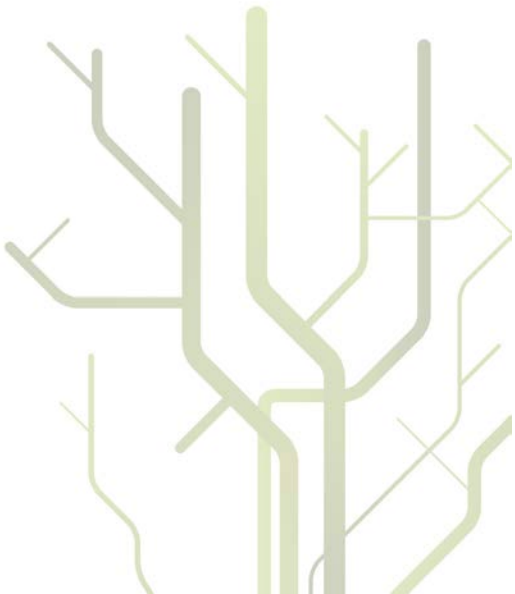
UNIVERSITY OF TROMSØ UIT

FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Detecting Events in Videos Using Semantic Analytics of Subtitles

Erik Bræck Leer

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



Abstract

Recently, television broadcasters such as the NRK and TV2 channels, have begun offering live internet television and movie archive along with their regular schedule, much like the known video archives such as Youtube and Vimeo. The amount of all television offered reduces the ability of the user to get an overview of the programs that are available at any given time, making the user will probably miss important events. Regular indexing engines for recommending does not generally work on media since it is hard to index media data. Tags and keywords describing a media file does only describe the whole file, making it difficult to use them for indexing and recommendation of specific scenes within the media.

This thesis presents a text event detection system for discovering interesting events based on video subtitles. By performing textual analytics, our system is able to discover events that are not discoverable through regular syntactic search. We have, based on related work, extended algorithms used for discovering semantic relationships between different words. Also, we have experimented with several algorithm for capturing the essence of each sentence, relating the prominent sense of each sense towards the events.

Our experiments illustrates how we can increase the accuracy of the algorithm used by performing a context exploration based on event keywords. The results shows that our system improved the base algorithm of the prototype by including more relevance methods like consecutive sentence similarity and similarity based on sentence internals.

Acknowledgement

I would like to thank my supervisor Dr. Håvard Johansen who's inspiration, constructive criticism and support proved invaluable. I also want to thank my co-advisor Professor Dag Johansen for his enthusiastic ideas, feedback and his great motivational talks.

I would also like to thank Dr. Cathal Gurrin from the Dublin City University for providing good ideas and an insight to information retrieval. I would also like to thank the rest of the iAD group for their support and input on how to do proper research.

My gratitudes to my fellow students Terje, Arild, Johan, and Øyvind for good ideas and valuable input during the whole process and for five great years. Øyvind, could not have done it without you.

Finally, I would like to thank my girlfriend, my family and my friends for their support and for always believing in me.

Table of Contents

List of Figures

Listings

Chapter 1

Introduction

Consumers have a large selection of TV channels to watch. With traditional TV broadcasters, like the Norwegian NRK and TV2 channels, making ever more content available over the Internet and with the ever growing content of online video archives like YouTube and Vimeo, the selection of content available to each consumer is becoming even larger. The unfortunate consequence of this trend is that users must spend ever more time navigating between available video channels in order to find interesting content. This is similar to the problem that the web encountered once it started growing beyond a certain size. Here, keyword search engines like Alta Vista, All the Web, and eventually Google, helped solving the problem. Video channels cannot as easily be indexed since the media data is constantly changing. It is however possible to extract and analyse the text that accompanies the media data such as subtitles.

To help users find an interesting video feed to watch, several methods have been proposed and are in use today. They range from simple reminders, as for instance, alarms and similar apps used on a smart phone, to advanced concept detectors, as for instance the TubeTagger and TubeFilter [1] that can be applied to home media centres. These concept detectors can scan several streams of media data and alert the user when some interesting event has taken place. A trend is to use annotated tags or keywords as the base for recommending. For instance, the DAVVI system [2] which is a prototype of the next generation multimedia entertainment platform, scrapes for instance live-commenatry from internet sites and use them to annotate soccer videos.

Two different commentators will typically have two unique ways of describing the same event, meaning a regular syntactic search will not be able to recognize the similarity between them. Another technique, yet to emerge in recommender systems, is the use of natural language processing (NLP) to detect events based on text and match it against user defined preferences. Only a semantic search can recognize the relatedness between the words and the concepts they represent by the use of for instance, a lexical database.

