NTNU
Norwegian University of
Science and Technology

YAPS.life

A Group Recommendation System for Real Estate

Kristian Elset Bø

Master of Science in Computer Science
Submission date: June 2018
Supervisor: Anders Kofod-Petersen, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Can you afford to buy a place of your own? If you could, how would you go about finding it? Due to rising real estate prices in cities all over the world, more people than ever choose to rent in shared accommodations to save on costs. With whom and where we live has a significant impact on our lives, yet despite its importance, it is still one of the most manual and laborious processes in our digital age.

In this thesis, a prototype of a group recommendation system for real estate is presented to solve this issue. The prototype uses the context of where the users are looking to move, their habits and personality, as well as what they are looking for in a property, to cluster them into match groups. Then it provides housing recommendations for the group from various data sources based on the group's preferences. The research has been conducted within the Design Science Paradigm.

Presented in this thesis, is the design and implementation of the prototype in addition to the state of the art of both group and real estate recommendation. It provides a quantitative analysis of the different clustering algorithms, and qualitative evaluations of the user interface and real estate recommendations. The analyses of the matching algorithms show that a cascading hybrid model doing an initial clustering with k-Means, followed by a final grouping with a modified version of k Nearest Neighbour performs best for this problem. Further, the results show that users have a positive experience with the prototype, both with the interface and the real estate recommendations. Several topics for future research have also been identified.

Sammendrag

Har du råd til å kjøpe et sted å bo? Hvis du kunne, hvordan ville du gå fram for å finne det? På grunn av stigende eiendomspriser i byer over hele verden velger flere enn noensinne å leie i bofelleskap for å senke kostnader. Med hvem og hvor vi bor har en stor innvirkning på våre liv, men til tross for det er det å finne et bra kollektiv fortsatt en av de mest manuelle og arbeidskrevende prosessene vi gjør.

I denne oppgaven presenteres en prototype av et gruppe-anbefalingssystem for bolig for å løse dette problemet. Prototypen bruker konteksten av hvor brukerne ønsker å flytte, deres vaner og personlighet, så vel som det de leter etter i en leilighet, for å sette dem sammen i grupper. Deretter gir systemet anbfalinger for hvor gruppen burde bosette seg utifra ulike datakilder basert på gruppens preferanser. Forskningen er utført i samsvar med Design Science Paradigmet.

Presentert i denne oppgaven, er utformingen og implementeringen av prototypen i tillegg til State-of-the-Art for både anbefaling for grupper og for bolig. En kvantitativ analyse av forskjellige gruppesammensetnings-algoritmer, samt kvalitative vurderinger av brukergrensesnittet og bolig-anbefalingene er også med. Analysenen av gruppesammensetnings-algoritmene viser at en hybridmodell, som gjør en innledende gruppering med k-Means, etterfulgt av en endelig gruppering med en modifisert versjon av k Nearest Neighbour virker best for dette problemet. Videre viser de kvalitative resultatene at brukerne har en positiv opplevelse med prototypen, både med grensesnittet og boliganbefalinger. Flere nye avenyer for videre forskning er også avdekket.

Preface

This thesis is the basis for the master degree in computer science for Kristian Elset Bø. The research has been conducted at the Norwegian University for Science and Technology and supervised by Anders Kofod-Pedersen.

I would like to thank my supervisor for his support for the research topic and the continual guidance he has given me throughout the project. Further, I would like to extend gratitude to my roommates, friends, and family for their support. I could not have done this without you.

Kristian Elset Bø
Trondheim, June 11th, 2018

Contents

List of Figures

List of Tables

Abbreviations

AS	Application Specific survey
API	Application Programming Interface
BI	Background Information survey
GRS	Group Recommendation System
REGRS	Real Estate Group Recommendation System
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
kNN	k-Nearest Neighbor
NTNU	Norwegian University of Science and Technology
RERS	Real Estate Recommendation System
RS	Recommendation System
SUS	System Usability Scale
QE	Quality Evaluation
UE	Usability Evaluation
URL	Uniform Resource Locator

Chapter 1

Introduction

All of us will sooner or later be in a position where we need to find a new place to live. Whether it is your first time moving out from your parents' house, your first place after college, or a new house for your expanding family, you need a way to search for and organize your housing options.

When moving, one often has a considerable amount of options to choose from. Luckily, recommendation systems have been developed to help us filter through such large amounts of options very successfully. Services like Netflix[1] for movies, Spotify[2] for music and Amazon[3] for consumer goods have developed increasingly sophisticated systems to help us sift through the endless options. In Real Estate, companies like Zillow(USA)[4], Finn.no(Norway)[5], and AirBnB[6] help us filter and sort housing options on criteria such as price, size, and neighborhood.

These platforms help thousands of people find a place to live every day, but none of them address this one fundamental fact. Fewer and fewer people live alone, and who we live with shape as much or more of our daily life as where we live. No platform helps us find new homes that are tailored toward groups.

The purpose of this thesis is to construct a platform that can group together users with similar living habits and real estate goals and help them find a place to live together. To do this, traditional recommendation system techniques, and

[1] `www.netflix.com`
[2] `www.spotify.com`
[3] `www.amazon.com`
[4] `www.zillow.com`
[5] `www.finn.no`
[6] `www.airbnb.com`

their group variants, have been implemented and applied to the real estate domain.

1.1 Background and Motivation

Where you live and who you live with has a huge impact on your day to day life. Despite this impact, most recommendation systems in real estate are little more than digital newspaper ads with some filters, and the renting process is often quite manual. All the exciting research being done in recommendation systems today, and seeing how these systems have such a great impact on user experience and end results in services like Netflix, Spotify, and Amazon, inspired the author to apply these techniques to the real estate domain.

Another aspect is group recommendation. This topic has been explored in the literature in regards to movies, music, and other classic recommendation domains, but to the author's knowledge, never in real estate. Real estate services like Finn.no and Hybel.no acts as open marketplaces where users can list their profiles as looking for roommates and initiate contact with other users, but there is no matching based on profile attributes.

1.2 Previous Work

This thesis is also inspired by the author's previous work during the course "Computer Science, Specialization Project (TDT4501)" at NTNU. During that course, the author produced a literature review of recommendation systems in real estate as well as a state of the art review of group recommendation. The literature review uncovered several different approaches to recommendation in real estate. Daly et al. [2014a] focused solely on commuter distance and price, and the tradeoff between these, to provide recommendations while Yuan et al. [2013] constructed a semantic network and used case-based reasoning. Ho et al. [2015a] tried using S-shaped utility functions to balance trade-offs and focused on making sure the acquired real estate would increase in value. Last, but not least, Li et al. [2017a] used a collaborative filtering approach based on machine learning on implicit data gathered from the real estate website Suumo.jp with the goal of increasing requests for viewings. While each approach optimized one part of consumer search process, none had a group aspect.

To summarize, the work done in the specialization project was:

- An extensive state of the art review of recommendation systems and group recommendation systems.

- An in-depth evaluation of the four different real estate recommendation systems presented in Daly et al. [2014a], Yuan et al. [2013], Ho et al. [2015a], and Li et al. [2017a].

- A discussion on future work for a prototype of a platform for group recommendation in Real Estate.

1.3 Goals and Research Questions

Goal *Develop and evaluate a platform for group recommendation of real estate.*

The primary goal of this master thesis is to develop, test and evaluate a group recommendation system for real estate. The system is called YAPS.life and can be found at `www.yaps.life`. The name is derived from *Young Aspiring Professionals*, the main target group for the application. Its purpose is to help individuals find the perfect roommates and then give them the tools to find the perfect place together. To achieve this, the system needs a powerful and robust matching mechanism, a beautiful and elegant user interface, and a way to recommend real estate that factors in the preferences of the entire group.

Based on this goal the following research questions were constructed:

Research Question 1 *What is the best way to match users looking for shared accommodation into groups?*

Many group recommendation systems work with arbitrary groups of people with different interests. The YAPS.life system can work with these as well, but its intended use is to help individuals find compatible roommates. To find these roommates there needs to be a way to find similar users and evaluate the quality of the created group.

Research Question 2 *How to present a group recommendation system for real estate?*

No matter how successfully the system matches users, or how relevant the real estate recommendations are, no one will use the system if it has a poor user interface. Through a usability evaluation the proposed user interface of YAPS.life will be investigated and through feedback the system can evolve into something users will want to use.

Research Question 3 *How do we provide accurate group recommendation of real estate?*

When a group is matched, and a good user interface is ready to present recommendations to the group, the recommendations themselves have to be of good quality. Listings from external data sources and from the YAPS.life system itself, will be matched with the groups using vector space techniques. The results will be evaluated through a qualitative survey.

1.4 Contributions

Even though several recommendation systems for real estate exist, and some research has examined group recommendation, no one has yet combined these two problems. The primary outcome of this research is a new computer-based platform that combines the field of group recommendation with new advances in the recommendation of real estate. The platform can be a strong foundation for future research into the area, to build on with new combinations of algorithms. In addition to presenting the design and code, an evaluation of the prototype is performed.

The contributions can be enumerated as follows:

1. A prototype of a platform for group recommendation of real estate, including the design of the algorithm and how they were implemented.

2. A quantitative analysis of different clustering algorithms for matching users into groups.

3. A qualitative usability- and recommendation quality evaluation of the prototype.

1.5 Thesis Structure

This thesis consists of 8 chapters and an appendix. The first, which is now concluded, gives an introduction to the background and motivation for solving the problem as well as a summary of the previous work done. It ends with the research questions posed, and contributions made by the thesis.

Next comes chapter 2 which takes a deep dive into the background theory of the subject matter. Different recommendation methods are presented, as well as group recommendation, and the challenges that come with the domain. In addition, specific clustering algorithms used for creating groups are investigated. The chapter finishes with a more in-depth review of different Real Estate recommendation systems investigated in the literature.

Chapter 3 details the research method followed in the writing of this thesis, and Chapter 4 gives a thorough description of the prototype that has been produced during the research. Chapter 5 and 6 gives the reader a better understanding of how the system has been implemented both algorithmically and technically by detailing the design of the real estate group recommendation algorithm and the architecture of the application.

Finally, chapter 7 and 8 will present the results from the evaluations and list the contributions of this thesis. In the end, plans for future work are detailed, and conclusions are made.

Chapter 2

Background Theory

To give a foundation for understanding the rest of the thesis, this chapter will give thorough introductions into the idea behind the prototype and recommendation systems in general. Furthermore, it will detail different approaches to group recommendation and how to use clustering techniques to put users into groups. Lastly, there will also be a review of current recommendation systems in real estate.

2.1 YAPS.life

2.1.1 The scenarios

YAPS.life is a website for group recommendation of real estate, specifically targeted towards young professionals, but essentially applicable to all real estate hunting use cases. Based on the location of a group of users' workplaces the service calculates the commute time to different areas of the city where the group is moving. Then it indicates which area they should start looking for apartments in. It also generates pre-filled search queries to different real estate providers to help them start their search. Now the group can rank new listings based on average commute time to workplace and price per person. The following are three scenarios where YAPS.life could be essential:

The perfect match

A young professional, e.g. a recent graduate, is planning to move to a new city where he has little or no network. Real estate prices are high for renting single rooms, let alone buying apartments, so flat-sharing is the only viable option. The user goes to yaps.life, creates a profile and fills out their preferences for their

new apartment as well as their habits when living in shared accommodation. The user will then be matched into a group of users with similar preferences regarding living habits and what they are looking for in a new apartment. Being matched on YAPS.life is a great way to kick-start their network in the new city and helps them find apartments with a location that works for everyone.

The friends

A group of friends are starting university together in the fall. They all create a profile and fill out their preferences on yaps.life. Then one of the friends creates a custom match and invites the others to it. The friend group can now use the service to rank different apartments they are looking at based on individual budget requirements and commute distance.

The landlord

As a landlord administering multiple properties, or just a person looking for someone to take over their lease, you do not necessarily want to expose your listing to as many people as possible. Instead, you want to show it to the people who would actually consider living there. Broadcasting your listing on public forums like finn.no or hybel.no in Norway, or Zillow.com or craigslist.com in the US often leads to lots of phone calls and viewings who do not end up in a signed lease agreement. By uploading your listing on yaps.life instead, your listing will be matched with groups of people actively looking for apartments in your area, and a direct chat link between the group and you as the landlord is created.

2.1.2 History of the service

The service was conceptualized during the spring of 2017 as a way to match students with summer internships in Oslo into shared apartments for June, July, and August. During that summer, five different flats were matched with students and the idea for the service started to grow into something bigger. The target was now young professionals, people who had just finished college or university, and looking to move to new unknown cities. The yaps (young aspiring professionals) would be matched into groups of like-minded people and be given the tools they needed to find an apartment together. After moving into an apartment found through their service, the group could start subscribing to services like housecleaning, laundry, food delivery, ride sharing and more, all at a discounted rate thanks to their shared living arrangement and YAPS.life as the broker. The technology behind the real estate group recommendation lies at the heart of this thesis.

2.2 Recommender Systems

Recommendation systems are designed to sift through large amounts of information and help users make decisions. Since their appearance in the mid-1990s they have become ubiquitous across the web.[Hill et al., 1995; Resnick et al., 1994; Shardanand and Maes, 1995]

The most popular areas for these systems are recommending consumer goods on services like Amazon [Linden et al., 2003], movies on Netflix and music on Spotify. However, other applications have also received attention. Everything from helping users find new restaurants [McCarthy, 2002] or recipes to try, to selecting the ultimate destinations for tourists [Ardissono et al., 2003], to finding new friends on social media. The recommendation of real estate is also a topic in the literature, but compared to the other domains the research is quite scarce. In general, we most often call on recommendation systems if we have an abundance of items to choose from or if we need to do complex computations to decide if we are making the right choice.

Also we also often divide the recommendation problem further into two subproblems, namely the *prediction problem* and the *n-rank problem*. Prediction is often related to media services like Netflix where we want to predict how a user will like several items side by side, while n-rank is all about giving an absolute ordering of options from best to worst. The latter is quite typical in the real estate domain as we will see in section 2.6.[Bø, 2017]

2.2.1 Content-based filtering

A content-based filtering recommender system uses the user's preferences to recommend items based on the user profile correlation with the item[Lops et al., 2011]. Both explicit and implicit feedback can be used to update the user's profile to give better recommendations in the future. Explicit feedback would usually be collected as a user's rating for an item on a set scale, while implicit feedback is inferred from events like how long the user views the item or the number of times the users visit the same item. Implicit feedback can improve the user experience seamlessly, but it can be hard to implement and also prone to errors because all updates are inferred, and no "negative" feedback can be collected.

Amazon presented its content-based recommendation approach back in 2001 with Linden et al. [2003] and has since been used many places since. An item is usually represented as a vector of n different values $X_n = (x_1, x_2, ..., x_n)$ where each index is either binary, a value on a set scale, or something else that represents an attribute for an item.

2.2.2 Collaborative filtering

Collaborative filtering, or CF, is a popular approach to recommendation systems where the following criteria are met:

1. A large user base with diverse tastes.
2. A system that makes it easy for users to indicate what they like.
3. A way to represent users so the system can figure out which users are similar.

As opposed to content-based filtering where the recommendations are based on user-item similarity, collaborative filtering looks at how similar users are and recommend similar items to similar users. Collaborative filtering is usually divided into two approaches, memory- and model-based. For memory-based solutions, recommendations are based on the entire data set of users, items and ratings. Whereas in a model-based approach a statistical model of the ratings is created to predict future ratings of other users[Sarwar et al., 2001]. This model can be created through machine learning algorithms such as Bayesian classification and clustering algorithms.

Companies such as Netflix and Spotify have had huge success with their collaborative filtering recommendation systems based on the matrix factorization approach. [Bennett et al., 2007; Johnson, 2014]

2.2.3 Knowledge-based systems

These systems rely on domain knowledge specific to the items they recommend. Knowledge-based systems usually come in one of two formats, case-based [Trewin, 2000] and constraint-based. The former uses a similarity function that estimates how a user's problem relates to solutions already archived in the system. The latter matches problems to predefined knowledge bases with explicit rules on how to relate problems to solutions.

Knowledge-based systems are more useful in scenarios where items are not purchased very often, such as a car, a house or a holiday package. For these items, there are rarely enough ratings on the individual items to give good recommendations based on content or collaborative filtering. Because purchases of some of these items happen so rarely, there is a good chance the user's preferences will have changed since their last purchase thus making recommendations on past purchases obsolete.

The cold start problem, described more closely in section 2.3.1, is a notorious challenge for most content-based and collaborative filtering recommender systems. Knowledge-based systems, however, sidestep this problem by looking for direct correlations between the user's preferences and the item's attributes without relying on previous ratings. Even though renting a property is a more frequent occurrence than buying one the feedback is still too scarce for content-based or collaborative filtering. That is why the YAPS.life system uses a constraint-based, knowledge approach as described in chapter 5, Design of Recommendation algorithm.

2.2.4 Hybrid systems

Sometimes a single approach is not enough to provide relevant recommendations. Combining different recommendation systems or algorithms using either weighted ensemble methods or voting, often provides higher accuracy recommendations than any algorithm can do on its own. This is because the weaknesses of each can be potentially eliminated by the others.[Adomavicius and Tuzhilin, 2005]

Burke [2007] breaks hybrid systems down further into the seven subtypes seen below.

Weighted Use weights on the output of the different recommendation components to combine their results.

Mixed A mixed hybrid system presents the different rankings from different recommendation systems side by side.

Switching Similar to the mixed system, a switching system will get individual recommendations from the different sub-systems, but in the end, only present the item with the highest confidence.

Cascading The cascade method runs one recommendation algorithm. The second algorithm uses the output of the first algorithm as its input. The produced output of the second algorithm is recommended to the user.

Meta-level Meta-level methods use a training phase to try to learn a model to give recommendations.

Feature combination This method sets up a pipeline where the first recommendation systems pass its output as input to the next one thus biasing it.

Feature augmentation The final method sees the first system produce an augmented set of features the next model can then use to make predictions.

2.2.5 The role of Context

Context is a vital part many recommendation systems. It is defined as "Any information that can be used to characterize the situation of an entity, where the entity is a person, place, or object that is considered relevant to the interaction between a user and its application, including the user and the application themselves" [Bazire and Brézillon, 2005]. Two popular contexts that are easy to apply to most systems is time and location. In Smaaberg [2014] the system developed is a context-based group recommender for concerts. Here it makes little sense to recommend concerts in the past or distant future, or in a location too far away to reach, to the user. Smaaberg uses this context to filter the potential recommendations to users while using all the collected data to train his recommender system. This filtering limits the dataset which makes it easier to do real-time computations.

Another way context is often applied is how preferences change depending on where the user is and what tools he has available. Movie recommendation, for instance, could change quite a lot depending on whether the user is nestled in the comfort of their own couch in their house instead of in a cramped airplane seat on a transatlantic flight.

Context plays a key role in the YAPS.life system as well. By filtering users on which city they are moving to, and in which time frame, the system avoids creating groups that can not move in together because of practical constraints, even though they are a good match. It also uses the location of the group members' workplaces to evaluate a property based on the average commuter time for the group.

2.2.6 Preference gathering with Feedback

A crucial component of any recommendation system is how it gathers initial preferences and updates them through feedback. There is no one correct way to do this as every recommender system application needs different information to provide useful recommendations.

Explicit feedback

This type of feedback happens when a user takes an explicit action against an item, e.g., leaving a rating or hitting a like/dislike button. This information can

be very compelling to the system because it reveals exactly how well the system in predicted whether the user liked the item or not. A further elaboration on evaluation follows in subsection 2.2.7. However, explicit feedback is far from perfect. Amatriain et al. [2009] found that ratings that are not extreme, i.e., 1 or 5 on a rating from 1 to 5, are incredibly inconsistent and Cosley et al. [2003] discovered that users put much less effort into rating a movie than a significant investment like a holiday. This fact means it is very tough to generalize models based on explicit ratings. Then, of course, there is the problem of getting users to leave explicit feedback at all.

Table 2.1: Examples of explicit and implicit feedback

Explicit feedback	Implicit feedback
Ratings	Clicks
Reviews	Time tracking eg. how much time a user spends on viewing an item
Like or dislike	Repeated viewings
Surveys	GPS coordinates when an action is made
	In a lab you can also track eye and facial movements as well as sound.

Implicit feedback

That is where implicit feedback comes in. Collecting implicit feedback is primarily analyzing usage data. In their 2008, paper Hu et al. [2008] discuss how implicit feedback has four different characteristics.

First, there can be no negative feedback. This is because it is hard to judge an item if the user has not seen or interacted with it. One could say that only viewing an item for a short amount of time is the same as leaving a bad rating, but the system cannot be certain. This is the main asymmetrical aspect between explicit and implicit feedback.

The next characteristic is that implicit feedback by definition contains a vast sum of noise. That means one can at best guess at whether an action implies that the user likes or dislikes an item. Take online shopping for instance. Viewing

an item does not necessarily indicate anything, maybe the user made an error, and view time can be affected by many things like stepping away from the computer or if the user is quickly browsing through many clothing items. One metric which says more, however, is visiting the same item multiple times because the user is less likely to click on the same item more than one time erroneously.

The third shows how numerical values between explicit and implicit feedback can be entirely different. If a user rates a TV series, it says more about how a user relates to it versus doing a view count which can be impacted by the length of the episode or the general number of episodes in the series.

The final characteristic is the evaluation of implicit feedback. There is a challenge with whether there are diminishing returns on feedback when a user buys an item multiple times, likewise for viewing or rating it several times.

Another essential aspect to keep in mind with implicit feedback is that as Kelly and Teevan [2003] point out; some feedback is only useful when spliced together with other information and useless on its own. Even though implicit feedback is noisy and it is impossible to determine something explicitly with it, it can still be extremely beneficial when the system has enough of it.

2.2.7 Evaluating recommendation systems

To evaluate if a recommendation system produces useful results, there are several different options available depending on what the system produces. If it is a typical content recommendation system like Netflix, where the system uses users' ratings to provide recommendations, the Root Mean Squared Error (RMSE) seen in equation 2.1 [Bennett et al., 2007] is a great tool. It estimates the rating of a given item for a given user and subtracts it with the rating the user actually gave the item.

$$\sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y_j})^2} \tag{2.1}$$

However, if a system produces a ranked list one would instead use the Normalized Discounted Cumulative Gain (nDCG) from equation 2.3. Here r_{uci} is the actual rating of a user u for the content i at position c in the ranked list. [De Pessemier et al., 2014]

$$DCG_n^u = r_{uc_1} + \sum_{i=2}^{n} \frac{r_{uc_i}}{log_2(i)} \tag{2.2}$$

$$nDCG_n^u = \frac{DCG_n^u}{maxDCG_n^u} \tag{2.3}$$

These are methods that are often used for academic purposes on well-established datasets like MovieLens [Mov, 2017] in what is called an "Offline" manner, as the datasets are not updated during computation. They are great for quantitatively testing the performance of different algorithms, but as to how useful a recommendation system feels, qualitative studies on users are often used. Multiple examples of this will be presented in the review of recommendation systems in real estate in section 2.6.

2.3 Challenges in Recommendation Systems

The many different approaches to developing recommendation systems often come with their own known challenges. In this section, the most widespread of them will be illuminated and related to recommendation in real estate.

2.3.1 The cold start problem

The cold start problem is a well known and documented problem for many recommendation algorithms, but especially collaborative filtering. When any new user joins the system, there is no information about his habits, usage or ratings that can be used to find similar users. This makes providing relevant recommendations very hard. The same problem arises when a new item is entered into the system. Because no users have rated it yet, the system will have no reason to present it to any users.

In order to combat this problem, many collaborative filtering systems make new users rate a random subset of items upon entering the system. These ratings let the system establish a slim profile for the user on which it can base further recommendations. For the item, it is possible to use a content-based approach and find similar items to it. Users who have a high rating for similar items can be predicted to have a high rating for the new item as well.

