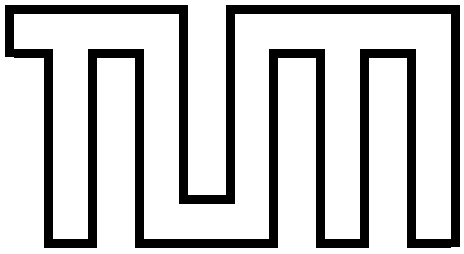


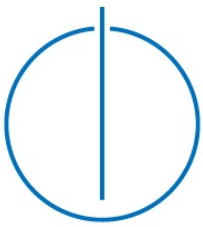
Technical University of Munich

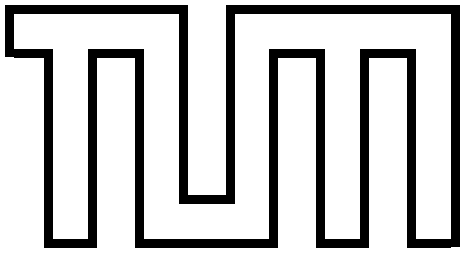
Department of Informatics

Bachelor's Thesis in Informatics

Emotion-based Recommender System for City Visitors Built on Analyzing Egocentric Images

Andreas Hitz





Technical University of Munich

Department of Informatics

Bachelor's Thesis in Informatics

Emotion-based Recommender System for City Visitors Built on Analyzing Egocentric Images

Emotionsbasiertes Empfehlungssystem für Stadtbesucher, aufbauend auf der Analyse egozentrischer Bilder

Author: Andreas Hitz

Supervisor: Prof. Dr.-Ing. Jörg Ott

Advisor: Prof. Dr. Stephan Sigg, Aalto University

Submission: 13.03.2018

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

München, 13.03.2018

(Andreas Hitz)

Acknowledgments

I especially want to thank Stephan, my advisor from Aalto, who has always been a great motivator, constantly and quickly gave me lots of constructive feedback and helped me to head in the right direction with this thesis.

A big "Thank you" also goes to Prof. Ott, my supervisor from TUM, who gave me the opportunity to write this thesis at Aalto and in this way spend another semester in Finland.

Last, but not least, it is important to appreciate the work of all the awesome people from StackOverflow, who have a solution for every problem, an answer to every question, and helped me to learn both Python and JavaScript. Without them, it would have been a lot more complicated to finish this thesis!

Abstract

This thesis designs and implements a recommender system for city visitors, which is based on the analysis of emotions of pictures as well as the user's location and own emotion.

Following a comprehensive literature review of related work on recommender systems, computer vision and emotion analysis, a conceptional architecture for the recommender system is designed and described step by step, covering all technical implementation details, such as used algorithms, libraries and technologies. Firstly, databases of pictures collected from different sources are created. The contents of these photos are analyzed to receive a textual content description and identify the emotion of an image. Each image is, in addition to its extracted associated information, tagged with an emotion score, which is based on the two dimensions pleasure and activity. Secondly, the Python-based recommender system generates personalized recommendations for places to visit by using these scores together with the user's location and emotion. A graphical user interface written in HTML and JavaScript serves as the base for accessing the system, where the pictures are marked on an interactive map, the user specifies his own location and emotion, and the recommendations can be retrieved.

The thesis continues with an evaluation of the system by performing a user study which demonstrates the usefulness of considering emotions for recommendations. It ends with a discussion about challenges, a view on future work, and a conclusive summary of the topic.

Inhaltsangabe

Diese Bachelorarbeit entwirft und implementiert ein Empfehlungssystem für Stadtbesucher, welches auf der Analyse von Stimmung von Bildern sowie dem Standort und der eigenen Emotion des Nutzers basiert.

Nach einer umfassenden Literaturanalyse verwandter Werke über Empfehlungssysteme, Computer Vision und Emotionsanalyse wird Schritt für Schritt eine konzeptionelle Architektur für das Empfehlungssystem entworfen und beschrieben, inklusive aller technischen Details der Implementierung, wie verwendete Algorithmen, Bibliotheken und Technologien. Zunächst werden Datenbanken von Bildern erstellt, die aus verschiedenen Quellen stammen. Die Inhalte dieser Fotos werden analysiert, um eine textuelle Beschreibung des Bildinhalts zu erhalten und die Stimmung eines Bildes zu identifizieren. Jedes Bild wird zusätzlich zu den ihm zugehörigen Informationen mit einem Wert für die Stimmung markiert, der auf den beiden Dimensionen Freude und Aktivität basiert. Anschließend generiert das auf Python basierte Empfehlungssystem personalisierte Empfehlungen für zu besuchende Orte unter Verwendung dieser Werte zusammen mit dem Standort und der Emotion des Nutzers. Eine graphische Benutzeroberfläche, programmiert in HTML und JavaScript, dient als Basis für den Zugriff auf das System, in dem die Bilder auf einer interaktiven Karte markiert sind, der Nutzer seinen eigenen Standort und die Emotion angibt, und die Empfehlungen abgerufen werden können.

Die Bachelorarbeit fährt fort mit einer Evaluation des Systems anhand der Durchführung einer Nutzerstudie, die den Nutzen der Berücksichtigung von Emotionen für Empfehlungen zeigt. Sie endet mit einer Diskussion über Herausforderungen, einem Ausblick auf weitere Tätigkeiten, und einer stichhaltigen Zusammenfassung des Themas.

Contents

Chapter 1

Introduction

1.1 Motivation and Problem Description

Recommender systems have been extensively used by popular online services in the past few years. Whenever similar movies are suggested on Netflix, or Amazon shows related items to a certain product, a recommender system is the foundation for these suggestions. In E-commerce, relevant recommendations can help the service providers to increase their sales numbers. With internet services becoming increasingly personalized, the need for highly customized and user-tailored solutions is even more accelerated. Consistent developments in the handling of big data, machine learning and power of computers benefit this development towards fully personalized software as well.

In the area of tourism, recommending places to visit is a common application. Most existing systems are built on analyzing only general facts about its users or characteristics of the places to be recommended, or contextual information such as location, time of day, or weather. The **motivation** for this thesis is that, besides these aspects, emotions are an important personal factor that can help optimizing the recommendations significantly. The level of personalization can be improved a lot by taking the emotion, i.e. mood, of users, and emotions expressed in pictures into account.

Objectives that need to be discussed are, first of all, to find sources for the recommendations, i.e. pictures of places to be recommended. An important question following this is how to determine emotions present in pictures, that is, how to assign emotions to pictures, and where the information about the user's emotion comes from. This requires a clear definition of the concept emotion. Furthermore, it is necessary to build a system capable of calculating and generating recommendations by taking the emotions of its user

and considered pictures into account, and elaborate appropriate means to present the resulting data in a user-friendly way. After all, it also needs to be examined whether the consideration of emotions can really improve the quality of recommendations.

Challenges are, amongst others, that the topic of emotions, closely connected to the area of psychology, is a rather subjective matter, and that it is necessary to create an automated, efficient solution that treats all data inputs equally, while still remaining accessible for all users. More challenges will be revealed later on during the evaluation and discussion in more detail.

1.2 Proposed Solution and Contribution

In this thesis, it is aimed to design and implement a location- and emotion-based recommender system as a technical **solution** for the described problem to enhance personalized recommendations. The system is written in Python on the backend side, where all the data is processed and the recommendations are calculated, and in HTML and JavaScript on the frontend side, i.e. user interface. In the system, the component of emotion is used as an extra parameter for recommendations, in addition to the location: As a general idea, the user of the system can select his own emotion, and based on this, nearby pictures expressing a similar emotion are recommended. This connects the topics of recommender systems and emotion recognition from images. The term emotion is defined as a value on two dimensions: pleasure (whether the user feels rather positive or negative) and activity (whether the user feels rather active or passive).

Projecting the recommendations on a website in a useful, clear way helps the user to get ideas for places he could visit, thus reaching a positive user experience for the recommender system. In addition, a user study is carried out after developing the recommender system which finds that, although many factors are challenging for the success and accuracy of the system, the overall rating of the participants, on a scale from one to five stars, for the recommendations based on the emotion is about half a star higher than when the emotions are not considered. This demonstrates that using the emotion component significantly improves the quality of recommendations, which is the main **contribution** of the thesis. In combination with other parameters, this can lead to recommender systems delivering far more personalized results than the state-of-the-art systems of today are able to do.

1.3 Outline

The thesis is divided into seven parts: Following this brief introduction, **Chapter 2** starts with an overview of related work on recommender systems, with a focus on mobile and tourism recommender systems, to give an extensive insight about the context of this thesis. The chapter also covers important background information about applications from computer vision, introduces egocentric vision, and focuses especially on the analysis of emotion in text and images. **Chapter 3** describes and documents the built emotion-based recommender system as a whole with all its needed components, divided into detailed sections, following a general overview, about data sources, used algorithms and programming languages, and the presentation via a graphical user interface. Every section discusses possible alternatives and explains the advantages of the eventually applied solutions. Examples are used to show the functionality of the system step by step, the analysis of the data produced during the recommendation process justifies the taken approaches. **Chapter 4** evaluates the system by conducting a user study, showing the usefulness of the emotion-based recommendations generated by the system, and demonstrating the reached improvement. In **Chapter 5**, many aspects of the system are discussed again, emphasizing possible future work and enhancements, before **Chapter 6** eventually closes with a summary and draws conclusions. An **Appendix** contains information about how the recommender system can be used, as well as the project description uploaded to GitHub.

Chapter 2

Related Work

This chapter gives detailed information about existing academic work related to the technical background of recommender systems, its general concepts, methods and challenges, and sums up in particular mobile and tourism recommender systems. Following this, a few applications of computer vision are described, namely activity, object and emotion recognition. After an introduction to egocentric vision, the topic of both textual and visual emotion analysis is discussed.

2.1 Recommender Systems

Recommender systems are software tools and techniques which try to predict the popularity of certain items, e.g. music or movies, for their users, and aim to give personalized, accurate recommendations. The user can possibly rate items, for instance on a scale from one to five stars, according to his satisfaction with the recommendations, which subsequently can lead to better suggestions. Recommender systems have been invented and implemented first in the 1990s, and are nowadays used in many popular online services, such as Amazon or Netflix, and for various different purposes. Recommender systems are as well subject to many research studies [Beel 16]. There are two main approaches for giving predictions, content-based filtering and collaborative filtering, which can be combined to hybrid systems [Ricc 15], as illustrated in Figure 2.1. These are introduced briefly in the following.

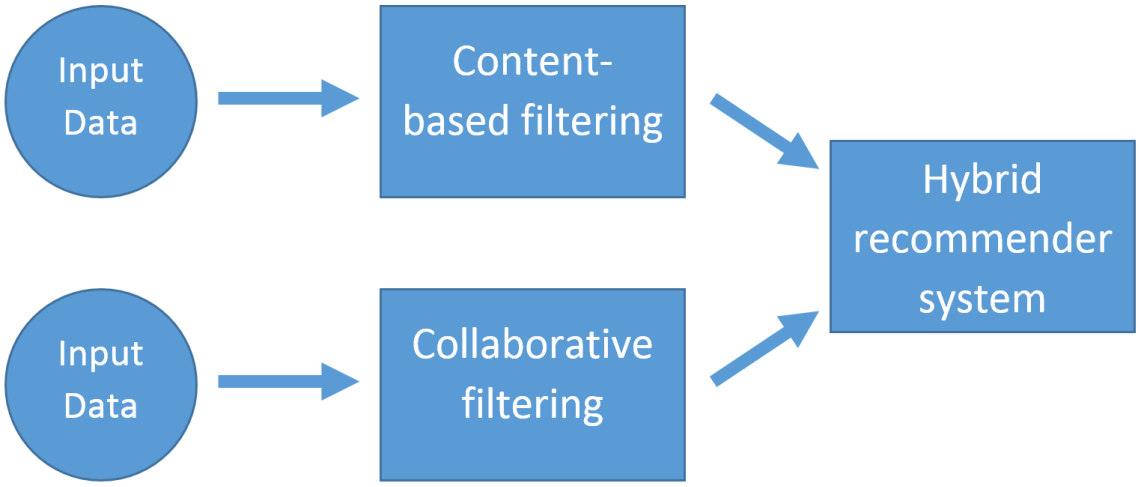


Figure 2.1: Recommender system approaches

2.1.1 Content-based Filtering

In **content-based filtering**, the recommendation takes place by analyzing and categorizing specifically defined characteristics of an item, e.g. through keywords, and finding others with similar properties, thus requiring appropriate means to measure similarity [Melv 17]. One example is Pandora Radio, which plays music with analogical characteristics to these of a song provided by the user. For each user of the system, a content-based profile is built containing information about his item preferences: The item features are weighted according to the importance of the features for the user. Simple algorithms or more advanced machine learning techniques are used for the computation of the weights. The profile is developed and adjusted over time when the user interacts with the system. New items are compared with items previously rated by the user, and the most fitting ones are then recommended by the algorithm [Ricc 15]. Direct user feedback can be additionally used to assign higher or lower weights on the importance of certain attributes, thus refining the results. Many online music or movie recommender systems are using the content-based approach, where the cold start problem, i.e. when in the beginning a big set of data is needed to get an accurate analysis, is not as big as at collaborative filtering, since the amount of inputs needed to start is a lot smaller. A disadvantage is the limited scope of this technique to one type of content - for instance, by knowing the music preferences of a user, a system can not draw conclusions to the movie preferences - and the smaller influence of the users towards recommendations compared to collaborative filtering.

2.1.2 Collaborative Filtering

In **collaborative filtering**, the predictions are based on users' decisions of the past, and on decisions which have been made by other, similar users. This means that a high amount of data is analyzed to understand user preferences and to eventually provide precise recommendations [Melv 17]. With the collected amount of information in internet services becoming constantly bigger, this technique is nowadays used more frequently. The data can be gathered explicitly, e.g. by specifically asking a user to rate an item, or implicitly, e.g. by analyzing what items a user has been looking at in an online service. A set of recommended items is generated by comparing this data to data from other users, and prioritizing the best matches. This process requires special algorithms. One of the most popular examples for collaborative filtering is Amazon's feature "Customers who bought this item also viewed / purchased" presenting a list of other products [Lind 98], which is shown on their website when viewing any product. Another commonly known example is Facebook's friend recommendation feature, which works by examining the network of connections between a user and their friends. An advantage of this approach is that the content of the items does not need to be machine analyzable, which means that even complex items can still be recommended without algorithms capable of analyzing the content itself. Disadvantages are, amongst others, the cold start issue, sparsity - due to a large amount of items, many items have only few ratings - and difficult scalability - the computation power needed to give recommendations for many users and items is extremely high [Ricc 15].

2.1.3 Hybrid Systems

Combining these two approaches into a **hybrid** system can make the recommendations even more accurate [Jann 10]. Such a system tries to combine the advantages of both filtering techniques. [Burk 07] systematically compares various recommender systems consisting of two parts, and defines seven different hybridization techniques. These include **weighted**, when the score of different recommendation components are combined numerically, **cascade**, when certain recommenders are prioritized over others, and **feature combination**, which combines different sources of information to create one single recommendation algorithm. Netflix, for instance, is using a mixed technique by both comparing watching habits of similar users, and recommending movies or series with user-preferred characteristics [Gome 15].

2.2 Mobile Recommender Systems

In the last few years, smartphones have become more powerful and mobile internet availability has improved significantly. Therefore, recommender systems have especially found their way into mobile devices. Information from the web can be used to enhance the experience of users, while they can at the same time directly interact with their surroundings. Recommender systems have originally been used on the web in the context of movie or music discovery, in e-commerce, or to personalize search results. The best fitting results are created through observing user behavior and in this way finding out personal preferences. Now, for the use in mobile devices, it is especially relevant for the users to get recommendations from their direct surroundings, thus drawing a focus on the concepts of context and location.

2.2.1 Definitions and Tasks

As described before, **classic recommender system** recommend a set of interesting items for each user according to their preferences, e.g. through collaborative or content-based filtering. **Location-based recommender systems** focus not only on user preferences, but additionally give more information related to the location of a user, for instance to calculate routes, give traffic information, and find restaurants, places or even friends which are nearby [Schi 04]. In the case of a system like Google Maps that suggests places to see, restaurants to visit, or public transportation, the most important factor for giving personalized recommendations is the user's position.

Solely **web-based recommender systems** focus on recommending content that people like, while computing implicit relationships between items via certain algorithms, e.g. through user feedback. **Mobile recommender systems** also take spatial relationships between items, e.g. geographical distances between places, into consideration, and therefore act on a context-aware level [Adom 15].

The user **context** is a multidimensional characteristic which can be classified in different ways. [Adom 15] mentions three theoretical aspects: The **observability**, i.e. the knowledge of a recommender system about the contextual factors, can be fully, partially, or not at all present. The **behavior** of contextual factors, i.e. whether and how their structure and importance change over time, can be static or dynamic. The contextual information can be **obtained** in a number of ways, depending on the use case: explicitly, implicitly or inferred. [Sche 15] categorizes context into two classes: **Environment-related** context

means features measured by sensors on the user's mobile device or obtained from external information services. Examples include location, current time, weather or temperature. **User-related** context contains high-level information about the user. Examples include the user's activity, emotional state, or social and cultural background. It is more difficult to measure and can be inferred from environment-related context.

According to [Lath 15], there are three different tasks or use cases for mobile recommender systems:

1. **Goal-oriented search**: "Where can I eat around here tonight?" Users who intend to take an action want to receive personalized results.
2. **Location discovery**: "Which places around here are interesting? What can I do tonight?" Users want to get to know their environment and receive actual, up-to-date information.
3. **Routing and transport**: "How can I get from here to there?" Users want to get directions to be mobile, which is especially relevant when being in a new city for the first time.

[Lath 15] defines three important features for such mobile recommender systems: data, algorithms and evaluation. These will be discussed in detail in the following, along with various examples of existing recommender systems.

2.2.2 Data

First of all, **data** has to be collected from the users and the surroundings to record signals which reflect user preferences. The amount of data in mobile context is larger than in an environment without mobility. There are three means of collecting data: **Explicit** data is created when a user likes, rates or checks in at a place to share his location with others. Several examples are widely known on the internet, such as Foursquare or Yelp, which are location-based recommendation services, the latter of which collects user reviews of places. The user could even be asked directly about his own personality. By performing activities such as watching videos or clicking on links, users generate **implicit** data. On mobile devices, they can, in addition to that, also take photos or track their physical movements; both actions produce a lot of data. Another feature of mobile devices are their various sensors which create **sensor** data, for instance in the form of location and mobility tracking.

When a user checks in multiple times at one place, it becomes visible that the user has a preference towards this place [Hu 08], although it could also simply mean that a user is visiting the place periodically in his daily routine, e.g. due to work, which covers between 50% and 70% of a user's mobility [Cho 11]. Therefore it is a main challenge for recommender systems to not only show old, known locations, but also enable the discovery of new places.

Mobile recommender systems need databases of Points of Interests (POIs). These can be generated either by analyzing explicit data - as e.g. Foursquare does [Noul 11], or implicit data - as e.g. the image- and video-hosting website Flickr does by clustering geo-tagged photos [Cran 09]. Out of this information, features of places can be extracted. Usage scenarios of the analysis of user behavior through geo-tagged photos include identifying trips or analyzing tourist navigation and travel behavior in cities, which can in return be used to find interesting items for users [Gira 08]. Other used information includes mobility data from the mobile network like GSM, or positioning data such as GPS [Zhen 08]. Constant usage of these sensors gives more details about places visited by the user, but at the same time drains the battery quickly, which is not as energy efficient as desired in mobile context [Rach 10]. It is possible to get a very detailed user profile using all the exact information, but the data collection must at the same time take place efficiently. Another challenge for a gapless collection of information is that e.g. Foursquare users do not always check in to every place they visit, indirectly manipulating the data through this behavior. All gathered data therefore has to be processed to make accurate recommendations. All in all, data acquired in a mobile context is very dynamic, thus requiring a recommender system to efficiently learn from both users and recommendable items.

Smartphone sensors can also deliver helpful data for monitoring, analyzing and detecting user activities for different purposes [Lane 10] [Khan 13]. One area is **activity recognition**, i.e. the context of a user, and whether he is walking, sitting, driving or talking [Chou 08]. It is also possible to analyze the **transportation mode**, using e.g. the accelerometer and GPS sensor [Sten 11]. Even the **sociability** and user interactions can be measured with Bluetooth, accelerometer and microphone [Rach 11].

2.2.3 Algorithms

The obtained data needs to be further processed with the right **algorithms**. Speaking about a recommender system which recommends items, i.e. places, based on ratings, the algorithms often make use of supervised learning approaches to take characteristics like the distance between places into account. Generally, such a recommender system contains

a set of items as well as of users, who rate a fraction of all items. The system has to recommend interesting items by predicting ratings and ranking items. In mobile context, challenges for the prediction include finding out whether the user is a local, on a travel or on a business trip, and whether he was satisfied with a visit or not, because these factors change the user's perception of an item. Machine learning techniques can help to address these problems and to further enhance the accuracy of predictions.

[Lath 15] defines four variants of this prediction problem in more detail: the recommendation of categories, next places, new places, and routes.

1. In **categorical recommendation**, items of certain categories are recommended, e.g. restaurants, shops or sights [Ardi 11]. Here the ranking of items takes place mainly based on the location, perhaps also on factors like the average cost of a venue.

2. When predicting the **next place** to visit after the current one, several inputs are taken into account to give accurate recommendations: the user and both his preferences and location history, the current location, and the current time. The formula used by the algorithm therefore depends on various aspects [Noul 12a]. A challenge here is again that a user may visit many places only because it is part of his daily routine.

3. Recommending **new places** requires a formal definition of discovery, which refers to visiting previously unvisited places, and rediscovery, which means that venues which have not been visited for a certain amount of time are visited again [Noul 12b]. The cold start problem is a challenge here as well, since not all necessary data might be initially available.

4. As a last variant, **routes** can be recommended, similar to suggesting a few next places. The additional aim is to find an optimized route according to the user's preference, taking e.g. the time to travel between two destinations, the opening times of venues, or the best time to visit them into account.

In addition to these, things such as the recommendation of (social) **events** are also possible [Skla 12].

Many algorithms use the method of collaborative filtering, representing users and venues, i.e. items, as vectors. Due to the context-aware, location-based nature of mobile recommender systems, recommendations must be pre- or post-filtered, so that results are only shown for a particular geographical area [Adom 15].

[Lath 15] defines two categories that can be used by algorithms for predictions: popularity and proximity. When recommending places, their **popularity** is not necessarily personalized, but a very important factor. Popularity can be measured by the number of visitors, frequency of visits or within certain categories. **Proximity** measures only take the current position into consideration and recommend the nearest places [Quer 10]; yet this is a powerful and reasonable factor, because people tend to travel rather shorter distances.

To get more personalized results, supervised learning can also be applied on the users' mobility preference data to find out the likelihood of them visiting specific places [Noul 12a]. [Lath 15] mentions three features: place, user and structure.

1. **Place** features can include the popularity of a place in general, at a specific time, in its category, e.g. restaurant or bar, or in its local area.
2. **User** features are for instance the amount of times a user has already visited a certain place or places of a certain category, his overall historical mobility, or even the mobility of his friends.
3. **Structural** features investigate the geographical distance between places or their rank distance, or, if enough data is available, the transitions between places, i.e. how likely it is that a user moves from A to B.

The algorithms can now create an instance for all user visits, and use them to train a **supervised learning** approach, e.g. decision trees or linear regression [Noul 12a]. Other methods use **random walks**: The set of users and venues, i.e. items, is represented as a graph, with nodes for all users and venues. If a user visits a venue, a link is created; weights on links stand for the transition probability. In the case of social networks, there can also be links between users [Noul 12b]. Algorithms from the field of machine learning include: **similarity measures** such as the Euclidean distance or cosine similarity applied on a suitably prepared set of data to find the best recommendations; **classification and regression** algorithms such as k-nearest neighbor search to categorize items; **clustering** algorithms such as k-means, which divides a set of items into k equally divided clusters, or DBSCAN, which uses a density-based approach to find groups of points that are closely packed together.

