A Context-based Approach to Reduce the Amount of Unknown Words in User Search Queries

Rixt M. Hielkema

March 2010

Master Thesis
Human-Machine Communication
Department of Artificial Intelligence
University of Groningen, The Netherlands

Internal Supervisor:
Dr. Jennifer Spenader (Artificial Intelligence, University of Groningen)

External Supervisor:
Dr. Begoña Villada (Professional Services, Q-go Dieme

Abstract

Unknown words lead to bad answers in Question Answering. Next to traditional Question Answering challenges, web search applications face another challenge when dealing with unknown words: they have to deal with the language found in user queries. User queries are frequently ungrammatical, have a telegram style and contain misspelled words, all of which make their automatic interpretation very difficult. Although little research has been done on finding unknown words in user search queries, we will show that this is a valuable goal.

Q-go is a company specialized in natural language search, but like most search engines, it has difficulty interpreting unknown words when these appear in user search queries. We investigated how often unknown words occur and also which lexical types are responsible for the unknowns Q-go encounters in three domains. This has provided a useful and unique insight in unknown words in a real world application. Many projects do not take the time to analyze the data manually.

The manual analysis showed that the majority of unknown words in the data are named entities (28.4%), so further research concentrated on the identification and semantic classification of named entities in user search queries.

Two context-based approaches to reduce the number of named entities were compared, Paşca (2007) and Pennacchiotti and Pantel (2006). Starting from a small set of seed entities we extract candidates for various classes of interest to web search users. We experimented with multiple parameters, similarity metrics and the three domains among others. The most promising results were obtained with the approach based on Paşca (2007). The productivity of a class seems to be the factor most predictive of success in finding new named entities of the correct semantic class. Further, the approach of Paşca (2007) was able to successfully deal with user query language, also an important result.

Acknowledgements

I would like to thank Begoña Villada for guiding me on the trip taken into the world of unknowns. On the one hand the unknown words and on the other hand the unknown issues in research. Furthermore, I would like to thank the people at Q-go, specifically the research department, that made me feel welcome and provided all the help I needed with the project.

I would also like to thank Jennifer Spenader for her valuable advice on the thesis. Even while being on maternity leave there was always an everlasting enthusiasm about this project which kept me motivated.

Many thanks to Johan, for listening, reading and patience and also to my friends for forcing me to relax from time to time.

Last but certainly not least, I would like to thank my family for always supporting me.

Table of Contents

1. Introduction

"Can I fly from Zixi to Amsterdam tomorrow?" This is a typical example of the kind of information we want to retrieve from the Web nowadays. We ask questions and the Web gives us the right answer.

Actually, this is what would happen in an ideal world. I expect that all readers of this thesis could effortless read the first sentence of this introduction and even understood what was meant. I also expect that probably none of you has ever heard of 'Zixi', a town in China. Why did this unknown word not prevent you from understanding the question? Because you were able to deduce what was meant. 'Fly from x to y' indicated that 'Zixi' probably would be a geographical location. Since 'y' is given and most people do know that Amsterdam is a location, the evidence that 'x' is also a location is even stronger. And by simply looking at the form of the word you acquired information. 'Zixi' starts with a capital letter which is a property of how most locations are written down. After having used these heuristics we still cannot tell that 'Zixi' is a town in China, but that information is not needed to understand what is meant. All these processes go on in our heads and we hardly notice them.

If a machine was reading this thesis there is a big chance that it would not be able to read the first sentence because the system has never heard of 'Zixi'. Although most Natural Language Processing (NLP) applications make use of extensive linguistic resources, it is inevitable that the applications come across words not available in the linguistic resources. There are several reasons for this. For example, there may be gaps in the resources, e.g. the word 'cockpit' missing in a dictionary specific for aviation. Also, new words are created all the time, e.g. 'blog' became a word in the late 1990s. Often proper nouns are missing in the dictionaries since it is an open class, e.g. 'Barack Obama' would be not in dictionaries in the 1970s, even though he already existed, but it is only recently that he became 'known' to the world. Another source of unknown words, are words that contain typos, e.g. 'fligt' instead of 'flight'. And as a last example, one can often refer to one object in many ways or to different objects in the same way, e.g. 'New York' 'NYC' and 'the Big Apple', all can refer to 'New York', the first example can refer to the state or the city, the second example refers to the city and the third example can refer to the city as a whole, a more specific a part of the city, Manhattan but also to just an apple that is large. It can be difficult to think of all the variations and have them all in the linguistic resources.

Unknown words lead to problems for many NLP applications. For instance machine translation becomes hard, if not impossible. If the machine has to translate an unknown word, the machine does not know to which word it must be translated and might leave it just not translated. In some cases this is the correct way of handling the word, e.g. 'Microsoft' or 'Balkenende' do not need translation. In most cases however, it is not best to keep the word not translated. Unknown words also cause failing syntactic analyses (parses) since the unknown word does not have a word category (e.g. 'noun', 'verb', 'adjective'). Speech recognition will fail if an unknown word is said and lastly, the success of Question Answering systems will decrease.

1.1. Research Questions

Currently, there are no established methods for identifying and dealing with unknown words. There are methods which look at the morphological features of a word (Mikheev, 1997; Tseng, Jurafksi and Manning, 2005) and methods which look at contextual cues (Van de Cruys, 2006; Paşca, 2007; Pennacchiotti & Pantel, 2006). They all have a different

goal with resolving unknown words, Mikheev (2007), Van de Cruys (2006) and Tseng, Jurafksi and Manning (2005) want to assign unknown words the right word category to improve parsing, Paşca (2007) wants to expand dictionaries with named entities and Pennacchiotti and Pantel (2006) are looking for binary semantic relations.

One thing that most work on unknown words has in common is that the work is done on edited document collections: collections containing grammatical, well-formed sentences, 'perfect' text. However, edited text gets sparser while the access to unedited text gets easier. Many pages on the Web are filled with easy accessible, unedited text. People make for instance their own websites, they write blogs or even are allowed to write parts of an encyclopedia (e.g. Wikipedia[1]). The everyday user of the Web is providing document collections. However, unedited text contains errors that real people make, think of spelling errors and abbreviations, 'imperfect' text. So a shift must be made from systems processing 'perfect' text to systems that are also able to process 'imperfect' texts.

The examples of unedited text mentioned above all contain running text. An alternative to running text is to use user search query data. Nowadays, if we are looking for information we often use web search applications. We enter a word or a sequence of words and we expect to get useful information. User search queries make up a lot of 'web text'. 'Google' for instance handles more than 235 million user search queries per day. Even though there is so many user search query data, it is hardly used to find unknown words in. However, Paşca (2007) has used such user search queries and has shown that it can be highly valuable.

In this work we have tried to reduce the number of unknown words encountered by a Question Answering system at a Dutch language technology company, Q-go. Q-go provided datasets to get information about the unknown words and datasets to evaluate methods on, datasets containing user queries. The research consists of two parts. On the one hand, we perform a data analysis. We investigated how big a problem unknown words are, when having a large, commercial dictionary available. Specific to this project, we wish to determine how big a problem unknown words are for Q-go.

Often such a data analysis is not performed on unknown words. In many projects on unknown words the focus is on one particular type of unknown words without checking how big the problem of the particular type is. With this analysis we can exactly see how many unknown words and what types of unknown words there are. It also tells us what kinds of unknown words occur in real world applications and how for instance a spell correction module handles actual unknown words, i.e. unknown words that are spelled correctly and should be in the linguistic resources.

On the other hand, we implemented two methods to deal with unknown words and evaluated how well the methods work. In short, the research specifically went through the following steps:

1. How big is the problem of unknown words within Q-go and does this lead to poor matching of user queries with responses?

To be able to develop a method to reduce the amount of unknown words we need to know what sort of unknown words are causing the most problems. Therefore the second research question is:

[1] www.wikipedia.com

2. What types of unknown terms are there and how often do they occur?

Based on those results the third research question can be posed:

3. What approaches can be applied to reduce the most frequent type of unknown words?

1.2. Structure of the thesis

In Chapter 2, we first describe the theoretical background of unknown terms; methods that have been applied already and the extent to which they are successful. Moreover we will discuss which issues we have to take into consideration when choosing an approach, since this is what we have to do eventually. Next to this, we will describe in more detail the type of textual data we will use to study the problem: user query language, i.e. actual user queries which are grammatically not so well-formed as running text. Last, we will describe briefly how Question Answering systems work since we receive data from a Question Answering system to perform an analysis on and to reduce the amount of unknown words in.

Following in Chapter 3, a data analysis is presented performed on the search query logs Q-go has provided. This data analysis will give insight in how often unknown words occur, what types of unknown words there are and how often each type occurs. From this chapter it follows that named entities are the most frequent unknown words so we will investigate this type in more detail.

In Chapter 4, we describe which requirements our solution should have to reduce the number of unknown named entities. We will discuss several approaches which reduce the amount of named entities. Two of them are described in more detail, the approach described in Paşca (2007) and the approach described in Pennacchiotti and Pantel (2006). Both approaches use lexico-syntactic patterns. To test the approach, Paşca (2007) uses user query language, Pennacchiotti and Pantel (2006) edited running text. Moreover Paşca (2007) is looking for named entities, Pennacchiotti and Pantel (2006) for binary semantic relations. These are approaches which meet all requirements and we will base our two own approaches upon.

Chapter 5 describes the two approaches we have implemented to reduce the number of unknown words: the Paşca approach, a customized approach based on Paşca (2007), and the Espresso approach, a customized approach based on Pennacchiotti and Pantel (2006). Both approaches use lexico-syntactic patterns to mine user query data for named entities but they differ in how the strength of lexico-syntactic patterns and named entity candidates are computed. The Paşca approach uses a vector-based model to compute similarity, the Espresso computes the reliability based on pointwise mutual information (pmi) as proposed in Pennacchiotti and Pantel (2006). We will also describe the experimental setting we have used to test the approaches with, the testing procedures per approach and the evaluation procedure. We will end with what the differences are between the two approaches.

In Chapter 6, we will describe and discuss the results of our two approaches. Since manual inspection of the results of the Espresso approach has shown that further evaluation would not be promising, we will only describe the results of the Paşca approach in more detail. For some classes (e.g. product name) and domains (e.g. aviation), the Paşca approach gives good results. We find that in our experiments increasing the data, number of seeds or number of lexico-syntactic patterns does not influence the results. The success of the method seems to be determined by the productivity of a class for a domain. Since we have

performed testing on datasets provided by Q-go, we will also discuss the generality of the result to other applications.

Finally, Chapter 7 summarizes the whole process and provides ideas for future work.

2. Background

2.1. The challenge of unknown words

Previous research on unknown words claims that natural language parses often fail due to words missing in lexicons and thesauri (Baldwin, Bender, Flickinger, Kim and Oepen, 2004) (Ciaramita and Johnson, 2003). This means that unknown words affect and decrease the robustness of a syntactic parser. Robustness is an important measure of success for NLP applications and especially for applications that are interacting with real users in real time. Users want satisfying results. If the application does not give a satisfying result the user will consider the application as unsuccessful.

There are two ways to deal with unknown words. One option is to manually maintain the lexical resources. The other option is to detect, categorize and deal with the unknown words automatically.

Manually maintaining the resources is not the best option. It is an expensive, labor-intensive option. Despite large improvements most lexicons are still quite incomplete. New words are added to language every day, for instance named entities.

Unknown words are often named entities i.e. entities with the word category 'proper name', like product names, personal names, company names and movie titles. This is due to the productivity of the named entity class, every day new names, companies, book titles originate and it is difficult to keep track of all of them. A lot of literature discussed later on is therefore specifically about named entities.

Moreover, there are also many words not in the dictionary because they have not been used before. This does not necessarily mean that the words are new words. Those words can be general words belonging to the vocabulary, but for instance low-frequency words.

Detecting, categorizing and dealing with unknown words automatically is a better option. This can be done by implementing rule-based methods based on human intuition. But then still humans are involved and it might happen that they overlook rules. Besides, certain rules may work well for one domain but that is no guarantee that they will return good results in other domains as well. This would mean that for every domain the rule database has to be rewritten. For these reasons machine learning techniques are preferred; techniques that automatically learn to recognize complex patterns and make intelligent decisions based on empirical data.

There are three main approaches possible to tackle unknown words by using machine learning techniques: approaches based on context, on content or on a combination of these two. We will discuss several methods for each approach in the following three sections. Next to an overview of what research has been done on unknown terms, it shows that there is not one perfect solution to handle unknown words. Which method works, depends on the goal. Therefore, after these three sections we will discuss the issues one has to take into consideration when choosing an approach to reduce the amount of unknown words. At that stage, it is not clear yet for each addressed issue what the best choice is for this project. Therefore some issues will remain open questions and will be answered further on in the thesis in Chapter 3.

2.1.1. Content-based approaches

Content-based approaches rely on the internal properties of a word to identify its type or meaning. Internal properties can be morphological features, but also orthographic features.

There are two types of morphology: inflectional morphology and derivative morphology. Inflectional morphology refers to the modification of a word to express different grammatical categories, like number (root: *dog*, singular: *dog*, plural:*dog-s*) or person (root: *love*, first person: *love*, second person: *love*, third person: *loves*). The word category, i.e. Part of Speech (PoS), and the meaning of the word, do not change.

Derivative morphology refers to the modification of a word to form new words, *happi-ness* and *un-happy* are derived from the root *happy*. The PoS and/or meaning changes. *Happy* is an adjective, by adding the prefix '*un-*', the antonym is created: the meaning has changed, the PoS is the same. By adding the suffix '*-ness*' both meaning and PoS have changed (from adjective to noun).

There are several PoS-taggers which use this morphological information to assign a PoS to a word. The Brill tagger (Brill, 1995) is one of the first PoS taggers which automatically induce rules to learn morphological rules. It describes an accuracy of 99.1% in automatically providing words with a PoS. Brill (1995) was one of the pioneers in this area.

After Brill (1995) a lot of research is done on resolving unknown words by looking at morphological features. Mikheev (1997) for example, has developed a method to automatically guess possible PoS tags by means of affixes (both suffixes and prefixes) of words. The learning is performed on a raw English corpus. Three kinds of rules have been statistically induced: prefix morphological rules, suffix morphological rules and ending-guessing rules. From these three types of rules unknown-word-guessing rules have been derived. These rules were then integrated in a stochastic and a rule-based tagger, which were then applied to texts with unknown words. On tagging the unknown words, an accuracy of 87.7%-88.7% is achieved.

Tseng, Jurafsky and Manning (2005) also present a method based on affix recognition. The words to tag with the correct PoS are unknown Mandarin Chinese words. The challenge here is that Mandarin Chinese uses a lot of frequent and ambiguous affixes, i.e. encountering an affix is not such a strong indicator for a specific PoS as affixes in English are (Brants, 2000). Tseng, Jurafsky and Manning (2005) show therefore a variety of new morphological unknown-word features which extend earlier work on other languages. The accuracy of tagging unknowns varies between 61% and 80%.

Adler, Goldberg, Gabay and Elhadad (2008) also propose a method to assign a PoS to unknown words based on morphology. Their method is not based on an annotated corpus or on handcrafted rules. Instead the known words in the lexicon are used. The letter similarity of the unknown words they find are compared to the known words with a maximum entropy letter model. The approach is tested on Hebrew and English data. Adler, Goldberg, Gabay and Elhadad (2008) report an accuracy of 85% on Hebrew unknown words and an accuracy of 78% on English unknown words.

These examples show that although the results can vary per language, PoS tagging methods based on morphology show promising results when dealing with unknown words.

The above mentioned approaches have in common that solving the PoS means solving the unknown words. In the field of NLP it is for some applications, like Question Answering applications, also important to add a semantic interpretation to the unknown word. A word can hold important information necessary for returning the correct response. To answer a question such as "What kind of flowers did Van Gogh paint?" it is necessary to know that a sunflower is a type of flower (Paşca and Harabagiu, 2001). Knowing that 'flower' is a noun in case of the PoS taggers or knowing that 'Van Gogh' is a person is not enough.

Woods, Bookman, Houston, Kuhns, Martin and Green (2000) have developed a content-based method to assign a semantic interpretation to unknown words. The goal of the method is to improve online search effectiveness. The method uses a lexicon containing syntactic, semantic, and morphological information about words, word senses, and phrases to provide a base source of semantic and morphological relationships that are used to organize a taxonomy. In addition, it uses an extensive system of knowledge-based morphological rules and functions to analyze the unknown words, in order to construct new lexical entries for previously unknown words. Because they link the unknown words to known words in the taxonomy the semantic interpretation can be given by using the knowledge about the related words in the taxonomy. When adding the unknown word module an improvement of search effectiveness is reported from 50% to 60.7%.

In the above mentioned literature the morphological features are inflectional and derivational cues: the word is split in its root and its morphological features, the root holds the meaning and the morphological features can change its meaning, the PoS or nothing. However, morphological features can also be parts of words which on itself indicate the meaning of the word without needing the meaning of the root. Some Named Entity classes for instance contain such morphological features; "Microsoft", "IPsoft" and "Lavasoft" all share the 'word' "soft" which indicates that the words are companies dealing with software. Micro, IP and Lava are words but not the roots of the word. Systems that learn affixes themselves without the need of knowing the root of a word are automatically able to gain knowledge of such features.

Such a system is presented in Cucerzan and Yarowski (1999). They attempt to recognize named entities by using among others the morphological features mentioned before. Therefore, it is not exactly clear to what extent these specific features are useful. However, the method uses a trie model to find these morphological features. Trie models are also known as prefix trees, a well-established data structure for compactly storing sets of strings that have common prefixes. They report an accuracy of 73% to 79% for five different languages.

Orthographic features such as capitalization or mixtures of digits and characters are the other source of content-based information. Unknown words as codes or named entities often show specific orthographic features. By codes we mean a set of letters that gives information about something, for example about a personal reservation for a hotel. Codes often follow a specific pattern, e.g two random characters followed by four random numbers can be the personal reservation code a hotel gives to their customers.

