites, we focus on the Austrian market.

A real estate is a property consisting of a land and its affixed physical properties and improvements such as buildings on it (Merriam-Webster, 2018). In this work we consider a real estate to be an object that satisfies the need for permanent accommodation. This does not include real estate objects that are used for business and storage needs. Thus, we mainly deal with apartments and houses which are accompanied by different characteristics. On the basis of these characteristics, we can describe and translate a particular object to be usable for a RS (Menzies, 2014a).

2.2 Real Estate Web Search based on Property Characteristics

The internet has become a source for real estate brokers to publish their available real estatea on several real estate broker websites. Hence, many users are not able to reduce the effort for searching their desired accommodation (Zumpano, Johnson, and Anderson, 2003). This is due to the fact that standard searching mechanisms do not include a multitude of personal preferences. Instead, constraining search and order options are available to refine the search result. Thus, a stepwise and time-consuming procedure is necessary to find your desired real estate (Ho, Chang, and Ku, 2015). According to Bond et al., 2000, most of the real estate websites are using a few common characteristics as listing information to describe the advertised objects (seen in figure 2.1). On the basis of these characteristics the users are often not able to decide if the real estate meets their needs and preferences.

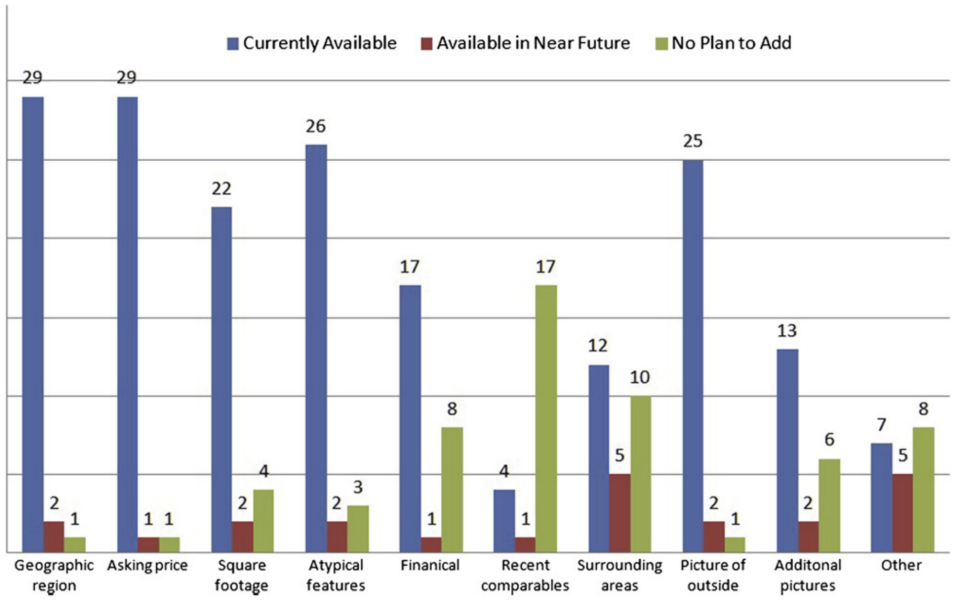

Figure 2.1: Listing information used on real estate websites
Source: Bond et al., 2000

Ho, Chang, and Ku, 2015 investigated the housing selection topic and introduced an Analytical Hierarchy Process (AHP) with three different layers as seen in figure 2.2. Layer one defines the overall goal, layer two the attributes used for the house selection and layer three is used for determining sub-attributes. Ho, Chang, and Ku, 2015 further group the attributes into housing value, structure, neighbourhood and location.

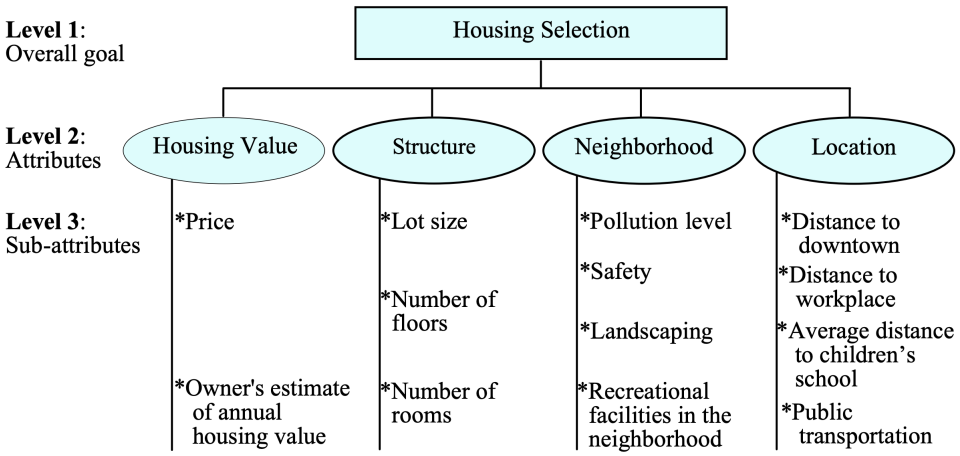

Figure 2.2: AHP for housing selection.
Source: Ho, Chang, and Ku, 2015

Since we focus on the Austrian market, we further investigate the most popular and most frequently used real estate websites in Austria. For the research we used the blog *10 beliebte Immobilienportale aus Österreich* from the year 2016 which is about ten popular websites for searching real estates (Leichtgemacht, 2016). From the sites listed, on this website we used immobilienscout24.at, findmyhome.at and willhaben.at for our research. The goal of this research is to determine the most common search features and listing information. This includes the most common features to adjust the search result and the detail view of a real estate.

We grouped the result into **search options**, **order options** and **detail view**. An x in a cell indicates that the option is available on the corresponding real estate website.

Search options	immobilienscout24.at	findmyhome.at	willhaben.at
Type of real estate	x	x	x
Type of payment	x	x	x
Location	x	x	x
Multiple locations		x	
Price (from/to)	x	x	
Square foot (from/to)	x	x	
Options (garden, balcony etc.)	x		x
Objecttypes (penthouse, garconniere etc.)			x
Text search	x		x

Table 2.1: Search options used on immobilienscout24.at, findmyhome.at and willhaben.at.

Order options	immobilienscout24.at	findmyhome.at	willhaben.at
Most recently adevertised			x
Price (ascending/descending)	x	x	x
Square foot (ascending/descending)	x	x	x
Location (zip ascending)	x	x	
Location (zip descending)		x	

Table 2.2: Order options used on immobilienscout24.at, findmyhome.at and willhaben.at.

Detail View	immobilienscout24.at	findmyhome.at	willhaben.at
Images	x	x	x
Description	x	x	x
General information	x	x	x
Detailed information	x	x	x
Infrastructure/surroundings		x	
Online contact details	x	x	x

Table 2.3: Detail view used on immobilienscout24.at, findmyhome.at and willhaben.at.

Based on the information stated in the tables 2.1, 2.2 and 2.3, we note that the websites used for our research also use search and order options to filter and adopt the search result. Additionally, all three websites use the same approach for their detail view. Only findmyhome.at, in addition to images, description, general information, detailed information and online contact, includes the rubrics infrastructure and surroundings. However, the user is not able to add preferences to the search mechanism to find an appropriate real estate more quickly.

2.3 Recommender Systems

After we had identified the common approaches of real estate websites in Austria regarding search mechanisms and object descriptions, we examined the core component of this work, the RS.

An RS is a decision support system which recommends items to users based on their preferences. An item is a particular object of the RS domain. This could be for example a book in an online bookstore, an article in a news-paper or a song in a music library. The design, the graphical user interface and core recommendation technique are all domain-related and customised to provide useful information for the user. A common approach is to use an RS as a ranking system which presents the most suitable items to the user in descending order. For the calculations and improvements it is necessary to collect information about the interactions of a user. Interactions are for example ratings of several products or the navigation to a particular view in the system's user interface. (Ricci, Rokach, and Shapira, 2015)

Depending on the domain and importance of the accuracy there are several approaches for RS. However, the three major recommendation approaches are collaborative filtering (CF), content-based filtering (CBF) and knowledge-based recommendation (KBR). Furthermore, there are combinations of the major methods which are called hybrid recommendations (HR). (Felfernig, Jeran, et al., 2014)

2.3.1 Collaborative Filtering

RS which are based on the CF approach use ratings of a user without having any information about the item (Koren and Bell, 2015). Schafer et al., 2007 state that the term collaborative filtering has been existing only for a few decades, but people have been sharing opinions with others since centuries which is the basic idea behind CF – the sharing of opinions with other people.

Most CF RS use a rating matrix of dimension $m * n$ where m is the number of users and n is the number of considered items (see table 2.4). A number in a cell represents the rating of a user of the corresponding item. As already mentioned, a CF RS does not use any information of an item such as its content or metadata. Thus, the similarity measurements are only based on the item ratings of all users represented in the stated rating matrix. (Hernando, Jesús Bobadilla, and Fernando Ortega, 2016)

	Item 1	**Item 2**	**Item 3**	**Item 4**	**Item 5**
Dem. User	5	2	-	5	4
User 1	-	3	-	4	5
User 2	1	5	2	-	1
User 3	-	-	3	2	5
User 4	1	2	3	1	4
User 5	2	1	-	3	-

Table 2.4: Rating matrix

There are two different types of CF: memory-based collaborative filtering and model-based filtering (Koohi and Kiani, 2016). In this work, we only focus on memory-based CF since a model-based CF determines a model such as a Bayesian classifier to predict ratings instead of similarity measurement methods based on historical data such as user ratings which we propose to use for our approach (J. Bobadilla, F. Ortega, et al., 2013).

The ranking calculation of the memory-based approach relies on a specific amount of nearest neighbours (NN) which are either users or items. Depending on the type of NN there are two different CF approaches: User-Based Collaborative Filtering (UBCF) and Item-Based Collaborative Filtering (IBCF). (Felfernig, Jeran, et al., 2014)

User-Based Collaborative Filtering

UBCF RS identify a set of NN which are, in case of this approach, users with similar preferences/ratings to provide recommendations for the demanding user. From these ratings, the RS predicts the rating for an unrated item. Therefore, a specific amount of neighbours who have already rated the target item is chosen from the NN set. (Cai et al., 2014)

An RS determines the set of NN from the similarity between users and their item ratings. Koohi and Kiani, 2016 state that there are two ways to measure similarity: traditional similarity measures and clustering algorithms. A commonly used method is the Pearson correlation coefficient (Liu and Lee, 2010) (Jannach et al., 2010) (J. Bobadilla, Serradilla, and Bernal, 2010) (Felfernig, Jeran, et al., 2014) (Koohi and Kiani, 2016); see equation 2.1.

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \overline{r_a})(r_{b,p} - \overline{r_b})}{\sqrt{\sum_{p \in P}(r_{a,p} - \overline{r_a})^2}\sqrt{\sum_{p \in P}(r_{b,p} - \overline{r_b})^2}} \tag{2.1}$$

The Pearson correlation coefficient 2.1 calculates the similarity between the rating vectors of the users a and b. P is the set of items which has been rated by the demanding user and all users of the NN. Ratings are indicated by the variable r. Thus, $r_{a,p}$ and $r_{b,p}$ are ratings of the item p and $\overline{r_b}$ and $\overline{r_b}$ are average ratings. The results of this equation can be between -1 and 1 where -1 indicates contrary vectors and 1 identically ones.

The final step of a CF RS is to predict ratings for unrated items. Therefore, it takes a defined amount from the NN set who has already rated items the demanding user has not rated yet. There are several methods to estimate the rating for an unrated item. However, a common approach is to calculate the

weighted average from the neighbours' ratings (Liu and Lee, 2010) (Koohi and Kiani, 2016); see equation 2.2. The weighting is defined by the similarity of the demanding user and the neighbour.

$$pred(a,p) = \overline{r_a} + \frac{\sum_{b \in NN} sim(a,b)(r_{b,p} - \overline{r_b})}{\sum_{b \in NN}(sim(a,b)} \tag{2.2}$$

As mentioned, equation 2.2 predicts the rating for an item p which has not been rated yet by the demanding user a. It approximates the rating by using the ratings $r_{b,p}$ and average ratings $\overline{r_b}$ from the neighbours b. Due to the fact that some users are more general than others, equation 2.2 takes the average rating $\overline{r_a}$ of the demanding user a as the basis for the prediction.

The RS executes the previous step multiple times and represents the items for example in descending order where the item listed first is the most similar one and the item listed last is the least similar result (Ricci, Rokach, and Shapira, 2015).

Item-Based Collaborative Filtering

In contrast to UBCF, an IBCF based RS uses the similarity between items instead of users (Cai et al., 2014). Thus, NN are in case of this approach items with high similarity to each other which were already rated by the demanding user (Kim et al., 2010). The remaining steps are similar to those of the UBF approach.

For both approaches, UBCF and IBCF, the same calculation methods for the similarity measurement and rating prediction can be used (Cai et al., 2014) (Felfernig, Jeran, et al., 2014). The similarity measurement is an essential aspect of an accurate RS since it is responsible for determining the most similar users or items which are fundamental for the rating prediction later on. Cai et al., 2014 states that, *"Pearson correlation coefficient, cosine-based similarity, vector space similarity, and so on are widely used in similarity measurement in CF methods"*.

2.3.2 Content-Based Filtering

CBF RS rely on the fact that users have monotonic preferences regarding a certain topic. For example, a user who in general positively rates fantasy movies will also like fantasy movies in the future. (Felfernig, Jeran, et al., 2014)

As the name suggests, a CBF based RS predicts ratings for items based on their content. The basic idea of this approach is to create a user profile from the content of liked items which reflects the user's preferences as item properties (Cai et al., 2014). A CBF RS includes two basic steps: first, the determination of the user profile; second, the rating prediction of items based on similarity measurements with the user profile. Figure 2.3 shows the process and data flow of a CBF based RS..

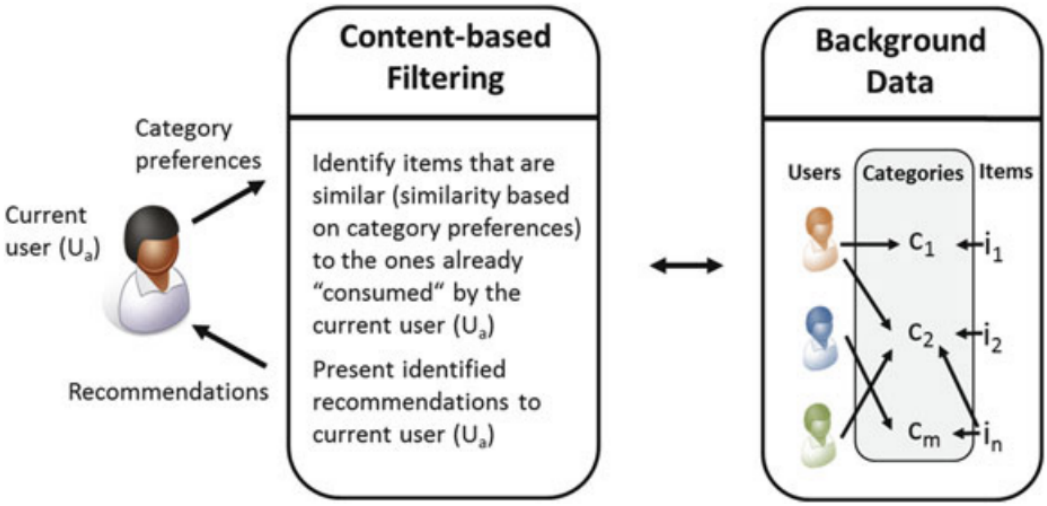

Figure 2.3: Dataflow of a CBF RS.
Source: Felfernig, Jeran, et al., 2014

One of the fundamental questions which appears in several literary works is (Pazzani and Billsus, 2007) (Barragáns-Martínez et al., 2010) (Felfernig,

Jeran, et al., 2014): *What is content?*. According to Felfernig, Jeran, et al., 2014, there are two types of content. First, the item content description such as the text of an article which is processed to determine keywords. Second, the metadata of an item called categories such as the genre of a movie. These keywords and categories are further used for the similarity measurement between the user and items. In this work, we only focus on the categories type since the available real estates we are using in this work includes mainly predefined properties.

A CBF based RS also builds a matrix for its recommendation calculations. In the case of this approach, the matrix includes m items and n different categories also called features (Pazzani and Billsus, 2007). We used an example with movies to visualise the format of a simple item/feature matrix as can be seen in table 2.5.

