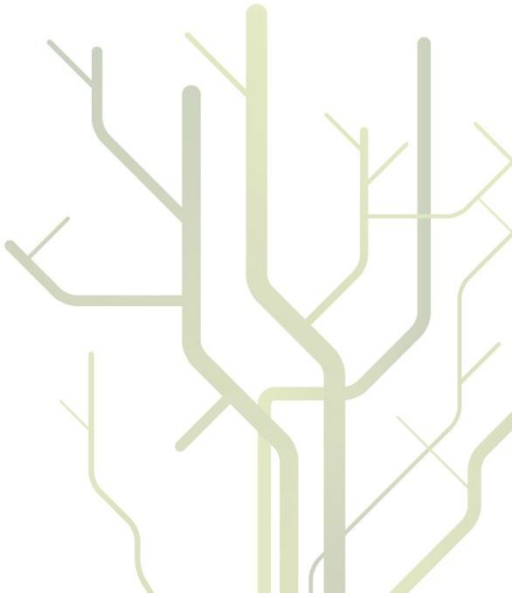
UNIVERSITY OF TROMSØ UIT

FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Summarizing image collections

David Sundby

Inf-3981
Master's Thesis in Computer Science
June, 2011



Abstract

The widespread use of digital cameras and mobile phones, along with a rapidly growth of image sharing, challenges current image retrieval techniques. It is difficult for image retrieval system to find the semantic meaning of images based on human's subjectivity and the size of current image databases makes it difficult to organize and search images.

This thesis shows that information retrieval techniques can be used to reduce the search space of existing image collections, by creating collection summaries that holds only the most representative metadata from existing image collections. The representative metadata are metadata that are most distinguishing for the specific image collection. The system take advantage of the metadata available for images, which includes user provided tags, date/time, GPS coordinates, and metadata augmented by the system using already available metadata. Higher level representations of user provided terms are located and ensures that the system captures the most descriptive properties of the image collections.

The system designed is able to produce collection summaries that capture properties of an image collection that support human's natural perception as long as enough metadata are available. Also, the system increase the contextual understanding of images as long as date/time (of capture) and GPS coordinates is available for all or some of the images in the collection.

The evaluation of the system indicates that grouping user provided terms into higher level representations is very useful for capturing the most important properties of an image collection. Also the evaluation expresses the usefulness of augmenting images with additional metadata and converting numeric metadata into readable terms. The comparisons made in the evaluation indicates that not only are the collection summaries similar to individual test users perception of image collections, but the summaries also includes more descriptive information of relevance.

Acknowledgements

To start with I would like to express gratitude to my supervisor, associate professor Randi Karlsen, for giving me the idea of this work and helping me along the way. Your guidance, constructive criticism, motivation and availability have been above all expectations and are greatly appreciated.

I would also like to express gratitude to my girlfriend, Lillann Sofie Thorstensen, for giving me great support and taking care of our son, Theo, through the hardworking month's laid down in this work. Your support, motivation, presence and loving words along the way are much appreciated. I love you.

Finally, I would like to express gratitude to my fellow student, Martin H. Evertsen, for making all these years of study unforgettable, both at University of Tromsø and at Lund University. We have collaborated, argued, struggled, supported and helped each other, laughed, partied, and had some interesting discussions of life through these years. Your friendship is highly appreciated.

Contents

List of figures and tables

Chapter 1

1. Introduction

1.1. Motivation

The widespread use of digital cameras has facilitated a rapidly increasing number of images in privately and publicly owned collections. Numerous images are currently available over the Internet for searching and browsing and the number of users and application areas are increasing. The huge amount of digital images challenges the current techniques used for organizing and retrieving images. There is a demand for new and more efficient technologies for image searching and browsing.

Users have a very abstract idea of what they are looking for when searching for images. Current image retrieval systems have tried to solve this problem by allowing image retrieval based on e.g. *semantic concepts*. Current techniques are still not sufficient to automatically comprehend the semantic meaning of images on the basis of human perception. This problem is known as the *semantic gap* problem and usually arises when human activities are transferred into a computational representation [1].

Many different approaches are used within image retrieval; the main approaches are *content based (CBIR)* and *text based image retrieval (TBIR)* [2].

Text based image retrieval (TBIR) will be the focus of this thesis. TBIR are focused on text connected with the images or in relation with the images. For text based image retrieval one typically uses *metadata* to describe images. Metadata can be defined as *data about data* [3]. Image metadata can be of different kinds, e.g. tags, keywords and descriptors of relevance for the image. This includes data added by the capturing device, e.g. time/date and GPS coordinates, keywords manually added by individual users to describe the image (*tags*) or *automatic image annotations* added by the image retrieval system to simplify search and indexing. The latter is usually referred to as *auto- annotations* or linguistic indexing [4].

Also efforts have been made by adding semantics to image *tags* by mapping them to *ontologies*. An ontology is a higher level description of terms and captures the *semantic meaning* of the specific word [5].

Text based image retrieval are much more suitable than CBIR techniques for the increasing number of images current image retrieval techniques are faced with. TBIR require less computer resources and are far more natural in supporting humans conceptual refined search queries than CBIR techniques.

Recent trends in *text based image retrieval* are to combine different context metadata to augment images with additional metadata from various sources [6-9].

Context is any information that can be used to characterize the situation of an entity [10]. In most image retrieval systems the entity is usually the capturing device. Context types that can help characterize the situation of the image are e.g. GPS location (i.e. latitude and longitude) and date/time. Augmenting images with additional metadata, can increase the knowledge about the images and may also help in closing the *semantic gap* in a more reliable and efficient manner. Locational information can alone sometimes suggest the semantic content of images [11, 12].

Manually added tags can be very helpful for the retrieval system, if available. Manually added tags are keywords added to the image by individual users. In theory they represent the individuals' natural perception of the image. One concern that remains in dealing with manually added tags is that the tagging patterns of humans are *subjective*, which means that individual users have different perceptions, and use different tags to describe the images [13]. The motivation for this can include factors such as social, contextual, time, cultural and so on. Also *collaborative tagging* has recently grown in popularity on community based image sharing sites [14], which helps researchers to great extents in understanding humans perception or adapting tagging patterns to other images [6, 15].

As modern technology has made capturing devices, i.e. cameras and mobile phones, more affordable, more portable and more accessible, the usage of photo capturing devices has increased exponentially. Also people share more images with each other through public image sites than ever before. Even if *text based image retrieval* are more suitable for the numerous amounts of images available, still the task of searching them, browsing through tags or features found in the individual images, looking for the best matches are very time and resource consuming. Also because of noisy, user specific and inconsistent metadata for the images in large image collections, search queries can lead to results in sizes that are colossal. Hence, it is important to scale down the search space, focusing on only the most relevant ones. Text based retrieval techniques are well established in the field of *Information Retrieval* (IR). The techniques have matured for decades with the goal of filtering out the most important textual documents in relation with specific user queries. Therefore, text based image retrieval has great potential in adapting the techniques and approaches used within IR.

1.2. Goal

The specific goal of this project is to design, implement and test a system that automatically generates a description of an image collection based on available image metadata. A collection are viewed as pre-existing, e.g. in the form of hierarchically structured images on personal computers, images on certain online domain specific image collections, blogs, online sharing sites etc.

To be able to handle the huge amount of images the system will analyse the metadata for all images in the collection with the intention of finding key characteristics. The key characteristics are viewed as representatives for the

image collection, with properties that stand out from others and work as a reduced search space for the image collection.

The representatives of an image collection are together used to create a summarized description of the image collection. The descriptive collection summary can later be used to efficiently filter out irrelevant collections and select the relevant ones (during an image search). The idea is that within the huge amount of image collection currently available there are always some that are more relevant for a specific search query than others; the trick is to be able to locate them in an effective way. This will also help streamline the system to a much greater extent. Collections that are found to be relevant can be further investigated to decrease the search space until a set of images as relevant as possible for the search query are the only ones left.

The aim of this project is to produce image collection summaries and make them available for efficient image retrieval. As current Image retrieval techniques are challenged to succeed with the ever increasing growth of images, I will in in this project purely focus on information directly concerned with the image through metadata. Further already available metadata will be used to accumulate additional semantic metadata for the images. This project is limited to the analysis of metadata and production of the reduced search space and not on the actual image retrieval itself.

1.3. Approach

The system designed and implemented in this thesis is used to find dominating metadata for collections analysed, i.e. properties that stand out from others. The system developed will use text based image retrieval techniques and adopt common approaches used within Information retrieval (IR). Important adoptions include ways to cleanse data and dropping metadata that provides little value for the retrieval system, selection and weighting of the metadata and embracing efficient indexing techniques.

Manually added tags will be grouped together into categories. The categories represents a higher level representation of terms (ontologies) connected with the images (e.g. the terms *car, suv, motorcycle,* has a the higher level representation "motor vehicle" in common, the higher level representation of "motor vehicle" are "self-propelled vehicle" and so on). Categories are used to strengthen terms through its category members, and used to find hierarchical thresholds which reflect the terms at their highest level.

Images will also take advantage of available metadata, such as Global position system (GPS), i.e. location (of the capturing device) and date/time (of when the image are taken) to augment the images with additional contextual metadata. This can enhance our knowledge of the images and may contribute to closing the *semantic gap* and improving effectiveness of image retrieval.

Also numeric metadata will be converted into more searchable terms such as day of the month (e.g. Saturday), season (e.g. summer), and month (e.g. January) etc.

Notably, the images analysed by the system do not require that all images within a collection have manually added tags, GPS coordinates and date/time stamps included, but take advantage of them if they are available. Nevertheless, to perceive the full potential of the system, the majority of the images used in the evaluation of this system are provided with all of these metadata fields. The image collections used are community based, collaborative tagged image collections provided with GPS coordinates gathered from Flickr[1].

Finally, after cleansing and removal, categorizing of terms, augmenting of contextual metadata and converting of metadata has been performed, the finalized metadata will be analysed and representatives will be located before the collection summary is produced.

1.4. Contribution

A number of systems have been developed for reducing the search space of large sets of images. I will discuss this further in section 3.2. Nevertheless, I have not found any previous work that use a combination of augmentation, categorization and finding higher level representations of metadata before locating and summarizing the most important properties of existing image collections. Therefore, the contribution of this thesis is to explore the possibility of reducing the search space for existing image collections, while at the same time capturing the most important properties of them. A reduced search space is captured in a collection summary which holds the most representative metadata of the images within the image collection. Further important aspects of this work will be to evaluate if the system developed will be beneficial for more efficient retrieval, both in terms of resources and results returned.

This master thesis is part of the Context-Aware Image Management[2] (CAIM) project. The CAIM project is focused on research and the development of tools and methods supporting context-aware image management, both in distributed and mobile environments. The research of the CAIM project is built on previous research at University of Tromsø, University of Bergen, Munich University of Technology, University of Hawaii and Telenor R&D.

1.5. Organization

The rest of this thesis is organized as follows. Some important background information is presented in chapter 2, followed by previous relevant work in chapter 3. The approach and design used in the developed solution is presented in chapter 4. Further some implementation specifics are presented in chapter 5, before the results and evaluation of the system developed are presented in chapter 6. Some potential future work is presented in chapter 7, before finally concluding in chapter 8. The location of the images used in

[1] http://www.flickr.com/
[2] http://caim.uib.no/

the evaluation is listed in appendix A, and the source code of the system in appendix B.

Chapter 2

2. Background

In this chapter I will present theory that is relevant for the further work of my project. I will start by introducing image retrieval and current approaches to the field (section 2.1). As the system developed in this thesis is focused on text based image retrieval, Information retrieval is highly relevant and presented in section 2.2.2. Further metadata, auto annotations, context and semantics are presented in section 2.3. Dealing with huge amount of data are easier to handle and work with when putting them into a database, some theory highly representative for databases are finally presented in section 2.4.

2.1. Image retrieval

Image retrieval is a system that is used for searching, browsing and retrieving images from a large repository of images [16].

2.1.1. Concept and content based

Current image retrieval systems are usually either text based (concept based) or content- based.

Image search engines such as Google Image[3] and Microsoft's Bing Image search[4] rely almost only on the text surrounding the Image. [17] In such systems, the surrounding text is analysed and the words or descriptions that for the system appear to be relevant, or semantically meaningful for the image, are included with the image in the image retrieval system. In current document retrieval systems analysing and indexing text documents for search has reached great heights. However using the same principles for image retrieval systems, depending on the surrounding text of the images, is not as successful often leading to noisy results. Within text based retrieval several forms of metadata can be seen as textual information important for the retrieval system. E.g. manually annotated tags can be very helpful. Manually added tags are keywords added for the image by individual users and in theory represent the individual's natural perception of the image. This metadata should be taken advantage of by the retrieval system if available. Anyhow the perception of humans are *subjective*, which in this setting means that individual users have different perceptions, hence use different tags to describe the images [13] depending on different factors such as, social, contextual, time, cultural and so on. Also collaborative tagging has recently grown in popularity on community based shared image sites [14], which can also be viewed as relevant for text based image retrieval.

[3] http://images.google.com/

[4] http://www.bing.com/?scope=images

In Content- based image retrieval systems (CBIR) images are usually retrieved using an example image as a query. The content of the query image is used to find similar images with respect to visual similarities, i.e. shape, texture and colours [1]. While this works well for certain domains, e.g. face recognition and fingerprint lookup, it is not convenient for large and complex image collections where advanced texture and shapes can make the system potentially faulty.

Pham [18] suggest that current CBIR techniques can mostly extract low-level features from images, while humans tend to associate images with high level concepts used in everyday life. Vailaya et. al. [19] shares a similar view. Also Enser [13] supports this claim noting that people's perception can mean different things at different times and at different contexts, and that no existing CBIR system can precisely classify categories in cases where the user- needs are not well defined and well understood. Further Pu [20] studies failed queries for online image retrieval and address the shortcomings in current image retrieval techniques. His main points are that user queries often does not include information included in the image itself and that there are gaps between user queries and current Image retrieval techniques. The work done by Pu is useful to be used as a conceptual model, when investigating the gaps between image queries and current image retrieval techniques. For example he notes that using CBIR techniques alone, which focus on perceptual image attributes, cannot satisfy requests for unique names, e.g. a person such as 'Arnold Schwarzenegger', or locational names such as 'France'. Further Pu refines that the study of failed queries suggests that users query images with far more conceptual than perceptual refined types. The latter, i.e. perceptual is the focus of CBIR techniques. These claims also suggest the gaps between human and computer perception that are left in CBIR systems and that image retrieval are in need of using different more conceptual approaches. Quite a great deal of CBIR techniques has been developed, and great results have been produced, still no universally acceptable image retrieval techniques has been developed [21].

More recent trends [6-9] use techniques for augmenting images with additional metadata using metadata already available for the images. Several user studies suggest the importance of textual information in image retrieval [22, 23]. The solution presented in this thesis does not use any CBIR techniques, but are focused on metadata connected with the image. This means that the system practice concept/ text based image retrieval, which makes *Information retrieval* techniques and approaches highly relevant. Information retrieval will be further presented in section 2.2.

2.1.2. Summarizing image collections

As current image collections grow enormous, both on people's personal computers and on Internet, challenges are introduced for current Image retrieval techniques. Large collections of images are often unstructured, and also the huge amounts of images which are daily rapidly increasing makes the task of organizing and searching them very difficult, not to mention very time and resource consuming.

It is difficult to find techniques and methods with direct focus of summarizing image collections, although some can be seen as relevant. Vailaya et. al [19] presents a CBIR system that use techniques for filtering out partitions of image collections. Conferring to Vailaya et. Al. within CBIR, *categorization* has been used to improve the performance of image retrieval systems, by filtering out the irrelevant classes and indexing on high level visual features in an image such as objects, e.g. trees, sky, people, water etc.

Nevertheless, in this thesis the main focus is on the metadata connected with the image, i.e. text based image retrieval, and not on the visual content of the images as in CBIR. For these reasons information retrieval (IR) techniques, methods and approaches will be used when scaling down and reducing the search space in an efficient manner. The motivation for this is not only to increase the speed of image retrieval systems but also to increase the relevancy of images returned.

Text based retrieval techniques are well established in the field of IR, and as more textual information are available for an image, IR can be seen as closely related and highly relevant for text based image retrieval. I will present Information retrieval and some relevant theory from the field in section 2.2, but first I will present some theory relevant for the selection of representative features as candidates for the collection summary.

2.1.2.1. Feature selection

In information theory, techniques for reduction of the search space are well established. E.g. Principal component analysis (PCA) reduces the space by analysing, say a sequence such as a text document, revealing the internal structure of the data in it and describe the data from its most informative view. PCA is an instance of the class of analysis algorithms called common factor analysis, which seeks the least number of dimensions [24].

Features or properties may be chosen important for several reasons, e.g. domain specific, discrimination specific (e.g. in face recognition, partition out images with certain hair colours), user relevance, location relevance, and so on [1]. The simplest form of reducing the search space is by *term frequency*. In information theory, this approach is usually used to count each word in a document to decide which words that are most frequently used and weight the documents to keyword searches according to the word frequency in the documents [1] (described further in section 2.2.2).

Weighting metadata in this way is also very relevant for text based image retrieval and may be used to locate the highly weighted and highly representative features, i.e. terms, to be part of the collection summary. One concern with the term frequency approach is that it does not capture the semantic meaning of the document as in PCA. Even though we have efficient techniques for reduction and description of data in information theory, it is difficult to transfer these techniques to image retrieval, simply because we usually do not have enough descriptive and continuous textual information connected with an image. Without enough textual information it is difficult to capture the semantic meaning or distinguished features

through analysis as in information theory, so other approaches have to be used. In the following section I will give a better introduction to information retrieval before presenting some theory relevant to semantics and augmenting images with additional textual information in section 2.3.

2.2. Information retrieval

The meaning of the term *information retrieval* are very wide ranging, but in relation to computer science a general definition provided by Manning et. Al. [25] is:

> *Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).*

In information retrieval, one usually wants to collect the documents to be indexed, tokenize the words or terms within them, cleanse the tokenized words and index them for efficient retrieval. In the next sub section I will present the major steps in the preparation of documents with goal to construct an index as presented by Manning et. Al.

2.2.1. Index construction

1. Obtain the documents to be Indexed:

When obtaining the actual text from the documents, translations have to be made. The documents can be in several different formats, e.g. .doc, .pdf, .odf and so on, which are represented as a byte sequence. Such byte sequences have to be transformed into a linear *sequence of characters*. Not only document type, i.e. byte sequence translation of the actual text, are important in this step, but also determination of encoding, e.g. ASCII or Unicode UTF-8. When the text has been transformed defining the *document unit* is the next phase, e.g. store each file in a folder as a document or separate it into several documents to be indexed.

2. Tokenize the text

Given a *sequence of characters* and a *document unit* the next step is to *tokenize*. The tokenization is the task of chopping the text into several tokens, discarding certain characters such as punctuations. Tokens are the terms or words within the document. For example:

Input: The man walked down the street.

Output: | The | man | walked | down | the | street |

3. Dropping common terms. (stop words)

In this step common terms are dropped. These are terms that provide little value in helping the user get a match on search queries, and are often

referred to as *stop words*. These can either be located by filtering out the most common terms through term frequencies (how frequent words are used within a document) or manually providing a common term list of terms to be removed. Examples of *stop words* that are desired for removal by an IR system are:

a, an, and, are, as, at, be, by, for, in and so on.

4. Normalization (equivalent classes of terms)

In many situations tokens (words) are not identical, but still a match is desired to occur. E.g. if you search for the term *pre-processing* you would like a match to occur on both documents containing *preprocessing* and *pre-processing*, or if you search for *USA* you want matches on both USA and *U.S.A.* Token normalization is the process of converting tokens so that such matches occur, despite the difference in the character sequence.

Another way of normalization is translating accents, e.g. creating mappings between *cliché* and *cliche*, or mappings between languages *Malaysie* (French for Malaysia) and *Malaysia.*

Also capitalizing / case folding character sequences can be viewed as highly relevant for normalization, e.g. lowercasing all tokens so that matches between e.g. *France* and *france* are made.

5. Stemming and lemmatization

Documents will contain different forms of a word, e.g. *organize, organizes* and *organizing*. Also there are grammatical different words with similar meaning e.g. *democracy*, *democratic* and *democratization*. The goal of both stemming and lemmatization is to reduce inflectional forms and words that are similar to a common form, such a reduction can be exampled as:

1. Man, men, boy, boys → Male

2. Car, cars, car's, cars' → car

6. Index the documents that each term occurs in

The documents are indexed to make the process of finding documents best matching a search query more effective. A common approach to weighting the importance of tokens is presented in the next sub section (2.2.2). Also I will present a common index structure in section 2.2.3.

2.2.2. Term Frequency- Inverse Document Frequency (TF-IDF)

When step 1 to 5 from the process described above is performed, i.e. when the documents are ready for indexing, the document is usually weighted according to their relevance to different search queries. Term frequency-inverse document frequency (TF-IDF) is one such approach. TF-IDF is a method used in information retrieval and text mining [1]. The *term*

frequency is used to weight a document according to how frequently a specific word is used in relation to how many words there are within the document totally. E.g. if a document contains 100 words and the word *car* appears 5 times in the text, the term frequency for *car* in this document is:

(1) TF(*car*) = 5/100 = 0.05.

The *Inverse document frequency* (IDF) is used to weight the word in relation with all documents available. If few of the documents available contain the specific word, the IDF are higher than if many documents contain the word. The inverse document frequency is calculated with a logarithmic equation taking into consideration the number of documents available and how many of them contain the specific word. E.g. say that 100 documents are available and 5 of these contain the word *car*, then the IDF for the term *car* is:

(2) IDF(*car*) = log(100 / 5) = 2.996.

The TF-IDF is the product of the *term frequency* and the *inverse document frequency*, i.e:

(3) TF-IDF = TF * IDF.

In the case of the term *car* for the given documents, the *term frequency – inverse document frequency* is:

(4) TF-IDF = (1) * (2) = 0.05 * 2.996 = 0.15.

2.2.3. Inverted Files Indexing

An inverted files index is a search structure usually used in information retrieval, but it is has also become a popular way of indexing in recent large scale image retrieval systems [9]. The inverted file structure is good candidate for large scale image retrieval system because of its superiority in efficiency. The search structure holds mappings between terms in a database and the location of the document(s) containing the specific term(s). The inverted index structure is structured as follows: [1]

- A term directory keeps track of all words that are present in a database. Each word holds a pointer to its corresponding inverted list. Also the number of documents containing the word is maintained in the directory.
- The inverted lists are usually held in a postings file which holds the actual pointers to the documents. Inverted lists are usually stored continuously in the postings file to reduce disk access.
- Also a search structure is usually used to efficiently access the directory of inverted files, and match them against query terms. Commonly used search structures are hash files for exact matches, B+ trees, tries and suffix automata. [1]

Figure 2.1 shows an example taken from Candan et. al's book, Data management for information retrieval [1]. The figure shows an example search query using an inverted files structure.

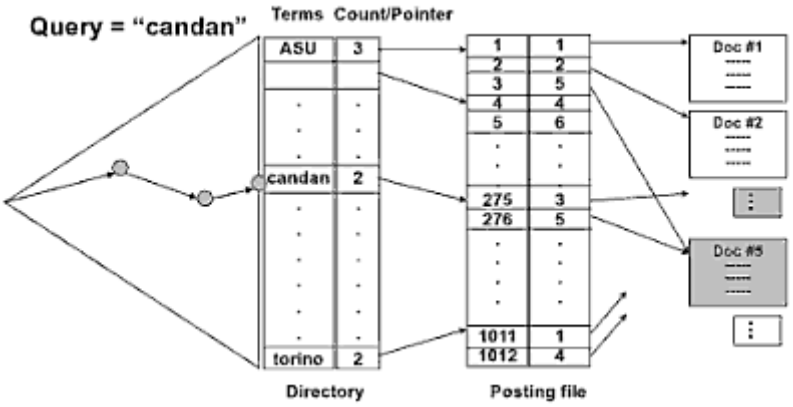


Figure 2-1: Inverted files structure, search example from Candan et. al. [1]

2.3. Metadata and auto-annotation

The relaxed definition of metadata is *data about data*. In image retrieval, image metadata can be tags, keywords or descriptions of relevance for the image. This can be data that has been added by the capturing device, i.e. date/time and GPS coordinate, keywords manually added by a person to describe e.g. contextual information related to the image or *automatic Image annotations* collected by the Image retrieval system to improve search and indexing. The latter is usually referred to as *auto-annotation* or *linguistic indexing* [4].

The most widely used standard for storing metadata for images today is Exchangeable image file format (EXIF). I will describe EXIF briefly in the next sub section (2.3.1).

2.3.1. EXchangeable Image Format (EXIF)

The EXIF tag structure is similar to that of TIFF files; it uses the existing JPEG file format for compressed files and the TIFF rev 6.0 format for uncompressed files, with the addition of metadata tags. In JPEG files the EXIF data is stored in one of JPEG's defined application marker segments, defined APP1. Standard Metadata tags defined in EXIF include, date and time information, camera information i.e. camera model and producer, image specific information such as aperture, focal length, ISO speed etc., image thumbnail information for previewing the picture and descriptions and copyright information. Also the EXIF holds standard tags for Geographical information, i.e. GPS positions, altitude, image direction and so on. [26]

2.3.2. Context

Recent trends in image retrieval have seen the importance of also including contextual information as metadata for the images. Context is in many ways a complex and difficult term. I will discuss context more thoroughly throughout this sub section to provide a more solid understanding of it.

2.3.2.1. Definition and history

It can be difficult to agree to a global definition of context in computing, but one wide-ranging definition provided by Dey [10] is:

> *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves."*

Specific to image retrieval the entity is usually the capturing device, e.g. a camera or mobile phone.

Context awareness all started with and is one of the main subjects and driven factors of ubiquitous computing (UbiComp) [27]. UbiComp was presented by Weiser in 1991 [28]. Weiser's vision for UbiComp was computers being part of everyday objects, everywhere. E.g. a coffee machine automatically producing coffee as you go to the bathroom in the morning, lights and television that automatically turns on when you enter the living room and turns of when no one is present in the room, i.e. objects being part of their surroundings. Only the imagination sets the limitations of Weiser's vision. To make it possible for the everyday objects to be context aware they have to be aware of its surroundings, they may have to communicate among each other and share sensed information with other objects [27].

UbiComp has been an important factor in the development of the concept of context aware systems, also known as *context aware pervasive systems*. According to Loke [29] the main functionalities of such a system are often divided into three; *sensing*, *thinking* and *acting*. In the subsequent three sub sections I will present these functionalities briefly.

2.3.2.2. Sensing

Sensing can be achieved by providing sensors that can help us acquire information about the physical world that again can help us determine physical actions most appropriate for the individual situations. Sensory information can be collected using sensors that measure temperature, pressure, light, motion and so on. Loke remarks that there are also other devices that can be regarded as sensors, i.e. microphones, computer clocks, radio frequency identifications (RFID), devices that measure movement and acceleration, location and position (e.g. global position system (GPS)), magnetic field and orientation, proximity etc. Sensory information can also be combined to produce even more important observations.

Among these, date/time and location are the most used context types in current research approaches in image retrieval, and can be used for organizing images into collections, for answering time or location related image requests, and for inferring additional image metadata. Global position system (GPS) used in cameras and mobiles of today use satellites with assistance from ground stations and gives accuracy within meters of the physical location [29].

2.3.2.3. Thinking

It is hard if not impossible to reason about the surrounding environment without having any knowledge about it. Context aware systems gather knowledge about the environment, metaphorically speaking, in the same way as humans, by sensing, as discussed in the previous sub section.

Once data about the environment has been collected through sensors, the next step conferring to Loke is to make sense of it. The system must utilize the sensory information in a way most relevant for the use of the system. Situations must be recognized and context information must be prepared using the sensed information. This is the thinking stage of context aware systems. Much work has been done to utilize sensory data and to obtain contexts in the real world, e.g. to estimate physical quantities such as distances [30], observe patterns in human acting in the real world [31] and building augmented worlds [32].

2.3.2.4. Acting

Once context information has been gathered or situations have been recognized, actions can be taken. Actions to be taken are specific for each application depending on what the system wants to achieve in taking the specific actions, e.g. an action can be as easy as collecting more information about the environment through sensing [29].

2.3.3. Collaborated tagging

Also textual information connected with the image can be metadata such as manually added tags added specifically for an image. Manually annotating images with tags are often referred to as annotation based image retrieval (ABIR) within the field of image retrieval [18]. Tags are usually in the form of keywords added by individual users to describe their perception of the image. To cope with the large amount of images online, and the challenges that comes with this, some internet services has made it possible for users to share their images and manually tag them, to make the task of image retrieval easier. E.g. Flickr[5] lets users upload images and tag them with information such as personal tags, location, time, and other context related tags. When many users collaborate on adding metadata to images or other content in the form keywords it is often referred to as *collaborative tagging*, and has recently grown exponentially in popularity on the web [14].

[5] http://flickr.com

Subjectivity:
Still manually added tags are *subjective*, which means that individual users have individual perceptions of how to best tag certain images with certain tags at certain times [13]. *Subjectivity* is the biggest concern when dealing with metadata that represent individual's perception. Nevertheless, previous work has been presented for how people manage their images for manual tagging, with focus on building systems that group images into categories that corresponds to the natural way people think about their images. [33, 34]

2.3.4. Auto- annotation

Automatic Image annotations or *auto- annotation* are often used within text based image retrieval systems to augment the images with additional metadata [4]. When more textual and contextual information are available the bigger are the potential for getting more successful retrieval, i.e. of course given that the metadata are reliable and relevant for the images.

Already available metadata can be used alone, or combined to supplement the images with important contextual metadata. For example in MediaAssist [7] a combination of locational metadata added by the capturing device (i.e. latitude and longitude) and date/time stamp are used to augment images with atmospheric conditions, e.g. temperature and weather condition provided from local weather stations. Also MediAassist use the locational data to augment images with metadata such as name of country, city and street. Cheng et al. [9] use a combination of GPS coordinates and orientation of the capturing device to decide whether subjects in certain images are of specific Points Of Interest, e.g. famous attractions and the like.

2.3.5. Semantic gap

Through *metadata* and *auto- annotations* image search can be facilitated through keyword search, and can make the image search semantically more meaningful. That is presumed that the metadata is reliable and that enough contextual information about the image is gathered. When gathering additional metadata for the image through *auto- annotations* it is hard to exclude problems concerning semantics. One such concern is the *semantic gap* problem. The semantic gap problem refers to the difference between two descriptions of an object or feature, where one is abstract and one is more specific and requires external knowledge. [1] The semantic gap problem usually arises whenever ordinary human activities, observations, and tasks are transferred into a computational representation. This refers to the vast differences between the perception capacity of humans and computers. In closing the semantic gap much work has been done, e.g. search engines and online shops has shown big success using *user relevance feedback*, looking at user patterns [35] and using CBIR techniques.

Recent trends in information retrieval and auto annotation has shown great progress in closing the semantic gap. Research has involved auto-annotating images with contextual location and time specific metadata. It

has been shown that a combination of metadata may be combined to accumulate semantic understanding, e.g. a combination of time and location can sometimes propose the semantic content of images. [11, 12]

Also in Information theory, many tools and methods are available for achieving more semantic retrieval, such as context accumulating algorithms. E.g. Singular value decomposition (SVD) identify patterns between terms and concepts contained in an unstructured collection of text and Latent semantic Analysis/Indexing (LSA/ LSI) use SVD to extract contextual content of text, analysing it and finding its semantic meaning. [1]

2.3.6. Ontologies

Semantics can be captured by adding a higher level of descriptions to the metadata or features of the image. E.g. semantics may concern specific words, such that a term or word can have many different meanings, e.g. the word *tiger* can have different meanings depending on in what context it is used, i.e. the animal *tiger*, the golf player *Tiger Woods* or the airplane *"de Havilland DH 82* ***Tiger*** *Moth"*. We can capture the semantics of certain words by mapping the words to ontologies. An ontology is a structured representation of the knowledge within some area which holds a higher level description of the terms and captures the semantic meaning of the specific word [5]. However, the use of ontologies usually involves a excessive amount of manual intervention [36]. E.g. in the situation of manually annotating images with tags, the tags would usually be mapped to ontologies at the time they are inserted. If not, e.g. if an image retrieval system automatically add semantics to annotated tags through ontologies, the system cannot be certain terms are mapped to its correct semantic meaning. This is of course in situations where terms have several different meanings, e.g. as for the term *Tiger*.

2.4. Database theory

When dealing with any large amounts of data, knowledge can more easily be discovered by structuring it into a database. [37]

2.4.1. Entity- relation approach

When designing a database one usually start by making an entity relation (ER) model. The E-R model is a conceptual representation of the entities, relations and constraints that make up a given design and provides a graphical summary of the database structure. The approach is very useful for the designer, gives a visual representation that supports how humans usually think and works as a good tool for validating the correctness of the design. [38] The entities model the objects that can be uniquely identified, e.g. in a student registration system, *student* and *course* would be entities. Relationships model the connection between entities, e.g. student *follows* course, where *follows* is the relation. *Constraints* are also an important aspect of the ER-design as in a relational model, e.g. a student can only follow one course at a given time on a given day.

2.4.2. From Conceptual model to relational database schemas

When a satisfactory ER- model has been produced, the next step is to convert it into a relational schema and ultimately into SQL create statements. According to Kifer et al. [38] there are a few basic step that can be taken to produce the database schema. I will summarize them here:

For entities the following transformations are performed:

1. Each strong entity, i.e. an entity that is not dependent on others and can exist on its own e.g. the *student* entity exampled above, can be transformed with the following steps:
   a) Each entity will become a relation.
   b) Each attribute of the entity will become an attribute of the relation.
   c) One of the key attributes of the entity will become a primary key for the table.
2. A weak entity is an entity that is dependent on one or more other entities to exist, e.g. a *child* entity is weak entity since it is dependent of its parent to exist. For each weak entities the following are performed:
   a) Create a table.
   b) All attributes of entity is included as attributes of table.
   c) Include all primary keys of *parents* as foreign keys in table.
   d) The primary key of table is a combination of the *parents'* primary keys.

For Relationships the following are performed

1. An M:N relation are in many ways similar to a weak entity, i.e. a table has to be created which is dependent on one or more *parents*. Each M:N relation are transformed with the following steps :
   a) For each relationship type a table will be created
   b) The primary keys from all entities participating in the relation will be included as foreign key attributes of the table.
   c) Also all attributes of the relation will become an attribute of the table.
   d) The combination of the foreign keys from b. and, if any, key attributes from the relation will form the primary key of the table.
2. Each 1:N (one to many) relation will be translated with the following criteria:
   a) The table of each N-side of the relation will be included with the 1-side's primary key as a foreign key attribute.
   b) Also all entity attributes on both side are included in the entities table, this has hopefully already been done in step 1.b.