Meulder and Daelemans (2003) have used orthographic features along with other features to extract named entities. Their memory-based method classifies named entities based on the features learned in a training set. Since their method also uses other features it is not clear what the exact effect is of the orthographic features. However, they report a precision of 76% when testing the method on un-annotated English data and a precision of 64% when testing the method on un-annotated German data.

2.1.2. Context-based approaches

Context-based approaches use contextual information. The most common approaches are approaches based on lexico-syntactic patterns. These approaches have the advantage that they can add semantic information immediately without being obliged to look at the morphological structure. For instance if a word occurs in the pattern 'X such as Y', this can indicate that 'Y' is a kind of 'X', 'X' is the super sense of 'Y'.

Approaches based on lexico-syntactic patterns are often used for expanding thesauri (Hearst, 1992), dictionaries (Rilof & Jones, 1999) or named entity gazetteers (Kozareva, 2006). A gazetteer is a dictionary containing entities of a specific class, e.g. a bird gazetteer contains bird names. With more words in the resources, more words will be known to the system and therefore fewer unknown words will be encountered.

A well-known approach based on lexico-syntactic patterns is the bootstrapping approach of Hearst (1992). Hearst (1992) was looking for the hypernym-hyponym relation (e.g. *cat, dog* and *fish* are hyponyms of *animal*, *animal* is the hypernym of *cat, dog* and *fish*) to be able to automatically expand thesauri. This approach was however not fully automatic. The lexico-syntactic patterns were built manually. When having determined the lexico-syntactic patterns, new candidates of the relation were found automatically by using the defined lexico-syntactic patterns. With the newly found candidates, new patterns could be found. This process can be repeated until a satisfying amount of candidates is found.

If the relation is hypernym-hyponym, the word pair can be added to the hierarchy in the form of an 'is-a' relation. The advantage of this approach is that semantic information of the word is added by knowing the super sense of a word. Hearst (1992) compares her results with the 'is-a' relation in WordNet[2] (Miller, Beckwith, Fellbaum, Gross and Miller, 1990). From the 226 unique found words, 180 words already existed in the hierarchy. Compared to WordNet not many unknown words have been found. However, what must be kept in mind that actually all words found were unknown words since only the patterns were provided.

Bootstrapping methods have attempted to extract other kinds of relations as well. A bootstrapping algorithm that tries to find many different relations is 'Espresso' (Pennachiotti & Pantel, 2006). This algorithm tries to find semantic relations of all kind of types in corpora, e.g. 'is-a' relations, 'succession' relations and 'production' reaction relations. 'Espresso' only needs a small set of seed pairs for a particular relation. The system learns the lexical patterns in which the seeds appear. The lexical patterns are used to find new candidates for the specific relation. The best scoring candidates then can be used as new seed pairs to find more lexical patterns. The advantage of this approach is the ability to find all kinds of patterns and not only the hypernym-hyponym relation. Pennachiotti and Pantel (2006) report precision scores varying from 49.5% to 85% across the varying types of relations.

Another bootstrapping approach is the approach described in Paşca (2007). His approach is based on lexico-syntactic patterns and a vector space model. It tries to extract named entities from anonymized web search query logs Google provided. This means that it is more or less dealing with the same sort of data we will analyze and test out approaches on: user queries. Paşca (2007) is able to extract named entities per predefined class based on small seed lists without any need for handcrafted extraction patterns. The result of this approach is the creation of gazetteer lists with candidates for the specific classes they were

[2] A hand-built online thesaurus

looking for. On average, over ten classes, Paşca (2007) reports a precision of 80% over 250 extracted candidates.

Next to these methods which add words to thesauri, dictionaries and gazetteer lists there are also methods which try to classify named entities online. For instance Guo, Xu, Cheng and Li (2009) try to classify named entities in web search queries like Paşca (2007) but they predict the classification online, based on a probabilistic context pattern model. Next to the named entities they learn in an offline situation, they use the probabilities they find for certain lexico-syntactic patterns and classes to guess the right named entity class. This online prediction can be important when the named entity class is immediately needed, for instance to be able to answer a specific question like 'Harry Potter walkthrough'. 'Walkthrough' is indicating that it is about the computer game 'Harry Potter', so no information about the book must be given.

There are also context-based approaches not based on lexico-syntactic patterns. An example of a context-based approach not based on lexico-syntactic patterns but based on PoS sequences can be found in Van de Cruys (2006). Most content-based approaches discussed earlier, which assign the correct PoS to an unknown word, do not provide a semantic interpretation. The approach Van de Cruys (2006) presents also does not provides semantic interpretation. In Van de Cruys (2006) unknown words are the words which led to a bad parse because the word did not get assigned a lexical category, or did get assigned more than one lexical category. Next, more sentences are searched for, which also contain the unknown word. In all sentences all possible PoS categories are assigned to the unknown word. Then, all sentences are being parsed. Each sentence will have a best parse. The lexical category the unknown word has in the best parse will be given as output. Because there is more than one sentence seen which contains the unknown word, a lot of 'best' lexical categories will be found for one unknown word. Based on the overall best category the most probable PoS can be assigned by using a maximum entropy classifier. This method reports a precision of 77.5%.

Another example of a context-based approach to predict PoS tags can be found in Nakagawa, Kudoh and Matsumoto (2001). They predict PoS tags by using Support Vector Machines. The features they use are substrings and surrounding context. They report a precision of 97.3% on known words and a precision of 86.9% on unknown words.

PoS taggers as mentioned above can in fact be used to add a semantic interpretation to words. Bouma, Mur, Van Noord, Van Der Plas and Tiedeman (2005) have tried to add a meaning to unknown Named Entities found by their QA system "Joost". They use a PoS tagger which is able to tag unknown named entities with the PoS: proper noun. Next, each proper noun is classified into a named entity class: person, organization, geographical location or miscellaneous. Miscellaneous is a very broad category, nonetheless the gain is that the named entity is known not to be a person, an organization or a location. The resulting classifier which combines lexical look-up and a maximum entropy classifier, achieves an accuracy of 88.2%.

2.1.3. Combination: content-based and context-based

There are also approaches which combine both content and contextual cues, they are already mentioned briefly in the previous sections. Meulder and Daelemans (2003) propose a combination of content-based and context-based methods to recognize named entities. They describe a memory-based approach. This method classifies newly seen named entities on the basis of features they have learned in a training set. There are 37 features to be learned. Seven features deal with context, seven with the part of speech,

seven with capitalization, six with affixes and prefixes and ten features deal with whether the word appeared in one of the ten gazetteers they used. The input data is a pre-tagged corpus and the goal is to learn the features to be able to classify new seen named entities.

Cucerzan and Yarowski (1999) also present an approach based on the combination of morphological features and contextual cues. Their goal also is to recognize named entities. They describe a language-independent use of a bootstrapping algorithm, based on not only re-estimation of the contextual pattern but also on re-estimation of morphological patterns. To do this they make use of smoothed trie models because of efficiency, flexibility and compactness.

2.1.4. Considerations when dealing with unknown words

As can be seen a lot of work has been done on unknown words. Different purposes require different methods so several questions must be answered before it can be decided what the right approach is to deal with unknown words in a specific context.

The most important question is when an unknown word is considered resolved. If the only goal is to get a correct syntactical parse, PoS information is enough information. If semantic interpretation of the unknown word is needed, like in QA systems, meaning is more important. But what is the meaning of the word? Is 'Miscellaneous' a meaning? In case of Bouma et. al (2005) it is. But for the algorithm Espresso (Pennachiotti and Pantel, 2006) it is not. Having both PoS information and meaning, provides most information about a word, like in Meulder and Daelemans (2003). But their approach is computational very expensive, something that is not wanted in a lot of situations. Moreover they rely on a lot of resources; it can be difficult to create them properly.

It also has to be considered when to resolve unknown terms. Must they be resolved online like Guo et al (2009), so an immediate reaction can follow? Or is unknown word finding done offline to add unknown words to lexical resources like Paşca (2007) by knowing that increasing lexical resources will decrease the probability of encountering unknown words?

For online recognition precision is more important than recall. High precision is necessary because the end user gets confronted with the results. If no answer can be found by the system for a user query which contains an unknown word, it is undesirable that the user gets an answer which does not makes any sense instead of the honest message that the system could not find an answer. It is also unwanted that user queries which contain an unknown word which get a good answer, will get a wrong answer because the unknown-word module interprets the unknown word in a wrong way.

For expanding linguistic resources, recall is more important. It is not too difficult to go through a list of possible candidates manually. It is more important to get as many new instances as possible.

Another important question is what type of unknown words the system must understand. Are they words which definitely should not end up in the unknown category? Like having 'understand' in the dictionary but not being able to process 'understandable' because it is not in the dictionary? Then a morphological approach is needed. However, in some languages like Mandarin Chinese (Tseng et al, 2004) morphological cues are not as strong indicators as they are for languages such as English (Brants, 2003). Or is it needed to cover the productive class of proper nouns? Then the approaches people choose tend to rely on context more than on content.

One more important question is: “How does the data in which the unknown words occur look like?” Most of the literature we reviewed, deal with edited document collections, i.e. texts from corpora consisting of complete and grammatically well-formed sentences, sometimes even annotated with linguistic information. However, the data we will analyze and we will test our approaches on, contains unedited text, and even a specific part of unedited text: user query language, i.e. user queries containing incomplete and grammatically not so well-formed sentences. In the next section, the properties of user query language are described in more detail together with the problems they can create.

Whether the text is edited or not seems to need different approaches when dealing with unknown words. It seems that systems developed for edited text (‘perfect’ text) work best for edited text and systems developed for unedited (running) text (‘imperfect’ text) work best for unedited text. Pennachiotti and Pantel (2006) for instance compare their ‘Espresso’ algorithm, which harvests semantic relations from edited data, to an approach described in Ravichandran and Hovy (2002) on the same edited text collections. The approach in Ravichandran and Hovy (2002) however, was designed to harvest named entities from unedited running text, scraped from websites, not from edited text. The ‘Espresso’ algorithm shows much better results on edited text than the approach of Ravichandran and Hovy (2002) on edited text while Ravichandran and Hovy (2002) did report good results in their own test setting.

All these considerations are important when choosing a method. There is not (yet) one optimal solution for all unknown words.

2.2. The challenge of user query language

This research focuses on a particular type of Question Answering systems, in concrete, web search applications. The way the question is asked in the example in the first sentence of the introduction, is the most straightforward and clearest way to make sure humans will understand what you want.

“Can I fly from Zixi to Amsterdam tomorrow?”

Web search applications often ask for questions in another format. That is, a format based on keywords, the words with the important information. User queries entered in web search applications tend to be short and not well-formed (Guo et al, 2009), i.e. user queries contain user query language.

By giving users the freedom to define their own keywords this adds difficulty on the one hand for the users and on the other one for the search engine. The user must decide which words are the keywords. By choosing the wrong keywords no good answer, fewer answers or less relevant answers can be obtained. In the ideal situations this should be no problem for a search engine, in reality it is. The freedom given to the users is problematic for the search engine because it must handle user query language. The question from the first example can be asked in a lot of ways. It can be asked as in the example:

“Can I fly from Zixi to Amsterdam tomorrow?”

But there are a lot of possible variations:

1. Flight Zixi Amsterdam
2. Zixi Amsterdam tomorrow

3. From zixi to Amsterdam
4. Flight zixi adam
5. Fly jjn ams
6. zixi asterdam flight

All these seven examples may require the same answer. There are a lot of differences present which show the variability of language real people use in unedited text. Users tend to be inconsistent with capitalization. Sometimes all words are capitalized, sometimes nothing is capitalized and everything in between can happen. This is seen across all examples. People vary in choosing the keywords they think are important. In Example 1, flight is an important keyword, the user in Example 2 chose tomorrow as an important keyword. Example 3 shows a user which considers the context 'from x to y' important. Users abbreviate words. The user in Example 4 abbreviated Amsterdam to adam. Without context 'adam' will be most likely categorized as a personal name. In the example however, it is more likely to be an abbreviation of Amsterdam and therefore a geographical location. Sometimes there are more options to name a word, naming variants. In example 5, the user did not use the geographical locations but naming variants: codes which represent the airports. Next to this all, a spelling error is easily made as can be seen in example 6, Amsterdam is not spelled correctly.

These variations show that we need an approach to reduce the amount of unknown words which is able to process user query language robustly.

2.3. Question Answering

Question Answering is the task of returning a relevant answer to the question a user poses. Questions can be posed using running text. This is different from search engines like 'Google', 'Yahoo!' or 'AltaVista', these engines require keyword search. The way Question Answering systems return answers is also different from the search engines mentioned. The search engines can give thousands of results. Question Answering systems only give the really relevant answers, often this is only one.

In this research data will be used from a Question Answering system. Therefore it is needed to know how Question Answering systems work, i.e. what the steps there are to get from the question to the answer. There are basically four steps to be taken:

- Lexical look-up
- Syntactical analysis
- Semantic analysis
- Relevant answer identification

We will discuss all four steps in the following sections. The last section will briefly describe the particular Question Answering system we have received datasets and evaluation material from; the Question Answering system of Q-go.

2.3.1. Lexical look-up

First of all, a lexical look-up takes place. This lexical look-up is facilitated by dictionaries. Most Question Answering systems reduce all words in the question to lemmas in the lexical look-up, e.g. 'am', 'are' and 'is' all get the lemma 'be', 'credit card' both gets reduced to the lemmas 'credit' and 'card' and the lemma 'credit card'. Moreover, PoS information is provided for the words. Because of ambiguity words can be assigned more than one PoS, e.g. 'book' is both a noun and a verb.

Dictionaries can be general dictionaries or dictionaries specific for a particular domain. We can use domain-specific dictionaries because we know what domain each question is from. The questions in the datasets provided by Q-go are posed on a client's website and the client represents a certain domain.

A general dictionary consists of general words which can be used in all kind of situations. The domain specific dictionaries consist of vocabulary which is typical for the domain, e.g. Dow Jones would be a lexical entry in the finance dictionary.

There are two reasons to make this distinction between the dictionaries. One reason is saving search space. Why search through information that we know is irrelevant. The other reason is avoiding noise. One would not want the ambiguous word *bank* with the meaning 'riverbank' in a financial setting.

When a word cannot be found in the dictionary it gets categorized as 'unknown'.

2.3.2. Syntactic analysis

After the lexical look-up the question consists of only lemmas together with its PoS tags. Next, syntactic analysis is done. The syntactical analysis uses grammar rules. The question is analyzed to determine its grammatical structure with respect to a given formal grammar.

Resolving ambiguity is one of the reasons to perform a syntactical analysis. For instance 'cash my check' needs a different analysis than 'check my cash'. The exact same words are used in the examples but the meaning is completely different.

One question can result in many parses. Often words have been assigned more than on PoS so different grammar rules will fire. Moreover, some words can be treated as a sequence of words or as an expression. For instance 'credit card' can get the tag that covers the whole expression, 'noun'. But 'credit card' can also be seen as a query consisting of two words which can both be assigned the tags 'noun' and 'verb'. So only the word credit card can already lead to five different analyses: 'noun', 'noun-noun', 'noun-verb', 'verb-noun', 'verb-verb'.

To determine which syntactical analysis is the best one, each analysis receives a score based on how probable the analysis is. How the scores are computed, is based on the internal grammar rules of a Question Answering system.

Syntactic analysis fails if the grammar rules cannot find a satisfying analysis. This can be due to missing or wrong grammar rules, but also to ungrammatical questions.

2.3.3. Semantic analysis

After the syntactical analysis, the semantic analysis takes place. Semantic analysis is the process of relating the syntactic analysis to the writing as a whole; a profile is created for the question.

A profile can consist of different types of information, depending on the type of Question Answering system. Typically information about the question type is provided. Is it a question asking for a location? Or is it asking in what manner something needs to be done? For web search applications interesting information is the question type, the action, the subject and the object.

For instance on a website of an airlines company the following questions can be asked:

1. How can my husband get a plane to Chicago?
2. Which way can I book a flight to Chicago?
3. How could we order plane tickets to Chicago?
4. How to order tickets to Chicago?

The question are all different, the required answer is the same. By reducing the sentences to profiles, all sentences can be reduces to one similar profile. The question type is a manner ('how?', 'which way?'), the action is 'order', the subject is 'person' and the object is 'ticket' and even more specific 'to Chicago'.

The following questions can also be asked:

1. Where can I book a flight to Chicago?
2. Where is the book about the flight to Chicago?

The questions look similar but the required answers are very different. Because of semantic interpretation, the sentences get a different profile. For both first and second question the question type is 'location' ('where?'). The action, subject and object differ. For the first question, the action is 'book', the subject is 'person' and the object is 'flight to Chicago'. For the second question, the action is 'to be', the subject is 'book about flight to Chicago' and no subject is available.

These examples show that the semantic analysis is a very important phase in the Question Answering system to understand what the intent of the user is.

2.3.4. Relevant answer identification

With the semantic analysis, a relevant answer to the question can be provided, if available. Relevant answers are found by looking at the semantic profile of the question. Assume a user has posed the following question:

'How to order tickets to Chicago?'

From the profile we know that it asks for a manner, so we have to provide an answer which shows in which manner something can be done. The action is 'order', so we look for an answer which explains how something can be ordered. The object is 'tickets to Chicago'. So

an answer must be found which reveals information about how tickets to Chicago can be ordered.

It might happen that there is no answer to the specific question, in this case for instance because nothing about 'Chicago' can be found. Then thesauri can be used to find out what relations the word 'Chicago' has.

Different types of semantic relations can be found in thesauri: hyponyms and hypernyms, antonyms ('poor-rich'), synonyms ('to buy- to purchase') and words that are related, but not synonyms ('to book- to purchase'). The concept is more or less the same, and the specificity of the concept is at the same level.

Assume that 'Chicago' has a hypernym relation with 'musical' and 'USA'. If there is information about how to book tickets to the USA, the answer to the question is found because the words in the profile of the question were changed into words which could be found with help of the thesauri.

Just as for the dictionaries there different types of thesauri possible for similar reasons: general and domain specific thesauri, to reduce search space and ambiguity.

The Question Answering system settings determine how many answers are provided.