	Title	Genre	Director	Actors	Year
Item 1	Lord of the Rings	Fantasy	Peter Jackson	Elijah Wood, Ian McKellen, Orlando Bloom	2001
Item 2	Harry Potter	Fantasy	Chris Columbus	Daniel Radcliffe, Rupert Grint, Richard Harris	2001
Item 3	Titanic	Drama	James Cameron	Leonardo DiCaprio, Kate Winslet, Billy Zane	1997
Item 4	Pulp Fiction	Crime	Quentin Tarantino	John Travolta, Uma Thurman, Samuel L. Jackson	1994
Item 5	Gladiator	Action	Ridley Scott	Russell Crowe, Joaquin Phoenix, Connie Nielsen	2000
Item 6	Avatar	Adventure	James Cameron	Sam Worthington, Zoe Saldana, Sigourney Weaver	2009

Table 2.5: Item matrix of movies

Coming to the detailed description of the two steps mentioned before: first, we investigate the user profile and afterwards determine similar items regarding the user's preferences.

There are several ways to create a user profile. Pazzani and Billsus, 2007 describe so-called user customisations the user can use to define one's preferences. Therefore, a user interface has generally different types of input controls such as checkboxes or input boxes for arbitrary texts. An issue

which is accompanied by this approach is the time-consuming definition of the user's preferences before one receives any recommendations. However, there arise possible synergies between the Austrian real estate market and this type of approach. As we described in section 2.2, most real estate broker websites define search and order options to refine a search result which we can use to build an initial user profile for the RS.

Another common approach is to create and update the user profile from the user's interactions such as visiting a detail view, buying a product or liking/rating an item Pazzani and Billsus, 2007 (Felfernig, Jeran, et al., 2014) (Gemmis et al., 2015). Such interactions are called feedback (Gemmis et al., 2015). Furthermore, Gemmis et al., 2015 states that it is necessary to distinguish between positive and negative feedback since positive feedback generally reflects the user's preferences.

As example we use the categories from 2.5 and expand them with a *liked* flag as can be seen in table 2.6.

	Title	Genre	Director	Actors	Year	Liked
Item 1	Lord of the Rings	Fantasy	Peter Jackson	Elijah Wood, Ian McKellen, Orlando Bloom	2001	Yes
Item 2	Harry Potter	Fantasy	Chris Columbus	Daniel Radcliffe, Rupert Grint, Richard Harris	2001	Yes
Item 3	Titanic	Drama	James Cameron	Leonardo DiCaprio, Kate Winslet, Billy Zane	1997	No
Item 4	Pulp Fiction	Crime	Quentin Tarantino	John Travolta, Uma Thurman, Samuel L. Jackson	1994	Yes
Item 5	Gladiator	Action	Ridley Scott	Russell Crowe, Joaquin Phoenix, Connie Nielsen	2000	Yes
Item 6	Avatar	Adventure	James Cameron	Sam Worthington, Zoe Saldana, Sigourney Weaver	2009	No

Table 2.6: Item matrix of movies with *liked* flag

For the user profile determination, we go through all liked movies step by step, analyse their categories and add a keyword to the corresponding category from the user if it is not included yet. This process results in a vector which looks like an item from table 2.5. For example, the content of the Genre category includes the keywords $< Action, Adventure, Crime, Fantasy >$.

Based on the extracted keywords which are assigned to the corresponding categories, the next step is to determine the similarity between the items and the user profile. Depending on the type of content there are several similarity measurement functions (Van Le, Nghia Truong, and Vu Pham, 2014). For our movie example we use the dice-coefficient (see equation 2.3) (Felfernig, Jeran, et al., 2014).

$$sim(u,i) = \frac{2 * categories(u) \cap categories(i)}{categories(u) + categories(i)} \tag{2.3}$$

The dice-coefficient 2.3 is based on quantity functions. Therefore, $categories(u)$ and $categories(i)$ define the amount of unique elements in the category of the user u and item i. After the equation is applied to multiple unrated items, a list of ranked items can be determined (ordered by similarity) and provided to the user.

2.3.3 Knowledge-Based Recommendation

The last approach of the three major recommendation methods is called Knowledge-Based Recommendation (KBR). Traditional RS such as CF and CBF approaches often have issues with complex items such as cars. KBR systems are built to tackle these problems (Felfernig, Friedrich, et al., 2015). KBR RS use a knowledge base about a specific domain, its users, the product assortment and the user preferences to provide appropriate items (Mandl et al., 2011). (Ricci, Rokach, and Shapira, 2015) (Lopes Rosa et al., 2018). The determination of the user and the presentation of the recommendation result are embedded in an iterative process (Mandl et al., 2011) (see figure 2.4).

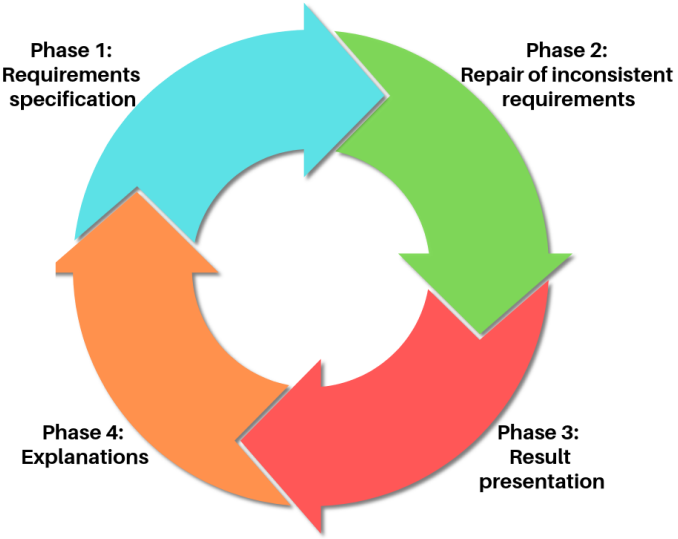

Figure 2.4: Iterative process of a KBR
Source: Created by author based on Mandl et al., 2011

As mentioned and seen in figure 2.4, Mandl et al., 2011 uses a four-step phase to describe the basic cycle of an KBR.

1. **Requirements specification:** In the first phase the system collects the preferences on the basis of the user's interactions.

2. **Repair of inconsistent requirements:** In the second phase the system executes repair actions if preferences could not be identified or were insufficient to determine appropriate items for the demanding user.

3. **Result presentation:** Depending on the domain, in the third phase, the recommendation result is presented to the user, for example as an ordered list.

4. **Explanations:** The fourth phase is used to explain to the user why the items were identified as an appropriate choice during the recommendation process.

Based on Smyth, 2007, Felfernig, Jeran, et al., 2014 and Felfernig, Friedrich, et al., 2015, we further distinguish, between constraint-based and case-based KBR systems.

Constraint-Based KBR

Constraint-Based KBR systems rely on explicitly defined *constraints* representing the user's requirements such as the maximum price for a car and a set of items Felfernig, Jeran, et al., 2014. As already mentioned for the requirements definition, in case of this approach these are constraints, a domain-related user interface is provided in which the user can explicitly define one's preferences. If an item fulfils all requirements, it is recommended to the user.

(Felfernig, Friedrich, et al., 2015) define two sets of variables and three types of constraints. The variables include the customer properties which represent the user requirements and the product properties which represent the product assortment and its characteristics such as the performance of a car. Furthermore, the constraints are split up into constraints, filter conditions and products. Constraints are restricting instantiations of customer properties (constraints which do not exclude each other). Next, filter conditions prevent the system from providing items which do not meet the user's requirements such as suggesting a professional camera to someone who is new to photography. Finally, products define the instantiated product properties based on the constraints and filter conditions.

Case-Based KBR

Case-Based KBR systems make use of similarity measurement functions to retrieve items which meet the user's requirements (Felfernig, Friedrich, et al., 2015). This approach has some similarities with CB RS since both approaches identify the content of items from the given domain. While CB systems often use unstructured content such as the text of an article, case-based RS rely on structured content Smyth, 2007. For example, a camera

could include properties such as the price, the resolution, the display size, its possible zoom settings and the weight.

After the user has defined her or his requirements in the system, the next step is to calculate the similarity between the items and the user model Smyth, 2007. For the similarity measurement, common functions such as the Pearson correlation coefficient are used as described in 2.3.1.

2.3.4 Hybrid Recommendations

In this section, we describe possible combinations of the major recommendation approaches categorised as hybrid RS (Felfernig, Jeran, et al., 2014) (Paradarami, Bastian, and Wightman, 2017). The goal of hybrid RS is to achieve better results as they make use of the advantages of the three major recommendation approaches.

Burke, 2002 defined the following hybridisations:

- **Weighted:** A weighted hybrid RS weights the included components of the system. Afterwards, the system combines the results into a single one. The simplest form of a weighted hybrid approach is a linear combination.

- **Switching:** This approach switches between recommendation techniques. Every switch is based on defined switching criteria (depending on the current situation). Thus, the system needs an additional parameterisation for the switching mechanics.

- **Mixed:** A mixed hybrid RS combines the output of the included components into a mixed result.

- **Feature Combination:** In case of this approach the features of the included components are combined to provide a broader bandwidth in terms of the content.

- **Cascade:** The cascade hybrid approach uses piping modules to pipe the result from the first recommendation technique as input to the next component of the system. Therefore, a priority scheme for the components is defined.

- **Feature Augmentation:** Like the cascade approach, this approach uses a staging mechanism. However, in case of this approach, the features of the first component are used by the second one.

- **Meta-Level:** In a meta-level hybrid RS the first component generates a model for the second component. In contrast to the feature augmentation this approach operates at a lower level. This means the model exists already in the feature augmentation approach.

2.3.5 Further Similarity Metrics

As already mentioned there are further similarity functions which can be used depending on the type of RS approach and content. In this section, we state the most common similarity functions for content values and vector comparisons.

For the adoption of numeric values, for example numeric content values in a CBF RS, different functions can be used to adopt the value. The choice of which function to use depends on the type of category and content such as maximising the processor speed of a PC. (McSherry, 2003)

Nearer Is Better (NIB):

$$sim(v_c, v_a) = 1 - \frac{|v_c - v_a|}{max(v_a) - min(v_a)} \tag{2.4}$$

More Is Better (MIB):

$$sim(v_c, v_a) = \frac{v_c - min(v_a)}{max(v_a) - min(v_a)} \tag{2.5}$$

Less Is Better (LIB):

$$sim(v_c, v_a) = \frac{max(v_a) - v_c}{max(v_a) - min(v_a)} \tag{2.6}$$

In the equations 2.4, 2.5 and 2.6 the variables c and a represent the current value and the category.

For the similarity measurement between vectors, for example item-to-item, there are several functions as well. Most of these equations can be applied to all three major RS approaches. (Polamuri, 2015) (Agarwal and Chauhan, 2017)

Pearson Correlation Coefficient (PCC):

$$sim(a, b) = \frac{\sum_{p \in P}(r_{a,p} - \overline{r_a})(r_{b,p} - \overline{r_b})}{\sqrt{\sum_{p \in P}(r_{a,p} - \overline{r_a})^2}\sqrt{\sum_{p \in P}(r_{b,p} - \overline{r_b})^2}} \tag{2.7}$$

For further details on PCC please see equation 2.1.

Cosine Similarity (CS):

$$sim(a, b) = \frac{R_a \cdot R_b}{||R_a|| * ||R_b||} \tag{2.8}$$

R_a and R_b represent a rating vector of an item rated by the users a and b.

Jaccard Similarity (JS):

$$sim(a, b) = \frac{|I_a \cap I_b|}{|I_a \cup I_b|} \tag{2.9}$$

I_a and I_b are set of items rated by the users a and b.

Mean Squared Difference (MSD):

$$MSD(a,b) = \sum_{p \in P} (r_{a,p} - r_{b,p})^2 \tag{2.10}$$

$$sim(a,b) = \frac{L - MSD(a,b)}{L} \tag{2.11}$$

P is the set of items which has been rated by the users a and b. $r_{a,p}$ and $r_{b,p}$ are a particular rating of an item p. L is a given threshold.

Euclidean Distance (ED):

$$ED(a,b) = \sqrt{\sum_{r \in R} (r_a - r_b)^2} \tag{2.12}$$

The ED calculates the distance between two vectors.

2.4 Data Mining

In this section we investigate the data mining processes which are necessary for the real estate domain based on the gathered knowledge from section 2.1. In general, the data mining process includes data preprocessing, model learning and result interpretation (Amatriain and Pujol, 2015). Since we build a content-based approach with similarity measurement methods for real estates, we mainly focus on the data preprocessing step. The other two steps, model learning and result interpretation, which are mainly used for classification approaches are therefore not relevant in our case. For further readings we refer to (Amatriain and Pujol, 2015) (Menzies, 2014b).

The data preprocessing is the first step of the data mining process. We identified three types of data: the items representing the real estates, the user data and the interactions with the systems including filter options and

the likes of a user. Liking is a gesture of the user which implies that the user is fond of the corresponding real estate.

Menzies, 2014b classified the data regarding an RS as follows:

- Tables including *items* as rows and *features* as columns
- Columns are *numeric* or *alpha-numeric*
- Some columns are *goals* (things we want to predict using the other columns)
- Features which are not included in an item called *missing values*

An RS executes data preprocessing steps on the raw data in order to be processable in the used approach and to improve the RS's parameters. (Amatriain and Pujol, 2015) describe four different challenges in the data preprocessing step which are explained below. Furthermore, we investigate feature scaling since every real estate has a broad range of features, data types and values (Bollegala, 2017).

Similarity Measures: Similarity Measures: First, the similarity metrics for the kNN identification (k represents the amount of NN) which we described in the sections 2.3.1 and 2.3.5.

Sampling: Sampling refers to the identification of a particular subset from a big amount of data. Sampling is often used to determine training and test sets from the provided data (Amatriain and Pujol, 2015). We skip the sampling methods since we neither need a classification of real estates nor training and test samples for our proposed approach. This is supported by the fact that we only use similarity measurements for the ranking determination.

Reducing Dimensionality: Reducing dimensionality is the process of defining a high-dimensional space with a low amount of features. Therefore, correlating features are dismissed to reduce the total amount of features. The most relevant dimension reduction algorithms are principal component analysis, single value decomposition and matrix factorisation (Amatriain and Pujol, 2015). In the context of this work a dimension reduction is not

necessary since only a small set of features is available in the context of real estates. A part of our future work is the filtering of features based on a correlation matrix.

Denoising: The last of the four challenges is called denoising. Denoising refers to removing unwanted effects through noise such as outliers or missing values included in the items (Amatriain and Pujol, 2015). O'Mahony, Hurley, and Silvestre, 2006 define two classes of noise: natural noise and malicious noise. Natural noise includes errors produced by the users during their interactions with the system such as defining their preferences. Hence, malicious noise occurs when an external source influences the RS. For example, if an author wants to promote her or his book. This step is the most relevant one for this work since items mainly do not provide the full amount of features in the context of real estates.

How noise or missing data are dealt with depends on the kind of problem. Swalin, 2018 states that there are two possibilities for handling missing data. One may either delete the data or impute data.

Deletion refers to deleting an explicit fragment of data.

- **Deleting rows (listwise):** Removes a row from an observation if one or more values are missing.
- **Pairwise deletion:** Analyses all cases in which variables are present and tries to maximise the available data based on an analytical basis. For example, determining the covariance between rows where a value is missing.
- **Deleting columns:** Drops the feature if too many items do not include the feature. Therefore, often a threshold is used.

Imputation uses the available data to determine a value for the missing features on a statistical basis. Furthermore, a classification of the problem is necessary to apply the right method for determining the missing values. This depends on two different problems. First, if the problem is accompanied by time-series issues such as if they exhibit a trend and seasonality. Second,

if it the data is continuous or categorical. The following methods can be applied to both of the problems.

- **Mean, Median and Mode:** Calculates the mean, median or mode for a feature. A disadvantage of this method is that it reduces the variance of the dataset.

- **Linear Regression:** This method uses a correlation matrix to predict several predictors for missing variables. Afterwards, the system's best predictor is identified as an independent variable for the linear equation while the missing data is defined as a dependent variable. For the equation only cases including all variables are used to generate the equation for the prediction later on. This process is then applied iteratively to the other missing values.

- **Multiple Imputation:** In the first stage, the entries of the incomplete dataset are imputed based on a Bayesian approach. Therefore, imputed values are used for the missing data. After that, the imputed values are analysed and integrated as a final result.

- **kNN:** In this case the determined kNN of an item are used to determine the missing value. For the NN determination, we refer to 2.3.1.