Also if there are multivalued attributes in any of the entities or relations these has to be transformed as well. I.e. for each multivalued attribute A, the translation is:

a) Create a table

b) The table will include an attribute corresponding to A,
c) Also a foreign key attribute is included, which references the table representing the entity/ relationship type that has A as an attribute
d) Primary key is combination of all its attributes

2.4.3. Normalization theory

The ER approach does not guarantee good relational design alone. Normalization is used to help database designers to evaluate the relation schema so that it is in a certain *normal form* by looking at problems concerning redundancy. According to Kifer et. al redundancy may concern updates, insertions and deletions anomalies in a database. Decompositions are used to deal with such problem. An example of a decomposition is in situation where multivalued are present in the ER- diagram as described in the end of the previous section. To get a more realistic view we can e.g. decompose the table;

Person(SSN, Name, Address, Hobbies)

, which have the multivalued attribute hobbies into two tables:

Person1(SSN, Name, Address)

Hobby(SSN, Address)

Conferring to Kifer et. al. the central tool for decompositions are functional dependencies, which generalizes the idea of key constraints. Further functional dependencies are used to define *normal forms*, which is a set of requirements desired in update intensive systems. This is why the theory of decomposition also often is called *normalization theory*. The normal forms are used to eliminate redundancy and potential update anomalies. Several normal forms are identified within database theory, where each normal form is characterized by a set of restrictions. Thus, if a database schema follows these restrictions it has certain predictable properties. Originally three normal forms namely 1NF, 2NF and 3NF where introduced by Codd [39], where each success imposes more and more restrictions. Later on the Boyce-Codd normal form (BCNF) where introduced to get rid of some weaknesses found in 3NF. I will briefly present Codd's original normal forms along with the BCNF.

The first normal form (1NF) defines that an attribute must be atomic, it must not be anything that has structure, e.g. a record cannot have multiple fields or a set of fields. The second normal form (2NF) defined that a schema must not have any functional dependency, X → Y, where X is a subset of that schema's key and Y has attributes that does not follow in any of the schema's keys. An example of the table that does not follow 2NF is the table **person** pictured above, splitting **person** into two, i.e. **person1** and **hobby** holds for the restrictions defined by the second normal form. The third normal form (3NF) holds if it for every functional dependency X → Y ϵ (set of functional dependencies) the following conditions are true:

1. $Y \subseteq X$ (i.e. this is a trivial functional dependency)

2. X is a superkey of the relational schema
3. Each attribute in A ϵ Y–X belongs to a candidate key of the relational schema

BCNF is a relaxation of 3NF where only criteria 1 and 2 from 3NF defined above need to be true.

Chapter 3

3. Related work

In this chapter I will present previous work that is related to my project. The main characteristics for my solution are contextual information connected to the image and summarization of large image collections. Therefore, I have decided to separate relevant work into these two characteristics. In section 3.1 I will present previous work for context information in image retrieval and relate it to my approach. In section 3.2 I will present previous work that are similar to my approach of summarizing image collections for efficient retrieval.

3.1. Context in image retrieval

Context is extremely wide ranging, but in this project I will mainly focus on two contextual properties as a basis, i.e. location (of subject or capturing device) and data/time (of when an image is taken). Hence, in this section work that use location, time/date and auto- annotations as a baseline are presented.

The Mobile Media Metadata system (MMM) presented by Davis et. al. [6], gather media content based on the temporal, spatial and social context at the time the image is taken. The system is developed for mobile phones, and enables user verification of the most important media content gathered for the image. The system is different from mine as the content is collected analysing statistical patterns of prior annotations made for images that have similar contextual data, e.g. similar time and location. My system does automatically annotate images, but not with manually added tags from other collections of images. Notably, my system also takes advantage of manually added tags, but only focus on those already present for the images used. Further the information is gathered from a database consisting of prior taken images. Davies et. al make it clear that MMM uses locational information for the subject in the image rather than the location of the camera taking the image. An improved version of the system (MMM2) is presented in [15]. MMM is concerned of images that are taken far away from the subject in the image, which is different of my system. My developed system sees the capturing device as the entity and not a potential subject in the image. While MMM's approach are very relevant for certain images I believe the fraction of images taken of subjects from far distances are globally very small. Since my system are made for a global domain, the majority of the images, which is assumed having the camera location close to the subject, has to be prioritized to make the system as less noisy as possible. Hence in my system the majority wins, location of the camera is kept and subject location is not.

Naaman et. al [40] presents PhotoCompas which use time and location metadata from private collections of images to organize photos into time based and location based collections of data. The time and location metadata is also used to create contextual information for the image, by gathering additional metadata using a combination of time and location. Similar to my

system PhotoCompas use location and time metadata with the intention of gathering additional metadata both through analysis and from online web-services. The difference between the two is that Naaman et. al limits the solution to personal use and not on online collections such as different domain specific image collections.

Toyama et. al [41] presents an end-to-end system that lets users manually tag their images with geographical location data and share these images with other users across the world. The system has a visual interface, including a map, where locations are selected by the user. The users can then look at other images taken at the same location as his, or simply look at images uploaded and tagged with locations of interest. All images are indexed by location in The World Wide Media eXchange (WWMX) database. The system only handles manual contextual information to be applied and does not support collection of any additional contextual or content specific metadata. My system assume that location specific metadata is collected and included in advance by the capturing device and have totally different intentions in how to use the location data further. When that has been mentioned Toyama et. al presents a system that involves manually added tags. Even though the solution of my system is developed to be fully automated, manually added tags are used, if available, based on capturing the natural perception of humans. This could be argued to be similar to Toyama et. al intentions for their solution.

The MediaAssist project presented by O'Hare et. al [7] enables organization and searching personal archives of images based on contextual and content information [42]. Some of the contextual information that is used is time and location. These contextual metadata are used to gather additional metadata, which is similar as in my approach. Also the images are analysed using different CBIR techniques, e.g. for face recognition, light status (e.g. day, night) and so on. My system strictly does not use any visual content analysis, but focus on text based image retrieval. MediaAssist include semi-automated annotation techniques which e.g. let's people add names to unrecognized faces or correct automatic-annotations if they are incorrect, which is a distinctive difference to my system. The location specific metadata is used in a similar manner as in my system. The difference is that in my system locational information (i.e. latitude and longitude) is assumed to be an automatic process performed by the capturing device, which is exploited to augment images with locational data. When it comes to the GPS metadata, in MediaAssist this process is done manually and not assumed being an automatic process by the capturing device. Another similarity between mine and MediaAssist is that date and time stamps is converted into keyword tags at different levels, e.g. month, day and season so that images can be filtered out through keyword searches such as " return all images taken in the summer on Saturday". MediaAssist also gather weather conditions for the images using local weather stations. The features mentioned above bear many similarities to my system, when it comes to annotating, a difference between the two is that MediaAssist use a hybrid approach, other than using ABIR and text based techniques the system also annotate pictures using CBIR techniques and mine does not.

Jeon et. al [8] suggest a system that uses different CBIR techniques to automatically annotate images based on a training set of images. The approach use probabilistic models to decide similarities between features in the image and features in images from the training set. Each image includes a set of blobs and a set of words included in the caption of the image that describe the blobs. The concept of a blob is actually a segment of the image, either objects (i.e. humans, buildings, boats etc.) or regions such as water, grass and sky. In the process of auto-annotating images, a set of images are analysed, each image is segmented into blobs and clustered together according to different similarity measures, e.g. shape, texture, colour. Then the system analyses the clusters of segments (blobs) and use relevance models to automatically annotate the images containing the blobs with words using the training set. E.g. if an image from the training set have similar blobs as the image being analysed the semantic descriptions are copied. This approach analyses the physical image itself, which is assumed to be unlabelled before the process of auto-annotating the images is started. The difference between this approach and mine is that, even if this approach can add relevant tags to the images, a small amount of contextual information about the image is collected. It is also worth mentioning as described in *section 2.1* , using such CBIR techniques bring no assurance that the regions are described correctly.

The CBSA system presented by Chang et. al [43] uses a semi- automated approach to annotate unlabelled images with *soft labels*. As for Jeon et. als system the CBSA also use a training set of manually labelled images, consisting of 25000 images divided into 116 categories, to train an ensemble using the statistical machine learning method Bayes Point Machine (BPM). When training the ensemble the system uses a One Per Class (OPC) scheme, which trains binary classifiers for classes. The classifier with the largest output determines the class of each data point. The trained BPM-OPC ensemble is applied to each image to predict label membership. Each image is in this way described with a bag of words, where each word's relevance is indicated using a probability value. Chang et. al.'s approach is also different from mine as they practice content based techniques in extracting features from the images, use a test set to aggregate new metadata along with involving users through user relevance feedback, hence changing the initial labelling through manual intervention. The latter makes the process of automatically gathering additional metadata semi- automated and not automated as in my approach. The CBSA system is not mentioned using any location, time and directional information in their labelling, but since this system is content based and are meant for unlabelled images I assume this aspect of the system also is different from mine.

Feng et. al [44] presents the Multiple-Bernoulli Relevance Model (MBRM). As for the CBSA system and Jeon et. al's this system also practice auto-annotation using a training set of images and CBIR techniques. The MBRM model uses a generative approach to deciding annotations. Usually the highest ranked words are included as an annotator for the image, and when annotated the system focus on the presence or absence of the words rather than there probabilistic importance. Feng et. als system is also different

from mine as annotation is done using a training set, they practice CBIR techniques and probabilistic models.

As mentioned in *section 2.1* image retrieval systems are usually text based (concept) or content based, All systems described in this section involves using content based techniques, i.e. they are focused on extracting features such as shape, colour and texture out of the image and using similarity measured, either probabilistic or machine learning techniques, in describing them with words. To prevent confusion some of the systems described so far involve using a combination of text based and content based techniques. Anyhow Cheng et. al [9] suggest an annotation technique that is in many aspects very similar to the one used in my project. The authors propose using different mobile sensory data such as GPS location and directional data (compass) to gather additional metadata for the image. The approach suggests combining GPS coordinates and directional compass data to correct the prediction of the location of the image. The authors are as in the MMM system, presented earlier in this section, focused on using the location of the subject in the image, e.g. a landmark, instead of focusing on the location of the camera as my approach does. The system download geo-tagged images from online image sharing websites and in an offline style mine important information about them. The images are clustered based on content similarities and location with intention to create clusters that include frequent tags for a potential Point Of Interest (POI) in the area. The closer an image added to the system are to a POI, while at the same time having a direction that points at it, the higher the likelihood is that a certain POI is the subject of the image. My system does not use any POI's, but potentially use online image collections. Anyhow these are only used as candidates for analysis and not for augmenting external images with tags. Cheng et. al use a combination of the cameras sensory data and CBIR techniques to improve assumptions of the image, as well as auto-annotating it with data gathered from online image collections. Although my system is not interested in semantically locating certain POI's using CBIR techniques the use of context data in Cheng et. als approach are very similar.

3.2. Summarizing Image collections

As discussed in section 2.1.2.1 summarizing large collections of documents has been addressed in the field of Information theory. Anyhow the same approaches are difficult to transfer and have not been successfully adopted by image retrieval systems. Roughly speaking the success in information theory has to deal with the amounts of textual data that are available in text documents, advanced algorithms for extracting semantic meanings from the data and the small differences in individual people's perception of a document in relation to search queries. Anyhow the recent trends in image sharing, tagging of images and metadata automatically added by the capturing devices have opened up new possibilities. Also converting metadata and augmenting images with additional metadata can help us understand the semantic meaning of collections of images to a greater extent. Having more textual data for the collections of images can help in describing large collections of images through their main characteristics,

and can be used to easily filter out large collections that are not relevant for search queries.

Kennedy et. al [45] analyse community contributed large scale image collections. The researchers use a collection of 20 000 000 images from Flickr which include metadata such as location, time and manually added keywords. The images are improved with precision for landmark and location based queries and summaries for collections of images are made by selection. The system analyse all images within a certain area, by assuming that within this area important images will be frequently tagged by photographers with important objects, events and landmarks. Following this approach, representative tags for each area is extracted, the most frequently used tags are reconsidered depending on how many photographers that have used it, simply to filter out as many deviants as possible. In this way, having e.g. a town say San Francisco, the images contained within the most important areas of San Francisco are clustered together. For each important area only the most representative tags are retained, and holds as a summarized description for the area. Semantic data are also extracted in the form of events and location, e.g. if a user searches for "golden gate bridge" the system knows that the search refers to a landmark in san Francisco instead of just am image holding the three tags, "golden", "gate" and "bridge", which could refer to something completely different. Also the representative cluster for an area are improved by using CBIR techniques, locating visual features in the image, and finding how representative the image are for certain descriptions or keywords, e.g. discarding similar images or duplicates, removing landmark images with families in front of the landmark and removing false positives. Kennedy et. al.'s. system is similar to mine as for a given set of images only the representative properties are retained , the difference is that in my system the metadata is separated for an existing image collection being analysed and it does not group the images being analysed into locational regions.

Simon et. al [46] use techniques for summarizing geographical areas and scenes by extracting the visual splendours from an certain area using large community based image collections. Images that are tagged with locational and other context relevant information are used to derive a one page summary which describes the key interests for certain scenes and locations, such as a city, a landmark etc. The images in such community based image collections are often in the size of millions and usually only contain a few tags each. This can lead to results for user queries in sizes that are extremely large. The solution that the authors suggest is a system that extracts images from several sites, focusing on a certain geographical area, analysing the set of images using CBIR techniques and finding what seem to be the most relevant scenes for the area. When the most relevant scenes are found, the images are clustered together according to these scenes. The most relevant images for each scene is located from the different sites and the most highly weighted images are kept as representatives or summaries for the certain scene. Again the scenes serves as summaries for the geographical area, included with representative images for each scene. The system developed serves as an interactive 3D browser which allows users to navigate huge community based collections of images with greater efficiently, both in time

and relevance. As for Kennedy et. al.'s approach Simon et. al. also focus on clustering images according to locational regions, which as discussed previously is different from mine. Also Simon et. al. practice CBIR techniques which also is a vital difference between the two.

As for the two systems described above, also Kennedy & Naaman [45] analyse community based collections of images and use Flickr as test data. The researchers use a combination of context and content based tools for gathering location and landmark representative sets of images. The approach is very similar to the work of Simon et. al [46], the difference is that they start by finding tags that represents landmarks, and generate views that represent the landmark, and not just a *scene* as in Simon et. als approach. Kennedy & Naaman only use tags and metadata when creating the views themselves, and only use CBIR techniques to further strengthen or weakening the assumptions of the views. Kennedy & Naaman proves that this approach speeds up the process of the management of the system compared to Simon et al's. approach, which focus on building clusters and summaries almost entirely on the visual content of the images. Finally, when the views has been analysed and improved by visual content analysis a summary for the view is made describing the main characteristics of the images in it. This approach is very similar as the previous to systems presented, and the difference in comparison with mine is obvious. The similarity is that the system focuses on locating representatives and reducing the retained amount of metadata for a given set of images.

Chapter 4

4. Approach and design

4.1. Terminology

Before I present the design and approach of the developed solution of this thesis I will present some common terminology that is used throughout this chapter.

Term – A *term* or *metadata term* is a keyword which is part of one of the images, usually in the EXIF header. A term can be of different kinds, e.g. *user provided terms* (tags and title terms), *date/time specific terms*, *locational terms*, *weather specific terms* etc. All *terms* with exception of *user provided terms* are terms that are either augmented to the images from external sources or converted from numeric metadata (e.g. date/time)

Hypernyms – user provided terms are grouped into a higher level representation of the terms. E.g. the higher level representation of the term *car* is *motor vehicle.* The ontology used in this system refers to such higher level representations as a *Hypernym*. Therefore, I will also use this definition.

Selection threshold – The weighting threshold used to decide whether or not a metadata term is representative for the collection.

Term frequency – The weighting value calculated for all metadata terms. If a term has a term frequency above the *selection threshold* the term is viewed as representative for the collection.

Inverse document frequency – The value used to weight a term in relation with all image collections available.

4.2. Overview

In this project I will process image collections with the goal of locating and extracting representative *terms*. Representatives are terms that describe the image collection as a whole from its most informative view. Further a representative is a term that should stand out from other terms in the collection. This means that in extreme scenarios where no terms or features of an image collection are prominent from the rest, no terms are viewed as representative for the collection.

Representatives are found by analysing textual metadata of the images located within an image collection and locating dominant terms that stands out from other. Textual metadata can be manually added terms in the form of tags, terms extracted from the title of the image or contextual information gathered by augmenting the images with additional metadata terms. In this project the latter involves supplementing images with additional metadata

using locational information (i.e. latitude and longitude) and date/time. That is of course if this information is available for the images.

When images within the collection have been augmented with additional metadata all terms will be cleansed and *user provided terms* will be grouped together into higher level representations (Hypernyms, definition in section 4.1). The hypernyms represents a higher level representation of terms connected with the images (e.g. the terms *car*, *SUV*, *motorcycle*, have a higher level representation *motor vehicle* in common, the higher level representation of *motor vehicle* is *self-propelled vehicle* and so on). Hypernyms are used to strengthen terms through its hypernym members, and used to find a hierarchical threshold which reflects the terms of the collection in the best manner as possible. Finally, when data has been cleansed and strengthened the resulting metadata terms are analysed and representatives are located before a collections summary is created.

4.3. System Architecture

On a high level, the architecture of the system can be divided into three main mechanisms, one that augment the images with additional metadata terms, the second which cleanse, strengthens and calculate term frequencies for the all metadata terms and the third which analyse the final metadata terms and locates the representatives for the image collection summary. The mechanisms are pictured in *figure 4.1*.

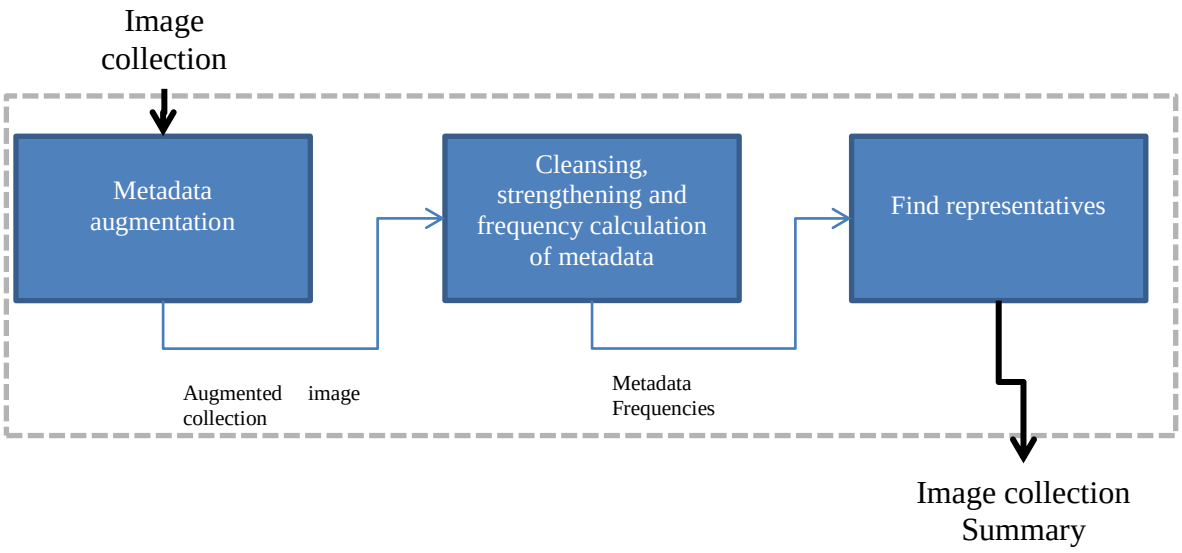


Figure 4-1: High level system architecture

From the high level architecture, the system takes an image collection as input and transforms the metadata within it to a collection summary that describes the collection through its representative terms, if there is any. Within the main components of the system there are several sub components or processes which help in producing the different outputs of the main components. A detailed description of the system is pictured in *Figure 4.2*.

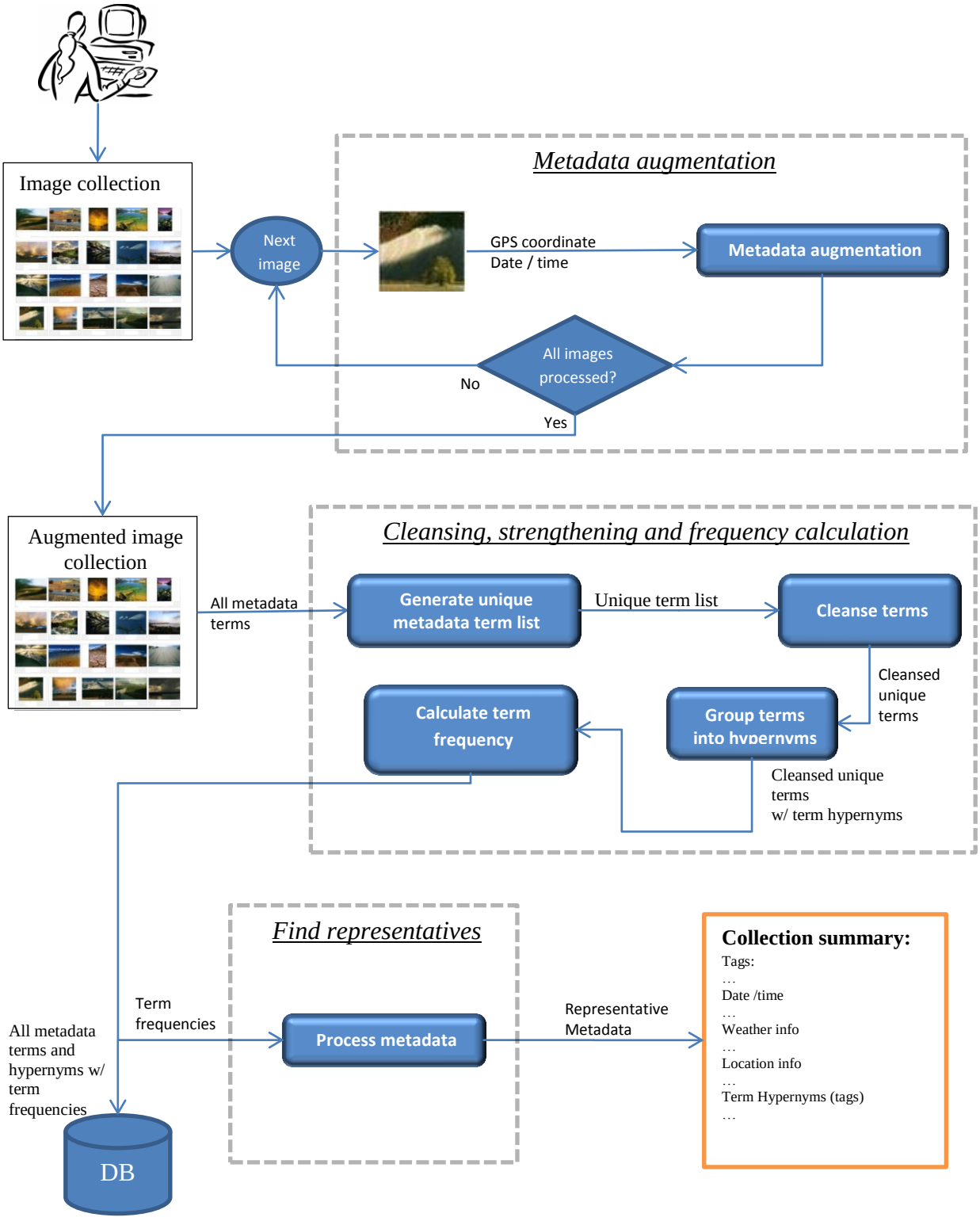


Figure 4-2: Detailed System architecture

From the detailed representation of the system the main components pictured in *Figure 4-1* are represented by the grey boxes, while the sub components within them are represented by the blue component boxes. Image collections come as input and a collection summary as output. The

user requesting a specific image collection is also pictured, with purpose to make the representation of the workflow described below more complete.

The different components of the system are described through the rest of this chapter. Metadata augmentation are described in section 4.6, general cleansing and strengthening of metadata terms are described in section 4.7, selection and calculation of term frequencies are described in section 4.8.2 and 4.8.3, while production of the representatives for the collection is described in section 4.8.4.

To get a more realistic view of the architecture I will present the workflow of the system in the scenario of producing a collection summary for a given image collection. The workflow gives us a top down approach to which processes the metadata goes through before the representative terms are listed in the final collection summary. Notably the workflow depictured is tailored an image collection gathered from Flickr.

1. The user of the system selects a collection to be gathered from Flickr from a predefined list of geo referenced image collections.
2. The system requests Flickr to return the given image collection, specifying that only geo-tagged images are returned and that each image should come with its title, tags and GPS coordinate metadata.
3. Each image is augmented with additional metadata using the image's GPS coordinates and date/time stamp. Additional contextual metadata gathered from external sources are; Locational data (i.e. name of country, region, city, street) and atmospheric conditions (e.g. temperature, weather condition, wind strength etc.). Also date /time stamps are converted into month (i.e. January, February etc.), weekday (i.e. Monday, Tuesday etc.), season (i.e. summer, winter etc.) and a combination of year month (e.g. 2009 January, 2008 Mars etc.).
4. When all images have been augmented with additional available metadata a unique list of metadata terms is created, i.e. no duplicates of terms are expressed in the unique list.
5. The unique metadata terms are then cleansed for whitespaces, character symbols, stop words, non-roman characters, conjunctions, prepositions, determiners, non-informative words and oversized tags.
6. Also term hypernyms is located, grouping words of similar meaning together, the terms used are those located in the *tags* and *title* metadata field of the image.
7. For all cleansed data, including the term hypernyms, term frequencies are calculated for each unique metadata term.
8. All term frequencies are compared against their selection thresholds to locate the representative terms of the collection.
9. The representatives from the different metadata (i.e. user provided terms, date/time terms, weather terms, locational terms, term hypernyms) are used to create the collection summary. The collection summary describes the image collection as a whole through its most representative terms.
10. Data gathered through the analysis is stored and structured in a database (described further in chapter 5).

Before digging deeper into the description of the architecture I will in the next section define the requirement specification for an image collection used as input by the system. In section 4.5 I will describe the specific image collections that I will use in the testing of the system implementation.

4.4. Definition and requirement specification

From the perspective of this project a collection can be viewed as pre-existing in difference ways. From one perspective a collection of images may be existing on a web page, which prioritize and provide images on a certain field or area (e.g. image collection of cars, planes, towers, boats), theme or event (e.g. vacation, camp, study trip, boat trip etc.), location specific images such as pictures of Italy, France and so on. Another scenario is providing collections of images hierarchically structured in folders on a personal computer.

In the first phase of this project the goal was to find a set of image collections where the following two requirements are met.

Desired specification list:

1. *Part of existing image collection:* The images that I will work with have to be existing as a collection correspondingly to one of the forms described above
2. Metadata: *One* or *more* metadata fields have to be present for an image. The different kinds of metadata desired are:
    a. *Tags:* Tags field consists of terms of information related to the image manually inserted by individual users
    b. *Title:* A manually inserted string of words, usually describing the picture through location, time and event.
    c. *Coordinates:* Locational data, i.e. latitude and longitude usually inserted by the capturing device. Coordinates and timestamp can be used to augment the image with additional metadata through auto- annotations
    d. *Date/time:* date and time stamp of when the Image is taken. All modern cameras provide this information at the time of capture.

As can be seen from the requirements list, I have decided to not include all metadata properties as absolutely necessary for the system to function. One vital property is that all images have to be part of an *existing image collection* of some kind. Other than that it is also necessary that at least one of the four metadata properties, *tags*, *title*, *locational data* and *date/time* stamp are included or part of the image. This means that three of the four may be absent as a maximum, but not all four.

These design choices has been made to make the system as tolerant and close to the reality of current and futuristic image collections as possible. Even if including tags, title and coordinates for an image are trends that are

more and more being used today on online *collaborative tagging sites*, most probably there will always be images with locational, title, tags and date/time metadata missing, or at least one of them absent. At least in private image collections I believe that this is a concern, where people simply do not bother or find time to manually tag their images.

Nevertheless, when only tags are available these can have great usability for the image retrieval systems, where locational data are mainly used to augment the images with additional metadata. Even if the latter are used to increase the semantic understanding of images and helps give better descriptions of the images described, providing *tags* and/or *title* can also be very helpful alone and vice versa. Date/time is converted into time specific terms which also increase the semantic understanding of images. Hence requiring only one of the four specifications to be present is sufficient for good image retrieval in my opinion. At least if one of them is missing in only parts of the image collection.

For example, the system can tolerate some of the images to have images with all metadata absent, but if the majority of the image follows the desired specification the collection is viewed as proper for further analysis. The threshold is set to 90 per cent, which means that at least 90 per cent of the images within an analysed collection have to have one of the desired metadata fields present for the collection to be accepted by the system.

In the next section I will present the image collections that I will use in the development of my solution, before presenting how these image collections are augmented with additional metadata to increase the textual information connected with it to help create more accurate and semantically more meaningful summaries. Further, I will explain how data are cleansed and improved before finally representatives for the collections is gathered to create summaries that describe the image collection from its most informative view.

4.5. Gathering image collections from Flickr

In the first phase of this project the goal was to find a source for images that where both manually annotated with tags and could be identified as a collection. The most obvious approach and my first suggestion were to manually structure a set of image collections. Even if not part of the requirement specification (section 4.4), to see the full potential of the developed system all images are also desired to contain date/time and GPS coordinate in the evaluation. Test users are assigned to manually augment the images with additional metadata in the form of user provided terms. I found out that this approach would first of all be very time consuming, but also it would be difficult to get the desired amount of data to work with.

An alternative had to be found and I started browsing for test collections of images, and found quite a few, but they were either too domain specific, did not contain tags, title, date/time or GPS coordinates and did not fit my desired specifications as described in section 4.4. The test collections available for download are mostly provided to use within the field of Content Based Image Retrieval, i.e. meant for extracting features such as

shape, texture and colours from the physical image itself through analysis (section 2.1). Therefore providing user provided terms connected with such image collections are usually not viewed as necessary.

The next suggestion was using image collections found on community contributed online image sharing sites. Such community based sharing sites have been more and more widespread during the last years. I found several sites available for this specific purpose, including Google's PicasaWeb[6], Flickr[7], smugmug[8], locr[9] and everytrail[10].

I found Flickr to be the most suitable for my experiment. Flickr has over 5 billion images, many of which are geo tagged, which is good to evaluate the full potential of my system. Also Flickr has made it available for their users to create groups. A group is a place where different users can contribute with images by following certain criteria's from the group description. E.g. groups are usually meant for images from a specific location, year, relevant for different themes etc. There are also a huge amount of groups which are intended only for geo-tagged images, which is perfect for my experiment. I have 17 groups to be used in the development of my system. Each of the chosen groups consists of 300 to 7500 geo-tagged images, and is individually viewed as an existing collection of images.

Flickr has a very functionality rich API[11] available which allows users to search for all images on their servers, specifying a great amount of criteria's. The resulting queries are returned as an *element tree* where each *element* represents an image. An *element* holds the most basic information for each image, such as owner, date taken, which server it is located on, title, tags etc.

It is also possible to specify additional metadata that you want to be returned for each image or *element*, if available. I have specified for the groups I have chosen, I only want images that are geo-tagged. I want the tags field, date, and the coordinates for the images to be returned, all specified to see the full potential of the developed system. Each returned element tree is stored on disk and then ready to be augmented with additional metadata, cleansed and analysed before a summarized description of the most representative terms of the collection is created. These steps will be described throughout the sections of this chapter.

4.6. Collecting contextual information through auto-annotations

IR is focused on finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers) [25]. A distinguishing reason for

[6] http://picasaweb.google.com
[7] http://www.flickr.com/
[8] http://www.smugmug.com/
[9] http://www.locr.com/
[10] http://www.everytrail.com/
[11] http://www.flickr.com/services/api/

the success in IR is that the material are textual which are of conceptual nature and easier to analyse and semantically understand in relation with user queries.

Even though we have efficient techniques for reduction and description of data in information theory, it is difficult to transfer these techniques to image retrieval. One reason for this is because we usually do not have enough descriptive and continuous textual information connected with an image. Without enough textual information it is difficult to capture the semantic meaning or distinguished features through analysis as in information theory, so other complementary approaches have to be used.

The developed solution in this thesis will therefore use available metadata, such as Global position system (GPS) location (of the capturing device) and date/time (of when the image are taken) to augment the images with additional contextual metadata. This can enhance our knowledge of the images and may thus contribute to closing the *semantic gap* and improving effectiveness of image retrieval.

Notably the system developed in this project is meant for different kinds of collections, i.e. multiuser or single user collections and collections with only some of the desired specifications available. This means that user provided terms, date/time and locational data are not all necessary required making images valid for the system, even if the majority of the images used in the test experiment include all these metadata fields. The motivation for this is to see the full potential of the system.

4.6.1. Location

Google provides application developers with different web services. I will use Google Maps API Service[12] in my implementation. Other than providing all search functionality which is included in the online version of Google maps[13], the Google maps API also includes great functionality for geocoding. In the system developed in this project the geocoding functionality is used to gather location specific information on different levels, i.e. name of country, region (e.g. district, area and state) city and street. This locational information is returned for each image in the collection by providing the GPS coordinates (latitude and longitude) to the Google Maps API. When the locational information is gathered for an image it is stored as its own attribute in the image *element* described in the previous section (4.5). Usually this kind of information are stored in the EXIF header of the image (see section 2.3.1), but since the image metadata used in this experiment are returned (by Flickr) as element trees containing several attributes for each element, augmented metadata are stored here. Notably all information gathered and augmented to the images in the collection are stored in the element tree. When all auto annotations are complete a new augmented image collection represented by the element tree are stored on disk and ready for further analysis.

[12] http://code.google.com/intl/no/apis/maps/documentation/webservices/index.html
[13] http://maps.google.com/

4.6.2. Atmospheric conditions

As part of the auto- annotations also weather condition will be collected using a combination of timestamp and location. This is achieved using a weather service provided by Weather Underground[14]. Weather underground allows users to access historic data from local location specific weather stations when providing a weather station ID. Weather Underground also supports searching a location providing GPS coordinates. A list of the closest weather stations are returned in sorted order, i.e. the closer the weather station is to the specific coordinate, the higher up the station ID is located in the list. Weather underground gathers weather information regularly from the local weather stations and provides users with historic information for specific locations, including temperature, humidity, visibility, wind direction, wind speed and weather condition (e.g. snow, rain, haze and clear). In the system developed temperature, wind strength and weather condition will be used.

Data represented by digits, will be translated into searchable terms to help create more accurate summaries for the image collection. E.g. wind speeds will be translated into searchable information by using Tropical Cyclone Classifications[15] and the Beaufort scale[16]. Examples would be translating wind speed into terms such as calm, light breeze, windy, storm, hurricane etc.

4.6.3. Converting date/time

The date and time of which the image is taken will be converted into several searchable terms which will be used to more easily see the diffusion of data. The date/time field is converted into month (e.g. January, February etc), day of week (e.g. Monday, Tuesday etc) season (e.g. summer, winter etc) and a combination of year and month (e.g. 2009 January). The motivation for these conversions is to increase the completeness of the collection summary and gather semantic information that is more efficient and natural for human requested search queries. Also it is much easier and lucid to locate representatives within certain time periods (e.g. Summer) instead of using certain dates which are less descriptive (e.g. 12. January).

[14] http://www.wunderground.com/
[15] http://cawcr.gov.au/bmrc/pubs/tcguide/ch1/ch1_3.htm
[16] http://www.stormfax.com/beaufort.htm

4.7. Cleansing and strengthening of data

After the image collections have been gathered from Flickr (section 4.5) and descriptive information has been added by augmenting images with additional metadata through auto- annotations (section 4.6) the image collections are almost prepared for the creation of the collection summaries. Before the representatives of the collection are selected for the collection summary (section 4.8), some cleansing and strengthening of the data are performed. I will describe this in the preceding subsections.