2.2.4 Evaluation

Eventually, an **evaluation** of the system needs to take place. This can be achieved through quantitative and qualitative methods. For **quantitative** methods, two sets, one for training and one for testing, are generated from the acquired data and tested to measure the

predictive power of the learning algorithms by using ranking metrics [Quer 10]. **Qualitative** methods include performing surveys or doing interviews with test users of the system. This gives a more detailed understanding about the users' experiences, but the scalability can become a problem, when trying to get a set of participants that is large enough for collecting sufficient data. In general, the approach is beneficial though, because real human users are testing the system and give their personal opinions, which can differ a lot from the results of the automated evaluation and eventually of the recommendations generated by the algorithms.

2.2.5 Summary and Challenges

As a conclusion, it is necessary to consider not only aspects also used in classic, web-based recommender systems, such as discovering new or other, diverse items, but also special characteristics, for instance the distance between items, the time of day or the venues' opening hours, to build a useful mobile recommender system. There are several challenges according to [Lath 15]. The first is the increased consideration of **user context**: Recommendations need to take not only the time of day or location into account, but also whether the users are alone or in groups, their activity - sightseeing, work, commuting - or mood [Adom 15]. Another are **hierarchical item sets**: Items are dynamically changing, their characteristics and structures have hierarchical and spatio-temporal relations [Jamb 10]. In particular, there is a trade-off between distance and preference: Nearby items are not necessarily the user's most preferred; likewise, items which are far away could be favored anyway. A recommender system must find an appropriate balance between these two factors. **Privacy** questions also arise: How can it still be guaranteed, if so much data is collected and analyzed and location tracking takes place? One possible solution is to obfuscate the data [Quer 11]. Recommender systems can also **proactively** send notifications to the user [Vico 11]. However, constantly pushing relevant information might also **interrupt** the user too often. As a last point, there is also the possibility of **different users and items**: Recommendations do not necessarily always have to be made from places for people; a wide range of other application fields exists as well.

In addition to that, mobile recommender systems have to face other challenges as well: One characteristic of mobile **data** is its huge **complexity**, because it is heterogenous and noisy [Ge 10]. Similarly, many recommendations within one system are not always suitable for all regions, for instance products which are only sold to specific markets. A service running on a mobile device furthermore requires that computation quantity and **energy consumption** remain as low as possible.

Table 2.1 summarizes the mentioned features of mobile recommender systems.

Tasks (2.2.1)	**Data sources** (2.2.2)	**Place recomm. types** (2.2.3)	**Prediction categories** (2.2.3)
- Goal-oriented search - Location discovery - Routing and transport	- Explicit - Implicit - Sensors	- Categories - Next place - New places - Routes - Events	- Popularity - Proximity

Mobility features (2.2.3)	**Applied algorithms** (2.2.3)	**Evaluation methods** (2.2.4)
- Place - User - Structure	- Supervised learning - Random walks - Similarity measures - Classification and regression - Clustering	- Quantitative - Qualitative

Table 2.1: Features of mobile recommender systems

2.3 Tourism Recommender Systems

One advantage of recommender systems is that they clearly reduce the overload of information which is present on the internet. When making recommendations for tourists, they can suggest various kinds of content to enhance their experience, including media such as photos or videos of the places to visit or ratings of other users for a specific location. These ratings, for instance, can have an impact on the user's choice especially when they were given by friends of the user, thus giving recommender systems even a role as social actors [Yoo 15].

Current developments in fields like web technologies, wireless networks, and social networking make it possible to deliver effective and accurate recommendations to the tourist, which consider the users' interests and preferences, as well as their social and environmental context, thus delivering tailored context-aware services. Advancements in mobile computing make information accessible everywhere at any time. According to the classification of [Gava 14], existing mobile recommender systems for the support of tourists from three different areas will be briefly introduced in the following.

1. Recommendations for **attractions** in cities, for instance museums, cultural monuments or churches, are a popular application area for mobile tourism recommender

systems. The **graphical presentation** of attractions can for example be done as a list or by highlighting icons on a map interface. While many systems base their **computation** on preferences stored in the user's profile, afterwards filtering by the location [Nogu 12], it is nowadays becoming more common to consider also contextual parameters, for instance time, weather, social environment, or the user's mood [Sava 12]. The **type** of recommended content can be text and images, sound and video, maps, or augmented reality views. [Neid 14] claims that tourism is a complex, emotional experience. The described system elicits and expresses user preferences by letting the user choose between pictures that illustrate certain types of travel destinations. After the user has selected pictures which look appealing to him, a travel profile is generated, mixing multiple types of travellers and trying to predict travel behavior. Recommendations are based on this profile.

2. Suggestions for **tourist services**, for instance restaurants, hotels or information services, are important for city visitors as well. Here it is especially relevant that users might select characteristics which they expect their recommendations to have, and in this way give the recommender system constraints already beforehand; these differ depending on the service. More sophisticated systems try to model the probabilistic influence of input parameters on the values attributed by the users towards the services, e.g. type and price for restaurants or hotels, or day time and temperature for the user context, and calculate a score for the specific place using advanced algorithms [Park 07].

3. Many recommender systems focus on collaborative **user-generated content**, additionally featuring characteristics of social networks. They try to encourage the users to share their experiences while exploring new places [Sava 12], while at the same time automatically logging user activity, and providing a central location for managing tourist-relevant content.

Besides these areas, there are also systems that recommend **routes and tours** or **multiple-day trips**.

Since mobile recommender systems in tourism are complex, diverse and multidimensional and feature various architectural, technological and functional aspects, [Gava 14] classifies them within three different schemes, focusing on the used methods for fetching content relevant for tourists, finding out user requirements, considering situational context, and ranking the recommended items in a specific order. The three categories are briefly introduced in the following, in combination with a few practical examples. Figure 2.2 shows an overview for an architecture of such a tourism recommender system.

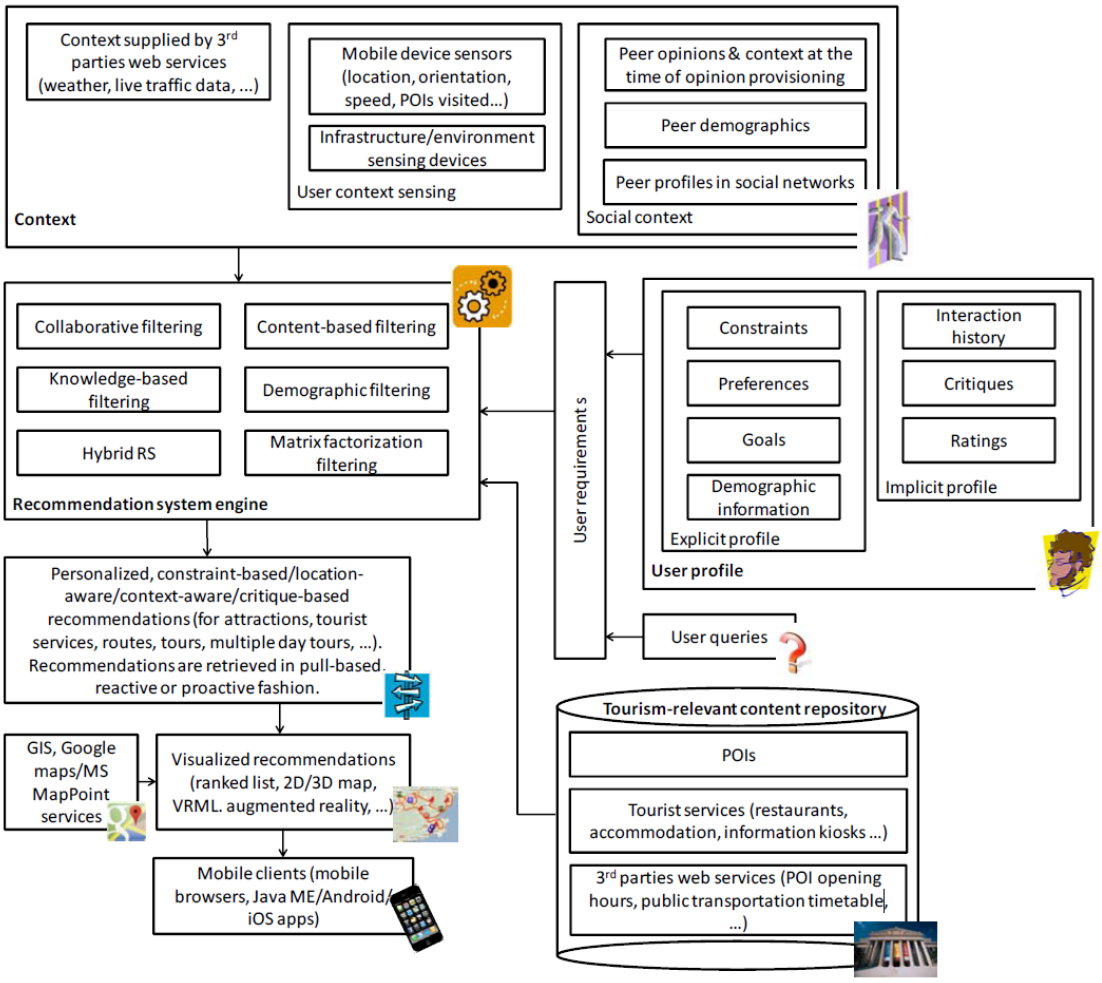


Figure 2.2: Generic architecture of a mobile tourism recommender system (from [Gava 14])

2.3.1 Architectural Styles

Web-based architectures clearly differ between the server part, which manages all the recommendation logic, and client part, which acts just as the presentation medium [Gava 12]. These systems are the most powerful, due to the server having high computational resources. However, network connectivity is required, since otherwise the client only has basic offline functionality, although still featuring user interface widgets and persistent storage.

Standalone architectures are full mobile applications, which include all the recommendation logic and tourist content, for example mTrip [1]. They are installed directly on the mobile device, and are capable of working offline. As a downside, it is due to the smaller

[1] `https://www.mtrip.com/`

feature capabilities not possible to use recommendation techniques which are based on matching different user profiles, for instance collaborative filtering, within these systems.

Web-to-mobile architectures combine these two approaches: Users utilize a web interface first, where they select all the content and build their tourist plan. Then they switch to the app installed locally on the mobile device, which basically operates offline, but can connect to a server on-demand, for example to update public transportation data [Kent 07]. Similar to standalone architectures, collaborative filtering is not possible here either.

2.3.2 User Involvement

In **pull-based** recommender systems, the delivery of recommendations is only performed upon user request [Kent 07]. In this way, the information delivery is controlled, limited, and less intrusive for the user.

Reactive recommender systems react to changing situational context when generating recommendations, independent from explicit interventions through the user [Bell 07]. The user might be able to define own settings for the handling of changing context in this type of system.

Proactive recommender systems go even one step further: They use not only historic and current context, but even pre-cache appropriate content from the server on the mobile device by trying to predict future context [McCa 06]. This enables high responsiveness and secures from unstable network conditions. With high speed mobile networks becoming widespread it lost on meaning nowadays, but can still be considered in cases where recommendations are changing quickly, or when the user must not be distracted through the system, for example while driving.

2.3.3 Recommendation Criteria

User constraints-based recommender systems create a user profile by finding out user preferences either xplicitly - e.g. in the form of a survey where the system asks its users about their demographic information and recommendation wishes - or implicitly through the users' interaction with the system. The systems are all location-aware and use to some extent contextual information. However, they do not use an actual recommendation engine, so they do not have domain-specific knowledge. One example is DailyTRIP, a mobile, web-based tour planning system, using location, user preferences, and the opening

days and times of places to maximize the "profit" for users, by suggesting them to visit as many relevant places as possible, while not exceeding the specified travel budget [Gava 12]. In the mobile app mTrip, a similar system is used where the user can select his desired travel destinations, and the app generates trip itineraries accordingly, in addition making use of augmented reality features to improve the user experience while visiting sights.

Pure location-aware recommender systems are early versions of context-aware systems, when smartphones were still new and not equipped with many sensors or functionalities; they consider only the location for the calculation of recommendations and are therefore unidimensional. One example is a 3D Geographic Information System (GIS), which uses a hybrid recommmendation engine, but limits recommendations only to an area around the user's location; in the mobile app, a three-dimensional representation of the user's area is shown [Nogu 12].

Context-aware recommender systems on the contrary act on a multidimensional base: Due to the evolution of ubiquitous mobile devices with high computational power, contextual dimensions can be inferred and added implicitly. The recommendations can be personalized and adjusted in a very accurate way through various data sources, including smartphone sensors [Camp 12], web services delivering information, or suitable supporting infrastructure. Many different parameters can be used in addition to location, for instance time, season, weather, monetary budget, means of transport, mobility history and social environment. One example is "I'm feeling Loco": Recommendations are based on inferred user preferences as well as constraints of location, time, and automatically recognized transportation mode. The system uses Foursquare data to generate the user profile; the user just has to state his own mood, i.e. the type of places he is currently interested in visiting [Sava 12]. Explicit user critique, where the user gives feedback for a recommendation, e.g. on a scale from one to five, can be used, too [Ricc 07].

2.3.4 Summary and Challenges

Amongst mobile recommender systems, tourism is a dominating application field. The falling of limitations for mobile devices and increase of features brings new opportunities and developments, but challenges as well. [Gava 14] mentions the demand for **intelligent user interfaces** using techniques from human-computer Interaction, which is especially important because the proper visualization of data on a small screen can be difficult, though it is essential to be able to see all relevant information at a glance. Within this area, gesture [Lei 09] or hands pointing recognition [Khos 09] can be used as new means of user interaction to provide even more accurate recommendations. Furthermore, sufficient

context inference mechanisms are needed [Ricc 07]; if a user is not explicitly asked about his preferences, the recommender system might make wrong assumptions about his context and subsequently deliver incorrect recommendations. Another aspect is the trade-off between **user effort and accuracy**: It might be hard for the user to evaluate his exact preferences before getting offered an actual recommendation - he might simply not be able to extensively formulate his interests or know what he is looking for already beforehand. Relying on much privacy-sensitive information like demographics, location and interaction history or user behavior, the **privacy protection** of user profiles is essential, especially in the mobile field, where identity threats become more common. As a last point, a practical issue is mentioned: Many recommender systems focus either on attractions or on tourist services. However, if for example hotels or restaurants are recommended, not only their price or type is important, but also their local distance to the attractions. For this reason, a more **unified recommendation perspective** focusing on both aspects at the same time is needed.

Table 2.2 summarizes the mentioned features of tourism recommender systems.

Recommendation areas (2.3)	**Architectural styles** (2.3.1)	**User involvement** (2.3.2)	**Recommendation criteria** (2.3.3)
- Attractions - Tourist services - User-generated content - Routes and tours - Multiple-day trips	- Web-based - Standalone - Web-to-mobile	- Pull-based - Reactive - Proactive	- User constraints-based - Pure location-aware - Context-aware

Table 2.2: Features of mobile recommender systems

2.4 Computer Vision and its Applications

Computer vision deals with techniques of computers for gaining high-level understanding from the diversely formed data of digital images or videos, trying to automate tasks that the human visual system is also capable of. Tasks include methods for acquiring, processing, analyzing and understanding digital data, to extract information from these images and videos and transform it into descriptions of the real world. Such artificial systems and models are for instance constructed with the aid of geometry, physics and statistics. Three technologies within the area of computer vision are briefly introduced in the following; these are activity, object and emotion recognition.

2.4.1 Activity Recognition

Activity recognition tries to recognize the activities and goals of a human from a series of observations on his actions and the conditions of the environment around him. The recognition of plans, goals and behavior is subject of research in this area, as well as the estimation of locations. There are different types of activity recognition that are all based on data from sensors, dependent on the number of people for which activities are recognized: either for **single users**, **groups**, or **multiple users**. Data mining and machine learning techniques are used to model human activities [Li 14]. These currently widely discussed topics, as well as the growing amount of sensor data and calculation power in smartphones make it possible to model human activities even in a mobile environment. It is therefore under research how ubiquitous computing can be better adapted to users' needs.

Usually, the noisy input data is analyzed step by step on several levels: First, with the help of statistical learning it is aimed to detect humans and find their **locations**. On the next level, statistical inference recognizes human **activities** extracted from tracking their location sequences and environmental conditions. On the highest semantic level, statistical and logical reasoning find and evaluate the **goals** of humans extracted from the activity sequences.

The approach of **logic and reasoning** keeps track of all explanations for observed actions which are logically consistent and therefore considers all possible plans. At **probabilistic reasoning**, probability theory and statistical learning are used to draw conclusions on actions, plans and goals under conditions of uncertainty. Research focuses, amongst others, on the identification of individual humans performing routine daily activities based on

machine learning [Hodg 07], the use of sensors to detect human plans, and on finding out the transportation mode of a user via RFID or GPS. New methods based on **data mining** define a pattern-based classification problem and attempt to describe significant changes between any two activity classes of data, so that sequential, interleaved and concurrent activities can be recognized in a unified solution [Gu 09]. Another approach in this area is to use two-dimensional corners (for space and time) to stage a process that hierarchically learns the most distinctive and descriptive features of an action or activity in an efficient way [Gilb 11].

There are several **applications** within fields related to computer science, where personalized support for the user is offered, e.g. human-computer interaction, as well as in other disciplines, e.g. medicine. Practical applications can be also found in user interface design, robot learning and surveillance. Current research tries to use depth cameras like the Microsoft Kinect to build real-time human models and recognize activities which had previously been unknown [Piya 13]. Steps in the process of vision-based activity recognition include detection of humans, tracking them, recognizing their activities, and eventually evaluating them. Further applications are also the assistance of sick and disabled people [Poll 03], security-related topics, logistics support and location-based services.

2.4.2 Object Detection and Recognition

Object recognition tries to find and identify objects in an image or video sequence. It is a challenging area for automated evaluation through computers, because objects are usually visible from different view points, they have varying sizes, scales or rotations, and the view might be partially obstructed. Exemplaric **applications** include face detection, content-based image indexing and visual positioning and tracking [Amit 14]. Methods to detect and recognize objects can be categorized into the following [Labe 14]:

1. One approach is to create models of the objects similar to **computer-aided design** (CAD), which can also be used to recognize a model by parts.
2. **Appearance-based** methods use example images, so-called templates, to target the circumstance that varying conditions make the objects look differently, e.g. lighting, viewing direction or shape. Exemplary methods are edge, greyscale and gradient matching, divide-and-conquer search, histograms, or large modelbases.
3. **Feature-based** methods use searches to find matches between object and image features; a single position of an object must account for all feasible matches. Methods extracting features from objects and images include surface patches, corners, and

linear edges. Sub-categories of feature-based recognition are interpretation trees, hypothesize and test, pose consistency, pose clustering, invariance, geometric hashing, scale-invariant feature transform, speeded up robust features and word bags.

4. **Genetic algorithms** are the newest development of research and reach extremely high accuracy [Lill 13]. Their advantages are that they work even without knowing a given dataset beforehand and that they can create recognition procedures on their own.

Table 2.3 summarizes the mentioned features of activity and object recognition.

AR number of users (2.4.1)	**AR detection levels** (2.4.1)	**AR approaches** (2.4.1)	**OR methods** (2.4.2)
- Single user - Group - Multiple users	- Locations - Activities - Goals	- Logic and reasoning - Probabilistic reasoning - Data mining	- CAD - Appearance-based - Feature-based - Genetic algorithms

Table 2.3: Features of activity and object recognition

2.4.3 Emotion Recognition

Emotion recognition identifies human emotion, usually from interpreting facial expressions from images. In emotion recognition performed by computers, techniques from signal processing and machine learning are used. It can be aimed to help companies sell products more effectively as well as to predict attitudes and actions. Further **applications** include real-time recognition of human emotions and marketing research. In embedded systems like cars, emotion recognition can for instance be used to measure the attention span of its users [McDu 13].

However, the application of emotion recognition is not limited to pure computer vision tasks such as the interpretation of facial expressions, but can be used in several other areas as well, as will be explained in more detail later on in the section about emotion analysis. This is also influenced and brought forward by recent developments in the field of egocentric vision, which is described in the following.

2.5 Egocentric Vision

Egocentric or first-person vision provides a human-centric perspective of the visual world. It includes ways of using, processing and evaluating images or videos taken from the user perspective. Often egocentric cameras are mounted on the user, for instance on his head; in this way, they can easily gather visual information from everyday human interactions. Egocentric images and videos from wearable cameras enable a new top-view perspective on actions. The area is highly related to and a part of computer vision and can have a positive impact on tasks from this field, such as visual detection, recognition, prediction and behavior analysis.

A practical example showing the influence of egocentric vision and its potential for activity and object recognition is shown in [Nguy 16]. The paper provides a literature review on the video-based recognition of activities of daily living (ADL), which is useful in ambient assisted living systems to support the independent living of older people. The authors claim that the main problem with classic camera-based systems is the limited field of view, and that current research on the recognition of these activities is mainly focusing on recognizing objects present in the scene, especially those associated with specific tasks. However, the problem with object-based approaches is their bad performance in unconstrained scenarios, i.e. in everyday situations during which the setting is not like in a constrained lab where subjects execute a certain set of activities always in the same environment; instead, they act in different environments. Wearable cameras can help addressing this issue by recognizing human behavior following a hierarchical structure, during which every level contains more semantic info and a longer time span: At the beginning, human **motion** is detected, which is done by finding the area of interest through eye tracking and detecting objects and hands. Afterwards, the **action** is defined through feature extraction and classification, and the **activity** as a combination or sequence of actions. Eventually, a certain human **behavior** is inferred. To verify the success rates, such techniques can be applied on different datasets.

[Raja 18] goes one step further: The authors claim that nowadays connected computing devices feature various sensing capabilities which can be used to gain insights about individual activities, locations, and social connections. Beyond that, they see the potential to not only track and link simple activities, but also detect sentiment from environmental, on-body and smartphone sensors. An affect map to gain data about emotion and mood of humans from these sensors is proposed, enabling the recognition of emotion and mood and large-scale affect sensing.

By taking advantage of the first-person point-of-view paradigm, there has been a lot of other progress recently in areas like personalized video summarization, activity analysis with inside-out cameras (a camera to capture eye gaze in combination with an outward-looking camera), recognizing human interactions and modeling focus of attention. In the 2016 "Workshop on Egocentric (First-Person) Vision" [2], many **new innovations and developments** from the area of egocentric vision were introduced and discussed, namely in these three categorized areas:

- **recognition advancements**: activity, hand movements, head gestures in conversations, interactions through pointing gestures, detection of text in stores
- **image and video analysis**: finding topics of images, scene understanding, discovering objects of joint attention, highlighting personal locations of interest, detecting social interaction
- **assistive technologies**: generating and delivering notifications for missing actions or forgotten items, place recognition for augmented reality wayfinding assistive technology, detecting visual and physiological signals

2.6 Emotion Analysis

Emotion recognition, which was mentioned before, can also be used to detect emotions in textual messages, or determine the type of emotion in a picture with any content. Papers examining **textual emotion detection** and **image emotion recognition** shall be briefly introduced in the following. To get a proper **definition of emotion** and what this construct actually means, two theories of psychology which have widely influenced further research in the field of textual and visual emotion analysis are described beforehand.