Natural language processing is based on text recognition and the sources for such a comparison could be any text ranging from sentences, subtitles, live-commentary internet-sites and speech-to-text systems. There exist several applications that can evaluate a word against another word [3, 4]. However, to our knowledge, no known application that can compute a satisfactory short text or sentence similarity. Existing solutions tend to not successfully calculate relationships between a term and a sentence [5].

Such a system we are proposing, is currently as of our knowledge, not used today. We find systems that are able to compute similarity for syntactic and semantic probability, they are however yet not combined in any system. Due to low efficiency and some in-correctness discovered in these systems [6, 7], these systems must either be modified or completely redesigned in order to produce a more correct result based on semantic similarity towards short text and sentences. Based on earlier work regarding semantic relatedness systems, we will look at the possibility of generating a context consisting of related words to an event, that is later used for similarity comparison. This can improve the event detection drastically because the reference probability is based on a more knowledgeable text [8].

1.1 Problem Definition

This thesis shall design, implement, and evaluate a similarity calculation service, based on probabilistic approaches. The focus will be on constructing algorithms that supports semantic similarity calculations between text. The basis for evaluating the algorithms will be by using human judges.

1.2 Interpretation, Scope, and Limitations

This thesis will explore techniques for systematically discovering similarity between text by using a semantic text analyser. The idea of our work is to build a system for automatic similarity calculation between given input.

Comparing text can prove difficult since a sentence may be interpreted in several different ways based on the context that it is evaluated in. Therefore, we propose an alternative way of discovering relatedness between text. We propose a system that will compare text towards known events. These event could be very specific, such as "someone crying", "an assassination" or the more generic event as for instance, "an emotion" or "a kill". By constructing a set of events and calculate the similarity between those events and the text, the relatedness between two set of texts can be discovered. The computed similarity describes the relatedness between the different set of texts, as defined by the event.

1.3 Methodology

The final report of the ACM Task Force on the Core of Computer Science divides the discipline of computing into three major paradigms [9]. These are theory, abstraction and design. Below is a short summary:

Theory is the mathematical approach rooted in development of a coherent and valid mathematical principles. Theorems about objects are proposed, and the scientist seeks to prove them in order to find new relationships and progress in computing.

Abstraction is related to the natural science, where the study of objects are done by modelling and simulate the different processes the scientist is investigating. The scientist seeks progress by formulating and testing hypotheses about algorithms and architectures.

Design is rooted in engineering. The scientist first formulates a problem. Then designs and builds prototypes in systematic order. These are compared in order to find the best solution to the given problem.

This thesis will utilize the design process of engineering. We have stated a problem and will systematically design and build a prototype to solve it. After the prototype is built, we will construct a series of experiments in order to evaluate our solution in contrast to the given problem. The prototype will be built according to similar approaches and ideas from related work.

1.4 Context

This thesis is a part of the information Access Disruption (iAD). The iAD Center targets core research for next generation precision, analytic and scale in the information access domain. The project investigates structuring techniques for future-generation large-scale information access applications. Most multi-media content and heterogeneous sensor data input must be supported, but more importantly is the real-time aspects involved. As such, next generation information access systems must blend in with data streaming application querying, processing, and delivering real-time data.

DAVVI is a prototype of the next generation multimedia entertainment platform. It delivers a personalized experience to a user in form of recommending video that is annotated based on live-commentary sites. Based on the annotation, the system offers a highly customizable video composition service that can, for instance, compose highlights of events on the fly and deliver it to the user in a torrent similar way.

For delivering video data, the system uses well known solutions, as for instance, data segmentation where each video segment is a self-contained media file. These segments can be delivered by simple HTTP GET requests. As feature extractions yet are very resource demanding and not accurate enough, metadata for annotation is scraped from trusted sites on the Inter-

net. The extracted metadata is converted from unstructured commentary text to annotations based on minute accuracy for their soccer example. The data extracted for annotation are scraped from several different sites. Since two people may use two different completely different words to describe the same event, these may be lost when searching for a given event. Therefore, we propose using NLP to discover relatedness between words that possibly represent the same concepts.

The DAVVI system has been described and we have discovered a problem with the current use of metadata used for annotation. By applying NLP analysis we hope to solve the problem.

1.5 Outline

The rest of this thesis is structured as follow. Chapter 2 contains relevant background information on topics this thesis explore, including both technical reviews and related work. The design is described in chapter 3 and the implementation in chapter 4, while chapter 5 presents the experiments and evaluation. Chapter 6 will conclude this thesis.

Chapter 2

Background

This chapter will introduce important concepts and topics for this thesis. We must evaluate and analyse text in order to calculate the relatedness between textual input. To calculate similarity and relatedness between text, we need to first analyse it, for this, we use Natural Language Processing (NLP). Tools for analysing words and sentences within NLP include part-of-speech (POS) tagging and word sense disambiguation (WSD). These tools locate and discover the sense and concept that the word represents. Used in NLP, is a collection of text that can be annotated and used to train classifiers such as POS taggers and WSD. Such an annotated text is called a corpus and will be introduced. To achieve a common level of word understanding, we look into ontologies to establish relationships between words. There exist several algorithms and approaches that calculates similarity between two different concepts. To produce results, ranking and recommendation are key elements and will be presented. In the end, we will summarize the different concepts that are the background for this thesis.