4.7.1. Generate unique metadata list

Before all metadata are cleansed the metadata are grouped together as a unique list of metadata to prevent redundancy in the data and to save runtime memory. All metadata hold the specific term, a term counter, number of users that have used this term within the collection and the images they are tagged in. This information will be used later on when calculating *term frequencies*, when updating database and so on.

4.7.2. Removing white spaces and character symbols

Since the user provided terms found in the images that are gathered from Flickr are manually inserted by individual users, many of the terms included in the *tags* or *title* field potentially include wrongly inserted words. Examples are misspelled words, words with symbols (e.g. "word#", "word_" or "word-"), punctuations (e.g. "word!", "word." or "word;") and white spaces (e.g. "word ", " word") that are not part of the word itself. This can interrupt the true reflection of the word frequency for the collection and it is therefore necessary to remove such characters to be able to define the correct term frequency for the collection. E.g. a tag added by a user can include a comma, an exclamation mark or other symbols and punctuations that can make similar words being separated into individual words, i.e. being viewed by the system as separate words with no relation to each other.

When it comes to white spaces this can concern the same aspects as for symbols and punctuations, i.e. that white spaces are included in the front or in the end of a word, but also simply by standing alone, being viewed by the system as an individual word. By inspection, the latter is the most frequent concern from the data that I am working with. Anyhow experimenting with vs. without removal of white spaces, punctuations and symbols suggests that cleaning the data increases the effectiveness of the term frequency analysis significantly.

4.7.3. Remove non roman characters

This project aims at finding representative features or properties that best describe the image collection as a whole. The image collections that I am working with are community contributed by users from all over the world. This means that potentially some of the images are provided with user provided terms in languages that are non- roman character based. Russian,

Chinese and Japanese are examples of languages that use non- roman character based symbols. Since these non- roman words are mixed with the roman words, the distribution of the words may seem confusing. Because of these actualities, along with that this project is limited to roman character based languages; I found it most useful to remove the non- roman character based language words from the collections that are used in the experiments of this project.

4.7.4. Removing stop-words and non-informative terms

Some of the most frequent terms, at least those provided by the users, are conjunctions, prepositions and determiners. Recall from section 2.2.1 that these are referred to as *stop-words* within IR. These are terms that do not have any specific meaning other than combining, helping or determining the gender of other words. In the context of image retrieval these are not helpful and irrelevant for retrieval and not helpful for describing a collection. The system removes conjunctions such as e.g. *and* and *or*, prepositions such as e.g. *until*, *from* and *to* and determiners such as e.g. *the*, *my* and *that*.

In the category of non- informative words are words such as camera model and the specific label *"geo-tagged"*. It seems that many cameras tag images it takes by adding the camera name and model number to all images, e.g. "Canon Eos 350D".

Also a large percentage of the geo tagged images on Flickr are tagged with the specific label *"geo-tagged"*. I am uncertain if this is a tag added by Flickr automatically as it locates location data in images or if it's manually added by users to mark the image as geo- tagged. I believe that the latter is the most logic explanation as I have seen small variations in the use of the term, e.g. "*geo-tagged"*, *"geo- tagged"* and *"geo tagged"*. Removing such words, which in some cases can be the most frequent terms throughout the image collection, increases the usefulness and importance of the most representative properties of a collection summary. Also the image collection summary becomes more informative when it is taken into consideration that it will be used with basis for efficient image retrieval.

4.7.5. Removing oversized tags

Some user provided terms can be very specific for an individual user, either by personal terms in the form of family or friends names, URL's to personal blogs or home pages, or simply chronically misspelled words as a product of wrong spelling instincts for individual users. I will refer to these kinds of tags as *special tags* throughout this section.

As explained earlier the system developed in this project is designed for analysing image collections of different kinds, i.e. not only for multi user collections, but also for personal image collections or collections of images added by few or by an individual user.

When experimenting with different approaches to calculate term frequencies one of the first approaches that I used was calculating term frequencies using frequency of term divided by number of terms available, instead of images available. With the first approach the more terms available the lower frequencies for given terms are, hence removal of as many useless tags such as *special tags* are desired when present. In this way frequencies of the tags that are left and potentially representative are strengthened. The weakness of this approach is that the more *special tags* that are not located for removal, the lower the ones left are weighted. The latter involves removing of less frequented tags, such as *special tags* and gives the more relevant tags an overall of higher frequencies.

This is one of the reasons why I chose to calculate *term frequencies* only taking into consideration the number of images without taking into account the total number of tags used (described in detail in section 4.8.3). When only taking into consideration the number of images within the collection, the precaution of removing useless tags such as *special tags* are not necessary since poorly weighted tags does not influence the outcome on others in a negative direction, actually the opposite has been observed. As I will describe later on user provided terms are grouped together into higher level representations (hypernyms). Hypernym members are potential synonyms to each other, which means that a hypernym can be strengthened by poorly frequent words, such as advanced less frequently used words, if they have similar meaning. In this way if many synonymic terms are present they can together become part of the collection summary if they together are representative enough.

For this reason removal of *special tags* are limited, the ones that are left for removal are simply those that are in conflict with the database constraints on variable lengths. Tags and terms that consist of strings of more than 35 characters are removed and assumed to be a *special tag.* The most highly frequent cases of such *special tags* are manually inserted tags that for some reason are not separated with white spaces, i.e. a combination of words merged together, e.g. “word1word2word3”. Another example are URL’s to individual users blogs, referenced online sites etc. Anyhow the removal of such tags does not affect the weighting of others and are only removed to prevent database constraint conflicts.

It can be argued that misspelled words added either unintentionally or intentionally through e.g. expressions, abbreviations or code words related to figure of speech or strings of merged words could be located and translated to its direct meaning. In this way a more semantic and complete description for the collection are retained. However, because of the complexity of locating and capturing the exact meaning of expressions, abbreviations or code words, along with resulting in only a small increase in effectiveness, I will not take this into consideration in this project.

4.7.6. Hypernyms

Hypernyms are for individual *user provided terms* located using the Wordnet search[17] Interface. WordNet is an English lexical database where words and terms are represented as an ontology, which includes descriptions that represent the semantic meaning of words. In WordNet, higher level representations of terms are referred to as hypernyms. To prevent confusion which includes confusion for those readers that are familiar with WordNets terminology I will adopt this terminology.

Other than *hypernyms*, WordNet also allows gathering of synonyms. Synonyms would increase the effectiveness of image retrieval, but does not have any benefits in the production of collection summaries. Synonyms in the available unique term list of the collection will anyhow be recognized in the same hierarchical level through *hypernyms* described next.

The WordNet Interface allows easy manoeuvring through the hierarchical structure of words, e.g. moving one level up in the hierarchical structure of the word *car* returns the *hypernym motor vehicle*. The *hypernym motor vehicle* also includes other motor vehicles such as *motorcycle*, *SUV*, *sedan*, *bus*, *coach*, *ambulance* etc. These hypernyms are gathered by the system with the intention to group together words that are found in the image collection and help make the summaries of the image collection more complete. The system uses the higher level representations of words to find representative hypernyms that may be worthy to include in the image collection summary. E.g. word hypernyms consisting of several words which altogether reach a frequency above the *selection threshold* are included as a word *hypernym* in the collection summary, with all its members included, even if all of the words alone have frequencies below the *selection threshold*. The idea behind this approach is to capture those words that alone are not found to be representative for the collection and strengthen them through their *hypernym* members.

Words within the same *hypernym* are in many cases so closely related to each other that using the *hypernym* as a representative helps providing a more complete and semantically meaningful summary. Say e.g. that a collection includes the terms *SUV*, *Sedan*, *Station Wagon*, *Ambulance*, *Truck* and *Motorcycle*, all of which have frequencies below the *selection threshold* and are therefore not included as representatives for the image collection. Members located within the collection that together are grouped together into a *hypernym* are viewed as representative for the collection, if they together form a term frequency above the *selection threshold*. In the context of a user requesting a search query to the system it is not necessarily a fact that he will specify the search on members of a hypernym, he may be looking for a wider range of results, e.g. by a hypernym such as *motor vehicle*. Hence, including *hypernyms* in the summary increase the usefulness of the representatives, as well as making the description of the collection more semantically meaningful.

[17] http://wordnetweb.princeton.edu/perl/webwn

Also words that are grouped together can help increase the effectiveness of the image retrieval system by giving alternative feedbacks to the user in cases where a specific word are not found to be frequent. E.g. if a user request a query for the word *car*, results under the hypernym *"motor vehicle"*, i.e. *SUV, Sedan, ambulance* and so on may be viewed as relevant alternatives.

Recall from section 2.2.1 that normalization, stemming and lemmatization where introduced. Using the WordNet API, grouping terms into hypernyms provides great ways for dealing with all of these.

In cases where *hypernym* members within a collection are not able to create term frequencies above the given *selection threshold*, the system supports further manoeuvring in the hierarchical structure. E.g. in the case of the term *car*, the first level is recognized to be the hypernym *motor vehicle*. If all members of *motor vehicle* together do not reach *term frequencies* above the given *selection threshold*, the *hypernym* members are taken to the next hierarchical level. In the case of *motor vehicle* the higher hierarchical level are *self-propelled vehicle*, which includes more alternatives included candidates located under *motor vehicle*. E.g. a *self- propelled vehicle* would extend *motor vehicles* with vehicles such as *tank, go-kart, steam locomotive* and so on. The system allows manoeuvring of up to three levels from its basis, where in the case of the term *car* the hypernym *motor vehicle* is the first level, *self-propelled vehicle* is the second level and so on.

4.8. Finding representative terms

4.8.1. Defining an representative

From the oxford dictionary [47] the definition of representative is the following:

> *Adjective: containing typical examples of many or all types.*

This definition is not far from how I use the term in this thesis. This means that a representative term is an exemplification of the class or kind, a *representative* for the collection summary. When summarizing image collections I have to deal with some reduction of the search space, by carefully choosing what *terms* to include in the summary of the Image collection that I am working with. The image collection summary will include e.g. *user provided terms* that are representative for the collection and describes the image collection through its main representatives and highly weighted terms. Representatives are terms that describe the image collection from its most informative view, and should stand out from other terms in the collection. This means that in extreme scenarios where no terms of an image collection are prominent from the rest, no terms are viewed as representative for the collection.

A term of an image can have different importance for different contexts. One such example is in the contexts of a search query requesting members of a defined *hypernym* instead of a specific term used in the image collection, e.g. a request for the hypernym *motor vehicle*.

With exception of converting of date/time, other contextual terms are metadata gathered during augmentation. I will start by describing how the *selection threshold* is decided for different kinds of *metadata terms* and how *term frequencies* are calculated to be compared against it.

4.8.2. Selection Threshold

When dealing with reduction of the search space it is important to find a way to locate representatives for different types of terms other than just through logic and semantic understanding. As mentioned in the previous sections a *selection threshold* value is used to decide whether or not a specific term is representative for the image collection. The *selection threshold* is different depending on what is most natural for the metadata to be compared against.

For metadata, such as *user provided terms* and *locational terms*, i.e. where the total amount of outcomes are not fixed, the *selection threshold* will simply be represented by a constant C. The *selection threshold* for metadata without a fixed number of outcomes can be defined as;

$$(1)\ Selection\ threshold(unfixed\ ) = C$$

The constant represents how many percentages of the images a specific term has to be an instance of to be viewed as representative for the collection.

For metadata that have a fixed number of outcomes, the *selection threshold* is set to one over total outcomes added with a constant C times one over total outcome, that is;

$$(2) Selection\ Threshold(fixed) = \left(\frac{1}{total\ outcomes}\right) + \left(C * \left(\frac{1}{total\ outcomes}\right)\right)$$

E.g. for the time specific metadata *month* which have 12 different outcomes where the constant C are set to ½, the *selection threshold* are;

$$Selection\ Threshold((date/time),\ month) = \frac{1}{total\ outcomes\ month} + \left(C * \left(\frac{1}{total\ outcomes\ month}\right)\right) = \frac{1}{12} + \left(\left(\frac{1}{2}\right) * \left(\frac{1}{12}\right)\right) = 0.125$$

The *selection threshold* for the time specific metadata *weekday*, which has 7 different outcomes, would be;

$$Selection\ Threshold((date/time),\ \ weekday) = \frac{1}{total\ outcomes\ weekday} + \left(C * \left(\frac{1}{total\ outcomes\ weekday}\right)\right) = \frac{1}{7} + \left(\left(\frac{1}{2}\right) * \left(\frac{1}{7}\right)\right) \approx 0.214$$

Using a *selection threshold* for metadata that have a fixed total number of outcomes (2) will always result in a *selection threshold* higher than the total number of outcomes as a fraction. This means that in extreme scenarios where images in a collection are equally distributed over e.g. the 12 different *months*, none is viewed by the system as representative. This means that no one is viewed as representative instead of all, which is much more logical when no terms stand out from the others and also helps in sustaining the definition of representativeness defined in the previous subsection (4.8.1). It is important to notice that a representative for the collection is terms or properties that stand out in the collection, and if no one stands out, none will be used as representatives for the collection either.

To get a more visual view of the *selection threshold* it has been pictured in *Figure 4-3*. The green area are the tags chosen to be the representative tags of a given image collection. Notably the whole scale has in advanced been cleansed through removal of words. Cleansing from this area includes white spaces, character symbols, non-roman characters, conjunctions, prepositions, determiners, non-informative terms and oversized tags as described in the previous section.

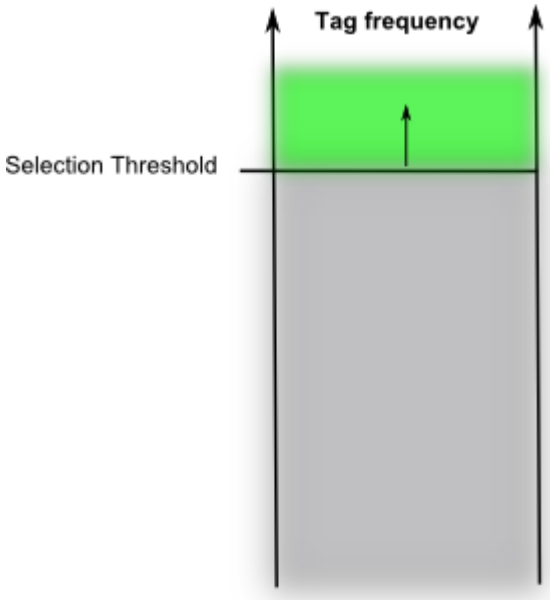


Figure 4-3: Selection threshold

4.8.3. Metadata weighting

To calculate the representativeness of terms I decided to use the Term Frequency – Inverse Document Frequency (TF- IDF) approach from the field of information theory (see section 2.1.2). I use this approach to calculate term frequencies for *user provided terms*. Also I calculate TF- IDF for contextual terms such as weather, location and date/time. All the different kinds of metadata are calculated in the same way. The only difference between them is that the *selection threshold* (section 4.8.2) varies depending on what is most natural for the metadata it is being compared against. In this section I will use the *user defined terms* as an example in calculating the TF and IDF, before comparing it against the *selection threshold* to decide whether or not it is viewed as a representative for the collection.

All *user provided terms* from all the images in the collection are together viewed as a single document, also all *locational*, *date/time specific* and different *weather specific terms* are viewed as individual documents in the eyes of the TF system. I call them documents as this approach is adopted from Information retrieval (section 2.2). For each term within the document in this sense, in this case *user provided terms*, a term frequency is calculated by the following formula;

$$TF\ (user\ provided\ term, term) = \frac{frequecy_{term}}{Total\ number\ of\ images\ in\ collection}.$$

Say that we have an image collection with 1000 images. Further say that there is 40 instances of the word *car* in *user provided term* within the collection, the term frequency for 'car' is then:

(1) $TF(user\ provided\ term, "car") = \frac{frequency_{car}}{Total\ number\ of\ images\ in\ collection} = \frac{40}{1000} = 0.04.$

Total amounts of terms used within an image collection are not taken into consideration when calculating the *term frequency* for metadata mainly because we are only interested in metadata that stand out from the collection as a whole, i.e. terms connected with a sufficient amount of images. Calculating the frequency over *total amounts of terms* instead of total *amounts of images* would work well for situations where the majority of the images are tagged, but not if just a few of the images are tagged:

E.g. if we have a collection with 100 images, where *user provided terms* are only present in 2 of the images. In these two images there are 2 unique terms for each image, i.e. 4 unique terms in total. Taking into consideration the *total amounts of terms* in this situation would result in all unique words

being 25 % of the collection, and would most probably be viewed as representative, which is in conflict with the definition of representativeness that I use in this project (see section 4.8.1). Calculating term frequencies taking into consideration *number of images* within the collection gives much more stable results when dealing with image collections with various degrees of manually added textual terms and is therefore preferred.

Notably, the developed system only use one unique user provided term per image. Say e.g. that a specific image A includes the title *"trip to Italy"*, while the tags field includes the tags; *summer*, *Italy*, *monument*, *family*. This means that *Italy* have are used in two occasions, nevertheless the system only counts the term *Italy* once, to prevent duplicates in the term frequency.

The Inverse Document Frequency (IDF) is calculated taken into consideration the number of image collections available and how many of the collections that contain the specific term. The inverse document frequency is calculated using the following formula;

$$IDF(user\ povided\ terms, term) = \log\left(\frac{number\ of\ avaliable\ image\ collections}{number\ of\ image\ collection\ containing\ user\ defined\ term}\right).$$

E.g. say that we have 50 image collections, where 9 of the documents contain one or more instances of the term *car* in the any of the *user provided metadata fields*, for this situation the IDF would be:

$$(2)\ IDF(user\ povided\ terms, term) = \log\left(\frac{50}{9}\right) \approx 1.61.$$

The TF-IDF is the product of the *term frequency* (TF) and the *inverse document frequency* (IDF). Using the two examples the TF-IDF for the term *car* would be:

$$(3)\ TF - IDF(tags, "car") = (1) * (2) = 0.04 * 1.61 \approx 0.064.$$

The idea behind the TF-IDF approach is that terms that are rarely used in the available set of collections should be more appreciated in the few collections they are. In this way it is a higher possibility that a specific rarely used term is marked as representative in one of the few image collections the term is an instance of.

E.g. say that the *selection threshold* in this case is set to 0.05, i.e. that at least 5 per cent of the images has to have a certain *term* as an instance for the *term* to be viewed by the system as representative for the collection. Since the TF from (1) is below the *selection threshold* the word would not be recognized as representative, but because of its uncommonness in the set of image collections available, the IDF (2) helps increase the term frequency

above the *selection threshold*, reflected by the product of the two (3). Hence the term is recognized as a representative for the collection.

It is worth mentioning that within the application area that this project aims for it is not given that the number of image collections available is known. This is an element of concern when dealing with dynamic online image collections. In such circumstances it is more probable that a recursive method scans different online domains before stopping at a given threshold. When enough image data has been found to satisfy the search query, it stops. In such cases the IDF is impossible to find, but also not needed.

Using the IDF to strengthen uncommon terms, which are close to a given threshold, with intention to include it as a representative is strictly applicable in stable and defined environments where number of available collections can be defined. In cases where total number of collections are not known, or useful, the term frequency TF are used alone to decide whether or not a term is representative for the collection.

4.8.4. Collection summary partitioning

Because of the different kinds of metadata used by the summarizing system, the collection summary itself is also represented in accordance with these metadata fields. As mentioned, a specific metadata field is viewed as an individual document in the TF- IDF system. Representative properties are selected using the *selection threshold* which corresponds to the natural representation of the specific metadata terms. The main sections of the image collection summary are *User provided terms, locational terms, weather specific terms, date/time terms and hypernym terms.*

Many of these main sections also have sub categories which are used to keep better track of specific semantic information about the data which is included within them. In *Figure 4-4* the main categories along with their sub categories are listed.

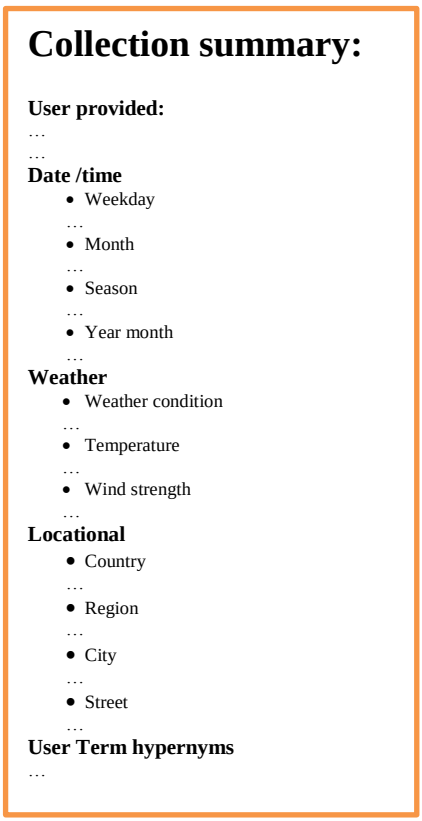


Figure 4-4: Collection summary partitions

From *Figure 4-4* we can see that only the context related data have sub categories. I have described the different partitions through this chapter, but to get a brief overview; *user provided terms* are strictly information gathered from the 'tags' and 'title' metadata fields from the images on Flickr, *locational* terms is annotated metadata using Google's Maps API (section 4.6.1), *weather* terms are weather information gathered using Weather underground (4.6.2), *date/time* terms is annotated information using time stamp conversions (4.6.3), and *user term hypernyms* are groups of words belonging to a specific hypernym from WordNet (4.7.6). Real outputs of the summarizing system will be presented and discussed in chapter 6.

Chapter 5

5. Implementation

In this chapter implementation specifics will be described. The overall view of the system has been presented in the previous chapter, so to prevent repeating myself I will in this chapter mainly focus on implementations choices that have had to be made using the libraries and API's included in the implementation of the system. I will also describe the formats of the data used. I will start by presenting the hardware on which the system has been developed, before presenting the API's and library packages included in the development of the system and how they have been used.

5.1. Hardware

The system is developed and tested on a Intel® Core™ i5-450M Processor (3M L3 cache, 2.40 GHz), having 4GB - DDR3 RAM, a SATA 5400 rpm hard drive running on a 64 bit version of Microsoft Windows 7 (Home Premium) SP1.

5.2. Summarizing system

The summarizing system is developed using python version 2.6.6 for win32. The system is far from optimized. E.g. the task of augmenting images with additional metadata is very time consuming because of the restrictions laid by the providers of locational and weather data, I will discuss the bottle necks for these in section 5.4 and 5.5. Anyhow great amounts of work have not been laid down into optimizing the system for runtime performance since that is not of importance in achieving the goal of this project. Anyhow the analysis and production of the collection summaries are quite fast. To mention some specifics that could easily be optimized is using indexes for all lists holding the data used during runtime, i.e. all data analysed for the collection. The sudo code for the summarizing system is located in Appendix B, the main of the system in appendix B-1.

Image collections used in this project is gathered from Flickr. The Flickr API is described in the next section (5.3). Images are augmented with locational data using the Google maps API, described in section 5.4. Also images are augmented with atmospheric conditions using historic data from Wunderground, which are described in section 5.5. Further manually added *tags* are grouped together as hypernyms (higher level representations) and synonyms are located for each *tag* if available using the WordNet API, described in section 5.6. MySQL database server is used to store the data used by the summarizing system and the python MySQL server interphase is used for communicating with the database. This is described in section 5.7.1. Finally, the representatives of the collection summary of the analysed image collection are stored to an Excel file (.xls) for simplifying the further analysis of the data. The XWLT Library used for this purpose is described in section 5.7.

5.3. Flickr API

The image collections used in the testing of this system are gathered from Flickr. To communicate and gather information from Flickr I have used the FlickrAPI package[18], version 1.4.2 compatible for python 2.6.*. The Flickr API allows developers to use all search functionality, and access all images in the same manner as the online version of Flickr. To get access to the Flickr's database an API key is required, which can be accesses for developers when registering as a Flickr user. The API key allows boundless access to Flickr's database. The documentation for the Flickr API is located on Flickr's API pages[19]

The implementation of the authorization and communication with Flickr are found in *Appendix B-1*.

The Flickr API is a very powerful and functional rich API, nevertheless in the implementation of my system I simply use the following two methods:

1. flickr.groups.search[20]
2. flickr.photos.search[21]

5.3.1. Flickr Groups search

In the testing of my system I use Flickr groups as image data, as they easily can be viewed as an existing collection of images. I have located 17 groups which I will use in my experiment, all of which are geo-tagged, included with tags, date/ time stamp and a title. 7 of these image collections are used in the evaluation of the system, presented in chapter 6.

Notably as presented in section 4.4 the requirement specification defines that not all images in a collection analysed by the system have to have all the defined metadata fields available. Nevertheless, in the experiments performed the majority of the images include all of these metadata fields. To prevent confusion with the requirement specification (section 4.4), this is mainly to see the full potential of the system.

User provided terms are desired for the experiment since this make the collection more natural to human perception as these fields are manually added by individual users. Geo-location is included in the images in the form of latitude and longitude values and is used for several purposes. Geo-location are alone used to auto-annotate the images with locational terms, i.e. name of country, city, region and street, while a combination of date/ time stamp and geo-location are used to augment the images with weather specific terms.

[18] http://pypi.python.org/pypi/flickrapi
[19] http://www.flickr.com/services/api/
[20] http://www.flickr.com/services/api/flickr.groups.search.html
[21] http://www.flickr.com/services/api/flickr.photos.search.html

To get access to images from a specific group through the *photos search* method, the *group id* has to be specified. The *groups search* method takes a search query as input and returns information about a group, including the *group id*. When the *group id* is located the images from the corresponding group are ready to be collected by the summarizing system.

5.3.2. Flickr photos search

When the given *group id* has been provided by the *groups search* method, the images from the group are gathered using the *photo search* method. The *photo search* method allows a wide range of criteria's to be specified. Other than specifying the group id of the images to be returned I also specify that only *geo-tagged* images should be returned and that, the metadata fields, *"tags"*, *"title"* and *"GPS coordinates"* should be included when returned.

5.3.3. Return Format

The image metadata returned from Flickr are returned in XML format. The XML data returned can be difficult to handle and difficult to grasp because of the huge amounts of data that I am working with. To make the data a bit more lucid, I have decided to handle the XML data returned by Flickr in the form of an element tree. For this I have used the element tree package[22], version 1.2.7_20070827. This library package is a light weight XML object model for python 2.6.*. The element tree gives the image metadata gathered from Flickr a basic tree structure which I found easier to visualize when handling and processing the data. Flickr returns a partition of the image results at a time in a so called page. A page may include from one to a maximum of 500 image elements per page. Using the element tree I insert all images from all pages into one, a single level in the element tree. The element tree only consists of two levels, the first node represents the *photos* node, and all image elements are a sub node of the main node.

In *Figure 5-1* an element tree with metadata as returned from Flickr are shown. Notably the return element tree, where each element represents an image, includes much more information related to an image than represented in the example. To prevent confusion I have only listed the ones that are vital for this part of the system.

The <photos> tag is the defined beginning of the element tree, whereas the <photo> tags represent the elements, i.e. the images in the collection. The red fields are the attributes of a photo element whereas the black fields are the values of the corresponding attribute.

[22] http://packages.pardus.org.tr/contrib/binary/ElementTree.html

```
<rsp>

<photos total="313">

        <photo datetaken="2008-07-22 09:47:45" latitude="2.819289"
                longitude="104.159408" tags="sea sky mer storm island vent boat
                Asia wind" title="Electric sky" ….. </photo>

        <photo datetaken="2008-06-05 11:19:29" latitude="6.112392"
                longitude="100.364999" tags="tower Malaysia Kedah alorstar
                park" title="Alor Star - Kedah"….. </photo>
        ….
        ….
</photos>
</rsp>
```

Figure 5-1: Example element Tree

5.4. Google Maps API

Locational terms are gathered using the Google maps API[23] (version 1.0.2) which works with all versions of python. The Google maps API requires the JSON (JavaScript Object Notation) package to work. Python 2.6.6 comes included with JSON version 1.9 which seems to work flawlessly for the used version of Google maps API.

The implementation for the gathering of locational information is found in *Appendix B-2.1*.

Pythons Google maps API is a simple, yet powerful tool, which not only allows access to all functionality which is found in the online version of the service, but also comes with great functionality for geocoding. The geocoding functionality is the part of the package that I use in my implementation.

Google maps Geocoder takes the GPS coordinates from each photo element as arguments and returns the coordinate's locational data, i.e. name of country, region, city and street. When locational terms have been collected, the system adds the collected metadata as a new attribute to the corresponding photo element. The result after augmenting the images in the element tree with locational metadata terms is presented in *Figure 5-2*. The greyed out area with the attribute "location-augmented" is the augmented metadata using the Google maps API's geocode functionality.

[23] http://pypi.python.org/pypi/googlemaps/1.0.2#downloads

```
<rsp>

<photos total="313">

        <photo datetaken="2008-07-22 09:47:45" latitude="2.819289"
                longitude="104.159408" tags="sea sky storm island vent boat Asia
                wind" title="Electric sky" location-augmented="street:tioman
                island, city:mersing, region:, country:malaysia"….. </photo>

        <photo datetaken="2008-06-05 11:19:29" latitude="6.112392"
                longitude="100.364999" tags="tower Malaysia Kedah alorstar
                park" title="Alor Star - Kedah"  location-
                augmented="street:lebuhraya sultan abdul halim, city:alor setar,
                region:kedah, country:malaysia"….. </photo>
        ….
        ….
</photos>
</rsp>
```

Figure 5-2 : Element tree augmented with locational data

The Google maps API requires an API key. The API key has restrictions to how many lookups that can be performed daily and how regularly lookups can be performed. I believe that the daily lookup limit is set to 15000 on a 24 hour basis. When it comes to lookup regularity I have implemented a function call which sleeps for 0.2 seconds between each lookup which seems to work like a charm. Leaving it out does not and may result in the API key being blocked for 48 hours conferring to Google support.

The rate limits set by Google has restricted the size of the data that I am working on, since it takes some time to augment image collections with additional metadata. Nevertheless, it only took two to three days to augment 35-45000 images gathered from Flickr which seems to be sufficient for my experiment. To prevent unnecessary lookups during my experiment, all element trees gathered, both clean returns from Flickr and trees augmented with additional metadata are stored to disk for further usage.

Notably it is possible to get a Premium API key from Google, which allows up to 100 000 geocode lookups every 24 hour. Presumably this is licensed; anyhow I didn't find this necessary for this project.

5.5. Wunderground historic

Weather information is collected using weather underground. Weather underground does not come with an API built for python, but historic information can be gathered using an online API[24] provided. Also

[24] http://api.wunderground.com

documentation on *how to use*[25] the API is available. The implementation for the gathering of atmospheric conditions is found in Appendix B-2.2.

I found the online weather underground API to be very unstable, resulting in server timeouts and complete hang ups. Consequently, I decided to gather atmospheric conditions by parsing the pure HTML code returned by the html version of weather underground historic[26]. To gather the html code in a reasonable fashion I use the built in python library urllib[27], which provides a high level interface for fetching data across the World Wide Web.

Weather underground historic requires a weather *station id* to lookup weather data for a given date on a specific location. Wunderground support gathering of these *Station id's* by providing a GPS coordinate, i.e. latitude and longitude. Provided with the coordinates the closest weather station and its ID is returned. The GPS coordinates within each image in the collection are used for this purpose. URL vice the closest weather station is looked up in the following manner:

> *http://wunderground.com/auto/wui/geo/WXCurrentObXML/index.xml?query="**Latitude, Longitude**"*

When the resulting html code has been returned and the closest weather station has been located with its *station id*, the correct URL to the service is built using the *station id* and the *date/ time* stamp located in the photo element already provided. An historic lookup is URL vice performed in the following manner.

> *http://www.wunderground.com/history/airport/**stationId/year/month/day**/DailyHistory.html*

The returned page contain hourly weather information, which includes temperature, humidity, wind strength, weather condition (i.e. rain, snow, clear etc.), wind direction and so on. The table containing the weather information is parsed before the best match for the image being analysed. This is located and stored as a new attribute for corresponding photo element. The best match is of course the hourly weather data that correspond to the time of which the images are taken. Notably the time format of the weather data is in 12 hour format while the time format of when the images are taken is in 24 hour format. For this reason the time field from the weather data is translated to 24 hour format for comparison at execution time.

The update of the element tree after atmospheric conditions has been gathered are presented in *Figure 5-3*, again the greyed out area are the newly added metadata for the images.

[25] http://wiki.wunderground.com/index.php/API_-_XML
[26] http://www.wunderground.com/history
[27] http://docs.python.org/release/2.6.6/library/urllib.html

```
<rsp>

<photos total="313">

        <photo datetaken="2008-07-22 09:17:45" latitude="2.819289"
                longitude="104.159408" tags="sea sky storm island vent boat Asia
                wind" title="Electric sky" location-augmented="street:tioman
                island, city:mersing, region:, country:malaysia" weather="9:00
                AM, 25.0, 24.0, 94%, 0hPa, 10.0kilometers, SW, 5.6km/h/1.5m/s,
                -, N/A, , Mostly Cloudy" ….. </photo>

        <photo datetaken="2008-06-05 11:19:29" latitude="6.112392"
                longitude="100.364999" tags="tower Malaysia Kedah alorstar
                park" title="Alor Star - Kedah" location-
                augmented="street:lebuhraya sultan abdul halim, city:alor setar,
                region:kedah, country:malaysia" weather="11:00 AM, 31.0, 26.0,
                75%, 1010hPa, 10.0kilometers, NNW, 1.9km/h/0.5m/s, -, N/A, ,
                Mostly Cloudy" ….. </photo>
        ….
        ….
</photos>
</rsp>
```

Figure 5-3: Element tree augmented with atmospheric conditions

Notably not all weather information gathered for the images are used by the summarizing system. The weather information used so far is *temperature* (represented as "31.0" in lower element of *Figure 5-3*), *wind strength* (represented as "1.9km/h/0.5m/s") and *weather condition* (represented as "mostly cloudy").

The *weather condition* is used as it is, while *temperature* and *wind strength* are translated into a more human readable and searchable format. I did not locate a *human readable* translation of *temperatures*, for this reason I have made my own scale represented as my personal perception of temperature, the translation scale used by the summarizing system is as follows:

Scale: *from* - *to* (in °C)	**Translation (human readable)**
(-100.0) - (-15.0)	Freezing
(-14.9) - (-10.0)	Ice cold
(-9.9) - (0)	Cold
0.1 - 10.0	Chilly
10.1 - 20.0	Moderate
20.1 - 25.0	Hot
25.1 - 35.0	Very hot
35.1 - 100.0	Extremely hot

Table 5-1: Temperature translation scale

For the translation of wind strength I've used the beaufort scale[16]. The wind strength is represented by both *km/h* and *m/s*. Anyhow the summarizing

system only use the *m/s* format. For this reason the beaufort scale is also translated. The scale used by the summarizing system is as follows:

Scale: *from* - *to* (in m/s)	Translation (human readable)
0.0 – 0.3	Calm
0.3 – 1.5	Light air
1.6 – 3.4	Light breeze
3.4 – 5.4	Gentle breeze
5.5 – 7.9	Moderate breeze
8.0 – 10.7	Fresh breeze
10.8 – 13.8	Strong breeze
13.9 – 17.1	Moderate gale
17.2 – 20.7	Gale
20.8 – 24.4	Strong gale
24.5 – 28.4	Storm
28.5 – 32.6	Violent storm
32.7 – 1000	Hurricane

Table 5-2: Wind strength- Beaufort translation scale

5.6. WordNet API

Hypernyms are gathered using the NLTK (natural language toolkit), version 2.0b9 usable for python 2.6.*. The NLTK package includes many tools and corpuses used for language processing and computational linguistics. The one that I use in my implementation is the WordNet Interface. Terms that are an instance of either "title" or "tags" are looked up from the wordnet database, with intentions to find hypernyms used to group words together. The approach has been thoroughly described in *section 4.7.6*, so I do not go deep into the specifics here. The implementation of the hypernym handler is found in *appendix B-6.*

Nevertheless, some elements are still worth mentioning. When a word is looked up a list of candidates is returned by WordNet, i.e. if the term looked up have different semantic meanings. If several candidates are returned, the first candidate is assumed to be the correct one. Nevertheless the candidates returned are sorted in prevailing order, which means that the most common used words are listed first and the more uncommon last. In *Figure 5-4* the returned candidates for the word *car* are listed.