2.3.2 Sparse data

The user-item matrix of most recommendation systems is usually extremely sparse as users usually only interact with a small subset of the items in the system. Imagine all the products on Amazon, or all the movies on Netflix for instance, a user cannot possibly buy or view all of them. Because of this sparseness, providing accurate recommendations can be challenging. On the flip side,

the fact that the matrix is so sparse allows us to do mathematical operations that would otherwise be computationally intractable such as matrix factorization.

2.3.3 Serendipity and Novelty

For a long time, the main objective of recommendation systems research has been to provide accurate recommendations based on metrics like RMSE, not based on what is proven to be actually useful for the user. McNee et al. [2006] says that "not only has this narrow focus been misguided, but it has even been detrimental to the field." As an example, we can use a book recommendation service. If a user enters that he has read the first Harry Potter book and all he gets recommended are the remaining six books this is a good recommendation statistically because they are very similar. However, there is a good chance the user was already aware of the remaining books and would have more benefit of having another similar book series recommended.

The degree of serendipity wanted by users is often related to the cost of the choice of choosing a recommendation. If a user watches a movie the recommender system recommended serendipitously, and ends up not liking it, it costs him only a couple of hours. The same in a real estate recommendation system could cost him a lot more time, or even worse, a lot more money.

2.3.4 Bias from popularity

In many recommendation domains, there is a popularity bias problem also known as "The rich get richer." This is what happens when, in a music recommendation system, for instance, a song gets very popular and receives exponentially higher listen counts than other similar songs. Because of its popularity, it gets recommended to more users, which in turn only boosts its popularity even more. This can make it difficult for new items to be recommended and can lead to the system only recommending a tiny subset of its items. To avoid this problem, many recommendations scale down their popular items with normalization. Collaborative filtering is, again, especially prone to this.

In real estate this is not a very large problem as most listings are not on the market for very long. A popular listing will only dominate newer ones until it is sold or rented, and then subsequently removed from the system.

2.4 Group Recommendation

A group recommendation system, GRS, applies traditional recommendation algorithms to groups of people. The two main approaches to this are:

1. Calculating recommendations to the individuals, then merge them into a set of recommendations or rank them for the group.
2. Construct a group persona based on the individual's preferences and recommend items directly to that group persona.

For both approaches, a myriad of different aggregation techniques can be used, the full list which can be viewed in Masthoff [2004]. There is no universal best strategy, so one will have to choose based on the problem at hand. The most popular are average aggregation, least misery aggregation and average without misery aggregation.

So far group recommendation has been mostly used in domains like movie recommendation for a group of friends[O'connor et al., 2001], music recommendations for bars and gyms [McCarthy and Anagnost, 1998] and travel itineraries suited for whole families[Jameson, 2004].

2.4.1 Group formation

An essential part of any group recommendation system is how the groups are formed. Borrowing from the authors specialization project we have that "All groups share the properties of being either *persistent* or *ephemeral*, *public* or *private*, and *actvily* or *passivly* joined.[O'connor et al., 2001]

A Priori Groups

An a priori group is an established group consisting of members who joined the group actively and often plan to stay in the group for a long time, i.e., the group is persistent. The group can be public, but is most often private like a group of friends. An example of an a priori group could be a group of friends who join together once a week to watch movies, play board games or discuss books. Since the group largely stays the same a system could use their preferences and feedback to recommend new activities continually. The friends from scenario 2 in section 2.1.1 are also an example of an a priori group.

Occasional Groups

An occasional group is a group that forms together once or several times to accomplish a specific goal. The main difference from the a priori groups is that

where previous relations was the basis of the former group, the goal is the basis for the latter. The most common occurrence of this kind of group is found in tourism, food and media recommendation. The aforementioned Polylens system from O'connor et al. [2001] is a good example of a system that works with occasional groups. Another is the famous pocket restaurant finder from McCarthy [2002].

Groups Determined by a Shared Environment

These kinds of groups are *ephemeral* by nature because people can at anytime leave and join the group space. Therefore recommendations must be recalculated quite often thus limiting how computationally intensive the recommendation can be. Recommendation applications for these kinds of groups usually regard public information displays and music or other media in shared spaces [De Carolis and Pizzutilo, 2009; McCarthy and Anagnost, 1998].

Automatically Detected Groups

When the group is not established beforehand by prior relations, activity or shared environment, it needs to be automatically detected. Cantador et al. [2008] describes how to detect Communities of Interest(CoI), users who share interests, by using K-Means clustering [MacQueen et al., 1967a] to cluster their profiles defined by a semantic ontology.

For a user to be clustered, it should be modeled as a vector where each index represents how much the user relates to a concept. Multiple schemes can be used, everything from simple present(1)/absent(0) to weighted models and even negative numbers to symbolize dislike. When we have the vector we can use similarity measures, described below in section 2.5.1, to calculate the distance from one user vector to another. Users with a smaller distance will be more similar and can thus be grouped together if the goal is a homogeneous group." [Bø, 2017]

2.4.2 Group model strategies

After a group is created the next problem to solve is how to represent this group in the system and how to give it recommendations. There are two main approaches to this in the literature. One is giving individual recommendations and then averaging results. The other is to create a group persona based on the aggregated preferences of the individual users. Both these approaches will be explored below, but first, a baseline model will be presented.

2.4.3 Candidate set

If the purpose of the group recommender is to supply a set of candidate solutions, one simple way of accomplishing it is simply finding a candidate solution set for each user and then presenting the union of these sets to the group. This presupposes the group will have an important role in the final decision as it does not explicitly say which option is best for the group [Masthoff, 2004]. While simplistic, this algorithm is often a good baseline to compare other group recommendation algorithms to as it is the easiest to implement when there already exists a recommendation system for individuals.

2.4.4 Group persona

The second popular approach is to create a new model out of the original user models, also called a superuser. This model encompasses all the preferences of the different users and can be run through a recommendation algorithm meant for an individual to calculate ratings. See Yu et al. [2006] for a detailed strategy on how to merge users through linear combinations, but know that it can simply be stated as:

1. Create a model G for the preferences of a group
2. Go through each item i and use G to predict the rating R_i for the group
3. Recommend the candidate set of solutions to the group with the highest R_i's

Notable examples of this approach are Ardissono et al. [2003] and Jameson [2004] recommenders of sequenced travel activities and Yu et al. [2006] own TV program recommender.

2.4.5 Aggregation of individual preferences

The second of the two more common ways of creating group recommendations is aggregating individual preferences. How to aggregate them has received much interest in the literature, yet most refer back to Masthoff [2004]. In her paper, she presents ten different ways of aggregating preferences, three of which are listed below. Each method is coupled with an example of three users u_1, u_2, u_3, 5 items A, B, C, D, E, and a rating scale of 1-5. Notable papers who used these methods are O'connor et al. [2001] who employed the least misery strategy in their Polylens system and Baltrunas et al. [2010] who tested several of the strategies on the Movielens dataset.

2.4.6 Aggregation strategies

Average aggregation

With this strategy, table 2.2, one simply averages users individual ratings/predictions for an item into a group score. This often works very well, but can lead to problems when users have very different ratings. If two uses rate an item 5 and another two rates it 1 the group score becomes 2.5, a decent score in most respects. If this were for a movie, then one would end up with two people being very excited and two people utterly miserable. That is where the least misery strategy comes in.

Table 2.2: Average aggregation

	A	B	C	D	E
u1	4	2	5	5	1
u2	1	2	5	3	3
u3	4	1	4	4	3
Group rating	3	1.6	4.6	4	2.3

Least misery

This strategy, table 2.3, looks at all the individual ratings and sets the lowest one as the group's rating. The argument is that the group is only as happy is the groups least happy member [Masthoff, 2004]. An excellent strategy when dealing with allergies in a restaurant recommendation system where recommending a place where one of the group members are allergic to the food could be catastrophic. However, this means that in cases where everyone else in the group are really keen to go, one person's preference overrules the rest.

Table 2.3: Least misery

	A	B	C	D	E
u1	4	2	5	5	1
u2	1	2	5	3	3
u3	4	1	4	4	3
Group rating	1	1	4	3	1

Average without misery

A mix of the two above, this strategy, table 2.4, uses a misery threshold to remove items from consideration when a group member rates an item below the threshold.

For the remaining items, the average strategy is used. Assuming an appropriate threshold is chosen, this strategy reflects the wishes of the group quite well in addition to removing items that would make some group members miserable.

Table 2.4: Average without misery

	A	B	C	D	E
u1	4	2	5	5	1
u2	1	2	5	3	3
u3	4	1	4	4	3
Group rating	-	-	4.6	4	-

2.5 Clustering

Clustering is a term for mapping an n-dimensional feature space into a set of smaller groupings, also known as clusters. The goal is to find the maximum amount of similarity in a cluster and to minimize the similarity between the clustered groups [Kim and Ahn, 2008]. Cluster analysis is defined as "a statistical classification technique for discovering whether the individuals of a population fall into different groups by making quantitative comparisons of multiple characteristics." There are two main families of clustering methods as defined by Jain [2010].

The first approach is called *hierarchical clustering* and it too has two sub-approaches, bottom-up and top-down. Bottom-up is an agglomerative method where each data point starts with being its own cluster before merging them with other similar clusters to create a hierarchy. The top-down approach does the opposite by starting out with all data points in one big cluster before dividing it into smaller clusters recursively. The second family of clustering is called *Partitional clustering*, and it works by partitioning the data and finding the clusters in parallel.

In chapter 5, Design of Recommendation Algorithm, different clustering algorithms are investigated for matching user profiles into groups in accordance with RQ1. The dataset is then the users of the application and the data points are the user's personal information with a focus on their habits and real estate goals. More background theory on the clustering techniques investigated later can be found below.

2.5.1 Similarity metrics

In order to find similar data points the clustering techniques needs techniques to measure distances between different points. These techniques are called Similarity Metrics and there are many to choose from. Some of the most popular ones like Manhattan, Euclidean, and Cosine distances, are described here. Similarity between two vectors is higher when the score is closer to 0, and worse the further away from 0 it is.

Manhattan distance

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p} \tag{2.4}$$

The Manhattan distance (sometimes also called Taxicab distance) metric is related to the Euclidean distance, but instead of calculating the shortest diagonal path ("beeline") between two points, it calculates the distance based on gridlines. The Manhattan distance was named after the block-like layout of the streets in Manhattan.

Euclidean distance

The Euclidean distance is a distance measure between two points or vectors in a two- or multidimensional (Euclidean) space based on Pythagoras' theorem. The distance is calculated by taking the square root of the sum of the squared pair-wise distances of every dimension.

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.5}$$

Euclidean distance squared

The same as regular Euclidean distance, but without the square root of the sum.

$$\sum_{i=1}^{n}(x_i - y_i)^2 \tag{2.6}$$

Cosine Distance

To get the Cosine Distance, the Cosine similarity has to be calculated first. Cosine similarity measures the orientation of two n-dimensional vectors irrespective of their magnitude. It is calculated by the dot product of two numeric vectors, and

it is normalized by the product of the vector lengths. This outputs a similarity score between -1 and 1, where 1 constitutes prefect similarity. To shift this into the positive space of the other similarity metrics we subtract the Cosine similarity score from 1 as seen in equation 2.8. Now the range is between 0 and 2.

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{y}||} \tag{2.7}$$

$$1 - cos(\boldsymbol{x}, \boldsymbol{y}) \tag{2.8}$$

Using weights

For distance metrics, it is sometimes appropriate to use weights to indicate that distances between some indexes should have more effect than others. This concept will be explored further in 5.6.

2.5.2 Curse of dimensionality

As the number of dimensions increases however, all these similarity metrics run into problems. This is sometimes called "The curse of dimensionality" and is often used as an umbrella phrase to describe the various phenomena that arise when doing cluster analysis or regression on data in high-dimensional spaces. A high dimensional space can span from a dozen dimensions, to hundreds or thousands. These phenomena do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience. The expression originates back to Richard E. Bellman in his 1957 paper on dynamic programming which has since been republished in Bellman [2013].

The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data becomes sparse. This sparsity makes it hard to find model coefficients that best explain the data, and you need exponentially more data as you increase dimensions to get the same level of model accuracy. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient. [Donoho et al., 2000]

To combat this, several algorithms can be used to reduce the number of dimensions before clustering.

2.5.3 K-Means

K-Means is one of the most well known and widely used clustering algorithms. Since its inception in the late 50ies and official coining by MacQueen et al. [1967b] in 1967, k-means has been a staple in fields like signal processing, market segmentation and, more recently, feature learning for machine learning applications. The reason for its popularity is because of the low threshold for implementation and relative ease of use. The algorithm works by plotting the n-dimensional dataset into vector space and then inserting k new data points, called centroids, at random locations. Then it systematically measures the distance between the centroids and all other points in the dataset and before moving the centroids in the direction of the center of the closest data points. When the centroids stop moving the algorithm is terminated, and the closest points to each centroid are outputted as a cluster. Formally we define a dataset of n elements, each element described by m attributes or features. We use $X_i = [x_1, x_2, ..., x_m]$ to define the m attributes for user i. Last, but not least, the clusters themselves are represented with K.

However, due to this simplicity it has its share of drawbacks. It deals poorly with overlapping clusters, and outliers can pull centroids out of the real clusters. In addition, there is a chance for the algorithm to get trapped in a local minimum during early iterations because of unfortunate initialization. While the algorithm works well on larger datasets, it can have trouble with smaller ones. $k < n$ is also a requirement. The choice of k is a dilemma in itself. Choose it to large, and the system ends up with many empty clusters, choose it too small, and the actual granularity of the dataset will not be represented.

In order to measure distances between data points k-means uses the Euclidean distance, described in section 2.5.1. The idea is to minimize the sum of squared roots for the distances between the clusters in the data set. K-means is often confused with k-Nearest Neighbour because of the shared k, but as we will see in the next section, the algorithms are quite different.

2.5.4 k Nearest Neighbours

Park et al. [2016] defines k Nearest Neighbor (kNN) as "an operation that selects k objects that are most similar to a given object among all the candidate objects in a dataset." In pattern recognition, the algorithm is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether kNN is used for classification or regression:

- In kNN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

- In kNN regression, the output is the property value of the object. This value is the average of the values of its k nearest neighbors.

This means all work is deferred until classification, thus making kNN a lazy learner. An active learner would have preprocessed the data so that less work would be needed at classification. It can often be useful to decrease the weight of the neighbors further out from the center, like giving a $weight = 1/d$ where d is the distance.

In this thesis, the algorithm is adapted somewhat differently. The goal is to construct clusters of a predefined size, in this case four. To do this, the algorithm starts with the first user, represented as a vector, in the dataset. Then it finds the three (k=3) most similar users to him and extracts all four into a cluster. Now the original data set has been reduced by four users and the algorithm proceeds to evaluate the next user in line. This continues until there are no users left.

2.6 Real Estate Recommendation Systems

From the author's specialization project [Bø, 2017] we have that the definition of real estate is a property consisting of the land, natural resources and the buildings on it, but for most of us, it is simply what we call home. Choosing real estate, either for buying or renting, are some of the most important and expensive decisions we make, and few have to make more than a handful during their lifetime. Real estate is one of the oldest industries we have and is traditionally old fashioned, but over the last decade, we have seen a wave of disruption. First, web technology allowed real estate properties to be represented and presented by structured data and new technologies like drones and virtual reality(VR) allows us to view listings from the comfort of our own home thus eliminating part of the time requirement going on multiple viewings of listings require.

Despite all this data about housing [Bond et al., 2000], research on real estate recommendation systems has been quite scarce [Konstan and Riedl, 2012; Li et al., 2017b], though it has picked up a bit in recent years. The following is an introduction to how traditional real estate web search is today and summaries of new recommendation models observed in chapter three in Bø [2017].

2.6.1 Traditional web-based housing search

Real estate search on the web has existed for years, yet consumers have enjoyed few of the recommendation system advances other industries have seen despite the competition and large amounts of data available [Grant and Cherif, 2016; Zheng et al., 2006]. There are far too many real estate search providers to list them all exhaustively, but a small sample can be found in table 2.5. These services

Table 2.5: Table showing different housing services

Engines	Location reach	URL
Finn.no	Norway	www.finn.no
Rightmove	United Kingdom	www.rightmove.co.uk
Zillow	USA	www.zillow.com
Trulia	USA	www.trulia.com

all share a search and filter approach where the user:

1. Enters an address or zip code to begin a search of an area.
2. Enters filters on price, housing type, number of bed and bathrooms, and size range, etc.
3. Receives some simple recommendations for similar houses to the ones the user is browsing.
4. Iterate until the user finds one or more listings he would like to view in person, or until he quits.
5. Some American search engines like Zillow and Trula let the user apply filters for government buildings like hospitals, public transport stops and schools as an overlay over a map of the listings.

Surprisingly, no large real estate broker incorporates important locations to the user as part of the search process as suggested in Yuan et al. [2013], Daly et al. [2014b] and Grant and Cherif [2016]. Neither does any system, either in literature or the real world, perform recommendation of housing for groups of people.

2.6.2 Multi-Criteria Journey Aware Housing Recommender System

The paper from Daly et al. [2014b] focuses on extending conventional metadata for housing, like price, number of bedrooms, etc. with the journey time to multiple locations. To make the journey time rating of a house even more accurate,

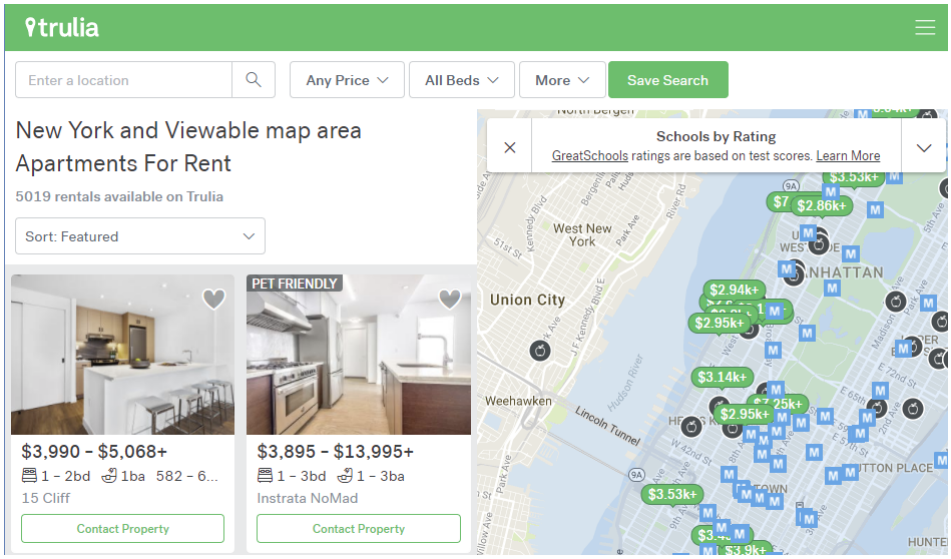


Figure 2.1: A search of listings in New York with metro stations and schools added as an overlay from www.trulia.com

they even check for traffic at multiple times a day. The end result was a web system with a map interface.

To evaluate their system, Daly conducted a user study of 20 participants where the houses up for review were a subset of 500 homes from the current housing market in Dublin. All homes were two bedroom apartments. After selecting three important locations, the participants followed a search process where they eliminated more and more homes in each step. Upon completion of the process, they usually had around 50 listings still fulfilling their criteria, i.e., a 10-fold reduction from the original set.

It is worth pointing out that 90% of users chose a house from the recommended subset, however maybe even more interestingly 45% already picked the house they ended up with in step one, before the consequent eliminations, i.e. they did not change their choice after more information was gathered.

2.6.3 Toward a User-oriented Recommendation System for Real Estate Websites

Yuan et al. [2013]'s "Toward a user-oriented recommendation system for real estate websites" is one of the most cited papers in the real estate recommendation literature. After reviewing the available real estate search engines in Korea to see what was working and what was lacking, they used questionnaires and direct observation to extrapolate user behavior in real estate search. Based on Choo et al. [2000] and their own process they generated the model seen in figure 2.2 for housing search online as well as these three personas:

- A working couple with two children, one of the children is school-age.
- A working couple without children, enjoying urban "high-life": fashion, entertainment, and convenience.
- Single, needs a small but well-constructed home, prefers a community with amenities.

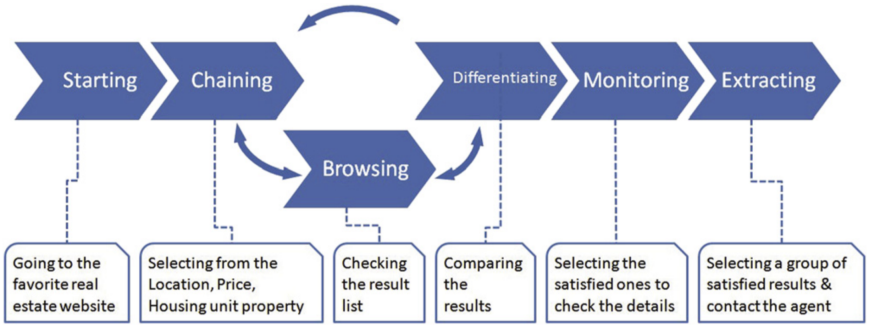


Figure 2.2: Yuan et al. 's housing search model

When they had the model in place they used the Corcho et al. [2005] method to define the ontology's needed to set up the semantic case base focusing on location, housing type, and price. Finally, they implemented a web-based system with a map centered interface on top of the semantic model.

As mentioned above Yuan conducted a user study to extrapolate user behavior on traditional online housing search so to test his new system he invited the same 30 people back. The user study focuses only on the user interface quality compared to mainstream online housing search and not on whether the system

actually found better apartments. Still, more than 70% of the participants found the Yuan's system very satisfying to use, and many pointed out that this approach was superior to other services they had tried before.

2.6.4 House Selection via the Internet by Considering Homebuyers' Risk Attitudes with S-shaped Utility Functions

Ho et al. [2015b] turns the housing recommendation problem into an n-rank problem based on risk attitudes and fuzzy goal setting. As discussed in section 2.6.1, the writers point out the weaknesses of modern real estate search engines in being very constraint-based. They define five important concerns for online-agents and home buyers:

1. There is much fuzzy information on real estate listings like "great quiet neighborhood with excellent schools" or "close to world-class shopping, dining and entertainment", yet it is not structured well enough to query.

2. Real estate websites should allow their users to prioritize constraints.

3. In theory the more houses one evaluates, the larger the chance of finding a better match.

4. Users should be able to view the predicted future value of the house, so they know they are making a good investment.

Compared to the other two articles reviewed so far Ho et al. uses a significantly larger sample size in the evaluation of his research at the cost of no in-depth interviews. They perform a Laboratory-quasi test with 250 middle-aged Taiwanese people, all with home-buying experience. The group is split into one test group to use the decision support system and one control group. Both users groups input their desired zip code and price range, but only the test group use the system to rank the results. Using a single factored ANOVA test(Analysis of variance) show a considerable gain in satisfaction for the users who employed the system compared those who did not.

When they looked at time spent directly on other real estate sites like Yahoo Real Estate[1] against their own, they found time spent reduced from 20-30 minutes to 8-10 minutes.

[1] `http://realestate.yahoo.com`

2.6.5 Web-Scale Personalized Real-Time Recommender System on Suumo

Li et al. [2017b] created a machine learning based recommendation system based on real-time log outputs from the most popular online real estate site in Japan called Suumo. They argue that inputting all user preferences beforehand is too time consuming and prone to error and that a much greater value would be had if the site learned the preferences solely by observing the user's usage pattern.

The paper also focuses on how to build a scalable architecture that runs well with massive amounts of data, and that is highly decoupled so that scientists can focus on improving core algorithms and deploy in quick iterative cycles without having to worry about breaking deployments.