Feature Scaling: As already mentioned, the features of real estates vary highly in terms of magnitude, unit and range. Thus, we have to scale all features in order to be of the same weight. A common approach for feature scaling is normalisation (Kantardzic, 2003) (Asaithambi, 2017). The article from (Asaithambi, 2017) published on Medium, an online publishing platform, introduced four different approaches on how to scale features.

Standardisation:

$$x' = \frac{x - \overline{x}}{\sigma} \tag{2.13}$$

Scales the value by using the standard deviation σ. The values are between

-1 and 1.

Mean Normalisation:

$$x' = \frac{x - \overline{x}}{max(x) - min(x)} \tag{2.14}$$

The values are between -1 and 1.

Min-Max Scaling:

$$x' = \frac{x - \min x}{max(x) - min(x)} \tag{2.15}$$

The values are between 0 and 1.

Unit Vector:

$$x' = \frac{x}{||x||} \tag{2.16}$$

The scaling is based on the whole feature vector.

3 Requirements

In this chapter, we describe the requirements for the integration of the RS into the application of the customer, Immoky. For a smooth integration, we identified three different requirements. First, a RESTful Application Programming Interface (API) for the communication between the customer's application and the RS. Second, a software architecture which ensures an easy adoption and extension of the RS. Third, a programming language with which the customer is comfortable and which is suitable for the implementations of the RS.

3.1 RESTful API

As already mentioned, we use a RESTful API for the communication between the customer's application and the RS. In this section, we shortly describe the fundamental principles and functionalities of a RESTful API.

Representational state transfer (REST) is a client-server architecture based on the HTTP protocol (Chen et al., 2017). RESTful web services are a popular approach for computer programmes to communicate on an internet basis. They are used, for example, for requesting data from a server. The applications which communicate via the interface can access and manipulate data with the defined HTTP standard methods *GET, POST, PUT, DELETE, HEAD* and *OPTIONS* (Abts, 2015).

- **GET:** A GET request is used to retrieve data with reading access. Thus, data manipulation is not possible (the source on the server stays the same).

- **POST:** The POST method is used to create a new resource on the server with a given URI. Additionally, an arbitrary process can be triggered.
- **PUT:** A PUT request is used to update an already existing resource.
- **DELETE:** With a DELETE request, the deletion of an existing resource from the server is triggered.
- **HEAD:** In contrary to the GET request, a HEAD request queries metadata from the server such as the status of a resource.
- **OPTIONS:** An OPTIONS call is used to get information about a resource such as the content type.

Listing 3.2 shows the answer as JSON object of an GET request using the route *http : //localhost* : 3333/*recommendersystem* of a simple API we implemented.

```
1 {
2     data: "Hello Recommender System!"
3 }
```

Listing 3.1: Answer of a GET request

The request of the call looks as shown in listing 3.2. We used Postman, an API development tool, as software Postman, 2018 to trigger the call.

```
1 GET /recommendersystem HTTP/1.1
2 Host: localhost:3333
```

Listing 3.2: HTTP GET request

3.2 Software-Architecture

After we defined the type of interface for the communication, the next step is to determine a software architecture for the RS which is modular and easy to extend.

Therefore, we use a Pipe-And-Filter approach. This type of architecture includes filters (components) and pipes (channels) (Philipps and Rumpe, 2014). The former part is the processing module, also called filter, while the latter is the connection module of the components. A component or filter is an independent entity which can be executed either sequentially or in parallel. (Kumar, 2014) defined three variations of the architecture: pipelines, bounded pipes and typed pipes.

- **Pipelines:** A pipeline allows to execute filters only sequentially (see figure 3.1).
- **Bounded pipes:** In this variation, pipes can have a restricted amount of data.
- **Typed pipes:** A typed pipe restricts the type of data within the system.

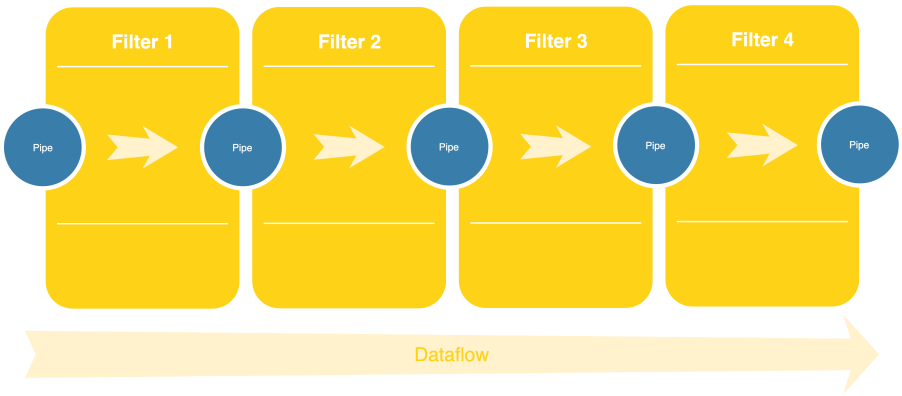

Figure 3.1: Pipeline approach of a Pipe-And-Filter software architecture
Source: Created by author.

3.3 Programming Language

In the last section of this chapter, we define the programming language which we use for the implementation of the RS. Therefore, we focus on an established programming language for machine learning algorithms. As a result, we concluded to use Python (Python, 2018), since Python is well known for its libraries such as Scikit (Scikit, 2018) or Numpy (NumPy, 2018) and because of its simplicity (Protasiewicz, 2018).

4 Approach

On the basis of our research questions, we chose to use a two-way approach for the RS to achieve the best possible accuracy with our proposed approach. First, we recommend real estates to the user with a content-based approach described in 2.3.2. Second, we refine values of the user's features through her or his behaviour and interactions. The customer must perform these two steps separately and sequentially since we need the data from the user behaviour represented through likes and dislikes for the feature update calculations.

In the following chapters, we describe the architecture of Immoky including the communication between the entities, the recommendation and the refinement of the user features.

4.1 Architecture

For nearly all processes of the mobile application, the customer uses services from Google Cloud Platform (GCP). These are mainly processes which query, mutate and access data from the database and file storage. Only for the web interface and iOS/Android App they use standalone Linux servers. The customer deploys all data processing processes as GCP service. Thus, we also got a GCP service on which we deploy the RS. 4.1 shows all processes and the corresponding communication. An arrow indicates the direction of an interaction between two processes.

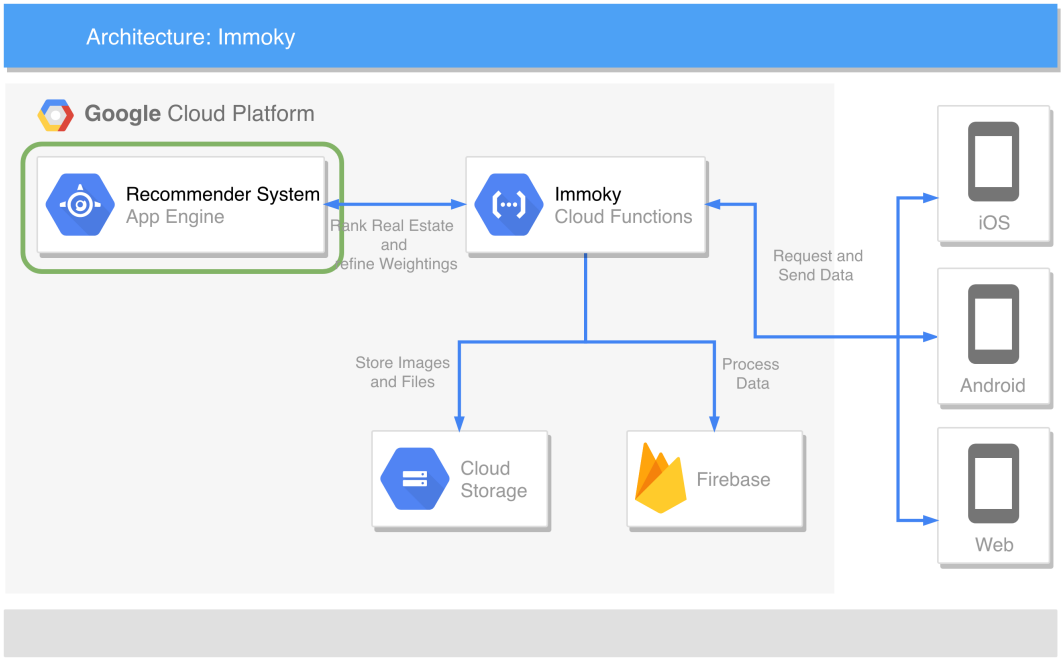

Figure 4.1: Architecture of the customer's application
Source: Created by author.

In this work, we focus only on the process marked in green, which is the RS, as seen in figure 4.1. The customer's application architecture strictly splits the presentation of the data from its processing. Server-side processes are grouped as the backend and client-side applications as the frontend.

Frontend: Frontend entities are client-side processes which are responsible for the presentation of the data retrieved from the server. In case of the customer's application, these are the mobile application (iOS and Android) and the web interface. These applications retrieve the data from the central unit, the Google Cloud Functions (GCF) which process the data on the server 4.1.

- **iOS:** This mobile application is explicitly implemented for Apple devices. These devices use iOS as an operating system. Thus, a separate application is necessary.

- **Android:** Due to compatibility reasons, the customer needs to implement a separate application for Android devices as well.

- **Web:** For the administration of real estate, user profiles and accounting, the customer uses a web interface. Currently, only the customer's team has access to the interface, but in further stages, estate agents can advertise, adapt and dismiss real estates on their own.

Backend: Backend entities, are server-side processes which send the data to the requesting client. Figure 4.1 states all included backend services.

- **Customer's GCF:** The GCF services on GCP are a serverless infrastructure approach. Instead of launching the programming code of, e.g. an API, a programmer needs to deploy only fragments of her or his system. These fragments are called cloud functions. To retrieve or process data, a client invokes a function directly on the server with a HTTP request using the POST, PUT, GET, DELETE and OPTIONS methods described in 3.1. Based on this service structure, Google applies a different cost structure for GCF. The customer has to pay only for the number of calls instead of the whole server (Malawski et al., 2017).
  The customer uses this service as a central data processing and communication unit for their application. They deploy functions for the following functionalities:
  - Endpoint for clients to request and manipulate data from and on the server
  - To organise real estates and user profiles in the database
  - To store files on the cloud storage
  - To retrieve recommendations from the RS
- **Firebase:** The customer uses the Firebase service on GCP as a database. The Google Firebase real-time database is a cloud-hosted NoSQL database using a JSON format for its data structure. The service provides many built-in functions such as instant messaging or user authentication which make the programming of an API easier (Li et al., 2018).

- **Cloud Storage:** The next entity is the cloud storage. The customer uses this service as a file server to save files and images of users and real estates. Again, this storage is a built-in service of GCP.

- **RS App Engine:** The last process is the RS of this work which we deploy as App Engine service on GCP. A Google App Engine allows a programmer to implement web applications or a backend with good possibilities to scale high. Based on the server utilisation, Google increases the server properties such as processing power or the used servers. A programmer has to administrate the settings on her or his own. These settings are used by GCP for the server setup and scaling. For users who are not comfortable with the setup and administration of servers, this service is, in general, a good alternative. (Mohsin, 2015)

As mentioned before, we only focus on the RS and its communication with the GCF. Figure 4.2 states the included processes of the RS for the recommendation and feedback function. An arrow indicates the direction of a communication.

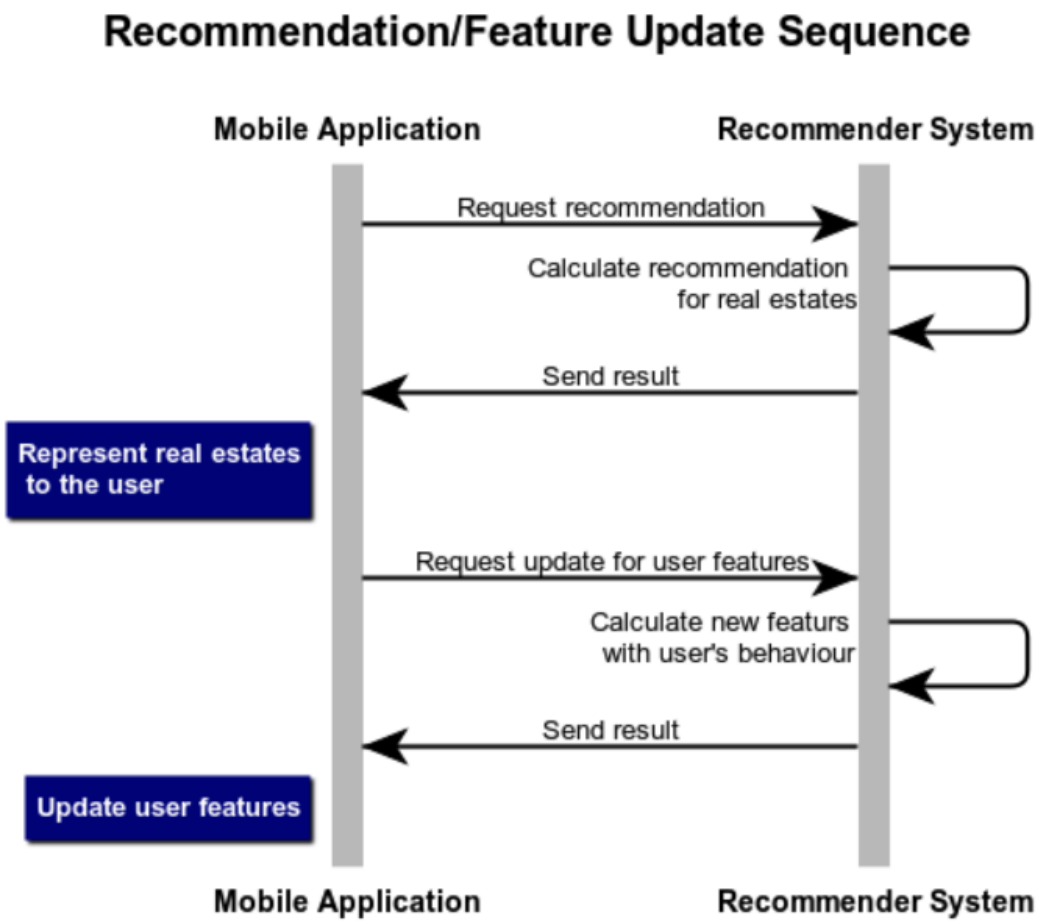

Figure 4.2: Sequence diagram for recommendation and feature update
Source: Created by author.

As shown in figure 4.2 there are two separate requests: a request to get recommendations and a request which triggers the feedback calculations for the features. For the latter request interactions (likes and dislikes of real estates) from the users are necessary. Thus, the general process is to request first recommendations for the user and then to update the features from the user based on her or his feedback. These steps are executed in sequential order and repetitive.

4.2 Application Programming Interface of the RS

In this section we explain the interface between the API of the customer (GCF) and the RS. As already described in chapter 3 we use a RESTful architecture for the communication. In case of this work, we have two routes, one for the recommendation and one for the feedback.

The next step is to define a data format for the recommendation and feedback request. Since the customer wants to extend and configure the RS in further stages, we need to implement a generic data model. For example, if the customer wants to add or remove real estate attributes. Therefore, we use a JSON object for each feature which includes the name of the feature, its data type, the value and a weighting if defined (see listing 4.1). In a CBF based RS, real estate attributes reflect the user's preferences. Thus, we use the same format for the features for both entities.

```
1 {
2   feature: String,
3   type: FeatureType,
4   value: Value dependent on FeatureType,
5   weight: [0-5]
6 }
```

Listing 4.1: Feature object

As shown in the listing 4.1, there are four different keys in the feature object: *feature*, *type*, *value* and *weight*.

- **Feature:** This is the unique name of the feature.
- **Type:** The type of feature defines the data type. Based on this information we convert the feature value to a value which is interpretable by our RS. Since real estate features have different types and formats, a property for this information is necessary. Based on our research and the test samples from the customer, we identified the following data types: number, date, the days between two dates, if in range

of two values, the distance between to addresses (coordinates) and enumerative values.

- **Value:** This is the actual value of the feature from the corresponding real estate or user profile.

- **Weight:** The weight key defines the importance of a feature based on the user's preferences.

This feature format implies a generic interface since we can add arbitrary features by using the unique name as *feature* and its type as *type*.

4.2.1 RESTful Setup

In this section we describe the RESTful setup of the RS. Python provides a library called *Flask* which provides simple functions for setting up a RESTful server architecture (Ronacher, 2018).