2.1 Natural Language Processing

NLP is a branch in computer science and linguistics dealing with the understanding of natural text for a computer. The field spans several areas in computer science such as Artificial Intelligence (AI), computational linguistics, statistics, and machine learning. Its history stretches from around 1950 until today. Alan Turing made the concept and field popular by his paper regarding an imitation game [10]. The paper states the question "Can machines think?" leveraging the idea of a learning machine.

NLP is not only about understanding, but also the ability to manipulate data, and in some cases, produce answers. Creating good interpretations does however require large amount of information to such a system. In this process we tend to utilize machine learning, making models or concepts of problems and solutions solvable to some AI, thus making NLP a sub field

of said AI. Algorithms used today utilize statistical machine learning, that is algorithms allowing computers to evolve behaviours (in this case, recognition or understanding) based on empirical data sets. Machine learning uses concepts from probability theory, pattern recognition, data mining, and statistics to mention a few.

NLP problems are under heavy research today. The IBM Watson[1] is an example on a complete system that is able to translate from human speech to text and process the given query related to the asked question, to ultimately produce a result satisfying the questionnaire. It does however require a data center with different data stores, schemas, and ontologies to power the Watson. The NLP problems that are most relevant to this thesis are the POS tagging, parsing and WSD. NLP is also strongly connected with information retrieval (IR) for searching and retrieving information by using some of these concepts.

A word can be classified into several linguistic word types. These could for instance be *hypernyms*, *homonyms* and *hyponyms*. A *hypernym* is a less specific instance than the original word. *Hyponyms* are the opposite, a more specific instance of the given word. *Homonyms* are equal words or synonyms.

2.1.1 Part-of-speech Tagging

POS tagging [11], is used to tag grammar within a text. It could for instance be to locate the word category of a given word within a sentence. In a system using NLP, a word can have several meanings, each based on what POS the word in the given context has. In English, it is common to learn nine part of speech: noun, verb, article, adjective, preposition, pronoun, adverb, conjunction, and interjection. A user may also find that there exists many more categories and sub-categories. In many languages, words (typically nouns) are also marked for their "role", grammatical gender, and so on within the sentence. Verbs are marked for tense, aspect, and other things. Discovering the relationship between words in a phrase, sentence or paragraph in known as the identification of nouns, verbs, adjectives and adverbs etc. Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms that associate terms.

POS tagging is hard since words can represent more than one part of the speech. For instance, the word "saw" can be represented as both a verb and a noun. The sentence: "He saw two dogs", can represent the meaning of a man seeing two dogs based on that "saw" is past tense of "see". However, if the word is flagged as a noun, we mean the cutting instrument used on e.g. a tree. For a human judge it is easy to differentiate between the different meanings based on how it is used, however, it may prove very hard for a

[1] http://www-03.ibm.com/innovation/us/watson/index.html

computer to do the same task. By performing semantic analysis on the sentence, a computer may find it unusual if the word "man" is followed by "saw" and "dogs" if all are nouns. If the computer analyse the POS of each word, it can conclude that the series noun-verb-noun is more probable than noun-noun-noun. For this reason, POS taggers are often trained with supervision by a human judge before used.

2.1.2 Word Sense Disambiguation

WSD is one of the main problem areas within natural language processing. WSD analyses a sentence and based on statistics, it tries identifying which sense of a word that is used in the sentence, if the word has several different senses. Its uses are within other field of computer linguistics and can, for instance, improve relevance of search engines. Research has progressed steadily to the point where WSD systems can achieve accuracy on 90-96% [12] on a variety of word types. Several techniques have been used to achieve this accuracy, which range from machine learning and trained classifiers to lexical resources like dictionaries and clustering of words and concepts.

To explain word sense disambiguation, consider two examples of the distinct senses that exist for the word "bass": It could be (1) a type of fish or (2) tones of low frequency. In a sentence it is used as (1) "I went fishing for some sea bass" and (2) "did you hear the bass line of that song?". For a human judge, it is obvious that the the two sentences use the different senses of "bass", however, it is not obvious to a computer.

Supervised learning approaches have been the most successful algorithms to date. Using the English language, accuracy at the coarse-grained level is above 90%. The simplest possible algorithm of always choosing the most frequent sense has a standard accuracy of between 51.4% and 57% [12].