They describe two methods. First an early attempt at content-based and collaborative filtering and then later a gradient boosting method.

Li et al. chose their main evaluation metric to be the conversion rate(eq 2.9) of their users. That means the rate at which users send a request for more information on a listing compared to how many listings they view. After tuning some parameters of the model with offline training, they set up a full A/B test pitting their machine learning system against their earlier content-based model. In the beginning, the content-based approach actually gave better results, but after a couple of days, the machine learning system outpaced it by miles. In the end, the machine learning system outperformed it by roughly 250%.

$$CVR = \frac{info\ requests\ on\ recommendation}{recommendation\ clicks} \tag{2.9}$$

2.6.6 Summary

In this section, we have reviewed four different papers, each representing a different approach to recommendation in real estate. The first uses content-based filtering to recommend properties based solely on the user's location and rent preferences. Yuan uses case-based reasoning, a form of knowledge-based recommendation, to incorporate the semantics of real estate search into his model. Ho goes for a more mathematical approach using fuzzy reasoning, and lastly, Li uses a form of machine learning.

The approaches are varied, and it is hard to compare them on an even level since they all measure different aspects of their systems in their evaluations, and none use the traditional methods described in section 2.2.7. All studies report

great satisfaction from the test users.

Both Yuan et al. [2013] and Ho et al. [2015b] refer to earlier research like Zumpano et al. [2003] from 2003 on how having more information and choices available in the housing search process has not reduced the time it takes to find a home. With the real estate economy and industry evolving as fast as it has in the last decade show more updated research is needed.

Chapter 3

Research Methodology

This chapter details the research methodology and strategy used for this thesis. It covers an introduction to design science research and how it is applied to the project before going through the experimental tools, and the experimental plan and setup.

3.1 Design Science Research

For this project, a Design Science approach was deemed appropriate. Design Science "seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished."[Hevner and Chatterjee, 2010] In his 2004 work, Von Alan et al. [2004] details a framework with seven guidelines on how to do rigorous design science research. This framewokr can be found in table 3.1

Application to this thesis

Design as an Artifact

The objective of this project is to create an artifact of the type instantiation. The artifact will be a web-based recommender system for people looking to live together, and the thesis will describe both the creation of the product as well as the algorithms used. The artifact can be found at `www.yaps.life`.

Table 3.1: Design Science research guidelines as presented in Von Alan et al. [2004]

Guidelines	Description
Design as an Artifact	The result of the Design Science must be an artifact of the type: Construct, model, method or instantiation.
Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well executed evaluation methods.
Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Research Rigor	Design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Problem Relevance

Shared accommodations are more popular than ever because of the ever increasing cost of real estate in cities, and according to Roberts [2013] and Heath [2016] the trend will not end anytime soon. Despite the huge impact where and with who we live has on our lives, most real estate systems today do not take a programmatic approach to recommendation for users, instead relying mostly on filters. An easy to use recommendation system to help people find housemates they get along with could improve the living situation for many.

Design Evaluation

A quantitative analysis of different clustering techniques will be performed to determine which is better for matching users into groups for the purpose of apartment hunting. The evaluation metric is presented in 3.2.1. To evaluate the usability of the artifact, there will be conducted a System Usability Scale(SUS) test, outlined in subsection 3.2.2. After further development, another user-centric test will evaluate the quality of the real estate recommendations as described in 3.2.3.

Research Contributions

The different algorithms used in the recommendation process will be explained, both in design and technical detail. The source code will also be made available on GitHub[1], at `https://github.com/kristianeboe/yaps.life`, so that others can verify the results.

Research Rigor

In order to maintain reproducibility of the results of this research, the algorithms, technologies, and frameworks will be described in detail. The open source codebase on GitHub will also make any attempted reproductions easier. As for the evaluations they are approached with tried and tested scientific tools like the System Usability Scale, described in subsection 3.2.2.

Design as a Search Process

To illustrate how this Design Science artifact is the result of a search process we can look at the iteration of the development process. From the early stages of the prototype to the finished product the visual design changed dramatically resulting in a more intuitive end result. The recommendation and clustering algorithms implemented were also increasingly complex.

[1] `www.github.com`

Communication of Research

Chapter 5 and 6 describes the technical implementations of the prototype and a more holistic description is given in chapter 4. The thesis will be available to the public through the Norwegian University of Science and Technology (NTNU), and all source code will be available on GitHub under a GNU (General Public License), GPLv2) for further testing and extension . The SUS test and other questionnaires can be found in the Appendix section of this thesis.

3.2 Evaluation Tools

In order to evaluate different aspects of the prototype, different tools are needed. For the qualitative evaluation, a version of the Silhouettes coefficient, introduced by Rousseeuw [1987] was used as it gives a good indication of how well clustered the dataset has become. In order to answer RQ2, "How to present a group recommendation system for real estate?", it was decided to use the well known SUS test in conjunction with an Application Specific survey to evaluate whether the user interface development was headed in the right direction. As for the evaluation of the real estate recommendations, a qualitative survey was used to capture whether the results were actually relevant to users, not just giving a good numerical score like in RMSE from section 2.2.7.

3.2.1 Group Cohesion Measure

As a way to test how well aligned the groups created by the different clustering algorithms were, a score called the Group Cohesion Measure was developed. It is found by iterating over the users in a group and taking the Cosine distance between all other users in the group. After finding all distances, they are added together and the sum is divided by the size of the group. This produces the average distance between users in the group. This is a simplified version of Rousseeuw [1987]'s Silhouette algorithm which is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation).

The reason for this simplification can be illustrated with an example. If eight users were lying on top of each other in the dimensional space, they should still be split into different matches. The matches should evaluate as good because the users are similar, but with a Silhouette score the overall algorithm would be evaluated as bad because the clusters would appear to have low separation.

3.2.2 System Usability Scale

The System Usability Scale was created by Brooke et al. [1996] in 1996 and has been a staple on user testing ever since. The test consists of ten questions using a Likert scale[Likert, 1932], i.e. all question's answers range between 1-5, and it gives a global subjective view of the usability of the application. As of ISO 9241, the test can only be considered valid if it is used on actual end users of the system, and it is preferable if the test is taken in environments mimicking the context where the application will be used.

The SUS test is a "reliable, low-cost usability scale that can be used for global assessments of systems usability."[Brooke et al., 1996] It produces a score on the scale of 0-100 which in theory makes it easy to compare multiple systems on their usability. However, due to the wide range of systems the SUS test can be applied to, the questions are by necessity quite general and it is once again important to take the context of which the systems will be used in before making comparisons. For example, the SUS score for an industrial application is probably not very correlated with that of a text editor.

To calculate the score, the following steps must be taken:

1. For each of the odd-numbered questions, subtract 1 from the score.
2. For each of the even numbered questions, subtract their value from 5.
3. Take these new values which you have found, and add up the total score. Then multiply this by 2.5.

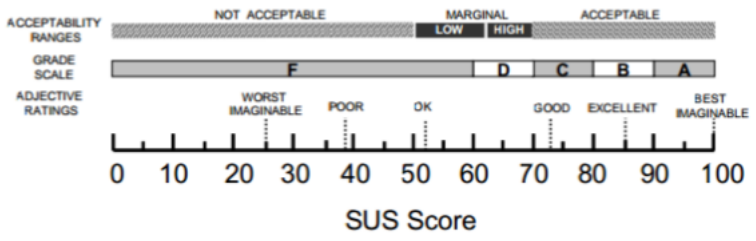


Figure 3.1: SUS adjective rating scale from Bangor et al. [2009]

Bangor et al. [2009] adds to the research with a proposition of adding an adjective scale to the SUS score to give a better indication of what the results actually

mean. The rating scale, which can be found in Figure 3.1, indicates that "products with scores above 70 are at least passable, better products have a score in the high 70s to upper 80s, and truly superior products score better than 90. Products with a score below 70 should be considered candidates for increased scrutiny and continued improvement".

3.2.3 Real Estate Recommendation Quality Survey

To evaluate the quality of the real estate recommendations, another survey was developed in Google Forms. It consists of two parts, one for Solo recommendation and one for Group recommendation. The test subject is tasked with creating a solo match and wait for real estate recommendations to be loaded. By manually inspecting the different listings the user will then decide on a rating for how well the system ranks the listings based on the user's own preferences. Then, the user is matched with a group of other real test subjects. New listings will be loaded and ranked by the system, and the users will now rate how the system ranks the listings in regards to the group's preferences. For both the solo and group part, the test subject is asked to rate how this way of searching for housing compares to traditional methods they have used before.

These questions will indicate whether the YAPS.life system provides accurate recommendations and whether this approach to housing search is worth pursuing further.

3.3 Evaluation Setup & Plan

When the matching algorithms have been implemented, a data set of user profiles will be created based on real users collected through the prototype. Then, with this data set, the different matching algorithms will be evaluated using the group cohesion measure from 3.2.1. After the second period of development, a user-centered usability test will be conducted using the System Usability Scale developed by Brooke et al. [1996]. Then, after another period of development, the quality of the recommendations will be assessed with a Quality Evaluation Survey.

3.3.1 Quantitative analysis of clustering methods

To find the answer to RQ1, "What is the best way to match users looking for shared accommodation into groups?", a series of tests on the performance of different clustering algorithms will be conducted. Two datasets will be used to test the algorithms on. The first is a collection of 38 real users that created their

profiles in the prototype. The other is 10 000 test users with personality and property vectors initialized randomly.

To get an idea of the theoretical minimum and maximum bounds that can be achieved in terms of Group Cohesion for a match from the dataset, the first step is to do a brute force search through the solution space. Setting group size to four, the amount of matches generated for these 38 users will be the same as the combinatorial of $\binom{38}{4}$ which is 73815.

Then, the different clustering algorithms will cluster the users into a list of matches. After sorting this list from best to worst, the matches are plotted on a graph with the match number on the x axis and the group cohesion of the match on the y axis. This chart will present the distribution of the group cohesion measures and it will give a good framework to compare the different algorithms side by side.

The similarity metric chosen is the Cosine distance because it bounds the cohesion between 0 and 2. The algorithms to be evaluated are the following:

- **Baseline**, simply divide users into groups of four without any clustering.
- **kNN with Euclidean distance**, go through the available users linearly, and for each user extract his top matches. Users are sorted with the Euclidean distance.
- **kNN with Cosine Distance**, same as above, but users are sorted with the Cosine distance.
- **k-Means**, divide the users into k clusters, where k is the log_2 of the number of users to be matched. Divide these clusters into four without further clustering.
- **Hybrid of k-Means and kNN with Euclidean distance**, after running the k-Means algorithm, clusters the users further with kNN with Euclidean distance.
- **Hybrid of k-Means and kNN with Cosine distance**, same as above, but with the Cosine distance.

After the evaluation of how the algorithms cluster the real users, the same test will be run again on the test users to see if the performance is the same and to see how the distributions change as more users are added.

3.3.2 Usability evaluation

For the usability evaluation of the prototype, 16 people will be asked to complete a series of steps using the prototype. There are three questionnaires involved. First, there is the Background Information (BI) questionnaire to gather demographic information, as well as familiarity with the apartment searching process and living in shared accommodations. Then the users are asked to perform a series of tasks on the platform, summarized below.

After completing these tasks, the test subjects will be asked to complete the System Usability Scale(SUS) survey and an Application Specific(AS) survey. The reason for this additional AS survey is that SUS is overly general in some areas, and even though it gives a good indication of general usability, the AS will give more direct feedback to influence further application development. These surveys were all developed using Google Forms[2] and can be found in the appendix, section C.

Here is a summary of the tasks users performed during the evaluation:

1. Answer the Background Information survey (BI), Appendix C.1.
2. Go to `yaps.life` and create a profile.
3. Complete their profile with preferences, see section 4.4 in chapter 4, Prototype.
4. Create a demo match with AI.
5. Try ranking a few apartments found from Finn.no.
6. Create a new custom match and add a friend.
7. Try ranking a few other apartments found from Finn.no.
8. Answer the SUS survey, Appendix C.1.
9. Answer the AS survey, Appendix C.1.

3.3.3 Quality of recommendation evaluation

Usability is not all that matters in a recommender system. Without actual quality in the recommendations, the system does not provide value to its users. If the system uses content-based or collaborative filtering, one popular method for evaluation is to use a mathematical approach like Root Mean Square Error (RMSE)

[2] `forms.google.com`

and n-Fold Cross-Validation as described in 2.2.7.

For YAPS.life however, a knowledge-based system, Gu and Aamodt [2006] indicate that the best way to evaluate the recommendations is through user testing. Four groups of students will be put together and asked to complete the following tasks:

1. Create a user and fill out your profile
2. Create a Solo match
3. Rate the ordering of the recommended listings.
4. Rate this way of searching for real estate compared to ways you have tried before.
5. Add flatmates to your match and do another apartment search.
6. Rate the ordering of recommended listings again.
7. Rate this way of searching for real estate as a group compared to ways you have tried before.

The first group consists of four people, the second of three people and the last two groups of two people. All ratings will be on a scale from 1 to 5, where 1 is very dissatisfied, and 5 is very satisfied. The form used for this evaluation can be found with the other surveys in the Appendix.

Chapter 4

Prototype

In this chapter, the application developed for this project will be presented. The application is called YAPS.life, as in *young aspiring professionals* due to that being the main target group for the application. Its purpose is to help individuals find compatible roommates and then give them the tools to find a home together based on their shared preferences. Each view of the application will be shown and related to the user journey. Seeing how the prototype looks will give the reader a more intuitive understanding of how the algorithm works when it is presented in the next chapter. Due to many users visiting the service from mobile devices, the design can be scaled down to accommodate those screen sizes as well.

An overview with links to different sections in the chapter is presented here:

4.1 App Header

Present in all parts of the application is the App header. As can be seen in figure 4.2 and 4.3 it looks different depending on whether the user is logged in or not.

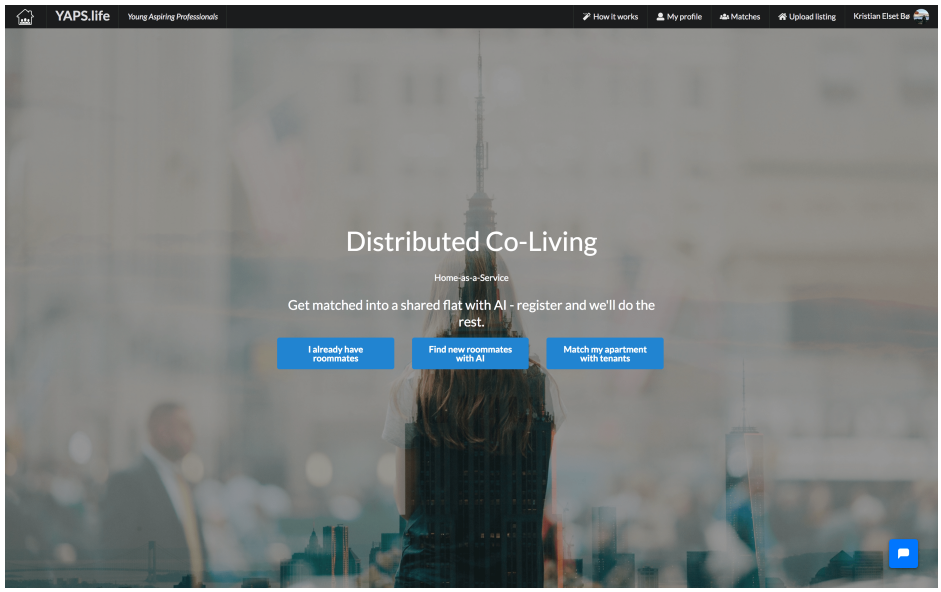


Figure 4.1: Landing page

If the user clicks on "My profile", "Matches" or "Upload listing" without being logged in, they will be redirected to the Login/Sign up page detailed in section 4.3. When the user is logged in the previously mentioned buttons will link to their corresponding pages as described in sections 4.4, 4.5 and 4.6. In addition, when logged in the "Log in" and "Sign up" buttons disappear in favor of a section with the user's name and profile picture. Clicking this will give the user options to log out of the application or go to account settings where he can delete his profile.

Figure 4.2: App header, logged in

Figure 4.3: App header, logged out

4.2 Landing Page

The landing page for YAPS.life is the main entry point for users of the service. The first section, seen in figure 4.1, introduces the user to the service and gives him the following three options:

1. I already have roommates
2. Find new roommates with AI
3. Match my apartment with tenants

Assuming the user is logged in the first button takes him to a profile page where he can adjust the information and preferences he has previously entered. The second takes him to a match list where he can view all previous matches and create new ones. The last button takes the user to the upload listing page.

Further down there are sections for Process, figure 4.4, Landlords, figure 4.5, Frequently Asked Questions and Answers, 4.6. Finally, there is a short About section with contact information and the bio of the creator in 4.6. In the Process section, the process of signing up, getting matched and moving in are described, and the Landlord section details how the service works for people looking to find tenants.

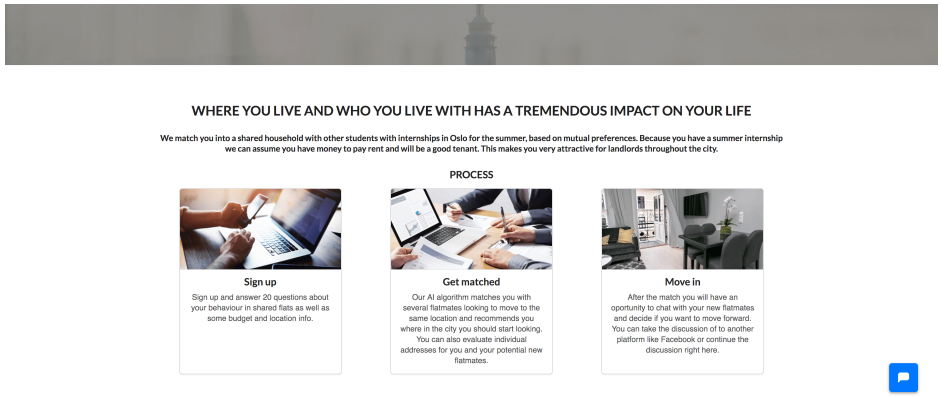


Figure 4.4: Process section

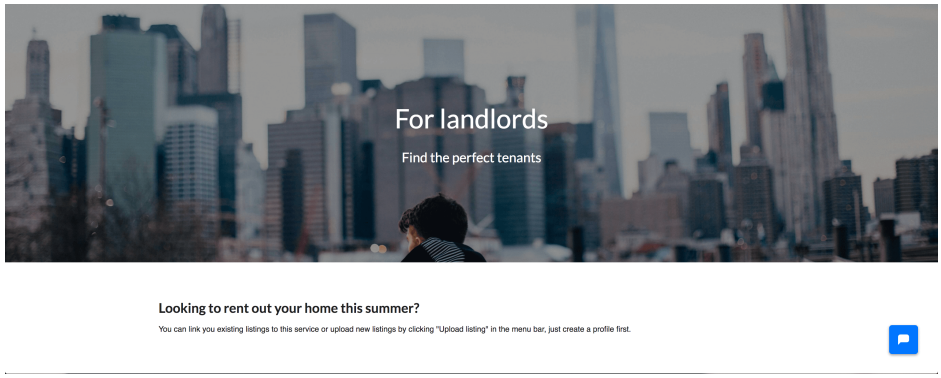


Figure 4.5: Landlord section

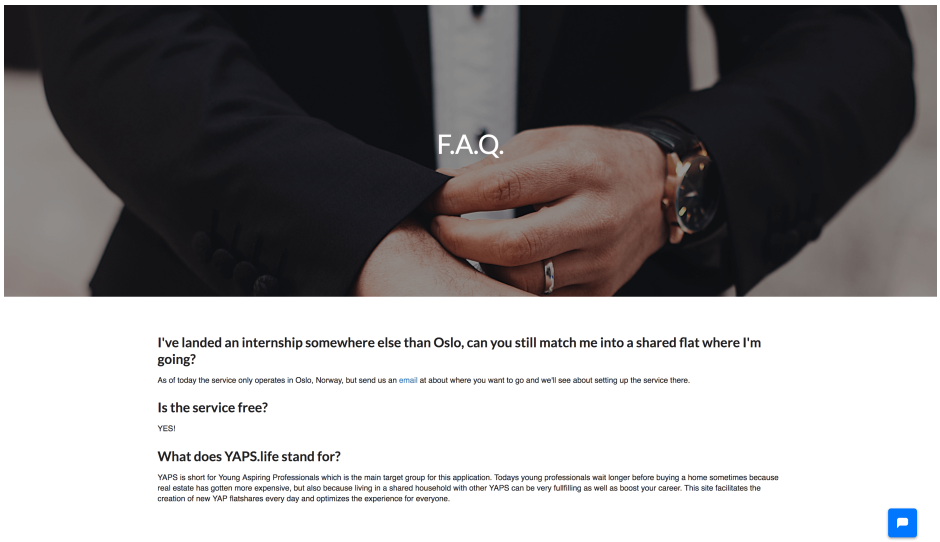


Figure 4.6: FAQ section

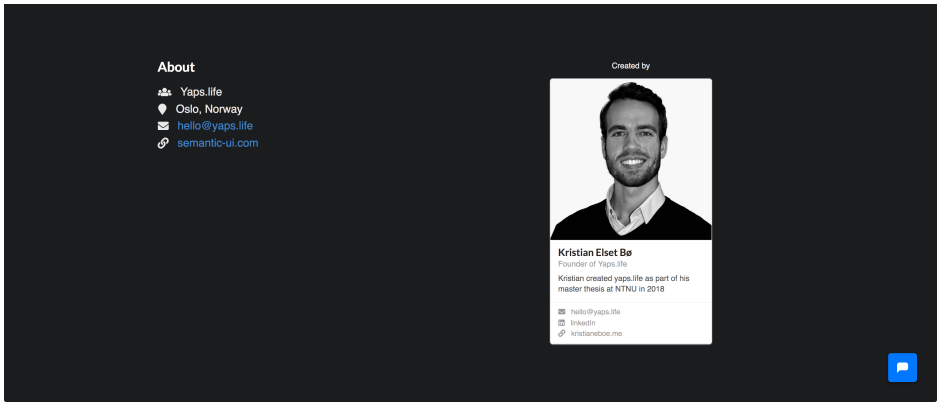


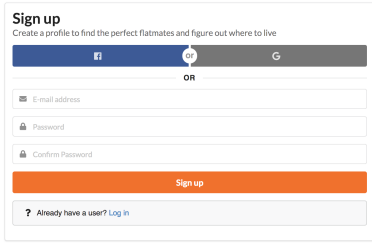
Figure 4.7: About section

4.3 User Creation

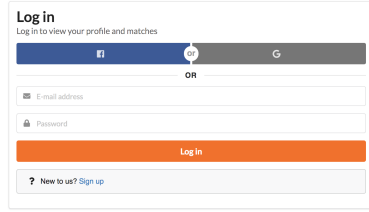
Here the user can log in to the application, or sign up if it is a first-time visit. As of now, there are three ways to create a user:

- The user's Facebook account
- The user's Google account
- Email and password

Signing up with either Facebook or Google is desired as the information from those profiles can be used to help fill out the rest of the users profile automatically. Any of the above will result in the creation of a user in the database, but the user can not fully use the service before he has filled out his profile in the profile page, see 4.4. In the future, a way to sign in with LinkedIn should also be implemented as that social network will have valuable information about the job history of a user.



(a) The Sign Up view. It has an extra field for making sure the password is entered correctly.



(b) The Login view.