2.3.5. Q-go

Web search applications are one particular type of Question Answering systems. Q-go is a Dutch company which is specialized in natural language search, providing web search applications on the websites of their clients. The strength of its application is that the user can pose the question in the way they want, either in a keyword format or in a complete sentence.

Q-go has databases for each client with possible answers to questions users pose. The answers are in fact also questions. So if a user asks: 'Are there any vegetarian meals aboard?', the answer can be 'Which flights provide vegetarian meals?'. By clicking on the answer, the user is redirected to the particular link that answers that question.

The relevant answer finding is done by comparing the question of the user to all questions available in the client specific database; the profiles of the input question are compared to the profiles of the questions in the database. Based on a match between the profile of the input question and the 'answer question', relevant answers are provided.

During the lexical look-up, Q-go comes across unknown words. The linguistic resources the Q-go language search technology uses are maintained manually. If an unknown word is found, a linguist must enter the word into the lexicon. Although Q-go does not know how big the problem of unknowns is, Q-go prefers to deal with unknown words automatically.

2.4. Conclusion

There are many approaches to handle unknown words. None of the approaches can deal with all unknown words. Therefore there are some considerations that need to be taken into account when choosing a method. One of the considerations is what type of data the unknown words must be found in, edited text, unedited text or even in a particular type of unedited text: user query language. The datasets we will use consist of the latter, user queries posed in the Question Answering system of Q-go. There are many variations of language real people use in unedited text which all need to be processed robustly. At this

moment Q-go is handling unknown words by manually adding them to the linguistic resources. There is no automatic process to identify or classify them.

3. Data analysis

This chapter will answer the first two research questions.

1. How big is the problem of unknown words within Q-go and does this lead to poor matching of user queries with responses?
2. What types of unknown words are there and which ones are the most frequent ones.

The Q-go language search technology categorizes the words which are not available in the linguistic resource, out-of-vocabulary (oov) words, into three categories:

- Spelling error (SE): Spell-correct the word to a known dictionary word, e.g the oov-word 'fligt' is spell-corrected to the known word 'flight'

- Compound (CMPND): Analyze the word as a compound, a word which is composed of two words and the meaning of the word can be compositionally derived of the two individual words, e.g. the oov-word 'bookshelf' is compounded into the known words 'book' and 'shelf': a shelf that you put books on.

- Unknown word (UW): No analysis possible, provide the word with the tag 'unknown', e.g. the oov-word 'dumbledore' cannot be analyzed.

The first two categories are meant to make the word known with regard to the dictionaries. If those two options are not able to make the word known to the system, the system is not able to analyze the word and provides the word with the tag 'unknown'.

In the tables presented in the chapters, we will refer to the three categories with the abbreviations presented in the enumeration above: SE, CMPND, UW.

The kind of unknown words we eventually want to reduce are the actual unknown words; words which neither contain spelling errors nor can be compounded into known words. Since 'unknown word' is one of the categories the system provides to the oov-words, it seems to be straightforward to look at only those words to get a good idea of how big the problem is of unknown words and which types there are.

However, the system is not working perfectly. We cannot assume that the system provides the right category for all words, the actual category. Therefore we need to annotate the data manually. Words which the system is categorizing as 'spelling error' or 'compound' can be actual unknown words. Words which get the system tag 'unknown' can be actual words containing spelling errors or actual compounds. With manual annotated data we can compare the system categories to the actual categories and derive how big the actual unknown word problem is.

Moreover, the system is not providing a type for the actual unknown words, e.g. the word 'rek.nr.' is an actual unknown word, but the system does not tell us that it is an abbreviation. However this kind of information is needed when we want to find which types of actual unknown words there are and how often each type occurs. So also for this manual annotation is needed.

On the annotated data a manual data analysis is performed which we will discuss in the following sections. First we will explain how the data in general looks and how we have

extracted representative samples from it which we have annotated. Next it is explained which information is annotated. Last the results are presented and discussed.

3.1. Data sampling

3.1.1. Population

The data used to perform the analysis on were obtained from search query logs from Q-go's clients. The query logs chosen, are all from Dutch websites of clients. Previous research in the field of information retrieval claims that language use changes per domain. Therefore, query logs from three clients divided over Q-go's most important domains are used. The domains and respective clients are:

Aviation: KLM (a Dutch airlines company)
Finance: ABN AMRO (a Dutch bank)
Insurance: OHRA (a Dutch insurance company)

On the extracted query logs the system has performed data tokenization. Data tokenization is needed to be able to process the queries. How the exact tokenization process looks like is defined per client. In general it means that special characters (e.g. an ampersand or hyphen) are conversed or simply deleted. In most cases it is useful to perform data tokenization but in some cases it is not. For instance the email-address *info@companyX.com* is after data tokenization split into *info@companyX* and *com*. The special character 'period' is deleted and leaves a space. Something similar can happen to codes. Sometimes codes contain a 'hyphen' (e.g. *CODE-123*). This hyphen is for most clients deleted so that the code gets split into two or more words (e.g *CODE* and *123*). In these two examples data tokenization actually should not take place since the whole email address or the whole code should be considered as one entity.

We focus on the queries which contain one or more words which are not present in the dictionary, out-of-vocabulary (oov) words. This is done to avoid looking at too many user queries in which nothing is wrong.

One of the consequences of this decision is that real-word-errors will not be taken into account. There are three types of real-word errors:

- Having the word in the dictionary but not the right meaning e.g. having the word 'windows' in the dictionary as the opening in a wall for admission of light and air but not as an operational system.

- Having the word in the dictionary but not its right word category (PoS), e.g. having 'book' in the dictionary only with the category noun and not with the category verb.

- Having individual words of a multiword expression in the dictionary but not the expression itself, e.g. having 'dinner' and 'table' in the dictionary but not the expression 'dinner table'.

In all three cases the words are in the dictionary and therefore will not show up in the queries we focus on containing oov-words.

To get an idea of how often the system encounters oov-words in the three domains we examined 100,000 user queries randomly collected from the Q-go database for each client. Table 1 presents basic statistics about these extracted queries. Both type and token

information is given. Tokens are the actually occurring instances of some phenomenon in a corpus of data. Types are the unique phenomena in the corpus of data. So 'a rose is a rose' contains five tokens, i.e. all words, but only three types, i.e. all unique words.

The KLM data contains the longest user queries in terms of number of words. In addition, KLM has more unique user queries than the other two clients. This can have something to do with the length of the user queries. The more words used, the higher the probability that words used in a user query are not the same. The ABN and OHRA data show a similar distribution across user queries types and the number of words a user query contains. This shows that the user queries vary per domain in length.

When looking at the user queries containing an oov-word, most queries of them contain a word which is spell-corrected followed by words that are compounded and words that are provided with the tag 'unknown'. This is similar across the three domains.

Table 1: Overview of how often the three types of out-of-vocabulary words occur among 100,000 user queries per client.

Information	KLM		ABN		OHRA	
	Types	Tokens	Types	Tokens	Types	Tokens
# of queries	53,206	100,000	38,428	100,000	32,476	100,000
# of words	232,579	322,953	111,862	211,856	89,542	199,819
% of UW of all words	1,1%	0,9%	1,2%	0,7%	0,8%	0,4%
% of SE of all words	6,1%	5,1%	8,0%	5,1%	6,7%	4,0%
% of CMPND of all words	1,2%	1,1%	1,5%	1,0%	1,9%	1,3%
Queries containing UW	4,0%	2,5%	3,3%	1,4%	2,0%	0,8%
Queries containing SE	21,3%	13,0%	20,6%	9,5%	17,4%	7,5%
Queries containing CMPND	4,9%	3,2%	4,3%	2,1%	5,3%	2,5%
Queries containing at least one oov-word	27,5%	17,0%	27,1%	12,6%	24,0%	10,7%

3.1.2. Sampling Method and Datasets

Now that we know what the data looks like per client in general, we concentrate on the user queries which contain oov-words, since those queries are the ones we will annotate. From the 100,000 user queries per client presented in the previous section, we took per client two samples containing only user queries which contain an oov-word. We took samples because manual annotation is performed. Annotating all user queries containing an oov-word would be too time-consuming.

We make a clear distinction between the two samples we extract per client. We will call them the samples:

- 'Overview' samples, n=1,000
- 'Unknowns' samples, n=500

The 'overview' sample is taken per client to get a good overview of how good the system performs in case of oov-words i.e. does the system categorize the words correctly or is it making categorization errors. This sample consists of 1,000 user queries which contain at

least one oov-word. The distribution across the three categories must be similar to the distribution across the categories in the samples of n=100,000. This is achieved by taking a random sample from the user queries which contain an oov-word, and resample it until the set of queries shows a similar distribution across the categories compared to the set of 100,000 queries described above. Table 2 shows per client the distribution across the three different categories in the 'overview' samples we have collected to annotate.

Table 2: Distribution across the three different oov- types in the samples with n=1000 per client. Type and token information.

Oov-type	KLM		ABN		OHRA	
	Types	Tokens	Types	Tokens	Types	Tokens
UW	13,3%	13,1%	12,0%	10,6%	8,7%	8,8%
SE	73,3%	72,9%	86,1%	75,9%	71,6%	70,0%
CMPND	13,4%	14,0%	15,1%	13,5%	21,9%	21,2%

We expect that the technology works well, i.e. most actual unknown words get the tag 'unknown' and only a few of the actual unknowns will get corrected for spelling or get compounded. Since we are interested in the actual unknown words we took a second sample per client, the 'unknowns' samples. Each 'unknowns' sample consists of 500 user queries extracted from the 100,000 user queries presented in the previous section containing at least one word which the system has assigned the tag 'unknown'. By analyzing the actual unknown words in these samples, a good idea can be obtained of which types of unknowns there are and how often each type occurs.

In one user query, more oov-words which have been assigned the system tag 'unknown' can be present. Eventually, we will analyze the unknown words and not the 500 user queries. Therefore we present in Table 3 per client the number of words which are assigned the 'unknown' tag. Type and token information is given about queries, not words, e.g. if *cockpit* is an oov-word which gets the system tag 'unknown', and two user queries are 'Can we see the *cockpit* during our flight' and 'See *cockpit* during flight', then there is one type when looking at the exact oov-word, but there are two types of unknown words when looking at the queries. We look at the queries so in this case we have found two unknown words.

When looking at the token information, we see that compared with ABN and OHRA, KLM user queries are more likely to contain more than one word which is assigned the system 'unknown' tag. The number of user queries extracted per client is 500. The user queries of KLM contain 633 words with the system tag 'unknown'. The user queries for the other two clients contain 554 and 524 words with this tag. These last two numbers lie much closer to 500 so not many queries contain more than one word.

When comparing the type and token information per client, we find that the user queries of OHRA in the sample consist of more equal user queries than the user queries of KLM and ABN. What we can conclude from this is that users ask questions for OHRA in a more similar way in comparison to KLM and ABN.

Table 3: Number of words assigned the tag 'unknown' by the system in the 'unknown samples'

Client	Types	Tokens
KLM	600	633
ABN	514	554
OHRA	438	524

3.2. Annotation

All oov-words in the two samples per client are manually annotated. The information which has being annotated about the oov-words changes per type of sample.

3.2.1. 'Overview' samples (n=1,000)

Per user query all oov-words are provided together with the category the system gives them: 'spelling error' (SE), 'compound' (CMPND) or 'unknown' (UW). To get a good overview of how good the system performs in categorizing the oov-words compared to the actual category of oov-words, information is annotated about what the actual category of the oov-word is. Below in Table 4, for each system category (columns), the possible actual categories (rows) are listed and defined. There are more actual categories than the three categories the system can assign to the oov-words.

As can be seen in Table 4 , real-word errors only occur in the compound category and not in the other two categories. This is possible because real-word errors only can arise when words are available in dictionaries. Neither words with spelling errors nor words provided with the tag 'unknown' are available in the dictionaries. Compounds, on the other hand, do consist of two words available in the dictionaries, otherwise the word could not have been compounded.

Table 4: For each system category the possible actual categories are listed and defined

Actual Category	System category		
	UW	SE	CMPND
SE	The word contains spelling errors (this does not indicate that the word is known if it is spelled correctly).	• Yes: The word is a spelling error and the right correction is provided as well. Unknown words which originate from forgetting spaces (e.g. flightfrom in "flightfrom Amsterdam to London") are also considered as spelling errors except if the space is forgotten in product names. • No: The word is a spelling error but the right correction is not provided.	The word contains a spelling error.
CMPND	The word is a compound.	The word is a compound	• Yes: The word is a compound, the meanings of the words of which the word consists of contribute to the meaning of the whole word. • No (1): The word is a compound and the right meaning is derived from the two words but the words are combined in the wrong order e.g. 'buitenlandverzekerden' (abroad insured) is compounded in 'buitenland' ('abroad') with as specifier 'verzekerden' ('insured'). It should be the other way around. The head is 'verzekerden' ('insured') and the specifier is 'buitenland' ('abroad').
UW	The word is an unknown word. Extra information is annotated, namely the type of unknown word (e.g. Abbreviation, Product Name, Personal Name etc.) and the Part of Speech (PoS) tag.	The word is an unknown word (without spelling errors). The type of the unknown word and its PoS tag are annotated.	The word is an unknown word. It does not consist of two words from which the meaning can be compositionally derived. The type of the unknown word and its PoS tag are annotated.
MWE	The word is part of a multiword expression (MWE).	The word is part of a MWE.	The word is part of a MWE.
RWE	-	-	The word is a real word error. The word consists of two words, but the technology does not derive the right meaning from the two words (e.g. 'huisinrichting' ('house furnishing'), gets separated into 'woning' ('house') and 'instelling' ('organization'))
NotCons	The word is not considered because the word/user query does not make any sense	The word is not considered because the word/user query does not make any sense.	The word is not considered because the word/user query does not make any sense.

3.2.2. 'Unknowns' samples (n=500)

Per user query the oov-words which have been assigned the tag 'unknown' are provided. For each actual unknown word, we annotated the type of the unknown word as defined in Table 5, and it PoS.

Table 5: The different types defined for the actual unknown words during type classifcation

Type of unknown word	Definition
Abbreviation	Words which are abbreviated. Acronyms are not abbreviations unless it is more common to use the words the letters in the acronym stand for instead of the acronym itself. *Example: bet.rek is an abbreviation of **bet**aal**rek**ening (bankaccount)*
Code	'Words' which consist of digits and/or characters. If the word consists of only characters, the characters must be mixed in a way that the user did not mean a 'normal' word. *Example: ZSXBE10*
Company	A company name. *Example: Shell*
Foreign word	A word in a language other than Dutch, i.e. a foreign word in a Dutch sentence, or if there is no context, words which are commonly accepted as Dutch words. *Example:'Connection' in 'connection vlucht' ('connection flight')*
Foreign sentence	Words in a non-Dutch sentence. *Example: 'Flight' in 'Flight to London'*
Geographical location	Words which indicate a geographical location. *Example: Singapore*
Interjection	A word expressing emotion and having no grammatical relation with other words in the query. *Example: alsjeblieft (please)*
Internet	Internet related words as email addresses and URLs. *Example: www.nu.nl*
Medical term	Terms which indicate something medical. *Example: stoma (colostomy)*
Named Entity	Words which are Named Entities but not fall into one of the other classes. *Example: Facebook*
Personal Name	A name that indicates a person. *Example: Balkenende*
Product Name	A name that indicates a product. *Example: Awardmiles*
Unknown Word	A word which should have been in the general resources, i.e. these words show the gaps in the linguistic resources. *Example: ontdooien (to melt)*
Currency	Currency. *Example: yen*
Airport	Name of an airport. *Example: Heathrow*
Aviation	Words which have something to do with aviation. *Example: cockpit*

Two types which are closely related are 'Foreign word' and 'Foreign sentence'. The borders between the two classes are not always clear. When is something a foreign sentence? If the user query consists of more than one foreign word? But what if the user query consists of two foreign words which can be easily used in the Dutch language? We tried to be as consistent as possible, though some type classifications were clearly subjective choices.

If a query did get a successful response, this information can also tell us what categories of oov-words are most problematic for the application of Question Answering. For this reason, details about whether or not a query with an oov-word resulted in a good or bad response match are included in the survey. Matching is in this case the process of finding a correct answer to the question. The system bases its decision of what answers will be returned on how well the user query and the predefined answers match, i.e. how similar the user query and the answer are. If the system does this correctly, it results in a good match. This means that the right answer is in the top three answers. It is also possible that the system thinks that the input user query matches better with irrelevant answers. This gives a bad match, the right answer is not in the top three of answers. Further, it is also possible that the system cannot find an answer based on the user query, this gives a no match. These no matches are also included.

3.3. Findings

This section will describe and discuss the findings. On the one hand, we will describe the findings on the performance of the system. On the other hand, we will describe the findings on the actual unknown words. For both sections we present both the findings when looking at all clients together and the findings we have found per domain. We do this to see whether there is a general trend and whether there are domain-specific issues.

3.3.1. System vs. actual category

This section addresses the first part of the first research question: 'How big is the problem of unknown words?' We compare the category the system provides to the actual (manual annotated) categories. From this we can derive how well the system performs, and also what the percentage of actual unknown words is across all oov-words.

All clients

The categories assigned by the system to an oov-word and the actual (manual annotated) categories can be found in Table 6. Because there is not a big difference in percentages between types and tokens, we choose to only show the percentages of the types from now on.

Table 6: The provided system category vs. the actual category.

Actual / System	SE	CMPND	UW	MWE	RWE	NotCons	TOTAL
SE	46,2%	1,3%	11,7%	5,0%	0,0%	9,0%	73,2%
CMPND	5,5%	6,7%	0,9%	0,4%	1,0%	1,3%	15,7%
UW	2,3%	0,5%	3,9%	1,3%	0,0%	3,0%	11,0%
TOTAL	54,0%	8,5%	16,5%	6,7%	1,0%	13,3%	100,0%

From all oov-words, 16.5% are actual unknown words. Combining this information with Table 1, this means that on average over the three clients 1.6% of all the words in unique user queries are unknown words.