WSD requires a list of senses that are to be disambiguated and a corpus containing language data to be disambiguated. The actual process have two variants: (1) Either all words within a text are processed from disambiguation or (2) just a small sample of short text previously processed for a more accurate precision. The former comprises disambiguating the occurrences of a small sample of target words which were previously selected, while in the latter all the words in a piece of running text need to be disambiguated.

The difference between word sense disambiguation and part-of-speech tagging is that WSD is used to discover the sense behind a word, thus require a more complete sentence while POS taggers are able to determine the part of speech simply by examining the adjacent terms.

2.1.3 Corpus

In linguistics, a *corpus* is a large set of text that can be structured. A corpus is used to do statistical analysis and testing hypothesis and may occur in

a single to multiple languages. To make a corpus more useful for linguistic research, they are often subjected to processing and annotation [8], although we find use of corpora as unprocessed text.

A lemma is the headword, that is the core of a word. For instance, in English, jump, jumping and jumped all originates from the lemma jump. When performing NLP analysis, POS tagging is often used. The part of speech could be information like verb, noun and adjective, while lemma processing requires us to find the base word. The size of corpora varies, annotated corpora are usually smaller for consistency reasons and can typically contain from around 100k to three million words [13] annotated, although it is possible to find corpora containing only 3000 words.

It is possible to further structure corpora, these are called treebanks or parsed corpora [13]. Here we also find that further processing and analysis are applied. Corpora are the main knowledge base in corpus linguistics. In computer linguistics, speech recognition and machine translation do all use corpora for analysis and processing. There are several notable corpora used today. The most known corpora is the Brown corpus by Kucera and Nelson from Brown University [14]. It is used in several academic and proposed work. Other notably corpora are the American National Corpus, Bank of English and British National Corpus, which are widely used together with the Corpus of Contemporary American English.

2.1.4 Information Content

Information content (IC) is a measurement of how specific a concept is, that means how much information that is possible to extract from a given word. For instance, vehicle is a generic concept if compared towards car. The word "car" will in this example be more specific, thus have more information about the concept it represent opposed to a "vehicle". Mentioned by Resnik [15, 16], it is an approach to determine the conceptual similarity between nodes within a hierarchy. Each node represents a unique concept described by a notion of information and is connected by edges between nodes that represent the given relationship and connection between themselves. If such edge exists, the similarity between nodes are measured by the information they have in common. The given similarity is calculated based on estimating the probability of the occurrence of a specific concept class within a collection of text.

Information content is by information theory defined as

$$\mathsf{IC}(\mathsf{c}) = -\mathsf{log}(\mathsf{P}(\mathsf{c}))$$

That is the probability of encountering an instance of the concept c written as $\mathsf{P}(\mathsf{c})$. It measures the specificity of a concept, as described above. When using IC, words are placed in a hierarchy. In this hierarchy, they are placed

according to their the sense or concept they represent. An example of this is shown in figure 2.1.

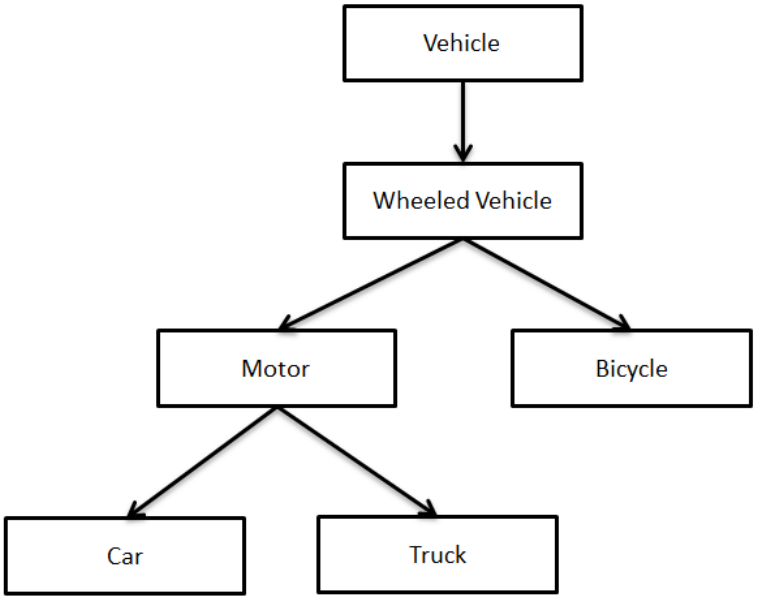


Figure 2.1: A hierarchy of words

For text, a more specific instance of a word is called a hyponym and is located below the initial word in a hierarchy. A less specific instance is called a hypernym and is located above. The higher value of a associated concept means that we are dealing with a more specific concept. An "entity" is typically the least specific term in a text hierarchy and will get the IC score of zero. At the bottom of the hierarchy, related concepts that are the most specific receives the score of 1. For instance, the concept car will have a higher score than the concept vehicle since car is more specific than a vehicle. This example is explained more thoroughly in section 2.2.1 and based on 2.1.