Figure 4.8: The Sign-up and Login forms for the application

4.4 Profile Page

The profile page has the role of setting the users information and preferences so that he can be matched with other users of the group recommendation system. By setting these precise preferences, it helps the knowledge-based recommendation system avoid the cold start problem as described in section 2.3.1. How the algorithm uses the information collected here is described in section 5.3

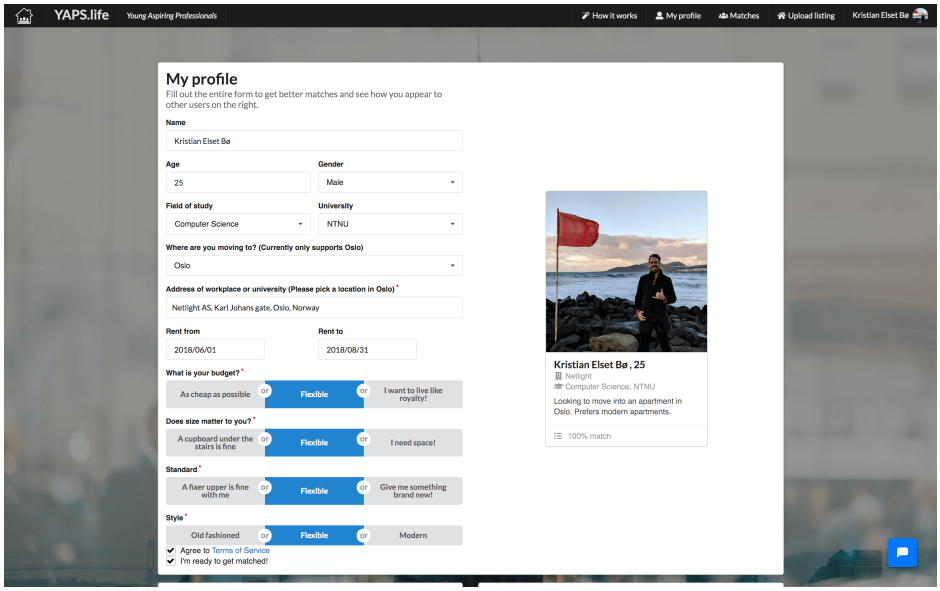


Figure 4.9: About me

4.4.1 About me

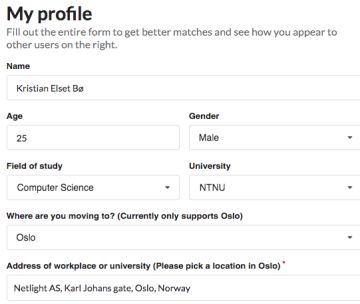
The following attributes describe the user:

- Name, age, and gender
- Field of study and university
- The city the user is looking to move to
- The name and address of the user's workplace

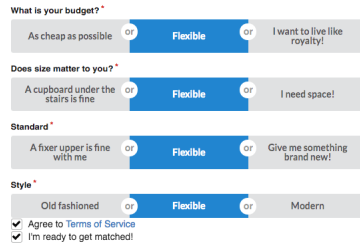
Furthermore, the user describes what they are looking for in a future property by defining the following concepts

- Budget
- Size
- Standard
- Style

These four attributes, as well as the address of the user's workplace, will be used in ranking the listings on the Match page, section 4.6.



(a) The input fields of the profile section



(b) Property goals questions

Figure 4.10: Where you define what you are looking for in a potential home

The above information also defines how the user will be presented to other users of the service as can be seen in figure 4.11. Property preferences are presented as a free text while name, age, workplace and education are presented in an itemized fashion.

4.4.2 Defining your habits in a shared accommodation

The other half of the profile section is where the user tells the system about his habits in a shared living environment. All questions are posed as a Likert style statement [Likert, 1932] where the answers are defined by how much one agrees with the statement on a scale from 1 to 5, where 1 means disagreeing completely, and 5 means agreeing completely. Furthermore, the questions are divided into four categories

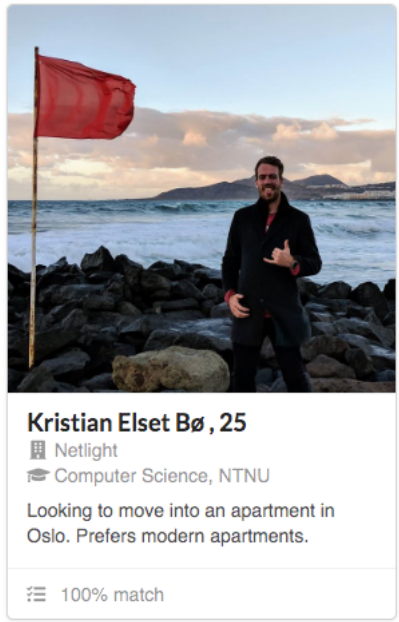


Figure 4.11: A card representing how the user will look to other users of the service

- Social habits, revolving around habits like partying, having friends over and when one comes in and out of the house.
- Cleanliness, a section about the attitude towards keeping the home clean and organized.
- Social openness, concerning how much time is spent alone in one's room versus out in the shared areas.
- Social flexibility, regarding flexibility towards flatmates sexual and religious orientations and openness to helping them out or letting them have friends over.

These 20 questions will be used to match the user's personality and habits to new roommates. How it is done is described in the next chapter in section 5.5. The full list of questions can be found in the appendix, section A.

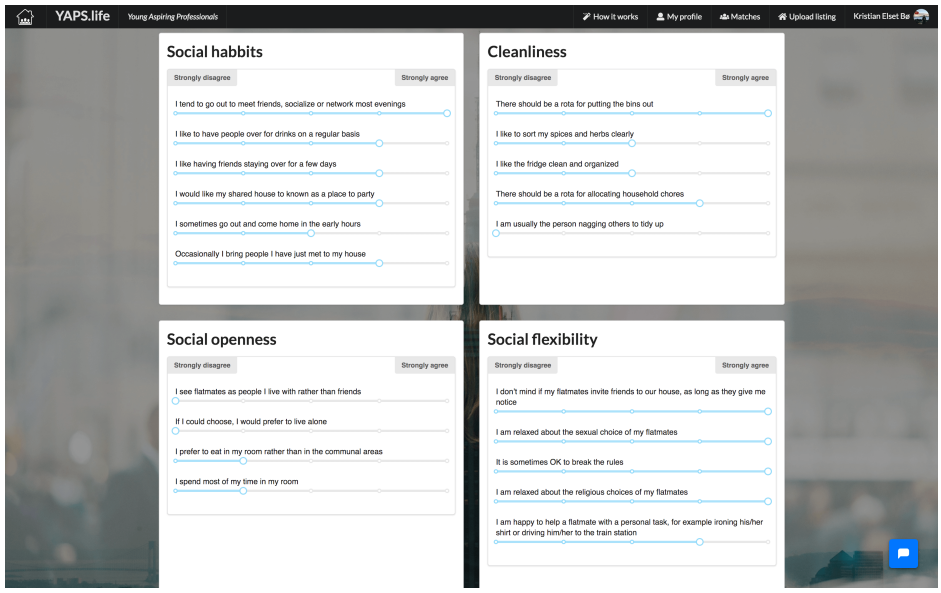


Figure 4.12: Habit questions

4.5 Match List

When the profile is set up, it is time to go to the match list page. In this view, the user can see a list of all current matches with metadata. He can also create new matches from here. By clicking the "create a demo match" button the system will set you up in a match with the best possible flatmates chosen from a 1000 test users. This allows the user to see how the interface will look when he is matched in the future. If the user chooses to create a new solo match he will be redirected to the Match page, 4.6, and get matched with single room listings. The match list is shown in figure 4.13.

The current matches are presented as a list with meta information about the match like:

- An automatically generated name based on the group size.
- The date of the match creation.
- The ideal starting position for the match to start their housing search.
- The alignment of the personalities in the match and their housing goals.
- Name and picture of the match participants as well as where they Work.

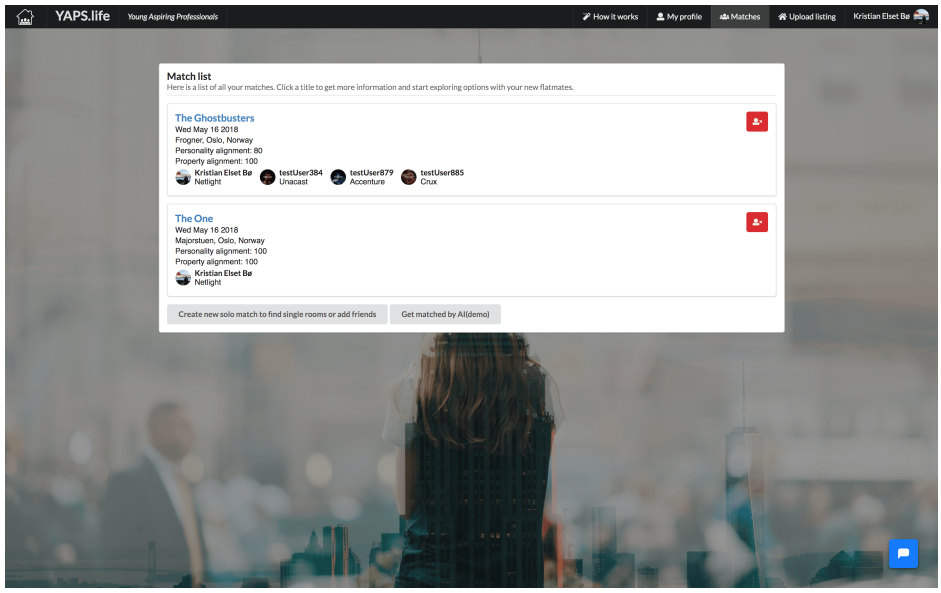


Figure 4.13: Match list

4.6 Match View

The match view is the main view for the actual recommendation system of real estate listings. Here, a user can see information about his new potential flatmates, add new flatmates, and evaluate listings either manually or through a link. Present is also a chat room where he can talk to new flatmates and discuss relevant listings.

4.6.1 Flatmates

The first section is a presentation of the participants in the match and the alignment of their personalities from the habit questions and their property goals from the property questions. If there are less than four flatmates in the match, either because someone leaves the match or a new custom match is created, the users can add other users to the match.

Upon adding another user to the match, the personality and property alignments are recalculated. As a privacy measure, one can only add new users by email.

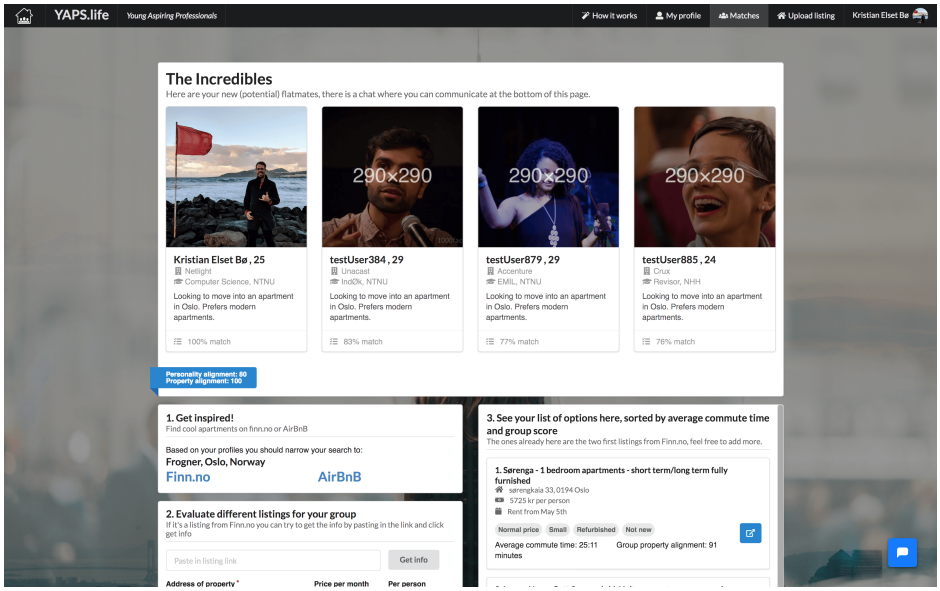


Figure 4.14: Match view

4.6.2 Real estate recommendation

The real estate recommendation section consists of the following three panels:

1. Inspiration
2. Evaluation
3. Presentation

Ideally, there would be no need for the Inspiration and Evaluation part, and instead present the user with options directly. However, since this is a general recommendation framework, not tied to a specific service or data set, the service has no access to relevant listings and needs the users help to import them from somewhere else.

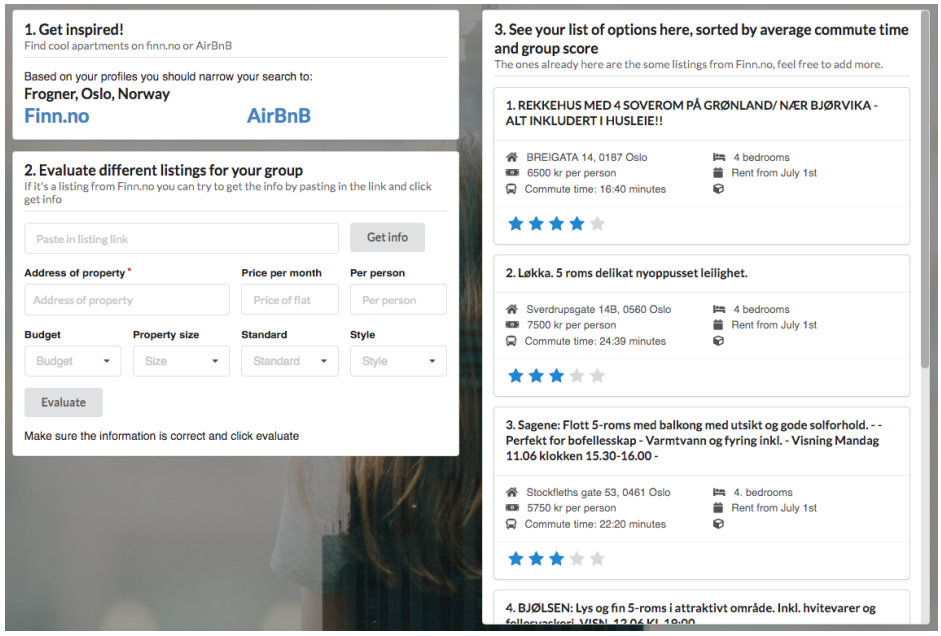


Figure 4.15: Match view

Inspiration

The inspiration panel contains three components. The first is the district of the city the user is moving to, marked in **bold**, that has been determined to be the

best origin for the match. How this is determined is presented in section 5.8.1. Then, there are two URL links to the services finn.no and airbnb.com. Both services use URLs to determine queries to their site and based on the group's preferences the yaps.life system constructs such queries. When the user clicks, they are redirected to the respective sites with queries pre-filled into the system. Query construction is described in section 5.8.3.

Evaluation

In this panel, the user is presented with tools to upload a listing of their own choosing. It can be a listing from the sites mentioned above, or ones like them. However, they can also upload listings they have heard of through family or friends that are not listed on the internet. As long as the listing has an address and a price, and can be given a rating for size, standard, and style it can be uploaded and ranked.

Presentation

Arguably the most important panel is the listings panel. Here the different listings found by the system, or uploaded by the user, are presented in a unified list ranked by commuter time, price per tenant, size, standard, and style. Each property has its own card with details about the property as seen in figure 4.16. The properties listed on each card are:

- Address
- Price per tenant
- Average commute time
- Number of bedrooms
- Rent from date, and to date if available
- Size

At the bottom of each card is a rating given to the card by the system based on the listings match for the group. If the listing was uploaded on the YAPS.life system there will also be chat box directly to the proprietor of the listing. Upon matching a message is injected into the conversation urging the group and land lord to get in touch. How the listings are ranked are described in 5.8.4. Based on price and commute time, two cards also get a label for being the cheapest or the fastest option.

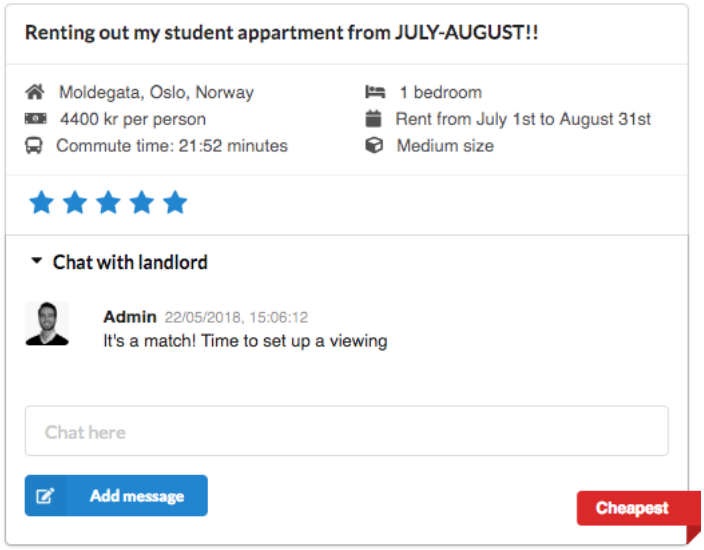


Figure 4.16: Listing card

4.6.3 Chat room

The addition of a chat room was deemed necessary to the service so potential flatmates could talk to each other and make sure the recommendation of the system was actually good. Activity in the chat could also be used to automatically remove inactive users from the match to allow new participants to join. The chat room is shown if figure 4.17.

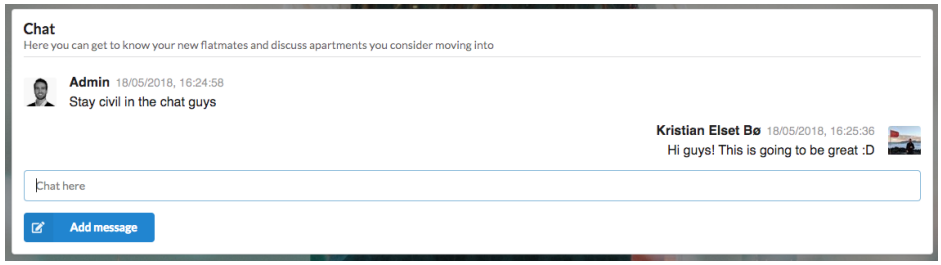


Figure 4.17: Chat room

Chapter 5

Design of Recommendation Algorithm

The recommendation algorithm is at the heart of this application. In this chapter, the algorithm itself is formalized through a definition before it is walked through step by step.

5.1 Requirements

As described at the beginning of the background theory section, 2.1, one of the scenarios envisioned for this application is the following:

Oslo, the capital of Norway, receives thousands of hopeful young professionals each year. Renting a studio apartment for yourself can easily escalate into 1500 - 2500 USD, not to mention actually buying a place, so flat sharing is the sensible option. There are several real estate marketplaces where you can view listings or contact potential roommates. However, there is no guarantee that it will be a good match and for people who have not lived in Oslo before it can be hard to know which areas of the city are good to live in. What they all need is a service where they can enter information about themselves, where they work and what they are looking for in an apartment, and then be recommended not just where to live, but also whom to live with.

Using this scenario, several requirements for the service have been extracted into table 5.1. The prototype aims to address these requirements.

Table 5.1: Requirements for the prototype

ID	Requirement
R1	Groups should consist of user with similar co living habits.
R2	Groups should consist of user with similar goals for an apartment.
R3	Groups should consist of users who want to live in the same location, in the same timeframe.
R3	Groups should be of size 3, 4 or 5 to provide meaningful savings on rent
R4	A user who has already been matched should not be matched again unless indicated by the user.

5.2 Definitions & Overview

To formalize the design of the recommendation algorithm a definition of a Real Estate Group Recommendation System, REGRS, is found below.

In a Real Estate Group Recommendation System, a set of users $\mathbf{U}$; matches, $\mathbf{M}$; homes, $\mathbf{H}$; and landlords, $\mathbf{L}$; are used to provide real estate recommendations to users and tenant recommendations to landlords. A match, $\mathbf{U} \cap \mathbf{H}$, consists of a group of users and the listings they are matched with. A user can be part of multiple matches, and a landlord can administer multiple homes. Each user $\mathbf{u}$ has completed a profile with preferences regarding personality and what they are looking for in a new home. This information is used to cluster users into matches and then create a group profile for property preferences. The goal of the recommendation system is to match the most relevant matches, $M' \subseteq M$ and homes, $H' \subseteq H$ together. The relationship between the sets can be viewed as a graph in figure 5.1.

The recommendation system works with multiple phases. The first being profile generation where the user first fills out personal information like age, gender, where they work, etc., and then four questions about property. Finally, they answer twenty questions about their habits in a co-living environment. When the algorithm is called to do its work, it extracts all users from the database and filters them on match location, i.e., which city they will be matched into. Then, the available users are clustered based on the dates they are looking for accommodation, the property options they are looking for and last, but not least

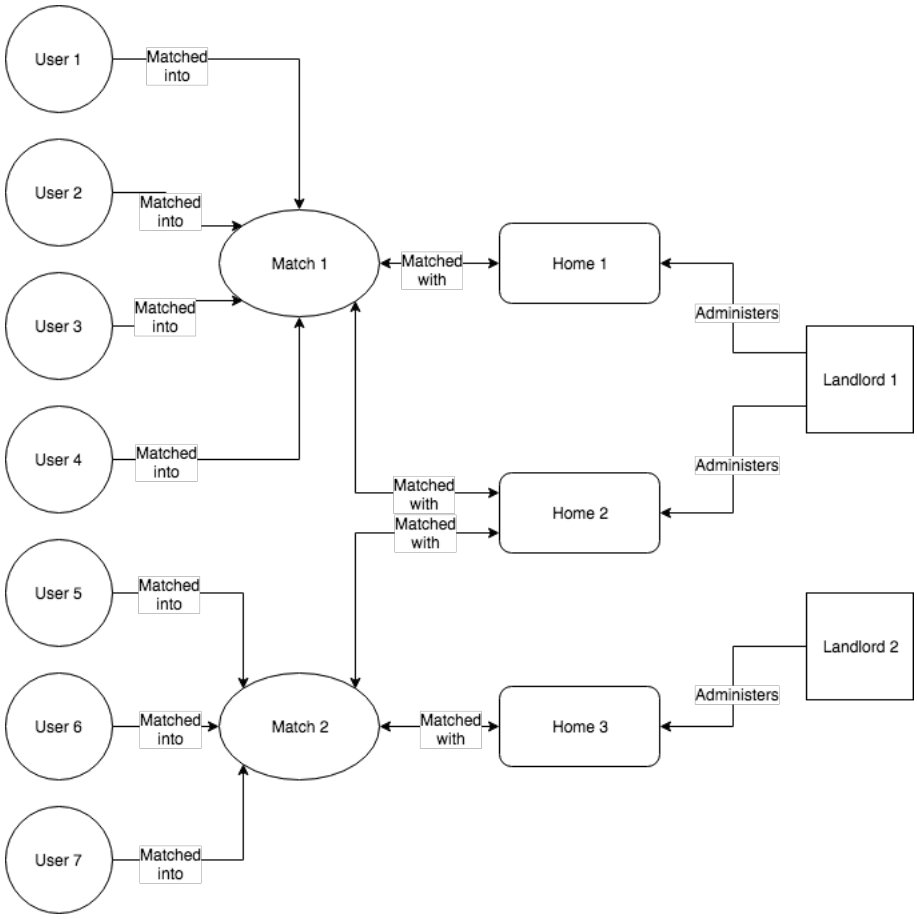


Figure 5.1: Real Estate as a Graph

their habits and interests.

When a match is created from the clustering process, several data sources will be consulted in order to find homes, including finn.no, Airbnb, and the services' own registered listings. If the home matches with what the group is looking for the listing will be injected into the search list of the match as displayed in 4.6.2.

5.3 Profile Generation

The full list of collected information during this stage can be found in the profile section, 4.4 of chapter 4. The information used for matching purposes are:

- Where the user works.
- The budget, size, standard and style of apartment the user desires.
- Twenty questions on a scale from 1-5 regarding different statements about co-living habits.