- S: (n) **car**, auto, automobile, machine, motorcar (a motor vehicle with four wheels; usually propelled by an internal combustion engine) *"he needs a car to get to work"*
- S: (n) **car**, railcar, railway car, railroad car (a wheeled vehicle adapted to the rails of railroad) *"three cars had jumped the rails"*
- S: (n) **car**, gondola (the compartment that is suspended from an airship and that carries personnel and the cargo and the power plant)
- S: (n) **car**, elevator car (where passengers ride up and down) *"the car was on the top floor"*
- S: (n) cable car, **car** (a conveyance for passengers or freight on a cable railway) *"they took a cable car to the top of the mountain"*

Figure 5-4: WordNet "car" query return

Figure 2-1 shows that the word *car* returns 5 candidates, where candidate one is the most common. In this situation the semantic meaning that first come up when thinking about *car*, seems to satisfy my demands. Since we do not have any contextual information about the *title* and *tags* metadata fields, the system can never be 100% certain that a term located using the WordNet interface is correct.

After studying the hypernyms located for the test data, it seems reasonable to consistently select the first candidate. By inspection, the results are rarely placed in wrong hypernym groups. The only words that we are interested in, is in the end highly used words which most likely are viewed by WordNet as prevailing, hence returned as candidate number one.

As mentioned in the *chapter* 4, hypernyms are located for terms manoeuvring in up to three levels in the hierarchical structure. In such situations hypernyms, which are grouped as hypernyms are looked up in the WordNet database in the same way as terms are in the first place, copying the terms located under the previous level as members of the next.

5.7. Storing collection summary data for later use

The analysis of a given collection and the creation of the collection summary may be costly in time and resources, at least for big collections with a large amount of images and image metadata. This is not a concern in general since the production and analysis of the data itself are quite fast. This concern is mainly a concern in my implementation. I have not developed any index structures for the lookups done in the unique term list and other lists processed in memory at execution time. Adding some index structures to these parts of the system would increase the effectiveness of the system to a great extent. Also some bottle necks are introduces by the limitations in the *request per second* set by the API provider used to augment images with additional metadata, discussed in section 5.4 and 5.5. Because of the limited time for this project these concerns are left unsolved, or at least unimplemented. The implementation of the communication with the database is found in *appendix B-10.*

Nevertheless further analysis on the produced data is more achievable if the data is stored and structured in a reasonable manner. For these reasons the data collected from the summarizing system are stored in a relational database, using MySQL. In this way the data collected can more feasibly be used in future projects without having to produce the outputs over again.

5.7.1. Environment specifics

For the purpose of storing the image collection data used by the summarizing system, MySQL server version 5.1.56 for win32 are used. All communication, insertion and updating of the database are performed through the developed system using the MySQL database interface MySQL-python (MySQLdb) package version 1.2.3. Available for win32 and python 2.3.* through python 2.7, The MySQLdb package are usable for MySQL

server version 3.23 through MySQL server version 5.0. The design of the database is presented thoroughly in section 4.8, hence not discussed any further here. The implementation of the communication with the database is presented in appendix X.

5.7.2. E-R Diagram

The entity relation diagram for the database system developed is presented in *Figure 5-5*.

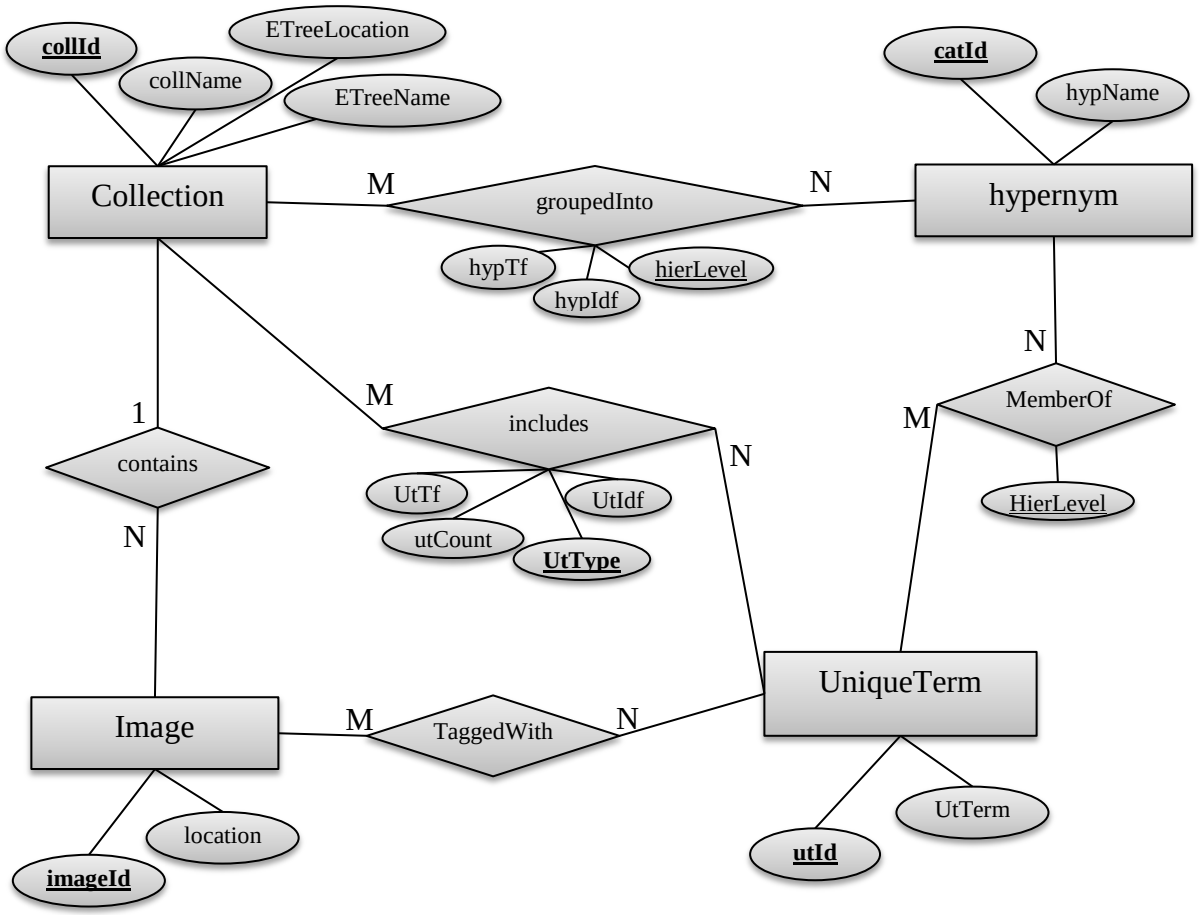


Figure 5-5: E-R diagram for storage of collection summary data

The E-R diagram consists of four entities, i.e. *collection*, *image*, *uniqueTerm* and *hypernym*, which together share five relations between them.

5.7.3. Representation of the Entities and Relations

The centre of attention is the *uniqueTerm* entity, which holds all different types of metadata connected with an image, which include contextual auto-annotated metadata and manually added metadata, i.e. *user provided terms*. The term- type is identified through the entity attribute *type* in the *includes* relation. UniqueTerm have three relations which makes its connections with collection, hypernym and collection. Notable all unique terms are located in

the database. Representatives are located through their *term frequency* and *inverse document frequency*.

Terms are connected with a specific image collection through the *collection* entity. The collection entity holds all image collections used in the test experiment, i.e. 17 image collections gathered from Flickr consisting of 300 to 6000 images. To prevent redundancy in *UniqueTerm* the relation between this entity and the collection entity includes the term frequency (ut_tf), collection specific term counts (ut_count) and inverse document frequency (ut_idf), i.e. all attributes that are not guaranteed unique for specific terms across different image collections.

Further the terms used within a collection are connected with one or more *hypernyms* The hypernym entity represents *hypernyms* gathered for terms looked up using WordNet, as mentioned earlier *hypernyms* gathered from WordNet is a higher level representation of a specific term, e.g. the *hypernym* for the term *car* is *"motor vehicle"* at the first level. As for the *UniqueTerms* redundancy is prevented based on that no duplicate hypernyms are inserted into hypernym, for this reason the relation between the two includes the attributes *collection_id* and *hier_level* which are the only attributes between them that would result in redundant tuples. The first attribute are introduced to separate the tuples from different collections and the latter to separate a unique term from all its hierarchical levels (at the most three). As discussed in *section 4.7.6*, hypernyms are gathered for a specific term using WordNet up to three levels from its basis, therefore to hold the normalization the uniqueTerms has to be included with its hierarchical structure level.

The last relation for *uniqueTerm* is the mapping between terms located in a specific collection and the images from the collection that includes these terms. It could be argued that it is sufficient to connect the *image* entity to the *collection* entity and since the *collection* entity are in relation with uniqueTerms, e.g. collecting all images that relates to a certain collection. But I found it also necessary to connect all images with the terms they are tagged with. The main motive for this design choice was that I found an *inverted files index structure* to be the most appropriate and well-suited candidate in the context of this project. The *inverted files index* structure is more thoroughly described in the theoretical background (*section 2.2.3*), but the main disciplines that I have adopted when designing the E-R diagram for the database are the following:

1. The data structure is consulted to identify whether the term is located in the database. In the context of my system this lookup would be on representative metadata for a given collections, i.e. terms and hypernyms above a certain threshold.
2. If the term is found in the database, the corresponding inverted files list which holds file pointers to the actual files that holds the looked up term(s) are located. In my case the inverted files list is the *lookupData* table while the *image* table are the file pointers that point to a certain image in a certain image collection.

3. When pointers are found documents/ images are located by following the located pointers. In my approach we would have to gather the *location* information from the *image* table collecting the image from where it is located, either on disk or on the internet.

We have addressed all entities from the E-R diagram including *UniqueTerm*'s three relations, but still we have two more relations within the design, i.e. the relation between (1) *collection* and *image* and the relation between (2) *collection* and *hypernym*. (1) is motivated simply since it seems natural to have the images close to the *collection* entity, for purposes such as gathering all images for a certain collection. (2) is motivated for more database design specific purposes. Consider the relation between a *uniqueTerm* and *hypernym.* This relation connects *user provided terms* to a higher level of representation on different levels and its purpose is to include hypernyms as representatives when several terms included within it together form frequencies above the *selection threshold*.

For this reason a term frequency for the hypernym has to be included. If this was included as an attribute in the relation between the two, each tuple for each term in the relation would have to include the hypernym TF, count and IDF which would leave the database redundant and include great update anomalies. So instead of including duplicate information for each entry in the relation between *uniqueTerm* and *hypernym* only one tuple for these variables are included for each hypernym at a certain level in a certain collection, i.e. in the relation between *collection* and *hypernym*. Each collection have exactly one entry for a specific hypernym on a specific level. As a result the relations from the uniqueTerm-hypernym and collection-hypernym can easily be mapped to each other if desired.

5.7.4. From E-R diagram to relational database schema

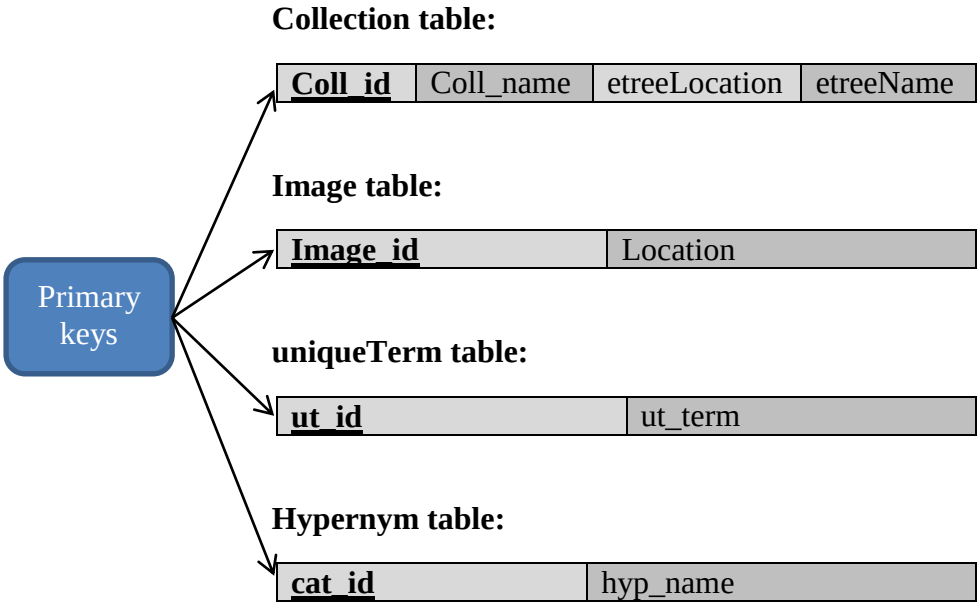
In mapping and translating the E-R diagram to a relational diagram my approach will be based on the theory from *section 2.4*. To summarize the following is of relevance and will be performed step by step:

1. Each strong entity, i.e. an entity that is not dependent on others to and can exist on its own, will be translated with the following criteria:
   a) Each entity will become a relation
   b) Each attribute of the entity will become an attribute of the relation
   c) One of the key attributes of the entity will become a primary key for the table
2. Each M:N relation will be translated with the following criteria:
   a) For each relationship type a table will be created
   b) The primary keys from all entities participating in the relation will be included as foreign key attributes of the table.

c) Also all attributes of the relation will become an attribute of the table.
d) The combination of the foreign keys from b. and, if any, key attributes from the relation will form the primary key of the table.

3. Each 1:N relation will be translated with the following criteria:
   a) The table of each N-side of the relation will be included with the 1-side's primary key as a foreign key attribute.
   b) Also all entity attributes on both side are included in the entities table, this has hopefully already been done in step 1.b.

From step 1 the following tables will be created, *collection*, *image*, *uniqueTerm* and *hypernym*. From step 1.b. all entity attributes will be inserted into the table created, also one of the key attributes will be selected as a primary key for the table from 1.c. above, since all entities in this case each only have one key attribute the selection are easy. The following tables are created.



Four of the five relations are a binary M:N relation see *figure 4.3.* Anyhow these relations will be translated using the procedure from step 2, i.e. create four new tables (2.a.) *includes*, *taggedWith*, *groupedInto* and *memberOf and contains*, all tables are included with all participating entity's primary keys as foreign keys (2.b.) all attributes of relations as attributes in table (2.c.) and primary key are formed by the combination of foreign keys from 1.b. and prospective entity attribute keys (2.d). The result are the following tables:

includes table:

collectionId	**uniqueTId**	**utType**	utCount	utTf	utIdf

taggedWith table:

uniqueTId	**imageId**

groupedInto table:

collectionId	**hypId**	hierLevel	hypTf	hypIdf

MemberOf table:

collectionId	**uniqueTId**	**hypId**	**hierLevel**

contains table:

collectionId	**ImageId**

Finnally step 3 is performed for the 1:N relation between the *collection* entity and the *image* entity. The primary key from the *collection* entity is included as a foreign key in the image table, the new table looks like:

Images table: (NEW)

imageId	Location	collectionId

The finished relational tables are the following:

Collection table:

Coll_id	Coll_name	etreeLocation	etreeName

Image table:

imageId	Location	**collectionId**

uniqueTerm table:

utId

hypernym table:

hyperName

includes table:

collId	**utId**	**utType**	utCount	utTf	utIdf

taggedWith table:

uniqueTId	**imageId**

groupedInto table:

collId	**hypId**	**hierLevel**	hypTf	hypIdf

MemberOf table:

collId	**utId**	**hypId**	**hierLevel**

contains table:

collectionId	**ImageId**

When building the relational schema the goal was to reach the Boyce- Codd Normal Form, see description of the different normal forms in section 2.4.3. The finished relational schema is in Boyce- Codd normal form. That is for the relational schema 1NF, 2NF and BCNF holds.

5.8. XLWT Library

The data obtained by the summarizing system are stored into a MySQL database as described in the previous section. The design of the database is presented in section 5.7. In the next chapter the summarized information is presented in MS excel diagrams. For simplified purpose the summarizing system stores the data from the collection summary into an excel file at the execution path that more easily can be used to create the tables when presenting them in the next chapter. The package used to create the excel files are XLWT version 0.7.2 usable for python version 2.3.*- 2.6.*. XLWT is a library for creating spread sheet files in .xls format. The implementation for this functionality is found in *appendix B-9.*

5.9. Other

Other implemented functionality not discussed in this chapter are *cleasning of metadata (*appendix B-3*), converting of metadata (*appendix B-4*), unique term handler (*appendix B-5*), calculation of term frequencies (*appendix B-7*)* and *finding representatives for the collection summary (appendix B-8).*

Chapter 6

6. Results and Evaluation

In the evaluation of the system developed in this thesis image collections gathered from Flickr in the form of Flickr groups are used. In section 6.1 I will introduce how the experiments for the evaluation of the system are performed, in section 6.2 the results of the experiments performed are presented and compared against the test user's data. The evaluation of the system versus the experiment along with a discussion is finally presented in section 6.3.

6.1. Experiment

In this project the implementation of the system developed has focused on 17 different Flickr groups with 300 – 6000 geo- referenced images. Each group with all its images included are viewed by the system as predefined collection as images. All images within a collection are processed, tags and title manually added by users on Flickr are collected, geo- location data and time/ date stamps are used to auto- annotate the images with additional contextual metadata. Further data are converted, cleansed before representatives for the collection is selected weighted on the metadata's frequency within the collection (term frequency).

For the experiments performed only the Term Frequency is used in weighing the terms within the collection, mainly because the experiments are viewed as independent. This means that a specific collection has no relation with the other image collections; hence the Inverse document frequency is not necessary to include.

The system developed creates collections summaries for image collections that reflect the representative terms of the images within it. The collection summaries can be used for two purposes. The first, describing the image collections through a textual document represented by the located representatives. Secondly, used to efficiently filtering out irrelevant collections in the process of image retrieval and locating only those that are. Since the system developed in this project present functionality for the production of the collection summaries and not on the actual image retrieval itself, the latter are difficult to evaluate at this stage.

Anyhow the augmenting of contextual metadata is obviously useful when requesting images using context related search queries such as, return images- "from France", "taken on a Saturday 2010", "of Hurricanes in USA", "taken in the summer when the weather condition is clear" and so on. In private collections search queries that would be more relevant for individual users are contexts that are known to the individual user. E.g. say that the user wants to locate all images taken in a specific summer vacation, of which he knows the year, season and location, e.g. return images "taken in summer in Paris 2009".

Also if a retrieval system had been developed a typical test could be having one system that views all images as one search space, and comparing this system to mine. In the comparison system a query Q would be on all set of images S. In my system the images would be separate the search space S into 17 number of image collections, filtering out those not relevant for search query Q using the collection summaries. Using this approach the effectiveness in case of runtime would be much better, and hopefully also the returned results would be better. Nevertheless at this point this claim is difficult to confirm or decline.

Other than augmenting the images within the collections with additional metadata, the system has also focused on reflecting the natural perception of users by taking advantage of the user provided terms by individual users on Flickr. The human perception of images is subjective, i.e. that different users have different perceptions on the same images for different purposes discussed earlier. The goal of the system developed is to find representatives for the collections which describes the collection from its most informative view, i.e. terms that stand out from other terms and reflects the collection in the best manner as a whole. Using manually added tags added by different users on Flickr this also involves finding properties that reflects the focal point of these users' natural perceptions of the images.

From these motives I find it most useful to evaluate the system up against individual's natural perception, focusing on the *user provided terms* in the collection summary. The experiment is performed by carrying out tests with external users on *appropriate image collections* used in the development of the system. With *appropriate image collections* I mean using small image collections in the experiment, consisting of 300-500 images. These sizes seem appropriate to prevent users from having problems remembering the majority of the images viewed when deducting the most representative properties of them.

From the 17 collections experimented on in the implementation I found 7 collections to be good candidates for the evaluation of the system. The users of the experiment are assigned to browse through all images in the image collections and write down keywords that they feel are most descriptive for the collection as a whole. Since all images used in the experiments are available on Flickr's online pages, the experiments are performed there. The images in the used collection are on Flickr divided over several pages holding 30 images on each page. The image collections used in the evaluation of the system are listed in appendix X. The experiment is performed in the following manner:

- The test users are assigned to browse through the images, using approximately a second on each image before moving on to the next page.
- The test users are encouraged to prevent to get caught up in any single images.
- When all images has been browsed through, the users are assigned to write down keywords that they feel best describes their perception of the images in the collection.

- When keywords are written down for the collection, the image collection is again browsed through by the test user allowing them to adjust the keywords to best reflect the user's individual perception

Notably the test users has been given very limited instructions, examples or advices on features to look for, preventing myself from influencing and constructing the experiment at a minimum, causing the results to be as subjective as possible.

When the keywords reflecting the individual test user's perception of the collection are collected, the collection summary is compared against the results from the user experiments. This is done manually in the manner of looking for direct hits or terms that hold similarities between the two representations. I will present and discuss the results in the preceding section.

6.2. Results

The experiment has been performed using three test users. The test users will be referred to as *user 1*, *user 2* and *user 3* as their results are presented. Along with the results from the test experiments the produced output of the summarizing system will also be presented, i.e. properties that are viewed by the system as most representative for the image collections used in the experiment. The latter will be referred to as the *collection summary*.

Direct hits between the collection summary and the test user's keywords will be represented as green keywords, similar or relevant properties between the collection summary and the user keywords will be represented by blue and keywords with no obvious relation to the collection summary will be represented by red. Green, i.e. direct hits are terms in the collection summary that are exact matches to keywords provided by the users. Notably a synonym word is also viewed as a direct hit. Blue, i.e. relevant, are keywords that are not physically present in the collection summary, but are viewed as relevant since many similarities or relevant terms are highly weighted in the collection summary. Red represents keywords that are not located in the collection summary and which have no obvious relation between the two.

Seven experiments performed are presented in the rest of this sub section. Each experiment is represented by its collection summery along with the keywords provided by the test users.

A

Test collection 1 : Geo Tagged – Malaysia Number of images: 313

User 1		User 2		User 3	
Keyword	**Description**	**Keyword**	**Description**	**Keyword**	**Description**
1.1. Ant	No, but insects and bugs	2.1. Bugs	Yes, under insects	3.1. Sky	no, but panorama and clouds
1.2. Bug	Yes	2.2. Nature - Trees	No, but park and green high ranked	3.2. Lizards	No
1.3. Butterfly	Yes butterfly and moth	2.3. Sunset	Yes, under tags	3.3. Skyscrapers	Yes, under building
1.4. Nature	No, but many related Sky, green, insects, animals etc	2.4. Beach	Yes, under tags	3.4. Insects	Yes, category, many members
1.5. Hot	Yes, high temperature	2.5. Tower	Yes, under structure	3.5. Birds	No
1.6. Beach	Yes	2.6. Asia	Yes, tags	3.6. Sun	Yes, both sunset and sunrise high ranked
1.7. Summer	Yes, highest (season)			3.7. Trees	No, but park and green high ranked
1.8. Sky	No, but panorama and cloud			3.8. Roads	No
1.9. Ocean	Yes, synonym sea			3.9. Clouds	Yes, weather cond. and tags
1.10. City	No, but building', sky- scraper, architecture, hotel			3.10. Sea	Yes, high ranked
1.11. Park	Yes, (category level 2)			**3.11. Nature**	No, but much tags relevant. Such as waterfall, park, green , animals, insects, wind, storm,
1.12. Boat	No				
1.13. Tower	Yes, (category - structure)				
1.14. Bridge	Yes, (category - structure)				

B

Type	Data	Term freq	Count	Type	Data	Term freq	Count
location	country:malaysia	1,0000	313	tags	Penang	0,1757	55
location	region:pulau pinang	0,3355	105	Tags	penangflickrgroup	0,1725	54
location	region:johor	0,1374	43	Tags	**Asia** (2.6)	0,1470	46
location	street:jalan teluk bahang	0,1086	34	Tags	**Asie** (2.6)	0,1438	45
location	city:teluk bahang	0,1086	34	Tags	Malaisie	0,1438	45
location	region:kedah	0,0990	31	Tags	Johor	0,1182	37
location	city:butterworth	0,0863	27	Tags	penangbutterflyfarm	0,0990	31
location	city:kuala lumpur	0,0831	26	Tags	kualalumpur	0,0958	30
location	city:johor bahru	0,0831	26	Tags	Island	0,0863	27
location	region:pahang	0,0767	24	Tags	**Beach** (1.6) (2.4)	0,0831	26
location	city:george town	0,0735	23	Tags	**Sea** (1.9) (3.10)	0,0767	24
location	street:jalan chain ferry	0,0639	20	Tags	**Green** (2.2) (3.7)	0,0767	24
location	region: kuala lumpur	0,0575	18	Tags	Pulaupinang	0,0767	24
weekday	Saturday	0,2939	92	Tags	Tanjung	0,0703	22
month	Oktober	0,2652	83	Tags	Heritage	0,0639	20
month	July	0,2364	74	Tags	**Sunset (2.3) (3.6)**	0,0639	20
month	November	0,1885	59	Tags	Unesco	0,0607	19
season	**Summer (1.7)**	0,4089	128	Tags	Worldheritage	0,0607	19
season	Fall	0,3738	117	Tags	Klcc	0,0607	19
dateHuman	2008 july	0,2364	74	Tags	Kedah	0,0607	19
dateHuman	2008 oktober	0,1757	55	Tags	Hijau	0,0607	19
dateHuman	2008 november	0,1693	53	Tags	**Panorama** (1.8) (3.1)	0,0575	18
dateHuman	2008 june	0,0927	29	Tags	Pulaumutiara	0,0543	17
dateHuman	2007 oktober	0,0831	26	Tags	Bokeh	0,0511	16
dateHuman	2008 may	0,0607	19	weather-cond	**mostly cloudy** (1.8) (3.1) (3.9)	0,8435	264
tags	outdoorgraphy™	0,6837	214	weather-wind	light air	0,3706	116
tags	Sirmart	0,3898	122	weather-wind	light breeze	0,3610	113
tags	Outdoorgraphy	0,3802	119	weather-wind	gentle breeze	0,1629	51
tags	Macro	0,1917	60	weather-temp	**very hot** (1.5)	0,8179	256
tags	Penangflickr	0,1789	56	weather-temp	**Hot** (1.5)	0,1629	51

C

Type	Level	Hypernym name	Term freq	Count	Hypernym members
hyp-tags	1	chromatic_color	0,1597	50	green, blue, red, yellow, pink,
hyp-tags	1	Digit	0,1342	42	1, 2, ii, 3, iii, 4, 5, 6, 7, eight, 8,
hyp-tags	1	Hour	0,0927	29	**sunset** (3.6), **sunrise** (3.6), noon,
hyp-tags	1	Land	0,0895	28	island, turf,
hyp-tags	1	body_of_water	0,0799	25	sea, waterfall,
hyp-tags	1	**Structure** (1.10)	0,0703	22	**tower** (1.13) (2.5), building, **bridge** (1.14), signboard, fountain, stadium,
hyp-tags	1	Person	0,0543	17	white, scorpion, tiger, jumper, have, fighter, travellers,
hyp-tags	1	Tract	0,0511	16	field, **park** (1.11) (2.2) (3.7), common,
hyp-tags	1	time_period	0,0511	16	night, morning, stage, daytime,
hyp-tags	2	**insect (1.1) (3.4)**	0,1150	36	**Butterfly** (1.3), **moth** (1.3), **insect** (1.1), **bugs** (1.1) (1.2) (2.1), **bug** (1.1) (1.2) (2.1), beetle, pupa,
hyp-tags	2	Arthropod	0,0671	21	bugs, bug, beetle, pupa, insect,
hyp-tags	2	Building	0,0799	25	temple, masjid, mosque, building, **skyscraper (3.1)**, architecture, house, hotel,
hyp-tags	2	atmospheric_phenomenon	0,0511	16	wind, storm,
hyp-tags	2	physical_phenomenon	0,0575	18	storm, **cloud** (1.8) (3.1) (3.9), **clouds** (1.8) (3.1) (3.9),
hyp-tags	2	geological_formation	0,1022	32	plage, beach,

Table 6-1: Experiment 1- Table A shows keyword results collected from test users, B shows the collection summary's tags and auto annotations and C shows the categorizations of hypernyms from the collection summary

A

Test collection 2 : Ruins in Digital - Geo-tagged				Number of images: 274	
User 1		**User 2**		**User 3**	
Keyword	**Description**	**Keyword**	**Description**	**Keyword**	**Description**
1.1. Castle	Yes, 2nd highest title	2.1. Rome	No	3.1. Plains	No
1.2. Ruins	Yes, 2nd highest tags	2.2. Italy	No	3.2. Ruins	Yes, high ranked
1.3. Ancient	No, but history ,old (cats)	2.3. Old houses	Yes, old and houses (also structures, ruins etc)	3.3. Old houses	Yes, old, ancient and houses / buildings /structures high ranked
1.4. Old	Yes, cats	2.4. Ruins	Yes	3.4. Structures	Yes, high ranked
1.5. Building	Yes (cats, 2nd level)	2.5. Architecture	Yes, under building	3.5. Shadows	No
1.6. Construction	Yes, structure	2.6. Roman	No	3.6. Sky	Yes
1.7. Tower	Yes, 2nd level cats	2.7. Street	No, but urban	3.7. Street	No, but urban
1.8. Palace	Yes, 1st level cats	2.8. Castles	Yes, tags	3.8. Grass	No
1.9. Broken	No, but ruin, devastation	2.9. Old	Yes	3.9. Ceiling	No, but very many structures with ceiling i.e. tower, building stadium, monument, church, castle, palace, cathedral, abbey, fortress, donjon

B

Type	Data	Term freq	Count	Type	Data	Term freq	Count
location	country:uk	0,3102	85	tags	friche	0,2701	74
location	country:france	0,2774	76	Tags	**Castle** (1.1) (2.8)	0,2080	57
location	region:nord-pas-de-calais	0,2701	74	Tags	**Ruins** (1.2) (2.4) (3.2)	0,1861	51
location	street:9 avenue marc sangnier	0,1861	51	Tags	Textile	0,1861	51
location	city:59280 armentières	0,1861	51	Tags	Armentieres	0,1861	51
location	country:usa	0,1460	40	Tags	England	0,1058	29
location	street:33b rue philippe lebon	0,0839	23	Tags	2008	0,0876	24
location	city:59100 roubaix	0,0839	23	Tags	Roubaix	0,0839	23
location	country:sweden	0,0730	20	Tags	Northwales	0,0766	21
location	city:flint	0,0693	19	Tags	**Sky** (3.6)	0,0693	19
location	region:sc	0,0657	18	Tags	Sweden	0,0693	19
location	street:castle dyke st	0,0511	14	Tags	**Old** (1.4) (2.3) (2.9) (3.3)	0,0657	18
weekday	saturday	0,4161	114	Tags	Church	0,0620	17
month	november	0,2482	68	Tags	Abbey	0,0584	16
month	desember	0,1168	32	Tags	Flint	0,0547	15
dateHuman	1999 november	0,1861	51	Tags	Hdr	0,0511	14
dateHuman	1999 desember	0,0839	23	Tags	västragötaland	0,0511	14
dateHuman	2009 april	0,0803	22	Tags	västergötland	0,0511	14
dateHuman	2010 january	0,0620	17	weather-cond	clear	0,3175	87
dateHuman	2008 may	0,0511	14	weather-cond	scattered clouds	0,1934	53
tags	winter	0,4818	132	weather-cond	mostly cloudy	0,1898	52
tags	abandoned	0,3248	89	weather-cond	partly cloudy	0,1825	50
tags	**Urban** (2.7) (3.7)	0,2847	78	weather-wind	gentle breeze	0,3066	84
tags	Ruin	0,2810	77	weather-wind	moderate breeze	0,1679	46
tags	Urbex	0,2810	77	weather-wind	light breeze	0,1569	43
tags	abandonné	0,2810	77	weather-temp	chilly	0,4745	130
tags	Usine	0,2701	74	weather-temp	moderate	0,3175	87
tags	nord	0,2701	74				

C

Type	Level	Hypernym name	Term freq	Count	Hypernym members
hyp-tags	1	season	0,50364964	138	winter, christmas, yuletide, autumn,
hyp-tags	1	**Devastation** (1.9)	0,46715328	128	**ruin** (1.9), ruins,
hyp-tags	1	**Mansion** (1.8)	0,21167883	58	**castle** (1.1) (2.8) , palace (1.8),
hyp-tags	1	**Past** (1.3)	0,10948905	30	**old** (1.3) , **history** (1.3),
hyp-tags	1	digit	0,06934307	19	1, 3, 4, 2, 5, 8, 6, 9,
hyp-tags	1	**Church** (3.9)	0,06569343	18	Abbey , cathedral,
hyp-tags	1	**structure** (1.5) (1.6) (3.4)	0,0620438	17	**tower** (1.7) , **building** (1.5) , **buildings** (1.5), stadium, monument,
hyp-tags	2	residence	0,06569343	18	monastery, priory, home,
hyp-tags	2	Creation	0,05109489	14	excavation, classic,
hyp-tags	2	defensive_structure	0,06934307	19	donjon, dungeon, dungeons, fortress, fortification, fortifications,
hyp-tags	2	Building	0,08759124	24	Chapel , temple, **house** (2.3) (3.3) , **architecture** (2.5), building,
hyp-tags	2	artifact	0,2044	56	bricks, brick, textile,
hyp-tags	2	atmosphere	0,0730	20	low, sky,
hyp-tags	3	vascular_plant	0,0620	17	tree, trees, herb, refinery, mill,

Table 6-2: Experiment 2- Table A shows keyword results collected from test users, B shows the collection summary's tags and auto annotations and C shows the categorizations of hypernyms from the collection summary

A

Test collection 3 : Taken FROM a bridge — Number of images: 279

User 1		User 2		User 3	
Keyword	**Description**	**Keyword**	**Description**	**Keyword**	**Description**
1.1. Bridge	Yes, in tags and under *structure*	2.1. London eye	London yes, eye no	3.1. Bridge	Yes, under *structure*
1.2. Water	Yes, in tags.	2.2. Rivers	Yes, tags and under *stream*	3.2. Canal	No, but water, stream and river
1.3. Ocean	Yes, under *body of water*	2.3. Bridges	Yes, in tags and under *structure*	3.3. Boat	Yes, tags, under *boat* and *vessel*
1.4. Blue	Yes, under *achromatic color*	2.4. Water	Yes, in tags.	3.4. Sea	Yes, under body of water
1.5. Construction	Yes, synonym *structure*	2.5. Boats	Yes, tags, under *boat* and *vessel*	3.5. Train	Yes under *public transport*
1.6. Railroad	Yes, under *line*	2.6. Railroad	Yes, under *line*	3.6. Rails	Yes, synonym to rail-road/-ways
1.7. Tower	Yes, under *structure*	2.7. Rails	Yes, synonym to rail-road/-ways	3.7. Dam	No