2.6.1 Definition of Emotion

The first theory is **Russell's Circumplex model of affect** [Posn 05], which uses two dimensions for the categorization of emotions: valence and arousal, or **pleasantness and activation**, on the horizontal resp. vertical axis. The model is illustrated in Figure 2.3. Possible combinations are consequently high pleasantness and high activation (with the emotions alert, excited, elated and happy), high pleasantness and low activity (contented, serene, relaxed, calm), low pleasantness and low activity (sad, depressed, bored), and low

[2] `http://www.cbi.gatech.edu/fpv2016/`

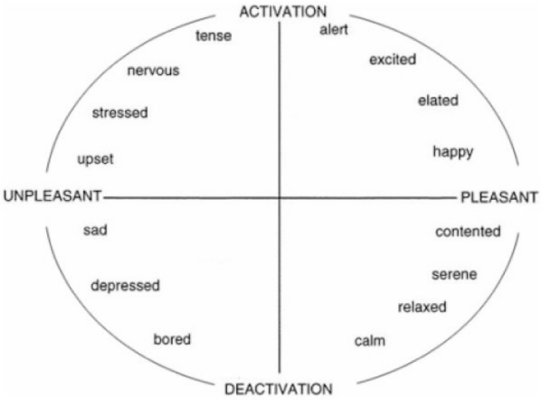


Figure 2.3: Russell's Circumplex model of affect (from [Posn 05])

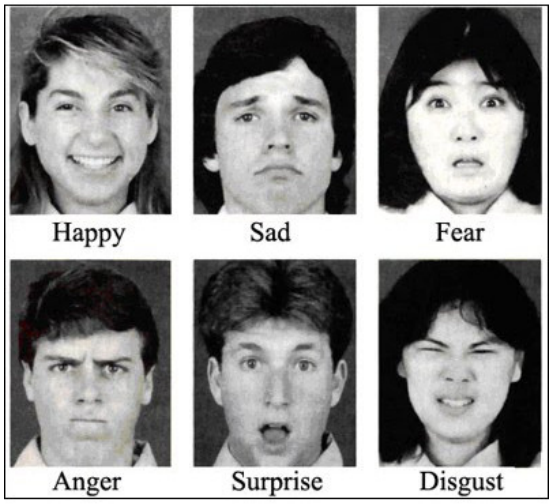


Figure 2.4: Ekman's six basic emotions

pleasantness and high activity (upset, stressed, nervous, tense). The second important model is **Ekman's theory of six basic emotions**: **anger, disgust, fear, happiness, sadness and surprise** [Ekma 99]. This is illustrated in Figure 2.4 (from [3]), showing facial expressions of people, each of which represents one of the basic emotions.

2.6.2 Textual Emotion Detection

[Hasa 14] claims that it has become quite common to express feelings and opinions in social media. Automatically classifying text messages can help to identify anxiety or depression, or measure the mood of both individuals and groups. To model emotional states, the authors use Russell's Circumplex model. Twitter messages are used as the input data, and the hashtags contained in them as labels. A database of over 130.000 tweets is analyzed, tweets are investigated considering features like used vocabulary, emoticons, punctuation and negations. A classifier is trained and different machine learning algorithms are applied to categorize the tweets and assign hashtags to one of the emotion classes Happy-Active, Happy-Inactive, Unhappy-Active, and Unhappy-Inactive, reaching high precision.

Similarly, [Robe 12] uses Twitter as the source of analyzable text. The authors point out the special feature of micro-blogging compared to other sources of text that personal emotions are expressed in a different way there, because they are shared on a frequent base and the messages have to be put into a strict length limit, as this is the case for

[3] `https://sites.google.com/site/societyemblems/`

tweets (140 characters limit). In addition to the six basic emotions defined by Ekman, love is supplemented. The paper creates a system that discovers and identifies these emotions within tweets and analyzes their distribution within a large number of text. It is found out that disgust and happiness are the most commonly used emotions in tweets, while love or hate letters as well as suicide notes, which have been investigated in other studies, are dominated by different emotions. Many semantic categories for both emotions and discussed topics (animals, politics, religion, current news, etc.) are defined, the linguistic style and features are extensively researched.

[Sutt 13] analyzes a database of six million tweets regarding their emotion. The described model selects bipolar pairs of emotions from the "Wheel of emotions" defined by Plutchik; the four pairs are joy and sadness, anticipation and surprise, anger and fear, trust and disgust. All tweets are labeled automatically - in addition to hashtags and emoticons, emojis are considered as well - and these labels are combined to train classifiers.

Whissell's "Dictionary of Affect in Language" [Whis 09] was created to measure emotions in any verbal material, like freely produced text, word lists or literature passages. It originally contained over 4.000 specifically emotional words, each word in the dictionary being accompanied by two scores - rated by humans - for the affective dimensions of pleasantness and activation, as defined by Russell. Low scores mean that the word has a rather unpleasant resp. passive connotation, high scores mean that the word has a rather pleasant resp. active connotation. Later, the dictionary was revised to include 8.742 words, which increased its applicability for analyzing natural language, reaching an overall matching rate of 90%. Besides pleasantness and activation, a third dimension called imagery, describing how hard, resp. easy it is to form a mental picture of a word, was added.

2.6.3 Image Emotion Recognition

[You 16] aims to show means how to predict the emotional reaction of people towards images, using computer vision techniques and Convolutional Neural Networks (CNN). The paper builds a large image data set with photos from Instagram and Flickr, using hashtags as weak labels for eight emotions, which are similar to Ekman's basic emotions: amusement, anger, awe, contentment, disgust, excitement, fear and sadness. Pictures with more than one emotion are removed, duplicates are deleted, and humans are utilized to verify the labels. In this way, over 23.000 photos are used to train two neural networks: the first by fine-tuning an existing pre-trained model with the verified pictures, and the second by fine-tuning another, using the weakly labeled pictures from the beginning. After

the calculation of the convolutional matrices, a randomized accuracy test shows that the first network recognizes over 58% of the pictures tagged by humans correct, outperforming the 32% of the existing pre-trained model. For three public datasets of exemplary photos, the accuracy rate is even over 80% for most of the emotions.

[Mach 10] uses theoretical and empirical concepts from psychology and art for the task of image emotion classification. Various low-level features of the images, such as color (e.g. saturation, brightness), texture, image composition (e.g. level of detail, dynamics), and image content (e.g. faces, visible skin), are extracted and combined to represent the emotional content of an image. The eight emotions used for classification are the same as in [You 16].

[Kim 17] uses Deep Neural Networks (DNN) to analyze object, background and semantic information of images and predict emotions based on it. The paper uses the Circumplex model of Russell; all pictures receive a rating for each of the two dimensions, valence and arousal, on a scale from one to nine. As the source of data, the authors use both pictures from Flickr tagged with emotion keywords, and pictures from the above introduced paper [You 16], adding up to a total of over 10.000 images in the database. Again, humans complete the task of emotion classification, which shows that overall more positive than negative images are shared. After extracting color, local, object and semantic features, the paper is stating that especially the contained object is higly correlated with the emotion expressed in a picture. The suggested learning emotion prediction model based on DNN performs scene segmentation, object classification, and takes low-level image features into account, reaching very high accuracies of over 90% for the rating of both valence and arousal. An example is shown in Figure 2.5.

Figure 2.5: Example pictures with predicted and ground truth values for valence resp. arousal, and resulting accuracy (from [Kim 17])

Chapter 3

Design of an Emotion-based Recommender System

In this chapter, the conceptional architecture, i.e. prototype, of a recommender system for city visitors is designed, described and implemented, including visualizations and implementation specifications for its components, like used programming languages, libraries, tools and other technical details, adding up to a full documentation. The aim is to get a specified model of how to implement a recommender system based on all the mentioned details, and to explain step by step how the implementation of the design is performed. The organization and design of the system are clarified by extracts of the source code and supportive graphics, which show the procedures within the system, and furthermore some screenshots to graphically illustrate the system design.

As a general **idea**, the system consists of two parts, which are communicating with each other with appropriate tools: A Python-based **backend**, which collects the pictures to be recommended and calculates the recommendations within the recommender system, and a HTML- and JavaScript-based **frontend**, which displays the data in an accessible way. The recommender system fetches various data and analyzes them via algorithms to generate personalized recommendations for places to see around the user's location, taking additionally his emotion into account. These recommendations are then shown to the user on the displayed website. The connection between frontend and backend is realized via a Python-based server.

At first, the **data** and information required for the recommender system is discussed and sources for its collection compared. Images serve as one part of the input for the recommendations - in particular public, geo-tagged pictures from Flickr, and self-made

pictures. Each picture represents a point of interest. From the photos, the information associated with them is extracted - *date, time taken, and geographical position* - and their emotion is analyzed by using both an API for image content analysis to receive a textual content description, and an emotion dictionary to calculate the emotion values. To gain insights about the picture databases that were built up in this way, a few diagrams show information about the pictures and its contents.

In the following, the techniques used in the **recommender system** are described and discussed. The personal data of the user itself is taken into account as well: His geographical position and his current emotion, i.e. mood, are considered - according to the personal state, the system can therefore customize the personal suggestions. The system analyzes this data together with the gathered data from the pictures to find out the places which are closest to the user and which fit best to his own emotion, in this way generating personalized recommendations. There are many different aspects which have to be considered in this section, including used algorithms, libraries and data structures.

Eventually, the graphical **presentation** of the recommendations takes part in a user-friendly way - after all, the recommender system is there to help and assist the user and present him the suggestions and a connection between them. Therefore, the user needs to be provided with explanations for using the system. This section informs about the used technologies and frameworks, such as Leaflet and Flask, how the user can provide his own emotion and create an own picture database, and how the recommendations are displayed in detail.

3.1 Data: Images

To collect **data** which can be used for the recommendations, databases of geotagged photos initially needed to be built up. These photos had to be processed step by step: First, their content needed to be analyzed, then they had to be categorized into, i.e. tagged with emotions. Eventually, all information was stored in CSV files. The whole process is done with code in the programming language Python.

3.1.1 Sources for Collecting Images

As a first step, images that are used to calculate the recommendations had to be gathered from appropriate sources. The main data sources for the system are photos with geotags of locations for which recommendations are performed. Therefore databases of photos

from online services - in this case, **Flickr** was selected - and of own pictures had to be built up, at the same time ensuring that all relevant information about the photos was recorded accordingly.

3.1.1.1 Flickr

The popular image sharing service Flickr offers an API [1] which has basically no restrictions and gives simple and fast access to images uploaded on the website and their respective metadata. Under [2] a map can be accessed which shows uploaded Flickr photos in regard of their geographic location. As of 14.02.2018, searching e.g. Helsinki returns a number of 107.306 geotagged photos, Munich returns 500.748 photos. The API can be tried out under [3], but to permanently be able to access the API, an API key needs to be obtained [4], hereafter referred to as `FLICKR_API_KEY`.

In the following, it is demonstrated how three different picture databases were built up. For each database, one example picture is selected and shown how it was analyzed and tagged step by step until all needed information was ready.

1. **flickr.csv** will contain information about pictures from Flickr taken in and around Helsinki.
2. **local.csv** will contain information about pictures from Flickr taken in and around Munich, which are all downloaded and stored locally.
3. **own.csv** will contain information about pictures taken by myself.

Figure 3.1: Picture from flickr.csv showing Esplanadi in Helsinki

The following parameters and its respective values were used for the API request to fetch the 250 pictures for the database `flickr.csv`. The detailed API documentation can be found at [5]. Comments explain the meaning of each parameter.

[1] `https://www.flickr.com/services/api/`
[2] `https://www.flickr.com/map`
[3] `https://www.flickr.com/services/api/explore/flickr.photos.search`
[4] `https://www.w3resource.com/API/flickr/tutorial.php`
[5] `https://www.flickr.com/services/api/flickr.photos.search.html`

- **sort: date-taken-desc** //to get recent pictures, sorted by date
- **has_geo: 1** //to get only geotagged pictures
- **lat: 60.167665** //latitude coordinates of the selected location (Helsinki, Kauppatori)
- **lon: 24.953678** //longitude coordinates of the selected location
- **radius: 20** //radius in kilometres around the selected location for which pictures shall be returned
- **extras: date_taken,geo,url_l** //to get the information about the date and time at which the photo was taken, the geotags, and the URL for the picture in large size (maximum dimension for width and height: 1024 pixels)
- **per_page: 250** //number of returned pictures; 250 is the maximum, because the API request is a geo query
- **format: rest** //to get a response in XML format
- **page: 1** //to get the first page of results

The following URL is generated when the aforementioned parameters are used: `https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=FLICKR_API_KEY&sort=date-taken-desc&has_geo=1&lat=60.167665&lon=24.953678&radius=20&extras=date_taken,geo,url_l&per_page=250&format=rest&page=1`

The Python code 3.1 gets the response from the Flickr API request for the provided parameters `lat` (latitude) and `lng` (longitude).

```
import sys
import requests
lat = sys.argv[1]
lng = sys.argv[2]
picNumber = 100
url = 'https://api.flickr.com/services/rest/?method=flickr.photos.search
   &api_key='+FLICKR_API_KEY+'&sort=date-taken-desc&has_geo=1&lat='+str(
   lat)+'&lon='+str(lng)+'&radius=20&extras=date_taken,geo,url_l&
   per_page='+str(picNumber)+'&format=rest&page=1'
response = requests.get(url)
```

Listing 3.1: get response from Flickr API request

Listing 3.2 shows a part of the XML response containing information about the picture 3.1.

```
<rsp stat="ok">
<photos page="1" pages="1628" perpage="250" total="406975">
  [...]
  <photo id="26135474108" owner="110842082@N02" secret="4bf45ef66f"
     server="4752" farm="5" title="Helsinki" ispublic="1" isfriend="0"
     isfamily="0" datetaken="2018-01-25 11:27:21" datetakengranularity="
     0" datetakenunknown="0" latitude="60.167492" longitude="24.946827"
     accuracy="16" context="0" place_id="GdRw9F5TUb15LVJ0hw" woeid="
     23701411" geo_is_family="0" geo_is_friend="0" geo_is_contact="0"
     geo_is_public="1" url_l="https://farm5.staticflickr.com
     /4752/26135474108_4bf45ef66f_b.jpg" height_l="683" width_l="1024"/>
  [...]
</photos>
</rsp>
```

Listing 3.2: example Flickr API response

The Python code 3.3 parses the response as XML; in case the specified number of pictures (`picNumber`) is not reached (e.g. in places like at the sea, where no pictures have been taken), it is stopped. Hereafter, unnecessary attributes which are not needed for the further processing are removed. The attributes are sorted alphabetically (and therefore ordered differently from the original API call). For few photos, some attributes are not present and therefore it needs to be checked whether they exist. For a very small amount of especially older photos, the API also does not return a link; in this case, an URL pointing to an empty picture with 1x1 pixels will be used later.

```
import xml.etree.ElementTree as ET
tree = ET.ElementTree(ET.fromstring(response.content))
root = tree.getroot()
pic = root[0]
if len(pic) < picNumber: sys.exit('Not enough pictures here!')
for i in range(picNumber):
  del pic[i].attrib['id'], [...]
  if 'place_id' in pic[i].attrib: del pic[i].attrib['place_id']
  [...]
  url_l = pic[i].attrib['url_l'] if 'url_l' in pic[i].attrib else 'https
     ://upload.wikimedia.org/wikipedia/en/4/48/Blank.JPG'
```

Listing 3.3: parse response as xml, remove unnecessary attributes

Figure 3.2: Picture from local.csv showing Königsplatz in Munich

Figure 3.3: Picture from own.csv showing nature

The XML keys and values left for the picture 3.1 after this step are as shown in Listing 3.4.

```
<datetaken="2018-01-25 11:27:21" latitude="60.167492" longitude="
   24.946827" url_l="https://farm5.staticflickr.com/4752/26135474108
   _4bf45ef66f_b.jpg"/>
```

Listing 3.4: example XML keys and values left

Secondly, the database `local.csv` of Flickr pictures around the Marienplatz, the centre of Munich, was created. The used parameters and values were the same as above, except for different coordinates (**lat: 48.137079**, **lon: 11.576006**), a smaller **radius** of 10 kilometres, and **page** values from 1 to 8, so that a total of eight requests to the Flickr API was executed to collect altogether 2.000 pictures. The selection of pictures was refined: As many Flickr users upload batches of pictures taken at the same location, these position duplicates were removed to reach a broader variety. The results without links were removed, too, as were the pictures (falsely returned by the API) with a date older than 2017, coming down to a number of 736 pictures. Some inappropriate pictures like such with persons, certain indoor pictures, etc. were removed as well, so that a total number of 642 pictures was reached in the end. The remaining pictures were now sorted alphabetically, after the full link to the images, and downloaded locally, so that the link prefix `https://farm5.staticflickr.com/4xxx/` did not have to be remembered any more.

The known information about the picture 3.2 so far are its parameters **file name** `24842660977_e86c0e0ba2_b.jpg`, **date taken** `2017-12-29 15:04:57`, and the position coordinates **latitude** `48.146632` and **longitude** `11.565728`.

The third database that was created is one of my own pictures, taken between 1.9.2016 and 1.1.2018; it consists of 4.406 pictures. The tool BR's EXIFextracter 0.9.14 Beta [6] was used to extract the EXIF data for each picture. In addition to the file name, the following attributes were selected for the CSV export: `Date, Time, GPS latitude, GPS longitude`.

In the resulting CSV `own-untagged.csv`, a line containing information about the picture 3.3 looks like this: `IMG_2359.JPG,2016:12:31,12:32:23,49.463844,10.565248`

3.1.1.2 Instagram

To get the most recent photos, the **Instagram** API could have been used as an additional source of photos. Therefore an access token to use the Instagram API was created [7]. However, since 01.10.2017, it is not possible any more to apply for the access of the needed API endpoint, `GET /media/search` [8], and search for recent photos in a given area. The required scope "public_content" is not awarded any more for new login permissions applications [9]. Instead, the Instagram Graph API was newly opened, which however is only targeted for business users and accounts.

3.1.2 Image Content Analysis

After collecting the pictures and parsing the API response into XML for further processing, the images needed to be analyzed. The aim was to get a text output of the contents of the image for each photo, thus requiring image analysis. A closer look is taken on the solutions provided by **Google** [10] and especially **Clarifai** [11], which eventually was used. Other alternatives are Amazon Rekognition [12], Microsoft Azure Cognitive Services Computer Vision API [13], and IBM Watson Visual Recognition [14].

[6] `http://www.br-software.com/extracter.html`
[7] `https://elfsight.com/blog/2016/05/how-to-get-instagram-access-token/`
[8] `https://www.instagram.com/developer/endpoints/media/`
[9] `https://www.instagram.com/developer/authorization/`
[10] `https://cloud.google.com/vision/`
[11] `https://clarifai.com/`
[12] `https://aws.amazon.com/rekognition/`
[13] `https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/`
[14] `https://www.ibm.com/watson/services/visual-recognition/`

3.1.2.1 Google Cloud Vision API

The Google Cloud Vision API offers a broad variety of features based on machine learning to analyze pictures, including label and landmark detection. It can be accessed for instance via cURL, Python or Java [15]. Usage examples provided by Google can be found under [16]; GitHub projects do also exist, such as annotating images based on Python [17] or [18]. Due to the limitation of only 1500 free requests per month, and because the API does not guarantee that a certain number of keywords, i.e. tags, is found when a picture is analyzed, this API was not chosen.

3.1.2.2 Clarifai

Clarifai is another provider for image content analysis. It was selected due to its many advantages, a simple to use JSON API [19], 5000 free requests per billing cycle, i.e. month, and the visual recognition feature that delivers a high number of accurate, precise textual predictions (keywords) for the contents of a picture, called "concepts". A tutorial can be found under [20], the full API reference is at [21]. Batch processing of photos is possible as explained in [22]; here, a custom solution is used instead. The API supports calls via various methods, e.g. cURL or Python.

To use it in Python, the package `clarifai` [23] needs to be installed, on Windows along with the required Microsoft Visual C++ Build Tools [24], including Windows 8/8.1 SDK. Hereafter, an API key was obtained, in the following referred to as `CLARIFAI_API_KEY`. The API uses so-called models for the prediction of image content: Interesting models for this use case are the **General** model [25], which is able to recognize over 11.000 different concepts, and the **Travel** model [26]. The only discovered restriction of the API is the

[15] https://cloud.google.com/community/tutorials/make-an-http-request-to-the-cloud-vision-api-from-java

[16] https://cloud.google.com/vision/docs/all-samples

[17] https://github.com/nchah/google-vision-api-image-annotate

[18] https://github.com/Hironsan/google-vision-sampler

[19] https://clarifai.com/developer/quick-start/

[20] https://sdk.clarifai.com/python/docs/latest/tutorial.html

[21] https://sdk.clarifai.com/python/docs/latest/clarifai.rest.html

[22] http://help.clarifai.com/api/batch-processing-with-python

[23] https://github.com/Clarifai/clarifai-python

[24] https://go.microsoft.com/fwlink/?LinkId=691126

[25] https://clarifai.com/models/general-image-recognition-model-aaa03c23b3724a16a56b629203edc62c

[26] https://clarifai.com/models/travel-image-recognition-model-eee28c313d69466f836ab83287a54ed9

maximum uploadable image size of 10485760 bytes (10 Megabytes), which is not a problem though, since the images from the Flickr API are, as discussed, requested in the "large" resolution, where the approximate file size is way smaller (for instance, the average file size of the database `flickr.csv` is about 250 Kilobytes).

The Python code 3.5 gets the response from the Clarifai API request, using the General model, limits the amount of concepts being returned (`conceptsNumber`) to 10 (this number is reached for every picture), and parses it as JSON.

```
def photoToEmotion(photo):
  import json
  from clarifai.rest import ClarifaiApp
  conceptsNumber = 10
  cApp = ClarifaiApp(api_key = CLARIFAI_API_KEY)
  model = cApp.models.get('general-v1.3')
  result = model.predict_by_url(photo, max_concepts = conceptsNumber)
  datax = json.dumps(result)
  data = json.loads(datax)
```

Listing 3.5: get response from Clarifai API request with concepts (limit maximum number) and parse as json

Listing 3.6 shows an example of a JSON output for the previously shown example picture 3.1 of the database `flickr.csv`. It takes a few seconds for each picture until the response is received.

```
{ [...]
  "outputs": [
    { [...]
      "data": {
        "concepts": [
          {
            "id": "ai_FWCjC8jZ",
            "name": "architecture",
            "value": 0.9876021,
            "app_id": "main"
          }, [...]
          {
            "id": "ai_m8rrtkhG",
            "name": "town",
            "value": 0.9265938,
            "app_id": "main"
          } [...]
}
```

Listing 3.6: example JSON output

The Python code 3.7 selects the concepts from the JSON response. The array `answers` will store all information collected in the further process of analyzing the picture. In this way, the ten predicted concepts which most likely represent the contents of the picture are saved, sorted by probability. The probability values are not used further.

```
concepts = ''
conceptsList = ''
answers = []
for i in range(conceptsNumber-1):
  concepts += data['outputs'][0]['data']['concepts'][i]['name']+'\n'
  conceptsList += data['outputs'][0]['data']['concepts'][i]['name']+', '
concepts += data['outputs'][0]['data']['concepts'][conceptsNumber-1]['
   name']
conceptsList += data['outputs'][0]['data']['concepts'][conceptsNumber
   -1]['name']
answers.append(conceptsList)
```

Listing 3.7: select concepts from json response

The concepts returned by the API are **architecture, building, winter, city, travel, snow, outdoors, sight, old, town** for the picture 3.1, **architecture, column, travel, monument, building, no person, neoclassical, city, marble, sculpture** for the picture 3.2 and **no person, water, landscape, outdoors, tree, winter, grass, sky, daylight, lake** for the picture 3.3.

The Travel model of the Clarifai API returned only a very small number of often improper concepts and is therefore not considered further.

3.1.3 Sentiment and Emotion

To analyze the concepts which have been returned for every image, an appropriate metric needed to be used. The task was now to assign emotions to pictures. There are two main approaches for rating the emotional content of text, namely **sentiment analysis** and **emotion analysis**, which are briefly introduced in the following.

3.1.3.1 Sentiment Analysis

One approach is sentiment analysis, or opinion mining, which aims to determine the attitude of someone regarding a certain topic or the overall contextual polarity or emotional reaction towards textual content, a product, or something else. The attitude can be a judgment, an evaluation, or an affective, i.e. emotional state. The technique is part of "Natural Language Processing" and often used in the context of recommender systems. On many social networks, online media services, or e-commerce websites, users can provide reviews, survey responses, comments or feedback to items and products. These user-generated texts provide a good source of user's sentiment opinions about the items and products. Applying sentiment analysis on this material can reveal both the related features or aspects of the item, and the users' sentiments on each of the features.