Other information like field of study, university, name, and age are needed in order to present the user to other users upon being matched. As opposed to Li et al. [2017a], described in the background theory in section 2.6.5, who indicated that asking the user to fill in too much information would be too cumbersome, the YAPS.life prototype asks for quite a lot. However, having an intuitive user interface, which makes it easy to input this information, is hoped to mitigate the user's encumberment. The reason for this extensive collection of user information and preferences is to let the system avoid the cold start problem described in 2.3.1, and thus be able to provide good recommendations with only a few users in the system.

5.4 Context Filtering

When a matching happens, the system gathers all available users and filters them based on match location, i.e., which city they are moving to. Another important context aspect is the date the user needs the apartment from. Filtering users based on this attribute at this stage was considered, however, it was decided that this context parameter did not need to be absolute, like the match location. Therefore the context filtering for date was moved to the user-user distance function described in section 5.6.

5.5 Group Formation

As described in section 2.4.1, group formation is an essential part of a group recommender system. In this prototype, two ways of creating a group are supported. The most straightforward method is to have users add each other's profiles into an A Priori Group. This is for the group of friends scenario described in 2.1.1. More interesting is when the system does Automatic Group Detection as per section 2.4.1 and in the scenario described at the start of this chapter.

Three clustering algorithms were implemented to do the group matching. The first was k-Means, an old and reliable clustering algorithm that clusters the users into k potentially large clusters that then have to be split into smaller matches. The second was kNN, or k Nearest Neighbour, which was adapted to find the best match per user and then remove those users from further matching. The third, was a hybrid of the two which first used k-Means to create initial broad clusters, and then kNN to split these into smaller matches. That way of combining two techniques is called a Cascading Hybrid system, as described in section 2.2.4.

After a careful evaluation of all three algorithms, which will be presented in chapter 7, the Hybrid approach was chosen as the main algorithm for the prototype.

5.5.1 Initial clustering with k-Means

The first step is to take the available users for each match location and group them into k clusters. At first, it was theorized that if there were a 100 users, k would be based on the number of users wanted in a group, see equation 5.1.

$$k = \frac{number\ of\ users}{flat\ size} \tag{5.1}$$

However, due to the random initialization of the centroids, the algorithm ended up with some clusters with many users and some clusters without any users at all. Lowering k to 5 for any amount of users ensures a higher probability of all clusters containing several users and in theory makes the clusters mimic different personality types.

In the end, it was decided to go with a k that scales as the user size increases, but only logarithmically to continuously make sure no cluster ends up empty. Because k needs to be an integer the result of the logarithm was rounded up to the nearest one. For a more thorough explanation of k means consult section 2.5.3 in the Background Theory chapter.

$$k = \log_2(nr\ of\ users) \tag{5.2}$$

5.5.2 Final match groups determined by k Nearest Neighbours

The initial broad clusters often have many more users than can fit into a single house, and that is where kNN comes in. By applying the kNN algorithm to each of the clusters produced with k-Means, the clusters are then divided into smaller

groups of an appropriate size corresponding to the k chosen. Note that this is a different k than the one used for k-Means. In the production system today, four is chosen as the k for kNN. This means that even if a user were to leave a match, the group would still have enough members to make the group recommendations meaningful. Setting the k to five could lead to groups who would have a hard time finding a place together because of the lack of homes that can accommodate that many people.

The algorithm works by first selecting a user from the initial cluster, then calculating the distances between the user and all other users in the same cluster. Then it sorts them from shortest to longest and extracts the top three. These, now four, users are then locked into a group and removed from further matching within the cluster. The algorithm repeats this procedure until all users from the large initial cluster are sorted into matches. Because the implementation is slightly different than the standard kNN algorithm, it is included here in Algorithm 5.1. The programming language used is JavaScript.

Listing 5.1 kNN algorithm one match per person

```
function knnClusteringOneMatchPerUser(vectors, K) {
  // vectors are the combined personality and property vectors to
       be clustered.
  //K is the group size the algorithm is supposed to produce.
  const clusters = []
  // Map vectors into an object that remembers initial index of
      vector.
  // This is used to map the clustered vectors back into users.
  const vectorsWithIndex = vectors.map((v, i) => ({ v, i }))
  // While there are users to cluster, do:
  while (vectorsWithIndex.length > 0) {
    const scores = []
    // Remove the first user from the user pool.
    const u = vectorsWithIndex.pop()
    // Iterate through remaining users and score them compared to
         the current user.
    vectorsWithIndex.forEach((v, j) => {
      const score = cosineDistance(u.v, v.v)
      scores.push({ i: v.i, j, score })
    })
    // Sort the socred users from best to worst.
    scores.sort((a, b) => a.score - b.score)
    // Extract the top K.
    const topK = scores.slice(0, K-1).sort((a, b) => b.j - a.j)
    // Remove the top k from the user pool.
    topK.forEach(el => {
```

```
24          vectorsWithIndex.splice(el.j, 1)
25        })
26        // Concat the original user with the topK and push them into
              the returned cluster vector.
27        clusters.push([u.i].concat(topK.map(el => el.i)))
28      }

30      return clusters
31  }
```

A note on using kNN alone

It is possible to forgo k-Means and simply use kNN to cluster all users into groups directly. The inherent strengths of this clustering method would be that all groups would consist of exactly four people, except possibly the last one. Further, the first person the algorithm picked would be guaranteed a good match. However, that is where the downside lies as well. While the first user gets the perfect match, the available user pool shrinks with every iteration. Thus the four last people in the pool will have no options but themselves when ranked for a match. Another scenario is when the first user the algorithm evaluates is an outlier. It could then potentially grab three other users who would have a much better correlation with someone else. This algorithm produced the best and worst matches of the ones investigated for user matching as will be presented in section 7.1 in the Evaluation chapter. The difference in running time on the datasets used for evaluation were negligible.

5.6 Feature Reduction and Similarity Calculation

The users are represented by both the answer to their 20 personality questions as well as their 4 real estate goals conjoined into one vector of 24 indices. Each index is on a scale from -2 to 2 to represent how much a user agrees with a statement. All clustering algorithms need a similarity metric to compare the user-user similarity of these vectors in order to produce good matches. From section 2.5.1, both the Euclidean distance and Cosine distance were considered as similarity metrics for this prototype.

Euclidean distance would compare the two users question by question and give them the similarity score of the combined distance between all answers. Cosine on the other hand, looks at the directions of the different vectors. This in beneficial because, as we have from Amatriain et al. [2009] back in section 2.2.6, users opinion on what constitutes agreeing strongly with a statement can differ vastly.

Cosine similarity is less concerned with the extremity of the users answers, but gives a better indication of the general direction the users answers indicate.

Comparing two vectors of length 24 is no computational challenge for either of these similarity metrics, however they are both quickly hit with the curse of dimensionality. As described in section 2.5.2, the curse of dimensionality is the fact that vectors in high dimentional spaces lose much of the intuitive distance between them. Beyer et al. [1999] reasons that because the volume of the space increases so quickly with the dimentionality resulting in that In this sense, nearly all of the high-dimensional space is "far away" from the centre.

To combat this, we use the fact that the 20 personality questions are divided into four categories to create an overall score for each category. This is done my summing over the individual questions. Now, the user vector consists of eight instead of 24 indices. However, the scale has been shifted drastically with the first part of the vector now ranging from values of -10 to 10. By weighting the property vector by 5 we increase its range to the same amount to ensure the personality questions won't dominate the distance score.

Deciding which similarity metric is better is often determined by the data you have as the intuition. As chapter 7 shows, the best combination for this prototype turned out to be first clustering the dataset using k-Means with the Euclidean distance, and then dividing the clusters into more granular matches using the cosine similarity.

5.7 Creating a Match

After the users are divided into groups, a match object is created. Upon creation, several alignment calculations are performed regarding the users' personality alignment, property alignment and a combination of the two. The way the alignments are calculated can be found in section 3.2.1.

When the group is settled it is time to decide on a group model strategy to perform recommendations on. As detailed in section 2.4.2, there are two main approaches to this. Either giving individual recommendations to all members in the group and then aggregating them, or merging the group members into a group persona and recommend items directly to it. For the YAPS.life prototype it was decided to go with the latter. To combine the individuals preferences, their property vectors are averaged into a group property vector, as can be seen with the example in figure 5.2. Using a group persona helps limit expensive API calls to external services like Google's Distance Matrix API and since the group has

Figure 5.2: An example of averaging the individuals vectors into a joint group property vector.

$$\begin{array}{rc} u_1 = & \left(\begin{array}{cccc} -2 & 0 & 2 & 0 \end{array}\right. \\ u_2 = & \begin{array}{cccc} -2 & 0 & 2 & 2 \end{array} \\ u_3 = & \begin{array}{cccc} -2 & -2 & 0 & 0 \end{array} \\ u_4 = & \left.\begin{array}{cccc} -2 & 0 & 2 & -2 \end{array}\right) \\ \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots\ldots\ldots \\ groupvector = & \left(\begin{array}{cccc} -2 & -0.5 & 1.5 & 0 \end{array}\right. \end{array}$$

already been clustered for similarity a simple average yields a good representation of the group persona. If the group had been more diverse, a strategy like least misery could be considered instead.

Creating a chat room

An additional step at this point is to create a chatroom for the users of the match to interact. While the algorithm technically guarantees the matches to be good on paper, real-world users will probably want to make sure their new flatmates can be reached online before actually deciding to move in together. The chatroom has no further bearing on the apartment recommendation.

5.8 Real Estate Recommendation

As the match is created in the database, either from the clustering algorithm or by a user on the website, a series of events are set in motion serverside. By using the new flatmates' work locations, the algorithm consults the Google Distance Matrix Service[1] on which district in the match city suits the group best according to average commuter time. Then, based on the group's composition, a search query for Finn.no and Airbnb.com are created. The one for finn.no is used to scrape the website for the ten first apartments in the area deemed most suitable for the group by the earlier stage of the algorithm as well as 20 more listings from a general search of housing in the match city. After combining the two lists and filtering them for duplicates the individual listing details are matched with the group's property vector. The listings with a sufficient score are then inserted into the Distance Matrix function with regards to the users' workplaces again and ranked on the combined Euclidean distance to the group property vector.

[1] `https://developers.google.com/maps/documentation/distance-matrix/intro`

5.8.1 Get best origin for match

While excellent, the distance matrix service from Google does suffer some limitations. First and foremost is the API limit constricting free use to 2500 elements per day. For each query, there is an average of four workplaces, and then even at only four city districts, that turns into a $4x4$ matrix of 16 elements. This means that for the service to stay free the maximum amount of calls are roughly $2500/16 = 156$ per day. Therefore, any initial evaluation of an apartment that can lead to its disqualification for the group is welcome, as it saves the system from making more API calls than necessary. If the service was monetized, this limit could be increased.

The locations in Oslo chosen are:

- Frogner
- Majorstua
- Grunerløkka
- Gamle Oslo

Together these four districts comprise the outskirts of Oslo's central business district and are thus well positioned as candidates for housing people with workplaces scattered all over the city.

An alternative to the Google Distance Matrix API was Mapbox. Their Distance Matrix API[2] has a more generous limit of 50 000 per day; however, they only provide travel durations for driving, walking, and cycling. As far as the author knows, Google is the only service with close to global coverage on public transport time calculation. In the future, a combination of the two APIs could be used to give users more choice in how they prefer to transport themselves to their workplace.

5.8.2 Get combined travel time and determine the best origin

The response from the Google Distance Matrix API is the travel time and length in kilometers from the four origins, preselected based on city, to the destinations, I.E., the workplaces of the users in the match. As travel time is more relevant than actual distance when it comes to getting to work on time, it is selected as the metric to average. The averaging is done by simply iterating through the matrix,

[2] `https://www.mapbox.com/help/define-matrix-api/`

adding the travel time results together before dividing on the number of flatmates.

When the scores are calculated, we simply pick the origin district with the lowest average time. Future work here could include implementing a Least misery solution as described in section 2.4.6 on aggregation strategies in group recommender systems.

5.8.3 Construct query strings & get properties

With the best origin now determined, the system constructs search queries for the sites finn.no and airbnb.com. Both services use a URL based querying system so adding and removing URL parameters affects the search. For this prototype, most of the focus lies on Finn.no as Norway will be the first place of use for prototype. The query parameters constructed and how they are determined can be found in table 5.2.

Table 5.2: URL parameters for finn.no

URL Parameter	How it is determined
City	The city constraint is determined by the groups match city. In the prototype it defaults to Oslo.
District	District is determined by the best origin algorithm described in section 5.8.1.
Number of bedrooms	Based on the number of people in the group. In Finn's system this parameter does not ensure the exact number of bedrooms needed, instead it returns all listings with the same number or higher.
Property types	Finn.no offers listings for many property types. For this group recommendation system the listings are pre-filtered on: Single house, Apartment, Town house, and Semi detached

The queries are then sent to the match object and then used by the server to do an actual search on the service finn.no. By scraping the 10 topmost results from the fine tuned query described above, and another 20 from a less granular

search, to get more breadth in the results, the results are combined and filtered for duplicates.

5.8.4 Evaluating a listing

Ideally, all properties would be rated equally. However, because of the above-mentioned API limits, it is unfeasible to get the commute time score for all these apartments. Instead, the algorithm takes a two-step approach.

Step one is a basic filtering on the collected apartments based on whether they have the exact number of bedrooms needed and an initial group score for the property based the Euclidean distance from the group's joint property vector to the derived property vector for the apartment. The apartments that make it through this filtering are then submitted to the Distance Matrix API to get their commute time score. Now the algorithm has all it needs to set a final score for the listing based on another weighted Euclidean distance.

5.8.5 Matching groups with properties in the system

After adding the external listings, a similar process is followed for matching the group with existing listings on the yaps.life platform.

5.8.6 Adding properties manually

To provide value to users who are not in a serviced match location one can also manually add apartments to the service through the user interface. These apartments are evaluated and connected to the match in the same method as the external listings found on match creation.

5.8.7 Adding properties uploaded by landlords

The last way an apartment can be injected into a match is if a landlord uploads it to the platform after the creation of the match. The apartment will be injected into all matches who have not indicated that they have decided on a place yet, and where the features of the apartment are a good match for the group.

Chapter 6

Architecture

Now that the recommendation algorithm has been presented conceptually it is time to elaborate on the actual technical implementation. In this chapter the development environment and tools used to develop the prototype are presented.

6.1 Development Environment

Modern web development requires a host of tools in order for developers to work effectively and this section presents the most relevant for developing this prototype. All code was written in Microsoft Visual Studio Code[1] which is a light weight text editor with support for linting, code completion and version control.

For package management it was decided to use YARN[2], which is an extension of the Node Package Manager[3]. Package management in web development is a way to manage external code integrations into the project. For continuous integration the service Travis CI[4], which is free for open source software, was used.

6.2 Architecture

The architecture of a web application is often presented as a view layer, a business logic layer and a data storage layer[Fowler, 2002]. Typically the view layer is a combination of HTML, CSS and JavaScript that talks to a server, the business layer, that then communicates with a database in the data storage layer.

[1] `https://code.visualstudio.com/`
[2] `https://yarnpkg.com/en/`
[3] `https://www.npmjs.com/`
[4] `https://travis-ci.org/`

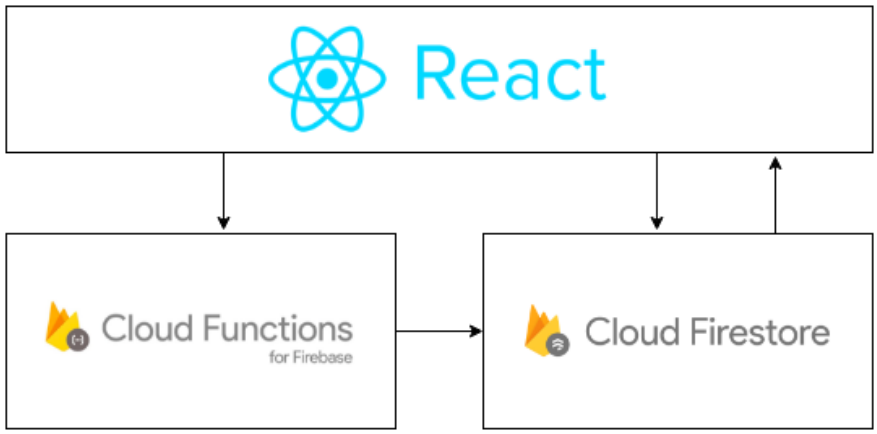


Figure 6.1: The prototype architecture in components

However, this is not the only way to structure a web application. A new trend called server-less computing has been made more accessible thanks to services like Amazon Web Services Lambda[5] and Cloud Functions from Google Cloud Engine[6]. Instead of deploying a server the developer simply writes the code to be executed on the server-side. Then, the service provider creates a HTTPS endpoint the function can be activated from. Another possibility is to attach the cloud function to a database event like a new document being created, or an old one edited, this is called a cloud function trigger.

On the front-end, the prototype uses the well known React.js library created by Facebook.

6.3 Front-end

The frontend of the application is where users interact with the system. Since this is a web based system, all interaction happens in a web browser on either a desktop or a mobile device. Web browsers can only display combinations of HTML and CSS with interactivity through JavaScript. In order to build this application it was decided to use the framework React.js by Facebook.

[5] AWS Lambda
[6] `https://cloud.google.com/functions/`

6.3.1 React.js

React is a declarative, component based JavaScript framework for frontend web applications. It was developed by Facebook and released in 2013. Now it is maintained by facebook, instagram, and a community of individual developers and corporations. The fact that it is component based means the application is composed of a hierarchy of components which are written individually and can be reused.

In order to load data from other sources, like the Firestore database, we use one of React's lifecycle methods called *componentDidMount* to make the request. An example of how the data for the Match List page from section 4.5 is collected can be found in listing 6.1

Listing 6.1 Loading external data in ComponentDidMount

```
componentDidMount() {
    auth.onAuthStateChanged((user) => { // Check if user is logged in
      this.setState({ user })
      if (user) { // If logged in
        this.unsubscribe = firestore.collection( users ).doc( user.uid).onSnapshot((doc) => { // Subscribe to datastream for user object from firestore
          const userData = doc.data()
          const { currentMatches, gettingCloudMatched } = userData
          const matches = currentMatches || {}

          this.setState({ gettingCloudMatched }) // Signal the user that something is happening

          Promise.all(Object.keys(matches).map(matchId => firestore.collection( matches ).doc(matchId).get()) // Get match documents
            .then(results => this.setState({ // Update state with the matches
              matchesLoading: false,
              userData,
              matches: results.map(res => res.data())
            }))
        })
      } else { // If not signed in, redirect to log in page
        this.setState({
          redirectToSignIn: true
        })
```

```
23          }
24      })
25  }
```

Routing

React produces single page apps, however separating the content into different URLs is still necessary for usability purposes. To do this the prototype uses react router[7] which dynamically displays app content based on the URL. This lets us split the prototype into pages, as seen in listing 6.2

Listing 6.2 Routing with React Router

```
render() {
    const { user, userData } = this.state
    return (
      <div className="app">
        <AppHeader user={user} newMatches={userData ?
            userData.newMatches : false} />
        <Switch>
          <Route exact path="/" component={Home} />
          <Route exact path="/create" component={Create} />
          <Route path="/matches/:matchId" component={Match} />
          <Route path="/matches" component={MatchList} />
          <Route path="/profile" component={Profile} />
          {/* <Route path="/landlord-view" component={
              LandlordProfie} /> */}
          <Route path="/apartment-finder/:matchId" component={
              ApartmentFinder} />
          <Route path="/TOS" component={TOS} />
          <Route path="/account-settings" component={
              AccountSettings} />
        </Switch>
      </div>
    )
  }
```

6.3.2 Create React App

Create React App [cre, 2018] sets up a development environment that has everything needed to build a modern React app with only one build configuration.

[7] https://reacttraining.com/react-router/

Create React App comes packaged with tools like Babel, ESLint, Jest and Webpack. Babel is a compiler, which allows transpilation of ES6/ES7 code to standard JavaScript that runs in the browser. In the project, Jest was used for writing unit tests. Facebook defines Jest as a "JavaScript testing solution. Works out of the box for any React project." [jes, 2018]. These are tools that were used all the time when developing YAPS.life, thus bundling them into one dependency was a great advantage.

6.4 Back-end

The back-end of this application consists of the database and the serverless cloud functions. Both are products made by Google in a service called Firebase, which is a BaaS (Backend as a Service).

6.4.1 Cloud FireStore

Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud Platform. Like Firebase Realtime Database, it keeps data in sync across client apps through realtime listeners and offers offline support for mobile and web so that responsive apps work regardless of network latency or Internet connectivity. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud Platform products, including Cloud Functions. [8]

Cloud Firestore is a cloud-hosted, NoSQL database that iOS, Android, and web apps can access directly via native SDKs. Cloud Firestore is also available in native Node.js, Java, Python, and Go SDKs, in addition to REST and RPC APIs.

Following Cloud Firestore's NoSQL data model, data is stored in documents that contain fields mapping to values. These documents are stored in collections, which are containers for documents that one can use to organize data and build queries. Documents support many different data types, from simple strings and numbers, to complex, nested objects. Subcollections can also be created within documents and build hierarchical data structures that scale as the database grows. The Cloud Firestore data model supports whatever data structure works best for the app in question.

Additionally, querying in Cloud Firestore is expressive, efficient, and flexible. Shallow queries can be created to retrieve data at the document level without

[8] `https://firebase.google.com/docs/firestore/`

needing to retrieve the entire collection, or any nested subcollections. Add sorting, filtering, and limits to queries or cursors to paginate the results. Realtime listeners can be added to keep data in apps current, without retrieving the entire database each time an update happens. Adding realtime listeners notifies the application with a new data snapshot whenever the data the client applications are listening to changes, retrieving only the new changes.

Access to the data in Cloud Firestore is protected with Firebase Authentication and Cloud Firestore Security Rules for Android, iOS, and JavaScript, or Identity and Access Management (IAM) for server-side languages.

6.4.2 Typescript

TypeScript[9] starts from the same syntax and semantics that millions of JavaScript developers know today. It compiles to clean, simple JavaScript code which runs on any browser, in Node.js, or in any JavaScript engine that supports ECMAScript 3 (or newer). Types enable JavaScript developers to use highly-productive development tools and practices like static checking and code refactoring when developing JavaScript applications.

Types are optional, and type inference allows a few type annotations to make a big difference to the static verification of the code. Types let the developer define interfaces between software components and gain insights into the behavior of existing JavaScript libraries. TypeScript offers support for the latest and evolving JavaScript features, including those from ECMAScript 2015 and future proposals, like async functions and decorators, to help build robust components. These features are available at development time for high-confidence app development, but are compiled into simple JavaScript that targets ECMAScript 3 (or newer) environments. [Typ, 2018]

Typescript was crucial for the backend service of YAPS.life because it allowed the use of the async/await pattern which would not have been available otherwise, due to Google's servers only running Node 6.11.

6.4.3 Firebase cloud functions

Cloud Functions for Firebase[10] is a service that automatically runs backend code in response to events triggered by Firebase features and HTTPS requests. The code is stored in Google's cloud and runs in a managed environment. There's no

[9] `https://www.typescriptlang.org/`
[10] `https://firebase.google.com/docs/functions/`

need to manage and scale other servers for the application. [Fir, 2018]

After deploying a function, Google's servers begin to manage the function immediately. The function can then be fired directly with an HTTP request, or, in the case of background functions, Google's servers will listen for events and run the function when it is triggered. As the load increases or decreases, Google responds by rapidly scaling the number of virtual server instances needed to run the function. Each function runs in isolation, in its own environment with its own configuration.

Using Cloud Functions sped up the development time for the prototype drastically because no server setup or configuration was needed. However, cloud functions are not always suitable for user tasks that require a low response time from the servers because the function might need to do a cold start if it has not been used for a while.