B

Type	Data	Term freq	Count	Type	Data	Term freq	Count
location	country:usa	0,3297	92	tags	**Viewonabridge** (1.1) (2.3) (3.1)	0,0968	27
location	country:uk	0,2796	78	tags	barco	0,0932	26
location	region:	0,2545	71	tags	barcos	0,0932	26
location	country:sweden	0,1720	48	tags	lake	0,0717	20
location	city:stockholm county	0,0932	26	tags	**onthebridgec** (1.1) (2.3) (3.1)	0,0681	19
location	**region:greater london** (2.1)	0,0502	14	tags	view	0,0609	17
location	street:västerbron	0,0502	14	tags	people	0,0573	16
weekday	sunday	0,2258	63	tags	europe	0,0573	16
weekday	saturday	0,2151	60	tags	mälaren	0,0538	15
month	april	0,2652	74	tags	**boats** (2.5)	0,0538	15
month	mars	0,1290	36	tags	**London** (2.1)	0,0502	14
season	spring	0,4194	117	tags	sunset	0,0502	14
dateHuman	2011 april	0,2079	58	tags	travel	0,0502	14
dateHuman	2011 mars	0,0968	27	tags	reflection	0,0502	14
tags	**Bridge** (1.1) (2.3) (3.1)	0,4265	119	tags	malaren	0,0502	14
tags	**River** (3.2)	0,1971	55	weather-cond	clear	0,3763	105
tags	**Water** (1.2) (2.4) (3.2)	0,1649	46	weather-cond	scattered clouds	0,1326	37
tags	stockholm	0,1362	38	weather-cond	mostly cloudy	0,1290	36
tags	sweden	0,1326	37	weather-cond	partly cloudy	0,0968	27
tags	**takenfromabridge** (1.1) (2.3) (3.1)	0,1147	32	weather-cond	overcast	0,0609	17
tags	scandinavia	0,1147	32	weather-wind	gentle breeze	0,2867	80
tags	sthm	0,1111	31	weather-wind	light breeze	0,2186	61
tags	stkhm	0,1111	31	weather-wind	light air	0,1756	49
tags	**Boat** (2.5) (3.3)	0,1039	29	weather-wind	moderate breeze	0,1183	33
tags	**viewfromabridge** (1.1) (2.3) (3.1)	0,0968	27	weather-temp	moderate	0,4444	124
tags	**Takenonabridge** (1.1) (2.3) (3.1)	0,0968	27	weather-temp	chilly	0,2832	79

C

Type	Level	Hypernym name	Term freq	Count	Hypernym members
hyp-tags	1	**Structure** (1.5)	0,4695	131	**Bridge** (1.1) (2.3) (3.1), **tower** (1.7), buildings, pattern, towers, patterns, building,
hyp-tags	1	**Stream** (3.2)	0,2401	67	**River** (2.2) (3.2), **rivers** (2.2) (3.2), creek, brook,
hyp-tags	1	vessel	0,1649	46	**Boat** (2.5) (3.3), **boats** (2.5) (3.3), ship,
hyp-tags	1	body_of_water	0,1577	44	lake, **sea** (3.4), stream, waterfall, waterfalls, **ocean** (1.3), bay, lakes, streams,
hyp-tags	1	line	0,1147	32	**railway** (2.7) (3.6), **railroad** (1.6, 2.6, 3.6) (2.7) , tracks, track, curve, ropes, row, route, **railways** (2.7) (3.6) , rope, watermark, curves,
hyp-tags	1	hour	0,0896	25	sunset, dusk, sunrise, twilight, dawn, crepuscule,
hyp-tags	1	consideration	0,0789	22	reflection, reflections,
hyp-tags	1	painting	0,0717	20	waterscape, waterscapes,
hyp-tags	1	orientation	0,0681	19	view, perspective,
hyp-tags	1	time_period	0,0645	18	morning, night, week, stage,
hyp-tags	1	digit	0,0645	18	1, 3, one, three, 2, two, ii, trinity, 7, 5, iii,
hyp-tags	1	person	0,0609	17	amateur, national, straight, white, victorian, worker, have,
hyp-tags	1	**Boat** (2.5)	0,0609	17	ferry, kayaking, paddling, **motorboat** (2.5) (3.3), **powerboats** (2.5) (3.3), **powerboat** (2.5) (3.3), **motorboats** (2.5) (3.3), **tugboats** (2.5) (3.3),
hyp-tags	1	motion	0,0573	16	travel, posing, speeding,
hyp-tags	1	sailing_vessel	0,0573	16	**Sailboat** (2.5) (3.3), **sailboats** (2.5) (3.3), sloop, barque,
hyp-tags	1	chromatic_color	0,0502	14	green, yellow, **blue** (1.4), red, purple,
hyp-tags	2	object	0,0896	25	island, ness, islands, archipelago, plain, location,
hyp-tags	2	geological_formation	0,0538	15	mountains, mountain, beach, shore, foreshore,
hyp-tags	2	platform	0,0538	15	quay, pier, deck,
hyp-tags	2	travel	0,0717	20	near, travel, trekking, commuting, cruising,
hyp-tags	2	movement	0,0538	15	ripples, waves, approaching, wave,
hyp-tags	2	people	0,0681	19	british, irish, people, public,
hyp-tags	2	location	0,0538	15	southland, top, location,
hyp-tags	2	group	0,0645	18	traffic, pile, people,
hyp-tags	2	building	0,0502	14	hospital, architecture, restaurant, house, skyscrapers, hotel, building, houses,
hyp-tags	2	liquid	0,1685	47	chocolate, water,
hyp-tags	3	artifact	0,0502	14	photo, picture, image, photograph, classic, art, surface,
hyp-tags	3	restraint	0,0502	14	bow, lock, locks, floodgate,
hyp-tags	3	public_transport	0,0502	14	commuters, **train** (3.5), **trains** (3.5), bus,

Table 6-3: Experiment 3- Table A shows keyword results collected from test users, B shows the collection summary's tags and auto annotations and C shows the categorizations of hypernyms from the collection summary

A

Test collection 4 : NYC Chinatown Number of images: 380

User 1		User 2		User 3	
Keywords	**Description**	**Keyword**	**Description**	**Keyword**	**Description**
1.1. Asia	Yes, high ranked tag	2.1. Chinatown	Yes, highest ranked	3.1. Dragons	No
1.2. Chinatown	Yes, very high rank tag	2.2. Chinese	Yes, high ranked	3.2. Lamps	No
1.3. culture	No,	2.3. Chinese new year	Yes, tags. Chinesenewyear high ranked	3.3. Skyscrapers	No, but towers, buildings etc
1.4. City	Yes, city, metropolis, municipality	2.4. People	Yes, under category person. Also inhabitant and person of color	3.4. Store windows	Yes, storefront under side
1.5. Asian/ Chinese people	Yes, both very high ranked, also asiatic	2.5. Black and white photos	No,	3.5. Fish	No, but seafood
1.6. People	Yes, under category person. Also inhabitant and person of color	2.6. Chinese Marked	Yes, Chinese and marked, also shopping, buying etc	3.6. Chinese	Yes, high ranked
				3.7. Plastic bags	Yes, high ranked tags. Bags, container
				3.8. People	Yes, person (category)
				3.9. statues	No, but characters, imaginary_being in category

B

Type	Data	Term freq	Count	Type	Data	Term freq	Count
location	country:usa	1,0000	380	tags	urbanexploration	0,0895	34
location	region:ny	0,9974	379	tags	curbed	0,0895	34
location	city:manhattan	0,6553	249	tags	urbanphotography	0,0895	34
location	city:new york	0,3447	131	tags	cityphotography	0,0895	34
location	street:manhattan bridge	0,1105	42	tags	chinatownnewyorkcity	0,0895	34
weekday	saturday	0,2368	90	tags	streets	0,0868	33
month	september	0,1474	56	tags	red	0,0816	31
month	february	0,1421	54	tags	lowereastside	0,0816	31
month	august	0,1211	46	tags	building	0,0763	29
dateHuman	2009 september	0,1289	49	tags	sidewalk	0,0763	29
dateHuman	2009 august	0,0947	36	tags	little	0,0737	28
dateHuman	2010 oktober	0,0895	34	tags	**characters** (3.9)	0,0684	26
dateHuman	2010 april	0,0684	26	tags	**Bags** (3.7)	0,0658	25
dateHuman	2011 february	0,0526	20	tags	usa	0,0632	24
dateHuman	2010 february	0,0500	19	tags	buildings	0,0632	24
tags	**chinatown** (1.2) (2.1)	0,9789	372	tags	architecture	0,0605	23
tags	nyc	0,7763	295	tags	rooftops	0,0605	23
tags	manhattan	0,6211	236	tags	manhattanbridge	0,0605	23
tags	new	0,5474	208	tags	graffiti	0,0605	23
tags	york	0,5395	205	tags	young	0,0579	22
tags	gothamist	0,5316	202	tags	view	0,0553	21
tags	**City** (1.4)	0,5132	195	tags	skyline	0,0553	21
tags	street	0,4316	164	tags	manhattanskyline	0,0553	21
tags	**chinese** (1.5) (2.2) (2.6) (3.6)	0,4316	164	tags	newyorkcityarchitecture	0,0553	21
tags	**asian** (1.5)	0,3553	135	tags	manhattanarchitecture	0,0553	21
tags	ny	0,3263	124	tags	manhattanbridgeview	0,0553	21
tags	urban	0,2711	103	tags	newyorkcitybuildings	0,0553	21
tags	newyorkcity	0,2658	101	tags	chinatownrooftops	0,0553	21
tags	newyork	0,2579	98	tags	manhattanbridgeviews	0,0553	21
tags	les	0,2342	89	tags	back	0,0553	21
tags	east	0,1921	73	tags	**chinesenewyear** (2.3)	0,0526	20
tags	candid	0,1895	72	tags	white	0,0500	19
tags	china	0,1632	62	tags	**shopping** (2.6)	0,0500	19
tags	side	0,1500	57	tags	fuzhou	0,0500	19
tags	lower	0,1500	57	tags	pretty	0,0500	19
tags	bridge	0,1342	51	weather-cond	clear	0,6368	242
tags	**asia** (1.1)	0,1316	50	weather-cond	overcast	0,1395	53
tags	broadway	0,1105	42	weather-cond	partly cloudy	0,0789	30
tags	chinatownnyc	0,1079	41	weather-wind	light breeze	0,3737	142
tags	man	0,0974	37	weather-wind	gentle breeze	0,3132	119
tags	woman	0,0947	36	weather-temp	chilly	0,3289	125
tags	lowermanhattan	0,0921	35	weather-temp	moderate	0,1895	72
tags	wnyc	0,0921	35	weather-temp	hot	0,1895	72
tags	girl	0,0895	34	weather-temp	very hot	0,1842	70

C

Type	Level	Hypernym name	Term freq	Count	Hypernym members
hyp-tags	1	dynasty	0,5421	206	york, qings,
hyp-tags	1	**municipality** (1.4)	0,5421	206	**City** (1.4), **metropolis** (1.4), **town** (1.4)
hyp-tags	1	thoroughfare	0,5184	197	street, streets,
hyp-tags	1	**inhabitant** (1.6) (2.4)	0,3605	137	asian, asiatic, american,
hyp-tags	1	**person_of_color** (1.6) (2.4)	0,3579	136	asian, asiatic,
hyp-tags	1	structure	0,2921	111	bridge, **building** (3.3), **buildings** (3.3), rete, pattern, housing, fountain, cross, balcony, **towers** (3.3),
hyp-tags	1	adult	0,2026	77	man, woman, women,
hyp-tags	1	region	0,1553	59	side, outside, district,
hyp-tags	1	chromatic_color	0,1526	58	red, yellow, pink, blue, green, purple,
hyp-tags	1	woman	0,1368	52	girl, lady, vamp, ladies, girls,
hyp-tags	1	male	0,1158	44	man, boy,
hyp-tags	1	**container** (3.7)	0,1105	42	**Bags** (3.7), **bag** (3.7), basket, box, cup, can, purse,
hyp-tags	1	female	0,1053	40	woman, women,
hyp-tags	1	digit	0,1053	40	2, 1, 4, 3, two, 7, one, 6, 5, 9, triad, three, 8, four,
hyp-tags	1	building	0,1026	39	architecture, restaurant, eatery, clubhouse,
hyp-tags	1	walk	0,1000	38	sidewalk, mall, marching,
hyp-tags	1	juvenile	0,1000	38	kid, preteen, teen, kids, child, teenager, children, teenagers,
hyp-tags	1	top	0,0842	32	rooftops, rooftop,
hyp-tags	1	**imaginary_being** (3.9)	0,0737	28	characters, character,
hyp-tags	1	body_part	0,0684	26	back, shoulder, small, shoulders,
hyp-tags	1	activity	0,0658	25	**Market** (2.6), games, work, solo, game, use, help,
hyp-tags	1	animal	0,0632	24	young, giant,
hyp-tags	1	mercantile_establishment	0,0632	24	shop, store,
hyp-tags	1	time_period	0,0632	24	night, year, festival, times, nap, morning,
hyp-tags	1	**person** (2.4) (3.8)	0,0605	23	white, worker, blonde, ethnic,
hyp-tags	1	happening	0,0526	20	fire, break,
hyp-tags	2	artifact	0,0789	30	graffiti, structure, toy, decoration, fabric,
hyp-tags	2	purchase	0,0526	20	**shopping** (2.6), buy,
hyp-tags	2	creation	0,0526	20	film, art, classic,
hyp-tags	2	motion	0,0816	31	crossing, sitting, travel, rush,
hyp-tags	2	time	0,0500	19	old, time, future,
hyp-tags	2	act	0,0500	19	waiting, going, getting, activity,
hyp-tags	2	food	0,0605	23	noodles, noodle, food, meat, produce, **seafood** (3.5),
hyp-tags	2	travel	0,0632	24	trip, ride, outing, crossing, travel,
hyp-tags	2	side	0,1684	64	**storefront** (3.4), facade, side, backside, front,
hyp-tags	2	decoration	0,0684	26	jewelry, graffiti, decoration,
hyp-tags	2	young	0,0605	23	piggy, young,
hyp-tags	2	sinitic	0,4342	165	cantonese, chinese,
hyp-tags	3	atmospheric_phenomenon	0,0579	22	snow, wind, storm,
hyp-tags	3	action	0,0500	19	sitting, travel, rush, warm, preparing, taking,
hyp-tags	3	garment	0,0632	24	jacket, coat, shirt, sweater, robes, shirts, laundry,
hyp-tags	3	substance	0,0526	20	trash, garbage, paper, packing, waste, food, stuff,

Table 6-4: Experiment 4- Table A shows keyword results collected from test users, B shows the collection summary's tags and auto annotations and C shows the categorizations of hypernyms from the collection summary

A

Test collection 5 : Geotagged: Delaware Number of images: 461

User 1		User 2		User 3	
Keywords	**Description**	**Keyword**	**Description**	**Keyword**	**Description**
1.1. Cops	Yes, cop, cops, police	2.1. Fire truck	Yes, both fire and truck	3.1. Boat	Yes, boat, sailboat, ship,
1.2. Fire Station	Yes, both fire and station high ranked, individual	2.2. USA	Yes, highest ranked location (country)	3.2. Police car	Yes, both police and car high ranked
1.3. Fire truck	Yes, trucks, fire	2.3. Birds	Yes, under category vertebrate (fowl, bird, wildfowl)	3.3. Fire truck	Yes, fire and truck high ranked
1.4. Car	Yes, high ranked	2.4. Dirt road	Yes, both dirt and road, high ranked under category	3.4. Creeks	Yes, under stream
1.5. Nature	No, but, high ranked, tree, bush, ocean, beach, bay, water, creek, river, park	2.5. Park	Yes, high ranked tags	3.5. Sea	Yes, sea, ocean, lake high ranked
1.6. Bird	Yes, fowl, bird, wildfowl	2.6. Military vehicle	Yes, under force and vehicle	3.6. Bird	Yes, bird, fowl, and wildfowl
1.7. Animal	Yes			3.7. Lighthouse	Yes, in tags
				3.8. Trees	Yes, in tags and under woody plant
				3.9. Train cart	Yes, train under instrumentality

B

Type	Data	Term freq	Count	Type	Data	Term freq	Count
location	**country:usa** (2.2)	1,0000	461	tags	rogers	0,0803	37
location	region:de	0,9436	435	tags	bethany	0,0759	35
location	city:wilmington	0,2299	106	tags	bethanybeach	0,0738	34
location	city:bethany beach	0,0933	43	tags	**Ocean** (1.5) (3.5)	0,0716	33
location	city:dover	0,0824	38	tags	lancerogers	0,0716	33
location	street:coastal hwy	0,0759	35	tags	brandywine	0,0694	32
location	city:lewes	0,0738	34	tags	**water** (1.5)	0,0607	28
location	city:rehoboth beach	0,0586	27	tags	2011	0,0607	28
location	city:newark	0,0542	25	tags	Sky	0,0607	28
weekday	saturday	0,3579	165	tags	**trees** (1.5) (3.8)	0,0586	27
weekday	sunday	0,2560	118	tags	**Fire** (1.2) (1.3) (2.1) (3.3)	0,0564	26
month	july	0,1497	69	tags	sussexcounty	0,0564	26
month	mars	0,1323	61	tags	**police** (1.1) (3.2)	0,0564	26
season	summer	0,3774	174	tags	fenwick	0,0564	26
dateHuman	2010 july	0,1215	56	tags	photoshopelements	0,0564	26
dateHuman	2011 mars	0,1041	48	tags	**Station** (1.2)	0,0542	25
dateHuman	2010 august	0,0976	45	tags	**lighthouse** (3.7)	0,0542	25
dateHuman	2011 april	0,0889	41	tags	wilmingtondelaware	0,0521	24
dateHuman	2011 february	0,0716	33	tags	Island	0,0521	24
dateHuman	2010 june	0,0586	27	weather-cond	clear	0,6443	297
dateHuman	2010 oktober	0,0521	24	weather-cond	mostly cloudy	0,1215	56
tags	delaware	0,6421	296	weather-cond	overcast	0,0803	37
tags	sussexcountyde	0,2039	94	weather-cond	scattered clouds	0,0738	34
tags	beach	0,1735	80	weather-wind	gentle breeze	0,3102	143
tags	delawareonline	0,1605	74	weather-wind	moderate breeze	0,2603	120
tags	wilmington	0,1280	59	weather-wind	light breeze	0,2495	115
tags	**Park** (1.5) (2.5)	0,1215	56	weather-temp	very hot	0,3471	160
tags	resort	0,0889	41	weather-temp	moderate	0,2169	100
tags	state	0,0868	40	weather-temp	chilly	0,2017	93
tags	lance	0,0803	37	weather-temp	hot	0,1605	74

C

Type	Level	Hypernym name	Term freq	Count	Hypernym members
hyp-tags	1	geological_formation	0,1974	91	**beach** (1.5), shore, oceanfront,
hyp-tags	1	body_of_water	0,1714	79	**ocean** (1.5) (3.5), bay, inlet, **sea** (3.5), stream, **lake** (3.5), waterfalls, waterway,
hyp-tags	1	tract	0,1562	72	park, field, midway,
hyp-tags	1	chromatic_color	0,1085	50	blue, yellow, red, green, pink,
hyp-tags	1	land	0,0976	45	island, cape, woodland, homestead, turf,
hyp-tags	1	woody_plant	0,0933	43	**trees** (1.5) (3.8), **tree** (1.5) (3.8), **bush** (1.5),
hyp-tags	1	administrative_district	0,0889	41	state, states,
hyp-tags	1	weapon	0,0889	41	lance, gun,
hyp-tags	1	motor_vehicle	0,0868	40	**car** (1.4) (3.2), **truck** (1.3) (2.1) (3.3), **cars** (1.4) (3.2), bike, **automobiles** (1.4), **trucks** (1.3) (2.1) (3.3), **automobile** (1.4), motorcycle,
hyp-tags	1	structure	0,0824	38	bridge, tower, building, balcony, pattern, cross, masonry, fountain, monument, buildings,
hyp-tags	1	force	0,0759	35	**Police** (3.2), **military** (2.6),
hyp-tags	1	building	0,0672	31	house, restaurant, architecture, theater, theatre, hotel, houses,
hyp-tags	1	policeman	0,0672	31	cop, cops,
hyp-tags	1	liquid	0,0629	29	water, spill,
hyp-tags	1	happening	0,0629	29	fire, case, break,
hyp-tags	1	digit	0,0586	27	quint, 2, 9, 4, 6, ii, two, 8, 3, six,
hyp-tags	1	facility	0,0586	27	station, airfield,
hyp-tags	1	natural_object	0,0586	27	stone, rocks, rock,
hyp-tags	1	tower	0,0564	26	lighthouse, watchtower,

hyp-tags	1	large_integer	0,0564	26	70, 13, 90, 28, 20,
hyp-tags	1	stream	0,0564	26	**Creek** (3.3), river,
hyp-tags	2	building	0,1627	75	resort, house, restaurant, building, architecture, theater, theatre, hotel, houses,
hyp-tags	2	agency	0,0738	34	police, usaf,
hyp-tags	2	environment	0,0586	27	preserve, surroundings,
hyp-tags	2	earth	0,0564	26	sand, clay, **dirt** (2.4),
hyp-tags	2	aircraft	0,0521	24	plane, airplane, airplanes, planes, aircraft,
hyp-tags	2	artifact	0,0651	30	**road** (2.4), marker, structure,
hyp-tags	2	line	0,0781	36	trail, directions, trails, walkway, line, route, railroad, railway, curves, horizon, heading, curve,
hyp-tags	2	hotel	0,0954	44	motel, resort, hotel,
hyp-tags	2	state	0,0976	45	peace, state, wild,
hyp-tags	3	skilled_worker	0,0499	23	officer, volunteer, marine, fishers, hanger, shoveler,
hyp-tags	3	worker	0,0542	25	officer, volunteer, marine, fishers, hanger, shoveler,
hyp-tags	3	**vehicle** (2.6)	0,0759	35	**Boat** (3.1), **boats** (3.1), **ship** (3.1), aircraft, **vehicle** (2.6), rocket,
hyp-tags	3	instrumentality	0,0564	26	**train** (3.9), **trains** (3.9), vehicle, container, equipment,
hyp-tags	3	organism	0,0542	25	indian, indians, white, fighter, straight, tiger, longer, juvenile, **animal** (1.7), **animals** (1.7), fungus, someone,
hyp-tags	3	way	0,0868	40	boardwalk, sidewalk, hiking, trail, path, directions, trails, walkway, road, way,
hyp-tags	3	vertebrate	0,0586	27	**fowl** (1.6) (2.3) (3.6), **bird** (1.6) (2.3) (3.6), **wildfowl** (1.6) (2.3) (3.6), raptor, **birds** (1.6) (2.3) (3.6), mammal,
hyp-tags	3	craft	0,0542	25	sailboat, sailboats, boat, boats, ship, aircraft,

Table 6-5: Experiment 5- Table A shows keyword results collected from test users, B shows the collection summary's tags and auto annotations and C shows the categorizations of hypernyms from the collection summary

A

Test collection 6 : Geotagged mountain summits — Number of images: 424

User 1		User 2		User 3	
Keyword	**Description**	**Keyword**	**Description**	**Keyword**	**Description**
1.1. Mountain	Yes, mountains, hills, berg, slopes, mount	2.1. Snow	Yes, high ranked tags	3.1. Snow	Yes, high ranked
1.2. Snow	Yes,very high	2.2. Mountain	Yes, high ranked tags	3.2. Mountain	Yes, high ranked
1.3. Ice	No			3.3. Sky	No, but panorama, clouds
1.4. Sky	No, panorama, clouds			3.4. Cloud	Yes, high ranked
1.5. Blue	No			3.5. Forest	No
1.6. Mountain range	Yes, hills, ridge, alps			3.6. Grass	No
1.7. Nature	Yes			3.7. Climbers	Yes, climb, climbing, mountaineering, alpinism
1.8. peak	Yes, peak, summit, high, extreme				
1.9. Tree	No				
1.10. Forest	No				
1.11. Mountain climbing	Yes, climb, climbing, mountaineering, alpinism				

B

Type	Data	Term freq	Count	Type	Data	Term freq	Count
location	country:usa	0,3231	137	tags	france	0,0825	35
location	country:france	0,1509	64	tags	germany	0,0778	33
location	region:ca	0,1203	51	tags	montagne	0,0778	33
location	country:germany	0,0849	36	tags	lake	0,0755	32
location	region:aquitaine	0,0849	36	tags	california	0,0708	30
location	region:wa	0,0731	31	tags	bavaria	0,0684	29
location	country:spain	0,0660	28	tags	pyrenees	0,0660	28
location	city:64490 lescun	0,0660	28	tags	**nature** (1.7)	0,0637	27
location	region:aragón	0,0566	24	tags	pirineos	0,0637	27
location	region:bavaria	0,0542	23	tags	travel	0,0637	27
month	september	0,1722	73	tags	**climb** (1.11) (3.7)	0,0613	26
month	august	0,1250	53	tags	lescun	0,0613	26
month	september	0,1722	73	tags	sierranevada	0,0542	23
dateHuman	2009 september	0,0755	32	tags	**climbing** (1.11) (3.7)	0,0542	23
dateHuman	2009 desember	0,0519	22	tags	rock	0,0542	23
tags	**Mountain** (1.1) (2.2) (3.2)	0,2807	119	tags	french	0,0542	23
tags	summit	0,1910	81	tags	bayern	0,0519	22
tags	**mountains** (1.1) (2.2) (3.2)	0,1580	67	tags	december2009	0,0519	22
tags	landscape	0,1509	64	weather-cond	clear	0,2807	119
tags	**Mount** (1.1) (2.2) (3.2)	0,1415	60	weather-cond	scattered clouds	0,0708	30
tags	**Peak** (1.8)	0,1203	51	weather-cond	mostly cloudy	0,0637	27
tags	hike	0,1038	44	weather-cond	partly cloudy	0,0542	23
tags	**snow** (1.2) (2.1) (3.1)	0,0991	42	weather-wind	light breeze	0,1462	62
tags	**panorama** (1.4) (3.3)	0,0991	42	weather-wind	calm	0,1274	54
tags	**alps** (1.6)	0,0943	40	weather-wind	light air	0,1156	49
tags	hiking	0,0873	37	weather-temp	moderate	0,1816	77
tags	view	0,0873	37	weather-temp	chilly	0,1745	74

C

Type	Level	Hypernym name	Term freq	Count	Hypernym members
hyp-tags	1	natural_elevation	0,4835	205	mountain, mountains, **ridge** (1.6), **hill** (1.6), **hills** (1.6), highland,
hyp-tags	1	degree	0,2005	85	**summit** (1.8), **high** (1.8), **extreme** (1.8),
hyp-tags	1	walk	0,1910	81	hike, hiking,
hyp-tags	1	limit	0,1250	53	peak, peaks,
hyp-tags	1	natural_object	0,1179	50	rock, rocks, stone, world, nest, stones,
hyp-tags	1	body_of_water	0,0943	40	lake, bay, lakes, stream, sea, waterfall,
hyp-tags	1	travel	0,0896	38	trekking, treking, journey, descending,
hyp-tags	1	quality	0,0660	28	nature, bad,
hyp-tags	1	motion	0,0660	28	travel, ascending,
hyp-tags	1	slope	0,0660	28	climb, ascent,
hyp-tags	1	physical_phenomenon	0,0637	27	**clouds** (1.4) (3.3) (3.4), **cloud** (1.4) (3.3) (3.4),
hyp-tags	1	representation	0,0566	24	photo, picture,
hyp-tags	1	person	0,0542	23	white, national, tiger, straight,
hyp-tags	2	equine	0,1439	61	mount, horse,
hyp-tags	2	weather	0,1108	47	snow, wind, weather,
hyp-tags	2	change_of_location	0,0590	25	climbing, rising, spread,
hyp-tags	2	geological_formation	0,0896	38	glacier, **berg** (1.1), crater, scree, cliff, **slopes** (1.1),
hyp-tags	2	object	0,0542	23	crater, scree, cliff, slopes, locations,
hyp-tags	2	rise	0,0896	38	**mountaineering** (1.11) (3.7), climbing, rise,
hyp-tags	2	climb	0,0967	41	**alpinism** (1.11) (3.7), climb, mountaineering,
hyp-tags	2	journey	0,0708	30	safari, trek, journey, trip, tour, ride, excursion,

Table 6-6: Experiment 6- Table A shows keyword results collected from test users, B shows the collection summary's tags and auto annotations and C shows the categorizations of hypernyms from the collection summary

A

Test collection 7 : Geotagged : France				Number of images: 537	
User 1		**User 2**		**User 3**	
Keyword	**Description**	**Keyword**	**Description**	**Keyword**	**Description**
1.1. Eiffel tower	Eiffel no, tower yes	2.1. Paris	Yes, under location (city)	3.1. Eiffel tower	Tower yes, Eiffel no
1.2. Tower	Yes, (category)	2.2. Eiffel tower	No, but tower high ranked tags	3.2. Palace	Yes, tags
1.3. France	Yes, location, tags	2.3. Structure / buildings	Yes, under structure	3.3. Cathedral	Yes, category
1.4. Paris	Yes, location, tags	2.4. Notre dame	No, but church and cathedral high ranked	3.4. River	No, but water
1.5. Culture	No	2.5. Vacation	No	3.5. Farm	No
1.6. City	No	2.6. Bridge	Yes, under structure category	3.6. Street	Yes, high ranked tags
1.7. Monument	Yes, category	2.7. Architecture	Yes structures and monument high ranked	3.7. Statue	Yes, statue, sculpture under "solid figure"
1.8. Construction	Yes, synonym structure			3.8. Plaques	No
1.9. Castle	Yes				
1.10. Ocean	No, but water				

B

Type	Data	Term freq	Count	Type	Data	Term freq	Count
location	**country:france** (1.3)	0,9870	530	tags	nationalcapital	0,2793	150
location	region:ile-de-france	0,6909	371	tags	novembre	0,1397	75
location	**city:75018 paris** (1.4) (2.1)	0,1341	72	tags	montmartre	0,1304	70
location	city:78000 versailles	0,1024	55	tags	cepatri	0,1173	63
location	**city:75004 paris** (1.4) (2.1)	0,0838	45	tags	cepatri55	0,1173	63
location	region:midi-pyrénées	0,0801	43	tags	unesco	0,1061	57
location	**city:75001 paris** (1.4) (2.1)	0,0745	40	tags	versailles	0,1024	55
location	street:4 rue de l'indépendance américaine	0,0670	36	tags	parisgeotagged	0,0912	49
location	**city:75007 paris** (1.4) (2.1)	0,0615	33	tags	**Street** (3.6)	0,0801	43
location	**city:75008 paris** (1.4) (2.1)	0,0559	30	tags	rue	0,0764	41
weekday	friday	0,2346	126	tags	**palace** (3.2)	0,0726	39
weekday	thursday	0,2142	115	tags	du	0,0708	38
month	may	0,3613	194	tags	yvelines	0,0670	36
month	november	0,3166	170	tags	**Church** (2.4)	0,0633	34
season	winter	0,3836	206	tags	**water** (1.10) (3.4)	0,0615	33
season	summer	0,3818	205	tags	park	0,0596	32
dateHuman	2002 may	0,3557	191	tags	2003	0,0559	30
dateHuman	2009 november	0,3054	164	tags	ottobre	0,0503	27
dateHuman	2003 oktober	0,0559	30	weather-cond	mostly cloudy	0,3259	175
tags	**Paris** (1.4) (2.1)	0,7002	376	weather-cond	clear	0,1155	62
tags	**France** (1.3)	0,5009	269	weather-cond	scattered clouds	0,0931	50
tags	**Parigi** (1.4) (2.1)	0,3631	195	weather-cond	partly cloudy	0,0670	36
tags	îledefrance	0,3613	194	weather-cond	light rain	0,0521	28
tags	europe	0,3594	193	weather-wind	gentle breeze	0,3277	176
tags	2002	0,3557	191	weather-wind	moderate breeze	0,2179	117
tags	westerneurope	0,3501	188	weather-temp	chilly	0,4171	224
tags	eurasia	0,3482	187	weather-temp	moderate	0,2812	151
tags	2009	0,3110	167				

C

Type	Level	Hypernym name	Term freq	Count	Hypernym members
hyp-tags	1	**structure** (1.8) (2.3) (2.7)	0,1136	61	**Tower** (1.2) (1.1) (2.2) (3.1), **monument** (1.7) (2.7), **bridge** (2.6), fountain,
hyp-tags	1	large_integer	0,0931	50	27, 29, grand, 26, 28, 11, 25, xiv, 18, 24, xxiii, grands,
hyp-tags	1	mansion	0,0857	46	palace, **castle** (1.9),
hyp-tags	1	herb	0,0801	43	rue, alexander,
hyp-tags	1	region	0,0782	42	exterior, interior,
hyp-tags	1	church	0,0540	29	basilica, **cathedral** (2.4) (3.3),
hyp-tags	1	stairway	0,0521	28	stairs, steps,
hyp-tags	2	tract	0,1080	58	plaza, piazza, park,
hyp-tags	2	solid_figure	0,0559	30	**Statue** (3.7), **sculpture** (3.7),
hyp-tags	2	thoroughfare	0,0819	44	boulevard, street,

Table 6-7: Experiment 7- Table A shows keyword results collected from test users, B shows the collection summary's tags and auto annotations and C shows the categorizations of hypernyms from the collection summary

6.3. Table layouts

From the tables from the experiment the total amounts of images included within the collection is provided in the header of table A. In table B the metadata type are located in the column *type*, the metadata term in the column *data*. Further the individual metadata term frequencies as described in section 4.7.3 in the column *term freq* and the number of images which includes this specific metadata in the column *count*. In table C the hypernyms are presented, the hierarchical level of the hypernyms (see section 4.6.7) are presented in the column *level*, the name of the hypernym in *hypernym name* and the term frequency of all tags included in this hypernym as hypernym members in the column *term freq*. Further the number of images which includes either of the hypernym members is located in the *count* column and all term found in the collection which is located to be part of the hypernym at this hierarchical level in the column *hypernym members*.

6.4. Selection threshold

The *selection threshold* used in the experiment is with small variations different depending on the type of the nature of the metadata. Recall from section 4.8.2 that for metadata with a *fixed outcome*, the *selection threshold* is calculated to be:

$$Selection\ Threshold(fixed) = \left(\frac{1}{total\ outcomes}\right) + \left(C * \left(\frac{1}{total\ outcomes}\right)\right)$$

In the experiments discussed in this chapter, the constant C is set to ½. Further the *selection threshold* for metadata with an *unfixed* outcome, which includes *locational data*, *tags* and *hypernyms*, the *selection threshold* is set to a singular constant C. In the experiments performed here, the *selection threshold* for metadata with *unfixed* outcome is set to 0.05. I.e. at a minimum 5 per cent of the images must be included with the specific metadata for that metadata to be viewed as representative for the collection.