Setup: For the setup we instantiate a Flask class for the server and Flask RESTful class for the API. The RESTful instance provides a $add_resource$ function which we use to register the recommendation and feedback route. .

```
1  app = Flask(__name__)
2  api = Api(app)
3
4  #Define routes
5  api.add_resource(Recommendation, '/recommendation')
6  api.add_resource(Feedback, '/feedback')
7
8  if __name__ == '__main__':
9      np.set_printoptions(suppress=True)
10
11     app.run(port='3000')
```

Listing 4.2: Server and API setup

In line one and two we instance the server *app* and the RESTful API *api*. Afterwards, as mentioned before, we register the routes for the recommendation */recommendation* and feedback */feedback* request in the *api*. The last line invokes the server to listen on port 3000. Thus, we access the server of the system, if locally hosted, at the address *localhost* : 3000.

4.2.2 Routes

As already mentioned in section 4.2, we implement two routes for the RS — one route for the recommendation and one for the feedback request. With the feedback request, we trigger an update of the user's preferences to improve the parameters of the RS for higher accuracy. Therefore, we update the features based on the user's interactions by considering the feature values of the liked or disliked real estate.

For both endpoints we use a HTTP POST request and receive the data for the calculations through its body.

Recommendation: The customer uses this route to only present real estates to a user which meet the user's preferences in the best possible way. In our approach, we have to consider two different scenarios. First, we use the user's answers from the tutorial as initial features for the first recommendations. Second, we update the features from the user's feedback regarding the previous recommendations. These features represent the preferences of the user in higher dimensional space. Thus, we either build a feature set from the user's answers from the tutorial or use an already existing and updated one from the user's interactions.

```
1 class Recommendation(Resource):
2     def post(self):
3         body = request.json
4 
5         #Set original data already here (normally DataSource)
6         dataPipe = DataPipe()
7         dataPipe.originalData = body
```

```
8
9           #Calculate ranking now
10          recommenderSystem = RecommenderSystem([FeatureFilter(),
11               ImputationFilter(),
12               NormalizationFilter(),
13               WeightingFilter(),
14              RankingFilter()])
15          recommenderSystem.run(dataPipe)
16
17          #Remove features to provide only the ids
18          for realEstate in dataPipe.realEstates:
19              del realEstate['features']
20
21          return dataPipe.realEstates
```

Listing 4.3: Recommendation route

As seen in listing 4.3 there are three coding parts headed with a comment. The first one sets up the connection object called *DataPipe*, the second one runs the ranking calculations with all necessary data processing filters and the last one extracts the identifiers from the ordered real estates. In the request body *request.json*, we receive the necessary data for the recommendation process.

Feedback: The feedback request is used to update the user's features based on her or his interactions with the mobile application. Currently the customer only transmits real estates the user liked or disliked. A like indicates that the user likes a real estate, a dislike means the opposite. We use this information to update the user's feature set which presents the user's preferences.

```
1 class Feedback(Resource):
2     def post(self):
3         body = request.json
4
5         #Calculate ranking now
6         feedbackSystem = FeedbackSystem()
7
8         #Make feedback calculation
```

```
9           return feedbackSystem.updateUserProfile(body['features'
                ], body['realEstates'])
```

Listing 4.4: Feedback route

In the code snippet seen in listing 4.4 there are two lines which are responsible for the update procedure. First, line six, where we instance the feedback system and second, line nine, where we update the user's feature set by calling the function *updateUserProfile*. The latter function receives the user's features and real estates marked with a liked flag as parameters. In the request body *request.json*, we receive the necessary data for the feedback process again.

4.3 Recommendation

In this section, we describe the implementation of the RS. As a software design, we use an adapted Pipe-And-Filter architecture as described in chapter 3.2. The adaptation is related to the filter mechanism. In addition to the filtering, we also transform and process the data in the filter components. We use a push-pipeline approach for our implementations in which data is only forwarded sequentially from one element to the following one. As a connection module for the components we define an object for the requesting user, the real estates and matrices used for the ranking calculations. To stick to the restrictions of a push-pipeline, we register the components in an array and execute them in the right order with the connection object as a parameter. Therefore, we build a Pipe-And-Filter base class with a generic process function. A deriving filter component of the base class has to implement the function with its data processing functionality.

```
1 class PipeAndFilterBase:
2     __metaclass__ = ABCMeta
3 
4     @abstractmethod
5     def process(self, dataPipe):
6         pass
```

Listing 4.5: Base used by a Pipe-And-Filter component

Listing 4.5 shows the implementations of the Pipe-And-Filter class. We use the annotation *@abstractmethod* to ensure that the deriving class implements the process function.

In addition to the base class, we use a parent object called *RecommenderSystem* that contains the component array described before and a *run* unction which executes the process function of the included components in the right order. This approach ensures the restrictions of the push-pipeline are adhered to.

```
1 class RecommenderSystem:
2     def __init__(self, entities):
3         self.entities = entities
4 
5     def run(self, dataPipe):
6         for entity in self.entities:
7             entity.process(dataPipe)
```

Listing 4.6: Parent object for executing the pipeline

Listing 4.6 shows the parent object which represents the RS. It has a function *run* which executes the process functions of the pipeline components sequentially by using a for-loop.

As already mentioned, we use a Pipe-And-Filter architecture because of our sequential data processing steps. First, a datasource determines the necessary data, then components filter, transform and process the data and last, a datasink further processes the result. In case of our approach, we create one datasource for the data retrieval (user profile and relevant real estates), five filters for the data preprocessing and recommendation calculations and one datasink for returning the result to the client. A client is, for example, the mobile application of the customer.

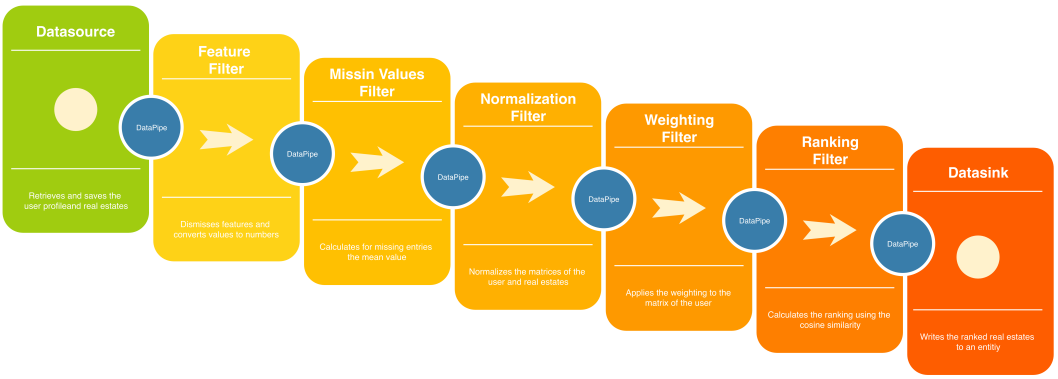

Figure 4.3: Pipe-And-Filter elements of the RS
Source: Created by author.

The pipeline seen in figure 4.3 states all included components of our RS: the datasource, all data filtering, transforming and processing components and the datasink.

4.3.1 Connection Object

As already described, we use a connection object called *DataPipe* for the communication between two components. The name *DataPipe* refers to the piping mechanism of the components. However, in this work, we also process the data saved in the connection object instead of only filtering. For our filters, we use two different types of properties: one for saving plain features and one for the recommendation calculations. The properties are described in detail below:

- **Original Data:** Represents the original data how we retrieve it from an external source such as the mobile application of the customer. This object will be further processed to extract the features from the corresponding user and real estates.

- **User-Features:** This property includes the user's features of the original data sorted by their key. The key is a unique identifier of the

feature.

- **Real Estates:** We save the real estate data objects separately from the original data object since we adopt them in the datasink to send their IDs ordered by similarity.

- **User Datamatrix:** The datamatrix is the object we use for the similarity calculations. This object is a numeric and one-dimensional matrix which contains the features of a user ordered by their keys.

- **Real Estates Datamatrix:** In contrast to the user datamatrix, the real estate datamatrix contains all real estates and their features. The count column equals the number of real estate, and the count row represents the number of the user's features. This means if a real estate contains a property which is not included in the user's feature set, we remove it. Whereas, if a real estate does not contain a feature from the user's feature set, we mark it with NaN which indicates that the value is missing. Furthermore, we sort the features in the same order as the user's features in the user datamatrix, i.e. ordered by the key of the feature.

4.3.2 Datasource

In this section we describe the datasource of the RS. As the name suggests, a datasource is used to determine data for the following process. This work focuses on two different implementations: one in which we determine the data from an external file and one in which we retrieve data from the customer's mobile application.

As already mentioned in the previous chapters, we need to achieve a generic interface for a smooth adoption which ensures an easy-to-extend system. Therefore, we defined an object of the JSON format including the user's profile and all relevant real estates. For both types we use an array of type feature to describe the corresponding object. The customer can easily extend

the array after the completion of the project since one only needs to adopt the feature configuration file.

```
{
  userId: ID,
  features: [FeatureType],
  realEstates: [
    {
      id: ID,
      features: [FeatureType]
    }
  ]
}
```

Listing 4.7: Body-format used for the recommendation request

As stated in listing 4.7, there are three different first keys in the first layer of the JSON object – the *userId* representing a unique identifier of the user, the *features* array which belong to the user and the *realEstates*-array including the relevant real estates. The type *FeatureType* represents the format of a feature described in listing 4.1.

File Datasource

Our first implementation step refers to a datasource based on a file. Since we could not connect to the API of the customer in the early stages, we read the data from an external file. On the basis of some test records provided by the customer, we created an initial file with real estates. Therefore, we used the same format from chapter 4.3.2 for the test set (user features and real estate). This approach ensured static progress for our RS.

RESTful Datasource

In this section, we refer to the RESTful interface described in 4.2. The recommendation request *localhostl* : 3000/*recommendation* route represents

the RESTful Datasink. This includes the parsing of the request body, the instancing of the *RecommenderSystem* class and the execution of the *run* function.

Feature Extraction Class: In addition to the interface, we implement a parsing class which extracts the features for a user or real estate from a database object of the customer. Therefore, we define a configuration object that specifies the name of a feature and its type and position in the database object. We use separate properties for the position of a user and a real estate since they have a different structure in the database. Listing 4.8 shows an example of a possible configuration.

```
const FEATURE_TYPE_NUMBER = "number";
const FEATURE_TYPE_DATE = "date";
const FEATURE_TYPE_IN_RANGE = "inRange";
const FEATURE_TYPE_ADDRESS = "address";
const FEATURE_TYPE_ENUM = "enum";

private relevantFeatures = [
  {
    name: identifier of feature,
    type: feature type defined by the constantsFEATURE_TYPE_*
    // Position of feature in database object
    userFeature: [keys defining position],
    realEstateFeature: [keys defining position]
  }
];
```

Listing 4.8: Configuration file for the feature extraction

As seen in listing 4.8, we define five different constants, *FEATURE_TYPE_NUMBER*, *FEATURE_TYPE_DATE*, *FEATURE_TYPE_IN_RANGE*, *FEATURE_TYPE_ADDRESS* and *FEATURE_TYPE_ENUM* which represent the possible types of a feature. The array called *relevantFeatures* defines the features and their position in the corresponding database objects. Therefore, we use four different keys which are explained below:

- **Name:** The name is a unique identifier that maps the feature of a user with the one of a real estate. This is necessary since a feature can have

different names for a user and real estate. For example the rental fee is named *max_rental_fee* for the user and *price* for the real estate.

- **Type:** This property refers to the feature types of the RS. These are indicated by the prefixes *FEATURE_TYPE_*$*$, where $*$ stands for a place-holder for the type.

- **User feature:** Defines the position of the feature in the database object of a user. Therefore, we use an array in which the first element defines the position of the first layer in the corresponding object,the second one in the second layer and so forth.

- **Real Estate Feature:** As already mentioned, the name of a feature in a user object may differ from the real estate one. Therefore, we define a separate property which contains the name of the feature in the real estate object.

Listing 4.9 shows a fictitious example of how we extract a feature and its values from a user and real estate object by using the described keys.

```
1  // Fictitious object defining a feature definition
2  const relevantFeature = {
3    name: "feature_1",
4    type: FEATURE_TYPE_NUMBER
5    // Position of feature in database object
6    userFeature: ["key_layer_1_b", "key_layer_2_a"],
7    realEstateFeature: ["key_layer_1_c"]
8  }
9
10 // Fictitious database object representing a user
11 const user = {
12   key_layer_1_a: value_layer_1_a,
13   key_layer_1_b: {
14     key_layer_2_a: numeric value of user which is used for
           feature_1,
15     key_layer_2_b: value_layer_2_b
16   },
17   key_layer_1_b: value_layer_1_b,
18 }
19
```

```
20 // Fictitious database object representing a real estate
21 const realEstate = {
22   key_layer_1_a: value_layer_1_a,
23   key_layer_1_b: value_layer_1_b,
24   key_layer_1_c: numeric value of real estate which is used for
         feature_1,
25   key_layer_1_d: {
26       ...
27   }
28 }
```

Listing 4.9: Feature extraction example from a user and real estate object

As seen in listing 4.9, we have an object called *relevantFeature* which defines the features to be extracted from the objects *user* and *realEstate*. Both objects have several and arbitrary values as content. Thus, we use the *userFeature* and *realEstateFeature* array to determine the position of the values in the user and real estate object. This value will then further be processed under the name $feature\_1$ and type *number* in the RS.

Due to no access to the customer's database, she or he integrates the feature extraction class directly into her or his system. If issues regarding performance arise, we integrate the interface directly into the RS and connect it to the Firebase database for direct access. Additionally, we modify the RESTful API and change the recommendation and feedback routes with a trigger request and fetch the necessary data directly from the database.

4.3.3 Filter

The next entities of the Pipe-And-Filter architecture are the filter components. As already described, a standard filter reduces data sequentially rather than processing or formatting it. However, we make use of the piping mechanism to process the data sequentially. In the following sections, we describe the included filters we use for the filtering, processing and transformation of the data.

Feature Filter

The first filter of the RS is the feature filter which relates to determining and converting features. It includes two steps: As a first step, based on the user's features, we check whether a feature exists or is missing in a real estate. Second, we convert the features to numbers and save them in the corresponding datamatrix. We use a function called *buildUpDatamatrices* to do so:

```
1     def buildUpDatamatrices(self, dataPipe):
2         dataPipe.userDataMatrix = np.zeros((len(dataPipe.
              userFeatures), 1))
3         dataPipe.realEstateDataMatrix = np.zeros((len(dataPipe.
              userFeatures), len(dataPipe.realEstates)))
4
5         #Loop through relevant features and build up matrix
6         for indexF, userFeature in enumerate(dataPipe.
              userFeatures):
7             #Extract value for user feature
8             dataPipe.userDataMatrix[indexF][0] = self.
                  getUserValue(userFeature)
9
10            for indexRe, realEstate in enumerate(dataPipe.
                  realEstates):
11                features = realEstate['features']
12                try:
13                    match = next(f for f in features if f['
                          feature'] == userFeature['feature'])
14                except Exception as inst:
15                    match = None
16
17                #Extract value for real estate feature
18                if (match == None):
19                    dataPipe.realEstateDataMatrix[indexF][
                          indexRe]= None
20                else:
21                    dataPipe.realEstateDataMatrix[indexF][
                          indexRe] = self.getRealEstateValue(
                          userFeature, match)
```

Listing 4.10: Function for the feature conversion and datamatrices setup

As seen in listing 4.10, we first initialise the data matrices with zero. For

both matrices we use the same amount of columns which is defined by the number of features of the user. Next, we iterate through the features and real estates to determine the values for both of the entities. Therefore, we invoke the function *getUserValue* to get the numeric value of a feature from the user and *getRealEstateValue* for each of the real estates. Furthermore, we check if a feature is missing in the real estate. If so, we use *NaN* as value for the feature.