Three actual categories cannot be provided by the system: 'MWE', 'RWE' and 'NotCons'. From Table 6 we can conclude that 21% of the oov-words fall into these actual categories and therefore cannot be categorized correctly by the system. This means that the system could be improved with 21 % if these categories can be recognized correctly. Since the previously mentioned categories cannot be provided, we compare only the three categories which can be provided by the system to the corresponding actual categories: 'SE', 'CMPND' and 'UW'.

In Table 7 we show for each actual category how the system categorizes the words in the actual category across the three system categories. The cells that are marked yellow are the percentages which are categorized correctly by the system, i.e. the actual category corresponds with the category the system has provided. We find that among the words which are actual unknown words, only 24% gets the right tag from the system (the tag 'unknown'). In terms of categorizing actual unknown words correctly there is room for improvement of 76%. For the other two categories the system performs quite well i.e. the

majority of actual words containing spelling errors and actual compounds are also categorized that way by the system.

Table 7: The actual category vs. the system category.

System \ Actual	SE	CMPND	UW
SE	86%	15%	68%
CMPND	10%	79%	7%
UW	4%	6%	25%
TOTAL	100%	100%	100%

This implicates that if we only consider the actual unknown words found in the 'unknowns' samples, which was the plan, 75% of actual unknown words will not be taken into account. Given these results, it makes more sense to consider the actual unknowns from the 'overview' samples.

Per client

Overall each client shows the same pattern as the overview of all clients does, this can be found in Table 8. Nevertheless there are very clear differences between the domains.

Table 8: The provided system category vs. the actual category per client

KLM

System \ Actual	SE	CMPND	UW	MWE	RWE	NotCons	TOTAL
SE	38,2%	1,2%	18,8%	5,5%	0,0%	9,6%	73,3%
CMPND	2,6%	5,9%	1,0%	0,4%	0,2%	3,4%	13,4%
UW	1,6%	0,7%	5,4%	0,8%	0,0%	4,8%	13,3%
TOTAL	42,4%	7,8%	25,1%	6,7%	0,2%	17,7%	100,0%

ABN

System \ Actual	SE	CMPND	UW	MWE	RWE	NotCons	TOTAL
SE	49,5%	1,1%	6,9%	5,6%	0,0%	13,1%	76,1%
CMPND	6,9%	4,6%	0,7%	0,3%	0,9%	0,0%	13,3%
UW	2,6%	0,4%	3,0%	1,4%	0,0%	3,2%	10,6%
TOTAL	58,9%	6,1%	10,6%	7,2%	0,9%	16,3%	100,0%

OHRA

Actual / System	SE	CMPND	UW	MWE	RWE	NotCons	TOTAL
SE	53, 0%	1,7%	7,5%	3,8%	0,0%	3,7%	70,1%
CMPND	7,6%	10,0%	1,2%	0,6%	2,1%	0,0%	21,4%
UW	2,8%	0,5%	2,9%	1,8%	0,0%	0,5%	8,5%
TOTAL	63,8%	12,1%	11,6%	6,2%	2,1%	4,2%	100,0%

In OHRA, the number that attracts attention is the percentage of actual compounds. This percentage is much higher than the percentage of compounds in the data of the other two clients. It is possible that for smaller clients, like OHRA, fewer linguistic resources might be behind this number of oov-words, but this is not the case for Q-go's system. In fact, the linguistic resources being used for OHRA have more words than those used for the other two clients. A more plausible explanation in this case is that users ask a lot about medical therapies and treatments. Such words are often compounds. So the high percentage of compounds is related to domain differences.

In KLM, the percentage of actual unknown words is much higher than for the other two clients. This is partly due to the fact that in comparison to the other two clients, many user queries are in a foreign, non-Dutch, language, mostly English. Foreign words are actual unknown words if they are not in the dictionaries. One foreign user query can therefore generate many actual unknowns. It makes sense that especially in the KLM data user queries are written in a foreign language. KLM is a company which operates worldwide. It brings people from one place in the world to another place in the world. Although KLM has websites in different languages, it is possible that a non-Dutch user is transferred to the Dutch website and asks its question in a language other than Dutch. So, again, the high percentage of a category, in this case actual unknowns, is related to domain differences.

The oov-words of ABN and KLM are much more often 'Not Considered' then the OHRA oov-words. In case of KLM, this high number is caused by an interface bug. Instead of user queries, cache information is provided. In case of ABN, the large number of 'Not Considered' oov-words are caused by hacking attacks and user queries from one IP-address, from which the system frequently receives irrelevant question with many spelling errors.

If we eventually want to reduce the amount of actual unknown words in the user search queries, we have to keep in mind that there are differences between the domains.

3.3.2. Actual unknown words

This section addresses the second research question: 'Which types of unknown words are there and how often does each type occur?' and the second part of the first research question: 'Do the unknown words lead to poor matching of user queries with responses?'. By describing the type classifications we have performed on actual unknown words and the details we have gathered about which response the queries containing actual unknowns got, we will be able to answer these questions.

All clients

To get a good picture of which types of actual unknowns are the most frequent ones and therefore interesting to investigate, we present the type classification of actual unknowns found in the 'overview' samples in Table 9. For each type of actual unknown word, the distribution across the three classes the system can give an unknown word is given.

Table 9: The type classification of unknown words found in the 'overview' samples, categorized per system tag.

System tag / Types	SE	CMPND	UW	TOT
Abbreviation	10,7%	0,6%	0,4%	11,7%
Code	1,6%	0,0%	5,1%	6,7%
Company	4,7%	0,6%	1,9%	7,1%
Foreign word	11,3%	0,9%	5,8%	18,0%
Foreign sentence	13,5%	0,8%	1,8%	16,1%
Geographical Location	4,1%	0,0%	1,8%	5,9%
Interjection	0,1%	0,1%	0,1%	0,3%
Internet	0,1%	0,0%	0,0%	0,1%
Medical term	1,1%	0,3%	0,3%	1,7%
NE	0,1%	0,0%	0,2%	0,3%
Personal Name	1,3%	0,0%	1,0%	2,3%
Product Name	6,8%	1,5%	2,8%	11,1%
Product name| Personal Name	0,0%	0,0%	0,3%	0,3%
Unknown words	11,2%	2,3%	3,6%	17,0%
Currency	1,4%	0,0%	0,0%	1,4%
TOTAL	68,0%	6,9%	25,1%	100,0%

The described classes in Table 5 (Section 3.2.2), are for the most part unique for each word. For only one word in the data (in the 'overview' sample) there were two classes possible. The word was 'Aalberts'. This word falls both into the category of 'Personal Name' and 'Company'. The word definitely was a Dutch last name but sometimes entrepreneurs use their last name as a company name. Because no context was provided in this particular case it was not clear which class to choose so both types have been added to the word.

One thing to keep in mind when interpreting number on 'Foreign sentence' words is that each 'unknown word' in that particular user query is counted. So if a user query consists of twelve words, all those words can be actual unknown words. Hence the numbers for this particular category can increase fast.

Besides which types of unknown words are the most frequent ones, we also show what class the type is categorized into by the system It is already known that most actual unknown words end up in the 'spelling error' category. This can also be found in Table 9. For each type the trend is that more or less 60% of all words of a type are categorized by the system as 'spelling error'. Three types show a slightly different pattern.

Of all 'Abbreviation' words, 92% and of all 'Foreign sentence' words, 83.9% get the system category 'spelling error'. This can be explained by the fact that abbreviations often look like words, the same can apply for short foreign words e.g. 'a', 'the' and 'for'.

The other type which does not follow the trend is 'Code'. Most codes get the right system category, namely 'unknown'. This is a logical consequence from the definition of the type. From a sequence of mixed digits and characters no analyzable word can be made.

Some of the types in Table 9 can be combined into one category: Named enitities (NEs). Those types are: 'Company', 'Geographical Location', 'Named Entity', 'Personal Name', 'Product Name' and 'Currency'. If those percentages are added the NE category ends up with 28.4% of the unknown words.

With this information a list can be made of which types are the most frequent ones. This can be found in Table 10.

Table 10: Overview of most occurring types among the unknown words.

Type	Percentage
Named Entities	28.4%
Foreign words	18%
Unknown Word	17%
Foreign sentence	16.1%
Abbreviation	11.7%
Code	6.7%

The three most frequent types in the 'Named entities' are 'Product name', 'Company' and 'Geographical location'. The three types are respectively responsible for 39%, 25%, and 21% of all named entities.

Next to the actual unknown words in the 'overview' samples, also a type classification is performed on the actual unknowns in the 'unknowns' samples. The distribution of actual unknown across both samples is similar. Table 11 shows the type classification of actual unknown words in both samples.

However, In Table 11 we can find that the distribution across the types is not similar for all types. Less data on actual unknown words is being analyzed in the 'overview' samples. However, this seems not to influence the distribution. Most differences can be explained. They are probably all caused by the way of annotating. For instance the 'Foreign' types show differences, but if they are merged the percentages are more are less the same (respective 27.7% and 30.8%). Some of the words may have interchanged category because sometimes it is not clear to which category a word belongs.

For 'Medical terms', 'Aviation', 'Currency' and 'Airports' the explanation of differences between the distributions lies in the fact that those categories are more specific categories which have been used more concisely in the 'unknowns' samples. During annotation we expected to perform the type classification only on the actual unknown words in the 'unknowns' sample. Therefore more distinctive categories have been created. Some of these types are also very sparse in the data so it is also possible that such a type never was seen in the 'overview' samples.

Table 11: Comparison between the types of actual unknowns in the 'unknowns' samples to the actual unknowns in the overview samples.

Type	'Unknowns' samples	'Overview' samples
Abbreviation	2,2%	1,5%
Code	20,7%	20,3%
Company	9,7%	7,5%
Foreign word	15,1%	23,2%
foreign sentence	11,6%	7,3%
Geographical Location	7,4%	7,2%
Interjection	0,6%	0,4%
Internet	0,0%	0,0%
Medical term	6,2%	1,1%
NE	1,6%	0,8%
Personal Name	3,5%	4,1%
Product Name	8,0%	11,2%
Product name | Personal name	0%	1,1%
Unknown word	11,4%	14,3%
Currency	0,2%	0,0%
Airport	0,8%	0,0%
Aviation	1,0%	0,0%
TOTAL	100,0%	100,0%

Per client

In Table 12, the type classification per client of actual unknown in the 'overview' samples can be found. Next to this, the percentage is given of how often the type occurs across all three clients. For some types the distribution across clients varies a lot. Most of the differences are explainable by the properties of the domain. This again shows us that there are very clear differences between the domains. The most frequent types of unknown words found for a client, seem to depend on the core business of the client. If the type is has something to do with the core business of the client, many unknown words of that type occur in the data.

For instance for the type 'Company', OHRA shows the highest percentage. This is because users ask questions about several companies that provide medical services. Since it is an insurance company people can also ask about the discount they get on their insurance because they are an employee of a company that has a special agreement with OHRA. In ABN, people roughly only want to know things about companies concerning the stock rates for the particular company. In case of KLM, users do not want such specific information about companies. Some users do want to know things about are other airline companies. However, most of them are multiword expressions (e.g. 'Malaysian airlines', 'Delta airlines') and therefore not taken into account.

Table 12: Type classification per client of actual unknown words found in the 'overview' samples. Behind the client name the number of annotated tokens can be found in parentheses..

Type	KLM (334)	ABN (119)	OHRA (119)	TOTAL
Abbreviation	7,3%	14,3%	13,4%	11,7%
Code	5,0%	9,2%	5,9%	6,7%
Company	2,0%	6,7%	12,6%	7,1%
Foreign Word	21,3%	16,0%	16,8%	18,0%
Foreign Sentence	33,2%	12,6%	2,5%	16,1%
Geographical Location	6,7%	10,1%	0,8%	5,9%
Interjection	0,9%	0,0%	0,0%	0,3%
Internet	0,3%	0,0%	0,0%	0,1%
Medical term	0,0%	0,0%	5,0%	1,7%
NE	0,9%	0,0%	0,0%	0,3%
Personal Name	1,2%	5,9%	0,0%	2,3%
Product Name	6,4%	9,2%	17,6%	11,1%
Product name | Personal Name	0,0%	0,8%	0,0%	0,3%
Unknown word	14,9%	10,9%	25,2%	17,0%
Currency	0,0%	4,2%	0,0%	1,4%
Airport	0,0%	0,0%	0,0%	0,0%
Aviation	0,0%	0,0%	0,0%	0,0%
TOTAL	100,0%	100,0%	100,0%	100,0%

The 'Geographical location' type does not completely follow the pattern that most unknowns fall into the core business type of the particular client. For ABN, more geographical locations are unknown than for KLM, while transporting people from one location to the other is the core business of KLM, i.e. locations often are part of user queries for KLM. This is probably true, but users ask ABN information on the location of their branches and they often add street information to the user queries. Street names are a highly productive, open class. Almost all Dutch words can be street names nowadays. Only important street names are included in the linguistic resources. Locations in KLM, on the contrary, are a rather closed class. Most of the locations are already in the linguistic resources. In OHRA there are not a lot of locations among the unknown words. This is probably caused by the fact that users do not need information on locations and if they do, the locations are already in the linguistic resources.

In particular for KLM, a big part of the unknowns have received the type classification 'Foreign Sentence'. KLM receives more often non-Dutch user queries than ABN and OHRA. Because each word in a foreign user query probably is unknown, the numbers increase very fast. This 33.2% is found in only 3% of the user queries annotated. The reason that especially KLM gets non-Dutch sentences can be explained by the fact that KLM supports websites in six languages in many different countries. ABN supports only English and Dutch in their website. End users just make mistakes and forget to change the language in the website they are accessing. As result they type French questions in the English website or English questions in the Dutch website.

Table 13 shows the type classification per client of actual unknown words found in the 'unknowns' samples. Also here differences in distribution can be found across clients. These differences can again be explained by the differences in domains, similarly to the differences found in the 'overview' samples.

Table 13 Type classification of unknown words in the 'unknowns' samples for all clients. Behind the client name the number of annotated tokens can be found in parentheses.

Type	KLM (600)	ABN (514)	OHRA (438)	TOTAL
Abbreviation	1,2%	1,5%	3,9%	2,2%
Code	22,4%	26,6%	13,2%	20,7%
Company	5,4%	11,6%	12,2%	9,7%
Foreign word	14,9%	18,1%	12,2%	15,1%
Foreign sentence	21,2%	2,5%	11,2%	11,6%
Geographical Location	8,7%	12,6%	1,0%	7,4%
Interjection	0,8%	0,5%	0,5%	0,6%
Internet	0,0%	0,0%	0,0%	0,0%
Medical term	0,0%	0,0%	18,5%	6,2%
NE	0,4%	3,5%	1,0%	1,6%
Personal Name	2,9%	4,5%	2,9%	3,5%
Product Name	3,7%	10,1%	10,2%	8,0%
Personal Name | Product name	0,0%	0,0%	0,0%	0,0%
Unknown word	12,9%	8,0%	13,2%	11,4%
Currency	0,0%	0,5%	0,0%	0,2%
Airport	2,5%	0,0%	0,0%	0,8%
Aviation	2,9%	0,0%	0,0%	1,0%
TOTAL	100,0%	100,0%	100,0%	100,0%

Next to this type information there is also information about the PoS of the actual unknowns. Across all samples the most unknown words were nouns[3]. There were no specialties found in the distribution of PoS across the types of unknown words or across clients therefore this information will not be shown.

We have gathered information about which response the queries got. A good match, bad match or no match. Although we expected that this would tell us what types are most responsible for bad responses, this is not what happened. It is very difficult to find out whether the specific unknown word was due to the good, bad or even no matching. Which response will be provided by the system depends on the whole user query, not on one single word. For this reason we were not able to answer the question whether unknown words led to bad responses.

3.4. Conclusion

The first research question is answered in the previous sections: 'How big is the problem of unknown words and do they lead to bad matching?'. In oov-words, 16.5% are actual unknown words. Only 25% of the actual unknown words get the corresponding system tag 'unknown'. If one wants to get an idea of how big the problem is one has to look at the all oov-words, otherwise 75% of the actual unknown words will not be found. Whether the actual unknown words lead to bad matching cannot be said. Matching depends often on many words in the user query. Not just one word is responsible for it.

[3] This is an interesting finding for probabilistic models since they could assign as a default category 'noun' to unknown words. Just this piece of information would be enough to guide the parser to a full syntactic analysis that Q-go now lacks.

The second research question is also answered: 'What types of unknown words are there and how often does each type occur?'. We have found in the type classification that there are several types, e.g. 'Abbreviation', 'Code' and 'Personal name'. Named entities are the most frequent type, 28.4% of all actual unknowns are named entities. The class type 'Named Entities' consists of the several named entity types defined in the type classification, i.e. types like 'Personal Name', 'Company' and 'Location' are added to get the general class type 'Named Entity.

During finding the answers to both questions, we have found that there are very clear differences between the domains. This means that to get best results, the eventual method to reduce the amount of unknown words must be applied to each domain separately.

3.4.1. Issues when choosing an approach

At this stage we can address more of the issues discussed in Section 2.1.3. First of all we can decide on which type to tackle. We decide to focus on named entities. It is the most frequent type among the actual unknown words. Moreover there is already much research on named entities that can be used. The 'Foreign word' and 'Foreign sentence' types are more a language guessing issue than an understanding unknown words issue, so it is not a good choice to focus on. The 'Unknown word' type, defined in the type classification is not a clearly defined type. Several kinds of words have ended up in this category. Deciding on a method would be very difficult. We also avoid the type 'Abbreviation'. Abbreviations make up a small amount of all the unknown words. Moreover, people are not consistent in how they make abbreviations; they abbreviate words as they want to. Identifying rules would be very difficult. It is not clear what methods would work. The focus will also not be on the type 'Code', codes often follow the same pattern per client, finding rules would be easy and it would therefore be not necessary to use an intelligent method.

We consider a named entity as resolved when a semantic type is provided. To be able to give a relevant answer to a question it is not enough to know that an unknown word is a named entity. We want to know what type of named entity the unknown word is. The semantic type will be the general class an entity belongs to, e.g. the type of the named entity 'Shell' will be 'Company'. The method we choose, however, must be a method to find all types of named entities with only one approach which can be tuned to a particular named entity type. We do not want to have a different approach for each named entity type.