6.5 Dataset

A dataset is a collection of data. Most commonly a dataset contains the contents of a single table from a database, or a statistical matrix of data. Each column of the table represents a particular variable, and each row corresponds to a given member of the dataset in question. Each value in the table is known as a datum. For YAPS.life, four different types of datasets were considered.

1. A dataset consisting of user profiles of real users.
2. A set of randomly generated user profiles.
3. A collection of listing data based on real addresses and properties.
4. A collection of listing data based on generated addresses and data.

Ideally, there would only be datasets based on real data, however in the infancy of an application like YAPS.life that kind of usage data does not exist. Since the user profiles consists mostly of strict, structured data it was decided to generate a set of 10 000 test users with their attributes randomly initialized. The generation code can be found in the appendix listing D.1. A dataset of real user profiles was also constructed when the development of the application had reached the point where users could enter their own information into the system.

As for the listing datasets, mocking the data proved more difficult. Using datasets from AirBnB was considered, but because no dataset provided listings relevant to test users of the application they were not used.

6.6 Recommendation Algorithm Implementation

In this section the implementation of the recommendation algorithm described in chapter 5 will be presented. Each section will be related to an activity from figure 6.2, which depicts the entire activity flow of the recommendation algorithm. Because of the resource intensive parts like the clustering and many connections to external services, the entire algorithm is performed through Cloud Functions, described in 6.4.3.

6.6.1 Context filtering on match location for users

At initiation, the entire user database is searched for users who have indicated that they are ready to be matched using the Query filters provided by Cloud FireStore. Then, iterating over the users, they are filtered into several location buckets like 'Oslo', 'New York', etc. With these buckets of users in place, several asynchronous calls are made starting a separate Cloud Function to cluster each bucket. The clustering then happens in parallel on Google's servers.

6.6.2 Clustering and match creation

Before any clustering happens, the amount of users to be matched are checked to be larger than 16 as that is the minimum amount of users needed to provide several matches with strong cohesion. Then, the users' personality vectors, from the habit questions, and their property vectors, from the property goals, are extracted and combined into a a list of single vectors.

This list, where each index in the list corresponds to the user in the user list, is sent into the clustering function. This function can be swapped easily as long as it accepts the list of vectors and returns a list of clusters, where a cluster is a list of indexes that link back to the original user list. If the clusters contain more than four users they are split into smaller fragments either using a simple modulo function or a more advanced k Nearest Neighbour(kNN) approach.

In the production system today, the hybrid approach described in chapter 5 is used. First, the users from the match location are piped into the k-Means clustering function where k is determined by the log_2 of the number of users to be matched. This produces a set of initial clusters which are then split up further by the kNN algorithm. The result is a nested list structure that is flattend into a single list of groups. Then, this list is mapped asynchronously to create a match object in the database from each group.

The match object consists of several different properties. Upon creation, the

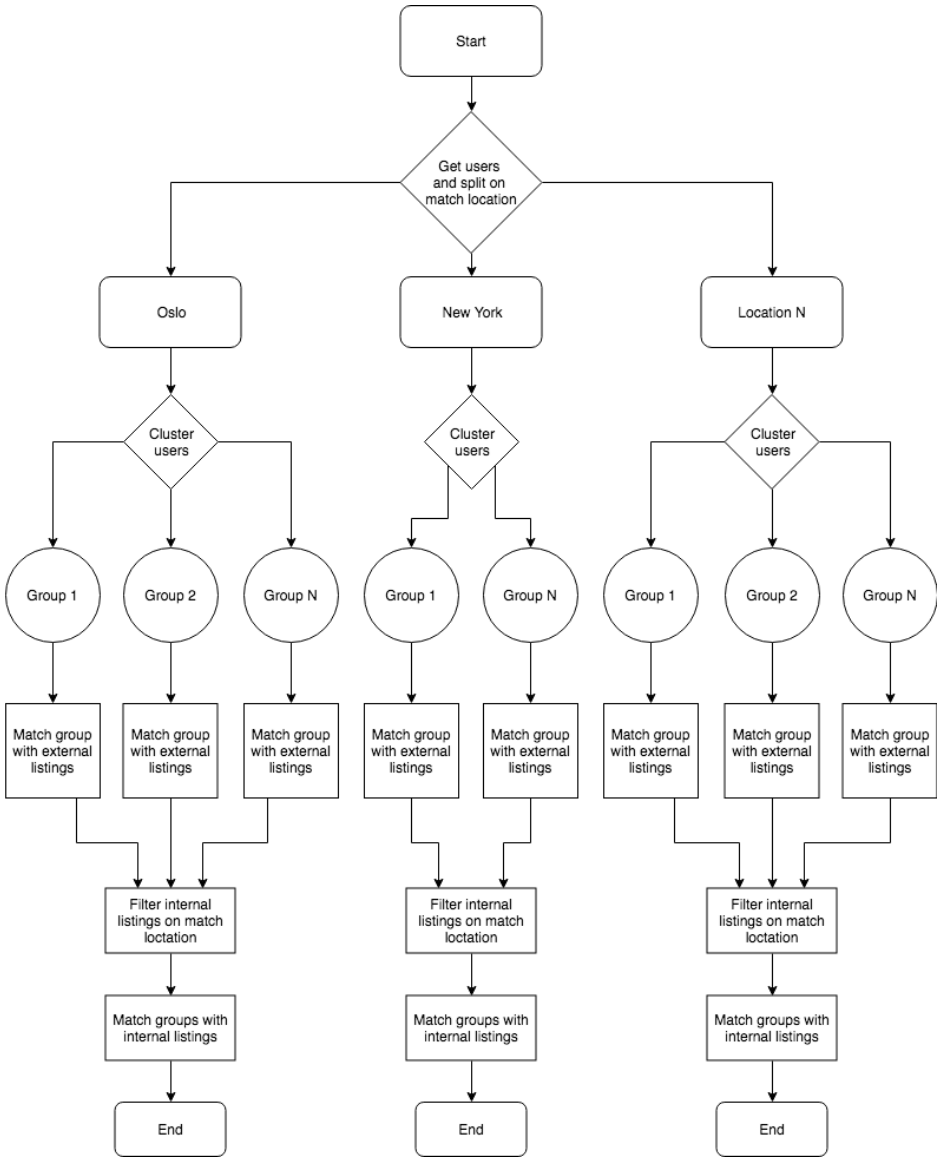


Figure 6.2: Recommendation algorithm, activity diagram

individual property vectors of the group members are combined into a single group property vector by averaging the columns. A personality alignment and property alignment is also calculated for the group based on the average distance between all personality and property vectors in the group. An initial "Best origin" is set to the match location and a nested object that will contain properties linked to the match is constructed. Finally, the object is initialized with a timestamp of when it is created.

To facilitate communication between the group members a chat room is also initialized. The messages are stored within the match document as a sub-collection in the FireStore database. The last step is iterating over the group members and updating their documents with the matchId they are now linked to.

6.6.3 Matching users with external listings

When a match is created in the database, another Cloud Function called "onMatchCreate" is triggered. The task of this function is twofold. The first is to update the match with the optimal area to start apartment hunting in the match city. The second is to consult several data sources like Finn.no, AirBnB, and the service's own listings, and match the group with listings that suit the group as a whole.

Get best origin

To get the best origin the function constructs a search query to the Google Distance Matrix API based on the users' workplaces and pre-selected districts in the match city. The result from this query is a $Nr\_of\_workpaces * Nr\_of\_districts$ matrix that details the distance and travel time between the different origins and destinations. This matrix is then passed to another function which returns the optimal origin for the group. The next step is to use this best origin, together with how many people are in the match, to create one search URL query for Finn.no and another for AirBnB.

Another point worth mentioning is the input to the Distance Matrix API. Upon profile creation, the users input the address of their workplace. While the address itself could technically be used the as input, it is prone to error because of formatting issues. A more stable way was discovered by Geolocating the address into longitude and latitude points, using another Google API [11], and then using that as input instead.

[11] `https://developers.google.com/maps/documentation/geolocation/intro`

Match with external listings

The two URLs and the best origin are then returned in the updated match object and used to gather two sets of listings from Finn.no. The first set is a maximum of ten listings from the specialized query. The other is a maximum of twenty listings gathered from a broad search of the match location. After merging these sets of URLs to filter for duplicates the detail of each listing is collected. In these details the algorithm parses the apartment into a property vector similar as the one possessed by the group.

- The Budget for the apartment is determined from the price per moth divided by the number of people in the group.
- The Size is determined by the size of the apartment.
- The Standard is not possible to parse as Finn has no attribute describing it, therefore it defaults to 0.
- The Style is set to a default 0 , unless the listing has an attribute describing it as modern in which case it is set to 2.

The next step is to give these listings an initial group score for how well they fit the group's preferences. This is done by taking the Euclidean distance from the group's property vector to the property vector of the listing. Because budget and size are easier to define and based on the data in the listing, and has more impact on whether the apartment is interesting to the group or not, these attributes are weighted higher. The reasoning is that a group would rather consider an apartment that fits with the budget and size constraint, and a different standard and style, than the other way around. The listings are then sorted based on this initial score, and the top six progress further.

These six apartments are then evaluated for average commuter time to the group member's workplaces. The reason for the initial screening is the API limits imposed by the Google Distance Matrix API. If all listings were to be evaluated for commuter time, it would not be possible to keep the service free. The commuter time, which is given in seconds, is then projected to the scale between -2 and 2 like the other attributes and concatenated into the listing's property vector.

The last step is to get a final group score for the listing, which is produced with another Euclidean distance from the group's property vector to the now enhanced property vector of the listing. The weights are adjusted slightly to give more influence to the commute time. The weights used in the production prototype can be found in figure 6.3.

Figure 6.3: Weights on the property vector distance calculation. CT: Commute Time, B: Budget, S: Size, Std: Standard, Sty: Style.

$$\begin{array}{r} attribute = \\ initial\ group\ score\ weights = \\ final\ group\ score\ weights = \end{array} \begin{Bmatrix} CT & B & S & Std & Sty \\ & 2 & 2 & 0.5 & 0.5 \\ 2 & 2 & 1 & 0.5 & 0.5 \end{Bmatrix}$$

If the group score for the listing is still within a reasonable distance of the group's property vector the final step is to add the listing to the match object so the group can view it in the flat list as seen in section 4.6.2. External listings are saved directly to the match object.

6.6.4 Match with internal listings

After external listings have been evaluated and injected into the match a similar process happens for internal listings. The Cloud FireStore database is queried for listings where the location of the listing matches the match location. All relevant listings are then evaluated for average commuter time and go through the same distance check to the group's property vector. If it is a good match, the listing is injected into the match, instead of saving all the listing data in the match, only the Id of the listing is injected. This way, if the listing is updated it will show in all matches the listing has been injected to. Another key difference is that a chat dialog between the group and the listing administrator is initialized. The listing document stores all chat information with all matches in its own document as sub-collections. The entire process can be viewed as a sequence in the sequence diagram in figure 6.4.

6.6.5 Handling new users, matches, and listings

When enough new users register for the YAPS.life service and finish their profiles, a new matching of users and listings will be triggered. Listings that are uploaded by landlords after a matching has already happened will be added to all relevant matches retroactively. While not implemented in this prototype, the planned functionality is to either run a matching every third day, or do periodic analysis of the available users and listings to see if good matches can be made. New iterations of the matching algorithm could potentially also be used to inject new users into old matches where some group members left the match or remained inactive.

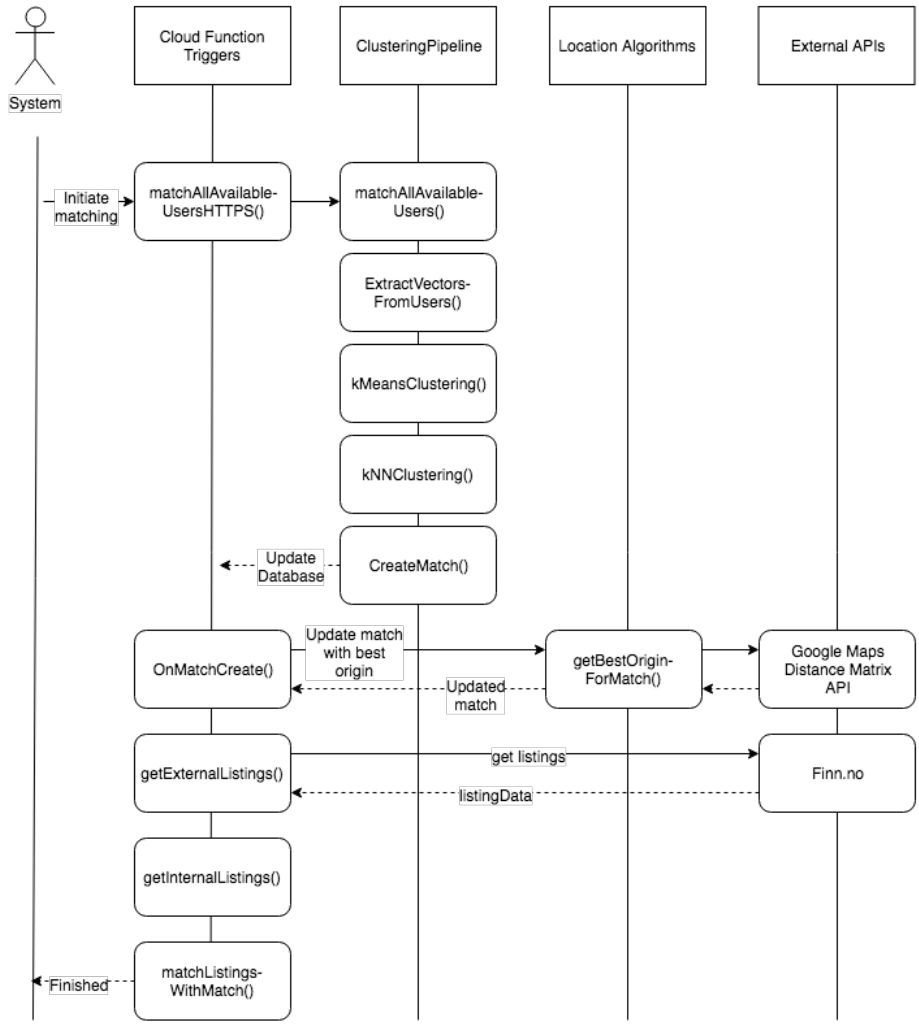


Figure 6.4: Recommendation algorithm, sequence diagram

Chapter 7

Evaluation

In this chapter, the results of this project will be evaluated against the research goals. There will also be a discussion of the strengths and limitations of the results.

Goal *Develop and evaluate a platform for group recommendation of real estate.*

After several months of development, a prototype service for group recommendation of real estate was finalized. It was given a soft release to the public through several social media channels, and at the time of writing has registered 73 user accounts. The development did take more time than anticipated and not all of the features planned were implemented. Still, the end product is a fully functioning platform that provides value to real users, today. It can be found at www.yaps.life.

The evaluation of the platform is presented in a review of the research questions.

1. First, a quantitative analysis of the performance of the different clustering techniques.
2. Then, a usability evaluation using the System Usability Scale test to see what users thought of the prototype.
3. Finally, a qualitative test measuring how well the system was able to perform actual real estate recommendations to groups.

7.1 Matching Evaluation Results

Research Question 1 *What is the best way to match users looking for shared accommodation into groups?*

The literature on group recommendation shows that using clustering techniques is the state of the art in order to detect groups. Boratto and Carta [2010] indicated that k-Means was used widely; however, there is no technique that always outperforms any other, so several had to be investigated. In this thesis the following six different algorithms were evaluated:

- **Baseline**, simply divide users into groups of four without any clustering.
- **kNN with Euclidean distance**, go through the available users linearly, and for each user extract his top matches. Users are sorted with the Euclidean distance.
- **kNN with Cosine Distance**, same as above, but users are sorted with the Cosine distance.
- **k-Means**, divide the users into k clusters, where k is the log_2 of the number of users to be matched. Divide these clusters into four without further clustering.
- **Hybrid of k-Means and kNN with Euclidean distance**, after creating initial clusters with the k-Means algorithm, the clusters will be split using kNN with Euclidean distance.
- **Hybrid of k-Means and kNN with Cosine distance**, same as above, but using kNN with Cosine distance.

The algorithms have been evaluated on two different datasets. The first is a set of 38 user profiles collected from users of the prototype with their consent. While not very large, this size of users to cluster is quite probable for the early stage of the YAPS.life system when few users in each match location have registered. In order to see how the system scales, a second data set of randomly generated test users was constructed. The test users mimic real users by randomly setting their attributes from the same selection of attributes that real users have available. A full test set of 10 000 test users was constructed, and then smaller sets of 40, 100, and 1000 were created from that base.

7.1.1 Brute-force search

Before comparing the algorithms, a lower and upper bound for how good a match could be was established using a brute-force search through the solution space. Setting the group size to four, the amount of matches generated for the 38 real users is the same as the combinatorial of $\binom{38}{4}$, which is 73 815. As can be seen in figure 7.1 the worst possible matches had a score that hovered around 1.2 and the best managed to reach 0. This score is called the Combined Alignment of the match and is the average Cosine distance between all user vectors in the match. See 3.2.1 for further details. The fact that a Combined Alignment of 0 is possible indicates that a perfect match can be found. The median match is also tracked for reference.

The same brute-force search on the test data can be found in figure 7.2. All series perform worse here, i.e., have a higher combined alignment, thus indicating the test data does not reflect the real data perfectly. Still, it is similar enough that further testing on the data was considered worthwhile. The largest brute force search possible on the available hardware was with 64 users, resulting in a total of 635 376 matches. Since it revealed no new trends in the graph it is included in the Appendix B.1, instead of here.

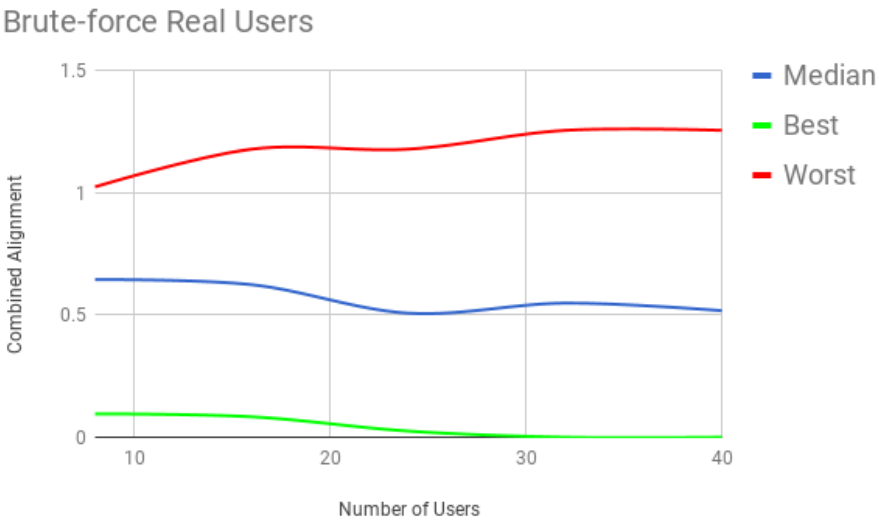


Figure 7.1: Brute-force Real Users

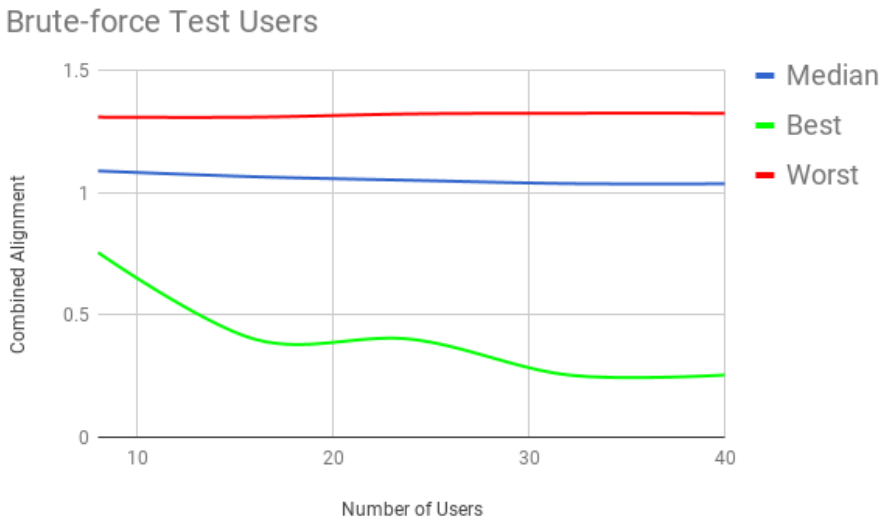


Figure 7.2: Brute-force Test Users

7.1.2 Match distribution

With the theoretical boundaries established we can better evaluate the different clustering algorithms. Instead of showing how the algorithms do as the user size increases we instead use snapshots of different user sizes and chart how the distribution of the matches produced by the different algorithms compare. Some algorithms produce more matches than others because of how they divide their users.

Figure 7.3 shows the clustering of the 38 real users. The baseline algorithm does surprisingly well with three relatively good matches; however, overall it performs worse than the other algorithms as expected. Both kNN Euclidean, just kNN in the figure, and kNN Cosine performs well with the first 7 matches, but actually worse than the baseline in the last ones. This is due to the effect described in 5.5.2 where the last users the algorithm evaluates have very few alternatives to be matched with. The kMeans and the two hybrid algorithms produce more matches than the others and achieve a lower distribution score with the Hybrid Cosine algorithm providing the best overall score.

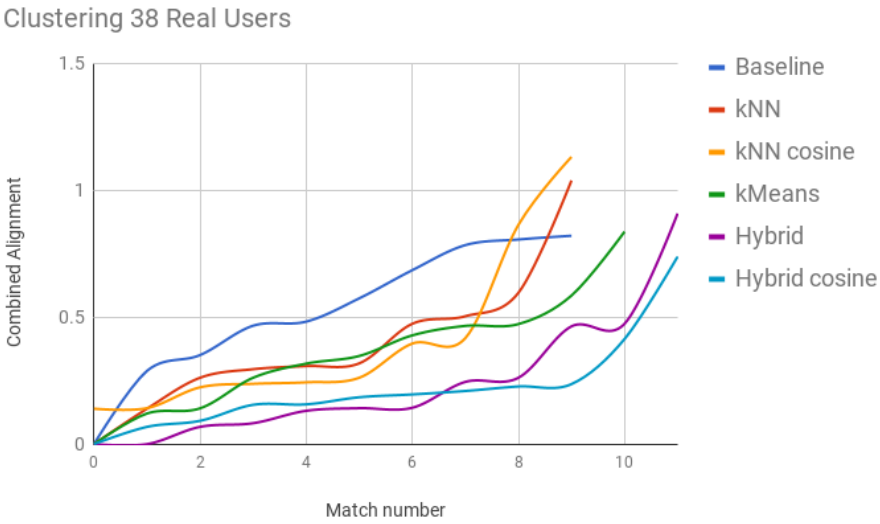


Figure 7.3: Clustering 38 real users

Moving on to roughly the same amount of test users we see all algorithms performing quite a lot worse. This is to be expected as the theoretical max and min were both shifted upwards in the test data set as described in the section above. It is still a tight race between the different algorithms, but once again the Hybrid Cosine methods wins out.