As this *selection threshold* is manually set, it can be argued whether or not the threshold selected is a good choice, as changing this value would affect the whole experiment. Looking at the results from the section 6.2, most of the direct hits are metadata with term frequencies far above the selection threshold. Anyhow in all individual experiment some of the terms that are relevant or a direct hit are located close to the *selection threshold*. This means that setting the selection threshold any lower would exclude some of the hits. I will list the lowest relevant or direct hit in each collection below:

Test collection nr.	Lowest hit frequency
1	0.0575
2	0,0620
3	0,0502
4	0.0500
5	0.0542
6	0.0542
7	0,0540

Table 6-8: Lowest hit frequency from test experiments

Almost all collections have metadata hits/of relevance that is close to the collection threshold, which concludes that the selection threshold chosen seems like a sufficient choice. Collection 1 could be raised e.g. to 0.055, and collection 2 to e.g. 0.065 to exclude some tags not relevant, but having a consistent selection threshold prevents unnecessary confusion. On the other side lowering the selection threshold would of course give more direct hits, but give longer collection summaries. To conclude; for the experiments carried out, the *selection threshold* chosen seems reasonable after inspection.

6.5. Discussion and evaluation

In this section I will discuss the results from the experiments presented in the previous section. This section is separated into 7 sub sections corresponding to the seven different experiments performed on the three test users of the system. Before I specifically discuss the individual experiments, I will discuss the more general aspects between them.

As assumed, notice from the results presented in the section 6.2 that most of the metadata from the collection summary that is found to be relevant or direct hits compared to the user's keywords are located within the *tags* or *hyp-tags* area. These represent the terms of the collection that are manually provided by individual users on Flickr. The goal of the experiment was mainly to focus on similarities between *keywords* from the experiment and the *user provided terms* from the image collections used, as both represents human perception and since contextual data such as location, weather data and date/time data are difficult to get a perception of only looking at an image.

Auto-annotations made by the system developed (e.g. location, weather data and date/ time data) were prior to the experiment believed by the author to mainly be interesting in the context of users requesting the retrieval system with such contextual queries. Context related metadata usually requires external knowledge about the images themselves, such as at which locations they are taken, what personal impressions that relates to these images, when they were taken or a combination of several contextual data (e.g. summer vacation in Italy, July 2010). Nevertheless even though most of the similarities between the collection summary and the user's *keywords* relates to the *user provided terms*, some interesting elements were observed in the

comparison of the contextual data as well. I will present and discuss these observations more specifically in the subsequent subsections of this chapter.

Notably the *collection summary* includes much more data than the *keywords* added by the test users. The *collection summary* includes contextual data such as location specific metadata (i.e. name of *country*, *region*, *city* and *street*), date / time metadata (i.e. *weekday*, *month*, *season* and a combination of *year and month*), weather metadata (i.e. *weather condition*, *wind conditions* and *temperature conditions*) and *user provided terms* standing alone or grouped together as categories through the *hypernyms* located on WordNet.

What is interesting here is that observably the *user provided terms* from the summary that are not addressed and directly compared with the *keywords* collected from the test users, have features that bear similarities to those that are. This means that in most situations the collection summaries describes the collections with representative terms that are alike, relevant or bear similarities to the perceptions of the individual test user's from the experiment. Also in many cases where a keyword is not located in the collection summary, it is obvious why the users decided to use these terms. Often the misses are because the users provided a term that are very high level, e.g. *nature* and *culture*. Observably in such situations the collection summary hold similarities to the high level term, e.g. in the case of *nature*, the collection summary holds lower level representations such as *insects*, *trees*, *mountains* and many *nature* elements are present. This suggests that the collection summary holds more descriptive terms and not so many high level terms, such as *culture* and *nature*. In this sense the high level terms can be argued to be relevant, but are simply too difficult to agree on in some situations, hence viewed as a miss.

Some of the *user provided terms* are remarkably noisy and some are redundant and describes certain properties of the collections that also are located in the auto-annotated contextual metadata. E.g. in *experiment 1* we can see that much of the *locational metadata* are also found within the *user provided terms*, e.g. locational data *"country:Malaysia"*, *"region:pulau pinang*" and *"region:kualu lumpur"* are also found to be *representative terms* in the form of *"Malaisie"* (French for Malaysia), *"kualulumpur"* and *"Pulaupinang"/ "Penang"/ "PenangFlickr"*.

It is worth mentioning that if an image is tagged with a term that is also located when auto-annotating the image with location data, the manually added *tag* is not used by the summarizing system. The intention of this design choice is to prevent duplicate metadata. However, in the situations above such *tags* are not located since they bear small textual variations that are not caught by the summarizing system, Nevertheless, duplicates of this kind at least gives a confirmation that the contextual metadata that the images are augmented with are correct.

Throughout the rest of this section I will discuss more thoroughly the individual experiments corresponding to the individual collection results presented.

6.5.1. Experiment 1

Collection name	Geo Tagged – Malaysia
Number of images (geo-tagged)	313
number of contributors	16

Table 6-9: Collection 1 specifics

In experiment 1 one can conclude that the comparison between the collection summary and the user's tests, is with small variations, quite good. I will not discuss all keywords added by the test users, mainly since many of them are either a direct hit in the form of exactly the same terms compared against the keywords provided by the test users, or missed terms for no discussable reason. Besides I will focus on the more interesting observations.

User 1 has described the collection with 14 keywords, where 9 of them are direct hit in the collection summary, 3 of them are related or similar and 2 of them are not present or viewed as a miss. User 2 has described the collection with 6 keywords where 5 are direct hits and 1 is viewed as relevant in the collection summary. User 3 varies a little in the amount of direct hits and relevant hits compared to the first two, where of a total of 11 keywords, only 5 are viewed as direct hits in relation to the collection summary. Further 2 are viewed as relevant and 4 are viewed as misses in the collection summary.

From the direct hits notice that not all are textual exact hits, but synonyms or words that has approximately the same meaning. These will be viewed as hits if present in the collection summary. E.g. tag 1.9 reads *ocean*, which is not physically present in the collection summary but is viewed as a direct hit as its synonym *sea* is present. Another more complex direct hit is keyword 3.6, i.e. *sun*. *Sun* is also not directly present in the collection summary, but is viewed as a direct hit as both *sunset* and *sunrise* are highly weighted representatives in the summary. It can be argued that viewing *sunsets and sunrise* as a hit is questionable, but think of viewing many images that are taken of *sunsets* and *sunrises*. It is not difficult to see that *sun* is a key element of these images. In this case I believe that it's such an important element that it's worth a direct hit.

Keyword 1.1, 1.8 and 1.10, i.e. *ant*, *sky* and *city* are not present in the collection summary, but the collection summary is viewed as related to the keywords as representative terms of similarity is present. *Ant* is viewed as related as the collection summary holds a great amount of similarities to this keyword. In this example the similarity is with *insects* which in the collection summary are included with *butterfly*, *moth*, *bugs*, *beetle*, *pupa* etc. Even though none of these are a direct hits or synonyms of *ant*, all are *insects* which have been highly weighted within the collection, hence are viewed as a similarity to the user's keyword.

Similarly, in the situation for *sky* and *city*, where for *sky* highly weighted similarities are *clouds* and *panorama*. The latter since *sky* is part of the panoramic view. In the case of *city*, similarities in the collection summary are that it holds many elements relevant to city, e.g. tower, fountain, stadium, mosque, hotel and skyscraper. These elements are in my opinion

synonym to *city* in such a way that it would be too harsh viewing it as a miss.

Also keyword 2.2, and 3.7, both represented with the keyword *trees* are viewed as relevant in the collection summary as the metadata *park* and *green* present in it. My perception of a *park* includes trees and *green* and in my opinion is a typical perception of *forest*, *trees*, *plants* and so on.

As described in the introduction of this section, I assumed that most of the metadata from the collection summary that would be most comparable to the user's keywords for the collections were the user provided terms. Anyhow some interesting elements in experiment 1 that concerns this, is the direct hits 1.5 (i.e. *hot*), 1.7 (i.e. *summer*) and 3.9 (i.e. *clouds*). *Hot* is located as a representative in the collection summary using temperatures from the weather data of the images, i.e. "*Very hot"* and "*hot"* temperatures are located in 98 per cent of the images. *Summer* is located under *season* with its 41 percentile, which is found converting the time/ date stamp of the images within the collection. Finally *clouds* get a hit on *weather condition*, which is metadata also extracted from the weather data of the images, which shows that 84 per cent of the images are taken under weather condition *mostly cloudy*. This shows that augmenting images with additional contextual data can increase the semantic understanding for the images, also in relation with human's natural perception.

The keyword *nature* is used by both user 1 and user 2, i.e. keyword 1.4 and 3.11. As mentioned by the description of the two, it could be argued that these could be viewed as relevant as many elements of *nature* are present in the collection summary, e.g. *waterfall*, *park*, *green*, *animals*, *insects*, *wind*, *storm*. However, I think that describing a complex and high level term as *nature* is not enough for the terms mentioned above. Of course one could get an impression of nature when viewing these together, but in my opinion they can be related to much more than *nature* and are therefore not viewed as a close relation to nature, hence *nature* is viewed as a miss by the collection summary.

6.5.2. Experiment 2

Collection name	Ruins in Digital - Geo-tagged
Number of images (geo-tagged)	274
number of contributors	48

Table 6-10: Collection 2 specifics

In experiment 2 all users came up with exactly 9 keywords for the collection. The collection summary compared to user 1's keywords had 7 direct hits and 2 were viewed as relevant, i.e. none were viewed as a miss from the collection summary's perspective. For user 2's keywords 5 where viewed as direct hits, 1 relevant and 3 misses. For user 3's keywords 4 direct hits, 1 viewed as relevant and 4 viewed misses in the collection summary.

Many of the characteristics described by the test users hold exact matches, e.g. *ruins* (keywords 1.2, 2.4, 3.2), *old houses* (2.3, 3.3), *castle* (1.1, 2.8) and

old (1.4, 2.9), all of which are direct hits in the collection summary and obviously are important characteristics for the image collection. Notice that *old houses* (2.3, 3.3) are a combination of two words. In the image collections used all tags are separated, i.e. it is no obvious way to know if certain words are supposed to stand together to give meaning, e.g. "Eifel tower", "Big Ben" and "old houses". This means that it is impossible to know if "old" and "house" stood together when the users provided the terms, i.e. if they were not added as a merged term, e.g. "oldhouses".

Nevertheless in a context of image retrieval that would not matter, if the two separated terms both is highly representative. Since both *old* and *house/ houses* are highly weighted in the collection summary, *old houses* are viewed as a direct hit in this situation. Keyword 1.6, i.e. *construction* is also viewed as a direct hit as its synonym *structure* is present in the collection summary. All other direct hits within this experiment are exact direct hits and are not worth discussing any further.

Keyword 1.3, i.e. *ancient* is viewed relevant since *old* and *history* is present, and 1.9, i.e. *broken* since *ruin* and *devastation* are present in the collection summary. Further *street* (keyword 2.7, 3.7) is also viewed as relevant as *urban* is highly weighted, being present in 28 per cent of the images within the collection. It can be argued that this is one of those relations that are questionable. Anyhow I feel that *urban* which is synonym to *city* and *highly populated area* is also synonym to *street*. Of course it is not so related that it is referred to as a direct hit, but I believe that the collection summary deserves being viewed as relevant to this term.

Ceiling (keyword 3.9) is one of those that easily can be seen in relation with the collection summary as it holds a lot of relevant characteristics such as structures "with" ceiling, i.e. tower, building, monument, church, castle, cathedral, and palace etc. Anyhow these structures have an enormous amount of other elements which also are essential for them to be called a structure, which in my opinion blends out *ceiling*. Even though it is easy to see why and how user 3 has chosen to use this keyword, *ceiling* is for the purposes described viewed as a keyword with no direct relation for the collection summary, hence viewed as a miss.

Finally, I would like to discuss user 2's three keywords viewed as misses that draw my attention, i.e. *Rome, Italy* and *Roman* (2.1, 2.2 and 2.6), which obviously all are closely related (to each other). Looking at the augmented locational data none of the representative scenes were either in *Italy* or in Rome. Due to my curiosity I asked the user why these keywords were chosen for the collection. The user told me that she recognized that some of the images were of ruins at "*Foro Romano*", which is an ancient site and tourist attraction to which the user has visited, located in *Rome, Italy*. Since the attraction in modern times are in ruins, along with that most of the other images within this collection also are of *ruins* in other locations in the world, the user draw the conclusion that several similar sites located in the images also probably were from *"Foro Romano"*, hence she viewed these keywords as vital aspects for the collection.

Looking at all unique metadata for the collection, including what is not part of the collection summary, I found two images tagged with *"Romano"*, with locational data Rome, Italy. This highlights the concerns situated when using CBIR techniques, i.e. the problem of locating wrong data when analysing the images. If humans are confused and interrupted when making up their mind on a perception which is somewhat vague in memory, this is obviously a concern for a CBIR system which lacks sense and natural intelligence.

6.5.3. Experiment 3

Collection name	Taken FROM a bridge
Number of images (geo-tagged)	279
number of contributors	124

Table 6-11: ***Collection 3 specifics***

From experiment 3 all users provided 7 tags. For user 1's the collection summary had all 7 as direct hits. For user 2, 6 are direct hits and 1 is viewed as relevant. For user 3's the collection summary had 5 direct hits, 1 viewed as relevant and 1 miss.

As can be seen from the collection the user's keywords are very consistent, i.e. they are very similar. E.g. all users provide keywords for *bridge* (keyword 1.1, 2.3, 3.1), *railroad* / *rails* (1.6, 2.6, 2.7, 3.6) and body of water, i.e. *water* (1.2, 2.4), *ocean*/*sea* (1.3, 3.4) or *river* (2.2).

Canal (keyword 3.2) is viewed as relevant as the collection summary holds a tremendous amount of relevant tags, to mention a few *water*, *stream and river* are viewed as relevant in the collection summary. Other than that there are not much to discuss other than the possible relations between the good results in this experiment and the main features of this collection, which together with experiment 5 will be discussed later on in section 6.5.5.

6.5.4. Experiment 4

Collection name	NYC Chinatown
Number of images (geo-tagged)	380
number of contributors	127

Table 6-12: Collection 3 specifics

In test experiment 4 user 1 and user 2 have each come up with 6 tags, where also for both the collection summary had 5 are direct hits and 1 miss. For user 3's keywords the collection summary had 4 direct hits, 3 relevant and 2 misses.

The results from this experiment give a majority of exact direct hits and are not much to discuss. Some aspects can be mentioned; the keyword *culture* (1.3) can probably be argued for being relevant, but I feel that *culture* as for *nature* (discussed in section 6.5.1) is such a complex and high level terms that such a discussion could keep going on for ever.

Nevertheless, because of the complexity and the difficulties in locating what is relevant and not for this term, it is viewed as a miss in relation with the collection summary as it is not an exact, direct hit. The collection summary is viewed as relevant for *skyscrapers* (3.3) since many similar structures are highly weighted within it, e.g. *tower* and *buildings*. Also the collection summary are viewed as relevant for *fish* (3.5) since it is not a direct synonym, but rather relevant for *seafood* which is a part of the collection summary. Finally, *statues* (3.7) are viewed as relevant since *characters* and *imaginary beings* are highly weighted within the collection summary.

As can be seen from the collections summary of test collection 4, this summary consists of a much bigger amount of data than the other collection summaries viewed so far. One reason for this may be that in average all images within this collection is provided with twenty *user provided terms* each after all cleansing and removal has been performed, that is excluding the auto-annotated metadata. Also the collection is included with many different kinds of images, added by 127 different individual users.

6.5.5. Experiment 5

Collection name	Geotagged: Delaware
Number of images (geo-tagged)	461
number of contributors	42

Table 6-13: Collection 5 specifics

The results gathered from test collection 5 are the best results given in the evaluation of the system, i.e. the experiment with the highest direct hit score compared to the users keywords. User 1 has provided 7 keywords for the collection, where the collection summary has 6 direct hits and 1 is viewed as relevant. User 2 has provided 6 keywords of which all 6 are direct hits in the collection summary, and user 3 has provided 9 keywords where the collection summary has 8 direct hits and 1 is viewed as relevant.

The high scores within this collection may be due to its consistencies in the images that are part of the collection itself. With this I mean that the images provided within it are of similar things and contexts, which make it easier for users to describe their perception of it with keywords that are synonym to the collection summary. To highlight this aspect, I would like to present a comment expressed by one of the test users of the experiment:

"I think that describing some of the image collections with a few keywords is a bit difficult, as many of the collections include many groups of different images. For this reason, I found it hard to decide which of these groups that is dominating the image collection, and describing them through keywords that represent my perception of them".

One collection that in my opinion reflects the aspect highlighted by the test user here, is the collection provided in the previous experiment, i.e. experiment 4. In this collection a lot of quite different user provided terms

and hypernyms are located to be representative for the collection by the summarizing system as the collection are provided with images with many different kinds of subjects. This is also an element present in experiment 6 and 1. Obviously this may frustrate the users, being fed with an overflow of different perceptions, making it difficult to filter them out in a reasonable manner. For experiment 5, presented here, it seems to be the opposite, i.e. that the image collection are consistent, hence it is easier for the collection summary to present direct hits with the users keywords.

Another aspect that may be playing a role in the good results from this experiment is that not only are the images consistent, but also the perception of the users tagging them. It could be that for the amount of images in the collection the amount of contributors tagging these images may be at a focal point. In this collection there are 461 images contributed by 42 users on Flickr. In the other collections there are some that has much fewer images per user (e.g. experiment 2 and 4) and some with more images per user (e.g. experiment 1 and 7), notably these gives variable results in comparison with the keywords from the test users. It could be that having a reasonable amount of images per contributor in the collection, gives a much more mediated perception represented by the collection summary, and prevent perceptions of individual users being too dominating. Also of course the images must be consistent and without to many dominating famous and known (by the user) attractions. In relation with these claims, the results from this experiment, along with those from experiment 3, suggest that having a reasonable amount of users under these circumstances, gives an average of more direct hits. Of course this would have to be specifically experimented on at a much larger scale to give any concrete evidence, but can be viewed as a possibly outcome.

As discussed in the results of experiment 2, strings of terms meant to stand together are usually separated in the collections that are used in the experimentation for this project. This means that e.g. “big ben” and “old houses” etc. are each separated into two words, i.e. “big” and “ben” and “old” and “houses” in advance by Flickr. For this reason keywords represented by a set of words are viewed as a direct hit if all terms within the set are highly weighted and part of the collection summary.

In experiment 5 many of these situations are present. “*Fire station*” (keyword 1.2) is viewed as a direct hit as both *fire* and *truck* is highly weighted in the collection summary, similarly is the case for “*fire truck*” (keyword 1.3, 2.1, 3.3), “*dirt road*” (keyword 2.4), “*military vehicle*” (keyword 2.6) and “*police car*” (keyword 3.2). One example of a situation that does not fall into this category is in the case of the keyword “*train cart*” provided by user 3 (keyword 3.9). In this situation only *train* is part of the collection summary and *cart* is not. Even though the most highly weighted term here are *train*, it is not representative to view the keyword provided by the user (i.e. “train cart”) as a direct hit as only one of the two terms is present within the collection summary. Hence the collection summary are only viewed as relevant for keyword 3.9.

6.5.6. Experiment 6

Collection name	Geotagged mountain summits
Number of images (geo-tagged)	424
number of contributors	93

Table 6-14: Collection 6 specifics

From the 6th experiment user 1 provided 11 keywords, where the collection summary had 6 direct hits, 1 was relevant and 4 misses. User two provided only two keywords where both 2 were direct hits, and user 3 provided 7 where 4 were direct hits, 1 relevant and 2 misses. Also in this collection the test users had a majority of keywords that hold similarities. For instance all users provided the keywords *snow* (keyword 1.2, 2.1 and 3.1) and *mountain* (keyword 1.1, 2.2, and 3.2). Other similarities between the test user's keywords are *mountain range* (keyword 1.6), *nature* (keyword 1.7), *peak* (1.8), *mountain climbing* (1.11) and *climbers* (keyword 3.7), which obviously holds similarities in the description of the collection. Also the user provided terms from the collection summary are closely related in some kind to these keywords, along with some contextual data that are not properly cleansed as discussed in the introduction of this section 6.3. Examples of similar terms found representative in the collection summary which is not directly connected to the users keywords are, *landscape*, *hike*, *view*, *travel* etc, which obviously holds similarities to the keywords mentioned above.

The collection summary has three misses, i.e *tree* (1.9), *forest* (1.10 and 3.5) and *grass* (3.6). I've chosen to view these as not directly relevant since not only are there no direct hits, but actually no obvious relations, such as other *woody plant* or other *outgrowths* or *plants*.

Nevertheless in relation with the collections summary, it is not hard to see that specific users get different perceptions of these aspects, perception is subjective and different users notice different features as more important than others. E.g. *nature*, *view* and *panorama* are features found representative within the collection summary and are perceptions that most probably are in relation with views that include perceptions such as *forest*, *grass* and *trees* of some kind. Other than that these keywords (i.e. *tree*, *forest* and *grass*) are close to the selection threshold looking at the unique word list, but not above, hence not part of the summary. Anyhow even if some relations can be seen between the keywords and the collection summary, I cannot say these are close enough, viewed as a miss by the collection summary.

6.5.7. Experiment 7

Collection name	Geotagged : France
Number of images (geo-tagged)	537
number of contributors	11

Table 6-15: Collection 7 specifics

From the last and 7th experiment user 1 provided 10 keywords, where the collection summary has 6 direct hits, 2 are relevant and 2 is not relevant. User two provided 7 keywords where the collection summary has 4 direct hits, 2 are viewed as relevant and 1 miss. Finally, user 3 provided 8 where the collection summary has 4 direct hit, 2 are relevant and 2 misses.

From the last experiment there is not much that not already has been highlighted and discussed in the previous experiments, but one element that are interesting, is that obviously a famous attraction is part of this collection. All users have used the keyword *Eiffel Tower* (keyword 1.1, 2.2, 3.1) in one of the first two keywords of their keywords list. The collection summary does not include the tag *Eiffel*, but only the tag *tower*; hence the collection summary is only relevant for the keyword "*Eiffel Tower*". Looking at the unique term list produced by the system *Eiffel* are provided as a tag in several images, but not so frequently that it is viewed as representative for the image collection. What is interesting here is that the *Eiffel Tower* is so highly famous that all users could identify it in the image collection, but most probably this gave such a distinguishing impression for the users that they just could not get it out of their mind. As a result the famous, and highly noticeable attraction located in some of the images gave the users such a distinguished impression that they viewed it as highly representative for the collection, which is different to the opinion of the summarizing system.

In my opinion what can be learned from this is that the users can easily get distracted when viewing something that is known to the individuals. This leaves the user with a subjective perception which influences their judgement. What is positive in this specific situation is that some of the users observably also could put the images on the map, i.e. *Paris, France*. Two of the users have provided the keyword *Paris* (keywords 1.4 and 2.1) and one of the users provided the keyword *France* (keyword 1.3). Most probably these locational keywords were provided by the influence of the recognized famous attraction, i.e. *Eiffel tower*. Both France and Paris are highly representative in the collection summary, *France* are recognized by the locational metadata in the collection summary in 99 per cent of the images, and *Paris* in approximately 40 per cent of the images.

6.6. Hit distribution

Below is the hit distributions of the classifications of the collection summary in relation with the keywords provided by the test users are presented, i.e. direct hit, relevant, miss. In *Figure 6-1* hit distributions compared to the results from the individual users in each experiment are presented. In *Figure 6-2* the average scores for all keywords provided by the test users in each experiment is presented, before finally, the overall average for all users in all experiments is presented in *Figure 6-3*. The hit scores varies to some extent for the individual users, nevertheless the overall hit rate is quite good. The average hit score in all experiments are as can be seen in table 6.5 is 69 per cent; overall relevance score is 14 per cent while overall score for keywords viewed as misses is only 17 per cent.

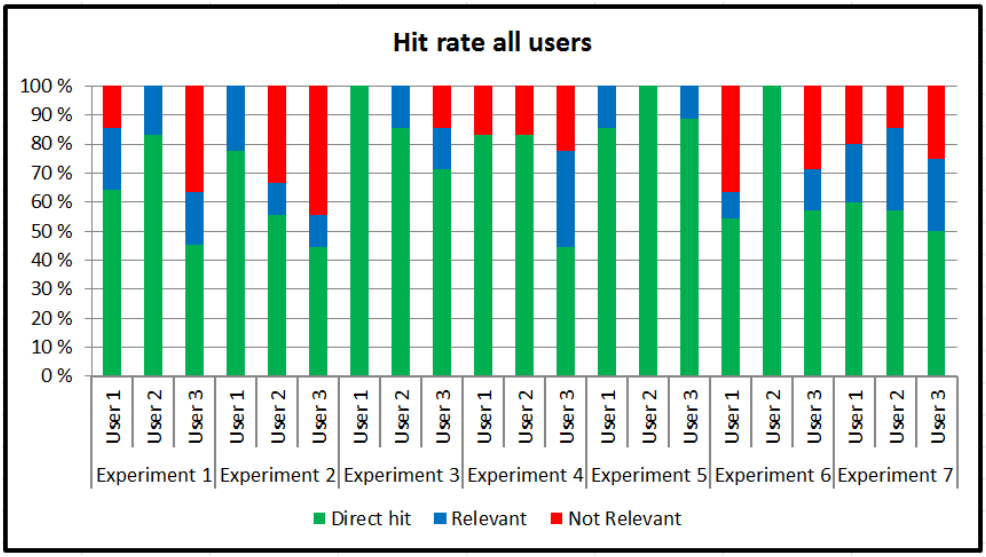


Figure 6-1: Hit rate all users, all experiments

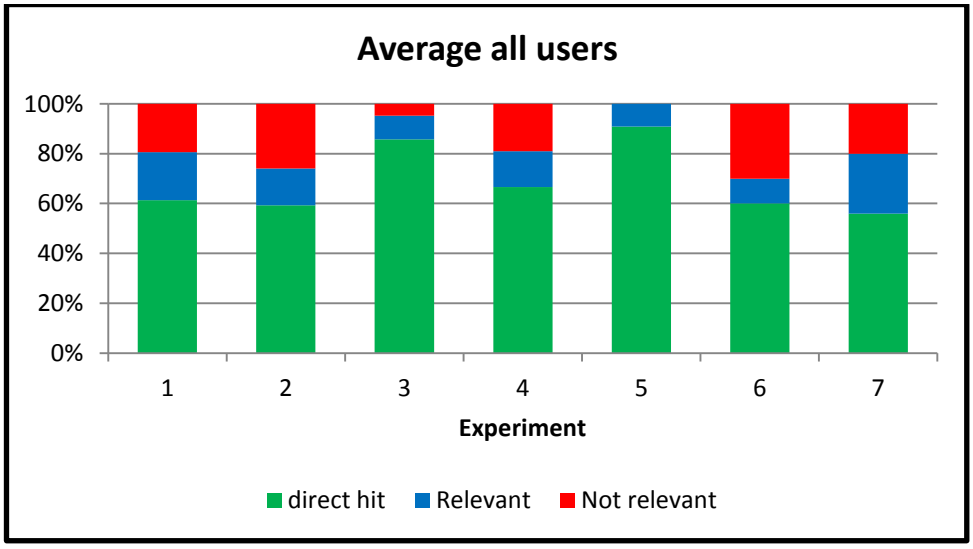


Figure 6-2: Average of hit distribution for all three users in the 7 different experiments

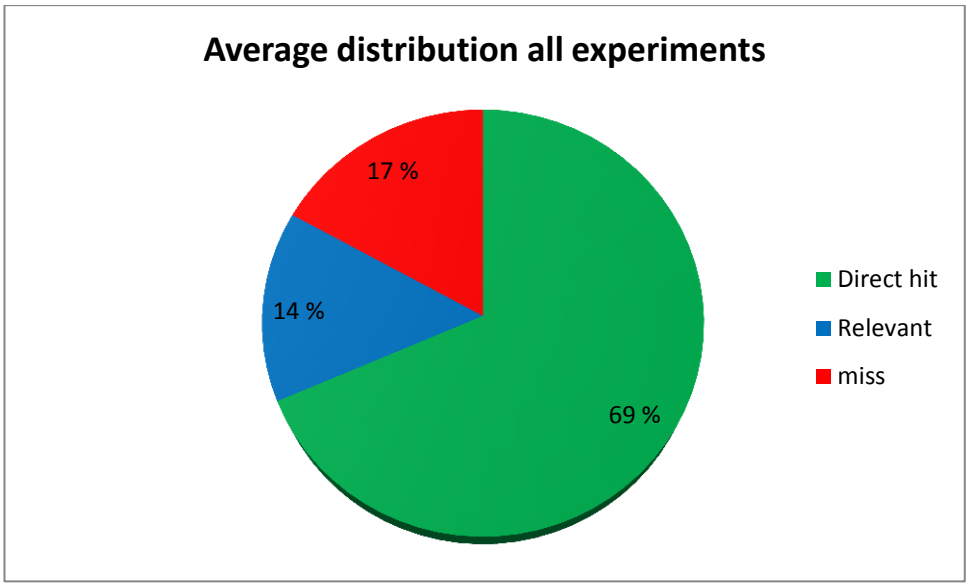


Figure 6-3: Average distribution of all users in all experiments

Chapter 7

7. Future work

In this chapter I will discuss some future work that is relevant for the project.

The system developed has been validated using different collaborative tagged image collections from Flickr. The results produced from these have shown to be a descent support for human's natural perception with an average of 69 % direct hit rate and 14 % relevant hit rate from all test experiments. An interesting enhancement of the system would be to take advantage of the images located on Flickr, using the available data as a training set for automatically annotating other images e.g. located in a private image collection. The Flickr API supports retrieval of images within a defined radius of a specific coordinate, these images could be analysed, representative tags located and automatically annotated to other images taken at the same location. A solution to this has been developed by Evertsen [48]. Inducing similar functionality as Evertesen's solution would be interesting to test in the system. This would at least be highly relevant for privately, *untagged* collections of images. Other interesting elements related with the retrieval system are reducing the search space based on scenarios and contextual aspects such as location and time depending on the contextual need for the user of the system.

The solution has not been tested with functionality for searching and retrieving images based on user queries. This makes it difficult to validate the usefulness of the contextual data that the images within the collection are auto-annotated with. The results from keyword 1.5, 1.7, 3.9 etc in section 6.5.1 (experiment 1), suggests that also human perception can be supported by the contextual data gathered from the images in the collection. Building a retrieval system, validating results returned for specific contextual search queries are desired, so that these constraints can be strengthened. This could be an option in a future project.

As discussed in chapter 4, the manually added tags from the image collections include potentially user misspelled words.. This can be misspellings, expressions, abbreviations or code words related to spoken language or strings of words that has not been separated at the time of insertion. An interesting expansion of this project could be to translate such occurrences by using an advanced language processing algorithm. This could increase the level of understanding of the metadata connected with the images and also has potential for resulting in a more meaningful collection summary.

Another interesting enhancement of the system is finding a more mature way of handling term selection. The developed system is limited to weight metadata by calculating term frequencies. In stable and defined environments the method adjusts the *term frequency* using the *inverse document frequency*. The features are then selected using a selection threshold varying on what is most natural for the specific metadata type.

This approach favours the most frequent terms, and cannot be said to be an advanced approach to term selection. Although categorization of terms raises the amount of important features included in the collection summary, a more mature approach to term selection are desired for the system as a future enhancement.

As discussed in chapter 2 several advanced algorithms are used within the field of information theory, as Principal Component Analysis (PCA) and sematic accumulating algorithms such as Singular Value Decomposition (SVD) and Latent Semantic Analysis / Indexing (LSA / LSI). These techniques are developed for analysing large amounts of textual documents. Using such techniques could help in carefully selecting features that are tailored for the image collections, taking into consideration more parameters. Interesting parameters would be to take into consideration the different contributors or users on Flickr, looking at user patterns, establishing machine learning techniques and user relevance feedback.

CBIR techniques, also discussed in *chapter 2*, have not been proven to be a stable solution; nevertheless they have shown great potential within image retrieval. Augmenting the images with additional metadata, or just strengthening available metadata using CBIR techniques would be of interest in an expansion of the project. Using advanced CBIR techniques would help in closing the semantic gap and provide deeper and more specific collection summaries. CBIR techniques are on the other side demanding for use of machine resources. Nevertheless, a potential enhancement for the system would be to test with variant advanced CBIR techniques.

The system developed is tailored the experiments performed in the testing and evaluation of the system as presented in this thesis. The system tested is of limited use. No built in connectors to private collections is implemented and no advanced and user friendly graphical interface is developed for the system. Built in connectors to private collections have to include functionality for reading the EXIF header of the images, as this is the most highly used format for connecting metadata with images. Functionality that supports easy accumulation of new image collections are also desired, either from the internet or private computers. Together these functionalities are highly desired as future enhancements of the system.

The system developed has been focused on analysing image collections and producing a collection summary represented by the most representative terms of the collection. As an effect of this, the system has not been fully optimized in terms of execution time. It should be taken into consideration for all methods used to enhance or optimize execution time. Enhancements of this kind would include solutions for more efficient auto-annotation which is related to the restrictions of the locational and atmospheric conditions that the images are augmented with. Solutions will include acquiring a premium API key from Google and using alternative weather data providers, since Wunderground only supports pure HTML returns. A

solution for the latter would be getting access to the API that should be available as a payable service [28] by the service provider AccuWeather[29].

Other improvements in terms of execution time, is optimizing the inverse document frequency calculation. A great amount of data is held in memory at execution time, the majority in array lists. These lists are not indexed which means that lookups on these exponentially raises in terms of time the bigger the collections being analysed are. Nevertheless, the collections used in the evaluation of the system are so small that collections summaries are produced in an average of around 11.5 seconds, hence indexing the array lists of metadata has not been prioritized when testing the system. Optimization in terms of indexing includes more efficient creation of unique term files, more efficient cleansing of data and more efficient allocation of term categories (hypernyms). All of these relates to acquiring efficient indexing techniques. Notably the data are not fed into the database until the collection summaries has been fully produced. The database is though optimized for updates and insertions performed by the summarizing system.

[28] http://www.programmableweb.com/api/accuweather
[29] http://www.accuweather.com/

Chapter 8

8. Conclusion

In this thesis a system that annotate, convert data, categorized manually added terms, locates representative metadata and creates a summarized description for the collection that can be used for efficient image retrieval has been developed and tested.

The evaluation of the system is carried out comparing the resulting collection summaries to keywords added by the test users from the experiment. About 83 % of the of the keywords provided for the image collections by the test users are captured in the collection summaries, either viewed as a direct hits or relevant hits. The selection threshold chosen seems like a sufficient choice in relation with the size of the collection summary and the number of hits located. *The evaluation clearly concludes that the collection summary captures the perception of the users.*

The collection summaries increase in cases where the image collections are included with a higher number of *terms per user*, or if the collection includes several groups of similar images. Nevertheless, by inspections bigger collections (up to 7500 images) does not provide any significantly longer summaries, which indicates that this approach will not grow out of proportions as larger image collections are included within it.

As expected the majority of the direct and relevance hits are captured by the *user provided terms*, as these are most natural to human perception. Nevertheless, the evaluation shows that also some of the hits are captured by the contextual metadata terms, e.g. *hot*, *summer*, *mostly cloudy* etc. The contextual metadata terms are the most useful for requesting contextual queries to a retrieval system. However, as suggested by the evaluation, this metadata also supports humans' perception, which includes weather specific terms, locational terms and the high level periodic (seasonal) terms.