Based on the results described earlier in Table 12, we will focus on the most frequent three types of named entities.

- Geographical locations (21% of named entities): a geographical location is a location which indicates the name of a place which is for everyone the same. 'Home' is a location, but means something different to most people. An airport code is a location, and has the same meaning for everyone.

- Product Names (39% of named entities): products are commodities offered for sale. Services are not included.

- Companies (25% of named entities): companies are institutions created to conduct business.

The types are not very fine-grained but since these types are the ones which are used during annotation, we kept them types. As can be seen, some types that have been

annotated during the data analysis are missing in this list. Those types are: ‘Personal name’, ‘Currency’, ‘Named entity’ and ‘Airport’. They are responsible for the rest of the unknown named entities (15%).

We choose to do offline recognition instead of online named entity recognition. We want to expand the linguistic resources with the approach. Offline recognition has the advantage that the user is not the immediate victim of a failure of the system.

4. Approaches to reduce the amount of named entities

Following the data analysis we can determine the requirements our approach to reduce the amount of named entities has to fulfill:

1. As many named entities as possible must be tackled with the same approach
2. Named entities must be classified semantically
3. User query language must be processed robustly
4. Off-line recognition

In this section we will take a look at what research has been done on named entity recognition and classification. As we can recall from Section 2.1 there are roughly two kinds of approaches to tackle unknown words and named entities in particular: content-based and context-based approaches. We will discuss in the next section which approach meets the requirements best.

4.1. Content-based vs. context-based

The goal of many content-based approaches is to provide a PoS rather than a meaning. For this research, it is more important to add information about the semantic type of named entities. We do think that providing the PoS "Named entity" can help classifying named entities. Recognizing that a word is a named entity is useful because classifying "normal" words as being named entities is less probable. But Q-go does not have a PoS tagger itself to help in this classification process.

If a PoS tagger would be used which is freely available for research it is still open to discussion whether this will provide correct PoS tags to the words in the user queries. As mentioned earlier user queries are not grammatically well-formed, users do not make similar queries when looking at the order of words. Can a PoS tagger cope with this? Because of time constraints we did not investigate this. As far as we can see now, a PoS tagger on itself will not meet the requirements of the approach except for the first one.

A simple option of a content-based approach would be to look at internal word cues as capitalization. A property of many named entities is that they are recognizable by the capitalization cues. In user queries however, capitalization is not very common. Users tend to not stick to orthographic rules concerning capitalization so only a few named entities will be found. Using capitalization as a cue also will not provide the class a category; it would be only extra evidence for being a named entity. Hence, this approach is not interesting for us because requirement one, two and three are not fulfilled.

A different content-based approach which could be useful for named entities and is mentioned in section 2.1, is to look at parts of words which can indicate a named entity and its class like 'soft' in 'Microsoft'. We are not convinced that this will occur very often to give enough evidence that a specific part indicates that the word is a named entity of the particular class. This is partly due to data sparseness. Moreover, it will only find those named entities which have the specific "recognizable" part. Most named entities will not contain such an "easy recognizable" pattern. If you for instance think of countries, it is difficult to find a pattern. We admit that it will help for some classes, for instance Dutch street names, can often be recognized by the word in which they are ending: -straat (street), -laan (lawn), -plein (square), these are only a few examples. But the approach we

are looking for must find as many named entities as possible in as many classes as possible. This approach is not expected to achieve that.

A last content-based approach mentioned in section 2.1 is the one which looks at the sequence of characters in a word (Goldberg, Gabay and Elhadad, 2008). This approach is only interested in assigning a PoS tag, although it can be used to classify named entities in their own target class as well. It must be possible to learn patterns of named entities per target class instead of learning a general "named entity pattern". The approach could be used as an extension to the approach mentioned above; the one who is finding common parts of named entities. That is, it indeed can find exact character sequences which indicate a target class, but also a more general idea of how the general character sequence may look like. Although it might be a powerful solution, we again have to keep data sparseness in mind; there might be too less data available to get a good model. In addition it will only find candidates which follow some pattern. We do not know how many named entities do this, so again only requirement two is met.

Context-based approaches are more likely to meet the requirements. As we can recall from section 2.1, especially the approaches based on lexico-syntactic patterns do. With one approach many classes can be covered, a semantic interpretation is provided, user query language is processed and gazetteer lists can be created, i.e. offline recognition is done.

We will discuss two context-based approaches which meet the requirements here in more detail since we base our approaches on these two. One is described in Paşca (2007) the other in Pennacchiotti and Pantel (2006).

The two approaches have a similar basis. Based on seeds representing a specific class, context patterns with which the seeds co-occur are extracted. The premise made is that the context in which the seed occurs might be an indication that if that context is present again around a word, this word also could be a candidate of the target class. By using the context patterns co-occurring with seeds new candidates can be found. The new candidates found are presented as a ranked list per target class. These lists can be transformed to gazetteers.

4.1.1. Paşca (2007)

The goal of Paşca (2007) is to extract named entities from anonymized web search queries. The extraction is done without handcrafted extraction patterns or domain-specific knowledge. By means of a small set of seed entities of a particular class pertaining named entities to that particular class of interest are acquired. Paşca (2007) shows that noisy user search queries contain valuable information for web-based entity extraction. Paşca (2007) claims that this work is the first endeavor in named entity discovery from web search queries.

Given a set of target classes, represented by seeds, new entities must be found. The method described in Paşca (2007) consists of 5 steps:

1. Identification of query templates that match the seeds
2. Identification of candidate instances
3. Internal representation of candidates
4. Internal representation of class
5. Candidate ranking

In Step 1, each user search query that contains a seed of a target class generates a query template: a lexico-syntactic pattern containing the remaining context around a matched seed. For instance the occurrence of 'vioxx' in the query 'long term vioxx use' generates the query template '$[\text{long term}]_{\text{prefix}}$ $[\text{use}]_{\text{postfix}}$.

In Step 2, a large pool of candidate instances is collected. All queries which match a query template found in Step 1 generate a candidate, the word(s) that co-occur with the query template. For instance the query 'long term xanax use' matches the template in the previous step and generates 'xanax' as a candidate.

In Step 3, an internal representation is made of the candidate instance. For all queries containing a candidate instance found in Step 2, the query template becomes an entry in a query template vector for that candidate instance. This vector is called the *search signature* of the candidate instance. The weight of an entry (the query template) is the frequency of the query template co-occurring with the candidate instance.

In Step 4, all created candidate *search signatures* in Step 3 are inspected to find the vectors which represent the seeds for the target class. The seed vectors are being added. This is the class *search signature*. It can be seen as a search fingerprint of the desired output for the particular class.

In Step 5, similarity scores are computed between the class *search signature* on the one hand and the candidate *search signature* of each candidate on the other hand. The similarity is computed by using the Jensen-Shannon similarity. The similarity score determines the order of the candidates in a ranked list.

The similarity is computed on a collection *C* containing all contexts patterns in the user search queries. In this *C* there is a collection C_q which contains all the context patterns which co-occur with the seeds. Similarly, there is a collection C_r part of *C*, which contains all context patterns that co-occur with a specific candidate. Thus, per candidate the collection C_r can change. Then there is the intersection of C_q and C_r, C_{qr} which represents the contexts which are found as well with the seeds as with the candidates. The visual representation can be found in Figure 1.

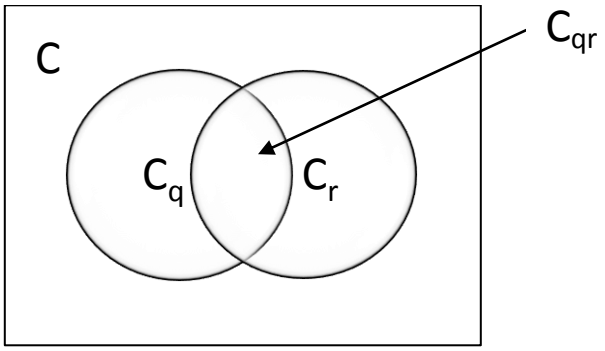

Figure 1: A visual representation of how the domain of context patterns looks like. C_q represents the context patterns co-occurring with the seeds. C_r represents the context patterns co-occurring with a particular candidate.

To compute the similarity, the *search signatures* are first transformed into probabilistic vectors. Each context pattern *c* in the class search signature *q*, is assigned the probability of co-occurrence with the seeds, *q(c)*. The probabilities, *q(c)*, become the new weights of the

entries which were raw frequencies. The probabilities are computed by dividing the raw frequencies of the entry by the total number of seeds in the data.

Each context pattern *c* in the candidate search signature *r,* is assigned the probability of co-occurrence with the candidate, *r(c)*. The probabilities, *r(c)*, become the new weights of the entries which were raw frequencies. The probabilities are computed by dividing the raw frequencies of the entry by the total number of occurrences of the candidate in all data.

The Jensen-Shannon similarity is based on computing the entropy. Entropy measures chaos. The less chaos there is between two vectors, the more similar the vectors are. Entropy is represented by the function *h(x)*. The Jensen-Shannon similarity value ranges between 0 and 1, 0 means in this case that the similarity is big, 1 means that there is no similarity. The way the Jensen-Shannon similarity is computed per candidate r is the following:

$$JS(q,r) = \log 2 + \frac{1}{2} \sum_{c \in C_{qr}} \Big(h\big(q(c) + r(c)\big) - h\big(q(c)\big) - h\big(r(c)\big) \Big), \qquad h(x) = -x \log x$$

Lee (1999) claims that the Jensen-Shannon similarity is the best measure to use for computing similarity in text processing. Lee (1999) for instance compares it to the cosine similarity, a similarity which is often used in text processing.

The cosine similarity is a measure of similarity which finds the cosine of the angle between the vector. The smaller the angle, the more similar the vectors are. The cosine similarity between two vectors ranges between 0 and 1, the value 1 means that the distribution of the vectors are equal, the value 0 means the opposite. The cosine similarity is computed by the following formula (*q, r, c,* and *C* stand for the same elements as explained for the Jensen-Shannon similarity):

$$\cos(q,r) = \sum_{c \in C_{qr}} q(c) r(c) \Big(\sum_{c \in C_q} q(c)^2 \sum_{c \in C_r} r(c)^2 \Big)^{-1/2}$$

The idea behind having a class *search signature* is to have a collection of query templates which represent what users ask about a specific class of named entities. By inspecting more query templates in the class search signature, guessing which class the templates represent gets easier each time. By creating a same kind of search signature, but then for a candidate, the system can guess by looking at what users ask about the candidate to which class it likely belongs.

Using search signatures also means that not only one context is responsible for a candidate to get a score. All contexts with which the candidate co-occurs influence the score. Depending on how similar the probability distributions are, a score is given. Therefore it is difficult to determine exactly why a wrong candidate has been extracted.

Paşca (2007) tests its approach on ten different entity classes. For all ten target classes, the first 250 candidates (named entities) on the ranked list have been manually judged on correctness. Next, precision scores have been computed at various ranks. The results in Paşca (2007) are promising. The average precision over all ten classes at rank 250 is 0.8 [4]. We will do similar evaluation.

[4] A precision of 1.0 at 250 means that all candidates up to and including 250 are correct.

Paşca (2007) states that the frequency of the seeds in the query logs does not have anything to do with the quality of the outcome. Moreover it is claimed that lower precision numbers are not necessarily meaningful. For some classes, like 'Country', there are just a limited number of candidates available so Paşca (2007) claims that precision cannot be 1.0 at rank 250 for this class.

Paşca (2007) also shows that more fine-grained classes do not lead to better results in all cases. The class 'Location' performs for instance better than the classes 'Country' and 'City'.

4.1.2. Pennacchiotti and Pantel (2006)

The goal of Pennacchiotti and Pantel (2006) is to harvest binary semantic relations from text with a weakly-supervised iterative algorithm which is called 'Espresso'. With a small set of seed instances for a given relation, the system learns the context patterns co-occurring with the seed entities. It uses the Web to filter and expand the instances. The work shows that Espresso extracts precise lists of various semantic relations.

To extract candidates representing a specific semantic binary relation, (e.g. 'is-a', 'part-of') 'Espresso' makes use of a small set of seeds as input (e.g 'Pablo Picasso is-a artist'). Starting from the seeds, 'Espresso' starts a four-phase loop:

1. Pattern discovery
2. Pattern filtering
3. Candidate discovery
4. Candidate filtering

Before this loop starts, one extra step is done. Pennachiotti and Pantel (2006) call this 'Term Definition'. Regular expressions are defined over PoS-tagged corpora to extract and define terms. A term can consist of one or more words. A term can for instance be 'record of a criminal conviction'. This way they are capable of capturing the part-of relation between 'record of a criminal conviction' and 'FBI-report'.

In the approach we implement, 'Term Definition' would not be a useful process. First of all we do not have a reliable PoS tagger. It is also questionable that if a PoS tagger is used this will assign the right PoS tag because the user queries are often not grammatically well-formed. Moreover we did not analyze the complex terms in the data analysis when investigating the unknown words; the focus is on single words.

Step 1 takes as input a set of instances *I'* and gives as output a set of context patterns *P*. The first iteration starts with manually provided seeds. First, all sentences containing an instance $i = \{x,y\}$ are being retrieved (e.g. for the relation 'is-a', $i = \{Pablo\ Picasso, artist\}$). Next, all sentences are generalized by replacing all terms by a terminological label (*TR*). Then, all context patterns linking x and y are extracted. The most frequent substrings represent the set of context patterns *P*. This generalization is done to ease data sparseness.

In Step 2, Espresso selects from the patterns *P* the most reliable patterns. The focus of 'Espresso' is more on precision than on recall, since some context patterns may have a high recall but a disastrous precision. For that reason, context patterns are preferred that are highly associated with the seeds. Therefore, the reliability of a context pattern is computed by a metric based on pointwise mutual information (pmi). This metric measures the

strength of the association between two events x and y. The formula for pmi is the following:

$$pmi(x,y) = \log \frac{P(x,y)}{P(x)P(y)}$$

The reliability r, of a pattern p, $r_\pi(p)$, is the average strength of association across each input instance i, weighted by the reliability of i. This can be found in the next formula:

$$r_\pi(p) = \frac{\sum_{i \epsilon I'} (\frac{pmi(i,p)}{max_{pmi}} * r_i(i))}{|I'|}$$

In this formula $r_i(i)$ is the reliability of an instance, defined in the next two steps, and max_{pmi} is the maximum pointwise mutual information between all patterns. I' is the set of candidates. The reliability measures range from zero to one, one is the most reliable, zero the least reliable. The instances which are supplied in the seed list for the first iteration all get a reliability score of one.

The pmi between instance i = {x,y} and context pattern p is estimated by the following formula:

$$pmi(i,p) = log \frac{|x,y,p|}{|x,*,y||*,p,*|}$$

The set of n most reliable context patterns are then selected for the next step. A limit is set to the number of candidates a context pattern is allowed to extract. Context patterns firing more than t candidates are discarded, no matter what its reliability score is.

In Step 3, all candidates co-occurring with a context pattern defined in Step 2 are extracted. If the number of extracted instances is too small to guarantee enough statistical evidence for the next iteration, a web expansion phase begins. New instances co-occurring with the context patterns are retrieved by using the search engine Google. Also a syntactic expansion is performed by 'Espresso': All sub-terminological expressions from a found term are extracted. For example, expanding the relation 'new record of a criminal conviction' part-of 'FBI report', the following new candidates are obtained: 'new record' part-of 'FBI report' and 'record' part-of 'FBI report'.

In Step 4, the reliability of all extracted candidates is determined similarly as determining the reliability of a context pattern in Step 3. The reliability of an instance i, $r_i(i)$ is the following:

$$r_i(i) = \frac{\sum_{p \epsilon P'} (\frac{pmi(i,p)}{max_{pmi}} * r_\pi(p))}{|P'|}$$

Here $r_\pi(p)$ is the reliability of the pattern, max_{pmi} the maximum pointwise mutual information between all patterns and all instances and P' the total number of previously claimed reliable patterns.

Espresso uses the most reliable *m* candidates, i.e. the highest scoring candidates, as input for the following iteration.

When a predefined stopping condition is met, the iteration over the four steps will stop. The stopping condition depends on the size of the corpus.

The corpora used are the CHEM dataset and the TREC-9 dataset. The CHEM corpus is a small dataset of 313.590 words extracted from a college level textbook on introductory chemistry. The TREC-9 dataset consists of 5951432 words, taken from a newswire text collection.

The semantic relations vary from the classical *is-a* and *part-of* relation to more exclusive relations like *succession* (e.g. Bush-Clinton) and *chemical reaction* (e.g. Hydrogen-Oxygen). The number of seeds used representing a semantic relation varies between 12 and 14.

Espresso is compared to two other systems. One approach is the one of Ravichandran and Hovy (2002): RH02. It learns context patterns from a set of seed instances of a particular relation. The other approach is an *is-a* extraction algorithm from Pantel and Ravichandran (2004).

For each output set per system per relation the precision is computed by extracting randomly candidates (50 for TREC-09 and 20 for CHEM). The quality is judged manually. The recall is computed relative to the Espresso's recall.

The results show that Espresso outperforms the other two methods on precision. Espresso extracts much fewer candidates with a higher precision but consequently with a low recall. For instance Espresso extracts 132 candidates for the relation *Part-of*, the RH02 extracts 12,828 candidates. The precision of Espresso is 80%, the recall is 1 (since it is recall relative to Espresso). The precision of RH02 of 35% and recall is 45.52.

5. Method

To be able to reduce the amount of unknown named entities Q-go comes across we compare two approaches to find unknown named entities in Q-go's user queries. From a set of user queries we want to create lists of entities belonging to a target class, i.e. a particular type of named entities. These lists can be transformed to gazetteers.

The first approach is based on the one Paşca (2007) describes. We call this first approach the Paşca approach. This approach is appealing because of its simplicity and good results in Paşca (2007). Moreover the data which is used are web search query logs: similar data as we will use.

The second approach is based on the Espresso algorithm described in Pennachiotti and Pantel (2006). We call this second approach the Espresso approach in. Pennachiotti and Pantel (2006) describe promising results when looking at precision scores. Although the method in Pennacchiotti and Pantel (2006) is not specifically on named entities it seems to be a solid method.