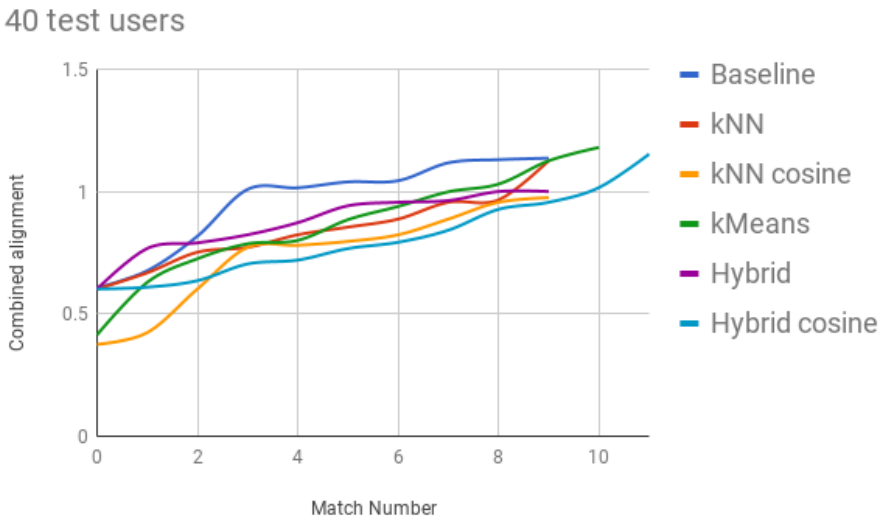


Figure 7.4: Clustering 40 Test Users

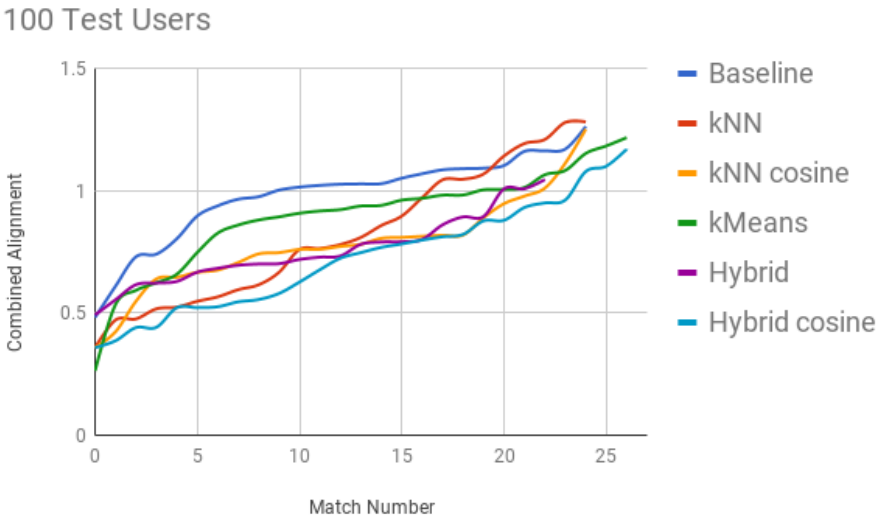


Figure 7.5: Clustering 100 Test Users

In the next figure, 7.5, the amount of test users has been increased to 100, and we see the number of matches has increased to around 25. There is a little more space between the different series now with Baseline and k-Means performing noticeably worse than the others.

When the number of test users is increased tenfold, up to 1000, in figure 7.6, a more explicit pattern emerges. Baseline and k-Means are left behind with the other algorithms performing much better. This trend is further exacerbated in the last figure, 7.7 where 10 000 test users are compared.

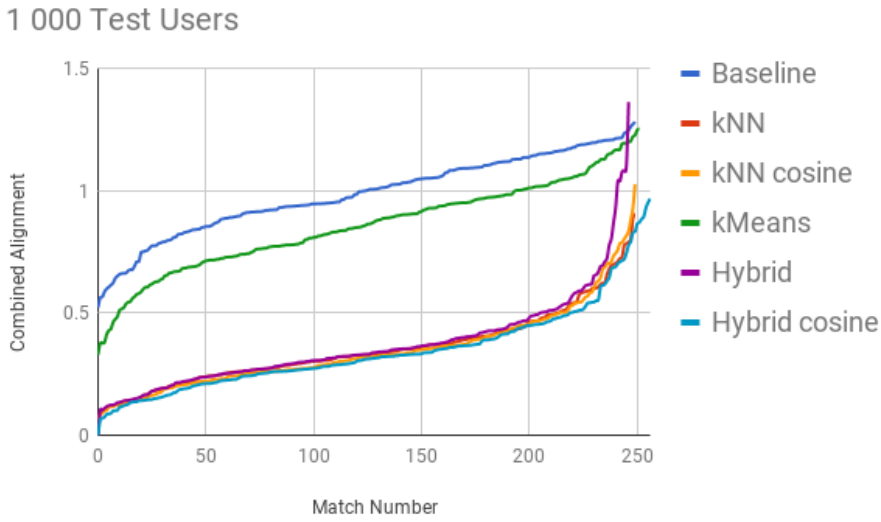


Figure 7.6: Clustering 1 000 Test Users

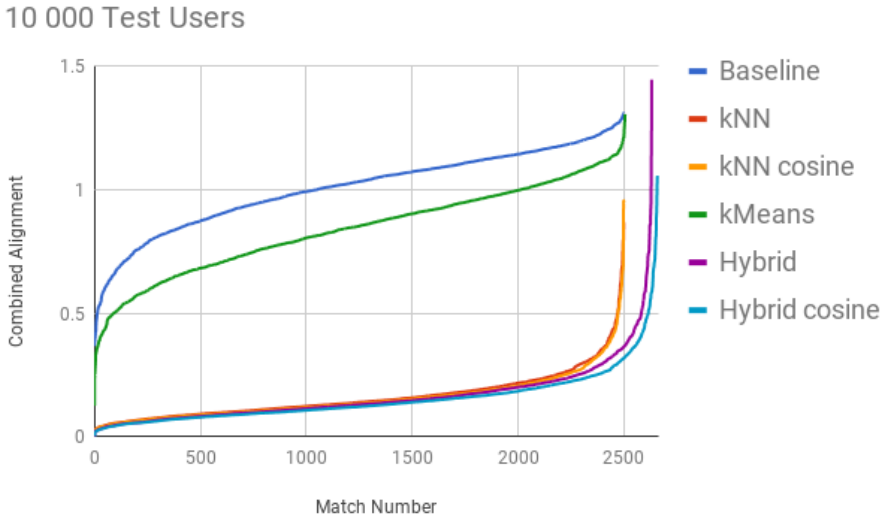


Figure 7.7: Clustering 10 000 Test Users

7.2 Usability Evaluation Results

Research question 2 *How to present a group recommendation system for real estate?*

In order to evaluate the usability of the prototype, 16 users submitted to follow the steps outlined in section 3.3.2. Briefly summarized they were to:

1. Fill out the Background Information(BI) survey.
2. Perform several tasks with the application outlined in User Tasks(UT).
3. Answer the System Usability Scale(SUS) survey.
4. Complete an Application Specific(AS) survey.

7.2.1 Number of test subjects

Nielsen [1994] indicates that only 5 users are enough to find most usability problems in a system, up to about 80%. In order to achieve this high number though the tests should be small, targeted and based on multiple iterations. Faulkner [2003] suggests that the percentage of errors found are highly dependent on the types of users participating in the user test. The range of errors found go all the way from 99% to only 55% depending on the test case. As Faulkner expanded the number of test subjects to 10, the minimum percentage of problems found were 82%, and at 15 it was 90%. This indicates that in general one should have as many test subjects as possible to uncover even more problems, but that having around 15 is usually enough.

7.2.2 Demographic and background information results

In table 7.1 a brief summary of the demographics of the test subjects are presented. All test subjects were students, with most of them in their final year as they are the target users for this application. The gender balance is biased towards male with 87.5%, and only 12.5% females. This is regrettable as the end application targets both sexes equally. However, due to the applications nature it will probably have more early adopters who are men. As for the age of the test subjects, it ranged from 19 to 27, perfect for the target group.

After the demographic questions came several others detailing users' past experience with searching for new apartments, and how many times they had lived in a shared accommodation. Further questions regarded how many people lived

Table 7.1: Demographic information from the Background Information(BI) survey

Question	Alternatives	Responses	%
Gender	Male	14	87.5
	Female	2	12.5
Age	18	0	0
	19	1	6.25
	20	1	6.25
	21	0	0
	22	4	25
	23	1	6.25
	24	6	37.5
	25	1	6.25
	26	0	0
	27	1	6.25
Occupation	Student	8	100
	Young Professional	0	0
Nationality	Norwegian	16	100

in those apartments and what number of housemates the user considered ideal. These results can be found in table 7.2 and the full survey can be found in appendix.

In summary, we see that all of the test subjects have searched for a new apartment more than once, and all subjects have lived in a shared accommodation at one point. Roughly 50% having lived in shared accommodation multiple times. The test users had between them lived in shared accommodations ranging in size from just 2 to more than 8, with most of them around 3,4, and 5. 3 emerged as the clear winner for optimal shared accommodation size with 57% favoring it.

Table 7.2: Other results from the Background Information(BI) survey

Question	Alternatives	Responses	as %
How many times have you searched for a new place to live, i.e switched apartment?	0	0	0
	1	0	0
	2	1	6.25
	3	5	31.25
	4	6	37.5
	5	4	25
How many times have you lived in a shared accommodation?	0	0	0
	1	4	25
	2	5	31.25
	3	3	18.75
	4	4	25
How many people lived in these share houses?	2	2	12.5
	3	9	56.25
	4	7	43.75
	5	3	18.75
	6	3	18.75
	7	2	6.25
	8	3	18.75
	More	3	18.75
What size of share house were you the most happy with?	2	4	25
	3	6	37.5
	4	4	25
	5	2	12.5

7.2.3 System Usability Scale results

Two tables regarding the SUS score has been produced. Table 7.3 shows the distribution of the answers for the different questions while table 7.4 shows the average score per question as well as the calculated final SUS score for the prototype.

Table 7.3: SUS results

	Strongly disagree		Disagree		Undecided		Agree		Strongly agree	
Question	N	%	N	%	N	%	N	%	N	%
SUS 1	1	6.25	1	6.25	7	43.75	5	31.25	2	12.5
SUS 2	4	25	9	56.25	3	18.75	0	0	0	0
SUS 3	0	0	0	0	3	18.75	9	56.25	4	25
SUS 4	14	87.5	2	12.5	0	0	0	0	0	0
SUS 5	0	0	0	0	6	37.5	8	50	2	12.5
SUS 6	6	37.5	9	56.25	1	6.25	0	0	0	0
SUS 7	0	0	0	0	2	12.5	10	62.5	4	25
SUS 8	5	32.25	6	37.5	3	18.75	2	12.5	0	0
SUS 9	0	0	0	0	4	25	7	43.75	5	31.25
SUS 10	12	75	4	25	0	0	0	0	0	0

By following the guidelines from section 3.2.2 the final SUS score for the prototype was 78.57. On the adjective scale outlined in the same section that equates to somewhere between *Good* and *Excellent*. Of course, there are no absolutes when it comes to usability testing, but this rating indicates that the prototype is on the right track when it comes to design and interactivity.

Table 7.4: SUS Average scores

Question	Average SUS score
SUS1: I think that I would like to use this system frequently	3.375
SUS2: I found the system unnecessarily complex	1.9375
SUS3: I thought the system was easy to use	4.0625
SUS4: I think that I would need the support of a technical person to be able to use this system	1.125
SUS5: I found the various functions in this system were well integrated	3.75
SUS6: I thought there was too much inconsistency in this system	1.6875
SUS7: I would imagine that most people would learn to use this system very quickly	4.125
SUS8: I found the system very cumbersome to use	2.125
SUS9: I felt very confident using the system	4.0625
SUS10: I need to learn a lot of things before I could get going with this system	1.25
Final SUS score	78.125

7.2.4 Application Specific survey

After filling out the SUS survey, the test subjects were asked to complete one final form, the Application Specific survey. In this questionnaire, five specific questions about the prototype were asked, as opposed to the SUS which is more general. Four questions had an answer range between 1 and 5, where 1 is strongly disagree, and 5 is strongly agree, and the last question was a simple yes or no. The questions and answers can be found in table 7.5.

Table 7.5: Application Specific survey results

Question	Alternatives	Responses	as %
I believe I could get use of this application in the future.	1	1	6.25
	2	0	0
	3	5	31.25
	4	6	37.5
	5	4	25
The application provided recommendations relevant for me.	1	1	6.25
	2	0	0
	3	2	12.5
	4	6	37.5
	5	7	43.75
I found it helpful to use distance to my workplace to find relevant apartments.	1	1	6.25
	2	0	0
	3	1	6.25
	4	9	56.25
	5	5	31.25
Would you like to have a large amount of matches to choose from?	Yes	10	62.5
	No	6	37.5

From the results we see that over 60% of the respondents indicated strongly that they could have use of such an application in the future. Further, 80% said the recommendations of roommates were relevant for them. The other two questions were geared more towards the future work for the application. 87% found using

the commute time to their workplace very relevant to rank apartments, and 62.5% wanted more than one match to choose from when it came to selecting groups.

7.2.5 Usability problems & Feature suggestions

Through the Usability test and Application Specific(AS) survey, several free text answers were collected regarding usability problems and potential new features for the prototype. The usability problems were fixed during the second period of development, but the feature suggestions were left for future work. The most relevant feedback is listed below.

Usability problems

- "Question text too close to the slider"
- "Full text on the user cards on the match page should be a bulleted list instead."

Feature suggestions

- "A search function"
- "Maps showing travel from potential apartments to work"
- "View what internships people have. See mutual friends on Facebook and LinkedIn."(Paraphrased from Norwegian)
- "Feedback from earlier roommates (rating system?)"
- "Other cities"
- "Would be nice to see a map of Oslo with the workplaces of the group members along with apartment locations."
- "Heat map overlay for Google Maps showing the potential for flatmates in one color and the potential available apartments in one color"
- "A video meeting function"

7.3 Quality of Real Estate Recommendations Results

Research question 3 *How do we provide accurate group recommendation of real estate?*

The last research question pertains to how well the system managed to recommend real estate to groups of users. To determine the quality of the recommendation another user test was performed, but instead of gauging the general usability of the application this one focused solely on the quality of the recommendations provided. One group of four, another of three, and two groups of two were asked to perform the survey.

They were asked to create a profile and answer all property and personality questions, and then create a new solo match. When the matching procedure was finished, the test subjects were asked to review the ranking of the proposed listings according to the preferences they had indicated. Then, one of the group members would invite the rest to their match so that they could rate the new recommendations for the group. As usual, 1 indicates very dissatisfied, and 5 indicates very satisfied. Figure 7.8 and 7.9 show the results of the test.

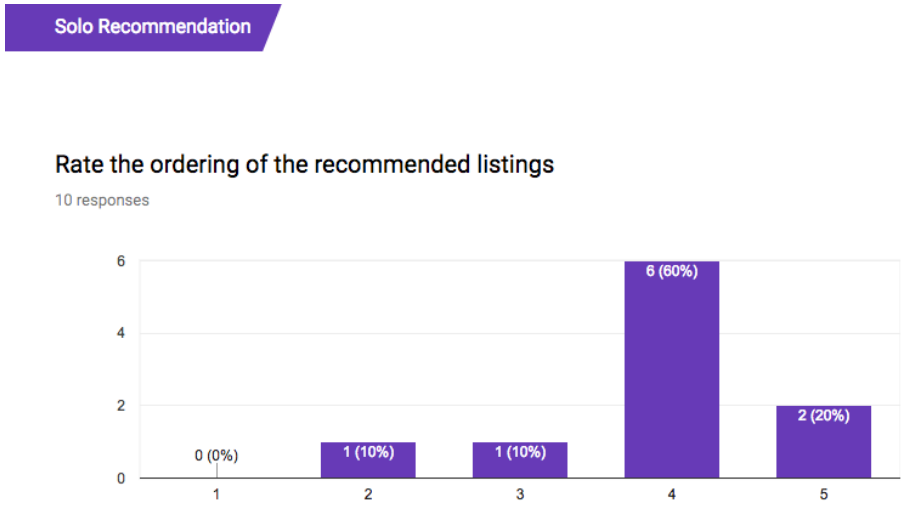


Figure 7.8: Solo Recommendation quality rating for listings

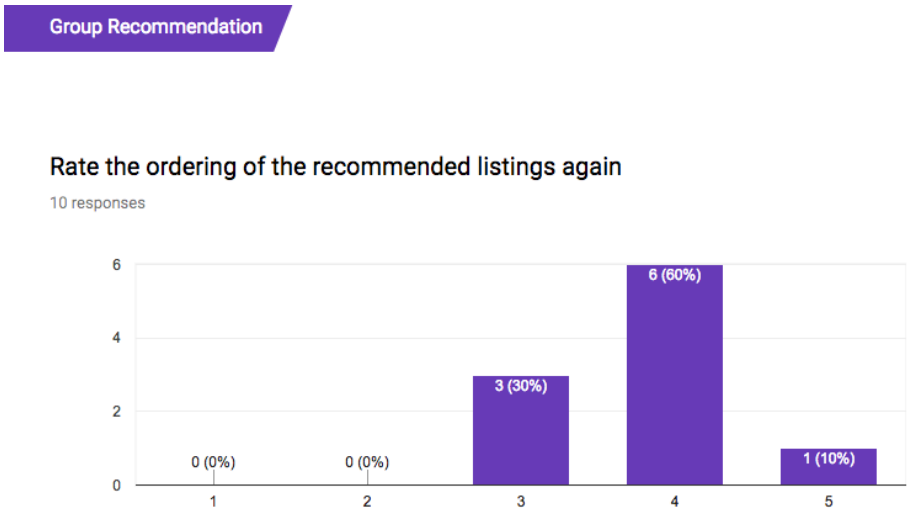


Figure 7.9: Group Recommendation quality rating for listings

The YAPS.life method of searching for housing compared to traditional methods

The test subjects were also asked to rate how they viewed this method of searching for housing compared other traditional methods they had used before, like Finn.no. The results indicate the YAPS.life way to be preferable to traditional methods. However, due to the low sample size of only ten people, no conclusions can be made. The results can be viewed in figures 7.10 and 7.11.

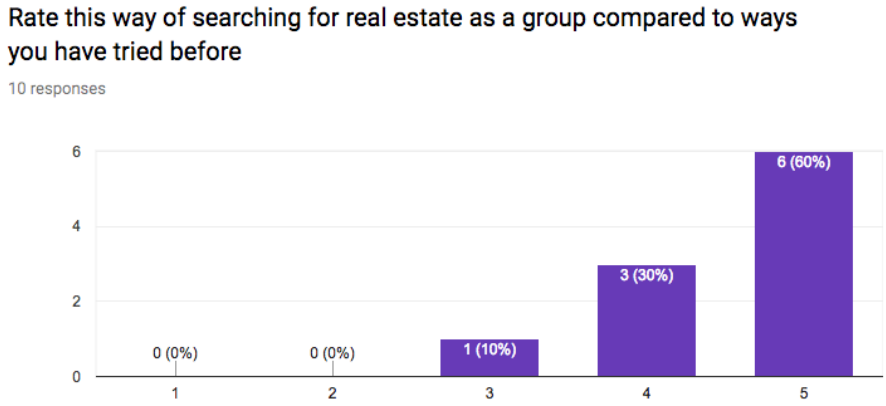


Figure 7.10: The YAPS.life search method for searching alone

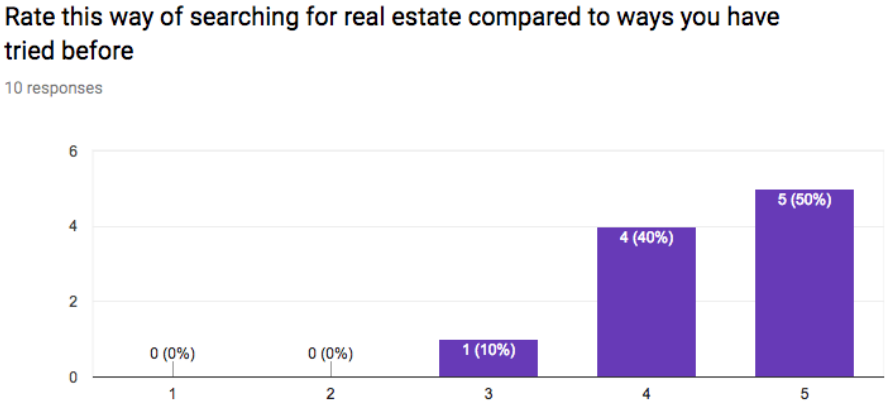


Figure 7.11: The YAPS.life search method for for searching as a group

7.4 Discussion

The results from these three evaluations indicate that the prototype for YAPS.life provides value to users already today. The clustering algorithms manage to provide better matches than simply splitting the users into random groups, the usability of the system is adequate, and the quality of the recommendations appear to be satisfactory.

7.4.1 Quantitative analysis of clustering methods

For the quantitative tests on the clustering algorithms for test users, a few observations can be made. Overall we can see that the scores are still quite high, both in the median and the best case. This is due to the test users in the dataset all have their personality and property vector initialized to random values between -2 and 2. With 24 indices in total, finding users with perfect alignment is very improbable unless millions of test users are created, which is an unrealistic amount for this service. This random initialization is also likely to undermine k-Means. Real users will probably not have such wildly different, and sometimes contradictory, preferences. This should lead to more distinct clusters which will likely improve k-Means a lot.

In the end, the Hybrid Cosine algorithm was chosen to be the main clustering algorithm for the prototype based on the results it produced. The fact that it created more matches than the rest is not at all a bad thing. Rather, it means that there will be a more even distribution in match sizes, ranging from 2 to 5, which means fewer of the groups will compete for the same apartments.

7.4.2 Usability evaluation

From the results of the Background Information survey, we see that all participants had a high degree of familiarity with the problem this prototype tries to solve, thus making them excellent test subjects. One of the more interesting findings from the Application Specific survey was that many users wanted several matches to choose from. This could give users the feeling of choosing their flatmates more autonomously. However, it could also lead to many matches being filled with users who had already decided to go with a different group. Technically, it could be implemented using a kNN All-to-All approach, where every single user got their top K flatmates picked for them. This would lead to some users being matched into many groups, and some, if they answered with many extreme values, only be matched into a few. Another round of usability evaluations would have to be conducted with the feature implemented in order to see whether this would be the right way to go forward.

7.4.3 Quality of recommendation evaluation

As for the quality of the real estate recommendations, the results are quite good, at least quantitatively. The ratings given relate to the overall ranking of the displayed listings according to the user's preferences. However, these preferences do not impact the amount of listings available from different data sources. Relying on external data sources is inherently difficult for any service as there is little control over the information at the data source. More accurate listing matches can be created from properties uploaded through the YAPS.life platform because more structured information about the property will be available.

Because YAPS.life is a knowledge-based recommendation system, the cold start problem is avoided to a large degree. This means the system does not need to have many users before it can provide useful recommendations. However, this also means that the system loses out on a lot of the potential learning content-based and collaborative filtering systems get, and will not improve over time.

The main reason for going with the knowledge-based approach at the beginning of the project was because of the perceived difficulty of getting relevant feedback on whether a listing was relevant or not. The most relevant listing is the one that leads you to the property you ultimately rent or buy, but integrating the whole real estate renting process to capture this information would be too big for the scope of this masters project. It is however, slated for future work for the platform. A more relaxed way of getting feedback could be to rely on the implicit data gathered through observing which listings users interacted with. The problem with that approach is that you would need the user activity of hundreds, if not thousands of users to achieve good results. Li et al. [2017a] after all had the most popular real estate in Japan with thousands of daily visitors to mine data from.

7.4.4 Threat to validity

Even though the results seem promising, it is important to keep in mind how they can have been skewed by the circumstances around the experiments. Starting with the quantitative evaluation we see that the sample size for the experiment with real users were 38. While this is certainly better than having 0 real users, it is still not a very large number. For the system to be able to generate truly good matches, a higher number of participants are likely needed, unless everyone signing up are very like-minded. Speaking of like-mindedness, it is also quite possible that the early adopters who did sign up have quite similar personalities. This could indicate that other ways of defining one's personality and habits should be investigated in order to more clearly separate users. Either way, 38 users does

probably not represent the wide specter of personalities that exist in the potential user base for the application.