2.1.5 Short Text and Sentence Similarity

While there has been lots of academic work in semantic similarity [15, 17, 18], it is mostly related to term versus term relatedness. According to several researchers, there exists problems regarding the discovery of the essence in a sentence or short text. It is not possible to utilize regular and well-known algorithms such as TF-IDF [19], regular syntactic search and basic semantic search [5] since relationships must be clearly defined such as, within a lexical database.

A large portion of the problem related to this field, is to capture the sense of a sentence. Nicely described by Kiddon and Brue [20] is the double

entendre identification, that is a joke that consists of saying "that's what she said" after someone utters a statement in a non sexual context that also could have been used in a sexual context. Take for instance the sentence:

> *I was taking an essay test when a girl raised her hand and half-jokingly said "Can we finish this orally, my hand is starting to hurt.*[2] *"*

By saying "that's what she said" afterwards, a sexual context is added, thus completing the joke. The core problem is to find the context of the sentence that has a double meaning. Their system, Double Entendre via Noun Transfer (DEviaNT) is basically a classifier trying to locate sentences with two meanings utilizing a support vector machine (SVM). The classifier targets the grammar and structure of a sentence by knowing that the structure: "object" "verb" "subject" is more likely to contain a double meaning than for instance "verb" "object" "subject". An important attribute is that the verb must also have a meaning in an erotic setting.

Other systems performing similar tasks are the Mihalcea's work on "Making computer laugh" [21] and the Cleverbot [22]. While Mihalcea locate humor in text, the Cleverbot[3] is a system responding to human text interaction by text. Both of these work relate to locating the sense of a sentence.

2.2 Ontology

In order to maximize the usability of NLP, we need to have defined an ontology. The term ontology has its origin in philosophy, and has been applied in many different ways. What many ontologies have in common in both computer science and in philosophy is the representation of entities, ideas, and events, along with their properties and relations, according to a system of categories. It was Gruber [23] who coined the term ontology as "a formal, explicit specification of a shared conceptualization" in 1993. Formal is defined as being in accordance with a set of rules or requirements, for instance XML, which is a set of rules for encoding documents in a machine readable form. The shared conceptualization refers to the common understanding of the entities and objects that exist in the area of interest. By explicit specification we mean a clear and detailed description of the relationships that exists between the objects within the conceptualization. An ontology is the formal representation of knowledge as a set of concepts within a given domain. Within the language domain, relations are defined between as relationships between words and the concepts that word represent.

Words with several senses are used differently based on their part-of-speech. In order to capture the "right" sense, we must understand the

[2] http://www.twssstories.com/best
[3] www.cleverbot.com

current context. This also goes for understanding humans, since a statement may have one or more meanings based on the context the second person bases the evaluation on. An example for this is the word "rose". If a person evaluates the statement based on a context of flowers and fauna, the meaning is the flower rose. Else, if the person uses the context of empires in the growth, the sense would be the past tense of "rise".

A domain ontology tries modelling a specific domain, or part of the world. It represents the particular meanings of terms as they apply to that domain. For instance, the word "card" has many different meanings. An ontology about the domain of poker would model the "playing card" meaning of the word, while an ontology about the domain of computer hardware would model the "punched card" and "video card" meanings.

2.2.1 Wordnet

Wordnet is a lexical database that consists of several dictionaries. It is used to find relationships between words and has been employed as a linguistic tool for learning domain ontologies. Wordnet is a general lexical database that is not tied to any specific domain and is perhaps the most widely used ontology for handling text. The main supported language is English, but Arabic and European localization are currently under development.

The purpose of using this kind of ontology, is the distinction between the actual word and the sense of the word. To a computer, a word by itself is simple a set of characters with no specific meaning. For linguistic processing, Wordnet offers the possibility of linking a word to a set of concepts that it represents to humans. For this, Wordnet utilizes the concept of synsets. A synset is a set of synonyms each with somewhat equal meaning. When calculating the relationship between two words, the shortest path between each of the two word's synset is discovered.

Due to different grammatical rules, Wordnet distinguishes between noun, verbs, adverbs and adjectives. Within each of these groups, each word can belong to several lexical groups as described in section 2.1. These could be *hypernyms*, *hyponyms*, *homonyms*, and coordination terms to mention a few. A word could be both a *hypernym* and a *hyponym*, however, to find the right meaning, we must parse and discover the context.