Our approaches are easily comparable because they have the same basis: the input is the same, the mechanism to extract candidates with is based on context patterns, and they gives a similar output.

Our approaches return a ranked list with candidates. The candidates are one word entities, we are not able to find multiword expressions. This is done to keep the pool of candidates small. Furthermore the data analysis described in section 2.1, is also not about multiword expressions, but about words.

Both our approaches extract the context patterns in the same way, therefore we start with describing that process. Next, we describe per approach what the steps are that have to be taken to go from context patterns to a ranked list of candidates.

After we have described the methods we describe the experimental setup, the testing procedure and the evaluation procedure. Last, we describe what the differences are between the two approaches.

Context pattern extraction

Given the seeds representing a target class, all context patterns with a maximum length of three around a seed are extracted. So the user query 'A flight from Amsterdam to Rome tomorrow' gives the following nine context patterns if 'Amsterdam' is a seed.

1. [a flight from] [seed]
2. [flight from] [seed] [to]
3. [from] [seed] [to rome]
4. [seed] [to rome tomorrow]
5. [flight from] [seed]
6. [from] [seed] [to]
7. [seed] [to rome]

8. [from] [seed]
9. [seed] [to]

5.1. Paşca approach

Choice of context patterns to use

We use the most frequent context patterns to extract candidates with. If a seed is present without any context, this "no context" also is a context. User queries often consist of only one word. A higher recall can be obtained by also taking 'no context' into account. We know that it can have a disastrous effect on precision.

Candidate extraction

First, we create vectors which represent the target class and each candidate for that specific class. These vectors are the *search signatures*. The idea behind the *search signatures* is explained in section 2.1.

Next, we compute the Jensen-Shannon and the cosine similarity between the class search signature and the candidate search signatures belonging to the class the class search signature represents. We compare the Jensen-Shannon similarity to the cosine similarity to find which similarity gives better results.

5.2. Espresso approach

Choice of context patterns to use

We use the most frequent patterns to find the most reliable patterns among them. We determine the reliability with the metric proposed in Pennacchiotti and Pantel (2006), based on pmi.

After computing the most reliable patterns we only keep the highest scoring patterns, i.e. most reliable, to do the candidate extraction.

Candidate extraction

With the most reliable context patterns, we extract all candidates. The context patterns that generate too many candidates are discarded. The exact threshold of the number of candidates that is allowed to be extracted differs per class and client. It depends on the preliminary results what the threshold is.

Then, we compute the reliability of all extracted candidates. Candidate reliability is also computed with the metric proposed in Pennacchiotti and Pantel (2006), based on pmi. The reliability scores are used to create the ranked list of candidate entities.

Next, we enter the iteration phase. The most reliable candidates are used as new seeds. The whole process, from extracting context patterns on, starts again. The number of iterations is not defined by a fixed stopping condition. We stop with iterating when the results start to get worse.

5.3. Experimental setup

Data We test the approaches per target class on data from one client per domain. The domains and clients are the same as those analyzed in the data analysis:

- KLM (aviation)
- ABN (finance)
- OHRA (insurance)

Each client's data represents a domain. Although we test on client data, we expect that the client-specific results can be generalized to the whole domain. We choose to use data from each domain separately and not data from all three domains together because context patterns are expected to be domain specific.

For each domain, we collect two datasets randomly from all the user queries they have received on the Dutch websites[5]. The sizes of the datasets are[6]:

- N=100,000 (per domain)
- N=200,000 (per domain)

We expect that increasing data will have a positive effect on the results. However, we test on both sizes to see whether a smaller dataset can already provide satisfying results. Increasing data often gives better results in NLP tasks but makes the process computationally more expensive which is unwanted, therefore the smallest dataset possible giving satisfying results is wanted.

Target classes We focus on three named entity target classes, as mentioned earlier in Chapter 3:

- Geographical location
- Product Name
- Company

Seeds Each target class is represented by a number of seeds. Each class is represented by the different seeds for each domain. The seeds chosen are seeds we expect the users want information about. All seeds chosen were also present in the datasets. Seeds can be multiword expressions[7].

- *Geographical location*
  The seeds for 'Geographical location' are Dutch location names. For 'Aviation' the target class is represented by international cities and countries; it is likely that users want to know things about the international locations they fly to. For 'finance'

[5] We expect most user queries to be Dutch, although we have seen during data analysis that sometimes questions are posed in a language different from Dutch.
[6] We only had access to 200,000 user queries in one of the client's data. Thus to be able to compare the results of the three clients we set the maximum number of queries to 200,000
[7] This is useful when creating the seed lists for the class Product Name. Product names tend to consist of more than one word.

and 'insurance', 'Geographical Location', is represented by Dutch cities since we take data from the query logs from the Dutch websites; probably users want to know locations of branches, or opening hours. The seeds per domain can be found in Table 14.

Table 14: Seed lists per domain for target class 'Geographical location'

Aviation	Finance	Insurance
Amsterdam	Amsterdam	Amsterdam
Paramaribo	Rotterdam	Rotterdam
New York	Utrecht	Utrecht
Londen	Maastricht	Maastricht
Barcelona	Eindhoven	Eindhoven
Parijs	Groningen	Groningen
Madrid	Zwolle	Zwolle
Nederland	Almere	Almere
Curacao	Tilburg	Tilburg
Stockholm	Amersfoort	Amersfoort

- *Company*
  For the target class 'Company' all three domains have different seeds that represent them. For 'Aviation' most of the seeds represent companies that have to do with traveling, this can vary from travel agencies, to airline companies, to a railway company. There is also one bank included because this was found in the user query data. The companies chosen as seeds for 'Finance' are in general big companies which are present on the stock market. For 'Insurance', the companies are companies which have a special agreement with the client which gives the employees of the company a discount on their insurance, other healthcare insurance companies, and companies that have something to do with healthcare, like an optician company. The seeds per domain can be found in Table 15.

Table 15: Seed lists per domain for target class 'Company'

Aviation	Finance	Insurance
Airfrance	Nuon	Pearle
Delta	Stamrecht	Dela
Arke	Unilever	Schadegarant
Lufthansa	KPN	ANWB
Cheaptickets	KLM	ING
NS	Shell	CZ
British Airways	DSM	Delta Lloyd
SGR	Vitens	KPN
Avis	Telfort	Vecozo
ABN Amro	Rabobank	Specsavers

- *Product name*
  The target class 'Product name' has for each domain client specific seeds. Each company has its own products. For 'Aviation', the products chosen are the products KLM provides. For 'Finance', the products are financial products, both general as ABN specific products. For 'Insurance', 'Product name' is not represented by products OHRA provides itself but by medical products. The seeds per domain can be found in Table 16.

Table 16: Client specific seed lists per domain for target class 'Product name'

Aviation	Finance	Insurance
Flying Blue Card	Levensloopplan	Bioresonantie
Silver Elite	Kinderbonussparen	Ooglidcorrectie
Flex award	Royaal rekening	Steunzolen
Award Miles	Prestige pakket	Logopedie
Frequent Flyer	Privelimiet Plus	Fysiotherapie
	Vreemde Valuta rekening	Prodimed
	Wereldpas	Bloedzuigerterapie
	Kapitaalgroei hypotheek	Hypnotherapie
	Directkwartaalsparen	Spiraaltje
	Bedrijfspandhypotheek	Bril

To be able to identify whether providing more seeds will increase performance, there are two seed lists per client, one of five seeds, and one of the same five seeds plus another five seeds[8].

We do not experiment with changing the seeds and see what the quality of seeds has on the results and whether there is an optimal seed list per target class per client. The explanation lies in the time consuming nature of manually judging the outcomes.

Number of context patterns We experiment with how many context patterns we allow to represent the target class. The different values of the number of context patterns we use to extract the candidates are[9]:

- 25
- 35
- 50

We restrict the number of patterns to a maximum to filter out the single time occurrences of the context patterns (hapax legomena) and to reduce computational power. Moreover it

[8] For 'Product name' for 'Aviation' only the first 5 products are provided since KLM does not have a lot of products.
[9] The maximum of 50 is based on the maximum of unique context patterns found in 'Geographical location', when using the first 5 seeds for the worst performing domain, 'Insurance' in this case. The maximum number of contexts was 52.

is interesting to see how many patterns are enough to give a good representation of the target class.

5.4. Testing Procedure

We want to get a good insight in what the influence of the various variables is per domain per class.

5.4.1. Paşca approach

The Paşca approach is ideal for extensive and structured testing. There are a lot of variables and they all can be tested for each class and each client.

We test the approach per class (n=3) per client (n=3) for seven variable settings[10]. A variable setting is a combination of variables. The exact variable settings can be found in Table 17.

Table 17: Variable settings which are tested per class per domain for the Paşca approach.

Variable setting	# of seeds	Size of dataset	# of contexts	Similarity measure
1	5	100.000	25	Jensen-Shannon & Cosine
2	5	100.000	35	Jensen-Shannon & Cosine
3	5	100.000	50	Jensen-Shannon & Cosine
4	5	200.000	25	Jensen-Shannon & Cosine
5	5	200.000	35	Jensen-Shannon & Cosine
6	5	200.000	50	Jensen-Shannon & Cosine
7	10	100.000	25	Jensen-Shannon & Cosine

In variable setting 7, the setting can be found in which the expansion of the seed list is tested. The other variables in variable setting 7, are the same as in variable setting 1; the smallest dataset and the fewest number of contexts. We only test expansion of the seed list once because we expect that a probabilistic representation of five seeds will be similar to the probabilistic approach of ten seeds.

5.4.2. Espresso approach

For the Espresso approach we did not do structured tests. What the influence is of changing a variable differs per class per domain. For instance for one target class we set the number of maximum patterns it was allowed to extract to 100, which was giving satisfying results. For another domain in the same target class, however, this gave very bad results and we had to set it to 2,000 before the results became better. We eyeballed the results after each test and then tweaked variables.

[10] In theory this would mean that in total 63 (7x3x3) tests will be done. However for 'Aviation' there is not a seed list consisting of ten products. Therefore the total number of tests is 62

5.5. Evaluation procedure

Manual examination of the results from the Espresso algorithm made very clear that the algorithm was not successful. For this reason, evaluation concentrated only on the Paşca approach.

Based on how well the variable settings perform, we manually judge an amount of results per variable setting per client per class. Correct entities of the class get a score of 1, incorrect entities get a score of 0. It depends on the client and on the class how many candidates are checked manually because the results can vary a lot.

Although we are not able to find multiword expressions, if part of a multiword expression is in the ranked list of candidates it is judged as correct. If for example "Los" is found on the Geographical location list, it is judged to be correct because "Los" probably is part of "Los Angeles".

Because of a bug in the methods we extract some candidates which are cut -off before a diacritic like a trema (¨ in ë) or accents (´, ` in é and è), e.g. Italië becomes itali, and Curaçao become cura. We judge these cases as correct.

On the manually judged items we compute precision scores at various ranks *N*. The precision at rank *N* is the number of extracted candidates that are correct named entities for that class up to and including *N,* divided by *N*.

Recall is also important to get a good picture of how well the approach works. However recall in this case is difficult to measure. The data is not annotated. Annotating the data to be able to compute recall would be too time-consuming.

This manual judging is affected by human subjectivity. Flaws can therefore be present in the results. It is difficult to set exact boundaries of what is allowed to belong to a class and what is not. Is a statement of account a financial product? Or is it more a financial service? Is a museum a geographical location? In the description of the target classes the boundaries of the target classes are described. In evaluation we try to keep these boundaries as much as possible in mind but in judging border cases human subjectivity is used.

5.6. Differences between the Paşca approach and the Espresso approach

One major difference is the way in which each method determines when a pattern is used to find candidates. The Paşca approach uses the most frequent patterns to find candidates. The Espresso approach uses the most reliable frequent patterns to find candidates. Moreover, the Espresso approach limits the number of candidates that each pattern is allowed to find. Patterns which generate too many candidates are discarded. Moreover, the Paşca approach allows 'no context' as being context, the Espresso approach does not allow this.

Next, the way the candidates are provided with a score differs. The Paşca approach computes the similarity between two vectors, based on entropy and on the angle between vectors. The Espresso approach gives the extracted candidates a score by means of pmi, based on association.

After finding the candidates, the Paşca approach returns a ranked list of entities. The Espresso approach runs several iterations of the pattern and candidate finding process and then returns a ranked list of entities.

The Paşca approach is tested and evaluated extensively. All seven variable settings are tested for each class and for each domain. For the Espresso, we did not perform extensive testing and evaluation because the contributing factors in the Espresso approach were less transparent, moreover by examining the preliminary results we found that further evaluation was not promising.

6. Results and discussion

In this chapter we will show which approaches were able to best reduce the amount of Named Entities.

First we describe and discuss the results of the Paşca approach. Following, we discuss the Espresso approach. Although the Espresso approach has been implemented, the approach has not been evaluated so we will not describe the results in detail. Next, we discuss which differences between the two approaches might have caused the differences in results. Then, we discuss the generality of the results. We end with a brief conclusion.

6.1. Paşca approach

We present and discuss the results of the Paşca approach in this section. First, we present the result of the two steps taken in the approach, first the extracted context patterns, second the extracted candidates. Next we describe how well the approach has performed, last we discuss how the results relate to Paşca (2007), we will discuss the differences in results and which differences between the approaches might have caused them.

6.1.1. Extracted Context patterns

Results

Context patterns that end up in a class search signature are not exactly similar for all clients. Classes can have domain related context patterns. For 'Geographical location' context patterns found for 'Aviation' are for instance 'van [] naar' *(from [] to),* 'vlucht naar []' *(flight to [])* and 'vanuit []' *(from[])*. All patterns are related to going from some place to another. For 'Geographical location - Finance', context patterns found are 'vestiging []' *(establishment []),* 'openingstijden []' *(opening hours []),* 'kantoor []' *(office [])* and 'filiaal []' *(branch [])*. All these patterns consist of words indicating locations of a particular branch of the company.

There are also overlapping context patterns. There are overlapping patterns across domains for one class, so for instance both 'Company-Insurance' as 'Company-Aviation' share contexts. But there are also overlapping contexts across classes per domain, for instance 'Company –Insurance and' 'Geographical Locations – Insurance' share contexts.

Examples of contexts can be found in Table 18. The first three examples are specific class-domain related context patterns. The last three examples show overlap of context patterns in and between classes and domains.

Table 18: Examples of context patterns used to find new entities in the data for a given class-domain combination

Class-domain combination	Context patterns
Geographical location – Finance	hoofdkantoor [ ] *(head quarters [])*
Geographical location – Aviation	vlucht [] *(flight [])*
Company – Insurance	lid van [] *(member of [])*
Company – Insurance	van [] *(from [])*
Geographical location – Insurance	van [] *(from [])*
Company – Aviation	van [] *(from [])*

In general we see that there are only a few very frequent patterns and that after the frequent ones many hapax legonema are found.

The frequencies of the context patterns vary among the class-domain combinations. Patterns with high frequencies do not always provide better results than patterns with low frequencies. This can be found in Table 19. We see for instance that the frequency of the context patterns with the highest frequency for both Product Name-Insurance and Product Name-aviation are high. When looking at the results Product Name-insurance has much better results than Product Name-aviation.

Table 19: Frequencies of the most frequent context patterns for a given class-domain combination including its precision score at the final manually judged rank.

Class-client	Highest frequency	Precision at final judged rank
Product Name – Aviation	686	.66 @50
Product Name – Insurance	225	.96 @200
Geo. Location – Finance	47	.92 @150
Product Name – Finance	3	.75 @100
Company - Finance	39	.16 @25

For all class-domain combinations the 'no context' context patterns is one of the first four most frequent patterns. For six combinations it is even the most frequent pattern. There can be big gaps between the 'no context' pattern and the next most frequent pattern. For instance for 'Product name – Aviation', 'no context' has a frequency of 686, the next context pattern has a frequency of 146.

Discussion

Some patterns are related to what the users want from the domain in general and the client in particular. This is a reason that keeping the domains apart from each other is important.

The overlapping patterns seem to be generic patterns. Generic patterns are not generic for only one class, they are generic over all classes. It seems not to be a good idea to get rid of generic patterns to increase precision. The generic pattern is part of the class search signature which is representative for the class, this means that the generic pattern is also representative for the class.

Perhaps intuitively, highly frequent context patterns are not as successful as might be first thought. This is for instance supported by 'Product Name-Finance'. The highest frequency of a context pattern used to extract candidates with is only three. However, it gives much better results than 'Product Name-Aviation' which has a much higher highest frequency (686). We think that this can be explained by the fact that probabilistic vector are used. High frequency does not necessarily mean a high probability for the context pattern. For instance 686 is a high frequency for a context pattern, but if that pattern occurs 5,000 times, the probability will not be so high.

The 'no context' pattern seems to be an important context pattern since it is for all class-domain combinations a high scoring pattern. It seems that if one wants to know something about an entity in a web search application it uses only that particular entity as a keyword.

So although a high frequency is not the key to success, it is definitely an important context pattern when creating a representative search signature.

6.1.2. Extracted Candidates

In this section we will take a closer look at the extracted candidates; at the correctly extracted candidates on the one hand and the incorrectly extracted candidates on the other hand. This will give again information about whether it was a good decision to keep the domains separate. We also can identify whether user query language is processed robustly. The incorrectly extracted candidates can give insight in why they are extracted so that the method can be improved.

Extracted candidates

Results

This section will show first per class and per domain what types of correctly extracted candidates have been found. The section is divided in four sections, the first three representing one class, the last one representing the incorrect candidates. The first three sections consists of a table representing per domain what types of correct candidates have been found together with some examples of the candidates.

When a type description in the table is bold, it means that this type was also present in the seeds which were representing the class for that specific client. The examples are taken from the results using the highest scoring variable setting[11].

The examples found are not necessarily only taken from the candidates which have been judged manually. Some examples come from further down on the list.

Geographical location In Table 20 we find extracted candidates for the target class 'Geographical location'. In 'Aviation', mostly international cities and countries are found. This is in line with the seeds provided for the domain for this class. For 'Finance' and 'Insurance', mostly Dutch cities and villages are found; the seeds representing those clients were also Dutch cities and villages.