The usability test ended up with 16 participants. According to Nielsen [1994], this is supposedly enough to find all major and most minor flaws with the system. However, it is hardly enough to build up any statistical significance that the user interface is understandable and easily navigable by everyone. The test subjects were uniformly students and the vast majority were men. To some degree this represents the intended target group for the application quite well, but it is not a desirable distribution of test users for an unbiased experiment. Another way to heighten the validity of the usability test results would have been to perform it at multiple iterations as the prototype was being developed.

The Quality of Real Estate Recommendation survey had fewer users than the usability test, down from 16 to 10. A higher number of test participants could potentially change the distributions of the answers from very positive to something more like a normal distribution. Having a more diverse user base here would also have given a stronger indication that the system is actually providing good, *general* real estate recommendations that are not skewed towards the young professional target group. Another aspect of the real estate recommendation that will have a large impact on the system in the future is the available data sources for listings for each city. For the user tests in this master thesis the focus was on moving to Oslo, so therefore Finn.no, which is the largest real estate broker in Norway, was chosen to be the main data source. Integrating with other real estate brokers around the world will have a large impact on the user's perceptions of the real estate recommendations.

Still, the results so far are promising and they do give an indication of the performance and usability of the prototype. In the end, the best way to further collect and analyze data related to the application is not from usability tests and quality evaluations, but from usage data of the website. How users engage with recommendations and how they navigate the user interface will reveal more accurate and unbiased results than any user test could hope to emulate.

Chapter 8

Future work and conclusion

In this final chapter, the contributions of the thesis will be presented along with the plans for future work. Finally, there will be a conclusion.

8.1 Contributions

Presented in this thesis is the first ever group recommendation system for real estate. The system is a computer-based product which combines state of the art algorithms from group recommendations and recommendation in real estate in a new way. It is a strong foundation for future research into the area to build on with new combinations of algorithms.

Within this contribution, lies the user interface of the prototype, the design of the recommendation algorithm as well as the technical architecture. The system has also been evaluated for usability and quality of recommendations in a rigorous manner, which can serve as guidelines on how to evaluate similar systems in the future. The way it ranks listings is novel compared to traditional real estate search engines. As opposed to just using price, publication date or another tangible asset of the property, the prototype instead uses a mix of the user's, or the users', preferences for budget, size, standard and style. In addition, the average commuter time to the users' workplaces from the apartment is calculated and mixed into the overall score.

8.2 Future Work

While much has already been accomplished with this prototype, many more avenues of research have presented themselves during development. The hardest problem with recommendations in real estate is the lack of feedback the system receives on whether a house has been rented or not. However, as end to end systems such as this prototype become more commonplace, we can see tenants and landlords sign digital leases on the platform. This gives a much stronger feedback than what is available on platforms today, with much of the renting process happening off platform. With this new feedback, avenues into content-based and collaborative filtering can be made. By looking at the demographic and socio-economic status of people renting and buying a very strong collaborative filtering system could be constructed.

A more short-term goal could be to implement user ratings on the recommendations provided by the system to help it train. Multiple ways of implementing such a rating system should be explored. One way would be to have the user use a traditional star-rating scale on each individual recommendation, but newer approaches like a swiping model, as seen the app Tinder[1], could also be used to train the system.

One unsolved problem with how the prototype determines the property vector values of external listings is that it is rule-based and absolute. By introducing fuzzy logic membership functions, a more continuous approach to determining the property vectors of listings could be established.

Regarding group recommendations, a more advanced strategy than pure averaging could be tried. Least misery could potentially hold some promise by eliminating houses that are below an acceptance threshold for some people in the group. Another strategy could be to do recommendations based on multiple approaches and showing all of them to the user in a hybrid mixed (section 2.2.4) approach.

Besides the lack of feedback, the second largest challenge of real estate recommendation today is the amount of information about a listing that is stored in an unstructured format, e.g. a listing text. By constructing a Natural Language Processing(NLP) pipeline to process these texts and add them as features for the listing could be a valuable asset for the recommendation system, as these texts often have a high impact on the prospective renter.

[1] `tinder.com`

8.3 Conclusion

In this thesis, a prototype for a Group Recommendation System of Real Estate has been presented. The overall goal of this research was to:

Goal *Develop and evaluate a platform for group recommendation of real estate.*

This goal led to three research questions being investigated and evaluated.

Research question 1 *What is the best way to match users looking for shared accommodation into groups?*

To find the best way to match users, six clustering algorithms were implemented and evaluated. From the rigorous quantitative analysis, where the match distributions of the algorithms were compared side by side, the best algorithm for the job turned out to be the Hybrid Cosine algorithm. This is a Hybrid approach that makes an initial clustering with k-Means, and then splitting these into smaller matches using a modified kNN algorithm.

Research question 2 *How to present a group recommendation system for real estate?*

Instead of trying to determine the unequivocally best user interface for a real estate recommendation system for groups, this thesis presents a proposed design that is then evaluated by the System Usability Scale test. Achieving a SUS score of 78, the prototype is still in need of further development and refinement. However, this score in the high seventies indicates that the development of the interface is headed in the right direction. Comments gathered from user testing also revealed new potential features.

Research question 3 *How do we provide accurate group recommendation of real estate?*

Because of the systems knowledge-based nature, a user-centric way of evaluating the recommendations was chosen. Although the number of test participants could have been higher, the results show a clear positive trend towards satisfaction with the recommendations and this way of doing housing search over traditional methods.

Based on the results of these three evaluations it would seem the YAPS.life platform could be a great starting point for further research in the area of group recommendation of real estate. After all, the problem is not going away any time soon, and further research into this area could provide immense value to people all over the world.

Bibliography

(2017). MovieLens datasets of movies and ratings. `https://grouplens.org/datasets/movielens/`. Accessed: 2017-12-07.

(2018). Cloud functions for firebase. `https://firebase.google.com/docs/functions/`. Accessed: 2018-05-27.

(2018). Create react app. https://github.com/facebook/create-react-app. Accessed: 2018-05-27.

(2018). Jest, testing framework. https://facebook.github.io/jest/. Accessed: 2018-05-27.

(2018). Typescript. `https://www.typescriptlang.org/`. Accessed: 2018-05-27.

Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749.

Amatriain, X., Pujol, J. M., and Oliver, N. (2009). I like it... i like it not: Evaluating user ratings noise in recommender systems. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 247–258. Springer.

Ardissono, L., Goy, A., Petrone, G., Segnan, M., and Torasso, P. (2003). Intrigue: personalized recommendation of tourist attractions for desktop and hand held devices. *Applied Artificial Intelligence*, 17(8-9):687–714.

Baltrunas, L., Makcinskas, T., and Ricci, F. (2010). Group recommendations with rank aggregation and collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 119–126. ACM.

Bangor, A., Kortum, P., and Miller, J. (2009). Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123.

Bazire, M. and Brézillon, P. (2005). Understanding context before using it. In *International and Interdisciplinary Conference on Modeling and Using Context*, pages 29–40. Springer.

Bellman, R. (2013). *Dynamic programming*. Courier Corporation.

Bennett, J., Lanning, S., et al. (2007). The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA.

Beyer, K., Goldstein, J., Ramakrishnan, R., and Shaft, U. (1999). When is "nearest neighbor" meaningful? In *International conference on database theory*, pages 217–235. Springer.

Bond, M., Seiler, M., Seiler, V., and Blake, B. (2000). Uses of websites for effective real estate marketing. *Journal of Real Estate Portfolio Management*, 6(2):203–211.

Boratto, L. and Carta, S. (2010). State-of-the-art in group recommendation and new approaches for automatic identification of groups. *Studies in Computational Intelligence*, 324:1–20. cited By 62.

Brooke, J. et al. (1996). Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7.

Burke, R. (2007). The adaptive web. chapter hybrid web recommender systems.

Bø, K. E. (2017). A state of the art review of group and real estate recommendation.

Cantador, I., Bellogín, A., and Castells, P. (2008). A multilayer ontology-based hybrid recommendation model. *Ai Communications*, 21(2-3):203–210.

Choo, C. W., Detlor, B., and Turnbull, D. (2000). Information seeking on the web: An integrated model of browsing and searching. *first monday*, 5(2).

Corcho, O., Fernández-López, M., Gómez-Pérez, A., and López-Cima, A. (2005). Building legal ontologies with methontology and webode. In *Law and the semantic web*, pages 142–157. Springer.

Cosley, D., Lam, S. K., Albert, I., Konstan, J. A., and Riedl, J. (2003). Is seeing believing?: how recommender system interfaces affect users' opinions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 585–592. ACM.

Daly, E. M., Botea, A., Kishimoto, A., and Marinescu, R. (2014a). Multi-criteria journey aware housing recommender system. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 325–328. ACM.

Daly, E. M., Botea, A., Kishimoto, A., and Marinescu, R. (2014b). Multi-criteria journey aware housing recommender system. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 325–328. ACM.

De Carolis, B. and Pizzutilo, S. (2009). Providing relevant background information in smart environments. In *EC-Web*, pages 360–371. Springer.

De Pessemier, T., Dooms, S., and Martens, L. (2014). Comparison of group recommendation algorithms. *Multimedia Tools and Applications*, 72(3):2497–2541.

Donoho, D. L. et al. (2000). High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, 1:32.

Faulkner, L. (2003). Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3):379–383.

Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc.

Grant, D. and Cherif, E. (2016). Using design science to improve web search innovation in real estate. *Journal of Organizational Computing and Electronic Commerce*, 26(3):267–284.

Gu, M. and Aamodt, A. (2006). Evaluating cbr systems using different data sources: a case study. In *European Conference on Case-Based Reasoning*, pages 121–135. Springer.

Heath, S. (2016). Young, free and single? *Routledge Handbook of Youth and Young Adulthood*, page 199.

Hevner, A. and Chatterjee, S. (2010). *Design research in information systems: theory and practice*, volume 22. Springer Science & Business Media.

Hill, W., Stead, L., Rosenstein, M., and Furnas, G. (1995). Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 194–201, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Ho, H.-P., Chang, C.-T., and Ku, C.-Y. (2015a). House selection via the internet by considering homebuyers' risk attitudes with s-shaped utility functions. *European Journal of Operational Research*, 241(1):188–201.

Ho, H.-P., Chang, C.-T., and Ku, C.-Y. (2015b). House selection via the internet by considering homebuyers' risk attitudes with s-shaped utility functions. *European Journal of Operational Research*, 241(1):188–201.

Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666.

Jameson, A. (2004). More than the sum of its members: challenges for group recommender systems. In *Proceedings of the working conference on Advanced visual interfaces*, pages 48–54. ACM.

Johnson, C. C. (2014). Logistic matrix factorization for implicit feedback data. *Advances in Neural Information Processing Systems*, 27.

Kelly, D. and Teevan, J. (2003). Implicit feedback for inferring user preference: a bibliography. In *ACM SIGIR Forum*, volume 37, pages 18–28. ACM.

Kim, K.-j. and Ahn, H. (2008). A recommender system using ga k-means clustering in an online shopping market. *Expert systems with applications*, 34(2):1200–1209.

Konstan, J. A. and Riedl, J. (2012). Recommender systems: from algorithms to user experience. *User Modeling and User-Adapted Interaction*, 22(1):101–123.

Li, S., Nomura, S., Kikuta, Y., and Arino, K. (2017a). Web-scale personalized real-time recommender system on suumo. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10235 LNAI:521–538. cited By 0.

Li, S., Nomura, S., Kikuta, Y., and Arino, K. (2017b). Web-scale personalized real-time recommender system on suumo. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10235 LNAI:521–538. cited By 0.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*.

Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80.

Lops, P., Gemmis, M. D., Semeraro, G., Lops, P., Gemmis, M. D., and Semeraro, G. (2011). Chapter 3 content-based recommender systems: State of the art and trends.

MacQueen, J. et al. (1967a). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

MacQueen, J. et al. (1967b). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

Masthoff, J. (2004). Group modeling: Selecting a sequence of television items to suit a group of viewers. In *Personalized digital television*, pages 93–141. Springer.

McCarthy, J. F. (2002). Pocket restaurantfinder: A situated recommender system for groups. In *Workshop on Mobile Ad-Hoc Communication at the 2002 ACM Conference on Human Factors in Computer Systems*.

McCarthy, J. F. and Anagnost, T. D. (1998). Musicfx: an arbiter of group preferences for computer supported collaborative workouts. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 363–372. ACM.

McNee, S. M., Riedl, J., and Konstan, J. A. (2006). Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 1097–1101. ACM.

Nielsen, J. (1994). *Usability engineering*. Elsevier.

O'connor, M., Cosley, D., Konstan, J. A., and Riedl, J. (2001). Polylens: a recommender system for groups of users. In *ECSCW 2001*, pages 199–218. Springer.

Park, Y., Hwang, H., and goo Lee, S. (2016). A novel algorithm for scalable k-nearest neighbour graph construction. *Journal of Information Science*, 42(2):274–288.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, pages 175–186, New York, NY, USA. ACM.

Roberts, S. (2013). Youth studies, housing transitions and the'missing middle': Time for a rethink? *Sociological Research Online*, 18(3):11.

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

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA. ACM.

Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating “word of mouth”. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 210–217, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Smaaberg, S. F. (2014). Context-aware group recommendation systems. Master's thesis, Institutt for datateknikk og informasjonsvitenskap.

Trewin, S. (2000). Knowledge-based recommender systems. *Encyclopedia of library and information science*, 69(Supplement 32):180.

Von Alan, R. H., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 28(1):75–105.

Yu, Z., Zhou, X., Hao, Y., and Gu, J. (2006). Tv program recommendation for multiple viewers based on user profile merging. *User modeling and user-adapted interaction*, 16(1):63–82.

Yuan, X., Lee, J.-H., Kim, S.-J., and Kim, Y.-H. (2013). Toward a user-oriented recommendation system for real estate websites. *Information Systems*, 38(2):231–243. cited By 19.

Zheng, S., Liu, H., and Lee, R. (2006). Buyer search and the role of broker in an emerging housing market: a case study of guangzhou. *Tsinghua Science & Technology*, 11(6):675–685.

Zumpano, L. V., Johnson, K. H., and Anderson, R. I. (2003). Internet use and real estate brokerage market intermediation. *Journal of Housing Economics*, 12(2):134–150.

Appendix A

Personality questions

A.1 Social habits questions

1. I tend to go out to meet friends, socialize or network most evenings
2. I like to have people over for drinks on a regular basis
3. I like having friends staying over for a few days
4. I would like my shared house to known as a place to party
5. I sometimes go out and come home in the early hours
6. Occasionally I bring people I have just met to my house

A.2 Cleanliness questions

1. There should be a rota for putting the bins out
2. I like to sort my spices and herbs clearly
3. I like the fridge clean and organized
4. There should be a rota for allocating household chores
5. I am usually the person nagging others to tidy up

A.3 Social openness questions

1. I see flatmates as people I live with rather than friends
2. If I could choose, I would prefer to live alone
3. I prefer to eat in my room rather than in the communal areas
4. I spend most of my time in my room

A.4 Social flexibility questions

1. I don't mind if my flatmates invite friends to our house, as long as they give me notice
2. I am relaxed about the sexual choice of my flatmates
3. It is sometimes OK to break the rules
4. I am relaxed about the religious choices of my flatmates
5. I am happy to help a flatmate with a personal task, for example ironing his/her shirt or driving him/her to the train station

Appendix B

Brute-force test with 64 Test Users

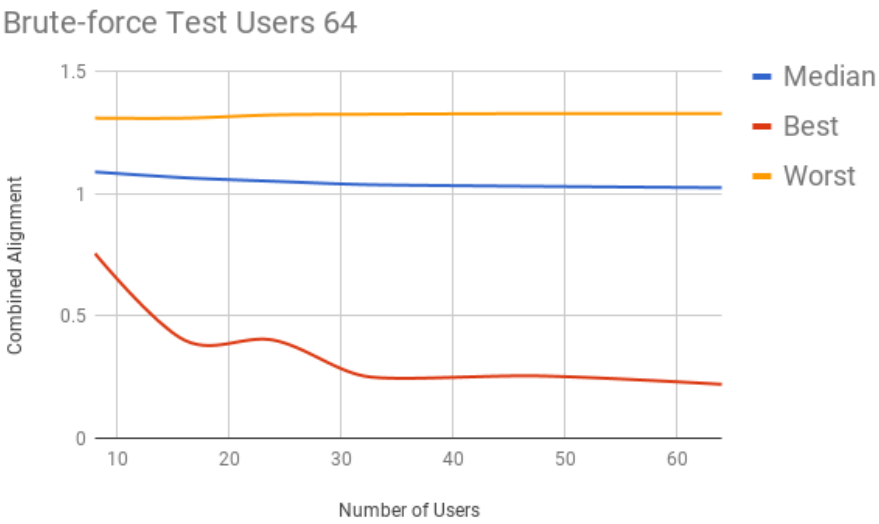


Figure B.1: Brute-force test on 64 Test Users

Appendix C

Surveys

The following pages are screenshots of the questionnaires that were used in the qualitative evaluations of the user interface and real estate recommendations. Feedback from the Application Specific survey are also presented.

C.1 Questionnaires

They are listed in the following order:

1. Background Information(BI) survey
2. System usability Scale(SUS) survey
3. Application Specific(AS) survey
4. Quality Evaluation(QE) survey

Background information (BI)

Age

Your answer

Nationality

Your answer

Gender

- ◯ Male
- ◯ Female
- ◯ Other:

Occupation

- ◯ Student
- ◯ Young professional

How many times have you searched for a new place to live i.e switched apartment

Your answer

How many times have you lived in a shared accommodation

Your answer

How many people lived in these share houses?

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5
- ☐ 6
- ☐ 7
- ☐ 8
- ☐ More
- ☐ Other:

What size of share house where you most happy with?

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5
- ◯ 6
- ◯ 7
- ◯ 8
- ◯ More
- ◯ Other:

In how many of those did you know the people you were moving in with

Your answer

In general, how satisfied were you with the apartment hunting process?

	1	2	3	4	5	
I was frustrated	○	○	○	○	○	It was easy

Have you heard of a service for group based recommendation of apartments?

- ○ Yes
- ○ No

Would you use it?

- ○ Yes
- ○ No
- ○ Probably

SUBMIT

Never submit passwords through Google Forms.

YAPS.life SUS survey

System Usability Scale, @DIgital Equipment Corporation, 1986

* Required

I think that I would like to use this system frequently *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I found the system unnecessarily complex *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I thought the system was easy to use *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I think that I would need the support of a technical person to be able to use this system *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I found the various functions in this system were well integrated *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I thought there was too much inconsistency in this system *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I would imagine that most people would learn to use this system very quickly *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I found the system very cumbersome to use *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I felt very confident using the system *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I need to learn a lot of things before I could get going with this system *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Application Specific Survey (AS)

Application Specific Survey (AS)

* Required

I believe I could get use of this application in the future *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The application provided recommendations relevant for me *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I found it helpful to use distance to my workplace to find relevant apartments *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Would you like to have a large amount of matches to choose from?

- ○ Yes
- ○ No
- ○ Other:

What additional functionality would you like to see in this application?

Your answer

How do you think the features in this application could be improved to better help find apartments that are relevant for your group of users?

Your answer

Do you have any other suggestions, comments or feedback?

Your answer

SUBMIT

Never submit passwords through Google Forms.

Quality Evaluation

The quality evaluation for yaps.life

Solo Recommendation

1. Go to yaps.life
2. Register/log in
3. Fill out your profile
4. Create a Solo match

Rate the ordering of the recommended listings

	1	2	3	4	5	
Very dissatisfied	○	○	○	○	○	Very satisfied

Rate this way of searching for real estate compared to ways you have tried before

	1	2	3	4	5	
Much worse	○	○	○	○	○	Much better

Group Recommendation

- Invite one or two other users to your match
- The listings will update to reflect your new group

Rate the ordering of the recommended listings again

	1	2	3	4	5	
Very dissatisfied	○	○	○	○	○	Very satisfied

Rate this way of searching for real estate as a group compared to ways you have tried before

	1	2	3	4	5	
Much worse	○	○	○	○	○	Much better

Comments

Your answer

SUBMIT

Never submit passwords through Google Forms.

C.2 Feedback

Table C.1: Application Specific survey, feedback for question 5

What additional functionality would you like to see in this application?
A search function
Maps showing travel from potential apartments to work
Se hvilke sommerjobber de har fått. Connect'e med facebook for å se felles venner, hvor de har gått, hvor de er fra, etc.
Feedback from earlier roommates (rating system?)
Other cities
Would be nice to see a map of Oslo with the workplaces of the group members along with apartment locations.
Heatmap overlay for google maps showing the potential for flatmates in one color and the potential available apartments in one color

Table C.2: Application Specific survey, feedback for question 6

How do you think the features in this application could be improved to better help find apartments that are relevant for your group of users?
A video meeting function
Maybe add other things than work that are important for apartment location, such as sports facilities, shops
Connect med facebook.
So many possibilities. I would love to see a well made matching algorithm for personalities and maybe an ai that could recognize things in the pictures of the apartment
Live collaboration on filtering and exploring appartment alternatives, an interface using google maps could be cool.

Table C.3: Application Specific survey, feedback for question 7

Do you have any other suggestions, comments or feedback?
Very cool idea!
Finjusteringen her og der som skrevet om tidligere, men ellers veldig bra Kristian <3 <3
Great site, and i really see the use for people looking for an appartment in Oslo during the summer! Great way to rent out the appartment as well!
Awsome idea. Could be really usefull for many applications
Google maps dude, google it!

Appendix D

Code

The code for the user generation is included here for convenience. All other code is can be found and reviewed at `https://github.com/kristianeboe/yaps.life`.

D.1 User generation

Listing D.1 Create test users

```
export function createTestUsers(n) {
  const users = []

  for (let index = 0; index < n; index += 1) {
    const personalityVector = []
    for (let j = 0; j < 20; j += 1) {
      personalityVector.push(getRandomInt(5))
    }
    const university =
      UNIVERSITIES[Math.floor(Math.random() * UNIVERSITIES.length
          )]
    const fieldOfStudy =
      STUDY_PROGRAMMES[university][
        Math.floor(Math.random() * Math.floor(STUDY_PROGRAMMES[
            university].length))
      ]
    const workplaceKey = Math.floor(Math.random() * Math.floor(
        Object.keys(WORKPLACES).length))
    const workplace = Object.keys(WORKPLACES)[workplaceKey]
    const workplaceLatLng = WORKPLACES[workplace]
```

```
19      const budget = BUDGETS[Math.floor(Math.random() *
            BUDGETS.length)]
20      const propertySize = PROPERTY_SIZES[Math.floor(Math.random()
            * PROPERTY_SIZES.length)]
21      const standard = STANDARD[Math.floor(Math.random() *
            STANDARD.length)]
22      const style = STYLE[Math.floor(Math.random() * STYLE.length)]
23      const propertyVector = [budget, propertySize, standard, style
            ]


26      const rentFrom = new Date(2018, 5, getRandomInt(14))
27      const user = {
28        uid: uuid.v4(),
29        displayName:  testUser${index} ,
30        matchLocation:  Oslo ,
31        seeNewUsers: false,
32        workplace,
33        propertyVector,
34        workplaceLatLng,
35        photoURL:  https://placem.at/people?w=290&h=290&random=${
              getRandomInt(100)} ,
36        university,
37        age: Math.floor(Math.random() * 10) + 20,
38        rentFrom,
39        tos: true,
40        readyToMatch: true,
41        gender: GENDERS[Math.floor(Math.random() * GENDERS.length)]
              ,
42        fieldOfStudy,
43        personalityVector
44      }
45      users.push(user)
46    }
47    return users
48  }
```