As figure 2.1 describes, Wordnet is built as a hierarchy. We can see that the most generic type of word is placed at the top, while more specific words are linked at a lower level. In the figure, the term "vehicle" is more generic than "wheeled vehicle", while that again have several sub categories. The figure includes both motor and bicycle, it can however contain several more of these categories. "Motor" vehicles do also have several sub categories, in this example "car" and "truck". In order to detect the difference between two terms, it is possible to calculate the distance between them. It is also possible to find the semantic difference between the different terms by first

finding which synonym set each term reside within, before calculating the difference between them.

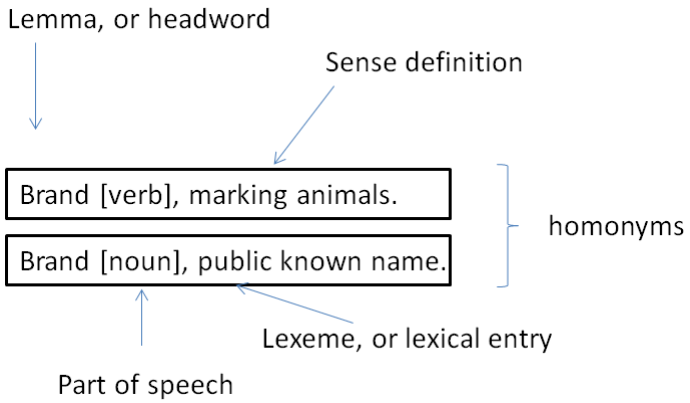


Figure 2.2: Explanation of how a dictionary works

For our example in using a lexical database, we must understand the meaning of a word. To further complicate the matter, a word or term may have several different senses. These are based on the part-of-speech, or the lexical category the word in its writing is in. In figure 2.2 the word "Brand" withdrawn from a lexical book. The lemma of the word, or headword is "brand". This is followed by the part-of-speech which could be the different word categories such as verb, noun, adverb, etc. In our example we have the word "brand" in both verb and noun form. After the part-of-speech, we find the lexeme, or the sense definition. In our case, "brand" is defined as the infinitive of brand in the verb category (brand, branding, brand etc.), while for noun, it is defined as a type of product manufactured by a particular company under a particular name. In a lexical database, the word "brand" in all its part-of-speech forms are defined as homonyms, that is equal words in the hierarchy. By using a lexical database such as Wordnet, all the different senses of this word is listed. More specific words are categorized as hyponyms while more generalized words are hypernyms. An example of these are "Vehicle" as a generalized term while "Bicycle" are a more specific term in the hierarchy as shown in figure 2.1

Similar approaches as Wordnet exists. Example of these are CYC [24], and Sensus [25].

2.3 Recommendation

Recommendation can be defined as a suggestion or a proposal as to the best course of action. In the context of this thesis, course of action signifies the best possible object to be recommended at the time being. By action we mean recommending an item to a user that is the most suitable. In

recommender systems, recommendation is a specific type of information filtering that tries present a user with items that are most likely to be of interest [26, 27].

By filtering we mean the possibility of removing items from a set based on different attributes that the items posses. Recommendation systems in general tries to recommend items in categories such as electronic advertising, movies, audio and books, to mention a few. Such systems are generally built upon extracting information from user profiles.

There are generally two different types of algorithms that are used: neighbourhoods and collaborative filtering. While neighbourhoods calculate relations based upon the distance between several concepts, actors, and their attributes, collaborative filtering utilize the combined view of a defined group as a filter. The k-nearest neighbour algorithm (k-NN) is used for classifying objects based on supervised training [28]. k-NN utilizes instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. k-NN is among the simplest algorithms used in machine learning where an object is only classified if receiving the majority of the neighbour's votes.

Collaborative filtering (CF) is the process of filtering for information or patterns using techniques involving collaboration among multiple sources [29]. Typical applications using collaborative filtering tend to use very large data sets. This could be, for instance, geochemical data used to describe the geological media of interest and resource evaluation describing the quality and quantity of a resource in an excavation.

Collaborative filtering is a method of making automatic predictions based on any interests a user has, by collecting similar information from other users. Such system may have underlying assumptions of equal preferences between group members. Based on group members, these approaches presume that if agreed in the past, members also agree in the future. For instance, based on users preferences and interests, a recommendation system may use CF to predict and propose TV-shows and movies to users. The difference from other recommending algorithms is that information is gathered from many users and proposed to individuals instead of a ranking based on, for instance, a scoring algorithm.

2.4 Ranking

Ranking is defined as the relationship between objects, whereas objects could range from music to television shows. This creates partial order of a set, since for any two numbers, one is either ranked higher than, lower than, or equal to the other. Search engines utilize ranking by relevance in order to retrieve documents that match the user's query. For many commercial systems that perform ranking, the actual algorithm is a closely kept secret.

To rank two or more items against each other, these must be associated with some kind of score. A major problem within ranking is the determination of how to score any given item. For any ranking algorithm, items are usually given an identifier and initial rank score before the algorithms simply generates a ranked list of these items. There exist several ways of scoring items. This is based on some pre-defined criteria.