If looking at what other types of locations are found next to the ones which are represented by the seeds, we see that for 'Aviation' airport codes and airport names have been found. For 'Finance', street names have been found and for both 'Finance' and 'Insurance' also countries are present in the results while they were not represented by the seeds.

In case of 'Aviation', we got the examples from the results obtained by variable setting seven in combination with the Jensen-Shannon measure. This means that ten seeds have been used which include also countries. In the first five seeds no countries are present. When inspecting the results obtained by the variable setting using only five seeds we also find countries.

We also find parts of multiword expressions like 'los', 'sint' and 'den' and words which are cut-off because of the diacritics, we find for instance 'itali' instead of 'italië'.

[11] By highest scoring variable setting we mean the variable setting with the highest score at the final manually judged rank.

For Aviation we find many spelling errors ('ansterdam' instead of 'amsterdam'), abbreviations ('adam' instead of 'amsterdam'), naming variants (airport codes) and spelling variants ('boedapest' and 'budapest').

Table 20: Types of correct extracted candidates per domain for the target class 'Geographical location' together with examples.

Domain	Extracted Candidates	
	Type Description	Examples
Aviation	**Cities**	Barcelona, Seattle, Melbourne
	Countries	Amerika, Engeland
	Airport codes	Sxm, sfo, ewr
	Airport names	Heathrow, Schiphol
Finance	Streetnames	Coolsingel, Osdorpplein, Meibergdreef
	Cities/Villages	Schijndel, Sassenheim , Bodegraven
	Countries	Dubai, Amerika, India
Insurance	**Cities/Villages**	Arnhem , Zoetermeer
	Countries	Japan, België

Product name For 'Product Name' we find that each class finds mostly products specific for the domain and often even more specific for the client itself. The examples can be found in Table 21.

Table 21: Types of correct extracted candidates per domain for the target class 'Product name' together with examples.

Domain	Extracted Candidates	
	Type Description	Examples
Aviation	**Products to get discount or extra privileges**	Awardmiles, tdc (travel discount certificate)
	Services provided	Boekingscode, stoelkeuze
	Products dealing with payment	Ideal, mastercard
Finance	**Bank account names**	Ondernemersrekening, Internetspaarrekening
	General financial products	microkrediet, leningen
	ABN specific financial products	Directsparen, groeigemak
	Insurances	Annuleringsverzekering, opstalverzekering
Insurance	**Medical therapies/ Treatments**	Paradontologie, Osteopathie
	Drugs	Concerta, Tamiflu
	Tools	Bril, Gehoorapparaat

In 'Aviation', products are found which give the user discounts or privileges. All seeds represented this type. Two other types are found as well, namely services which KLM provides and products dealing with payment.

Next to the three types which are represented by the seeds for 'Finance', one other type of products is found: insurances. The financial products are both general financial products as ABN specific products.

In the products found for 'Insurance', medical treatments and therapies are the most frequent. These types are together with tools represented by the seeds. A new type which is found in the results but not present in the seeds is 'drugs'.

For Aviation and Insurance we find many parts of multiword expressions, spelling errors, spelling variants and naming variants.

Company The companies which are extracted are different for each domain. The examples can be found in Table 22.

In the results of 'Aviation' mostly airline companies are found and secondly travel agencies. This is in agreement with the seeds provided for that class. A type which is found but not present in the seeds is the type representing companies which have something to do with transport other than aviation.

For 'Finance' no type classification is done. It is not possible to say something about the companies extracted because there are so few. In the first 25 extracted candidates using the best experimental setup only four companies have been found. Among those four correct candidates, two of them were seeds (KPN and Stamrecht).

Table 22: Types of correct extracted candidates per domain for the target class 'Company' together with examples.

Domain	Extracted Candidates	
	Type Description	Examples
Aviation	**Airlines**	Delta, Martinair
	Travel agencies	Corendon
	Companies that have something to do with transport	Avis, NS, Skyteam
Finance	Too few companies	KPN, Stamrecht, MVO, Telfort
Insurance	**Competitors**	CZ, VGZ
	Companies which have a care agreement with OHRA	KPN, Makro
	Companies providing medical products	Starshoe, Pearle

In case of 'Insurance' three types of companies are found. The type found mostly is 'Companies which have a care agreement'. All the types found are also present in the seed list

Incorrect candidates Not all instances we find are correct ones. We identify three reasons why incorrect ones are found:

- Generic context patterns; contexts which are generic and therefore will find many candidates and be present in many classes, e.g. the context patterns *in [] (in [])*, *met [] (with [])*, *bij [] (at [])*, *van [] (from [])*.
- Strong context patterns; contexts which are very representative for the class and domain but in some cases can be used in combination with words not belonging to the class.
- 'Wrong' choice of keywords: Mainly in the results for 'Insurance' when dealing with product names wrong extracted candidates can be caused by using a 'wrong' word, for instance if a user wants to know whether he or she will get a reimbursement for a treatment of the medical issue the user is suffering from, the user does not uses the treatment as a keyword, but the disease the user suffers from.

We do not present exact results on what has caused the errors since the results are due to a complete search signature. However by eyeballing the results it seems that generic context patterns are causing the most false positives, secondly strong context patterns and lastly the wrong choice of keywords. The three types can be found in Table 23 together with some examples

Table 23: Causes of false positives in the results with examples. The context patterns and the candidates are in italic, the candidates are red.

Generic Context patterns	Strong context patterns	'Wrong' keywords
Prod – KLM: Naar Amsterdam *met kl6068*, is dit haalbaar? (To Amsterdam *with kl6068*, is this realizable?)	Geo – KLM: Vervoer *van vishengel naar* Canada (*Transport *from fishing rod to* Canada, Transport *of fishing rod to* Canada)	Prod – OHRA: *Psoriasis vergoeding* *(Psoriasis reimbursement)*
Geo – ABN: Betalen *in winkels* buitenland (Pay *in shops* abroad)	Prod – OHRA: *Vergoeding ziekenhuiskosten* *(Reimbursement hospital costs)*	
Comp – ABN: Hoe zit het *met hotelkosten* *(*How is it *with hotel costs*, What about *hotel costs)*	Geo – ABN: *Openingstijden winkels* *(Opening hours shops)*	

Discussion

The results tell us that not all types of seeds necessarily have to be in the representative seed lists. What we also can infer is that using more fine-grained classes can lead to worse results because if two types show similar behavior wrong candidates will end up in the type which is wanted in the first place. For instance both countries and cities are present in the candidates extracted for Finance in the class 'Geographical location', while only cities were represented in the seed. Cities and countries seem to show the same behavior in the context patterns. If the class 'City' is defined, contexts will be found which also find countries, so also countries will end up in the results of 'City'.

Moreover we see that the entities retrieved are heavily dependent on the domain. This is underlines again the need keep domains separated.

By using only domain specific data from user queries we are aware of the fact that some named entities are missing in the classes we have proposed: named entities which are not mentioned in the user queries. We believe this is not a defect of the approach. The idea behind the approach is that users want to know those things of the client they are asking for. If a user never asks for a specific named entity, what is the use of knowing it is a named entity? For instance for the target class 'Company' it is not needed to know all the companies in the world, which is quite impossible too, we want to know the companies which users use in their user search queries for a specific domain. Looking at previous user search queries gives a good indication of which companies that have been. Moreover if a general list of companies is wanted which can be used for all classes, which data must be used?

We identify four types in the candidates which were also briefly mentioned in Chapter 2.

- Spelling errors: Entities which are not spelled correctly
- Abbreviations: Entities which are an abbreviation of an entity
- Spelling variants: Entities which have different correct ways of spelling
- Naming variants: Entities which are named differently by users. For instance international locations can be named by the Dutch name or by its official name (Florence – Firenze).

Therefore, we infer that the approach is able to process user query robustly. One might argue that most of these types can be corrected at this moment by means of spell correction; spelling errors, spelling variants and maybe even abbreviations. Nevertheless not all extracted candidates will be corrected and especially naming variants will not get corrected so it indeed is important that user query language is processed robustly. Moreover, spelling variants, abbreviations and naming variations are spelled correctly so actually they should not end up as words with spelling errors.

The strength of the approach in case of user query language follows from the following example: For 'Geographical Location-Aviation' the abbreviation 'adam' of the entity Amsterdam is found. The more known meaning of the entity 'adam' is: 'personal name'. But since the word has been found in 'Aviation' user queries it is much more likely that it is an abbreviation of Amsterdam, and indeed should belong on the list of named entities for 'Geographical Location – Aviation'.

We find that the characteristics of user query language are not present in all class-domain combinations to a same extent. For instance in 'Geographical Location-Aviation', many types of user query language are found, while for 'Geographical Location-Finance/Insurance' this is not the case. This can be explained by the fact that users often do not know how to write the names of non-Dutch locations, moreover there are differences in spelling between languages and some locations have completely different names in different languages. Dutch locations however are known to the users. Moreover it is likely that users want to get information on the locations in the neighborhood in case of 'Finance' and 'Insurance', e.g. for 'Finance' it is likely that the user wants to know the opening hours of a branch close to where he or she lives, so probably the user will spell the word correctly.

In the 'Product name' class, we find different types of user query language for 'Finance' and 'Aviation'. This can be explained by the fact that product names tend to be fancy long names representing a product, often consisting from more than one word. It is difficult to exactly remember the product name. Take for instance a product of KLM: "Flying Blue card". Users do not often mention this exact entity. Sometimes words are replaced by its Dutch variant, 'Flying Blue kaart' (*Flying Blue card*), or users misspell the entity because not all users are familiar with English or just make a mistake, e.g. "Flying Bleu kaart" (*Flying Bleu* card) or users abbreviate words for simplicity like 'fb kaart' (*fb card*), another possibility is to forget spaces, e.g. "Flying Bluecard". For the class 'Company' it is less likely that the mistakes mentioned above show up because a company is known under one name and less variability is possible.

Although we have identified three reasons why wrong candidates are found, it is difficult to avoid errors. We do not see a relation between generality of context patterns and bad results, e.g 'Geographical Location-Aviation' consists of many generic contexts (e.g. 'van []' (*from []*), 'naar []' (*to []*), 'in []' (*in []*), '[] en' (*[] and)*) but has also very good results, so generic contexts are important. 'Wrong' keywords will always be used since the user is in charge. For strong context patterns it is more likely that they will help identifying more correct candidates than wrong candidates.

6.1.3. Performance of the approach

Results

We present precision scores for two domains in one class for all seven variable settings. At each rank both the results obtained by the Jensen Shannon similarity (JS) are shown as the results obtained but the cosine similarity (Cos). Table 24 shows precision scores for 'Geographical location-Aviation'. Table 25 shows precision scores for 'Geographical location-Insurance'. Important to keep in mind is that the ranks at which the precision has been measured, differ between the two clients. The highest rank for 'Aviation' is 200, the highest rank for 'Insurance' is 50.

First of all we can see that the results of 'Aviation' (.995 @200) are better than the results of 'Insurance' (.52 @50).

The increase of data, that is the comparison of V1 to V4, V2 to V5 and V3 to V6, does not have effect on the results of 'Aviation' while for 'Insurance' there is a slightly positive effect noticeable. Increasing contexts, that is the comparison of V1 to V2 to V3 and V4 to V5 to V6, has no effect on 'Aviation' data but again a slightly positive effect on 'Insurance' data. The increase of the number of seeds from five to ten, the comparison of V1 to V7, shows no

effect for both 'Aviation' and 'Insurance'. The Jensen-Shannon similarity gives better results than the Cosine similarity for both domains.

Table 24: Precision scores at various ranks for all variable settings for Geographical location-Aviation. Behind the variable setting a specification can be found of the parameters, the first number indicates number of user queries, the second the number of context pattern and the last one the number of seeds.

Variable setting		Rank							
		@50		@100		@150		@200	
		JS	Cos	JS	Cos	JS	Cos	JS	Cos
V1	100,000; 25; 5	1	.94	1	.85	1	-	.97	-
V2	100,000; 35; 5	1	.9	1	.85	1	-	.975	-
V3	100,000; 50; 5	1	.84	1	.86	1	-	.98	-
V4	200,000; 25; 5	1	.96	1	.92	0.99	-	.99	-
V5	200,000; 35; 5	1	.96	1	.87	0.99	-	.99	-
V6	200,000; 50; 5	1	.96	1	.92	0.99	-	.99	-
V7	100,000; 25; 10	1	.84	1	.87	1	-	.995	-

Table 25: Precision scores at various ranks for all variable settings for Geographical location-Insurance. Behind the variable setting a specification can be found of the parameters, the first number indicates number of user queries, the second the number of context pattern and the last one the number of seeds.

Variable setting		Rank					
		@10		@25		@50	
		JS	Cos	JS	Cos	JS	Cos
V1	100,000; 25; 5	.7	.5	.6	.4	.4	.34
V2	100,000; 35; 5	.7	.4	.68	.36	.44	.34
V3	100,000; 50; 5	.7	.4	.72	.4	.44	.3
V4	200,000; 25; 5	.7	.4	.6	.36	.48	.36
V5	200,000; 35; 5	.7	.5	.72	.48	0.52	.34
V6	200,000; 50; 5	.7	.5	.72	.48	0.52	.34
V7	100,000; 25; 10	.8	.5	.64	.48	.38	.46

For completeness we present a qualitative summary of how the parameters in the variable settings influence the results in all class-domain combinations in Table 26. For each class-domain combination (rows) the influence of the four parameters in the variable setting (columns) is given. The four parameters are the following:

- Increasing data (from 100.000 to 200.000), comparing v1 to v4, v2 to v5 and v3 to v6

- Increasing contexts (from 25 to 35 to 50), comparing v1 to v2 to v3 and v4 to v5 to v6
- Increasing number of seeds (from 5 to 10), comparing v1 to v7
- Jensen-Shannon or Cosine, comparing the precision scores at the same rank for both measures in the corresponding variable setting

For the first three parameters we indicate in the table with +,- and +/- whether the variable has respectively a positive, a negative or no influence. In case of the similarity measure the number is given how often each measure performs better than the other measure. 'NA' indicates that there are no results available.

Table 26: Influence of the parameters in the variable setting per class per domain

Target class	Domain	Variables			
		Increasing data	Increasing contexts	Increasing # of seeds	JS / Cosine
Geographical Location	Aviation	+/-	+/-	+/-	JS 7
	Finance	+	+/-	-	JS 7
	Insurance	+	+	+/-	JS 6, cos 1
Product Name	Aviation	+	+	NA	JS 6
	Finance	-	+/-	-	JS 6, cos 1
	Insurance	+	+/-	+/-	JS 7
Company	Aviation	+	+	+/-	JS 4, cos 2
	Finance	-	+	+	JS 6, cos 1
	Insurance	+	+/-	+/-	JS 7

In Table 26, we see that all parameters have a different influence as well across classes as per class. For instance 'increasing data' shows for all three classes a different influence, 'Geographical Location' shows another pattern than 'Product name'. 'Increasing data' also shows a different influence in 'Geographical location' for all three domains.

In Table 27, we present the precision scores at various ranks for each class-domain combination to give insight in how each class-domain combination performs. We picked the precision scores from the highest scoring variable setting. By highest scoring variable setting we mean the setting that has the highest precision at the final evaluated rank for the specific class-domain combination.

In the results we present, the lowest final rank is @25 and the highest final rank is @200. To indicate that the precision on a rank is not measured the specific cell displays a hyphen (-).

What we can see in Table 27 is that some class-domain combinations show better results than other combinations.

In Paşca (2007) the same evaluation procedure is used. However, in Paşca (2007) ten target classes are tested and there are no specific domains. For all ten classes results have been manually judged up to rank 250. At rank 250 an average score is achieved over the ten classes of 0.8. The individual precision scores at rank 250 vary from 0.54 to 1. This is better than the results we achieve.

Table 27: Performance of the best variable setting per domain per client

Target class	Domain	Highest scoring variable Setting – Similarity Measure	Rank					
			@10	@25	@50	@100	@150	@200
Geographical Location	Aviation	V7 – JS	1	1	1	1	0.993	0.995
	Finance	V4 - JS	1	1	1	0.973	0.92	-
	Insurance	V6 - JS	0.7	0.64	0.48	-	-	-
Product Name	Aviation	V6 - JS	1	0.84	0.66	-	-	-
	Finance	V1 – JS	1	0.96	0.88	0.75	-	-
	Insurance	V6 - JS	1	0.96	0.98	0.98	0.96	0.96
Company	Aviation	V6 - JS	1	0.84	0.56	-	-	-
	Finance	V5/6 - JS	0.3	0.16	-	-	-	-
	Insurance	V5/6 - JS	0.6	0.48	-	-	-	-

Discussion

There are three variables that have been changed for each target class:

- Variable setting: Different variable settings have been tested for each class-domain combination.
- Data: For each class-domain combination, domain specific data has been used.
- Seeds: For each class-domain combination, different seeds have been used.

Therefore, we discuss the influence of these three variables which might explain the differences in results.

Influence of variable setting The parameters in the variable settings do not seem to be the reason why some class-domain combinations perform better than others. There is not one variable setting which performs best for each class-domain combinations and gives similar results for each combination.

While increasing data usually gives better results, it does not in this case. This might be caused by the fact that data is not increased enough. From 100,000 user queries to 200,000 user queries is not a very large step. Increasing data more might influence the results.

The number of contexts which represent the class-domain combination also does not have a big influence. In general there are only few context patterns which are very frequent and after the frequent patterns, the frequencies of the following ones decrease quickly. This supports the fact that users ask questions in a similar way. The infrequent context patterns are hapax legomena and therefore not needed because there is a small chance that the context pattern will occur again in future queries. So is seems the number of context patterns will reach a sufficient amount of context patterns and become stable afterwards. What the exact number is for reaching stability cannot be determined.

Increasing the number of seeds neither shows a clear effect. It seems that if the seeds are chosen representative for a class-domain combination, the probabilistic representation in the class search signature of five seeds will be similar to the probabilistic representation in the class search signature of ten seeds.