For the conversion functions *getUserValue* and *getRealEstateValue*, we use the type from the feature. Depending on the type, we convert the features as follows:

- **Number:** Is a numeric value which we do not process further.
- **Date:** For both entities, user and real estate, we determine the difference between the provided date and today's date. Afterwards, we use the range in days for the corresponding entity.
- **Range-Days:** Date of a property like the year of construction. For the user, we use 0 as a uniform value, and for the real estate, we determine the days between the user's and real estate's date.
- **In-Range:** Defines if the real estate's value has to be between a minimum and maximum value from the user. For the conversion, we use 1 as a uniform value for the user, whereas we use 1 for the real estate value if in range and 0 if not.
- **Address:** An address is represented as coordinates (latitude and longitude). We calculate the straight-line distance in kilometres between the address of the real estate and the desired location of the user. Next, we use the distance for the real estate and 0 as a uniform value for the user. The range calculation is based on the Haversine formula (**Alam2016**).
- **Enumerate:** An enumerate represents characteristics of a real estate property such as *builtInKitchen* in the feature *equipment*. In addition

to the characteristic, we add a counter which indicates the importance of the property for the user. We increment the counter if the user liked a real estate with this entry and decrement it otherwise. As with the types, date, in-range and address, we use 0 as a uniform value for the user. The calculation of the real estate value looks as follows: First, we determine the intersection between the user's and the real estate's array. Then we calculate the sum of the counters of the intersected entries. Listing 4.11 states an example calculation of this type.

```
{
  feature: "equipment",
  type: "enum",
  value: [
    {
      name: "builtInKitchen",
      count: 5
    },
    {
      name: "basement",
      count: 1
    }
  ]
}
```

Listing 4.11: User feature of enumerate type

Missing Values Filter

The first filter which adapts the numeric values in the data matrices is the Missing Values Filter. A missing value refers to attributes which were not provided through the corresponding real estate. Since Immoky does not administrate all objects on its own, there may be several real estates with sparse data. This is due to various reasons. The information is either not available, does not exist for the real estate at all, or the real estate is badly administrated. For such missing attributes we used NaN as already

described in section 4.3.3.

Missing values may falsify the result of an RS or make it even impossible. Thus, we determine the mean value of the feature including the user and all other real estates. The following matrices show an example of the filtering result.

$$M = \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{array} \begin{array}{c} \begin{array}{cccc} u & v_1 & v_2 & v_3 \end{array} \\ \left(\begin{array}{c|ccc} 0 & 11.1 & 100.2 & 240.8 \\ 0 & 3 & 59 & NaN \\ 1 & NaN & 1 & NaN \\ 1200 & 1200 & NaN & NaN \\ 52 & 49 & 42 & 30 \\ 650 & 630 & 540 & 440 \end{array}\right) \end{array} \rightarrow \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{array} \begin{array}{c} \begin{array}{cccc} u & v_1 & v_2 & v_3 \end{array} \\ \left(\begin{array}{c|ccc} 0 & 11.1 & 100.2 & 240.7 \\ 0 & 3 & 59 & 20.7 \\ 1 & 1 & 1 & 1 \\ 1200 & 1200 & 1200 & 1200 \\ 52 & 49 & 42 & 30 \\ 650 & 630 & 540 & 440 \end{array}\right) \end{array}$$

The matrix is a joint matrix consisting of the user's and real estate's data-matrix. Thus, u represents the user's features, $v_{1...4}$ are four real estates and $i_{1...6}$ represent the features. As already mentioned NaN indicates a missing value. We calculate the mean value row-wise and save it in the right matrix. The mean value is calculated as follows:

$$x = \frac{u_i + v_{i,1} + ... + v_{i,n}}{1 + N}$$

i...row index of feature, n...real estates, N...amount of real estates

Normalization Filter

First, we take care of the missing values, next we normalise the data matrices to a unit norm between 0 and 1. This is necessary since the features of a real estate vary highly in magnitude, unit and range. A higher-valued feature would be more important than one with lower values. To fix this problem, we normalise the data matrices as described in 2.4. The following matrices show an example of the normalisation process.

$$M = \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{array} \begin{array}{c} \begin{array}{cccc} u & v_1 & v_2 & v_3 \end{array} \\ \left(\begin{array}{c|ccc} 0 & 11.1 & 100.2 & 240.8 \\ 0 & 3 & 59 & 20.7 \\ 1 & 1 & 1 & 1 \\ 1200 & 1200 & 1200 & 1200 \\ 52 & 49 & 42 & 30 \\ 650 & 630 & 540 & 440 \end{array}\right) \end{array} \rightarrow \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{array} \begin{array}{c} \begin{array}{cccc} u & v_1 & v_2 & v_3 \end{array} \\ \left(\begin{array}{c|ccc} 0 & 0.04 & 0.38 & 0.92 \\ 0 & 0.05 & 0.59 & 0.33 \\ 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 \\ 0.59 & 0.56 & 0.48 & 0.34 \\ 0.57 & 0.552 & 0.47 & 0.39 \end{array}\right) \end{array}$$

As in the previous section 4.3.3 u , $v_{1...4}$ and $i_{1...6}$ represent the user, the real estates and the features. The left matrix, which includes the values before the normalisation, has a broad range from 0 to 1200. We normalised the joint matrix which resulted in the left matrix. All values lay between 0 and 1 and are therefore of the same importance.

Weighting Filter

The last step before we calculate the similarity between the real estates and the user's preferences (features) is the weighting of features. In general, some attributes are more important for users than others. A possible approach to consider these tendencies is to make features more important by weighting them. In this work, we multiply a weighting factor with the corresponding feature. The customer provides the factor as additional property in the feature of the user. We define weights initially for a user which she or he can directly overwrite in the mobile application. For example, the rental fee is generally an important feature for a user. The weighting factor can lay between 0 to 5, where 0 stands for an irrelevant feature and 5 for an essential one. Therefore, we use the following equation to determine the new values.

$$x_i = x_i * weight_{u_i} * 0.2$$

x...user or real estate feature, i...row index of feature, weight...weighting of feature (0...5)

We apply the equation to all features. As an example, we use the values 0.5 and 0.75 to describe the calculation. Before applying the weighting, the difference between the values is 0.25. Afterwards, the difference decreases to 0.05 and thus has less impact on the calculations as before. If no weighting is provided for a feature, we use 3 by default. The following matrices show an example before and after we applied the weighting to the features.

$$M = \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{array} \begin{array}{c} \begin{array}{cc|ccc} u & & v_1 & v_2 & v_3 \end{array} \\ \left(\begin{array}{cc|ccc} 0 & & 0.04 & 0.38 & 0.92 \\ 0 & & 0.05 & 0.59 & 0.33 \\ 0.5 & & 0.5 & 0.5 & 0.5 \\ 0.5 & & 0.5 & 0.5 & 0.5 \\ 0.59 & & 0.56 & 0.48 & 0.34 \\ 0.57 & & 0.552 & 0.47 & 0.39 \end{array}\right) \end{array} \rightarrow \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{array} \begin{array}{c} \begin{array}{cc|ccc} u & & v_1 & v_2 & v_3 \end{array} \\ \left(\begin{array}{cc|ccc} 0 & & 0.03 & 0.23 & 0.55 \\ 0 & & 0.03 & 0.57 & 0.2 \\ 0.3 & & 0.3 & 0.3 & 0.3 \\ 0.3 & & 0.3 & 0.3 & 0.3 \\ 0.12 & & 0.11 & 0.1 & 0.07 \\ 0.57 & & 0.55 & 0.47 & 0.39 \end{array}\right) \end{array}$$

In the example above, we use the weighting 5 for the feature $i = 5$ and 1 for $i = 6$. Only the feature on which we applied the weighting 5 stayed the same since 5 represents 100 percent of the value. All other values decreased, whereas the values of feature $i = 6$ with the lowest importance sank the most.

Ranking Filter

The filters described in the previous sections are used to apply different processing methods on the original data. These steps are necessary to make a user profile or real estate usable for our RS. In this section, we describe the calculations which are necessary for the similarity measurement.

For our RS, we use a CBF-based approach to recommend real estates to a user. The term content relates to the information which describes an item as already described in section 2.3.2. In case of real estates, these are common characteristics such as the monthly rent, the footage or the number of rooms. With the same characteristics, we reflect the user's preferences. Furthermore, we use the user's interactions with the system to refine the features.As with

the types of RS, there are also several functions for comparing items. Besides the comparison functions, we can further parameterise the system as well. For the RS of this work we use the similarity function Cosine Similarity described in section 2.3.5.

An RS sorts the items, for example, by their similarity with user's preferences. For the calculations, we use the datamatrix and apply the similarity function to the items. Therefore, for each column in the real estate datamatrix we calculate the Euclidean distance with the user datamatrix. A column in the real estate datamtrix represents the numeric form of a particular real estate. Afterwards, we save the distance in the corresponding real estate by using the same index of the loop mentioned before. When all similarities are determined, we sort the real estates accordingly in ascending order. Thus, the first entry in the real estate array represents the item which meets the user's requirements best. The higher the distance between the user and the real estate vector the more they differ. We overwrite the real estate's array of the *DataPipe* object to pipe the result to the datasinkfor further processing. We applied the calculations on the example matrices used in the previous sections and get the following result:

$$M = \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{array} \begin{array}{c} \begin{array}{cccc} u & v_1 & v_2 & v_3 \end{array} \\ \left(\begin{array}{c|ccc} 0 & 0.026 & 0.23 & 0.553 \\ 0 & 0.029 & 0.566 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.3 \\ 0.118 & 0.111 & 0.095 & 0.068 \\ 0.569 & 0.552 & 0.473 & 0.385 \end{array}\right) \end{array}$$

$$\rightarrow sim(\boldsymbol{u}, \boldsymbol{v}_1) = 0.9984, sim(\boldsymbol{u}, \boldsymbol{v}_2) = 0.7217, sim(\boldsymbol{u}, \boldsymbol{v}_3) = 0.687$$

4.3.4 Datasink

The datasink is the last component of our RS. Here, the result of the filter components gets further processed. We either save the result in a file or send it back to the customer's mobile application. In both cases, we first extract the unique identifiers, the IDs, from the sorted real estate array of the connection object, the *DataPipe*. Therefore, we delete all properties within a real estate object except for the id.

File Datasink

We use the file-based datasink for testing the RS without connecting to an external system. As with the file-based datasource from section 4.3.2, this component uses a file in a specific path in which we save the IDs of the sorted real estates.

RESTful Datasink

Since the RESTful datasource 4.3.2 refers to the recommendation request *localhost* : 3000/*recommendation*, the RESTful datasink is merely the response to this call. Thus, it only represents the returning of the result to the customer's mobile application.

4.4 Feedback

The second part of our work is to improve the RS by using the interactions of the user with the customer's mobile application. Therefore, we consider two different types of interactions: **filter options** and **swipeable cards** which represent real estates.

4.4.1 Filter Options

For the first recommendation for a user, we use the answers from the customer's tutorial which are considered as filter options. These filter options are saved in the user profile and used in two different ways. First, the customer pre-filters the real estates based on some knock-out criteria such as object type or pricing model for the RS later on. Therefore, some filter options are pre-defined as knock-out criteria such as object type or pricing model. Second, we use the answers as initial features for the RS by applying the feature extraction class described in paragraph 4.3.2 on the user's profile. Since a user can customise her or his filter options, we modify the RS-related features as well. We assume that if a user changes her or his filtering options and therefore her or his preferences, the user is not satisfied with the recommended real estates. Thus, we reset the features by using the new filter options of the user.

Furthermore, a user can choose between four different object types. These are apartments, houses, commercial space and investment offerings. In this work, we consider only those objects which satisfy the need for accommodation, such as apartments and houses. For these two object types, we create a separate feature profile for the user. Thus, we can apply the feature profile regarding the searched object type.

As already mentioned before, we apply configurations used in a feature extraction class to build up our feature profile for a user. These configurations need to be related to the object type since each type has different attributes and positions for a feature. Therefore, we extend the configurations object of the feature extraction class 4.3.2 by adding a separate array for each of the remaining object types. These arrays define the position of a feature, such as the *userFeature*, and *realEstateFeature* objects do. This results in the object seen in listing 4.12.

```
1 private relevantFeatures = [
2   {
3     name: identifier of feature,
4     type: feature type
5     // Position of feature in database object
```

```
6       userFeature: [keys defining position],
7       appartmentFeature: [keys defining position],
8       houseFeature: [keys defining position]
9     }
10  ];
```

Listing 4.12: Defining position object for a feature

Listing 4.12 shows which keys are used to extract a feature from a database object. We use one for the user called *userFeature* and two for the real estate object types called *appartmentFeature* and *houseFeature*. These keys represent the position of the feature in the database object as already described in section 4.3.2. If a particular feature is not available in an object type, we set the corresponding key to null. This indicates that we skip the feature during the extraction process. Thus, if we want to determine the initial features for an apartment, we loop through the defined feature objects and consider only features in which the *appartmentFeature* is defined.

With the use of these filter options, we overcome the known cold start problem of a content-based RS described in section 2.3.2. The next step is to refine the features and their values by using the swipeable cards to provide better recommendations for a user.

4.4.2 Swipeable Cards

As described in the previous section 4.4.1, for the first recommendations, we use the answers from the customer's tutorial described in section 4.4.1. Next, we use the user's feedback related to the recommended real estates represented as swipeable cards. The user can swipe a card to the left or right and indicates if she or he likes or dislikes a real estate. We use these interactions to further improve the features of the user. The process looks as follows:

1. **Real estates as swipeable cards:** First, we calculate recommendations for a user based on her or his filter options as described in 4.4.1. The customer always uses the received real estates and presents them to the user as swipeable cards.

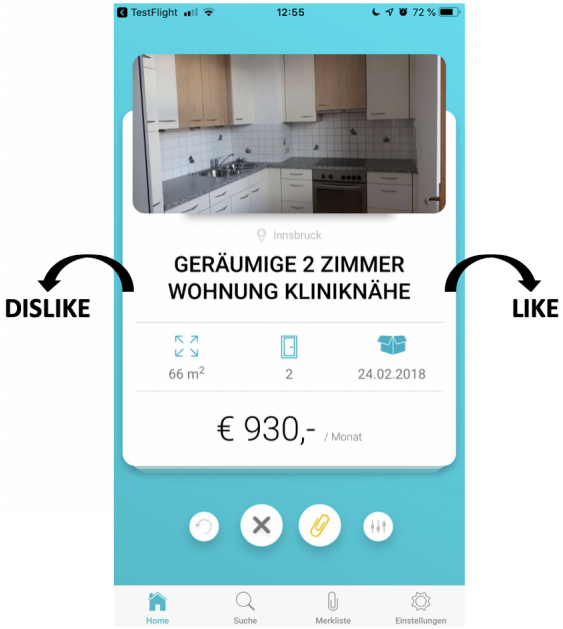

Figure 4.4: Real estate represented as swipeable card
Source: Created by author based on screenshots from Immoky. (Zangerl, 2018)

2. **User's likes and dislikes:** Afterwards, the user analyses the presented real estates and likes or dislikes them by swiping the cards. In parallel, the customer saves the information of the interactions (likes and dislikes) for the feedback procedure afterwards. After the user has seen all real estate offers, the customer requests the feedback procedure with the *feedback* route and transmits the gathered likes and dislikes. Thus, the call is used to determine new features for one particular user and object type.

3. **Feature Determination:** Next, we determine new features and calculate new values for already included ones. We add a feature to the user's preferences if it misses and is provided through a liked real estate. For the value update, we consider the liked flag and the corresponding real estate value of the feature. Afterwards, we send back the new features to the customer.

4. **Updating user profile:** Last, the customer overwrites the feature pro-

file related to the object type in the user profile. For the next recommendations, the customer uses the new features.

Feature Update

In this section, we investigate the feature update process. Therefore, we use the feedback from the user regarding real estates.

For the feature update, the customer invokes the *feedback* route of the RS described in section 4.2. The necessary data (information on likes and dislikes) is provided through the body of the request. Therefore, we use the same format for data, as for the recommendation call and extend the real estate object with a Boolean property called *liked* (see listing 4.13).

```
{
  userId: ID,
  features: [FeatureType],
  realEstates: [
    {
      id: ID,
      liked: Boolean
      features: [FeatureType]
    }
  ]
}
```

Listing 4.13: Body-format used for the feedback request

In the API of the RS we parse the body and execute the feedback procedure with the user features and real estates. Therefore, we iterate through the real estates and and features. Since a real estate can have more features than the amount in the user's feature set, we have two scenarios for the feature update. In the first scenario, a real estate feature is not included in the user's feature set, in the second one it is. The former case happens most likely when a user uses the mobile application for the first time and thus

has only a few features in her or his set (e.g. answers from the tutorial). On the basis of these scenarios we implement two methods to determine the value for a feature.

Feature is not included: First, we check if the feature of the real estate is already included in the user's feature set. Next, we check if the current real estate was either liked or disliked by the user. Since a dislike indicates that the user does not like the attributes of the real estate, we do not add the feature to the user's feature set. If the user liked the real estate, however, we add the feature to the user's feature set with its value.