The best known ranking algorithm today are Hub Authorities [30] and PageRank [31]. Hub Authorities also known as the Hyperlink-Induced Topic Search where a hub represented a page that linked to many other sites and an authority represented a page that was linked to by many hubs. PageRank is known for its link analysis, weighting links within a set of Internet documents. When performing ranking, several parameters count towards the final result as giving the best, most relevant, up-to-date and comprehensiveness.

Other known ranking algorithms are for instance the reddit social new ranking [32] which utilize *hot ranking*. It is a logarithmic scale that weights votes based on their $\log(t)$, that is the logarithmic scoring algorithm based on the initial timestamp. The score will not decrease over time, however, newer items are ranked higher based on a higher initial score.

2.5 Similarity Measurement

Similarity measurement is the concept of finding the equality between two or more objects. These objects could range from word syntax to the concepts a word may represent to the difference in length and grading. Similarity is the probability that these objects are equal. There exist several ways of finding such probability. We have the two different models, syntactic probability and semantic probability. While syntactic probability only inspects the difference in syntax, semantic probability tries understanding the concept behind a term or word [15]. Semantic similarity often requires extra knowledge in order to understand concepts and NLP processing is often used. Such extra knowledge could be lexical databases where we find defined relationships between words. These are often presented in a hierarchy.

2.5.1 Syntactic Similarity Algorithms

Most of the syntactic similarity algorithms are based on the "edit distance" theory [4]. Edit distance algorithms transform one string from a two strings set and counts the number of operations needed to fully transform them to equal. These algorithms are mostly used in spell checking, where their application area spans from spam filtering and medical use.

[4] http://nlp.stanford.edu/IR-book/html/htmledition/edit-distance-1.html

The Levenshtein distance [33] is a metric for measuring the difference in two sequences. The edit distance is the difference between two words, that is, the number of edit operation that must be completed before the two words are equal. Edit operations defined are: insertion, deletion and substitution of single characters for the Levenshtein algorithm. The Levenshtein distance between "kitten" and "sitting" is three, as three edit operation must be preformed to make the transformation:

1. kitten - sitting
2. **s**itten - sitting
3. sitt**i**n - sitting
4. sittin**g** - sitting

Damerau-Levenshtein distance [34] is a more advanced form of edit distance. The algorithm also calculates the minimum number of operations needed to transform a string into the other. In this algorithm, the transport of two adjacent characters is also considered an edit operation.

Take for example the edit distance between CA and ABC. The Damerau-Levenshtein distance between CA and ABC is 2. This is because of the edit operations going from CA to **AC** to A**B**C, but the optimal string alignment (OSA) distance of CA to ABC = 3.

The JaroWinkler distance [35] is a measure of the similarity between two strings and mainly used in the area of duplicate detection. The score produced by the Jaro-Winkler distance is higher the more similar two strings are. The distance metric used is designed and best suited for short strings such as person names. The score is normalized such that 0 equals to no similarity and 1 is an exact match. The Jaro distance dj of two given strings s1 and s2 is:

$$D_j = \frac{1}{3}(\frac{m}{s1} + \frac{m}{s2} + \frac{m-t}{m})$$

m is the number of matching characters while t is the number of transpositions.

2.5.2 Semantic Similarity Algorithms

Several other similarity measures are provided for use with a lexical database such as Wordnet: Leacock-Chodorow, Wu-Palmer, Resnik, Jiang-Conrath, and Lin. We tend to divide the algorithms into those how utilize a corpus and those that do not. Jiang-Conrath [36], Resnik [15] and Lin [37] all use algorithms that access a corpus while Wu-Palmer [38], Leacock-Chodorow [39] and the regular PATH algorithm all use a variant of edge counting. The PATH algorithm is one of the simplest edge counting algorithms used to determine a distance within a hierarchy. For semantic analysis, all these

algorithms access Wordnet to discover relationships between words. These algorithms based their similarity measurement on locating the least common subsumer (LCS). That is by definition the common ancestor deepest in the taxonomy, not closest to the two senses. Where multiple candidates for the LCS exist, that whose shortest path to the root node is the longest will be selected. Where the LCS has multiple paths to the root, the longer path is used for the purposes of the calculation.

PATH returns a score denoting how similar two word senses are, based on the shortest path that connects the senses in the is-a (hypernym/hyponym) taxonomy. The score is in the range 0 to 1, except in those cases where a path cannot be located (will only be true for verbs as there are many distinct verb taxonomies), in which case -1 is returned. A score of 1 represents identity i.e. comparing a sense with itself. Leacock-Chodorow Similarity returns a score denoting how similar two word senses are, based on the shortest path that connects the senses (as above) and the maximum depth of the taxonomy in which the senses occur. The relationship is given as: $-\log(\frac{p}{2d})$ where p is the shortest path length and d the taxonomy depth. Wu-Palmer Similarity returns a score denoting how similar two word senses are, based on the depth of the two senses in the taxonomy and that of their Least Common Subsumer (most specific ancestor node).