The influence of the similarity measure, however, is clear. The Jensen-Shannon similarity performs better than the cosine similarity. This follows the observations of Lee (1999).

Influence of data Productivity seems to be the best predicting factor of how successful the method will be. A class is productive for a domain if many entities of that class are observed in user queries.

Looking at the results we infer that the productivity of a class, and therefore success, can be predicted by what the core business is of the domain. If the class has something to do with the core business of the domain, the class is productive. For instance there are many geographical locations in 'Aviation' data. Transferring users from the one location to the other location is the core business of companies in 'Aviation'. The combination 'Geographical location-Aviation' shows good results. 'Company – Finance' does not show good results. The core business of companies in 'Finance' is helping the users by providing them financial services often in one of their branches. Their core business does not involve other companies.

One might argue that the productive classes are those classes which possibly contain many candidates. There are for instance many geographical locations, so it makes sense that many of them are found. This is true to a certain extent but then for instance 'Insurance' and 'Finance' should get similar results because both Dutch locations are expected to be found. However, we find that 'Finance' shows much better results. So only looking at how many possible candidates there might be in the class, is not a good predictor. One has to take the domain into account. For 'Finance' 'Geographical location' is a productive class, helping users in one of their branches is one of core businesses of the domain. For 'Insurance' 'Geographical location' does not have anything to do with the core business: providing reimbursements, not at some particular location. Therefore it seems to be a good decision to keep the domains separate.

One might think that the success is not caused by productivity on its own, but that the explanation also must be sought in small datasets and therefore infrequent entities. Intuitively, infrequent seeds will not find enough context patterns to create a good class search signature. However, Paşca (2007) shows that it is not about the frequency of a seed. Good context patterns must be found, this does not depend on the frequency of the seeds. If there are no good context patterns for a class-domain combination, it will not return good results, no matter how much data is added.

Influence of seeds If the seeds have caused the difference in results, the reason must lie in the representativeness of the seeds. The seeds representing each class and client have been picked by humans as being representative, but it might be possible that 'wrong' seeds have been chosen; unrepresentative seeds. In general, non-representative seeds lead to a worse class search signature so worse results are obtained.

An example of a bad performing class is 'Company'. It is possible that if the seeds would have been chosen better, in this case more representative, results might have been better. By more representative we can think of more fine-grained seeds. However, in an earlier section we show that this will not necessarily improve the results. Paşca (2007) supports this.

6.1.4. Comparison Paşca approach to Paşca (2007)

When comparing the results of the Paşca approach implemented in this thesis to the results of how Paşca (2007) did, Paşca (2007) shows better results. There are several differences which might have lead to the better results of Paşca (2007) so we discuss them here.

Multiword Expressions

Paşca (2007) extracts multiword expression and words, we only extract words. The results show that especially extracted product names for 'Finance' and 'Aviation' and extracts geographical locations for 'Aviation' contain multiword expressions. This might have had influence on the results. However, we do judge extracted candidates as correct when there is a hunch that the word is part of a multiword expression. But when context patterns consist of a prefix as well as a suffix, the multiword expressions co-occurring with these context patterns are not found, so candidate signatures of multiword expressions might not get as strong as they should be.

Context pattern length

Different from Paşca (2007) we choose to set a maximum length on the context patterns to not get hapax legomena influence the results. We think that the longer a user query is the more room there is for variation and the less likely it is that context patterns are being shared. We infer that instead of achieving positive results by using the complete context patterns we will achieve worse results because the patterns get too specific too find new candidates. The entity can often already be extracted by only a few words, e.g. we do not need the whole context around 'Amsterdam' in the query 'I want to go from Amsterdam to Rome'. 'From X to' already indicates that Amsterdam is an entity.

We also think that recall gets higher when cutting the length to four. We count all possible patterns around the seed so we find more generic patterns and therefore more entity hapax legomena. This is a positive effect instead of a negative effect.

Concluding, we infer that the results are not being negatively influenced by cutting the length of context patterns to a maximum length of four, instead it is more likely that it has had a positive influence on the results in comparison to Paşca (2007).

Number of context patterns

We have set a maximum to the number of context patterns which are allowed to represent the class for a client, Paşca (2007) did not. Above we explained that setting the number of context patterns to a maximum does not have influence on our results. We infer that it also does not have influence on the results we obtained compared to the results reported in Paşca (2007). Once there are enough contexts, adding hapax legomena will not give better results.

Data

Although both Paşca (2007) and the Paşca approach presented in these thesis test the approaches on user search queries, there is a difference in data. The data used in Paşca (2007) are search query logs of Google, the data used in this thesis are search query logs of a web search application at domain specific websites. Results have shown that is a fair decision to use domain specific data since there are clear differences between the domains.

We infer that keeping the domains separate has caused in our tests better results instead of worse results so this does not explain the differences.

We infer that the main reason for the differences in results is the different size of data used. In Paşca (2007) 4 million user queries are used while we have tested the approach on only 200,000 user queries. More data gives in general better results. Although this trend is not visible in the experiments described in the results, we do think that increasing data will help. Testing on 4 million user queries instead of 200,000 is a big difference, while testing on 100,000 user queries in comparison to 200,000 is not.

6.1.5. Summary

From all the findings described above we infer the Paşca approach is a good approach to create named entity lists for the classes we have tested. Although it is not a promising method for all class-domain combinations to a same extent, for productive combinations it is able to give good results.

6.2. Espresso approach

We implemented the Espresso approach but it was obvious from manual inspection that evaluating the results would not be promising. What we found by eye-balling the preliminary results was that filtering out the context patterns that extracted too many candidates was not giving better results. We also found that instead of getting a better performance, iteration caused a worse performance. We infer that this happened since wrong candidates have already been found, so if wrong candidates are used to find new context patterns with, wrong contexts will be found.

The 'Espresso' algorithm reported on in Pennacchiotti and Pantel (2006), did obtain promising results. There are differences between how we implemented the Espresso approach and how Pennacchiotti and Pantel (2006) did. We will discuss the differences which might have caused the bad results of the Espresso approach we have implemented.

6.2.1. Comparison Espresso approach to Pennacchiotti and Pantel (2006)

A reason for the fact that we did not achieve good results while Pennacchiotti and Pantel (2006), might be that Pennacchiotti and Pantel (2006) are looking for context patterns linking two words together. We only look for context patterns linking one word, the named entity. Linking two words gives less room for variability because a context pattern in between the exact two words is needed. Nevertheless, this probably is not the only reason for the differences since context patterns linking only one named entity can give good results which can be seen when we discuss the results of the Paşca approach.

We have tested our Espresso approach on smaller datasets than the datasets Pennacchiotti and Pantel (2006) have tested their algorithm on. Moreover Pannacchiotti and Pantel (2006) enter a web expansion phase. In general more data leads to better results, so it is possible that increasing data would give better results for the Espresso approach we have implemented.

However, we think that the main reason for performing worse in this research is that Pennacchiotti and Pantel (2006) test their algorithm on edited running text while we are testing the Espresso approach on user query language, unedited text. One reason to believe this is that Pennacchiotti and Pantel (2006) compare the results of their work to an algorithm which is originally tested on unedited running text. The algorithm to which

Pennacchiotti and Pantel (2007) compare the results of Espresso is presented in Ravichandran and Hovy (2002). Ravichandran and Hovy (2002) achieve good results when testing on unedited running text. Pennacchiotti and Pantel do not achieve good results with the approach of Ravichandran and Hovy (2002) when testing it on edited running text. We infer that approaches suitable for edited text are less suitable for unedited text and vice versa.

6.3. Paşca approach vs. Espresso approach

The Paşca approach gets better results for each class-client combination compared to the Espresso approach. The extracted context patterns are quite similar even though they are in a different order. However, the found candidates are different. There is one big difference in the contexts, the Paşca approach allows 'no context' as a context pattern, the Espresso approach does not allow the 'no context' pattern.

Earlier we inferred that the 'no context' pattern was important to create a representative search signature. It might have been also an important context in the Espresso approach, however, we do not allow context patterns that extract too many candidates. Since the 'no context' pattern is one of the top four highest frequency patterns, this probably would mean that the pattern would have been filtered out by this rule and therefore would not have had any influence on the differences between the two approaches. Therefore we must seek the explanation in another factor.

We infer that the difference in results lies in how the reliability of the patterns are computed in the Espresso approach on the one hand, and computing the similarity between search signatures in the Paşca approach on the other hand. Creating search signatures and compute similarity seems to be a better method to use on user query language than computing the reliability of patterns and candidates based on pmi. The latter seems to better for edited text. This follows the observations we have made in the previous section about the differences between how we implemented the Espresso approach, and how Pennacchiotti and Pantel (2006) did. This makes sense because a search signature takes more information into account. A representation is made for a class. All context patterns found for the class say something about the suitability of a candidate. If a candidates meets the requirements of the class it is likely that it is a candidate. The metric based on pmi only takes each context pattern and each candidate on itself into account and not specifically the class on its own.

6.4. Generality of the results

Although the approaches have been tested on domain specific data provided by Q-go we believe that the results can be interpreted in a broader sense. We think that some approaches are better suitable to process user query language, and other approaches are better suitable to process edited running text.

Paşca (2007) claims to be presenting the first endeavor in named entity discovery from user search queries. We have tried to use the same approach to find named entities in Q-go's user search queries. It seems that the approach works on this data as well. So we believe that the approach is suitable for all user search queries in all web search applications.

We think that all user search queries are similar in a way. They all contain noisy underspecified user information needs, but when looking at a lot of noisy user queries

there is a pattern visible in how users make their user search queries, how users ask for information on subjects and on which subjects users want information for a specific class, and even for specific domains.

We are not sure how applicable the Paşca approach is to other NLP applications. We think that the approach must be tuned to the application, for instance whether edited text or unedited must be processed. We do think that the underlying idea, of bootstrapping for candidates by using context patterns is applicable to many NLP applications. For instance for Machine Translation, contextual cues can be used to determine what the unknown word is. But also for Speech Recognition contextual cues can give important information about what is said.

The approaches are tested on three target classes. We infer that the approach can be used successfully for other classes as well if the class is productive for the domain. Moreover, also gazetteer lists can be made for one particular 'product'. If one is for instance interested in what types of accounts users want information on, the seed 'account' can be provided to the approach and all types of accounts can be found by looking at the context patterns.

6.5. Conclusion

We believe that the Paşca approach is more suitable approach to reduce the amount of unknown words in raw user search queries than the Espresso approach. User search queries provide (good) enough contexts to get good results. The results are promising although not for all classes.

The factor influencing the results most seems to be the productivity of a class for a domain. If a domain has productive data for a given class, the results are good. Increasing data, context patterns or number of seeds do not show clear trends on the results. We believe that Jensen-Shannon similarity is a better similarity measure than the cosine similarity. Also representativeness of seeds might have influenced the results. However this is not investigated.

The Paşca approach is able to process user query language robustly. Moreover, we again see that it is important to keep the domains separated.

Although the research is done on data provided by Q-go, we expect that the Paşca approach is applicable to all kinds of entity extraction from user search queries.

7. General conclusion and future work

7.1. Summary

In this thesis research is done to reduce the amount of unknown words in user search queries. The user search queries are provided by Q-go. One of the challenges is dealing with unedited texts and even more specific with user query language. A type of data which has been not investigated a lot.

Spell correction is often needed in user search queries. Users tend to make mistakes easily. However, the spell correction must not be too over generating when correcting words. A data analysis showed that most unknown words, spelled correctly, end up as a spelling error. This teaches us that when dealing with language the user uses in search applications, it cannot be blindly trusted that only the words which are left over by the spell correction are the actual unknown words.

Named entities cover the biggest part of the actual unknown words (28.4%). The focus in reducing the number of unknown words was on this type.

Two context-based approaches have been investigated to reduce the number of named entities because context-based approaches add a semantic interpretation to a word which was an important requirement. Context-based approaches also find as many named entities as possible with a same approach and they are able to deal with user query language. We reduce the number of unknown words by creating gazetteers, i.e. doing offline recognition.

We propose the Paşca approach as being a good approach to reduce the number of unknown named entities. The approach does not require specially prepared data. How successful the approach is, depends on the productivity of a domain in the data. It is necessary to keep data for different domains separate.

An important aspect of the approach is that it is indeed capable of handling user query language. Different characteristics of the type of language people use in unedited text have been found by the approach. These characteristic are important to recognize because entities can be found which humans might not think of and therefore would never end up in the resources.

The approach can be applied to many other applications, although not necessarily in this exact same form.

7.2. Future work

We recommend a number of improvements to the current study to get better results.

At this moment multiword expressions are not yet being recognized by this method. We expect by doing this that results get better. Especially product names often consist of more than one word. Finding those would get more reliable results.

Research can be done on the seeds. The seeds must represent the class. It might be possible that gains could be made by using better seeds. Together with this it might be worth looking at more fine-grained seeds to get more fine-grained classes. It was difficult to find good contexts for companies. But if some specific seeds would have been chosen representing a specific vertical of companies, better results might be achieved.

It is not yet known how good the results are in terms of unknown words. In this research no use has been made of dictionaries and other linguistic resources, i.e. every found entity was an unknown word. This is not how reality looks like. In reality linguistic resources are used, so if a good picture is needed of how well the approach performs as extension, a closer look must be taken at what words have been found which were not available in the resources.

At this moment offline recognition is done. However in web search applications users want to get answers immediately and not wait for a day before their question is answered. Online recognition therefore would be wanted of unknown words. It remains to be seen whether this indeed is possible with approach proposed in this thesis. At this moment the combination of the contexts with which the candidates co-occurs is needed to give a score in terms of how likely the candidate will be a correct candidate. When doing online recognition the system only knows the one context co-occurring with the unknown words. Thus only the probability score of the context in the class signature gives an indication. It is interesting how well the contexts perform on this task and whether this could stimulate online recognition.

More research can be done to whether the approach is applicable in other languages. For languages similar to Dutch we expect it to be a good approach, although it cannot be used in the way the approach is implemented at this moment. First multiword expressions must be extracted correctly because for instance for English many 'words' are multiword expressions. In English, words cannot be added together without a space as in Dutch.

Bibliography

Adler, M., Goldberg, Y., Gabay, D., & Elhada, M. (2008). Unsupervised Lexicon-Based Resolution of UnknownWords for Full Morphological Analysis. In *Proceedings of ACL-08*, (pp. 728-736). Columbus.

Baldwin, T., Bender, E., Flickinger, D., Kim, A., & Oepen, S. (2004). Road-testing the English Resource Grammar over the British National Corpus. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004)*. Lisbon.

Bouma, G., Mur, J., van Noord, G., van der Plas, L., & Tiedemann, J. (2005). Question Answering for Dutch using Dependency Relations. In *Proceedings of the CLEF 2005 Workshop*.

Brants, T. (2000). TnT: a Statistical Part-of-Speech tagger. *ANLP 6* .

Brill, E. (1995). Transformation-based Error-Driven Learning and Natural Language Processing: A Case Studi in Part-of-Speech Tagging. *Computational Linguistics* , 543-565.

Ciaramita, M., & Johnson, M. (2003). Supersense Tagging of Unknown Nouns in WordNet. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, (pp. 168-175).

Cruys, T. v. (2006). Automatically Extending the Lexicon for parsing. In *Proceedings of the eleventh ESSLLI student session*.

Cucerzan, S., & Yarowski, D. (1999). Language Independent Named Entity Recognition Combining Morphological and Contextual Evidence. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, (pp. 90-99).

De Meulder, F., & Daelemans, W. (2003). Memory-based named entity recognition using unannotated data. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003* , (pp. 208-211). Edmonton.

Guo, J., Xu, G., Cheng, X., & Li, H. (2009). Named Entity Recognition in Query. In *Proceedings of the 32nd international ACM SIGIR conference* , (pp. 267-274). Boston.

Hearst, M. (1992). Automatic Extraction of Hyponyms from Large Text Corpora. In *Proceedings of the 14th International Conference on Computational Linguistics*, (pp. 539-545). Nantes.

Jones, R., & Riloff, E. (1999). Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping. In *Proceedings of the sixteenth National Conference on Artificial Intelligence*, (pp. 474-479).

Kozareva, Z. (2006). Bootstrapping named entity recognition with automatically generated gazetteer lists. In *Proceedings of the Eleventh Conference of the European Chapter of the*

Association for Computational Linguistics: Student Research Workshop, (pp. 15-21). Trento, Italy.

Lee, L. (1999). Measures of distributional similarity. In *Proceedings of ACL-99*, (pp. 25-32). Maryland.

Mikheev, A. (1997). Automatic Rule Induction for Unknown-Word Guessing. *Computational Linguistics* , 405-423.

Miller, G., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. (1990). Introduction to WordNet: An On-line Lexical Database. *International Journal of Lexicography* , 235-244.

Nakagawa, T., Kudoh, T., & Matsumoto, Y. (2001). Unknown Word Guessing and Part-of-Speech Tagging Using Support Vector Machines. In *Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium*, (pp. 325-331).

Paşca, M. (2007). Weakly-Supervised Discovery of Named Entities Using Web Search Queries. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, (pp. 683-690). Lisbon.

Paşca, M., & Harabagiu, S. (2001). The Informative Role of WordNet in Open-Domain Question Answering. In *Proceedings of the NAACL 2001 Workshop on WordNet and Other Lexical Resources: Applications, Extensions and Customizations*, (pp. 138-143). Pittsburgh PA.

Pennacchiotti, M., & Pantel, P. (2006). A Bootstrapping Algorithm for Automatically Harvesting Semantic Relations. In *Proceedings of ACL-2006*. Sydney.

Ravichandran, D., & Hovy, E. (2002). Learning Surface Text Patterns for a Question Answering System. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, (pp. 41-47). Philadelphia.

Tseng, H., Jurafsky, D., & Manning, C. (2005). Morphological Features Help POS Tagging of Unknown WOrds Across Language Varieties. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing* .

Woods, W., Bookman, L., Houston, A., Kuhns, R., Martin, P., & Green, S. (2000). *Linguistic Knowledge can Improve Information Retrieval*. Mountatin View, CA, USA: Sun Microsystems, Inc.