As discussed in *section 2.1.1* humans tend to associate images with more high level terms than low level terms. The evaluation suggests that this claim is two sided. The users has provided *low level* keywords such as *tree, ant, butterfly* while other provide more *high level* terms such as *forest*, *nature, bug*. For a high level term such as *nature* it can be difficult to claim that the collection summary is relevant if it is not a direct hit. By inspection, in most such situations the collection summary includes lower level terms that can be viewed as relevant. This indicates that the collection summary provides more descriptive terms for the image collection, which is positive.

Recall from section 4.7.6, that the system group terms into higher level representations, referred to by WordNet as a *Hypernym*. Grouping *user*

provided terms into *hypernyms* shows great support for *lower level* terms provided by the test users. *Hypernyms* both provides a higher level description of common terms, through the *hypernym* names, as well as keeping track of the lower level terms grouped together. Notably, a great amount of the direct hits of the low level terms provided by the test users are recognized by the representative *hypernym* members from the collection summary.

Some of the misses observed in the evaluation of the system indicates that users tend to get confused when recognizing well known attractions in the image collections. In such situations the users tend to recognize the attraction as representative for the collection even if only present in a few of the images, hence not recognized as a representative by the summarizing system.

In favour of the collection summary, what has shown positive in such situations is that also the locational terms comes into use. The evaluation has shown by inspection that test users have localized famous attractions down to *city* and *country* level. This is positive in collections where images with less distinguishing subjects are taken in the same area. Also this suggests that the collection summary is much more trustworthy than individual user's perception, where the users can easily lose focus while the collection summary allocates representatives without a subjective meaning.

The goal was to produce *collection summaries for image collection using available metadata*. This included making them available for more efficient image retrieval and supporting selection of the most relevant image collections based on the collection summaries during an image search. The system developed and evaluated in this thesis suggests that the collection summaries produced captures the most important aspects (terms) of the image collections analysed. At the same time the collection summary has a significantly reduced search space and is more semantically meaningful. The latter is a consequence of augmenting the images with additional metadata and grouping terms into hierarchical representations. The work done indicates that using this approach considerably increases the effectiveness of an image search, both in terms of time and resources, and at the same time returning more relevant images. Of course, producing collection summaries is of no use if it does not give good and trustworthy results. However, the developed system and the evaluation carried out suggests that the approach is successful, although there might exist systems that produce similar or better results.

9. References

[1] K. S. Candan and M. L. Sapino, *Data Management for Multimedia Retrieval*: Cambridge Univ Pr, 2010.

[2] Y. Rui, *et al.*, "Image Retrieval: Current Techniques, Promising Directions, and Open Issues," *Journal of Visual Communication and Image Representation,* vol. 10, pp. 39-62, 1999.

[3] O. Lassila, "Web metadata: a matter of semantics," *Internet Computing, IEEE,* vol. 2, pp. 30-37, 1998.

[4] R. Datta, *et al.*, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Computing Surveys (CSUR),* vol. 40, p. 5, 2008.

[5] J. T. Pollock and R. Hodgson, *Adaptive information: Improving business through semantic interoperability, grid computing, and enterprise integration*: Wiley-Blackwell, 2004.

[6] M. Davis, *et al.*, "From context to content: leveraging context to infer media metadata," 2004, pp. 188-195.

[7] N. O'Hare, *et al.*, "Combination of content analysis and context features for digital photograph retrieval," 2005, pp. 323-328.

[8] J. Jeon, *et al.*, "Automatic image annotation and retrieval using cross-media relevance models," 2003, pp. 119-126.

[9] A. J. Cheng, *et al.*, "GPS, compass, or camera?: investigating effective mobile sensors for automatic search-based image annotation," 2010, pp. 815-818.

[10] A. K. Dey, "Providing architectural support for building context-aware applications," Citeseer, 2000.

[11] M. Naaman, *et al.*, "From where to what: Metadata sharing for digital photographs with geographic coordinates," *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE,* pp. 196-217, 2003.

[12] R. Sarvas, *et al.*, "Metadata creation system for mobile images," 2004, pp. 36-48.

[13] P. Enser, "The evolution of visual information retrieval," *Journal of Information Science,* vol. 34, p. 531, 2008.

[14] S. A. Golder and B. A. Huberman, "Usage patterns of collaborative tagging systems," *Journal of Information Science,* vol. 32, p. 198, 2006.

[15] M. Davis, *et al.*, "MMM2: mobile media metadata for media sharing," 2005, pp. 1335-1338.

[16] S. P. Mathew and P. Samuel, "A novel Image Retrieval System using an effective region based shape representation technique," *International Journal of Image Processing (IJIP),* vol. 4, p. 509, 2010.

[17] J. Cui, *et al.*, "Real time google and live image search re-ranking," 2008, pp. 729-732.

[18] T.-T. Pham, *et al.*, "Latent semantic fusion model for image retrieval and annotation," presented at the Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, Lisbon, Portugal, 2007.

[19] A. Vailaya, *et al.*, "Image classification for content-based indexing," *Image Processing, IEEE Transactions on*, vol. 10, pp. 117-130, 2001.
[20] H. T. Pu, "An analysis of failed queries for web image retrieval," *Journal of Information Science*, vol. 34, p. 275, 2008.
[21] S. Deb and Y. Zhang, "An overview of content-based image retrieval techniques," in *Advanced Information Networking and Applications, AINA. 18th International Conference*, 2004, pp. 59-64 Vol. 1.
[22] A. Hughes, *et al.*, "Text or pictures? An eyetracking study of how people view digital video surrogates," *Image and Video Retrieval*, pp. 1-6, 2003.
[23] Y. Choi and E. M. Rasmussen, "Users' relevance criteria in image retrieval in American history," *Information Processing & Management*, vol. 38, pp. 695-726, 2002.
[24] I. Jolliffe, "Principal component analysis," *Encyclopedia of Statistics in Behavioral Science*, vol. 3, pp. 1580-1584, 2002.
[25] C. D. Manning, *et al.*, *Introduction to information retrieval* vol. 1: Cambridge University Press Cambridge, UK, 2008.
[26] (2003). *Exchangeable image file format for digital still cameras: Exif Version 2.2 (2.2 ed.)*. Available: http://www.exif.org/Exif2-2.PDF
[27] G. D. Abowd and E. D. Mynatt, "Charting past, present, and future research in ubiquitous computing," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 7, pp. 29-58, 2000.
[28] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, pp. 94-104, 1991.
[29] S. Loke, *Context-aware pervasive systems: architectures for a new breed of applications*: Auerbach Pub, 2006.
[30] D. L. Hall and J. Llinas, *Handbook of multisensor data fusion*: CRC, 2001.
[31] B. P. Clarkson, "Life Patterns: structure from wearable sensors," 2003.
[32] D. Nicklas and B. Mitschang, "The nexus augmented world model: An extensible approach for mobile, spatially-aware applications," 2001.
[33] D. Frohlich, *et al.*, "Requirements for photoware," presented at the Proceedings of the 2002 ACM conference on Computer supported cooperative work, New Orleans, Louisiana, USA, 2002.
[34] K. Rodden and K. R. Wood, "How do people manage their digital photographs?," presented at the Proceedings of the SIGCHI conference on Human factors in computing systems, Ft. Lauderdale, Florida, USA, 2003.
[35] A. Spink, *et al.*, "Use of query reformulation and relevance feedback by Excite users," *Internet research*, vol. 10, pp. 317-328, 2000.
[36] J. Brank, *et al.*, "A survey of ontology evaluation techniques," 2005, pp. 166–170.
[37] C. Ming-Syan, *et al.*, "Data mining: an overview from a database perspective," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 8, pp. 866-883, 1996.

[38] M. Kifer, *et al.*, *Database Systems: An Application-Oriented Approach, second edition*, 2 ed.: Addison Wesley Publishing Company, 2005.
[39] E. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, pp. 377-387, 1970.
[40] M. Naaman, *et al.*, "Context data in geo-referenced digital photo collections," 2004, pp. 196-203.
[41] K. Toyama, *et al.*, "Geographic location tags on digital images," 2003, pp. 156-166.
[42] N. O'Hare, *et al.*, "Using text search for personal photo collections with the MediAssist system," 2007, pp. 880-881.
[43] E. Chang, *et al.*, "CBSA: content-based soft annotation for multimodal image retrieval using Bayes point machines," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, pp. 26-38, 2003.
[44] S. Feng, *et al.*, "Multiple bernoulli relevance models for image and video annotation," 2004.
[45] L. Kennedy, *et al.*, "How flickr helps us make sense of the world: context and content in community-contributed media collections," 2007, pp. 631-640.
[46] I. Simon, *et al.*, "Scene summarization for online image collections," 2007.
[47] D. Oxford. (2011, 19.05.2011). *Oxford dictionary*. Available: http://oxforddictionaries.com/view/entry/m_en_gb0702250#m_en_gb0702250
[48] M. H. Evertsen, "Automatic Image Tagging based on Context Information," Master Thesis, Computer Science, University of tromsø, Tromsø, 2010.

Appendix A: List of image collections

In this appendix the image collections used in the evaluation of the system are listed. The image collections used are gathered from Flickr groups.

OBS! Notably, the image metadata used in the evaluation was gathered by the system 24.05.2011. This means that some of the groups may have different amounts of images compared to the evaluation. Using the Flickr API it is possible to specify the update date of images gathered, e.g. specifying that you do not want images added after 24.05.2011, but this is not possible when viewing images online.

The images in the Flickr groups used can be viewed at the following addresses:

- Experiment 1: http://www.flickr.com/groups/geotag-malaysia/pool/
- Experiment 2: http://www.flickr.com/groups/ruinas/pool/
- Experiment 3: http://www.flickr.com/groups/takenfromabridge/pool/
- Experiment 4: http://www.flickr.com/groups/612535@N22/pool/
- Experiment 5: http://www.flickr.com/groups/geotagdel/pool/
- Experiment 6: http://www.flickr.com/groups/mountainsummit/pool/
- Experiment 7: http://www.flickr.com/groups/geotagged-france/pool/

The images are not owned by Flickr but by the users on Flickr. Nevertheless, developers must follow the creative commons[30] license unless otherwise is agreed upon by the developers and individual users on Flickr. Cotion

[30] www.flickr.com/creativecommons

Appendix B: Implementation

B-1: Gather collections from Flickr (Menu)

Notably, menu, API keyw and large print procedures are removed from appendix.

```
# -*- coding: cp1252 -*-
import os
from time import strftime
import time
import sys
from pprint import pprint
from elementtree.ElementTree import Element, SubElement, ElementTree

import flickrapi

import datetime
from datetime import date

global api_key
global fullDocName

import codecs
import re

# Repository files
import findRepresentatives
import augmentLocation
import wunderground
import converter
import cleanser
import uniqueTermHandler as utHandler

import py_compile



class tagForm:
    def __init__(self):
        self.type = ''
        self.tag = ''
        self.count = 0
        self.tf=0
        self.idf = 0
        self.categories = []
        self.wnLookupCount = 0
        self.users = []
        self.imageIds = []
        self.userCount = 0


# Used to check if group name (or other search criterias) matches the
# search query exactly
def findExactGroupName(groups, inName):
    query = re.sub(r'"','',inName)

    # for all groups returned
    for group in groups[0]:

        # If group name is exactly alike search 'query', get images for
        # group
        if group.attrib['name']==query:
            return group, True
        else:
            continue;

    return group, False
```

```
# Prepare image collection data before finding representative "tags"
# 1. Separate tags into location, tags, title and date tags array
# 2. Sort all arrays on tag.count
# 3. Call findrepresentatives analyze function
def findBestCandidates(docPath, numOfImages, collectionName):

    uniqueLoc, uniqueDate, uniqueTag, uniqueTitle, uniqueWeather, uniqueWind, uniqueTemp =
[],[],[],[],[],[],[]

    # Get uniqueTerms from Unique term handler
    uniqueTags = utHandler.returnUniqueTags()

    # 1. Separate tags into location, tags, title and date tags array
    for i in uniqueTags:
        if i.type == 'location':
            uniqueLoc.append(i)
        elif i.type == 'tags':
            uniqueTag.append(i)
        elif i.type == 'title':
            uniqueTitle.append(i)
        elif (i.type == 'weekday') or (i.type == 'season') or (i.type == 'dateHuman')
                            or (i.type == 'month'):
            uniqueDate.append(i)
        elif (i.type == 'weather-cond'):
            uniqueWeather.append(i)
        elif (i.type == 'weather-wind'):
            uniqueWind.append(i)
        elif (i.type == 'weather-temp'):
            uniqueTemp.append(i)
        else:
            print 'DEBUG findBestCandidates()!!!'+ i.type

    # 2. Sort all arrays on tag.count
    temp = sorted(uniqueLoc, key=lambda tag: tag.count, reverse=True)
    uniqueLoc = temp
    temp = sorted(uniqueTag, key=lambda tag: tag.count, reverse=True)
    uniqueTag = temp
    temp = sorted(uniqueTitle, key=lambda tag: tag.count, reverse=True)
    uniqueTitle = temp
    temp = sorted(uniqueDate, key=lambda tag: tag.count, reverse=True)
    uniqueDate = temp
    temp = sorted(uniqueWeather, key=lambda tag: tag.count, reverse=True)
    uniqueWeather = temp
    temp = sorted(uniqueWind, key=lambda tag: tag.count, reverse=True)
    uniqueWind = temp
    temp = sorted(uniqueTemp, key=lambda tag: tag.count, reverse=True)
    uniqueTemp = temp

    # 3. Call findrepresentatives analyze function
    findRepresentatives.findReps(uniqueLoc, uniqueDate,uniqueTag,
                                 uniqueTitle,uniqueWeather, uniqueWind,
                                 uniqueTemp, docPath , numOfImages,
                                 collectionName)

    return 0



# Analyze collection
# Finds unique terms and count freqencies for them
def analyzeCollection(pages, locSet, dateSet, weatherSet):

    i=0;

    # Browse through all pages of images from the collection
    # collected from flickr in main()
    for page in pages:
        # For all images in page
        print 'Currently analyzing page: '+page.attrib['page']+' of ' +page.attrib['pages']
        for image in page:
            i+=1

            # If Location tag set
            if locSet == True:
                # Gather location from location augmenter, returns country, region city and street
```

```
        location = augmentLocation.gatherLocation(float(image.attrib['latitude'])
                                    ,float(image.attrib['longitude']))

        locations = location.split(', ')
        image.attrib['location-augmented'] = location
    if dateSet == True:
        # Convert date to human readable (e.g. "2008 january")
        humanReadable = converter.convertDateToHumanReadable(image.attrib['datetaken'])
        image.attrib['dateHuman-augmented'] = humanReadable[0]

        # Add month of year picture taken as human readable
        humanRdMonth = []
        humanRdMonth.append(humanReadable[0].split(' ')[1])
        image.attrib['month-augmented'] = humanRdMonth[0]


        # Get weekday from date, return human readable
        # weekday (monday, tuesday etc)
        weekday = converter.findWeekday(image.attrib['datetaken'])
        image.attrib['weekday-augmented'] = weekday[0]

        # Get season of year, i.e. summer, winter etc
        season = converter.findSeason(image.attrib['datetaken'].split(' ')[0])
        image.attrib['season-augmented'] = season[0]

    if weatherSet == True:
        wInfo = wunderground.getWeatherInfo(image.attrib['latitude'],
                                    image.attrib['longitude'],
                                    image.attrib['datetaken'])
        print wInfo
        if wInfo == 'exit':
            # Not found
            image.attrib['weather'] = "empty"
        else:
            image.attrib['weather'] = wInfo.lower()

    # Get all single words from 'title' field of image and insert
    # if unique
    tmp = image.attrib['title'].lower()
    wordsTitle = tmp.split(' ')

    # Get all single words from 'tags' field of image and insert
    # if unique
    tmp = image.attrib['tags'].lower()
    wordsTags = tmp.split(' ')

    if locSet == True:
        # Remove metadata from 'title' and 'tags' that are
        # allready found using locational info to prevent
        # duplicate frequncy count
        for loc in locations:
            if loc in wordsTitle:
                wordsTitle.remove(loc)
            if loc in wordsTags:
                wordsTags.remove(loc)

        # Remove weekday from 'Title' and 'Tags'
        if weekday[0] in wordsTitle:
            wordsTitle.remove(weekday[0])
        if weekday[0] in wordsTags:
            wordsTags.remove(weekday[0])

        # Remove month from 'Title' and 'Tags'
        if humanRdMonth[0] in wordsTitle:
            wordsTitle.remove(humanRdMonth[0])
        if humanRdMonth[0] in wordsTags:
            wordsTags.remove(humanRdMonth[0])

        # Remove season from 'Title' and 'Tags'
        if season[0] in wordsTitle:
            wordsTitle.remove(season[0])
        if season[0] in wordsTags:
            wordsTags.remove(season[0])

    # Sleep to prevent gmaps to overflow with convert queries
    time.sleep(0.10)
```

```
    return pages


# Store xml page (elementtree) from flickr to file
# args:
#   nameOfFile - Name of file to store to (remember path)
#   page - xml data to store:
#       example: page = flickr.photos_search(group_id=nsid, has_geo=True,
#                per_page=50, page=1, extras='geo, tags, date_taken,
#                owner_name, date_upload')
def storePageToFile(nameOfFile, pages):

    fileName = utHandler.changeCarBeforePrintOrWrite(nameOfFile)

    try:
        ElementTree(pages).write(re.sub(r'"','',fileName))
        return True
    except IOError:
        print 'OBS: '+ IOError.args
        return False

# Get page from fileName to store it into elementTree
# returns True and page elementTree if success
def readPageFromFile(fileName):
    fileName = utHandler.changeCarBeforePrintOrWrite(fileName)
    try:
        tree = ElementTree(file=re.sub(r'"','',fileName))
        pageElems = tree.getroot()
        return True, pageElems
    except IOError:
        print 'OBS: '+ IOError.args
        return False,

# Get api key from file
def getApiKeyFromFile(keyName):
    f = codecs.open("./API_KEYS.txt", mode='r')
    retData="blank"
    for i in f:
        line = i.strip('\n')
        if len(line)!=0:
           data = line.split('=')
           if len(data)==2:
              if(data[0]==keyName) and (len(data[1])>1):
                 f.close()
                 return data[1].strip('"')

          print "Could Not find given API key: "+keyName
          print "Check that key correctly inserted in API_KEYS.txt"
          f.close()

    return "missing"

# Locate given group from Flickr and gather all image metadata
def main(query, searchType):
    # get api key from file
    flickr_api_key = getApiKeyFromFile("Flickr_API_key")
    if (flickr_api_key=="missing"):
        return False

    # Create instance of the flickrapi client
    flickr = flickrapi.FlickrAPI(flickr_api_key, cache=True)

    groups = flickr.groups_search(text=query)
    group, found = findExactGroupName(groups, query)

    if found == False:
        print 'Sorry could not find any groups according to search query: '+query
        print 'Please try again'
        exit()



    # Collect nsid for group, required to get images using image_search
    nsid = group.attrib['nsid']

    # Get images from group
    # Variables set:
```

```
    #   * Get images for group by nsid
    #   * has_geo -> only return images that are geo tagged
    #   * per_page -> images returned per page,
    #   * page -> which page to return images from

    # getAll used to insert all pages in one elementtree
    if searchType == 'group':
        allPages = flickr.photos_search(group_id=nsid, has_geo=True, per_page=50, page=1,
                                        extras='geo, tags, date_taken, owner_name, date_upload,
description')

    print allPages[0].items()
    print
    print 'numbers returned per page: ' + str(allPages[0].__len__())
    print 'number of pages: '+ allPages[0].attrib['pages']
    print 'Total number of images: '+ allPages[0].attrib['total']
    print int(allPages[0].attrib['pages'])
    i=0

    pageCount = 1;
    pageTot = int(allPages[0].attrib['pages'])
    #Insert first page of images

    # clear, get ready to insert all pages
    allPages.clear()

    while pageCount <= pageTot:
        photos = flickr.photos_search(group_id=nsid, has_geo=True, per_page=50, page=pageCount,
                                      extras='geo, tags, date_taken,
                                              owner_name,description,date_upload')
        print photos.items()
        # insert page into getAll (element tree)
        allPages.append(photos[0])


        print '-----------------------------------------------------'
        print allPages[pageCount-1].items()
        print pageCount
        print '-----------------------------------------------------'
        pageCount += 1;



    print 'finished collection, print to file'
    print
    print '-----------------------------------------------'
    storePageToFile(fullDocName, allPages)



if __name__=="__main__":

    utHandler.initUniqueTags()
    path = 'c:/Users/David/Documents/Skole/Master oppgave/Program/Collection Data/'
    menu = MenuChoices()
    # Menu removed from appendix
    # …….
```

B-2: Augmentation

This sub appendix holds two functionalities:

B-2.1: Augmentation of location data
B-2.2: Augmentation of weather data

B-2.1: Augmentation of locational data

```
LOCATION AUGMENTER
from googlemaps import GoogleMaps

#repository files
Import flickr_api_comm as comm


def gatherLocation(latitude, longitude):
    # Get API key
    gMaps_api_key= comm.getApiKeyFromFile("GMaps_API_key")
    if (gMaps_api_key=="missing"):
        return False

    gmaps = GoogleMaps(gMaps_api_key)

    # Get location data from lat, long using google maps API
    address = gmaps.latlng_to_address(latitude, longitude)
    time.sleep(0.10)

    # Get geographic information using full address, 'address'
    # will only be used if location information not found in
    # placemark
    try:
        ret = gmaps.geocode(address.encode('utf8'))
        placemark = ret['Placemark'][0]
    except:
        placemark=''

    if len(placemark) > 0:
        addresses = address.split(', ')
        try:
            region = 'region:'+
                                placemark['AddressDetails']['Country']['AdministrativeArea']
                                    ['AdministrativeAreaName'].lower()
        except:
                region = 'region:'
        try:
            city = 'city:'+
                        placemark['AddressDetails']['Country']['AdministrativeArea']['Locality']
                                ['LocalityName'].lower()
        except:
            if len(addresses) > 2:
                city = 'city:'+addresses[1].lower()
            else:
                city = 'city:'
        try:
            street = 'street:'+
                                placemark['AddressDetails']['Country']['AdministrativeArea']
                                    ['Locality']['Thoroughfare']['ThoroughfareName'].lower()
        except:
            if len(addresses) > 1:
                street = 'street:'+addresses[0].lower()
            else:
```

```
            street = 'street:'

    addresses.reverse()      # reverse list
    if len(addresses) >= 1:
        country = 'country:'+addresses[0].lower()
    else:
        country = 'country:'
    location = (street +', '
            + city +', '
            + region +', '+country)
    print (""+ country +
             ', '+region+
             ', '+ city+
             ', ' + street)
else:
    print 'Adress NOT Found for coordinates'
    location = address

return location
```

B-2.2: Augmentation of weather data

```
Augmenter - WunderGround

import urllib2
import urllib
from BeautifulSoup import BeautifulSoup
import datetime
import socket

# OBS: MAYBE PUT THIS FUNCTION INTO "CONVERTER.PY"?
# Convert 12 hour clock to 24 hour clock
# Takes hour (AM/PM) as input and returns hour 24 hour format
def to24(hour12, isPm):
    return (hour12 % 12) + (12 if isPm else 0)

# Get weather information given location (latitide, longitude) and
# date/ time. Format has to be YYYY-MM-DD HH:MM:SS
#
# Function does the following
# 1. Find year, month, day, hour, minute and second from argument
#    'lookupdate'
# 2. Find closest weather station and its station id using coordinate
# 3. Using the station id found in (2.) get daily history according to
#    date (lookUpDateTime)
# 4. Find all paragraphs (p) , table heads (theads) and table bodies
#    (tbody) from daily history html code
# 5. Check if daily history found
# 6. If found(5.), From last tbody (reversed earlier) get weather
#     information from table elements (td)(i.e last table lists weather
#     info), also clean data
# 7. Find table element that matches lookup date best, using time of day from weather table element
# 8. Return best match
def getWeatherInfo(lat, lon, lookUpDateTime):


    # 1. Find year, month, day, hour, minute and second from argument
    # 'lookupdate'
    inTimeSplit =  lookUpDateTime.split(' ')
    year = int(inTimeSplit[0].split('-')[0])
    month = int(inTimeSplit[0].split('-')[1])
    day = int(inTimeSplit[0].split('-')[2])
    hour = int(inTimeSplit[1].split(':')[0])
    minute = int(inTimeSplit[1].split(':')[1])
    second = int(inTimeSplit[1].split(':')[2])

    if ((year==0) or (month == 0) or (day== 0)):
        return 'empty'

    # 2. Find closest weather station and its station id using coordinate
    #    Try to connect to api a maximum of three times before preceeding
    connFail = 0
    url ="http://wunderground.com/auto/wui/geo/WXCurrentObXML/index.xml?query="+lat+","+lon
    while(1):
        try:
            page = urllib.urlopen(url)
            break
        # Catch URL ERRors
        except urllib2.URLError:
            connFail +=1
            if connFail >= 3:
                print 'Failed three times: exit'
                return 'exit'
            else:
                print "Failed contacting: "+url
                print 'Tries to reconnect: Attempt: '+str(connFail)
        # Catch connection timeout
        except socket.timeout:
            connFail +=1
            if connFail >= 3:
                print 'Failed three times: exit'
                return 'exit'
            else:
                print "Failed contacting: "+url
```

```
            print 'Tries to reconnect: Attempt: '+str(connFail)
    # Catch unknown failures
    except:
        connFail +=1
        if connFail >= 3:
            print 'Failed three times: exit'
            return 'exit'
        else:
            print "Failed contacting: "+url
            print 'Tries to reconnect: Attempt: '+str(connFail)
soup = BeautifulSoup(page)

# Get station Id from returned xml
stationId = soup.find('station_id').text.encode('utf8')
print "Station id located: " + stationId

# 3. Using the station id found in (2.) get daily history using lookup date
#    Connect a maximum of three times if timed out before preceding to next image
connFail = 0
url = "http://www.wunderground.com/history/airport/"+stationId+"/"+str(year)+"/"+str(month)+"/"+str(day)+
          "/DailyHistory.html"
while(1):
    try:
        req = urllib2.Request(url)
        page = urllib2.urlopen(req)
        break
    except urllib2.URLError:
        connFail +=1
        if connFail >= 3:
            print 'Failed three times: exit'
            return 'exit'
        else:
            print "Failed contacting: "+url
            print 'Tries to reconnect: Attempt: '+str(connFail)
    except socket.timeout:
        connFail +=1
        if connFail >= 3:
            print 'Failed three times: exit'
            return 'exit'
        else:
            print "Failed contacting: "+url
            print 'Tries to reconnect: Attempt: '+str(connFail)
    except:
        connFail +=1
        if connFail >= 3:
            print 'Failed three times: exit'
            return 'exit'
        else:
            print "Failed contacting: "+url
            print 'Tries to reconnect: Attempt: '+str(connFail)


print url

# 4. Find all paragraphs (p), table heads (thead) and table bodies
# (tbody) from daily history's html code
soup = BeautifulSoup(page)
Ps = soup.findAll('p')
theads = soup.findAll('thead')
tbodys = soup.findAll('tbody')
if len(tbodys) ==0:
    print 'No tbodys!!!'
    return 'empty'

# Reverse all lists (the last table is the one interesting for
# us(weather information table))
Ps.reverse()
theads.reverse()

# get last tbody, which is hourly weather condition
tbodys.reverse()
noData = False

# 5. Check if daily history found
#    One of the paragrahps include the text 'No daily or hourly...'
#    if weather information not found
for i in Ps:
```

```
    if i.text == 'No daily or hourly history data available':
        print i.text
        noData = True
        break


# 6. If found(5.), From last tbody (reversed earlier) get weather
#    information from table elements (td)
#    (i.e last table lists weather info), also clean data
lines = []
line =''
if noData == False:

    # Get Last table (tbodys reversed above)
    weather_conditions = tbodys[0]

    # From table get all weather inputs (weather elements)
    # during the day
    get_headers = weather_conditions.findAll('td')
    count = 0

    # For all weather elemets clean data and insert into array
    for i in get_headers:
        if (count==0) or (count==3) or (count==11):
            if (count ==11):
                line += (i.text.encode('utf8'))
            else:
                line += (i.text.encode('utf8')+', ')
        elif (count == 1) or (count==2):
            tmp = i.text.encode('utf8')
            tmp2 = tmp.replace(' &deg;C','')
            line += (tmp2 + ', ')
        else:
            tmp = i.text.encode('utf8')
            tmp2 = tmp.replace(' ','')
            line += (tmp2 + ', ')
        if count ==11:
            line += '###'
            count =0
        else:
            count +=1

else:
    print 'Weather Data not found! ' + i.text
    correctLine = 'empty'
    return correctLine

lines = line.split('###')

# Make lookup date, date/ time format
#(used to compare with weather table elements)
inDateTime = datetime.datetime(year, month,
                  day,
                  hour,
                  minute,
                  second)
if len(lines) == 0:
    correctLine = 'empty'
elif len(lines) == 1:
    correctLine = lines[0]

# 7. Find table element that matches lookup date best, using time of
#    day from weather table element
elif (len(lines) > 1):
    correctLine = ''
    prev = ''
    # For all weather table elements
    for i in lines:
        # Split element and gather time
        string = i.split(', ')
        tmAMPM = string[0].split(' ')
        tm=tmAMPM[0].split(':')
        try:
            hour24 = to24(int(tm[0]),
                          True if tmAMPM[1]=='PM' else False)
        except:
```

```
            print lookUpDateTime
            return 'exit'
        # get compare time and insert it into same day year month as
        # lookupdate for easier comparison
        cmpDateTime = datetime.datetime(year, month,
                     day,
                      hour24,
                      int(tm[1]))
        # If lookupdate higher that compare date (from weather table)
        # continue
        if cmpDateTime < inDateTime:
            prev = i
            continue
        # If not, insert match
        else:
            if prev =='':
                correctLine = i
            else:
                correctLine = prev
            break
# 8. return best match
return correctLine
```

B-3: Cleansing of metadata

```
CLEANSER

import findRepresentatives
import unicodedata as ud
import re

# Check if uchr is in latin form
def is_latin(uchr):
    latin_letters= {}
    try: return latin_letters[uchr]
    except KeyError:
         return latin_letters.setdefault(uchr, 'LATIN' in ud.name(uchr))

# Check that all characters in a string is roman
def only_roman_chars(unistr):
    return all(is_latin(uchr)
           for uchr in unistr
           if uchr.isalpha()) # isalpha suggested by John Machin

# Remove and clean words form 'title' and 'tags' list
# 1. remove words from a defined list of words
# 2. Remove noisy words. I.e. words used by only one user
# 3. Remove all words that are not roman, i.e. not asian or russian characters
def removeWords(array):
    remList = ['the', 'and', 'is', 'a', 'de', 'la', 'this','in', 'on',
               'to', 'for', 'by', 'from','at','me','my', 'you','with',
               'canon', 'eos', 'geotagged', 'geotag', 'of', '400d',
               'cannoneos400d','from', 'over', '50d', 'leica', 'nikon',
               '1020mm', '90mm','photoshopalbum', '35mm', 'flickr',
               'fujifilmfinepixs8000fd','nikond60','d60', 'flickrmeetup']
    newAfterRemove = []

    # 1. Remove words from list
    for i in array:
        if i.tag not in remList:
            newAfterRemove.append(i)
        else:
            continue

    newAfterClean = []

    # 2. Remove noisy words, longer than 50 char
    for i in newAfterRemove:
        if (len(i.tag) >= 50):                    # Noisy words
            continue
        # If word used by more than one user
        else:
            newAfterClean.append(i)

    onlyRomanWords = []
    # 3. Remove all words that are not roman, i.e. not asian
    # or russian characters
    for i in newAfterClean:
        if only_roman_chars(i.tag)== True:
            onlyRomanWords.append(i)


    return onlyRomanWords

# Removes special characters from manually added terms
def cleanData(array):
    count = 0
    temp =[]
    for i in array:
        # Remove special characters from element.tag
        new = re.sub(r'[,. ;:"#!/()^=+*&$?-]','',i)

        # If string empty after removal, remove element
        if len(i) != 0:
            temp.append(new)
    return temp
```

B-4: Converting of metadata

```
# Convert numeric wind strength, wind strength represented by m/s
def beaufortScale(strength):
    scale = [['calm', 0.0, 0.3], ['light air', 0.3, 1.5],
             ['light breeze', 1.6, 3.4],['gentle breeze',3.4, 5.4],
             ['moderate breeze', 5.5,7.9],
             ['fresh breeze', 8.0, 10.7], ['strong breeze', 10.8, 13.8],
             ['moderate gale', 13.9, 17.1], ['gale', 17.2, 20.7],
             ['strong gale', 20.8, 24.4], ['storm', 24.5, 28.4],
             ['violent storm', 28.5, 32.6],['hurricane', 32.7, 1000]]
    for beaufort in scale:
        if (strength >= beaufort[1] and strength <= beaufort[2]):
            description = beaufort[0]
            break
    return description

# Convert numeric temperature (Celsius)
def convertTemperature(temp):
    scale = [['freezing', -100.0, -15.0],['ice cold', -14.9, -10.0],
             ['cold', -9.9, 0.0], ['chilly',0.1, 10.0],
             ['moderate',10.1, 20.0], ['hot', 20.1, 25.0],
             ['very hot', 25.1, 35.0], ['extremely hot', 35.1, 100]]
    # If temperature not avaliable
    if temp == 'N/A':
        return 'N/A'

    for coldHot in scale:
        if temp >= coldHot[1] and temp <= coldHot[2]:
            desc = coldHot[0]
            break

    return desc

# Convert numeric value, wind strength to beaufort scale
def convertWind(windString):
    # Windstring format '18.5km/h/5.1m/s'
    try:
        windStrengthString = windString.replace('km/h', '')
        windStrengthString = windStrengthString.replace('m/s', '')
        windList = windStrengthString.split('/')

        # wind strength represented by m/s
        windStrength = float(windList[1]);
        desc = beaufortScale(windStrength)
    except:
        return windString
    return desc

# Convert wind strrength and temperature to human readable
# Arguments:
# weaterdata - weather data Array []
def convertWeatherData(weatherData):
    windStrengthString = weatherData[4]
    # Check if temperature avaliable
    try:
        temperatureFloat = float(weatherData[10])
    except: # if not avaliable
        temperatureFloat = 'N/A'
    return convertWind(windStrengthString), convertTemperature(temperatureFloat)


# Returns a weekday of the form saturday, sunday etc
# Arguments:
# dateStr -> string date of the form 'YYYY-MM-DD HH:MM:SS'
def findWeekday(dateStr):
    dateTime = dateStr.split(' ')
    YMD = dateTime[0].split('-')
    try:
        date = datetime.date(int(YMD[0]), int(YMD[1]), int(YMD[2]))

        weekdayInt = date.weekday()
        weekday = []
        if weekdayInt == 0:
```

```
            weekday.append('monday')
        elif weekdayInt == 1:
            weekday.append('tuesday')
        elif weekdayInt == 2:
            weekday.append('wednesday')
        elif weekdayInt == 3:
            weekday.append('thursday')
        elif weekdayInt == 4:
            weekday.append('friday')
        elif weekdayInt == 5:
            weekday.append('saturday')
        elif weekdayInt == 6:
            weekday.append('sunday')

    # Throws exception if date format is wrongly tagged in image
    except:
        print YMD[0]+',' + YMD[1]+',' + YMD[2]
        weekday = []
        weekday.append('not found')
    return weekday

# Find season from date, i.e. spring, summer, fall, winter
# Date on the form of "YYYY-MM-DD"
def findSeason(inDate):

    date = inDate.split('-')
    season = []
    try:
        springStart = datetime.date(int(date[0]), 3, 1)
        summerStart = datetime.date(int(date[0]), 5, 1)
        fallStart = datetime.date(int(date[0]), 9, 1)
        winterStart = datetime.date(int(date[0]), 11, 1)

        cmpDate = datetime.date(int(date[0]), int(date[1]), int(date[2]))

        if (cmpDate >= springStart) and (cmpDate <= summerStart):
            season.append('spring')
        elif (cmpDate >= summerStart) and (cmpDate <= fallStart):
            season.append('summer')
        elif(cmpDate >= fallStart) and (cmpDate <= winterStart):
            season.append('fall')
        else:
            season.append('winter')

    # Throws exception if date format is wrongly tagged in image
    except:
        season.append('not found')


    return season



# Handle converting of date/time
def convertDateToHumanReadable(date):

    # date is on the form e.g. "2008-07-22 09:48:05"
    monthTranslater = [[1,'january'],[2,'february'],[3,'mars'],
                       [4,'april'],[5,'may'],[6,'june'],[7,'july'],
                       [8,'august'],[9,'september'],[10,'oktober'],
                       [11,'november'],[12,'desember']]

    # Split date and time
    dateTime = date.split(' ')
    # get date
    date = dateTime[0].split('-')
    month = 'not found'                 #only if not new value set below
    for monthData in monthTranslater:
        if monthData[0]==int(date[1]):
            month=monthData[1]
            break;

    # Convert year month in the form "YYYY month" e.g. "2008 january"
    # Rreturn as array because of the form of appendUnique()
    humanReadable = []
    humanReadable.append(date[0]+' '+ month)
    return humanReadable
```