Sentiment analysis therefore aims to find out the positive or negative polarity, i.e. meaningfulness of words. Many tools are listed under [27]; a lot of approaches are based on lists that assign a score between -1 and 1 to each word in terms of its polarity, objectivity, or subjectivity. For instance, [28] features an opinion lexicon that inclused lists with around 2000 positive words and 4800 negative words.

Current research studies the sentiment of emojis, which became sort of the successors of emoticons [Nova 15]. After labeling and analyzing 1.6 million tweets, an emoji sentiment lexicon is provided, assigning polarity scores (positivity, negativity and neutrality) for each emoji. The paper additionally finds out that most emojis express a positive sentiment. [29] shows the positive and negative sentiment score of many emojis. The creators of [30] tune a neural network model to perform automatic visual sentiment analysis of images, i.e. to rate whether their content is rather positive or negative, in their academic paper.

[27] https://medium.com/@datamonsters/sentiment-analysis-tools-overview-part-1-positive-and-negative-words-databases-ae35431a470c

[28] https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html

[29] https://www.npmjs.com/package/emoji-emotion

[30] https://github.com/imatge-upc/sentiment-2017-imavis

3.1.3.2 Emotion Analysis

Several **APIs** analyze text inputs and return values of the basic emotions contained in them as an output. The article [31] mentions many, and e.g. refers to Qemotion, PreCeive and Indico. Sentic [32] and ParallelDots [33] are other examples. At [34], this was done using a machine learning toolkit.

Another method is to use **dictionaries**. DepecheMood [35] contains 37.000 English words; every word was automatically assigned a value for each of the eight emotional values *afraid, amused, angry, annoyed, dont_care, happy, inspired and sad*, adding up to 1 (100%). Some words appear more than once, if they can be written in the same way, but as different word types, e.g. the noun and verb "train". Under the webpage [36] many interesting dictionaries are listed, such as the Affect Intensity Lexicon, containing information about words and their associations with emotions, feelings, or attitudes.

Eventually, "Whissell's Dictionary of Affect in Language" [37] from [Whis 09] was selected, as it is multi-dimensional and was created in an understandable, comprehensive and scientifically documented way. It contains 8.742 English words; every word has a rating on a scale from one (lowest score, corresponds to 0%) to three (highest score, corresponds to 100%) for each of the three parameters **pleasure, activity and imagery**. The file `dictionary_English.txt` from the installed program was used as the emotion dictionary `dict.txt`; the first lines of the file containing explanations about the dictionary were removed, to enable a line-by-line searching through Python.

The Python code 3.8 looks into the dictionary and calculates the average of the values for pleasure (`p`) and activity (`a`). The last parameter, imagery, is not used in this system. As can be seen from the concepts before, the Clarifai API sometimes returns concepts consisting of two words, e.g. "no person" or "transportation system". In these cases, only the first word is checked, i.e. "no" or "transportation", because the dictionary only contains single words in each line. The code checks for each concept if the word is present in the dictionary, and adds its respective pleasure and activity values to `p` and `a`. In the end, the average value is calculated by dividing the total score of `p` and `a` through the number of hits (`counter`) and rounding it by four decimal digits. For none of the pictures

[31] `https://medium.com/@sifium/top-five-emotional-sentiment-analysis-apis-116cd8d42055`
[32] `http://sentic.net/api/`
[33] `https://www.paralleldots.com/text-analysis-apis#emotion`
[34] `https://www.microsoft.com/developerblog/2015/11/29/emotion-detection-and-recognition-from-text-using-deep-learning/`
[35] `https://github.com/marcoguerini/DepecheMood/releases`
[36] `http://saifmohammad.com/WebPages/lexicons.html`
[37] `https://www.god-helmet.com/wp/whissel-dictionary-of-affect/index.htm`

word	pleasure	activity	imagery
architecture	1.8333	1.8333	2.4
building	1.8571	1.8750	3.0
winter	2.3333	2.3750	3.0
city	2.1000	2.3000	2.6
travel	2.5714	3.0000	1.6
snow	2.3333	1.4000	3.0
outdoors	not found	not found	not found
sight	2.2857	1.7500	1.8
old	1.5000	1.2000	2.4
town	2.1667	1.6000	2.8
∅	2.109	1.9259	not used

Table 3.1: example contents of dict.txt

in the databases the pleasure or activity value is 0, meaning that there were always hits for words. On average, about 70% to 80% of the concepts could be found in the dictionary, making it an appropriate metric for the emotion tagging of photos.

```
counter = 0
p = 0.0000
a = 0.0000
for line in concepts.split('\n'):
  word = line.split(' ', 1)[0]
  with open('dict.txt') as dict:
    for line in dict:
      hit = line.split(' ', 1)[0]
      if word == hit:
        counter += 1
        p += float(line[40:46])
        a += float(line[50:56])
p = 0 if counter == 0 else p/counter
a = 0 if counter == 0 else a/counter
answers.append(str(round(p,4)))
answers.append(str(round(a,4)))
```

Listing 3.8: look into dictionary and calculate average of the values for pleasure and activity

Table 3.1 shows the contents of the dictionary for the example picture 3.1 from `flickr.csv`. The last line shows the average scores for **pleasure** (1.8333 + 1.8571 + 2.3333 + 2.1000 + 2.5714 + 2.3333 + 2.2857 + 1.5000 + 2.1667) / 9 = 2.109 and **activity** (1.8333 + 1.8750 + 2.3750 + 2.3000 + 3.0000 + 1.4000 + 1.7500 + 1.2000 + 1.6000) / 9 = 1.9259.

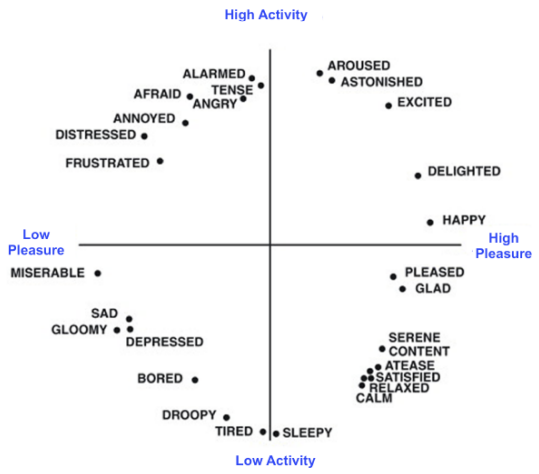


Figure 3.4: 28 emotions, taken from the paper (from [Hasa 14])

tense	High Activity	alert
nervous		excited
stressed		elated
upset		happy
Low Pleasure		High Pleasure
miserable		contented
sad		serene
depressed		relaxed
bored	Low Activity	calm

Figure 3.5: 16 emotions, as assigned in getEmotion()

The Python code 3.9 evaluates the values of **pleasure** and **activity** and assigns an emotion to each picture, based on the paper [Hasa 14], which uses Russell's Circumplex model of affect to categorize emotions. This works in the following way: A total number of 16 of the originally 28 emotions was selected for setting a unique emotion tag for each picture. In this way, all images were categorized into, i.e. tagged with emotions.

```
def getEmotion(p,a):
  if p>=1 and p<2 and a>=2.75 and a<=3:
    return 'tense'
  [...]
  elif p>=2 and p<=3 and a>=1 and a<1.25:
    return 'calm'
  else:
    return 'undefined'
emotion = getEmotion(p,a)
answers.append(emotion)
return answers
```

Listing 3.9: evaluate and get emotion for values of p and a: assign one of 16 affect words as the picture's emotion tag

Figure 3.4 shows the original graphic with all 28 emotions, taken from the paper [Hasa 14]. Figure 3.5 shows how the 16 remaining emotions were assigned by the method `getEmotion()`.

The Python code 3.10 calls the method `photoToEmotion()` specified in the previous code parts for each picture, i.e. URL, and stores all information in the array `final`. It converts the `datetaken` information into the same format as it has been output before by BR's EXIFextracter, when the CSV file containing the information about the pictures from `own.csv` was generated. This leads to databases structured in an equal way. The information about concepts, pleasure, activity and emotion (from `final`) is added to the string `csvLine` and everything is saved as a CSV file, separated with commas. CSV is a proper, suitable format, since its advantages compared to XML are that there is no overhead, parsing is easily and quickly possible, and the created file can be read with any text editor. To notify the user about the progress of the tagging process, a message is shown for each successfully tagged picture.

```
piclib = open('flickr.csv', 'w')
[...]
  final = photoToEmotion(url_l)
  datetaken = pic[i].attrib['datetaken']
  date = datetaken[0:4]+':'+datetaken[5:7]+':'+datetaken[8:10]+','+
     datetaken[11:19]
  csvLine = url_l+','+date+','+pic[i].attrib['latitude']+','+pic[i].
     attrib['longitude']+',\"'+final[0]+'\",'+final[1]+','+final[2]+','+
     final[3]
  if i == picNumber-1: piclib.write(csvLine)
  else: piclib.write(csvLine+'\n')
  print('Picture ' + str((i+1)) + ' tagged!')
print('Finished! Data written to flickr.csv.')
```

Listing 3.10: xml: add concepts, pleasure, activity and emotion, save as csv

The resulting file **xmlParser.py** can be used to create an own Flickr picture database at the selected location with the arguments `latitude` and `longitude`.

The final CSV line of the example picture from `flickr.csv` now looks like this:

```
https://farm5.staticflickr.com/4752/26135474108_4bf45ef66f_b.jpg,
2018:01:25,11:27:21,60.167492,24.946827,"architecture, building, winter,
city, travel, snow, outdoors, sight, old, town",2.109,1.9259,contented
```

The final CSV line of the example picture from `local.csv` now looks like this:

```
24842660977_e86c0e0ba2_b.jpg,2017:12:29,15:04:57,48.146632,11.565728,
"architecture, column, travel, monument, building, no person,
neoclassical, city, marble, sculpture",2.0498,1.8814,contented
```

To tag the pictures of `own-untagged.csv`, a slight modification of the code 3.3 is needed. The Python code 3.11 shows this: The picture database is parsed as a CSV, for every picture the concepts, pleasure, activity and emotion attributes are added (from the array `final`), and everything is saved as a CSV file. The user is notified about each successfully tagged picture. The code 3.10 is not needed any more.

```
import csv
with open('own-untagged.csv', 'r', newline='') as csvIn, open('own.csv',
    'w') as csvOut:
  database = csv.reader(csvIn)
  writer = csv.writer(csvOut, lineterminator='\n')
  for count, row in enumerate(database):
    final = photoToEmotion('img/'+row[0])
    row.append(final[0])
    row.append(final[1])
    row.append(final[2])
    row.append(final[3])
    writer.writerow(row)
    print('Picture ' + str((count+1)) + ' tagged!')
print('Finished! Data written to own.csv.')
```

Listing 3.11: csv: add concepts, pleasure, activity and emotion, save as csv

The resulting file **csvParser.py** can be used for picture databases with pictures stored locally. It does not need any arguments to be used. The method `photoToEmotion()` is identical with the one used in **xmlParser.py**, except for the method called by Clarifai, which is now `predict_by_filename` instead of `predict_by_url`.

The final CSV line of the example picture from `own.csv` now looks like this:

```
IMG_2359.JPG,2016:12:31,12:32:23,49.463844,10.565248,
"no person, water, landscape, outdoors, tree, winter,
grass, sky, daylight, lake",2.3582,1.7557,contented
```

3.1.4 Database Analysis

The solution of the code 3.9, where emotions were assigned to pictures, assumed that there are 16 categories of emotions, with the borders between them being clearly defined. To verify if this is an acceptable solution, the picture databases `local.csv` and `own.csv` are analyzed in detail, showing the distribution of the values for **pleasure** and **activity** for over 5.000 pictures in various ways.

The first four plots have been created with Microsoft Excel.

Figure 3.6 and Figure 3.7 visualize the pleasure and activity values of all pictures from the databases `local.csv` resp. `own.csv`. Every dot represents one picture. It is quickly visible that most of the values are in the inner part of the plot, thus showing that extremely high or low values are almost not present at all.

To see the more interesting, inner part better, it is zoomed there, i.e. only showing values in the range from 1.5 to 2.5.

In Figure 3.8 and Figure 3.9 it is clearly visible that for both databases, the right side and the lower half contain more dots than the left side, resp. upper half.

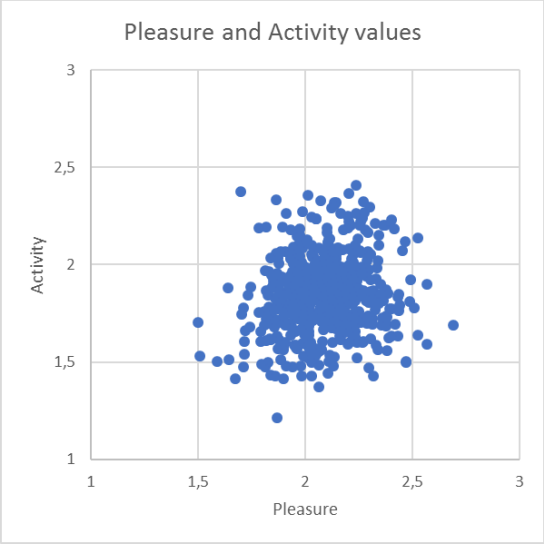


Figure 3.6: Pleasure and activity values from database local.csv

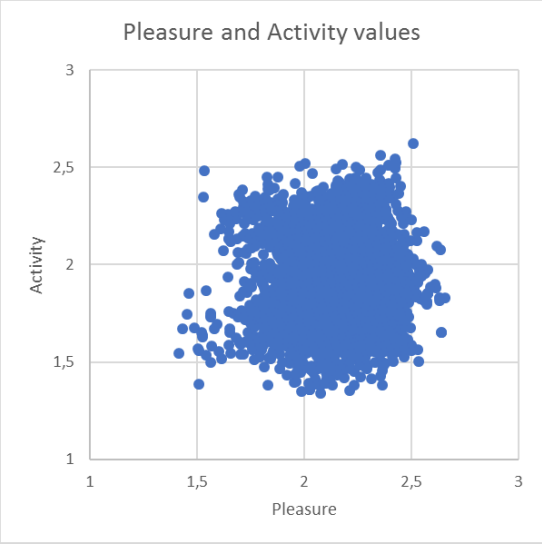


Figure 3.7: Pleasure and activity values from database own.csv

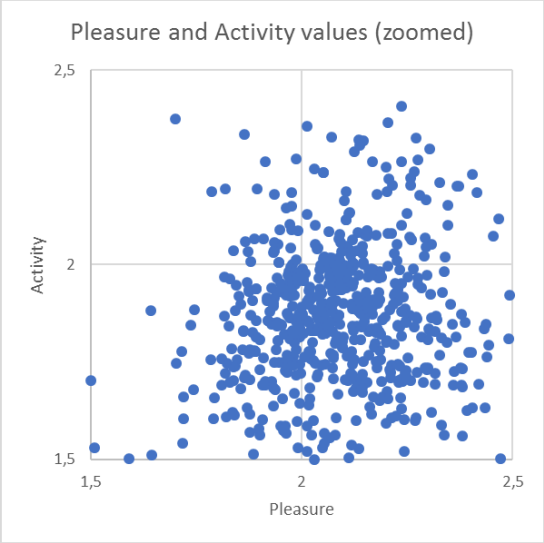


Figure 3.8: Pleasure and activity values from database local.csv (zoomed)

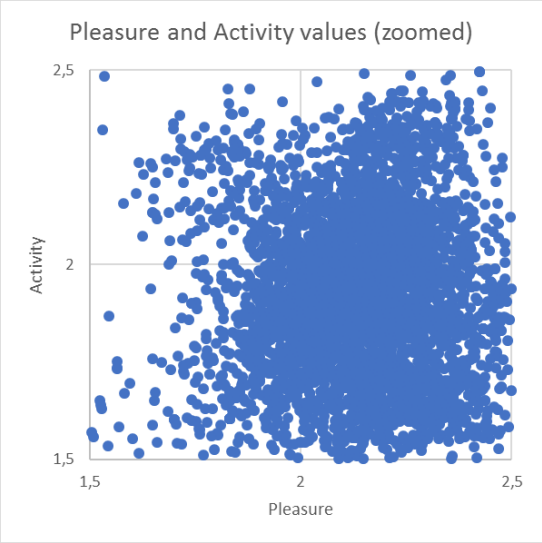


Figure 3.9: Pleasure and activity values from database own.csv (zoomed)

To further investigate this, four two-dimensional histogram plots have been created with Python, using the packages `SciPy` and `Matplotlib` [38], showing the density areas of dots, i.e. pictures.

Figure 3.10 and Figure 3.11 show this tendency for both databases again.

Also at this step, further zooming to the inner part is done to better see the interesting area.

The zoomed two-dimensional histograms Figure 3.12 and Figure 3.13 again visualize the values which are most often present in the databases.

The four one-dimensional histograms, again created with Microsoft Excel, show the distribution of values from another perspective, individually for pleasure and activity.

Figure 3.14 and Figure 3.15 as well as Figure 3.16 and Figure 3.17 show again the result that both databases have average pleasure scores slightly above the middle, and activity scores slightly below the middle (the blue color highlights values below the middle, the red color highlights values above the middle). The tendency to higher pleasure scores can be explained with the fact that people tend to upload and take pictures which are rather positive. Interestingly, both the distribution of pleasure and of activity values is very similar to the results found in [Kim 17].

[38] `https://matplotlib.org/api/pyplot_api.html`

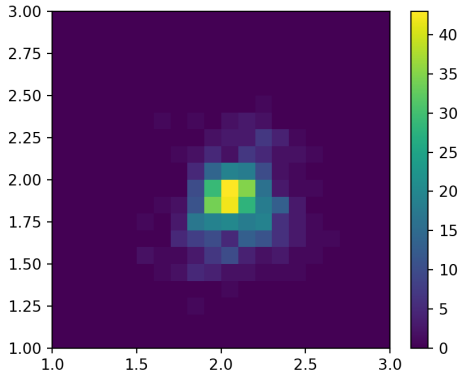


Figure 3.10: 2D histogram from database local.csv

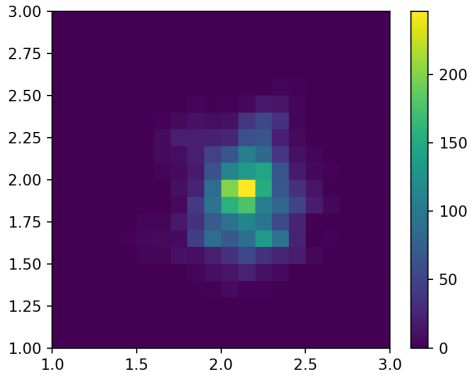


Figure 3.11: 2D histogram from database own.csv

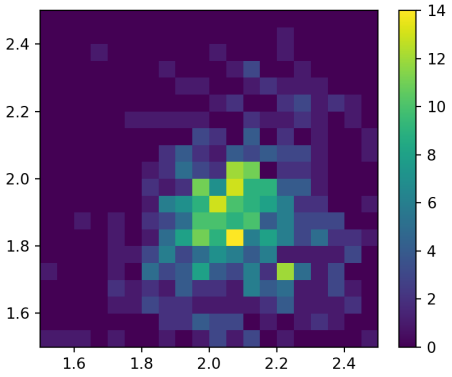


Figure 3.12: 2D histogram from database local.csv (zoomed)

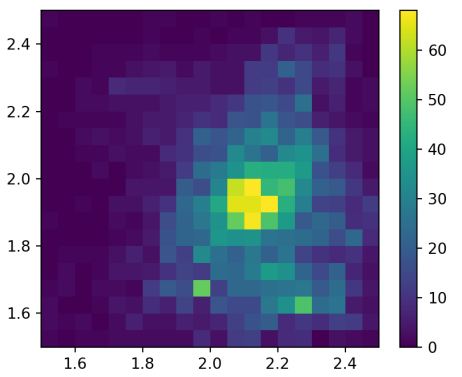


Figure 3.13: 2D histogram from database own.csv (zoomed)

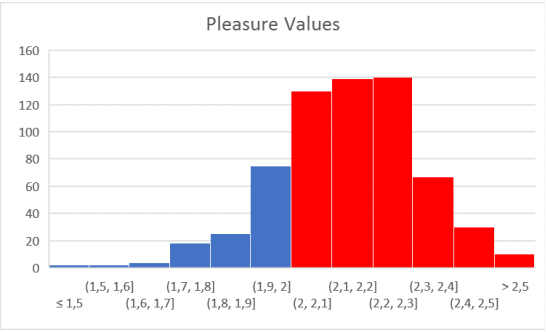


Figure 3.14: Pleasure values from database local.csv

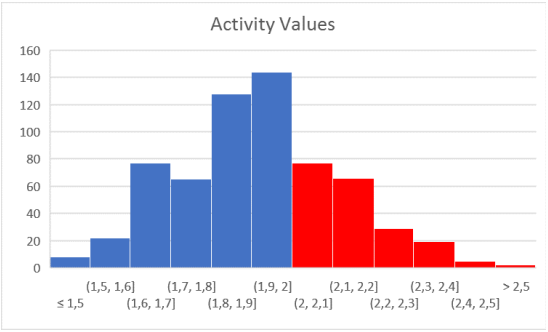


Figure 3.15: Activity values from database local.csv

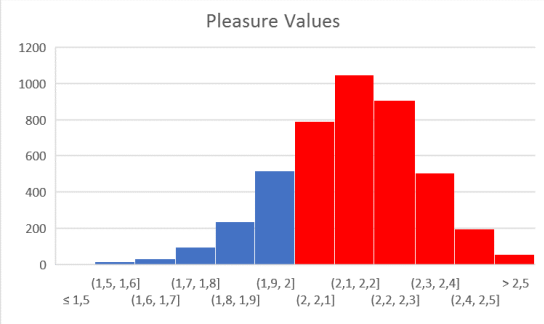


Figure 3.16: Pleasure values from database own.csv

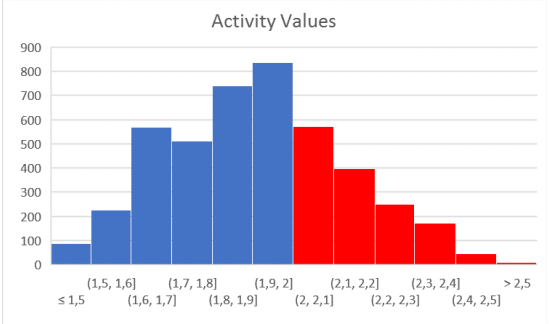


Figure 3.17: Activity values from database own.csv

Altogether, analyzing the emotion scores of the pictures in the databases has proven that it is not possible to directly separate pictures into emotions with clearly defined borders, in the way as it was described before by the code 3.9. A different solution needed to be found.

3.1.5 Summary

A few steps had to be completed - from getting the pictures over the tagging process - until the creation of the databases containing all information needed for the recommender system could be finished completely. Figure 3.18 shows a graphical illustration of the required steps that were taken; Python was utilized for the whole process.

Figure 3.18: Summary of the steps taken to generate the picture databases

3.2 Recommender System

Having collected all information about the pictures in the databases, the recommendations could now be calculated. This required building an underlying **recommender system** performing all necessary calculations. Before all steps that are done by the custom system, which was created, are explained together with the respective code, a brief overview of similar available systems is given.

3.2.1 Available Systems

Available mobile recommender systems similar to the use case needed here can be put into different categories: There are existing GitHub projects in which **apps** for Android devices have been created, such as FlickrMaps [39], which lets the user search for Flickr photos and display them via Google Maps at the locations where they were taken, or RoxAndroid [40], which is a recommender system for tourists that recommends routes of Points of Interests, which can be navigated through via Foursquare check-ins, based on the user's and his friend's likes.