Feature is included: If the feature of a real estate is in the user's feature set, we update the value. Our standard approach for the calculation is based on the mean value between the user's and real estate's value. Next, we multiply the determined mean value with a scaling factor and add the result to the user value. The used scaling factor depends on the liked flag of the real estate. We take 0.5 if the real estate was liked and -0.2 if it was not. A calculation which looks for a feature of the type number looks as follows:

For this example, we use the property price as a feature, choose €550 as user value and €600 as real estate's value. Furthermore, in this example the user liked the real estate. Thus, the scaling factor is 0.5 which means that we move the value half to the mean value.

1. **Mean Value:**

$$\bar{x} = \frac{user_j + realestate_{i,j}}{2} = \frac{550 + 600}{2} = 575$$

   i...index of current real estate, j ... index of the current feature of the real estate

2. **Delta User Value/Mean Value:**

$$\Delta x = \bar{x} - user_j = 575 - 550 = 25$$

3. **Scale delta and add to user value:**

$$user_j = user_j + (\Delta x * 0.5) = 550 + (25 * 0.5) = \underline{\underline{562.5}}$$

The result of the equations, 562.5, is then used as the new value for the user feature. Since a feature can occur in multiple real estates, we repeat the calculations as often as the feature occurs. Thus, the final value which we return to the customer is the one of the last calculation.

The example for the value update stated before is based on numeric values. However, since a feature can have different types, we adopt the calculations according to the type. Therefore, we use the following approaches based on the feature type:

- **Number:** The calculation for numeric values looks like the example we showed before, only the scaling factor changes to -0.2 if the user disliked a real estate.

- **Date:** A feature of the date type is used to determine the days between the feature's value and today's date. In case of this type, we compare the user value with the real estate. Therefore, we use the mean value, the middle of the days, again for the calculation. Afterwards, we add the scaled mean value, dependent on the liked flag, to the user's date.

- **Range Days:** As with a date feature, we determine the days between two dates again. But instead of using today's date, we take the real estate's date as second value to the user's value. The calculation for the new value looks the same as for the date type. We determine the mean value and add the scaled mean value to the user's date.

- **In-Range:** A feature of the in-range type is used for real estates to define a minimum and maximum border such as the level from/to. We use for the recommendation calculation Boolean values. If in range, the value is 1, otherwise it is 0. For the update, we use the middle of the borders as mean value and apply our standard approach.

- **Address:** The address property is used to determine the range between a real estate and the desired location of the user in kilometres. Therefore, we use the Haversine formula to determine the distance between these addresses as mentioned in section 4.3.3. In our refine-

ment calculations, the middle between those two addresses represents the mean value. Afterwards, we apply the same calculations as for the previous feature types. Figure 4.5 provides a visual example showing in which direction the location is moved based on the scaling factor.

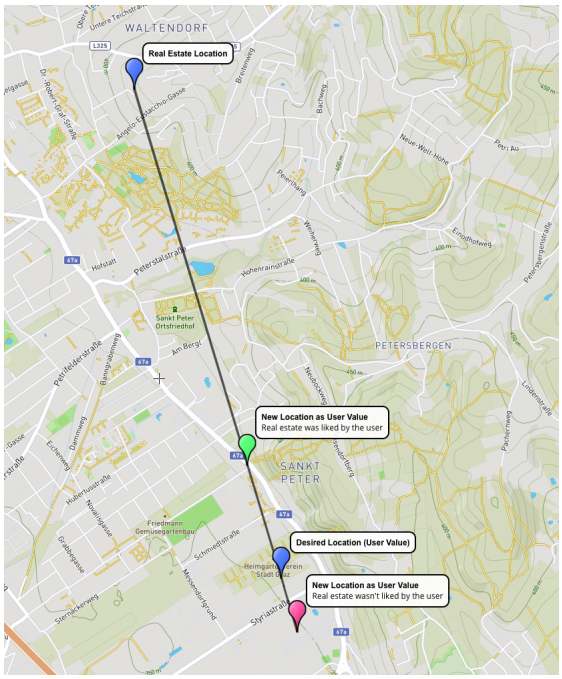

Figure 4.5: Example: New location for user value based on liked flag
Source: Created by author.

As seen in figure 4.5 we used two points to represent the real estate's and the user's location before the calculation. The green point indicates a movement towards the real estate's address when it was liked and the red one vice versa. The green point was moved further away from the user's old address since the scaling factor for a liked real estate is higher than for a disliked one.

- **Enumerate:** For the enumerate type we use a completely different approach for the feature value determination. As described in section 4.3.3 we use an array and a counter to define several characteristics within a feature.

```
{
  feature: identifier of feature,
  type: "enum",
  value: [
    {
      name: identifier of enumerate feature,
      count: current value
    }
  ]
}
```

Listing 4.14: Recap: Feature of enumerate type

On the basis of the liked flag, we adopt the counter of the entries in the feature value. When the real estate was liked, we increase the counter and otherwise we decrement it. We apply this step to all enumeration entries in the real estate's value. As a basis for the counter we use the old value array of the user's feature. If an enumerate does not exist in the user's feature we use 0 as a basis for the counter instead.

We use the following two values to describe the feature adoption if the real estate was liked (liked flag = 1).

Real estate value:

```
{
  feature: "equipment",
  type: "enum",
  value: ["builtInKitchen", "basement", "
     fullyFurnished"]
}
```

Listing 4.15: Enumerate feature *equipment* of real estate

User value before adoption:

```
{
  feature: "equipment",
```

```
3     type: "enum",
4     value: [
5       {
6         name: "builtInKitchen",
7         count: 5
8       },
9       {
10        name: "basement",
11        count: 1
12      }
13    ]
14  }
```

Listing 4.16: Enumerate feature *equipment* of user before adoptionn

User value after adoption: The entries and their counters of the user's value will be increased by 1 since the real estate was liked by the user.

```
1   {
2     feature: "equipment",
3     type: "enum",
4     value: [
5       {
6         name: "builtInKitchen",
7         count: 6
8       },
9       {
10        name: "basement",
11        count: 2
12      },
13      {
14        name: "fullyFurnished",
15        count: 1
16      }
17    ]
18  }
```

Listing 4.17: Enumerate feature *equipment* of user after adoption

The counter of the entries *builtInKitchen* and *basement* were increased by one while a counter of 1 was added to the *fullyFurnished* since it was not present beforehand.

5 Evaluation and Results

As a first step we implement the RS for real estates. Next, we evaluate the test results of our used approach based on our defined research questions and goals. Since we implemented the system in a modular and integrative way, we did not expect any issues during the final tests (modular and integrative in terms of the implementation, validation and verification process). Due to our proposed pipeline software architecture, we were able to implement and test every component separately and in an integrative manner with the already implemented components.

For our tests we built an offline experiment (Shani and Guy, 2010) (Gunawardana and Shani, 2015) since when the evaluation was carried out, Immoky only included the RS in its system without having the user interface finished. Thus, it was necessary to build a test scenario for participants to get a valuable amount of test data which we used as historical data to simulate a user's behaviour (Gunawardana and Shani, 2015). The test scenario includes the following steps:

1. **Filter options as initial features:** First, a participant defines filter options as initial features for the RS. Thus, she or he was asked to share the maximal rental fee, footage and type of real estate she or he prefers.

2. **Searching real estates:** The next step is to search for real estates on an arbitrary broker website which meet the preferences of the participant including a tolerance limit of about 30 percent. This means if the user prefers a flat with a footage of 50 square metres, we also take a flat with 35 and 65 square metres into account. Next, we transfer the real estate offers including their features except for images and description into a table from which we extract the data for the RS afterwards. For

one test scenario we search for 50 real estates.

3. **Participant evaluates real estates:** After we gathered the real estates, we show them to the participant. She or he evaluates then all of the offers and marks them as liked or disliked.

4. **Real estates data to JSON:** Next, we extract all real estates including their features and likes/dislikes from the table we created before and convert them into a JSON object. For the JSON object we use the defined format for the RS as seen in listing 4.7.

5. **Calculate recommendations:** The RS then determines the similarity for the real estates with the preferences defined in the step *filter options as initial features*. As a result, we get the IDs of the real estates in an ordered way.

6. **Evaluate recommendations:** For the evaluation, we consider the first 10 as recommended and the remaining 40 as not recommended. Finally, we calculate precision and recall (Gunawardana and Shani, 2015). Precision is the amount of liked real estates in the recommended set while recall refers to the fraction of liked real estates from the recommended set over the total amount of liked real estates (Gunawardana and Shani, 2015) (see table 5.1 and equations 5.1 and 5.2).

	Recommended	**Not recommended**
Liked	True-positive (tp)	False-negative (fn)
Not liked	False-positive (fp)	True-negative (tn)

Table 5.1: Classification of a recommendation

$$precision = \frac{\#tp}{\#tp + \#fp} \tag{5.1}$$

$$recall = \frac{\#tp}{\#tp + \#fn} \tag{5.2}$$

7. **New values for features via feedback:** First we evaluate the result and after that execute the feedback function to calculate new values for the participant's features. Therefore, we take the 10 recommended real estates and forward them to the RS with the likes/dislikes we already gathered from the participant. The new features are then used for the next recommendation cycle.

The recommendation cycle including the steps *calculate recommendations*, *evaluate recommendations* and *new values for features via feedback* is executed three times. As a result, the accuracy should increase since the RS can refine the features by using the feedback from the user.

In the next section, we show an example of the described iterative scenario. First, we define the setup of the scenario. Next, we calculate rankings and evaluate the first recommendation results based on the participant's likes and dislikes. In the last two steps, we adopt the features of the participant via the feedback function for a further recommendation calculation. At the end of a recommendation cycle, we calculate precision and recall to state if the accuracy increases.

5.1 Test Scenario

For the test scenario of this section we asked a student from Graz, Austria who was searching for a flat while studying at university to be our test participant. Table 5.2 states the participant's preferences for the flat search. We use these preferences as initial features for the first recommendation cycle.

	Max. rental fee	Footage	Real estate type	Location
Student	€600	50 m²	Flat	Inffeldgasse 8010 Graz

Table 5.2: Participant's preferences for test scenario

Next, we searched for 50 real estates and used the available features on the website for the RS. We did not consider images and arbitrary texts such as the description for the ranking calculations. If a feature was not available, we used NaN as value for the real estate instead. Table 5.3 states the number of real estates, liked/not liked real estate offers and features of the test scenario.

	Searched real estates	Liked	Not liked	Features
Test scenario	50	22	28	11

Table 5.3: Amount of real estates, liked/not liked and features of the test scenario

Since the approach of this work is one of the customers unique selling propositions, we are not allowed to publish the feature combination and their weightings which were used for the test scenario. Thus, we include only the rental fee, the footage and the location in the stated results.

5.1.1 Cycle 1

In the first recommendation step we used the defined preferences from the user as initial features for the recommendation request (see table 5.4).

	Rental fee	Footage	Location
Student	€600	50 m²	Inffeldgasse 8010 Graz

Table 5.4: Features used for the recommendation request of cycle 1

The values used for precision 5.3 and recall 5.4 are based on the results stated in table 5.6.

$$precision = \frac{\#tp}{\#tp + \#fp} = \frac{8}{8+2} = 0.8 \tag{5.3}$$

$$recall = \frac{\#tp}{\#tp + \#fn} = \frac{8}{8+14} = 0.364 \tag{5.4}$$

For the feedback request of the first cycle, we used the features from the recommendation request and the recommended real estates (see tables 5.4 and 5.6). The new values for the rental fee, footage and location are shown in table 5.5.

	Rental fee	Footage	Location
Student	€630.1	51.16 m²	Petersgasse 8010 Graz

Table 5.5: New feature values after the feedback request of cycle 1

The values for the rental fee and footage increased which means the participant tends to a bigger flat and is also prepared to pay more. Also, the location changed and moved towards the centre of Graz.

ID	Rental fee	Footage	Location	Similarity	Liked
14	590	49	Petrifelderstrasse, 8041 Graz	0.00441	false
31	592	50	Anton Wildgangs-Weg 15, 8043 Graz	0.00664	true
46	629	50	St.Peter Hauptstrasse, 8042 Graz	0.00714	true
9	581	52	Neufeldweg 75, 8010 Graz	0.00715	true
10	620	53	Neufeldweg 75, 8010 Graz	0.00952	true
26	640	49	Niesenbergergasse 41, 8020 Graz	0.01098	false
48	649	51	Muenzgrabenstrasse, 8010 Graz	0.01148	true
13	649	51	Grazbachgasse, 8010 Graz	0.01178	true
47	649	51	Wartingergasse 30, 8010 Graz	0.01259	true
15	630	54	Petrifelderstrasse, 8041 Graz	0.01315	true
49	592	53	Mariatrost, 8044 Graz	0.01354	true
21	655	49	Billrothgasse, 8010 Graz	0.01368	true
23	549	48	Florianigasse, 8020 Graz	0.01376	false
28	652	48	Niesenbergergasse 41, 8020 Graz	0.01413	false
12	610	45	Geidorf, 8010 Graz	0.01458	false
45	650	53	Muehlgasse 60, 8010 Graz	0.0151	true
32	670	51	Plueddemanngasse, 8010 Graz	0.01615	true
22	670	50	Billrothgasse, 8010 Graz	0.01661	true
30	670	51	Wartingergasse 30, 8010 Graz	0.01694	true
35	650	54	Mariengasse, 8020 Graz	0.0171	true
50	676	51	Dietrichsteinplatz, 8010 Graz	0.0176	true
43	682	51	Obere Teichstrasse 59, 8010 Graz	0.01902	true
7	561	56	Annenstrasse 7/7a, 8020 Graz	0.01921	false
2	539	55	Elisabethinergasse, 8020 Graz	0.02007	false
25	583	57	Oeverseegasse, 8020 Graz	0.02009	false
27	687	52	Niesenbergergasse 41, 8020 Graz	0.0212	false
24	523	46	Florianigasse, 8020 Graz	0.02122	false
36	690	48	Augasse, 8020 Graz	0.02269	true
1	700	53	Geidorf, 8010 Graz	0.02452	true
29	500	47	Koenigshoferstrasse 23, 8020 Graz	0.02533	false
6	700	54	Keesgasse 4, 8010 Graz	0.02538	true
17	700	54	Keesgasse 4, 8010 Graz	0.02538	true
11	710	56	Geidorf, 8010 Graz	0.03019	true
39	529	41	Algersdorferstrasse, 8020 Graz	0.03067	false
40	519	41	Hauseggerstrasse 45, 8020 Graz	0.03173	false
8	576	61	Krottendorfer Strasse 50, 8052 Graz	0.03196	false
20	670	60	Vinzenzgasse, 8020 Graz	0.03267	false
4	485	41	Wielandgasse, 8010 Graz	0.03599	false
33	499	60	Josef Huber Gasse 12, 8010 Graz	0.03603	false
37	469	42	Kossgasse, 8010 Graz	0.03692	false
3	529	37	Geidorf, 8010 Graz	0.03932	false
5	644	64	Handelstrasse 10/10a, 8020 Graz	0.04049	false
44	505	35	Schillerplatz, 8010 Graz	0.04642	false
38	740	62	Algersdorferstrasse, 8020 Graz	0.04648	false
42	482	36	Lazarettguertel, 8020 Graz	0.04697	false
16	480	36	Petrifelderstrasse, 8041 Graz	0.04708	false
19	470	36	Kossgasse, 8010 Graz	0.04836	false
34	742	65	Wielandgasse, 8010 Graz	0.05226	false
18	750	65	Keesgasse 4, 8010 Graz	0.05341	false
41	714	57	Peter-Rosegger-Strasse 117	0.99985	false

Table 5.6: Recommendations of cycle 1 (green rows indicate recommended real estates and red ones not recommended)

5.1.2 Cycle 2

For the second recommendation step we used the adopted features from the feedback request done in the first cycle (see table 5.7).

	Rental fee	Footage	Location
Student	€630.1	51.16 m²	Petersgasse 8010 Graz

Table 5.7: Features used for the recommendation request of cycle 2

The values used for precision 5.5 and recall 5.6 are again based on the results from the recommendation request stated in table 5.9.

$$precision = \frac{\#tp}{\#tp + \#fp} = \frac{6}{6+4} = 0.6 \tag{5.5}$$

$$recall = \frac{\#tp}{\#tp + \#fn} = \frac{6}{6+16} = 0.273 \tag{5.6}$$

For the feature adoption we executed the same scenario as in the first cycle 5.1.1. However, we did not use the original user data but the feedback we got in cycle 1 (see tables 5.7 and 5.9).

	Rental fee	Footage	Location
Student	€662.74	50.33 m²	Brandhofgasse 8010 Graz

Table 5.8: New feature values after the feedback request of cycle 2

In the second feedback request, only the rental fee changed significantly and increased about €30. The footage and the location only slightly changed. Thus, we assume that the participant is prepared to pay more than the initial maximum rental fee to get her or his desired real estate.