B-5: Unique Term handler

```
import codecs
import re

# Repository files
import flickr_api_comm as flick
import converter
import cleanser

global uniqueTags
# Create unique term file
def initUniqueTags():
    global uniqueTags
    uniqueTags = []
def returnUniqueTags():
    global uniqueTags
    return uniqueTags

# Handle insertion of unique terms to uniqueTerms list
def appendUniqueTags(words, tagType, userId, imageId):
    global uniqueTags

    # for all words
    for word in words:
        # Check if unique, insert word if True or increment frequency
        # counter for word if not
        check = checkUnique(word, tagType, userId, imageId)

        if check == False:
            a = flick.tagForm()
            a.type = tagType
            a.tag = word
            a.count = 1
            a.users.append(userId)
            a.imageIds.append(imageId)
            a.userCount=1
            uniqueTags.append(a)

    return uniqueTags

# Check if unique, insert word to uniqueTags if True or increment
# frequency counter for word if not
def checkUnique(word, tagType, userId, imageId):
    global uniqueTags

    for elem in uniqueTags:
        # Increment element counter
        if word == elem.tag:
            elem.count+=1
            elem.imageIds.append(imageId)
            if userId not in elem.users:
                elem.users.append(userId)
                elem.userCount +=1

            if (tagType == 'location') and (elem.type != 'location'):
                elem.type= tagType

            elif (tagType == 'weekday') and (elem.type != 'weekday'):
                elem.type= tagType
            return True

    return False

def printUnique(fullDocName):
    global uniqueTags
    temp2 = []
    temp = uniqueTags


    # Sort unique tags reverse on count element
    temp2 = sorted(temp, key=lambda tag: tag.count, reverse=True)
    uniqueTags = temp2
```

```
    # Write to file
    try:
        fullDocName = changeCarBeforePrintOrWrite(fullDocName)
        f = codecs.open(re.sub(r'"','',fullDocName)+'#freqTags.txt',
                        encoding='utf-8', mode='w+')
        for elem in uniqueTags:
            i=0
            imagesStr = ''
            for img in elem.imageIds:
                if i==0:
                    imagesStr += str(img)
                else:
                    imagesStr += '--'+str(img)
                i+=1

            f.write(elem.type + ', ' + elem.tag + ', ' +
                    str(elem.count)+', ' + str(elem.userCount)+', '
                    + imagesStr+'\n')
            if elem.count > 0:
                print elem.type + ', ' + elem.tag + ', '+ str(elem.count)+', ' +
                          str(elem.userCount)

        f.close()
    except IOError:
        print 'Write Error: '+ IOError.args

    return uniqueTags, fullDocName

# Handle all pages in order to produce unique term files
def createUniqueTagsFile(pages, fullDocName):
    global uniqueTags
    #uniqueTags = []
    i=0;

    # Browse through all pages of images from the collection
    # collected from flickr in main()
    for page in pages:
        # For all images in page
        print 'currently analyzing page: '+page.attrib['page']+' of ' +page.attrib['pages']

        # Process all images
        for image in page:
            i+=1
            location = image.attrib['location-augmented']
            imageId = int(image.attrib['id'])
            locations = location.split(', ')
            appendUniqueTags(locations, 'location',
                             image.attrib['owner'], imageId)

            # Convert date to human readable (e.g. "2008 january")
            humanReadable = []
            humanReadable.append(image.attrib['dateHuman-augmented'])
            appendUniqueTags(humanReadable, 'dateHuman',
                             image.attrib['owner'], imageId)

            # Add month of year picture taken as human readable
            humanRdMonth = []
            humanRdMonth.append(humanReadable[0].split(' ')[1])
            appendUniqueTags(humanRdMonth, 'month', image.attrib['owner'],
                             imageId)


            # Get weekday from date, return human readable
            # weekday (monday, tuesday etc)
            weekday = []
            weekday.append(image.attrib['weekday-augmented'])
            appendUniqueTags(weekday, 'weekday', image.attrib['owner'],
                             imageId)

            # Get season of year, i.e. summer, winter etc
            season = []
            season.append(image.attrib['season-augmented'])
            appendUniqueTags(season, 'season', image.attrib['owner'],
                             imageId)
```

```
# Get weather information (IF found)
weatherString =  image.attrib['weather']
if (weatherString  != 'empty') and (weatherString != 'exit'):
    weatherArray = weatherString.split(', ')
    weatherArray.reverse()
    weatherCond = []
    weatherCond.append(weatherArray[0].lower())
    # Add weather cond
    appendUniqueTags(weatherCond, 'weather-cond',
                     image.attrib['owner'], imageId)

    # Add Wind strength, temperature
    windString, temperatureString = converter.convertWeatherData(weatherArray)
    wind, temperature = [], []
    wind.append(windString.lower())
    temperature.append(temperatureString.lower())

    appendUniqueTags(wind, 'weather-wind',
                     image.attrib['owner'], imageId)
    appendUniqueTags(temperature, 'weather-temp',
                     image.attrib['owner'], imageId)

    #



# Get all single words from 'title' of image and insert
# if unique
tmpString = image.attrib['title'].lower()
wordsTitle = tmpString.split(' ')

# Get all single words from 'tags' of image and insert
# if unique
tmpString = image.attrib['tags'].lower()
tempList = tmpString.split(' ')
wordsTags = []

# no duplicate terms in image, either in tags or title field
for tag in tempList:
    if tag not in wordsTitle:
        wordsTags.append(tag)

# Remove metadata from 'title' and 'tags' that are allready found
# using locational info to prevent duplicate frequncy count
for loc in locations:
    realLoc = loc.split(':')
    try:
        if loc[1] in wordsTitle:
            wordsTitle.remove(loc[1])
        if loc[1] in wordsTags:
            wordsTags.remove(loc[1])
    except:
        continue

# Remove weekday from 'Title' and 'Tags'
if weekday[0] in wordsTitle:
    wordsTitle.remove(weekday[0])
if weekday[0] in wordsTags:
    wordsTags.remove(weekday[0])

# Remove month from 'Title' and 'Tags'
if humanRdMonth[0] in wordsTitle:
    wordsTitle.remove(humanRdMonth[0])
if humanRdMonth[0] in wordsTags:
    wordsTags.remove(humanRdMonth[0])

# Remove season from 'Title' and 'Tags'
if season[0] in wordsTitle:
    wordsTitle.remove(season[0])
if season[0] in wordsTags:
    wordsTags.remove(season[0])

newWordsTitle = cleanser.cleanData(wordsTitle)

# Argument 2 tags since tags field and title field merged
uniqueTags = appendUniqueTags(newWordsTitle, 'tags', image.attrib['owner'],
                 imageId)
```

```
            newWordsTags = cleanser.cleanData(wordsTags)
            appendUniqueTags(newWordsTags, 'tags', image.attrib['owner'],
                             imageId)

    # Store unique terms to file
    printUnique(fullDocName)

# Special cases where collection names includes the charachter
# ":", e.g. "Geotagged : France"
def changeCarBeforePrintOrWrite(string):
    sList = list(string)
    i=0 # counter to prevent ":" after hard drive letter removed
    for char in sList:
        if (char == ':' and i!=1):
            sList[i]="!"
        i+=1

    ret = "".join(sList)
    print ret
    return ret

# Get unique terms from file and store to unique term list (in memory)
def testGetTags(fullDocName):
    global uniqueTags

    fullDocName = changeCarBeforePrintOrWrite(fullDocName)

    f = codecs.open(re.sub(r'"','',fullDocName)+'#freqTags.txt',encoding='utf-8', mode='r')
    for i in f:
        line = i.strip('\n')
        if len(line) != 0:
            words = line.split(', ')
            a = flick.tagForm()
            a.type = words[0]
            a.tag = words[1]
            a.count = int(words[2])
            a.userCount = int(words[3])

            #get all imageIds
            tmpStrArray = words[4].split('--')
            tmpIntArray = []
            for elem in tmpStrArray:
                tmpIntArray.append(int(elem))

            uniqueTags.append(a)                # insert into list

    f.close()
    return uniqueTags
```

B-6: Hierarchical level handler (Hypernym grouping)

```
import findRepresentatives as findReps
from nltk.corpus import wordnet as wn

class catForm:
    def __init__(self):
        self.type = ''
        self.tag = ''
        self.synset = ''
        self.count = 0
        self.tf=0
        self.idf = 0
        self.subs = []

class catNewLevelForm(catForm):
    def __init__(self):
        catForm.__init__(self)
        self.rootTerm = ''

# Check if found in array, returns true if found false if not
def checkUnique(word, array):
    for elem in array:
        if word == elem.tag:
            return True
    return False

# Finds hypernyms for each words contained in array using wordnet api
def findCategories(array, typ):
  categories = []
  for i in array:
    word = wn.synsets(i.tag)
    # Always use first word
    if len(word) > 0:
      i.wnLookupCount = len(word)
      word1 = word[0]

      cats = word1.hypernyms()
      for cat in cats:
        temp = cat.name
        catName = cat.name.split('.')
        i.categories.append(catName[0])

        found = checkUnique(catName[0], categories)

        # If found insert hypernym into category list
        if found == False:
          s = catForm()
          s.type = 'cat-'+typ
          s.tag = catName[0]
          s.synset = cat
          categories.append(s)


  return array, categories

# For all categories check if array has some elements that equal the
# category if so increase count of category and include element from
# array in category.subs
def checkCategories(array, catList):
    for cat in catList:
        for i in array:
            if cat.tag == i.tag:
                cat.subs.append(i.tag)
                cat.count += i.count
            elif cat.tag.replace('_',' ') == i.tag:
                cat.subs.append(i.tag)
                cat.count += i.count
            elif cat.tag.replace('_','') == i.tag:
                cat.subs.append(i.tag)
                cat.count += i.count

    return catList
```

```
# Find all words from array that has a word from 'catList' (category list)
# in its hypernym list 'elem.categories' and counts the categories from
# catList as all related words from array
def reCreateCountFromCategories(array, catList):
  for i in array:
    for cat in i.categories:
      for j in catList:
        if cat == j.tag:
          j.subs.append(i.tag)
          j.count += i.count

  return catList


# Move one step up in hierarchcal hypernym structure
def newLevelInTree(array, category, allreadyAbove, level):
    newCategory = catNewLevelForm()

    synset = category.synset
    if level == 2:
        newCategory.rootTerm = category.tag
    elif level == 3:
        newCategory.rootTerm = category.rootTerm

    syns = synset.hypernyms()
    if len(syns) == 0:
        return False, newCategory

    # initialize new hypernym (category) level
    newCategory.synset = syns[0]
    syn = syns[0]
    newCategory.subs = category.subs
    newCategory.count = category.count
    newCategory.tf = category.tf
    newCategory.idf = category.idf
    newCategory.type = category.type

    synName = syn.name.split('.')
    realName=(synName[0])               # realname of hypername

    # locate new hypernym members from metadata (terms)
    newCategory.tag = realName
    for i in array:
        if(i.tag == realName):
            if i.tag not in newCategory.subs:
                if i.tag not in allreadyAbove:
                    newCategory.subs.append(i.tag)
                    newCategory.count += i.count
                    newCategory.tf += i.tf
                    newCategory.idf += i.idf
        else:
            for j in i.categories:
                if j == realName:
                    if i.tag not in newCategory.subs:
                        if j not in allreadyAbove:
                            newCategory.subs.append(i.tag)
                            newCategory.count += i.count
                            newCategory.tf += i.tf
                            newCategory.idf += i.idf

    return True, newCategory

def produceNewCatLevels(array, catList, numOfImages, threshold, xlsFile):
    sortedList = []
    temp = catList

    # Sort unique tags reverse on count element
    sortedList = sorted(temp, key=lambda tag: tag.count, reverse=True)
    catList = sortedList

    allreadyAbove, secondLevel, finalLevel, nextRound = [],[],[],[]
    if(len(array)==0): return False

    print "-----------------------------------"
    print "second level " + array[0].type
    print "-----------------------------------"
    for elem in catList:
```

```
        tf = elem.tf
        # Prevent hypernyms with only one category member to be include,
        # Such occations are already above in tags
        if  tf>= threshold and len(elem.subs)>1:
            allreadyAbove.append(elem.tag)

        # Append categories to new level above selection threshold
        else:
            before = elem.count
            found, newCat = newLevelInTree(array, elem, allreadyAbove, 2)
            after = newCat.count

            # If hypernym has been increased in size
            if after > before:
                tf = newCat.tf
                # If TF above selection threshold and more than one member
                if tf >= threshold and len (elem.subs) > 1:
                    allreadyAbove.append(newCat.tag)
                    print "%20s %5s %5s %20s %s" % (newCat.rootTerm,
                                                    str(before),
                                                    str(after),
                                                    str(newCat.tag),
                                                    str(newCat.subs),)
                    # Add row to .xls file
                    xlsFile.addRowCategory(newCat, 2)
                    secondLevel.append(newCat)
                # If TF is NOT above selection or only one member
                else:
                    nextRound.append(newCat)

    print "-----------------------------------"
    print "Third level " + array[0].type
    print "-----------------------------------"
    # Process hypernyms for level two
    for elem in nextRound:
        before = elem.count
        found, newCat = newLevelInTree(array, elem, allreadyAbove, 3)
        after = newCat.count
        # If hypernym has been increased in size
        if after > before:
            tf = newCat.tf
            # If TF above selection threshold and more than one member
            if tf >= threshold and len(elem.subs) > 1:
                allreadyAbove.append(newCat.tag)
                print "%20s %5s %5s %20s %s" % (newCat.rootTerm,
                                                str(before),
                                                str(after),
                                                str(newCat.tag),
                                                str(newCat.subs),)
                # Add row to .xls file
                xlsFile.addRowCategory(newCat, 3)
                finalLevel.append(newCat)
    return secondLevel, finalLevel, xlsFile
```

B-7: calculate term frequencies

```
# -*- coding: cp1252 -*-
import math
import codecs

import flickr_api_comm as flick

class thresholdValues:
    def __init__(self):
        self.thresholdTags = 0.0
        self.thresholdTitle = 0.0
        slef.thresholdLocation = 0.0
        self.thresholdDateHuman = 0.0


# return term frequency for array of terms
# tf -idd: term-frequency-inverse document frequency
def tfArray(array, word, numOfImages):
    return (freq(array,word) / numOfImages)

# return term frequency for single term
def tf(count, imagesCount):
    return count / float(imagesCount)

# return counter of given term in array
def freq(array, word):
        for i in array:
            if i.tag == word:
                return int(i.count)

# Return inverse document frequency
def idf(word, documentList, typ, docPath):
    return math.log(len(documentList) / numDocsContaining(word,documentList, typ, docPath))

# Returns nymbers of documents containing a certain words,
# only counts douments on certain path
def numDocsContaining(word, documentList, typ, docPath):
    docCount = 0
    for docName in documentList:
        f = codecs.open(docPath+docName,encoding='utf-8', mode='r')
        for i in f:
            line = i.strip('\n')
            if len(line) != 0:
                words = line.split(', ')
                a = flick.tagForm()
                a.type = words[0]
                a.tag = words[1]
                if (a.tag == word) and (a.type == typ):
                    docCount += 1
                    break

        f.close()
    return docCount
```

B-8: Find representatives

```
# -*- coding: cp1252 -*-

import re
import os
import math
import codecs

# Repository imports
import flickr_api_comm as flick
import DBHandler
import cleanser
import calculateFrequencies as calcFreq

import hierarchicalLevelHandler as hierHandler
import createXlsFile as xls

global thresholdTags
global thresholdTitle
global thresholdLocation
global thresholdDateHuman
global thresholdWeekday
global thresholdMonth
global thresholdSeason
global xlsFile


# sorts array of unique words and prints it out
# Arguments:
# Array: list of unique words
# typ: Type of list; categories list or ordinary
def printUnique(array, typ):

    sortedList = []
    temp = array
    # Sort unique tags reverse on count element
    sortedList = sorted(temp, key=lambda tag: tag.count, reverse=True)
    array = sortedList

    for elem in array:
        if elem.count > 0:
            if typ == 'cat':
                print elem.type + ', ' + elem.tag + ', ' + str(elem.count)
                      + ' '+ str(elem.subs) + ', '+ str(elem.tf)
            elif typ == 'ord':
                print elem.type + ', ' + elem.tag + ', ' + str(elem.count)
                      + ', '+ str(elem.tf)


# Count total word count from array of word frequencies
def wordCount(array):
    count = 0
    for i in array:
        if i.count > 1:
          count += i.count
    return count


# Print list of array
def printAllLists(array, typ, header, sortBy):
    # REMOVED FROM APPENDIX
    #......

# Separted date list into one list for each type
def seperateDateList(dateArray):
    weekDay = []
    month = []
    season = []
    dateHuman = []
    for i in dateArray:
        if i.type == 'weekday':
            weekDay.append(i)
        elif i.type == 'month':
```

```
            month.append(i)
        elif i.type == 'season':
            season.append(i)
        elif i.type == 'dateHuman':
            dateHuman.append(i)
    return weekDay, month, season, dateHuman

# Calculate TF and IDF for given List
def getTFandIDFforArray(array, docList, docPath, dataType, numOfImages):
    global thresholdTags
    global thresholdTitle
    global thresholdLocation
    global thresholdDateHuman

    # Calculate for all that is one third of threshold
    # (code word categories)
    if dataType == 'tags':
        threshold = (thresholdTags / float(3))
    elif dataType == 'title':
        threshold = (thresholdTags / float(3))
    elif dataType == 'location':
        threshold = (thresholdLocation / float(3))
    elif dataType == 'dateHuman':
        threshold = (thresholdDateHuman / float(3))
    else:
        threshold = 0.000000001
    # Calculate idf
    for i in array:

        termFreq = calcFreq.tf(i.count, numOfImages)
        i.tf = termFreq
        if i.tf >= threshold:
           i.idf = calcFreq.idf(i.tag, docList, i.type, docPath)
           if i.idf == float(0.0):
               i.idf = float(1.0)

# Update database regular metadata
def updateDataBaseRegular(dataList, threshold, DBObj):
    DBObj.db.begin()
    DBObj.insertListOfVariablesRegularForm(dataList, threshold)
    DBObj.db.commit()
# Update database hypernym metadata
def updateDataBaseHypernym(dataList, termList, level, DBObj):
    DBObj.db.begin()
    DBObj.insertListOfVariablesCategoryForm(dataList, termList, level)
    DBObj.db.commit()

# Find representative description of image collection
def findReps(locList, dateList, tagsList, titleList, weatherList,
             windList, tempList, docPath, numOfImages, collectionName):
    global thresholdTags
    global thresholdTitle
    global thresholdLocation
    global thresholdDateHuman
    global thresholdWeekday
    global thresholdMonth
    global thresholdSeason
    global xlsFile

    time_k = 2/float(5)
    non_fixed_k = 0.05
    dbOK = False

    # Check if Database correctly set up
    try:
         # Initialize database
         DBComm = DBHandler.dataBaseCommunicator("localhost", "root", "password?",
                                                 "database?", collectionName)
         DBComm.setCollectionId(collectionName)
    except:
         print "Error locating Database. Will continue without inserting data in database"
         dbOK = False


    # Seperate dateList inte weekend, month, season, year, datehuman
    weekDayList, monthList, seasonList, dateHumanList = seperateDateList(dateList)
```

```
numberOfTagsWords = float(wordCount(tagsList))
numberOfTitleWords = float(wordCount(titleList))
numberOfTitleWords = float(wordCount(titleList))
numberOfLocWords = float(wordCount(locList))
numberOfDateHumanWords = float(wordCount(dateHumanList))


thresholdTags = non_fixed_k
thresholdTitle = non_fixed_k
thresholdLocation = non_fixed_k
thresholdDateHuman = non_fixed_k
thresholdWeekday = ((1/float(7)) + time_k*(1/float(7)))
thresholdMonth = ((1/float(12)) + time_k*(1/float(12)))
thresholdSeason = ((1/float(4)) + time_k*(1/float(4)))



print "Cleansing data"
titleList = cleanser.removeWords(titleList)
tagsList = cleanser.removeWords(tagsList)

# Update thresholds
numberOfTagsWords = float(wordCount(tagsList))
numberOfTitleWords = float(wordCount(titleList))
thresholdTags = non_fixed_k
thresholdTitle = non_fixed_k

# Find categories/ "category of words" for Tags
tagsList, catListTags = hierHandler.findCategories(tagsList, 'tags')
print 'recreate count from categories and categorize terms'
catListTags = hierHandler.reCreateCountFromCategories(tagsList, catListTags)

print 'checking categories in word list'


print 'done Tags'

# get all frequency docs, used to calculate idf
dirAll = os.listdir(docPath)
docList = []
for fname in dirAll:
    if 'freqTags' in fname:
        docList.append(fname)

print 'Calculate tf and idf for Location'
getTFandIDFforArray(locList, docList, docPath, 'location',
                            numOfImages)
print 'done'
print ''
print 'Calculate tf and idf for weekday'
getTFandIDFforArray(weekDayList, docList, docPath, 'weekday',
                            numOfImages)
print 'done'
print ''
print 'Calculate tf and idf for month'
getTFandIDFforArray(monthList, docList, docPath, 'month', numOfImages)

print 'done'
print ''
print 'Calculate tf and idf for season'
getTFandIDFforArray(seasonList, docList, docPath, 'season',
                            numOfImages)
print 'done'
print ''
print 'Calculate tf and idf for dateHuman'
getTFandIDFforArray(dateHumanList, docList, docPath, 'dateHuman',
                            numOfImages)
print 'done'
print ''
print 'Calculate tf and idf for tags'
getTFandIDFforArray(tagsList, docList, docPath, 'tags', numOfImages)
print 'done'
print ''
print 'Calculate tf and idf for title'
getTFandIDFforArray(titleList, docList, docPath, 'title', numOfImages)
print 'done'
print ''
```

```
 print 'Calculate tf and idf for weather'
 getTFandIDFforArray(weatherList, docList, docPath, 'weather', numOfImages)
 print 'done'
 print ''
 print 'Calculate tf and idf for wind'
 getTFandIDFforArray(windList, docList, docPath, 'wind', numOfImages)
 print 'done'
 print ''
 print 'Calculate tf and idf for temperature'
 getTFandIDFforArray(tempList, docList, docPath, 'temperature', numOfImages)
 print 'done'
 print ''
 print 'Calculate tf and idf for hyp tags'

 # Find tf for hypernym lists
 for i in catListTags:
     for sub in i.subs:
         for k in tagsList:
             if k.tag == sub:
                 i.tf += k.tf
                 i.idf += k.idf
                 break

 print 'done'


# Update database Regular metadata
if dbOK == True:
      updateDataBaseRegular(locList, thresholdLocation, DBComm)
      updateDataBaseRegular(weekDayList, thresholdWeekday, DBComm)
      updateDataBaseRegular(monthList, thresholdMonth, DBComm)
      updateDataBaseRegular(seasonList, thresholdSeason, DBComm)
      updateDataBaseRegular(dateHumanList, thresholdDateHuman, DBComm)
      updateDataBaseRegular(tagsList, thresholdTags, DBComm)
      updateDataBaseRegular(weatherList, thresholdTags, DBComm)
      updateDataBaseRegular(windList, thresholdTags, DBComm)
      updateDataBaseRegular(tempList, thresholdTags, DBComm)

      # Update database Hypernym metadata level 1
      updateDataBaseHypernym(catListTags, tagsList, 1, DBComm)

# initialize excel file object
xlsFile = xls.xlsCreator(collectionName + ".xls")

printAllLists(locList, 'ord', 'Location', sort)
printAllLists(weekDayList, 'ord', '--Date: Weekday--', sort)
printAllLists(monthList, 'ord', '--Date: month--', sort)
printAllLists(seasonList, 'ord', '--Date: season--', sort)
printAllLists(dateHumanList, 'ord', '--Date: year /month--', sort)
printAllLists(tagsList, 'ord', '--Tags--', sort)
printAllLists(titleList, 'ord', '--Title-', sort)
printAllLists(weatherList, 'ord', '--Weather--',sort)
printAllLists(windList, 'ord', '--Wind--',sort)
printAllLists(tempList, 'ord', '--Temperature--',sort)
printAllLists(catListTags, 'cat', 'cats Tags',sort)


# Find new hypernym levels
secondLevelCategories, finalLevelCategories, xlsFile = hierHandler.produceNewCatLevels(tagsList, catListTags,
                                                                                       numOfImages,
                                                                                       thresholdTags,
                                                                                       xlsFile)

# Write data to xlsfile
xlsFile.writeFile()

# Update database Hypernym metadata level 2
if dbOK == True:
      updateDataBaseHypernym(secondLevelCategories, tagsList, 2,DBComm)
      # Update database Hypernym metadata level 3
      updateDataBaseHypernym(finalLevelCategories, tagsList, 3,DBComm)

      DBComm.db.close()
```

B-9: Creation of excel file

```
CREATE XLS FILE

from xlwt import Workbook

# Handles creation of .xls file containing representaitve metadata from
# collection summary
class xlsCreator:
    def __init__(self, filename):
        self.filename = filename.replace('"', '') # if filename as full string with ""
        self.curRow = 0
        self.curCol = 0
        self.book = Workbook(encoding='utf-8')
        self.sheet1 = self.book.add_sheet('Sheet 1')

    # Add rows for ordinary metadata (not hypernyms)
    def addRowOrd(self, elem):
        self.sheet1.write(self.curRow, 0, elem.type.encode('utf8'))
        self.sheet1.write(self.curRow, 1, elem.tag.encode('utf8'))
        self.sheet1.write(self.curRow, 2, elem.tf)
        self.sheet1.write(self.curRow, 3, elem.count)
        self.curRow +=1

    # Add rows for hypernym metadata
    def addRowCategory(self, elem, level):
        self.sheet1.write(self.curRow, 0, elem.type.encode('utf8'))
        self.sheet1.write(self.curRow, 1, level)
        self.sheet1.write(self.curRow, 2, elem.tag.encode('utf8'))
        self.sheet1.write(self.curRow, 3, elem.tf)
        self.sheet1.write(self.curRow, 4, elem.count)
        subsString =''
        self.curCol = 5
        for sub in elem.subs:
            subsString +=sub + ', '
            self.curCol+=1
        self.curCol = 0
        self.sheet1.write(self.curRow, 5, subsString)
        self.curRow+=1

    # Create file in root folder of running application
    def writeFile(self):
        self.book.save(self.filename.replace(':','!'))
```

B-10: Communicate with database (mysql)

```
INSERT INTO DB

# -*- coding: cp1252 -*-
import MySQLdb


# Repository files
import flickr_api_comm as flick
import findRepresentatives


# Handles communication (updates and insertions) to mysql database
# metadata used by system are stored
class dataBaseCommunicator:

    # Initialize database object and local variables
    def __init__(self, host, user, passwd, database, collName):
        self.db = MySQLdb.connect(host=host, user=user, passwd=passwd,
                                  db=database, use_unicode=True,
                                  init_command='SET NAMES utf8')
        self.cursor = self.db.cursor()
        self.cursor.execute(""" SET CHARACTER SET utf8 """)
        self.cursor.execute(""" SET character_set_connection=utf8 """)
        self.host = host
        self.user = user
        self.passwd = passwd
        self.dbInUse = database

        self.collId = self.setCollectionId(collName)

        self.deleteCollectionInfoIfAllreadyInDB()

    # Gather collection id from table "collection"
    def setCollectionId(self, collectionName):
        self.cursor.execute("""SELECT collId FROM collection WHERE collName = %s""",
                            (collectionName,))
        retTuple = self.cursor.fetchone()
        return retTuple[0]

    # if data allready present in db for collection, make clean
    def deleteCollectionInfoIfAllreadyInDB(self):
        print "delete-----"
        self.db.begin()
        deleteCount=self.cursor.execute("""delete ut.*, i.* from uniqueTerm as ut, includes
                                           as i where i.collId=%s""", self.collId)
        self.db.commit()
        print deleteCount

    # Handles insertions for all ordninary metadata (not hierarchcal
    # levels of hypernyms)
    def insertListOfVariablesRegularForm(self, array, threshold):

        # Process all elements (terms)
        for elem in array:
            utCount = int(self.cursor.execute("""select * from uniqueTerm where utId = %s""",
                                              (elem.tag.encode('utf8'),)))
            if utCount == 0:
                try:
                    self.cursor.execute(""" INSERT INTO uniqueTerm (utId) VALUES (%s)""",
                                        (elem.tag.encode('utf8')))
                except:
                    print "data to long for insertion: "+elem.tag + "  TF: " +str(elem.tf)
            try:
                self.cursor.execute(""" INSERT INTO includes (collId, utId, utType, utCount,
                                        utTF, utIDF) VALUES (%s, %s, %s, %s, %s, %s)""",
                                        (self.collId, elem.tag.encode('utf8'), elem.type,
                                         elem.count, elem.tf, elem.idf))
            except:
                # some translation errors between utf8 in python and mysql
                # e.g. 'è' and 'e' are both converted to 'e'
                # before insertion
                print "error insert duplicate: " + elem.tag
```

```
        # Update database in acordance with images related to
        # specific element (term)
        for image in elem.imageIds:
            count = int(self.cursor.execute("""select * from image where imageId = %s""",
                                            (str(image),)))
            if count == 0:
                self.cursor.execute(""" INSERT INTO image (imageId, location) VALUES
                                        (%s)""", (str(image), "not set"))

            self.cursor.execute(""" INSERT INTO contains (collectionId, imageId) VALUES
                                    (%s, %s)""", (int(self.collId), str(image),))
            self.cursor.execute(""" INSERT INTO TaggedWith (utId, imageId) VALUES
                                    (%s, %s)""", (elem.tag.encode('utf8'), str(image),))


        # Update database in acordance with hypernyms (level 1)
        # related to specific element (term)
        for category in elem.categories:
            # check if hypernym allready in db
            count = int(self.cursor.execute(""" select hyperName from hypernym
                                                where hyperName=%s""", (category,)))
            # Not in db (hypernym)
            if count == 0:
                self.cursor.execute(""" INSERT INTO hypernym (hyperName) values (%s)""",
                                    (category.encode('utf8'),))

                self.cursor.execute(""" INSERT INTO groupedInto (collId, hypId,
                                        hierLevel) VALUES (%s, %s, %s)""",
                                        (self.collId, category.encode('utf8'),1,))
                self.cursor.execute(""" INSERT INTO memberOf (collId, utId,
                                        hypId,hierLevel) VALUES (%s, %s, %s, %s)""",
                                        (self.collId,elem.tag.encode('utf8'),
                                         category.encode('utf8'),1,))
                catHandlerId = self.db.insert_id()


            # allready in db (hypernym)
            elif count == 1:

                countGroupedInto = int(self.cursor.execute(""" select hypId from
                                                           groupedInto where collId=%s and hypId=%s and
                                                           hierLevel=%s""", (self.collId, category,1,)))
                if countGroupedInto==0:
                    self.cursor.execute(""" INSERT INTO groupedInto (collId, hypId,
                                            hierLevel) VALUES (%s, %s, %s)""",
                                            (self.collId, category.encode('utf8'),1,))
                self.cursor.execute(""" INSERT INTO memberOf (collId, utId,
                                        hypId,hierLevel) VALUES (%s, %s, %s, %s)""",
                                        (self.collId, elem.tag.encode('utf8'),
                                         category.encode('utf8'), 1,))
                catHandlerId = self.db.insert_id()

# Handles insertions for all metadata for hierarchical levels
# Hypernym
def insertListOfVariablesCategoryForm(self, arrayCat, arrayAllTerms, level):
    # Level 1 only update database with TF and IDF
    if level == 1:
        for elem in arrayCat:
            self.cursor.execute(""" UPDATE groupedInto gi, hypernym h, memberOf mo,
                                    collection c set gi.hypTF=%s, gi.hypIDF=%s
                                    where gi.hypId=%s and gi.collId=%s and gi.hierLevel=%s
                                    and h.hyperName=gi.hypId and mo.hypId=h.hyperName
                                    and c.collId=gi.collId""",
                                    ( elem.tf, elem.idf, elem.tag.encode('utf8'), self.collId, level,))
    # For higher levels
    else:
        for elem in arrayCat:
            for term in elem.subs:
                # count for hypernyms returned from db
                count = int(self.cursor.execute(""" select hyperName from hypernym
                                where hyperName=%s""", (elem.tag,)))
                # Not in db (hypernym)
                if count == 0:
                    self.cursor.execute(""" INSERT INTO hypernym (hyperName) values
                                            (%s)""",(elem.tag.encode('utf8'),))
```

```
        self.cursor.execute(""" INSERT INTO groupedInto (collId, hypId,
                                 hierLevel, hypTF, hypIDF) VALUES
                                (%s, %s, %s, %s, %s)""",
                                (self.collId, elem.tag.encode('utf8'),
                                  level,elem.tf,elem.idf,))
        self.cursor.execute(""" INSERT INTO memberOf (collId, utId,
                                hypId,hierLevel) VALUES (%s, %s, %s, %s)""",
                                (self.collId, term.encode('utf8'),
                                  elem.tag.encode('utf8'),level,))

    # allready in db (hypernym)
    elif count == 1:
        countGroupedInto = int(self.cursor.execute(""" select hypId from
                                    groupedInto where collId=%s and hypId=%s
                                    and hierLevel=%s""",
                                    (self.collId, elem.tag.encode('utf8'),
                                      level,)))
        if countGroupedInto==0:
            self.cursor.execute(""" INSERT INTO groupedInto (collId, hypId,
                                   hierLevel, hypTF, hypIDF) VALUES
                                   (%s, %s, %s, %s, %s)""",
                                   (self.collId, elem.tag.encode('utf8'),
                                    level,elem.tf, elem.idf,))
        self.cursor.execute(""" INSERT INTO memberOf (collId, utId,
                                 hypId,hierLevel) VALUES (%s, %s, %s, %s)""",
                                  (self.collId,term.encode('utf8'),
                                     elem.tag.encode('utf8'), level,))
```