Many **libraries** for recommender systems have been created as well: LibRec [41] is a Java library that implements many recommendation algorithms, and especially resolves the tasks of rating prediction and item ranking. RankSys [42] is another Java library, focusing on the ranking task problem and implementing particularly collaborative filtering recommendation algorithms.

The Python programming language offers a lot of advantages that are useful for building recommender systems [43], such as its resource-saving and efficient data processing capabilities. Therefore, many **Python-based** recommender systems exist: Surprise [44] is a tool based on a SciKit, an add-on of the `SciPy` package, to build and analyze recommender systems, including prediction algorithms, similarity measures, built-in datasets, and tools to evaluate, analyse and compare the performance of algorithms. The demonstration of a recommender system here [45] is based on the `Pandas` package.

[39] `https://github.com/PROGrammar/FlickrMaps-Android`
[40] `https://github.com/dan-zx/rox-android`
[41] `https://github.com/guoguibing/librec`
[42] `https://github.com/RankSys/RankSys`
[43] `https://www.datacamp.com/community/tutorials/recommender-systems-python`
[44] `https://github.com/NicolasHug/Surprise`
[45] `http://enhancedatascience.com/2017/04/22/building-recommender-scratch/`

Tensorflow [46] is an open-source machine learning framework using data flow graphs, being the base for some **Tensorflow-based** recommender systems written in Python: TensorRec [47] is a recommendation algorithm framework, allowing to develop own recommendation algorithms, customize them using Tensorflow, and consuming three pieces of data, namely user features, item features and interactions. TF-recomm [48] and Recommender [49] use a factorization model in the context of collaborative filtering and the computing capabilities of Tensorflow.

3.2.2 Custom System used here

The recommender system used here has been built from scratch in Python, as all parts of the picture collection and analyzing process before have been as well. To begin, installing the packages `NumPy`, `SciPy` and `skicit-learn` gives access to a huge base of algorithms specifically designed for recommender systems.

An overview of techniques for recommender systems can be found at [50]. Another comparison between different recommendation techniques, namely content-based filtering, item-to-item and user-to-item collaborative filtering, can be found at [51]. The type of recommender system built here features **collaborative filtering**, as it considers the user behavior, i.e. data, as well as the pictures, i.e. item data, for the recommendations. Furthermore, it is **memory-based**, because the system memorizes and stores the data for the generation of recommendations. Some sort of similarity measure - a distance algorithm, e.g. Euclidean distance, cosine similarity or Pearson correlation [52], or classifier such as the k-nearest neighbor algorithm - is needed for the system.

In the following, it is shown step by step how all the information necessary for the calculation of the recommendations is read, how they are calculated, and how the relevant resulting data is selected for further processing.

[46] `https://www.tensorflow.org/`
[47] `https://github.com/jfkirk/tensorrec`
[48] `https://github.com/songgc/TF-recomm`
[49] `https://github.com/felipessalvatore/Recommender`
[50] `http://dataaspirant.com/2015/01/24/recommendation-engine-part-1/`
[51] `https://blogs.gartner.com/martin-kihn/how-to-build-a-recommender-system-in-python/`
[52] `http://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/`

3.2.2.1 Saving the Parameters

The Python code 3.12 saves the provided parameters used for the recommendation calculations. `gpsPos` holds the location (**latitude** and **longitude**), `paVal` contains the emotion values (**pleasure** and **activity**) for which recommendations are to be calculated. Formally, a dataset containing various information is used to apply appropriate algorithms on; each item of the dataset consists of one record with certain features. One dataset's record here equals a **picture**, its features are **latitude, longitude, pleasure and activity**. Summarized can be stated that both as user data / categories and as item data / categories serve these same four parameters.

The picture database for which recommendations are performed is specified (in `db`), as well as whether the recommendations should consider the emotions, i.e. the pleasure and activity values, or not (`rand`), thus in the latter case recommending randomly - this is just for demonstration purposes and will be explained below in more detail. Two `NumPy` arrays for the GPS and the pleasure and activity data, `gpsA` and `paA`, are created, each taking the parameters `fileNumber` (total number of pictures in the database) and `closeNumber` (number of pictures that are considered for the final calculation step) for their dimensions into account. The filling of these arrays will be demonstrated in the next steps.

```
import sys
import numpy as np
gpsPos = [[sys.argv[1],sys.argv[2]]]
paVal = [[sys.argv[3],sys.argv[4]]]
db = sys.argv[5]
rand = sys.argv[6]
fileNumber = sum(1 for line in open(db))
closeNumber = 50
gpsA = np.zeros(shape=(fileNumber,2))
paA = np.zeros(shape=(closeNumber,2))
```

Listing 3.12: save all parameters, create arrays

3.2.2.2 Finding the Closest Places

The Python code 3.13 fills the array `gpsA` with all the GPS position data of the database. It then performs a k-nearest neighbor search for this data by (1) fitting the nearest neighbor object to the dataset and thus checking the GPS data, (2) finding the k-neighbors of each point in the dataset by calling `kneighbors()` (this step is optional) and (3) finding the k-neighbors of the selected position `gpsPos`, returning only the indices, and selecting the

50 closest elements, i.e. pictures. `closeResults` now holds the respective picture indices, sorted after closeness to the selected position. The core idea is based on the article [53].

```
import csv
from sklearn.neighbors import NearestNeighbors
with open(db, 'r', newline='') as csvIn:
  database = csv.reader(csvIn)
  for counterGPS, row in enumerate(database):
    gpsA[counterGPS] = row[3], row[4]
gpsNeighbors = NearestNeighbors(n_neighbors=closeNumber, algorithm='
   ball_tree').fit(gpsA) # (1)
gpsDistances, gpsIndices = gpsNeighbors.kneighbors(gpsA) # (2)
closeResults = gpsNeighbors.kneighbors(gpsPos, return_distance=False) #
   (3)
```

Listing 3.13: fill gps data array, calculate 50 closest recommendations

To demonstrate the behavior, an example for calculating recommendations is shown step by step for the following parameters (the location is the Marienplatz in Munich): **latitude: 48.137079, longitude: 11.576006, pleasure: 2, activity: 2, database: local.csv, random: false**.

Output of `print(gpsA)`:

```
[[ 48.174342  11.553561]
 [ 48.143267  11.582808]
 [ 48.161877  11.596605]
 ...,
 [ 48.106191  11.553313]
 [ 48.177255  11.555119]
 [ 48.131324  11.548004]]
```

Output of `print(closeResults)`:

```
[[227 516 345 106 461 447 122 365 376 128 325 281 422 543 123 525 615 487
  589 629 584 479 512 102 308 542 153 573 617 130 238 163 561 465 448 628
  321 580 411 159 627 616  33 466 300 261 566 103  84 368]]
```

[53] http://www.data-mania.com/blog/recommendation-system-python/

3.2.2.3 Calculating the Recommendations

The Python code 3.14 fills the array `paA` with all the pleasure and activity data only of the 50 pictures which are closest to the selected location, which have been stored in `closeResults` before (0). It also stores all information about these pictures in the array `info`, sorted in order of appearance in the database, separated with comma. It then performs a k-nearest neighbor search for this data by (1) fitting the nearest neighbor object to the dataset and thus checking the pleasure and activity data, (2) finding the k-neighbors of each point in the dataset by calling `kneighbors()` (this step is optional) and (3) finding the k-neighbors of the selected pleasure and activity values `paVal`, i.e. emotion, returning only the indices, and selecting the three best elements, i.e. pictures. `topThree` now holds the respective picture indices, sorted after closeness to the selected emotion.

```
info = []
with open(db, 'r', newline='') as csvIn:
  database = csv.reader(csvIn)
  gpsRows = [row for elem, row in enumerate(database) if elem in (
      closeResults[0])] # (0)
  for counterPA, row in enumerate(gpsRows):
    paA[counterPA] = row[6], row[7]
    info.append(','.join(row))
paNeighbors = NearestNeighbors(n_neighbors=3, algorithm='ball_tree').fit
   (paA) # (1)
paDistances, paIndices = paNeighbors.kneighbors(paA) # (2)
topThree = paNeighbors.kneighbors(paVal, return_distance=False) # (3)
```

Listing 3.14: fill pleasure and activity data array, calculate 3 best recommendations

The above mentioned example is continued. Output of `print(paA)`:

```
[[ 1.9714  1.8634]
 [ 2.1853  1.9823]
 [ 1.9762  1.751 ]
 ...,
 [ 2.135   1.9134]
 [ 2.146   2.0004]
 [ 1.9294  1.9395]]
```

Output of `print(topThree)`:

```
[[ 3 46  8]]
```

3.2.2.4 Selecting the Data for Further Processing

The Python code 3.15 converts the recommendation data for a proper representation and further processing; `closeIndices` in the end holds the indices of the 50 closest pictures; with their help, all the information about these pictures, i.e. source, date and time taken, etc., can be restored from the original database file. If *real* recommendations are selected, `topInfo` holds the information about the three best pictures, i.e. fitting most to the selected emotion, or, if *random* recommendations are selected, about three distinct random pictures of the 50 closest.

```
from random import sample
closeIndices = str(closeResults.tolist()).replace(',','')
randints = sample(range(50),3)
topInfo = str(info[topThree[0][0]])+'\n'+str(info[topThree[0][1]])+'\n'+
    str(info[topThree[0][2]]) if rand == 'false' else str(info[randints
    [0]]+'\n'+info[randints[1]]+'\n'+info[randints[2]])
print(closeIndices[2:len(closeIndices)-2]+'...'+topInfo)
```

Listing 3.15: convert recommendation data for proper representation

Final output of `print(topInfo)` for the example:

```
39608589202_cd546da37e_b.jpg,2018:01:09,16:02:28,48.137582,11.575459,
architecture, no person, church, travel, cathedral, city, tower,
building, goth like, outdoors,1.9909,1.9657,miserable
39205177094_54e7e114dc_b.jpg,2018:01:26,17:53:19,48.137275,11.575436,
architecture, no person, church, building, travel, religion, cathedral,
city, Gothic, outdoors,2.0325,2.0514,happy
25044228537_616a5cd8d8_b.jpg,2018:01:26,22:04:01,48.136505,11.575755,
city, architecture, roof, no person, travel, church, cityscape, town,
building, skyline,2.066,1.9726,contented
```

The resulting file **RecSys.py** can be used for calculating recommendations at the selected location for the chosen emotion with the arguments `latitude`, `longitude`, `pleasure`, `activity`, `picture database`, and `rand` (recommendations either real or random).

3.2.3 Summary

As a summary, it can be stated that all calculations are performed through Python, directly and locally on the computer. The focus of the recommendations is on the influence of the factors **location**, defined by latitude and longitude coordinates, and **emotion**, defined by pleasure and activity values. The two calculations steps each first consider all pictures, take the location into account, select the 50 closest, then take the emotion into account, considering only these 50 pictures, and eventually result in the three best pictures. The calculations are performed with the k-nearest neighbor search, using the *ball tree* algorithm [54]. This technique belongs to the field of machine learning, in particular to the category of unsupervised learning. The concrete algorithm configuration was selected due to its simplicity and fast calculation [55]; besides that, many other recommender systems also use the k-nearest neighbor algorithm. Other algorithms and metrics, e.g. clustering techniques such as DBSCAN or k-means [56], were not useful for this use case, since it was not possible to put the data into (reasonable) clusters, as tests with the DBSCAN algorithm showed.

3.3 Presentation: User Interface

An appropriate way of displaying the calculated data needed to be found, to serve as the main point for the user to access the recommendations. One open question was which **architecture** should be selected. It was decided to choose an architecture with a **client**, where the user can input and manage all his specified data, and a **server**, where the calculation of recommendations takes place. For the client, a **graphical user interface** (GUI) was needed which shows an interactive map and which can process the picture databases to show the pictures on the map. In addition, the GUI had to feature options where the user can select his current emotion, i.e. the mood he wants to receive recommendations for, and where the recommendation results are presented. All used techniques and possible alternatives are explained and added step by step in the following.

There are some **existing solutions** for such applications, similar to the area of geographic information systems and using Python. Geotiler [57], for instance, is a Python library that allows to create maps using tiles from a map provider, e.g. OpenStreetMap, which then

[54] http://scikit-learn.org/stable/modules/neighbors.html

[55] http://jakevdp.github.io/blog/2013/04/29/benchmarking-nearest-neighbor-searches-in-python/

[56] http://scikit-learn.org/stable/modules/clustering.html

[57] https://wrobell.dcmod.org/geotiler/index.html

can be used by interactive applications or to create data analysis graphs. It is based on the Python port of Modest Maps [58], a display and interaction library for tile-based maps written in JavaScript. osmapi [59] and Smopy [60] are Python wrappers for the OpenStreetMap API, the latter of which suggests using the Folium project, which is briefly introduced below.

3.3.1 Folium

As a first approach for graphically showing the calculated information, Folium [61] has been used. The project tries to combine the data manipulation advantages of Python and the visualization of Leaflet [62], a JavaScript-based library to build interactive maps, making it possible to visualize data that has been manipulated in Python. It offers a variety of functions [63], documented modules [64] which add features, and plugins [65]. A tutorial can be found at [66].

The Python code 3.16 shows a minimum working example, creating a blue marker for every picture in the database `local.csv`, putting a red marker to the Marienplatz in Munich, and saving everything as a HTML file which can be opened in a browser.

```
import folium
import csv
m = folium.Map(location=[48.137079, 11.576006])
with open('local.csv', 'r', newline='') as csvIn:
  database = csv.reader(csvIn)
  for row in database:
    folium.Marker(location=[float(row[3]), float(row[4])]).add_to(m)
folium.Marker(location=[48.137079, 11.576006], icon=folium.Icon(color='
   red', icon='cloud')).add_to(m)
m.save('index-py.html')
```

Listing 3.16: Folium example

[58] https://github.com/stamen/modestmaps-js
[59] https://github.com/metaodi/osmapi
[60] https://github.com/rossant/smopy
[61] https://github.com/python-visualization/folium
[62] http://leafletjs.com/
[63] http://python-visualization.github.io/folium/docs-v0.5.0/quickstart.html
[64] http://python-visualization.github.io/folium/docs-v0.5.0/modules.html
[65] http://python-visualization.github.io/folium/docs-v0.5.0/plugins.html
[66] https://georgetsilva.github.io/posts/mapping-points-with-folium/

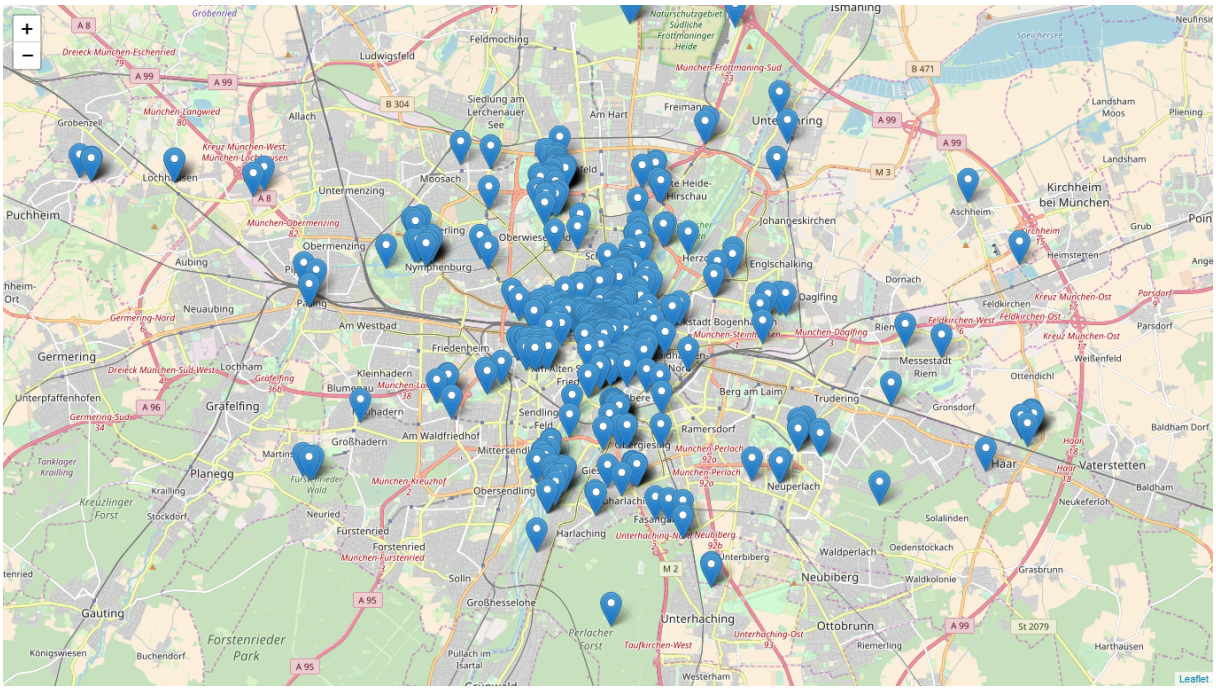


Figure 3.19: Folium markers on map (Munich area)

Figure 3.19 shows how the generated HTML file with the markers looks like.

However, the solution of Folium holds some limitations, as it does not give access to the full functionality of the underlying Leaflet framework: Since there was no direct access to the HTML code of the pop-ups, marker pop-ups could not be linked with the pictures that belong to each of the markers, so that these pictures would be visible when the pop-ups are clicked. Additionally, it was not possible to customize the output HTML in any way.

3.3.2 Leaflet

LeafletJS [67], as already mentioned before, is a library based on JavaScript that can be used to display interactive maps [68]. The reference can be found at [69]. Leaflet, in addition, allows to use map tiles from many different sources [70] and can be extended as desired.

The HTML code 3.17 shows a minimum working example, using the cascading style sheet (CSS) and JavaScript (JS) files from Leaflet, creating a Leaflet `map` by adding it as a `<div>` element, and in the `<script>` part centering the map to the coordinates of the

[67] `https://maptimeboston.github.io/leaflet-intro/`
[68] `http://leafletjs.com/examples/quick-start/`
[69] `http://leafletjs.com/reference-1.3.0.html`
[70] `http://leaflet-extras.github.io/leaflet-providers/preview/index.html`

specified starting `position` (here again the Marienplatz), adding `osmLayer`, a tile layer drawing a map from OpenStreetMap, and a `marker` that can be moved.

```
<!DOCTYPE html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <title>Leaflet</title>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.3.1/dist/
        leaflet.css" />
    <script src="https://unpkg.com/leaflet@1.3.1/dist/leaflet.js"></
        script>
    <style>html, body {width: 100%; height: 100%; margin: 0; padding: 0;}
        </style>
    <style>#map {position: relative; width: 100.0%; height: 100.0%; left:
         0.0%; top: 0.0%;}</style>
</head>
<body>
    <div id="map"></div>
</body>
<script>
var position = [48.137079, 11.576006];
var map = L.map("map", {center: position, zoom: 4});
var osmLayer = L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{
    y}.png", {attribution: null, maxZoom: 18}).addTo(map);
var marker = L.marker(position, {draggable: true}).addTo(map);
</script>
```

Listing 3.17: Leaflet example

The resulting HTML file, which is stored as `index.html`, looks like the Folium example, without the additional markers for the pictures of the database. This HTML file will now be continuously expanded. It uses a few different CSS and JS files.

3.3.3 Leaflet Sidebar

One aim was to display all relevant information and content within one single HTML page, so that the user never has to switch between different HTML sites. To do so, a few possible approaches were considered, such as a *CSS sidebar* [71] allowing to switch between content via provided links, or possible *fullscreen overlays* [72]. In addition, various plugins have been built to make use of LeafletJS and extend its functionality. The Leaflet

[71] https://www.w3schools.com/w3css/w3css_sidebar.asp
[72] https://www.w3schools.com/howto/howto_js_fullscreen_overlay.asp

sidebar plugin "sidebar-v2" [73] has been eventually used due to its seamless integration into LeafletJS and good customization features. The sidebar is not used via a remote CDN, but locally instead, because four values had to be changed in the CSS file (version 0.4.0: lines 23, 114, 176, 196) to adjust the sidebar to a 50 pixels higher width, resulting in a total width of 500 pixels to have more space for content display.

Therefore the Leaflet sidebar, consisting of CSS and JS, is integrated with altogether three tabs in the `<body>` part. The `sidebar` is drawn and added to the map in the JavaScript code. Its icons are drawn by the FontAwesome CSS. Each of the tabs has its own content pane: The first tab **i** gives a brief explanation about how to use the system. The second tab ▶ will represent the recommender system. The third tab ➤ will show the recommendations, i.e. pictures, with all its affiliated information. Figure 3.20 shows the sidebar and the content of the first tab. The detailed contents and functionalities of the tabs are explained in the following.

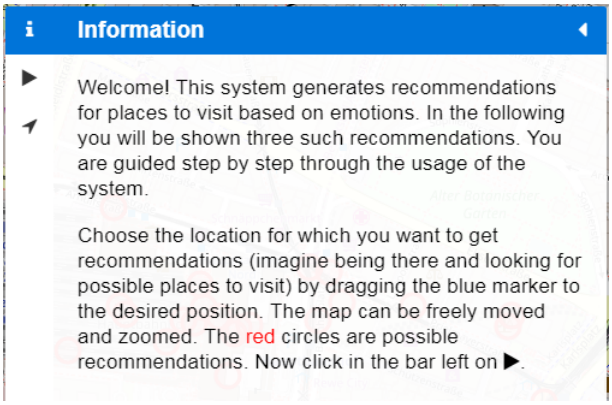


Figure 3.20: Leaflet sidebar

3.3.4 Flask

A critical point was to get the Python-based backend and the HTML- and JavaScript-based frontend to communicate with and call methods from each other. There are several approaches to solve this problem: The Python module for Common Gateway Interface (CGI) support [74] can be used to run Python scripts in a web environment. Another project uses the Tornado server to create a bridge between a Python server and a browser with JavaScript on the client side [75]. For Node.JS, which is a run-time environment for executing JavaScript code server-side, exists an available plugin called "Python Shell" [76].

A local Python server can be started using purely built-in methods with `python -m http.server` [77]. However, browsers are, without modification, not able to access local files - which is needed for the picture databases though - directly from an Ajax request

[73] `https://github.com/Turbo87/sidebar-v2`
[74] `https://docs.python.org/3/library/cgi.html`
[75] `https://github.com/patrickfuller/python-js-bridge`
[76] `https://github.com/extrabacon/python-shell`
[77] `https://docs.python.org/3/library/http.server.html`

(which can handle the communication between JavaScript and Python). This behavior is due to security reasons, the so-called same-origin security policy. To fix this for instance in Google Chrome, the browser either has to be invoked with the command line flag `--disable-web-security`, or an extension from the Chrome Web Store [78] needs to be installed, to avoid the error "Cross origin requests are only supported for (certain) protocol schemes" preventing local file access. Alternatively, sending the header `'Access-Control-Allow-Origin'` to the Python server via a script fixes this, too, and eventually allows Cross-Origin Resource Sharing for all protocol schemes.

Eventually, to build all methods together in one place, the server framework Flask [79], which is based on Python [80], and the corresponding Python package `flask_cors` were used. Its advantage is that it bundles all required functionality in one solution.

The Python code 3.18 starts the Flask server and allows Cross-Origin Resource Sharing. If a function is called via Ajax in JavaScript, Flask routes the request to the specified Python method, as e.g. under `getFile`, where the corresponding methods generates an URL internally used by Flask. All local files need to be located in a folder called `static`. To start the server locally on `localhost` on port `5000`, the code (saved as **app.py**) simply needs to be run.

```
from flask import Flask, request, redirect, url_for
from flask_cors import CORS
app = Flask(__name__)
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0 # cached files expire
    immediately (only when modified through server)
CORS(app)

@app.route('/getFile/')
def func():
  return redirect(url_for('static', filename=request.args.get('file')))

if __name__ == '__main__':
    app.run()
```

Listing 3.18: Flask (excerpt)

The JavaScript frontend can now call methods of the backend via Ajax. The backend routes these calls to execute the respective methods.