ID	Rental fee	Footage	Location	Similarity	Liked
31	592	50	Anton Wildgangs-Weg 15, 8043 Graz	1.18522	true
48	649	51	Muenzgrabenstrasse, 8010 Graz	1.1913	true
13	649	51	Grazbachgasse, 8010 Graz	1.1913	true
30	670	51	Wartingergasse 30, 8010 Graz	1.19133	true
23	549	48	Florianigasse, 8020 Graz	1.19159	false
24	523	46	Florianigasse, 8020 Graz	1.19164	false
7	561	56	Annenstrasse 7/7a, 8020 Graz	1.19184	false
36	690	48	Augasse, 8020 Graz	1.19217	true
8	576	61	Krottendorfer Strasse 50, 8052 Graz	1.19224	false
1	700	53	Geidorf, 8010 Graz	1.19225	true
35	650	54	Mariengasse, 8020 Graz	1.19236	true
29	500	47	Koenigshoferstrasse 23, 8020 Graz	1.19248	false
4	485	41	Wielandgasse, 8010 Graz	1.19292	false
19	470	36	Kossgasse, 8010 Graz	1.19297	false
37	469	42	Kossgasse, 8010 Graz	1.19308	false
33	499	60	Josef Huber Gasse 12, 8010 Graz	1.19348	false
40	519	41	Hauseggerstrasse 45, 8020 Graz	1.19382	false
46	629	50	St.Peter Hauptstrasse, 8042 Graz	1.19851	true
47	649	51	Wartingergasse 30, 8010 Graz	1.19852	true
26	640	49	Niesenbergergasse 41, 8020 Graz	1.19853	false
45	650	53	Muehlgasse 60, 8010 Graz	1.19854	true
32	670	51	Plueddemanngasse, 8010 Graz	1.19855	true
15	630	54	Petrifelderstrasse, 8041 Graz	1.19855	true
14	590	49	Petrifelderstrasse, 8041 Graz	1.19856	false
10	620	53	Neufeldweg 75, 8010 Graz	1.19856	true
50	676	51	Dietrichsteinplatz, 8010 Graz	1.19857	true
28	652	48	Niesenbergergasse 41, 8020 Graz	1.19857	false
9	581	52	Neufeldweg 75, 8010 Graz	1.19858	true
27	687	52	Niesenbergergasse 41, 8020 Graz	1.19858	false
49	592	53	Mariatrost, 8044 Graz	1.1986	true
17	700	54	Keesgasse 4, 8010 Graz	1.19864	true
6	700	54	Keesgasse 4, 8010 Graz	1.1987	true
11	710	56	Geidorf, 8010 Graz	1.19874	true
43	682	51	Obere Teichstrasse 59, 8010 Graz	1.19891	true
12	610	45	Geidorf, 8010 Graz	1.19891	false
18	750	65	Keesgasse 4, 8010 Graz	1.19945	false
3	529	37	Geidorf, 8010 Graz	1.19947	false
34	742	65	Wielandgasse, 8010 Graz	1.19955	false
16	480	36	Petrifelderstrasse, 8041 Graz	1.19972	false
44	505	35	Schillerplatz, 8010 Graz	1.19984	false
39	529	41	Algersdorferstrasse, 8020 Graz	1.19988	false
38	740	62	Algersdorferstrasse, 8020 Graz	1.19996	false
42	482	36	Lazarettguertel, 8020 Graz	1.20082	false
2	539	55	Elisabethinergasse, 8020 Graz	1.20229	false
22	670	50	Billrothgasse, 8010 Graz	1.21206	true
21	655	49	Billrothgasse, 8010 Graz	1.21211	true
20	670	60	Vinzenzgasse, 8020 Graz	1.21247	false
5	644	64	Handelstrasse 10/10a, 8020 Graz	1.24174	false
25	583	57	Oeverseegasse, 8020 Graz	1.2513	false
41	714	57	Peter-Rosegger-Strasse 117	1.56159	false

Table 5.9: Recommendations of cycle 2 (green rows indicate recommended real estates and red ones not recommended)

5.1.3 Cycle 3

In the last recommendation step we used the features from the feedback request of the previous cycle 5.1.2 5.1.2 to find the similarities for the real estate offers (see table 5.10).

	Rental fee	Footage	Location
Student	€662.74	50.33 m²	Brandhofgasse 8010 Graz

Table 5.10: Features used for the recommendation request of cycle 3

The values used for precision 5.7 and recall 5.8 are based on the results of the recommendation request stated in table 5.12.

$$precision = \frac{\#tp}{\#tp + \#fp} = \frac{8}{8+2} = 0.8 \tag{5.7}$$

$$recall = \frac{\#tp}{\#tp + \#fn} = \frac{8}{8+14} = 0.364 \tag{5.8}$$

For the third feedback request we also used the features stated in table 5.10 and the recommended real estates as shown in table 5.12.

	Rental fee	Footage	Location
Student	€665.66	51.15 m²	Hartiggasse 8010 Graz

Table 5.11: New feature values after the feedback request of cycle 3

In the third cycle we reached a point at which all features had changed only slightly. Based on the stated results and a further feedback step we conclude that the features and thus the participant's preferences converge starting from the third feedback request.

ID	Rental fee	Footage	Location	Similarity	Liked
31	592	50	Anton Wildgangs-Weg 15, 8043 Graz	1.31767	true
30	670	51	Wartingergasse 30, 8010 Graz	1.32317	true
13	649	51	Grazbachgasse, 8010 Graz	1.3232	true
48	649	51	Muenzgrabenstrasse, 8010 Graz	1.3232	true
1	700	53	Geidorf, 8010 Graz	1.32389	true
36	690	48	Augasse, 8020 Graz	1.32389	true
35	650	54	Mariengasse, 8020 Graz	1.32412	true
29	500	47	Koenigshoferstrasse 23, 8020 Graz	1.32443	false
19	470	36	Kossgasse, 8010 Graz	1.32474	false
32	670	51	Plueddemanngasse, 8010 Graz	1.32966	true
28	652	48	Niesenbergergasse 41, 8020 Graz	1.32968	false
47	649	51	Wartingergasse 30, 8010 Graz	1.32968	true
45	650	53	Muehlgasse 60, 8010 Graz	1.32968	true
26	640	49	Niesenbergergasse 41, 8020 Graz	1.32969	false
27	687	52	Niesenbergergasse 41, 8020 Graz	1.3297	false
46	629	50	St.Peter Hauptstrasse, 8042 Graz	1.32972	true
6	700	54	Keesgasse 4, 8010 Graz	1.32972	true
15	630	54	Petrifelderstrasse, 8041 Graz	1.32972	true
17	700	54	Keesgasse 4, 8010 Graz	1.32973	true
50	676	51	Dietrichsteinplatz, 8010 Graz	1.32973	true
14	590	49	Petrifelderstrasse, 8041 Graz	1.32978	false
11	710	56	Geidorf, 8010 Graz	1.32979	true
43	682	51	Obere Teichstrasse 59, 8010 Graz	1.32983	true
10	620	53	Neufeldweg 75, 8010 Graz	1.32984	true
9	581	52	Neufeldweg 75, 8010 Graz	1.32987	true
12	610	45	Geidorf, 8010 Graz	1.3299	false
18	750	65	Keesgasse 4, 8010 Graz	1.33041	false
38	740	62	Algersdorferstrasse, 8020 Graz	1.33066	false
3	529	37	Geidorf, 8010 Graz	1.33073	false
39	529	41	Algersdorferstrasse, 8020 Graz	1.33075	false
44	505	35	Schillerplatz, 8010 Graz	1.33086	false
16	480	36	Petrifelderstrasse, 8041 Graz	1.33089	false
42	482	36	Lazarettguertel, 8020 Graz	1.33159	false
23	549	48	Florianigasse, 8020 Graz	1.33998	false
24	523	46	Florianigasse, 8020 Graz	1.33999	false
7	561	56	Annenstrasse 7/7a, 8020 Graz	1.3411	false
37	469	42	Kossgasse, 8010 Graz	1.34142	false
8	576	61	Krottendorfer Strasse 50, 8052 Graz	1.34151	false
40	519	41	Hauseggerstrasse 45, 8020 Graz	1.34169	false
33	499	60	Josef Huber Gasse 12, 8010 Graz	1.34189	false
4	485	41	Wielandgasse, 8010 Graz	1.342	false
20	670	60	Vinzenzgasse, 8020 Graz	1.34242	false
49	592	53	Mariatrost, 8044 Graz	1.34605	true
34	742	65	Wielandgasse, 8010 Graz	1.34693	false
2	539	55	Elisabethinergasse, 8020 Graz	1.35001	false
22	670	50	Billrothgasse, 8010 Graz	1.35792	true
21	655	49	Billrothgasse, 8010 Graz	1.35795	true
5	644	64	Handelstrasse 10/10a, 8020 Graz	1.38462	false
25	583	57	Oeverseegasse, 8020 Graz	1.41003	false
41	714	57	Peter-Rosegger-Strasse 117	1.66419	false

Table 5.12: Recommendations of cycle 3 (green rows indicate recommended real estates and red ones not recommended)

6 Discussion and Conclusion

In this thesis we implemented a CBF-based RS for real estates in order to facilitate and speed up the search procedure on real estate broker applications. We identified three research questions which represent the goal of the RS. On the basis of the previous chapters and our test results, we discuss and conclude our achievements, the challenges we faced and our future work to improve the recommender system.

First, we investigate the three research questions and use the test results to show if we achieved an accuracy of at least 70 %. Last, we refer to the limitations of the approach and state the content of our future work.

The first research question is fundamental for this work since it determines whether we are able to implement an RS for real estates at all. Our first actions were to analyse the content which is available on real estate broker websites. As a result, we identified the most commonly used listing information and search/order options. Later on we used this data as content information for real estates and initial features for our RS. Knowing this, we researched the major RS approaches to examine those which are based on content. As described in the RS section 2.3 of our work, there are two possible approaches. The first one is CBF which determines the similarity between an item's content and the user's preferences reflected as item properties. The second approach is KBR which recommends items based on the knowledge about the item domain and their content. The former approach is the one we used for this work.

Next, we focused on research question number two which questions if a CBF RS is suited to recommending real estates. We chose this question and used the word *suited* for our measurement since we want to use it for

commercial purposes. To measure whether a CBF RS is suitable, we defined an accuracy of 70 %. This means a user has to like at least seven of ten recommended real estates. A like indicates that the real estate meets the preferences of the user.

The results we gained from research question number one, showed that there is content available to describe a real estate. To feel fully confident with implementing a CBF RS, we generated a little test example and did the calculations by hand to prove that our proposed approach works. Therefore, we created about fifteen test records without missing values, included only a few features such as the rental fee or footage and marked them with a liked flag. Already after the first cycle the results were convincing. Thus, we kept going with the project.

After we verified that it is possible to recommend real estates using a CPF approach, the next step was to analyse the content of a real estate. Therefore, we categorised the content and created feature types to make the features interpretable for an RS. We applied the feature extraction method to a few test sets we got from our customer and identified two challenges: missing values and features with different norms. On the basis of the research we had done before we knew that these two challenges are common issues in the context of RS. In order to tackle the problem of missing values we calculated the mean value of the corresponding feature and for features with different norms, we applied the normalisation function. As a result, we got a matrix with no missing values and a single norm which we further used to determine the Euclidean distance as similarity between the user preferences represented as item vector and the real estates. Thus, it is possible to recommend real estates using a CBF approach.

Coming to the accuracy, we achieved with our implemented approach. Therefore, we refer to chapter 5 and the described test scenario within. We executed three recommendation steps on a test set of 50 real estates. In the first step we already achieved a precision of 0.8 (= accuracy of 80 %) with three features. After the first feedback request which adopted and advanced of the initial three features, the accuracy decreased to 60 %. This mainly happened because of the new features added to the user's feature set. However, in the final recommendation step, we achieved an accuracy of 80 % again since the features which were added before were adapted in

the second cycle. We also executed the whole process a fourth time and did not identify any further changes as already mentioned in the last section of chapter 5.

On the basis of the research we conducted, we proved that it is possible to recommend real estates with an accuracy of at least 70 % using a CBF approach.

Finally, we investigated the feedback topic of an RS which is the last question of this thesis. In general, the purpose of an RS is to recommend items which meet the preferences of a user as much as possible. As described in chapter 2 an RS interprets the user's interactions with the system to improve its parameters. In case of this work, a user likes or dislikes recommended real estates. We regard a like as a positive feedback and a dislike as a negative one. Thus, we have two types of information which we used for the parametrisation of the RS: the content of the real estate and the corresponding liked information from the user. Based on this information we either move the user's features in the direction of a real estate or vice versa. Since a user profile reflects the attributes of a real estate, we can easily adopt the features.

The test results stated in section 5.1 also showed that the feedback of the participant could have been interpreted without a loss of accuracy. In the first recommendation step only three features were used while in the following two the number of features increased to 11 reasoned by the feedback cycles. This means we reflect the participant's preferences on a higher-dimensional space. Also, the values of the initial features changed in the latter cycles which indicates that the participant is initially not able to define her or his preferences accurately.

In total, our chosen approach for the given domain was a success since we could answer all research questions positively and also exceeded the indented accuracy of at least 70%. The next section focuses on some improvements we want to implement in the future.

6.1 Future Work

In the last section we state the content of our future work. In our future work we focus on improving two different parts: first, the features and second, the feedback process.

During our tests we identified some issues with the feature selection and weighting. The initial setup of this work included a feature extraction component and three data preprocessing components. However, none of these components reduced the dimension of the features or identified features which were not relevant for a user. Since we used a software architecture which eases the extension of the recommender system, we would implement a correlation filter and a feature selection filter in a future system. The former component filters features based on a correlation matrix. If a feature correlates strongly with another, the feature will be removed from the feature set. For the latter one, we intend to use a liked real estate set and calculate the variance of the features. If a feature varies strongly, we remove the feature from the feature set since we assume that a high variance indicates non-relevance for a user. For example, a user liked several real estates with many different building types such as old, new or renovated. Furthermore, we want to implement two different types of enumeration values. The types should indicate if a real estate can have single or multiple values of a feature. For example, a real estate can only have one building type while it can have several furnishings.

The last part of our future work relates to the feedback process. Currently, we are using two scaling factors for the feature adoption. 0.5 of the mean value between the user and real estate value if the real estate was liked and -0.2 if not. At this point, we want to try different scaling factors for different feature types and other methods if a real estate was disliked.

Bibliography

Abts, Dietmar (2015). "REST-basierte Web Services mit JAX-RS." In: *Masterkurs Client/Server-Programmierung mit Java*. Wiesbaden: Springer Fachmedien Wiesbaden, pp. 277–330. DOI: 10.1007/978-3-658-09921-3_10. URL: http://link.springer.com/10.1007/978-3-658-09921-3%7B%5C_%7D10 (cit. on p. 27).

Agarwal, A and M Chauhan (2017). "Similarity Measures used in Recommender Systems: A Study." In: URL: http://ijetsr.com/images/short%7B%5C_%7Dpdf/1498555415%7B%5C_%7D619-626-ieteh326%7B%5C_%7Dijetsr.pdf (cit. on p. 21).

Amatriain, Xavier and Josep M. Pujol (2015). "Data Mining Methods for Recommender Systems." In: *Recommender Systems Handbook*. Boston, MA: Springer US, pp. 227–262. DOI: 10.1007/978-1-4899-7637-6_7. URL: http://link.springer.com/10.1007/978-1-4899-7637-6%7B%5C_%7D7 (cit. on pp. 22–24).

Asaithambi, Sudharsan (2017). *Why, How and When to Scale your Features – GreyAtom – Medium*. URL: https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e (visited on 12/23/2018) (cit. on p. 25).

Barragáns-Martínez, Ana Belén et al. (2010). "A hybrid content-based and item-based collaborative filtering approach to recommend TV programs enhanced with singular value decomposition." In: *Information Sciences* 180.22, pp. 4290–4311. ISSN: 0020-0255. DOI: 10.1016/J.INS.2010.07.024. URL: https://www.sciencedirect.com/science/article/pii/S0020025510003427%7B%5C#%7Dbib33 (cit. on p. 13).

Bobadilla, J., F. Ortega, et al. (2013). "Recommender systems survey." In: *Knowledge-Based Systems* 46, pp. 109–132. ISSN: 0950-7051. DOI: 10.1016/J.KNOSYS.2013.03.012. URL: https://www.sciencedirect.com/science/article/pii/S0950705113001044 (cit. on p. 10).