[78] https://chrome.google.com/webstore/detail/allow-control-allow-origi/nlfbmbojpeacfghkpbjhddihlkkiljbi

[79] http://flask.pocoo.org/

[80] https://www.makeuseof.com/tag/python-javascript-communicate-json

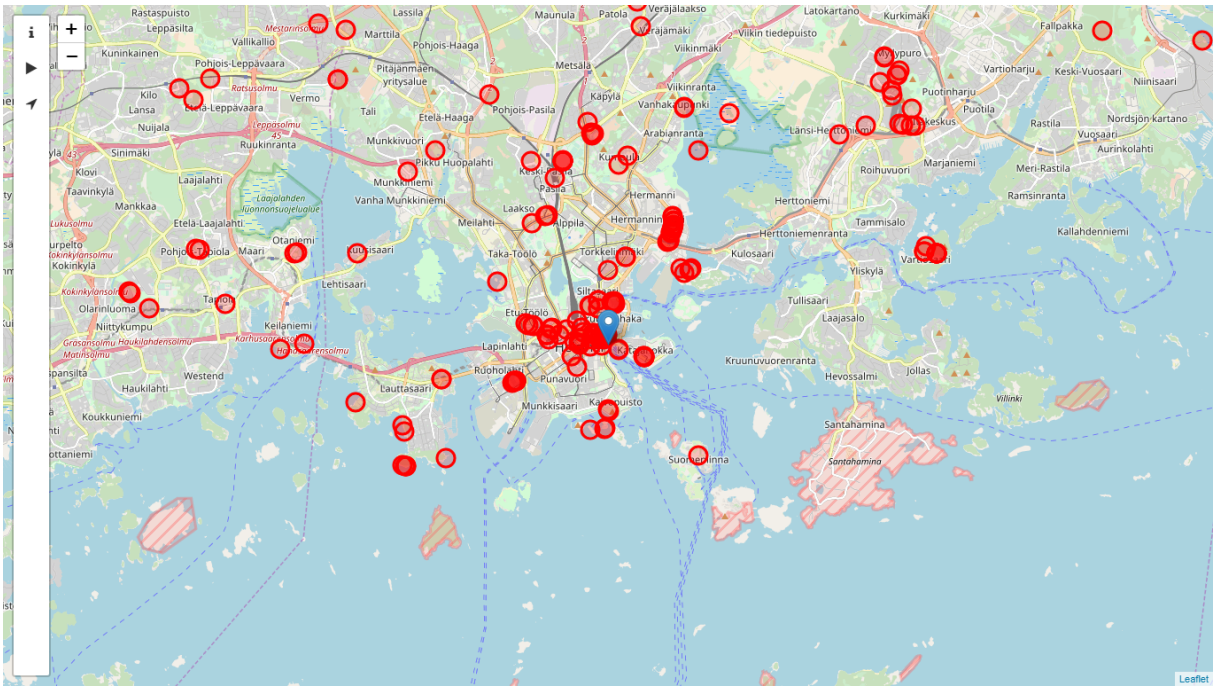


Figure 3.21: Markers of database flickr.csv (Helsinki area)

3.3.5 Displaying the Images

The frontend JavaScript function `drawMarkers()` shows round red circles of the Leaflet type `circleMarker` (the type `marker` used in the Folium example was replaced, due to performance problems when a high number of these complex shaped markers has to be drawn on the map) for each picture from the database; if the marker is clicked, the picture taken at there (at the marker position) is shown in a pop-up. The picture can be stored either locally in a folder called `img`, as in the databases `local.csv` and `own.csv`, or under a web link, as in `flickr.csv`. The `jQuery` JavaScript library gets the database file contents via Ajax and parses them in the variable `lines`. Each line of the file, i.e. picture, is represented by a `circleMarker` object (this step could have also been done with [81]). `drawMarkers("local.csv")` is executed as default.

The Figures 3.21, 3.22 and 3.23 show for each database how the Leaflet map looks like when the picture markers are drawn.

[81] `https://github.com/evanplaice/jquery-csv`

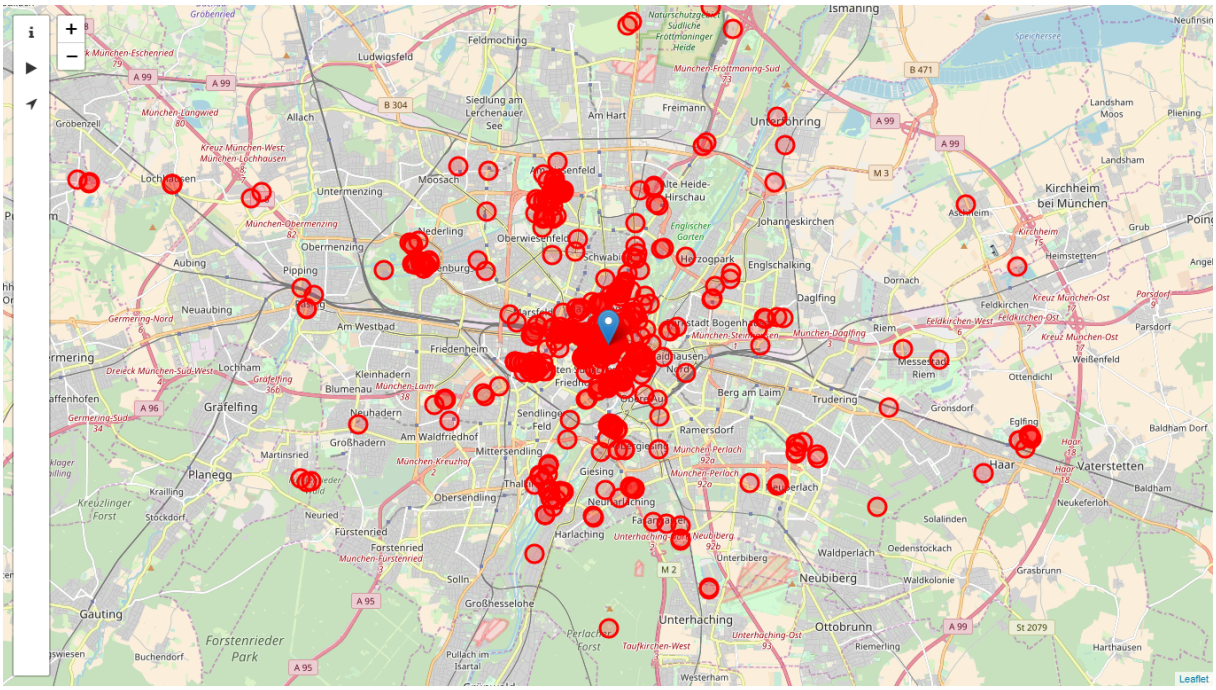


Figure 3.22: Markers of database local.csv (Munich area)

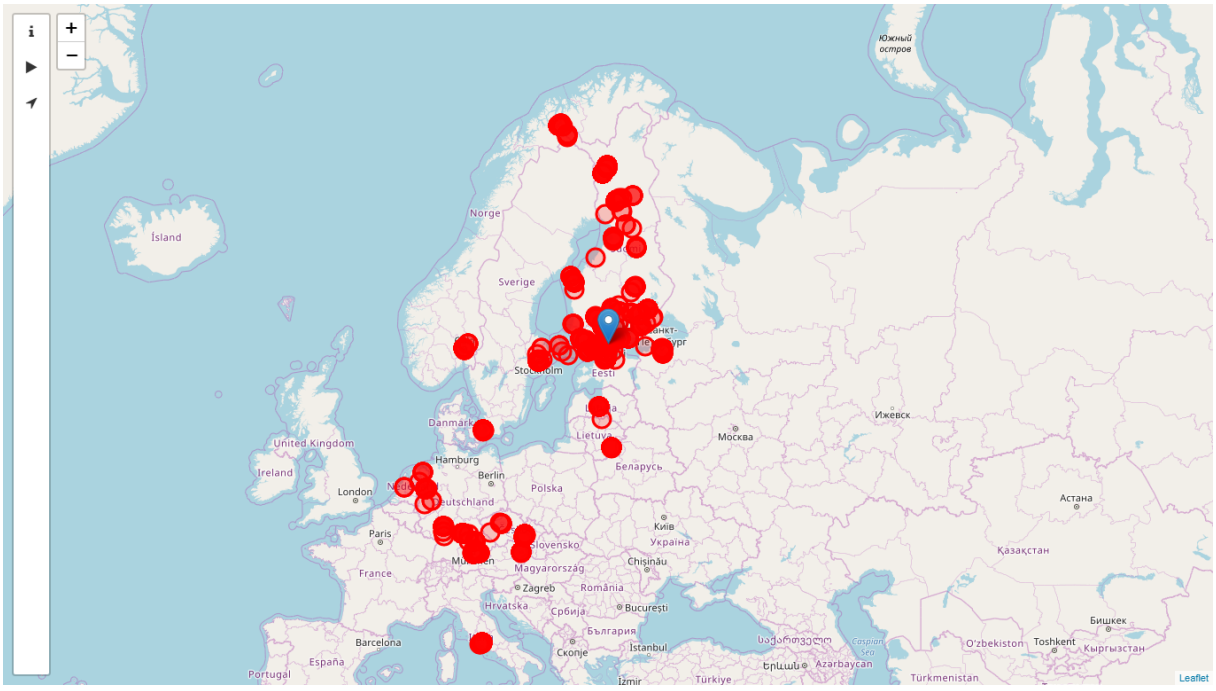


Figure 3.23: Markers of database own.csv

3.3.6 Choosing the User's Emotion

The second tab ▶ contains all input and information needed for the recommender system. It displays the current position of the marker as GPS coordinates at the top; when the marker is dragged, the position is updated, rounded and shown. This is realized in the frontend function `marker.on("dragend", function())`.

A very important part of the whole system was how to find out or select what the current emotion or mood of the user is, i.e. to build a link between the user's and the pictures' emotions. It was decided to use **explicit** collection of emotion data. Consequently, the system needs the user's answer to the question "What is your current mood?", demanding him to explicitly clarify his emotion at the moment. Therefore, the focus is on the evaluation of emotions, and not on how to get them implicitly or inferred. The recommender system acts for the user as a blackbox, meaning that he does not need to know how the calculations are actually done, but only needs to specify his emotion.

It was now essential to find some way for the user to input the emotion. A requirement was that a similar model of the one with **pleasure** and **activity** values from the emotion tagging of pictures had to be applied again. As a first try, two *draggable sliders* were used, one for pleasure and one for activity, each ranging from 1 to 3. Their values were newly drawn when they were dragged, and updated when they were clicked. However, this procedure was not really intuitive, since adjusting values on a scale from 1 to 3 can not automatically be connected with an emotional state, and their two-dimensional connection was not emphasized in this way. Other tries with two-dimensional plots were not successful either.

As the database analysis had shown before, the current evaluation of the pictures' emotions based on the 16 selected emotions from [Hasa 14] was not reasonable and consequently ignored in the further process. However, the idea of using an illustration of the emotions in a two-dimensional graphic was still a helpful approach. Therefore, the next try was to use a *clickable graphic*.

Instead of using an overloaded graphic with 28 emotions, only nine remaining emojis were selected: **happy, excited, angry, frustrated, sad, bored, relaxed, satisfied and neutral**. This is shown in Figure 3.24. The user's decision is furthermore simplified and supported by displaying an emoji next to the verbal name of each emotion. The emojis used in the graphic were taken from Emojipedia [82]. The resulting graphic, as shown in Figure 3.25, lets the user intuitively choose his emotion by clicking anywhere; this

[82] `https://emojipedia.org/apple/`

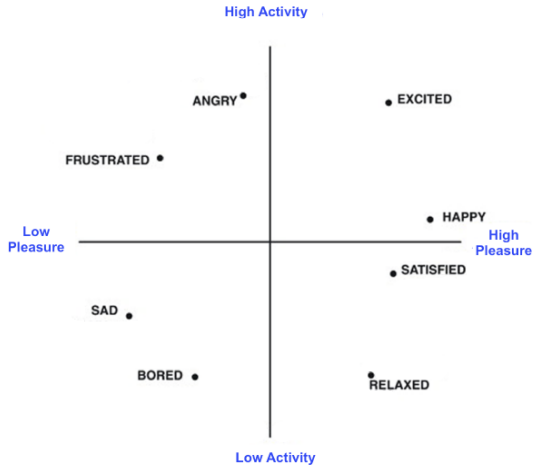


Figure 3.24: Emotions reduced to nine most important ones (from [Hasa 14])

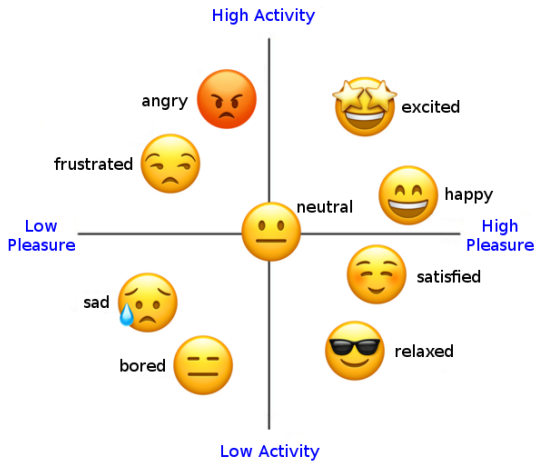


Figure 3.25: Emotion graphic with added emojis

is possible because the graphic is drawn onto a HTML5 canvas [83]. A canvas has many advantages: The exact mouse coordinates can be saved, the canvas can be manipulated by clicking, and therefore the own emotion can be adjusted exactly, meaning that the user does not have to select one specific emoji.

It was necessary to decide which positions should be used for the emotions in the graphic. To verify and proof the selection and the arrangement of the emotions taken before, a systematic step-by-step approach was performed: The machine learning technique of k-means clustering [84] was applied on the databases `local.csv` and `own.csv` for `k=9` and visualized in two-dimensional plots, again using the Python packages `SciPy` and `Matplotlib`. This is shown in Figure 3.26 and Figure 3.27; the respective cluster centroids are marked as red dots. The plots show that the selection and arrangement are reasonable.

The second Leaflet tab also shows statistics about the mouse coordinates, i.e. its position within the canvas, and `pV` and `aV`. These are the values for pleasure and activity, as they would be saved if the mouse would be clicked, and are therefore always updated when the mouse is *moved* within the canvas. They are in reality saved as values between 1 and 3, corresponding to the values from the emotion dictionary, but for displaying mathematically converted to values in % to be more intuitive for the user. This functionality is realized in function `showStats()`.

[83] `https://www.w3schools.com/html/html5_canvas.asp`
[84] `https://docs.scipy.org/doc/scipy/reference/cluster.vq.html`

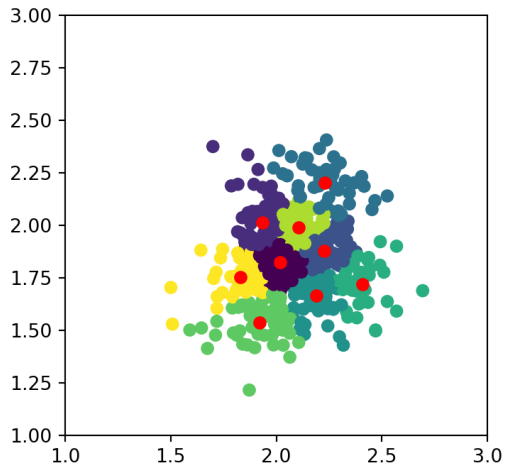


Figure 3.26: K-means algorithm for database local.csv with marked centroids

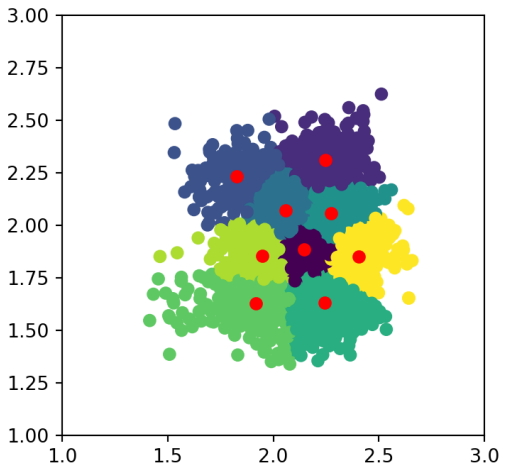


Figure 3.27: K-means algorithm for database own.csv with marked centroids

`pleasure` and `activity`, with the default values 2 and 2 (50% and 50%), are the values which are actually saved for the emotion selected by the user, and are therefore always updated when the mouse is *clicked* in the canvas, as well as converted to %. In addition, a black dot is drawn on the canvas, on the spot where the user clicked with his mouse. This functionality is realized in function `savePAandDraw()`. Figure 3.28 shows the representation of all described elements.

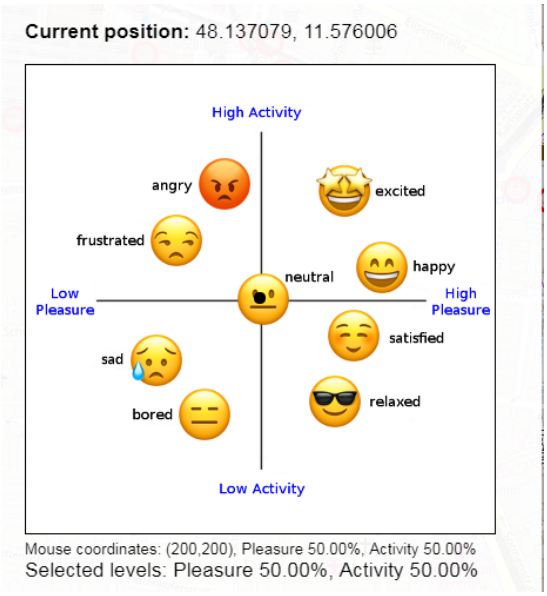


Figure 3.28: Choosing the user's emotion

3.3.7 Creating an Own Database

The system furthermore supports the ability to create an own picture database by clicking a link ("create your own by clicking **here**").

The frontend function `flickr()` 3.19 performs an Ajax call to the Python backend in **app.py** to create an own Flickr picture database at the marker location. The called method there, `xmlParser()`, offers basically the same functionality as the file **xmlParser.py**. There are only a few small edits: `import.sys` and `sys.argv[]` are not needed any more, but replaced with `request.args.get[]` instead, because the argu-

ments are delivered directly from the user via the provided URL. The calls to `sys.exit()` resp. `print()` are replaced by `return` statements. The file location of `dict.txt` is now `static/dict.txt`, the output database is saved under `static/flickr-own.csv`. When the process is complete, the browser sends an alert.

```
function flickr() {
  $.ajax({
    type: "GET",
    url: "http://localhost:5000/xml?lat="+position.lat.toString()+"&lng=
       "+position.lng.toString(),
    success: function(response) {alert(response);},
    error: function(response) {console.error(response);}
  });
}
```

Listing 3.19: function flickr() in HTML

3.3.8 Getting and Displaying the Recommendations

Now the database can be switched by clicking on the respective buttons (function `drawMarkers()`), and the calculation of recommendations happens by clicking on one of the two buttons.

The frontend function `recommend(rand)` performs an Ajax call to the Python backend in **app.py** to calculate the recommendations at the marker location for the chosen emotion (pleasure and activity values) and picture database, real or random. The called method there, `RecSys()`, offers basically the same functionality as the file **RecSys.py**. There are only a few small edits: As above, `import.sys` and `sys.argv[]` are not needed any more, but replaced with `request.args.get[]` instead. The call to `print()` is replaced by a `return` statement.

The method response is parsed, splitting the results that contain the information about the close and recommended pictures at the separating "...". The **first** part contains the indices of the 50 pictures that are closest to the position: They are drawn as yellow circle marker objects on the map. The indices are enough, because the database file is already parsed in the JavaScript variable `lines`. The **second** part contains the infos about the three pictures which are recommended. The information is stored in five arrays that are gradually filled, the pictures are drawn on the map as blue circle marker objects, connected with a line together with the draggable marker (which represents the user's position). The line can also be drawn using the Leaflet Routing Machine with its CSS

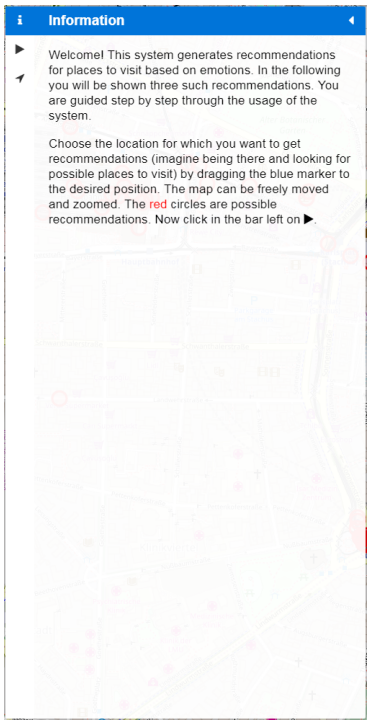


Figure 3.29: First tab

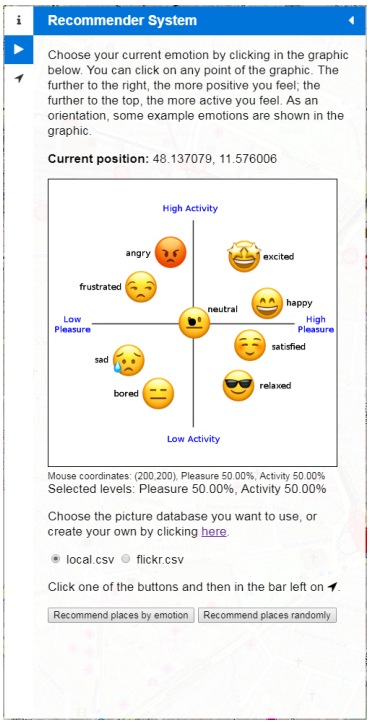


Figure 3.30: Second tab

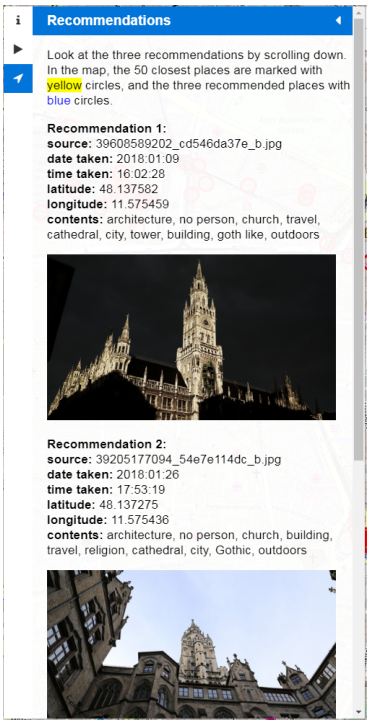


Figure 3.31: Third tab

and JS files, showing a detailed navigation between all places. The map eventually flies to the position of the marker and changes to a high zoom level. The recommendations, i.e. pictures, are shown together with all their information at the third tab ➤.

The system has now been finally built with all its components. The screenshots demonstrate its functionality.

Figures 3.29, 3.30 and 3.31 show all three sidebar screens, with the same values as demonstrated in the previous section about the recommender system. Figure 3.32 shows the three calculated recommendations, and how they are displayed on the map.

Listing 3.20 shows a brief summary of the file structure (used variables, methods, etc.) of **index.html**.

```
<!DOCTYPE html>
<head> //contains links to used CSS and JS, style definitions </head>
<body> //sidebar tabs and content, Leaflet map </body>
<script>
//variables for position, map, sidebar, layers, draggable marker
marker.on("dragend", function(e)) {}
//variables for file (picture database), prefix (picture location),
   lines (file content), picMarkerGroup (group of picture markers),
   recMarkerGroup (group of recommendation markers)
```

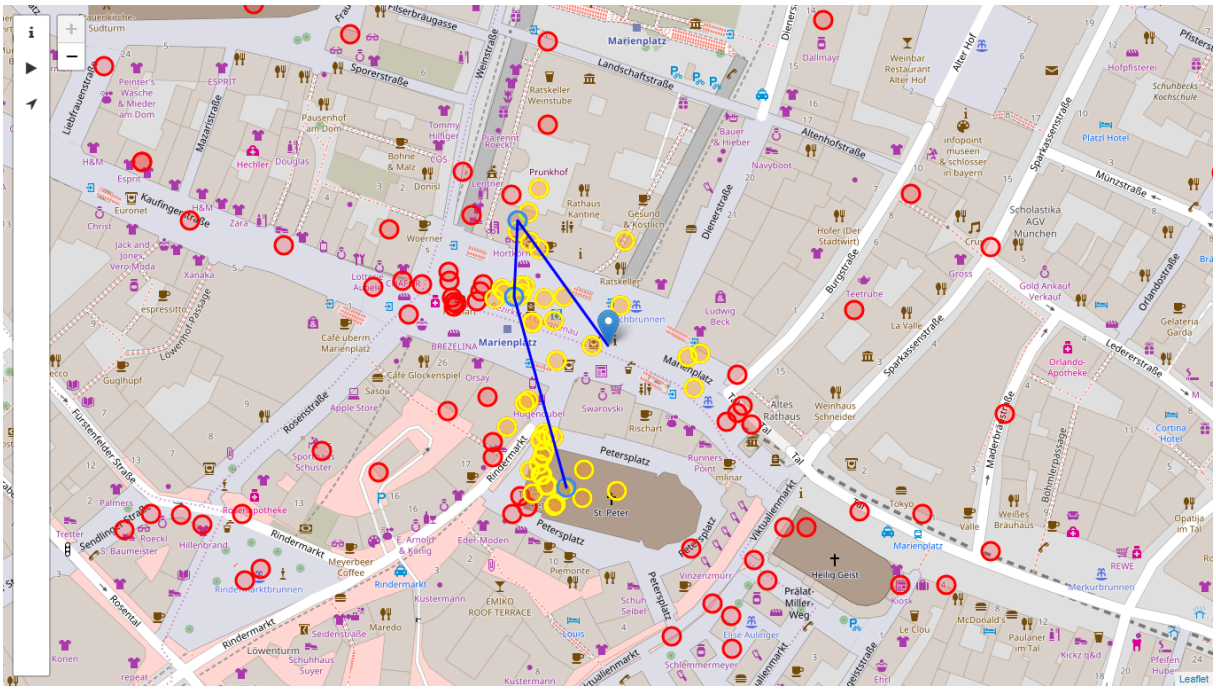


Figure 3.32: Recommendations on the map (Munich example)

```
function drawMarkers(selectedFile) {}
drawMarkers("local.csv");
//variables for shown and saved pleasure and activity values
function showStats(e) {}
function savePAandDraw(e) {}
function flickr() {}
function recommend(rand) {}
</script>
```

Listing 3.20: file structure of index.html

3.3.9 Summary

The system's ℹ tab shows general information about the recommender system and its usage. The map visualizes the position specified by the user with a marker, and all geo-tagged photos from the selected database (two modes are supported, `local.csv` and `flickr.csv`) in red. After the user has chosen his emotion on the ▶ tab, the closest 50 photos are marked in blue and the top three in yellow, i.e. showing the photos matching the selected emotion best. The ➤ tab shows three recommendations together with all its information, a route between them is drawn on the map. Since the recommendations are

performed via the Python server, the system runs entirely without an internet connection, except for the loading and caching of the map tiles from OpenStreetMap; otherwise, no network connection is needed. The used technologies for client and server are illustrated in Figure 3.33.

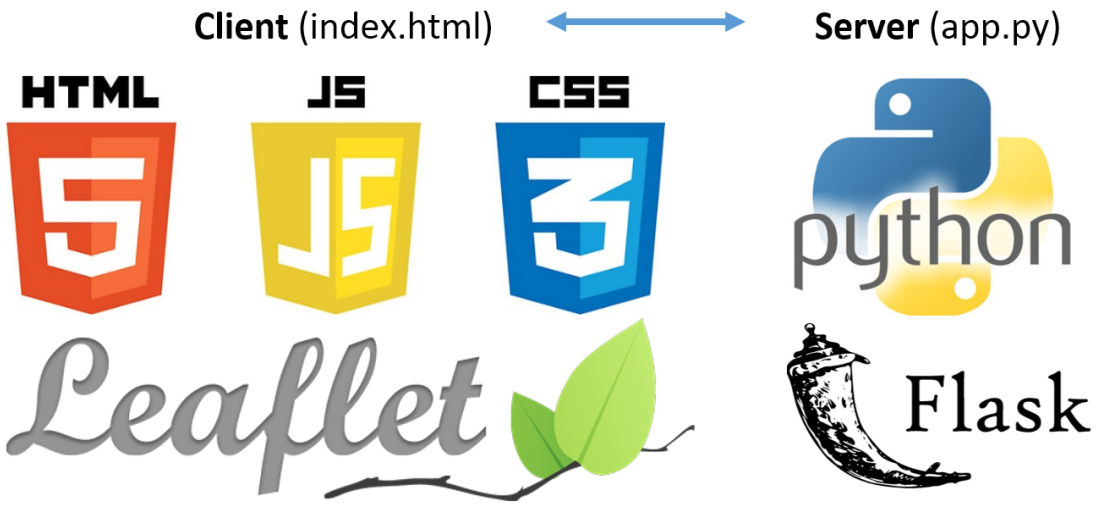


Figure 3.33: Summary of the technologies used for client and server

Figure 3.34 analyzes the communication between user and server for the two use cases of creating an own picture database is created, and of calculating the recommendations. It lists the information that is transferred when the user sends **requests** to the server via the **frontend** functions, and the **responses** that are sent to the client when the server has finished the execution of the **backend** methods.

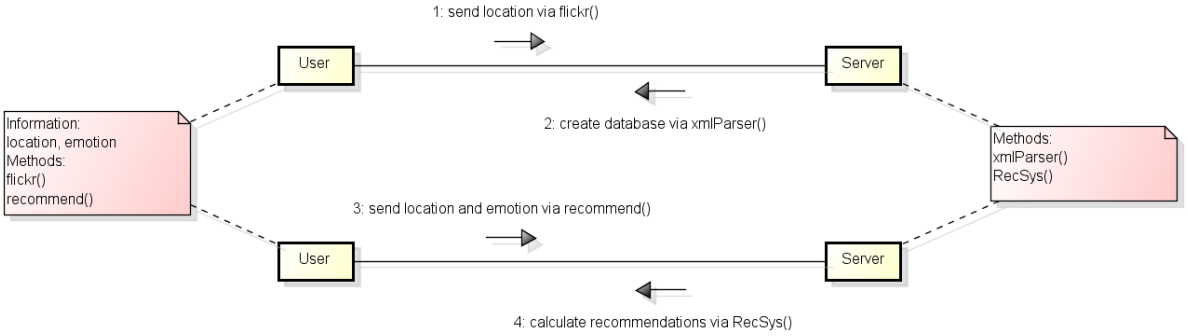


Figure 3.34: UML communication diagram showing requests and responses between user and server

Chapter 4

Evaluation

To scientifically ensure that the additional emotional component helps to improve the place recommendations, a user study was designed and carried out. The aim of this evaluation of the system was to show that, if the emotion is used in addition to the location as a parameter for the recommendation system, an improvement is made, compared to a system where only the location is considered. It was therefore aimed to demonstrate that it is useful from the user point of view to include the results of the images' emotion tagging process in the suggestions.

4.1 User Study Design

The two possible recommendation variants, recommending nearby places considering the emotion vs. recommending nearby places randomly, were used in the study. This is called **A/B testing**: This form of a statistical hypothesis test with two variants, A and B, is a controlled experiment often used in web analytics to compare two versions of a single variable by testing participants' responses to variable A against variable B, and determining which of the two variables is more effective, e.g. which variant leads to higher conversion rates. To compare the recommendations unbiased, it was essential that the participants were not told which of the variants they were seeing. Besides the recommendation variant, all other elements and parameters of the system, such as used picture database and number of shown recommendations, had to stay the same.

To perform this test, a new HTML file `index-study.html` was created and edited in a few ways in comparison to `index.html`. All distracting displayed objects, which were not needed for the execution of the study, were hidden with `style="display:none"`: the

current position, the statistics (mouse coordinates, values for pleasure and activity), the radio buttons to choose the picture database, and the buttons to create an own picture database or to select whether to calculate recommendations real or randomly. For the two variants, the `rand` argument of the `recommend()` method was used. A second ➤ tab was added to the sidebar; one of these tabs contained the real recommendations, one the random, without revealing this arrangement to the participants. Additionally, on the ➤ tabs, no information about the pictures, such as date and time taken, was shown, but only the pictures themselves. As a last tab, ★ was added, where the questions for the study were asked.

4.2 Conducting the Study

The study was carried out in the Munich area amongst a total of 28 participants. It was performed locally and not remotely over the web, because it was easier in this way to directly interact with the participants, handle possible questions about the system and immediately answer them.

Only the database `local.csv` containing pictures from Munich was used for the study, and all texts in the explanations and the emotions graphic were furthermore translated to German. The participants were guided step by step at using the system via the instructions on the screen, especially asked to take a close look at the pictures while also considering the map showing the markers.

The questions which were to answer on the last ★ tab were, besides gender and age (grouped in ranges of ten years, thus adding up to a total of five answer options), the participants' rating for both sets of recommendations (real and random), on a scale from one to five stars. The corresponding question was formulated like this: "How suitable - in terms of your chosen emotion - did you find the three recommendations? How do you rate quality and accuracy of the recommendations?" This was therefore testing the quality and usefulness of the system. The participants could also give additional comments and suggestions for improvements.

4.3 Results

The following diagrams show the results of the questions which were asked.

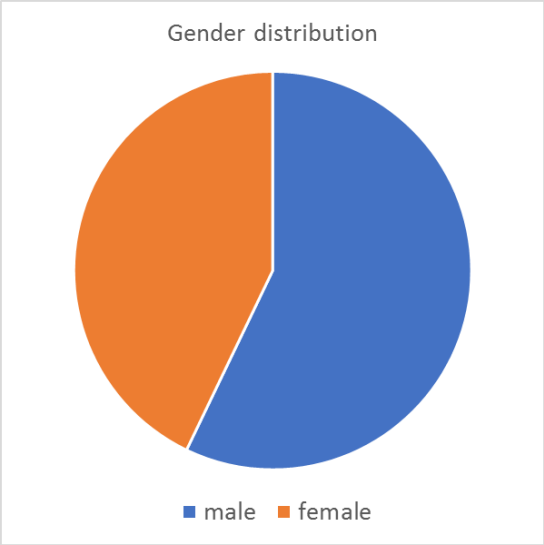


Figure 4.1: Gender distribution

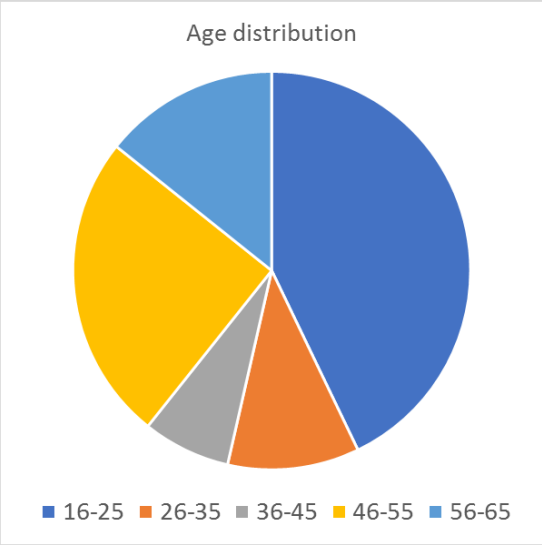


Figure 4.2: Age distribution

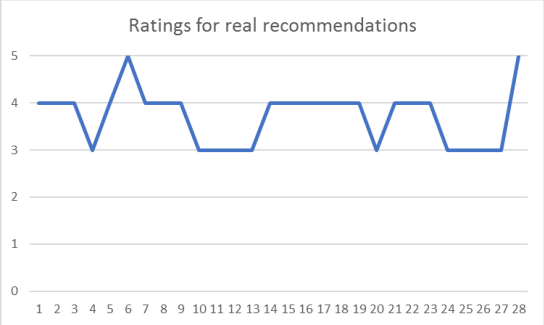


Figure 4.3: Ratings by the participants for real recommendations

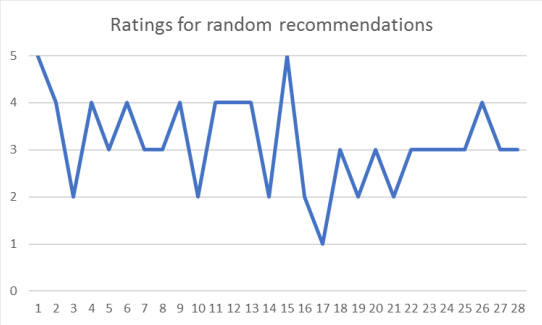


Figure 4.4: Ratings by the participants for random recommendations

Figure 4.1 shows the gender distribution of the participants, Figure 4.2 shows the age distribution. To get a broad spectrum of opinions from a wide demographic range, it was tried to consult participants with different genders, ages and also interests.

Figure 4.3 and Figure 4.4 show the ratings of the participants for the emotion-based and for the random recommendations, respectively. The average of the "real" ratings is **3.71**, the average of the random ratings is **3.14**, thus showing an increase of more than half a star for the recommendations that consider the emotions.

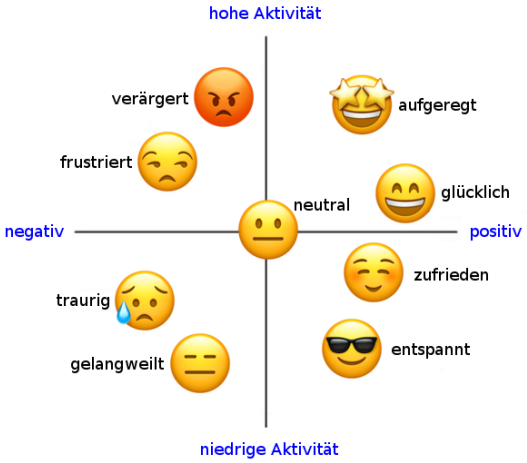


Figure 4.5: Emotion graphic translated to German

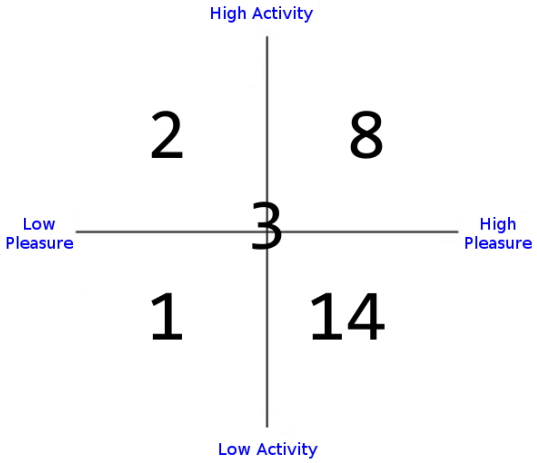


Figure 4.6: Emotions selected by the participants

Figure 4.5 shows the emotion graphic in German. Figure 4.6 reveals from which areas the participants selected emotions. The most popular choices were *satisfied* (10x) and *happy* (7x). 10 choices featured a rather active emotion (upper half), 15 choices featured a rather passive emotion (lower half). 22 choices were rather positive (right half), 3 choices were rather negative (left half).

4.4 Interpretation

An interesting aspect was that almost only positive emotions were chosen; consequently the question arised about how to deal with negative emotions: Does it, for instance, make sense to recommend sad pictures, i.e. places where sad pictures were taken, when the user is sad, or would it in general also sometimes be beneficial to get recommendations for an emotion different from the selected? Due to its construction, the recommender system tries to suggest pictures that express the same emotion as the user's. Therefore it was interesting to know which recommendations the participants *want* to receive, i.e. from which area in the two-dimensional graphic, in dependence of their *chosen* emotion in one of the areas in the graphic, and how *useful* the participants found the suggestions they received in general.

The participants commented a lot on this topic of a **negative-positive gap** between the chosen emotion and the desired recommendations: Many said that, if negative emotions

are selected, recommendations should rather turn away from the emotion, so suggest a positive location instead. Some wanted only positive recommendations, also when the selected emotion was negative, or at least a slightly more positive recommendation than the selected emotion. Another participant stated that if a negative emotion is selected, the recommendation should rather be a calm location, or a location which makes the user feel more positive; if he is sad, not a location with many people should be recommended. However, others suggested that there could be a choice, so that the user could select what kind of recommendations are given, especially when the user's emotion is negative, and that in the case of a positive selected emotion, also neutral suggestions would be all right. Some participants even said that the recommendations should always match the chosen emotion, and that there is definitely a given coincidence and overlap between the own emotion and expected recommendations.

Further opinions were that not only the negative-positive, but also the **passive-active gap** could be addressed: If a user is, for example, bored, the given recommendations could be more active. The recommended places themselves and the **user's familiarity** with them are also a factor for rating the usefulness of the results: If the system is used at a known location, it would be preferred to receive recommendations for new or not only touristic, but rather special places; the distance of the places between each other might also matter in terms of physical activity, since the suggested route could sometimes be a bit far. This perception might be different in an environment people are not familiar with: Then they might be willing to move and explore more and also want to see the highlights, i.e. more touristic places.

As a conclusion, there are some **possible uncertainties** and reasons why the system might not always work as desired. Some participants were saying that the whole recommendation process is a very subjective matter, not only due to own, personal interpretations of emotions. Since the pictures are on average rather positive, the suggestions for negative emotions might not be equally accurate, especially when considering the fact that people might tend to dislike negative recommendations. Furthermore, the emotion of a picture does not necessarily tell something about its content, i.e. its usefulness as a recommendation at a particular location. It can also happen that different pictures of the exact same place contain different emotions, possibly due to different content, objects or surroundings, such as the season, people in the picture, or present weather, which can lead to a slightly different emotion score.

Chapter 5

Discussion & Future Work

As already previously mentioned, some aspects of the system are discussable and could lead to unwanted or inaccurate results. In addition, many parts of the system leave room for enhancement and further improvement, e.g. by selecting different alternatives, in particular in the areas of usable data sources, image content and emotion analysis, the recommender system itself, and the user interface.

5.1 Data Sources

Besides images from Flickr, **other websites** and photo communities such as 500px [1], Scoutt [2] or also Twitter [3] for more recent photos could be used as sources of images as well, provided that they allow access to their respective APIs. Alongside photos, **other types** of input data could be used additionally, for example videos - the Clarifai API, for instance, is able to analyze the contents of videos as well - or text: Tweets on Twitter can have a geotag, and there has been related work on analyzing the contents of text towards emotions expressed in it, using hashtags and emojis to find an emotion classification or rate a score.

Nevertheless, image emotion recognition is a difficult task and especially highly dependent on the starting data, i.e. the pictures. **Filtering inappropriate pictures**, such as batches of pictures taken at the exact same location (position duplicates), as done for the picture library database `local.csv` from Munich, could be automated and would increase the

[1] `https://500px.com/`
[2] `http://www.scoutt.com/`
[3] `https://twitter.com`

quality of the recommendations. A possible solution are the Flickr parameters `woeid` and `place_id`, which assign each place specific IDs and could be used to filter position duplicates by e.g. removing all IDs for one place except one.

5.2 Image Content and Emotion Analysis

The Clarifai API delivers very good results in the recognition of images, as could be seen from many manual tests and comparisons of the image itself and the returned concepts. However, **different APIs** could maybe lead to slightly different classifications; the APIs and how they work kind of remain as blackboxes. The same holds true for the choice of the emotion dictionary: A hit rate of about 70% to 80% from the returned concepts within the dictionary is a good score, and many manual tests again showed that the interpretation and scoring is reasonable. However, **another emotion dictionary** or other method of interpreting emotions from words could lead to slightly different results in regard to the calculated pleasure and activity values. After all, the perception of images' and personal emotions through humans is a very subjective, abstract procedure and complicated to automate.

5.3 Recommender System

As demonstrated here, the recommender system explicitly asks the user about his emotion. Therefore, he has to decide how he feels at the moment. An **implicit collection of emotion data** could be possible with modern devices, e.g. with wearables, when a camera capable of recognizing the user's eye movements is pointed to the eye. The Pupil glasses [4] are capable of eye tracking, can be integrated into Virtual and Augmented Reality (VR and AR) hardware and could for instance be used to determine the user's emotion from this. Building an **explicit user profile** to optimize the recommendations over time, e.g. when the user gives feedback and evaluates previous recommendations on a star-based scale, as it has been done in the evaluation, would be a possible further enhancement.

Personal preferences or expectations could be taken into account as well, if the user can initially explicitly specify between **different operating modes**, before he is starting to use the system: Possible modes could be *business trip*, *vacation* - in these two modes, it is assumed that the user does not know the place and has not visited it before - or a

[4] `https://pupil-labs.com/`

local resident mode, when the system is used by someone who is already familiar with the surroundings; in that case, a recommender system can still provide recommendations for new, not already discovered places, or give at least tips for commuting. This idea has been mentioned also by participants in the study. The system could be refined in such a way that the user can for instance also select a price category or range in which he wants to receive recommendations for, e.g. for restaurants or accommodation in a city.

Besides location and emotion, i.e. mood, **additional parameters** could be used to improve the recommendations, such as time of day - recommendations might change when requested in the morning or in the evening; a location which has been photographed only in the evening might be more inviting at this time - or weather. Furthermore, recommendations could be based on the type of activity, e.g. indoor vs. outdoor activities. This requires some sort of additional recognition: The Flickr API, for instance, features the parameter `geo_context` for pictures, which holds the values 1 (taken indoors), 2 (taken outdoors) or 0 (undefined). 1 and 2 are only returned for a very small amount of pictures though.

Another technical difficulty and issue of the recommender system is not only the possibly small, but also possibly long distance between the recommendations: The recommender system suggests the three pictures fitting best to the emotion specified by the user, selected out of the 50 closest. However, the pictures are not equally distributed in the databases. This leads to a problem for mobility: The local distance between suggestions can be far, or the closeness of points to each other can be very high. This did not lead to any problems in the tests though. A possibility to configure the system in a way that there needs to be a **minimum (and maximum) distance between recommendations** could still be found.

5.4 User Interface

As of now the server is running on `localhost`. If the Python server would run **online**, the webpage would be accessible also from the browser of a smartphone with internet access (or within a mobile app), where the user's position could be easily automatically accessed via GPS, Bluetooth or WiFi, or - with some modifications - on a wearable device. This would lead to an architectural change: Right now, client and server are the same device, namely the computer on which the Python server is executed and the webpage is opened. In the other case, the client would only run the HTML page, while sending and receiving responses and requests from the remove Python server.

If used on a smartphone or especially wearable such as Google Glass [5] or its succeessor in the business area, the Glass Enterprise Edition [6], a possible use case would be to use **live pictures or video** from large amounts of users and dynamically tag this data. More portable (IoT) devices, like wearables or glasses, would mean a higher number of produced and relevant data. This scenario would open up the possibility for **real-time recommending**, possibly realized as a platform service with a cloud-based infrastructure: A busy day with many events happening in one place would lead to a high amount of recent, taggable pictures. Many pictures with a positive mood taken from the same location at the same time indicate a popular or busy location. The user could be notified and in this way get to know about an event happening right now. However, the scope of this application is big, and its implementation would be computationally intensive. Furthermore, such a service would require clear rules for security and privacy, as this would be an even bigger data-intensive application than the recommender system already is now in its current state.

Accessing powerful APIs with big databases, like the Google Places API [7], could **add relevant information about places**, such as opening times, popularity, ratings, price category and similar items, for instance other sights, categorized by restaurants, bars, shops, gas stations, etc., as done e.g. in Google Maps.