Bobadilla, J., F. Serradilla, and J. Bernal (2010). "A new collaborative filtering metric that improves the behavior of recommender systems." In: *Knowledge-Based Systems* 23.6, pp. 520–528. ISSN: 0950-7051. DOI: 10.1016/J.KNOSYS.2010.03.009. URL: https://www.sciencedirect.com/science/article/pii/S0950705110000444%7B%5C#%7Dsec1 (cit. on p. 11).

Bollegala, Danushka (2017). "Dynamic feature scaling for online learning of binary classifiers." In: *Knowledge-Based Systems* 129, pp. 97–105. ISSN: 0950-7051. DOI: 10.1016/J.KNOSYS.2017.05.010. URL: https://www.sciencedirect.com/science/article/pii/S0950705117302216 (cit. on p. 23).

Bond et al. (2000). *Uses of Websites for Effective Real Estate Marketing*. DOI: 10.2307/44154442. URL: https://www.jstor.org/stable/44154442 (cit. on p. 6).

Burke, Robin (2002). "Hybrid Recommender Systems: Survey and Experiments." In: *User Modeling and User-Adapted Interaction* 12.4, pp. 331–370. ISSN: 09241868. DOI: 10.1023/A:1021240730564. URL: http://link.springer.com/10.1023/A:1021240730564 (cit. on p. 19).

Cai, Yi et al. (2014). "Typicality-Based Collaborative Filtering Recommendation." In: *IEEE Transactions on Knowledge and Data Engineering* 26.3, pp. 766–779. ISSN: 1041-4347. DOI: 10.1109/TKDE.2013.7. URL: http://ieeexplore.ieee.org/document/6407132/ (cit. on pp. 11–13).

Chen, Xianjun et al. (2017). "Restful API Architecture Based on Laravel Framework." In: *Journal of Physics: Conference Series* 910.1, p. 012016. ISSN: 1742-6588. DOI: 10.1088/1742-6596/910/1/012016. URL: http://stacks.iop.org/1742-6596/910/i=1/a=012016?key=crossref.5ac4928dafd50f6aca2d9aad3b7ad8c0 (cit. on p. 27).

Felfernig, Alexander, Gerhard Friedrich, et al. (2015). "Constraint-Based Recommender Systems." In: *Recommender Systems Handbook*. Boston, MA: Springer US, pp. 161–190. DOI: 10.1007/978-1-4899-7637-6_5. URL: http://link.springer.com/10.1007/978-1-4899-7637-6%7B%5C_%7D5 (cit. on pp. 16, 18).

Felfernig, Alexander, Michael Jeran, et al. (2014). "Basic Approaches in Recommendation Systems." In: *Recommendation Systems in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 15–37. DOI: 10.1007/978-3-642-45135-5_2. URL: http://link.springer.com/10.1007/978-3-642-45135-5%7B%5C_%7D2 (cit. on pp. 9, 11–16, 18, 19).

Gemmis, Marco de et al. (2015). "Semantics-Aware Content-Based Recommender Systems." In: *Recommender Systems Handbook*. Boston, MA: Springer US, pp. 119–159. DOI: 10.1007/978-1-4899-7637-6_4. URL: http://link.springer.com/10.1007/978-1-4899-7637-6%7B%5C_%7D4 (cit. on p. 15).

Geymüller, Philipp and Michael Christl (2014). "Teurer Wohnen." In: *agenda-austria.at*. URL: https://www.agenda-austria.at/wp-content/uploads/2018/04/agenda-austria-studie-teurer-wohnen.pdf (cit. on p. 1).

Gunawardana, Asela and Guy Shani (2015). "Evaluating Recommender Systems." In: *Recommender Systems Handbook*. Boston, MA: Springer US, pp. 265–308. DOI: 10.1007/978-1-4899-7637-6_8. URL: http://link.springer.com/10.1007/978-1-4899-7637-6%7B%5C_%7D8 (cit. on pp. 67, 68).

Hernando, Antonio, Jesús Bobadilla, and Fernando Ortega (2016). "A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model." In: *Knowledge-Based Systems* 97, pp. 188–202. ISSN: 0950-7051. DOI: 10.1016/J.KNOSYS.2015.12.018. URL: https://www.sciencedirect.com/science/article/pii/S0950705115005006%7B%5C#%7Dtbl0009 (cit. on p. 10).

Ho, Hui-Ping, Ching-Ter Chang, and Cheng-Yuan Ku (2015). "House selection via the internet by considering homebuyers' risk attitudes with S-shaped utility functions." In: *European Journal of Operational Research* 241.1, pp. 188–201. ISSN: 0377-2217. DOI: 10.1016/J.EJOR.2014.08.009. URL: https://www.sciencedirect.com/science/article/pii/S0377221714006444 (cit. on pp. 6, 7).

Horstmann, Felix (2017). "Wohnraum für Studenten wird immer teurer." In: *WiSt - Wirtschaftswissenschaftliches Studium* 46.7-8, pp. 58–58. ISSN: 0340-1650. DOI: 10.15358/0340-1650-2017-7-8-58. URL: https://elibrary.vahlen.de/index.php?doi=10.15358/0340-1650-2017-7-8-58 (cit. on p. 1).

Jannach, Dietmar et al. (2010). "Collaborative recommendation." In: *Recommender Systems*. Cambridge: Cambridge University Press, pp. 13–50. DOI: 10.1017/CBO9780511763113.004. URL: https://www.cambridge.org/core/product/identifier/CBO9780511763113A014/type/book%7B%5C_%7Dpart (cit. on p. 11).

Kantardzic, Mehmed (2003). "Preparing the Data." In: *Data Mining*. IEEE. DOI: 10.1109/9780470544341.ch2. URL: http://ieeexplore.ieee.org/search/srchabstract.jsp?arnumber=5265980 (cit. on p. 25).

Kim, Heung-Nam et al. (2010). "Collaborative filtering based on collaborative tagging for enhancing the quality of recommendation." In: *Electronic Commerce Research and Applications* 9.1, pp. 73–83. ISSN: 1567-4223. DOI: 10.1016/J.ELERAP.2009.08.004. URL: https://www.sciencedirect.com/science/article/pii/S1567422309000544 (cit. on p. 12).

Koohi, Hamidreza and Kourosh Kiani (2016). "User based Collaborative Filtering using fuzzy C-means." In: *Measurement* 91, pp. 134–139. ISSN: 0263-2241. DOI: 10.1016/J.MEASUREMENT.2016.05.058. URL: https://www.sciencedirect.com/science/article/pii/S0263224116302159 (cit. on pp. 10–12).

Koren, Yehuda and Robert Bell (2015). "Advances in Collaborative Filtering." In: *Recommender Systems Handbook*. Boston, MA: Springer US, pp. 77–118. DOI: 10.1007/978-1-4899-7637-6_3. URL: http://link.springer.com/10.1007/978-1-4899-7637-6%7B%5C_%7D3 (cit. on p. 10).

Kumar, Ashish (2014). *Software Architecture Styles a Survey*. Tech. rep. 9, pp. 975–8887. URL: https://research.ijcaonline.org/volume87/number9/pxc3893768.pdf (cit. on p. 29).

Leichtgemacht (2016). *10 beliebte Immobilienportale aus Österreich - leichtgemacht.at Blog*. URL: http://blog.leichtgemacht.at/10-beliebte-immobilienportale-aus-oesterreich/ (visited on 12/06/2018) (cit. on p. 7).

Li, Wu-Jeng et al. (2018). "JustIoT Internet of Things based on the Firebase real-time database." In: *2018 IEEE International Conference on Smart Manufacturing, Industrial & Logistics Engineering (SMILE)*. IEEE, pp. 43–47. ISBN: 978-1-5386-3183-6. DOI: 10.1109/SMILE.2018.8353979. URL: https://ieeexplore.ieee.org/document/8353979/ (cit. on p. 33).

Liu, Fengkun and Hong Joo Lee (2010). "Use of social network information to enhance collaborative filtering performance." In: *Expert Systems with Applications* 37.7, pp. 4772–4778. ISSN: 0957-4174. DOI: 10.1016/J.ESWA.2009.12.061. URL: https://www.sciencedirect.com/science/article/pii/S0957417409011075 (cit. on pp. 11, 12).

Lopes Rosa, Renata et al. (2018). "A Knowledge-Based Recommendation System that includes Sentiment Analysis and Deep Learning." In: *IEEE Transactions on Industrial Informatics*, pp. 1–1. ISSN: 1551-3203. DOI: 10.

1109/TII.2018.2867174. URL: https://ieeexplore.ieee.org/document/8445585/ (cit. on p. 16).

Malawski, Maciej et al. (2017). "Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions." In: *Future Generation Computer Systems*. ISSN: 0167-739X. DOI: 10.1016/J.FUTURE.2017.10.029. URL: https://www.sciencedirect.com/science/article/pii/S0167739X1730047X (cit. on p. 33).

Mandl, Monika et al. (2011). "Consumer decision making in knowledge-based recommendation." In: *Journal of Intelligent Information Systems* 37.1, pp. 1–22. ISSN: 0925-9902. DOI: 10.1007/s10844-010-0134-3. URL: http://link.springer.com/10.1007/s10844-010-0134-3 (cit. on pp. 16, 17).

McSherry, David (2003). "Similarity and Compromise." In: *Case-Based Reasoning Research and Development*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 291–305. DOI: 10.1007/3-540-45006-8_24. URL: http://link.springer.com/10.1007/3-540-45006-8%7B%5C_%7D24 (cit. on p. 20).

Menzies, Tim (2014a). "Data Mining." In: *Recommendation Systems in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 39–75. DOI: 10.1007/978-3-642-45135-5_3. URL: http://link.springer.com/10.1007/978-3-642-45135-5%7B%5C_%7D3 (cit. on p. 5).

Menzies, Tim (2014b). "Data Mining." In: *Recommendation Systems in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 39–75. DOI: 10.1007/978-3-642-45135-5_3. URL: http://link.springer.com/10.1007/978-3-642-45135-5%7B%5C_%7D3 (cit. on pp. 22, 23).

Merriam-Webster (2018). *Real Estate — Definition of Real Estate by Merriam-Webster*. URL: https://www.merriam-webster.com/dictionary/real%20estate (visited on 11/30/2018) (cit. on p. 5).

Mohsin, Shafique Hijazee (2015). *Mastering google app engine : build robust and highly scalable web applications with Google app engine*. Packt Publishing. ISBN: 9781784396671 (cit. on p. 34).

Mundt, Alexis and Wagner Karin (2017). "Regionale Wohnungspreisindizes in Österreich – erste Erkenntnisse auf Basis hedonischer Modelle." In: *oenb.at*, pp. 28–47. URL: https://www.oenb.at/dam/jcr:229f337c-d877-4730-9a2c-6a0a2ee0a42a/stat%7B%5C_%7D2017%7B%5C_%7Dq1%7B%5C_%7Danalyse%7B%5C_%7Dmundt%7B%5C_%7Dwagner.pdf (cit. on p. 1).

NumPy (2018). *NumPy*. URL: http://www.numpy.org/ (visited on 12/23/2018) (cit. on p. 30).

O'Mahony, Michael P., Neil J. Hurley, and Guénolé C.M. Silvestre (2006). "Detecting noise in recommender system databases." In: *Proceedings of the 11th international conference on Intelligent user interfaces - IUI '06*. New York, New York, USA: ACM Press, p. 109. ISBN: 1595932879. DOI: 10.1145/1111449.1111477. URL: http://portal.acm.org/citation.cfm?doid=1111449.1111477 (cit. on p. 24).

Paradarami, Tulasi K., Nathaniel D. Bastian, and Jennifer L. Wightman (2017). "A hybrid recommender system using artificial neural networks." In: *Expert Systems with Applications* 83, pp. 300–313. ISSN: 0957-4174. DOI: 10.1016/J.ESWA.2017.04.046. URL: https://www.sciencedirect.com/science/article/pii/S0957417417302968 (cit. on p. 19).

Pazzani, Michael J. and Daniel Billsus (2007). "Content-Based Recommendation Systems." In: *The Adaptive Web*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 325–341. DOI: 10.1007/978-3-540-72079-9_10. URL: http://link.springer.com/10.1007/978-3-540-72079-9%7B%5C_%7D10 (cit. on pp. 13–15).

Philipps, Jan and Bernhard Rumpe (2014). "Refinement of Pipe-and-Filter Architectures." In: DOI: 10.1007/3-540-48119-2_8. arXiv: 1411.2414. URL: http://arxiv.org/abs/1411.2414%20http://dx.doi.org/10.1007/3-540-48119-2%7B%5C_%7D8 (cit. on p. 29).

Polamuri, Saimadhu (2015). *Five most popular similarity measures implementation in python*. URL: http://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/ (visited on 01/03/2019) (cit. on p. 21).

Postman (2018). *Postman — API Development Environment*. URL: https://www.getpostman.com/ (visited on 12/23/2018) (cit. on p. 28).

Protasiewicz, Jakub (2018). *Why Is Python So Good for AI, Machine Learning and Deep Learning?* URL: https://www.netguru.co/blog/why-is-python-so-good-for-ai-machine-learning-and-deep-learning (visited on 12/23/2018) (cit. on p. 30).

Python (2018). *Welcome to Python.org*. URL: https://www.python.org/ (visited on 12/23/2018) (cit. on p. 30).

Ricci, Francesco, Lior Rokach, and Bracha Shapira (2015). "Recommender Systems: Introduction and Challenges." In: *Recommender Systems Handbook*. Boston, MA: Springer US, pp. 1–34. DOI: 10.1007/978-1-4899-

7637-6_1. URL: http://link.springer.com/10.1007/978-1-4899-7637-6%7B%5C_%7D1 (cit. on pp. 9, 12, 16).

Ronacher, Armin (2018). *Flask: web development, one drop at a time*. URL: http://flask.pocoo.org/ (visited on 10/27/2018) (cit. on p. 37).

Schafer, J. Ben et al. (2007). "Collaborative Filtering Recommender Systems." In: *The Adaptive Web*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 291–324. DOI: 10.1007/978-3-540-72079-9_9. URL: http://link.springer.com/10.1007/978-3-540-72079-9%7B%5C_%7D9 (cit. on p. 10).

Scikit (2018). *scikit-learn: machine learning in Python*. URL: https://scikit-learn.org/stable/ (visited on 12/23/2018) (cit. on p. 30).

Shani, Guy and Guy (2010). "Tutorial on evaluating recommender systems." In: *Proceedings of the fourth ACM conference on Recommender systems - RecSys '10*. New York, New York, USA: ACM Press, p. 1. ISBN: 9781605589060. DOI: 10.1145/1864708.1864710. URL: http://portal.acm.org/citation.cfm?doid=1864708.1864710 (cit. on p. 67).

Smyth, Barry (2007). "Case-Based Recommendation." In: *The Adaptive Web*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 342–376. DOI: 10.1007/978-3-540-72079-9_11. URL: http://link.springer.com/10.1007/978-3-540-72079-9%7B%5C_%7D11 (cit. on pp. 18, 19).

Statista (2018). *Europäische Union & Euro-Zone: Urbanisierungsgrad von 2007 bis 2017*. URL: https://de.statista.com/statistik/daten/studie/249028/umfrage/urbanisierung-in-der-europaeischen-union-eu/ (visited on 10/19/2018) (cit. on p. 1).

Swalin, Alvira (2018). *How to Handle Missing Data – Towards Data Science*. URL: https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4 (visited on 12/23/2018) (cit. on p. 24).

Van Le, Thanh, Trong Nghia Truong, and Tran Vu Pham (2014). "A Content-Based Approach for User Profile Modeling and Matching on Social Networks." In: Springer, Cham, pp. 232–243. DOI: 10.1007/978-3-319-13365-2_21. URL: http://link.springer.com/10.1007/978-3-319-13365-2%7B%5C_%7D21 (cit. on p. 16).

Willhaben (2018). *Wohnung mieten oder vermieten Graz - willhaben*. URL: https://www.willhaben.at/iad/immobilien/mietwohnungen/steiermark/graz/ (visited on 11/01/2018) (cit. on p. 2).

Zangerl, Armin (2018). *Immoky Tutorial* (cit. on pp. 4, 58).

Zumpano, Leonard V., Ken H. Johnson, and Randy I. Anderson (2003). "Internet use and real estate brokerage market intermediation." In:

Journal of Housing Economics 12.2, pp. 134–150. ISSN: 1051-1377. DOI: 10.1016/S1051-1377(03)00018-4. URL: https://www.sciencedirect.com/science/article/pii/S1051137703000184 (cit. on p. 6).