Another alternative instead of the removal of inappropriate pictures is MarkerCluster for Leaflet [8], consisting of CSS and JS. This Leaflet add-on can **solve the problem of pictures not being visible**, because they are overlayed by others when these are at the exact same location, i.e. position duplicates. However, this plugin is incompatible with the Leaflet sidebar.

[5] `https://developers.google.com/glass/`
[6] `http://www.x.company/glass/`
[7] `https://developers.google.com/places/`
[8] `https://github.com/Leaflet/Leaflet.markercluster`

Chapter 6

Summary

Within the scope of this thesis, a full emotion-based recommender system was built, combining many different components, techniques and technologies. Its emphasis is on the analysis and interpretation of emotions of users and in pictures, but the system also covers aspects from fetching the pictures to presenting the calculated data. The system has a client-server architecture and is built together from various sources like open source projects, API solutions, research work and studies. Tools, services and algorithms used for the processing were analyzed, discussed and compared to its alternatives in each step.

An internet connection at the client side is solely needed to display the interactive map and pictures from remote URLs or to create an own picture database; the data processing and analysis as well as the calculation of recommendations are done entirely directly on the server. This leads to a high recommendation speed and keeps the processing load very low for the client, which is especially wanted in a mobile environment, where computation and energy usage must be limited. In addition, the amount of data to be stored is very small on both client and server side; only the graphic in the user interface, the picture databases and possibly locally downloaded pictures use up space.

Figure 6.1 shows the architecture of the system and all its components, illustrated in a class diagram. It tries to visualize the whole application logic and all implemented parts with its respective functions and methods, together with the most important attributes of each part. The HTML client and Python server communicate with each other via Ajax. The user is able to access the capabilities of the backend via the JavaScript functions `flickr()` and `recommend()`; the respective methods are then called on the backend side, here also modeled as own objects in a hierarchy.

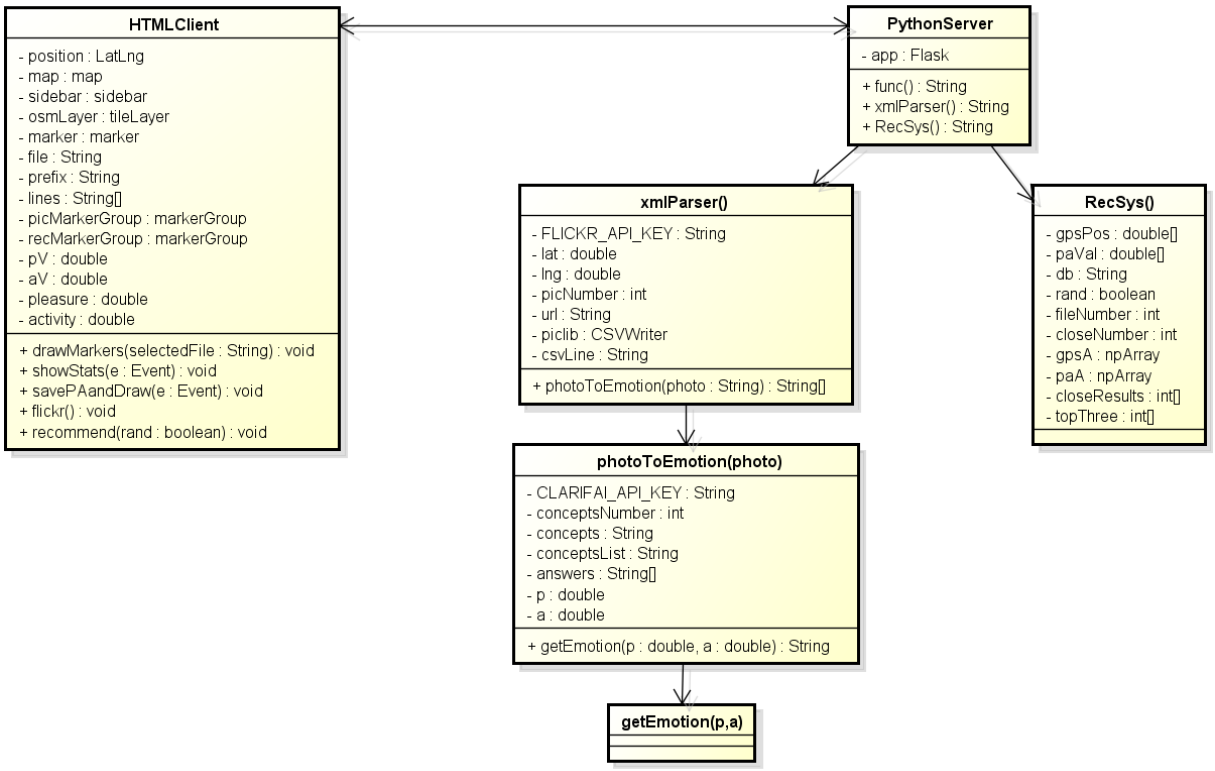


Figure 6.1: UML class diagram showing architecture and structure of the system

6.1 Conclusions

The thesis focused on the usefulness of the factor emotion, defined by the two dimensions pleasure and activity, for recommendations. The conducted study was able to proof that adding the emotion component can help to improve the recommendations, even though there are several aspects that make the process of recommending complicated, such as the subjectivity of emotions and the multistage procedure of analyzing and tagging pictures with emotions.

It would therefore definitely be beneficial for existing recommender systems with many features to include emotion in general and both the **emotion of users** and **emotions expressed in pictures** in particular as an extra factor for analyzing places to visit and calculating recommendations from them. In systems where recommendations are already based on other information as well, the use of emotion as an additional emotion component can significantly improve the results and especially their level of personalization.

Appendix: Usage of the Recommender System

There are a few possibilites to make the built application available for everyone to easily install and use it. Tools such as PyInstaller [1] are able to convert Python code to Windows executable binaries. Python virtual environments [2], originally meant to be used for testing Python code running under different versions of Python, can be provided with all required packages as well.

For the smallest overhead and the possibility to openly share this project on GitHub, the method of installing all required Python packages via a simple text file was used. The following is a copy of the file `README.md` in the repository `https://github.com/andi1996/emotion-recommender`. It is a brief summary of the project with its files, and contains instructions for installing and using it.

An emotion-based recommender system

This project implements an emotion-based recommender system, built within the scope of my bachelor's thesis **Emotion-based Recommender System for City Visitors Built on Analyzing Egocentric Images**. It recommends places to visit on a map, based on the emotion of pictures which were taken there, the location and the emotion of the user. It uses, amongst others, the Flickr API [3], the Clarifai API [4], an emotion dictionary [5], Leaflet [6], a sidebar for Leaflet [7], and Flask [8]. The frontend (`index.html`) is based on

[1] `http://www.pyinstaller.org/`
[2] `http://docs.python-guide.org/en/latest/dev/virtualenvs/`
[3] `https://www.flickr.com/services/api/`
[4] `https://clarifai.com/developer/quick-start/`
[5] `https://www.god-helmet.com/wp/whissel-dictionary-of-affect/index.htm`
[6] `http://leafletjs.com/`
[7] `https://github.com/Turbo87/sidebar-v2`
[8] `http://flask.pocoo.org/`

HTML and JavaScript, the backend (`app.py`) is based on Python. The communication between them is realized via Ajax.

How to get started

The project requires a **web browser** (successfully tested on Chrome and Firefox) and **Python 3.x** to be installed.

To begin, run `pip install -r requirements.txt --upgrade` once before the first run to install the newest versions of all required Python packages.

Note: The `clarifai` package installs an older version of the `requests` package, if installed manually, and not via the `requirements.txt` file; however, this version is working as well. If you are on Windows and installing the package returns an error, you might also need to install the Microsoft Visual C++ Build Tools [9], including Windows 8/8.1 SDK.

How to run the project

(1) Run `python app.py`
(2) Open `index.html`

All used files can be found in the folder `static`.

Note: The calculation of the first set of recommendations might take a few seconds, all following recommendations should be ready immediately.

Picture databases

- `flickr.csv` contains information about pictures from Flickr taken in and around Helsinki.
- `local.csv` contains information about pictures from Flickr taken in and around Munich.

Both databases can be found in the folder `static`.

Note: The database `local.csv` is meant to contain information about pictures which are all downloaded and stored locally in the folder `img`. For copyright reasons, the `img` folder does not contain any pictures. Instead, the pictures can be manually downloaded via the links in the file `links.txt`, e.g. by using a download manager.

[9] `https://go.microsoft.com/fwlink/?LinkId=691126`

Creating an own picture database

If you want to create your own picture database via the recommender system ("create your own by clicking **here**"), you need to set API keys for Flickr and Clarifai in `app.py`. They are specified in the variables `FLICKR_API_KEY` and `CLARIFAI_API_KEY`. The keys can be obtained for free from the services' websites. The database creation takes a few minutes, because 100 (default number) pictures need to be uploaded via the Clarifai API for the image content analysis. A message is shown on the commend line for each successfully analyzed picture, and the browser sends a notification when the database has been successfully stored in the file `flickr-own.csv`. You can rename it to `flickr.csv` and overwrite the old database to use the new one from now on.

Additional files

`xmlParser.py` and `RecSys.py` have the same capabilities as the respective functions called in `app.py`. They can be used with command line options, as shown within the files.

`csvParser.py` can analyze locally saved pictures and add the collected information to an existing picture database. An example can be found in the folder `example`.

`plots.py` can create plots to visualize the data contained in the picture databases.

List of Figures

List of Listings

Bibliography

[Adom 15] G. Adomavicius and A. Tuzhilin. *Recommender Systems Handbook*, Chap. Context-Aware Recommender Systems, pp. 191–226. Springer, 2015.

[Amit 14] Y. Amit and P. Felzenszwalb. *Computer Vision*, Chap. Object Detection, pp. 537–542. Springer, 2014.

[Ardi 11] L. Ardissono, T. Kuflik, and D. Petrelli. "Personalization in cultural heritage: the road travelled and the one ahead". *User Modelling and User-Adapted Interaction*, pp. 73–99, 2011.

[Beel 16] J. Beel, B. Gipp, S. Langer, and C. Breitinger. "Research-paper recommender systems: a literature survey". *International Journal on Digital Libraries, Volume 17, Issue 4*, pp. 305–338, 2016.

[Bell 07] V. Bellotti and B. Begole. "Activity-Based Serendipitous Recommendations with the Magitti Mobile Leisure Guide". *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1157–1166, 2007.

[Burk 07] R. Burke. *The Adaptive Web*, Chap. Hybrid Web Recommender Systems, pp. 377–408. Springer, 2007.

[Camp 12] A. Campbell and T. Choudhury. "From Smart to Cognitive Phones". *IEEE Pervasive Computing, Volume 11, Issue 3*, pp. 7–11, 2012.

[Cho 11] E. Cho, S. A. Myers, and J. Leskovec. "Friendship and Mobility: User Movement in Location-Based Social Networks". *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1082–1090, 2011.

[Chou 08] T. Choudhury *et al.* "The Mobile Sensing Platform: An Embedded Activity Recognition System". *IEEE Pervasive Computing, Volume 7, Issue 2*, pp. 32–41, 2008.

[Cran 09] D. Crandall, L. Backstrom, D. Huttenlocher, and J. Kleinberg. "Mapping the World's Photos". *WWW*, pp. 1–10, 2009.

[Ekma 99] P. Ekman. *Handbook of Cognition and Emotion*, Chap. Basic Emotions, pp. 46–60. John Wiley & Sons Ltd., 1999.

[Gava 12] D. Gavalas, M. Kenteris, C. Konstantopoulos, and G. Pantziou. "Web application for recommending personalised mobile tourist routes". *IET Software, Volume 6, Issue 4*, pp. 313–322, 2012.

[Gava 14] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou. "Mobile Recommender Systems in Tourism". *Journal of Network and Computer Applications, Volume 39*, pp. 319–333, 2014.

[Ge 10] Y. Ge, H. Xiong, A. Tuzhilin, K. Xiao, M. Gruteser, and M. J.Pazzani. "An Energy-Efficient Mobile Recommender System". *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 899–908, 2010.

[Gilb 11] A. Gilbert, J. Illingworth, and R. Bowden. "Action Recognition Using Mined Hierarchical Compound Features". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 883–897, 2011.

[Gira 08] F. Girardin, J. Blat, F. Calabrese, F. D. Fiore, and C. Ratti. "Digital Footprinting: Uncovering Tourists with User-Generated Content". *IEEE Pervasive Computing, Volume 7, Issue 4*, pp. 36–43, 2008.

[Gome 15] C. A. Gomez-Uribe and N. Hunt. "The Netflix Recommender System: Algorithms, Business Value, and Innovation". *ACM Transactions on Management Information Systems, Volume 6, No. 4*, pp. 1–19, 2015.

[Gu 09] T. Gu, Z. Wu, X. Tao, H. K. Pung, and J. Lu. "epSICAR: An Emerging Patterns based Approach to Sequential, Interleaved and Concurrent Activity Recognition". *IEEE International Conference on Pervasive Computing and Communications*, pp. 1–9, 2009.

[Hasa 14] M. Hasan, E. A. Rundensteiner, and E. Agu. "EMOTEX: Detecting Emotions in Twitter Messages". *Academy of Science and Engineering*, pp. 1–10, 2014.

[Hodg 07] M. Hodges and M. Pollack. "An 'Object-Use Fingerprint': The Use of Electronic Sensors for Human Identification". *International Conference on Ubiquitous Computing*, pp. 289–303, 2007.

[Hu 08] Y. Hu, Y. Koren, and C. Volinsky. "Collaborative Filtering for Implicit Feedback Datasets". *IEEE 8th International Conference on Data Mining*, pp. 263–272, 2008.

[Jamb 10] T. Jambor and J. Wang. "Optimizing Multiple Objectives in Collaborative Filtering". *Proceedings of the fourth ACM conference on Recommender systems*, pp. 55–62, 2010.

[Jann 10] D. Jannach, M. Zanker, and A. Felfernig. *Recommender Systems*, Chap. Hybrid Recommendation Approaches, pp. 124–142. Cambridge University Press, 2010.

[Kent 07] M. Kenteris, D. Gavalas, and D. Economou. "An innovative mobile electronic tourist guide application". *Personal and Ubiquitous Computing, Volume 13, Issue 2*, pp. 103–118, 2007.

[Khan 13] W. Z. Khan, Y. Xiang, M. Y. Aalsalem, and Q. Arshad. "Mobile Phone Sensing Systems: A Survey". *IEEE Communications Surveys & Tutorials, Volume 15, Issue 1*, pp. 402–427, 2013.

[Khos 09] M. Khosravy and L. Novik. "Portal services based on interactions with points of interest discovered via directional device information". 2009.

[Kim 17] H.-R. Kim, Y.-S. Kim, S. J. Kim, and I.-K. Lee. "Building Emotional Machines: Recognizing Image Emotions through Deep Neural Networks". *Computing Research Repository*, pp. 1–11, 2017.

[Labe 14] S. Labeznik. *Computer Vision*, Chap. Object Class Recognition (Categorization), pp. 533–536. Springer, 2014.

[Lane 10] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. "A Survey of Mobile Phone Sensing". *IEEE Communications Magazine, Volume 48, Issue 9*, pp. 140–150, 2010.

[Lath 15] N. Lathia. *Recommender Systems Handbook*, Chap. The Anatomy of Mobile Location-Based Recommender Systems, pp. 493–510. Springer, 2015.

[Lei 09] Z. Lei and P. Coulton. "A mobile Geo-wand Enabling Gesture based POI Search an User Generated Directional POI Photograph". *Proceedings of the International Conference on Advances in Computer Entertainment Technology*, pp. 392–395, 2009.

[Li 14] W. Li, Z. Liu, and Z. Zhang. *Computer Vision*, Chap. Activity Recognition, pp. 13–16. Springer, 2014.

[Lill 13] K. Lillywhite, D.-J. Lee, B. Tippetts, and J. Archibald. "A feature construction method for general object recognition". *Pattern Recognition, Volume 46, Issue 12*, pp. 3300–3314, 2013.

[Lind 98] G. Linden, J. Jacobi, and E. Benson. "Collaborative recommendations using item-to-item similarity mappings". 1998.

[Mach 10] J. Machajdik and A. Hanbury. "Affective Image Classification using Features Inspired by Psychology and Art Theory". *Proceedings of the 18th ACM international conference on Multimedia*, pp. 83–92, 2010.

[McCa 06] K. McCarthy, M. Salamó, L. Coyle, L. McGint, B. Smyth, and P. Nixo. "Group Recommender Systems: A Critiquing based Approach". *Proceedings of the 11th international conference on Intelligent user interfaces*, pp. 267–269, 2006.

[McDu 13] D. McDuff, R. el Kaliouby, T. Senechal, M. Amr, J. F. Cohn, and R. Picard. "Affectiva-MIT Facial Expression Dataset (AM-FED): Naturalistic and Spontaneous Facial Expressions Collected "In-the-Wild"". *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 881–888, 2013.

[Melv 17] P. Melville and V. Sindhwani. *Encyclopedia of Machine Learning and Data Mining*, Chap. Recommender Systems, pp. 1056–1066. Springer, 2017.

[Neid 14] J. Neidhardt, L. Seyfang, R. Schuster, and H. Werthner. "A picture-based approach to recommender systems". *Information Technology & Tourism, Volume 15, Issue 1*, pp. 49–69, 2014.

[Nguy 16] T.-H.-C. Nguyen, J.-C. Nebel, and F. Florez-Revuelta. "Recognition of Activities of Daily Living with Egocentric Vision: A Review". *Sensors (Basel), Volume 16, Issue 1*, pp. 72:1–24, 2016.

[Nogu 12] J. M. Noguera, M. J. Barranco, R. J. Segura, and L. Martínez. "A Mobile 3D-GIS Hybrid Recommender System for Tourism". *Information Sciences, Volume 215*, pp. 37–52, 2012.

[Noul 11] A. Noulas, S. Scellato, C. Mascolo, and M. Pontil. "An Empirical Study of Geographic User Activity Patterns in Foursquare". *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media*, pp. 570–573, 2011.

[Noul 12a] A. Noulas, S. Scellato, N. Lathia, and C. Mascolo. "Mining User Mobility Features for Next Place Prediction in Location-based Services". *IEEE 12th International Conference on Data Mining*, pp. 1038–1043, 2012.

[Noul 12b] A. Noulas, S. Scellato, N. Lathia, and C. Mascolo. "A RandomWalk Around the City: New Venue Recommendation in Location-Based Social Networks". *International Conference on Social Computing*, pp. 144–153, 2012.

[Nova 15] P. K. Novak, J. Smailovic, B. Sluban, and I. Mozetic. "Sentiment of Emojis". *PLoS ONE, Volume 10, Issue 12*, pp. 1–22, 2015.

[Park 07] M.-H. Park, J.-H. Hong, and S.-B. Cho. "Location-Based Recommendation System Using Bayesian User's Preference Model in Mobile Devices". *Proceedings of the 4th international conference on Ubiquitous Intelligence and Computing*, pp. 1130–1139, 2007.

[Piya 13] L. Piyathilaka and S. Kodagoda. "Gaussian Mixture Based HMM for Human Daily Activity Recognition Using 3D Skeleton Features". *IEEE 8th Conference on Industrial Electronics and Applications*, pp. 567–572, 2013.

[Poll 03] M. Pollack, L. Brown, D. Colbry, C. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan, and I. Tsamardinos. "Autominder: An Intelligent Cognitive Orthotic System for People with Memory Impairment". *Robotics and Autonomous Systems, Volume 44, Issues 3-4*, pp. 273–282, 2003.

[Posn 05] J. Posner, J. A. Russell, and B. S. Peterson. "The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology". *Development and psychopathology*, pp. 715–734, 2005.

[Quer 10] D. Quercia, N. Lathia, F. Calabrese, G. D. Lorenzo, and J. Crowcroft. "Recommending Social Events from Mobile Phone Location Data". *IEEE International Conference on Data Mining*, pp. 971–976, 2010.

[Quer 11] D. Quercia, I. Leontiadis, L. McNamara, C. Mascolo, and J. Crowcroft. "SpotME If You Can: Randomized Responses for Location Obfuscation on Mobile Phones". *31st International Conference on Distributed Computing Systems*, pp. 363–372, 2011.

[Rach 10] K. Rachuri, C. Mascolo, and M. Musolesi. "Energy-Accuracy Trade-offs of Sensor Sampling in Smart Phone based Sensing Systems". *Mobile Context Awareness*, pp. 65–76, 2010.

[Rach 11] K. Rachuri, C. Mascolo, M. Musolesi, and P. Rentfrow. "SociableSense: Exploring the Trade-offs of Adaptive Sampling and Computation Offloading for Social Sensing". *Proceedings of the 17th annual international conference on Mobile computing and networking*, pp. 73–84, 2011.

[Raja 18] M. Raja, A. Exler, S. Hemminki, S. Konomi, S. Sigg, and S. Inoue. "Towards pervasive geospatial affect perception". *GeoInformatica, Volume 22, Issue 1*, pp. 143–169, 2018.

[Ricc 07] F. Ricci and Q. N. Nguyen. "Acquiring and Revising Preferences in a Critique-Based Mobile Recommender System". *IEEE Intelligent Systems, Volume 22, Issue 3*, pp. 22–29, 2007.

[Ricc 15] F. Ricci, L. Rokach, and B. Shapira. *Recommender Systems Handbook*, Chap. Recommender Systems: Introduction and Challenges, pp. 1–35. Springer, 2015.

[Robe 12] K. Roberts, M. A. Roach, J. Johnson, J. Guthrie, and S. M. Harabagiu. "EmpaTweet: Annotating and Detecting Emotions on Twitter". *Proceedings of the Eight International Conference on Language Resources and Evaluation*, pp. 3806–3813, 2012.

[Sava 12] N. S. Savage, M. Baranski, N. E. Chavez, and T. Höllerer. *Advances in Location-Based Services*, Chap. I'm feeling LoCo: A Location Based Context Aware Recommendation System, pp. 37–54. Springer, 2012.

[Sche 15] M. Schedl, P. Knees, B. McFee, D. Bogdanov, and M. Kaminskas. *Recommender Systems Handbook*, Chap. Music Recommender Systems, pp. 453–492. Springer, 2015.

[Schi 04] J. Schiller and A. Voisard. *Location-Based Services*. Morgan Kaufman Publishers, 2004.

[Skla 12] M. Sklar, B. Shaw, and A. Hogue. "Recommending Interesting Events in Real Time with Foursquare Checkins". *Proceedings of the sixth ACM conference on Recommender systems*, pp. 311–312, 2012.

[Sten 11] L. Stenneth, O. Wolfson, P. S. Yu, and B. Xu. "Transportation Mode Detection using Mobile Phones and GIS Information". *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 54–63, 2011.

[Sutt 13] J. Suttles and N. Ide. "Distant Supervision for Emotion Classification with Discrete Binary Values". *CICLing 2013: Computational Linguistics and Intelligent Text Processing*, pp. 121–136, 2013.

[Vico 11] D. G. Vico, W. Woerndl, and R. Bader. "A Study on Proactive Delivery of Restaurant Recommendations for Android Smartphones". *Proceedings of the fifth ACM conference on Recommender systems*, pp. 273–276, 2011.

[Whis 09] C. Whissell. "Using the Revised Dictionary of Affect in Language to Quantify the Emotional Undertones of Samples of Natural Language". *Psychological Reports, Volume 105, Issue 2*, pp. 509–521, 2009.

[Yoo 15] K.-H. Yoo, U. Gretzel, and M. Zanker. *Recommender Systems Handbook*, Chap. Source Factors in Recommender System Credibility Evaluation, pp. 689–714. Springer, 2015.

[You 16] Q. You, J. Luo, H. Jin, and J. Yang. "Building a Large Scale Dataset for Image Emotion Recognition: The Fine Print and The Benchmark". *Thirtieth AAAI Conference on Artificial Intelligence*, pp. 308–314, 2016.

[Zhen 08] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma. "Understanding Mobility Based on GPS Data". *Proceedings of the 10th international conference on Ubiquitous computing*, pp. 312–321, 2008.