Resnik Similarity returns a score denoting how similar two word senses are, based on the Information Content of the Least Common Subsumer (most specific ancestor node). Note that for any similarity measure that uses information content, the result is dependent on the corpus used to generate the information content and the specifics of how the information content was created. Jiang-Conrath Similarity returns a score denoting how similar two word senses are, based on the Information Content of the Least Common Subsumer (most specific ancestor node) and that of the two input Synsets. The relationship is given by the equation

$$\frac{1}{IC(s1) + IC(s2) - 2 \times IC(LCS)}$$

$IC(s1)$ and $IC(s2)$ is the depth and path for the given synonym set in the information content. Lin Similarity returns a score denoting how similar two word senses are, based on the Information Content of the Least Common Subsumer (most specific ancestor node) and that of the two input Synsets. The relationship is given by the equation

$$\frac{2 \times IC(LCS)}{IC(s1) + IC(s2)}$$

Here, the algorithm uses the least common subsumer located in the information content that is built from a corpus. Then, the depth of the synsets of the senses are located in the information content.

2.6 Summary

In this chapter we have described the required background needed as a base for this thesis. While NLP is used heavily today, it is both a time consuming and error prone field of research where algorithms exchange computational time for accuracy. This includes the different classifiers used in NLP such as POS tagging and WSD. We have elaborated some of the existing problem areas such as discovering of the context and the current state of art. Also, we have given an introduction to similarity measurements, both syntactic and semantic algorithms and approaches. We have chosen to use the lexical database Wordnet in our approach. It is the standard lexical database used in most academic work within the field of NLP [40, 41].

Chapter 3

Design

The main contribution of this thesis is the algorithms and how they are used for similarity comparison. To be able to use such algorithms, we must first agree on a system model for our prototype. This chapter will discuss several approaches before we present the architecture of this prototype with all the needed components.

3.1 System Model and Architecture

Our event detection system will in general generate a similarity score of how similar two concepts are. Our semantic text analyser is defined as the system. Based on the semantic analysis, the system will compute the relatedness between the input.

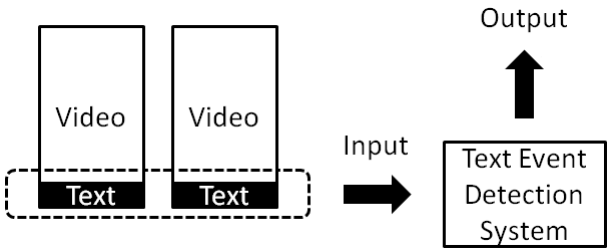


Figure 3.1: High level system model

The main overview is shown in figure 3.1. From the system model, we can see that the system receives input, and produces output. The input is defined as text parsed from video files.

3.1.1 Components

From the high level model displayed in section 3.1, the system is defined as a text event detector. Our event detection system generates a similarity

score based on how related two concepts are. From the input received in the system, it is possible to compute the similarity of, for instance, two words. Input accepted by the system may be parsed and corrected based on certain rules within the system, to avoid errors when further parsing the input in the system. From the system model, we derive a component model as shown in figure 3.2

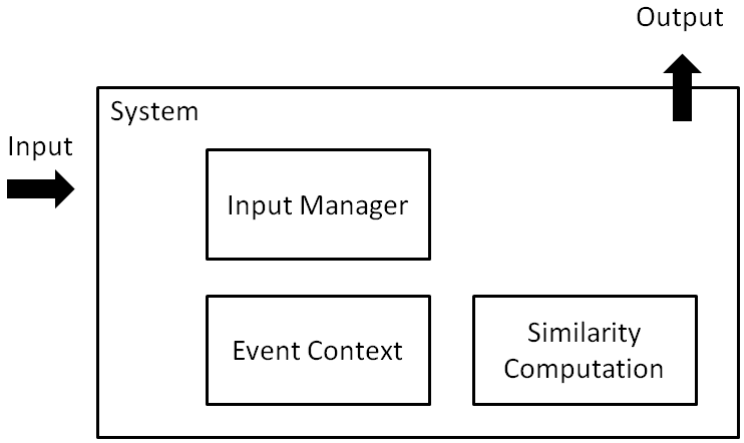


Figure 3.2: System model containing both a similarity computation module, an event context module and the input manager.

To produce the required output, the system must have a set of modules that are able to handle and process data from the moment it enters our system to the moment the similarity score is delivered. The modules described here are the input manager, the event context module and a similarity computation module.

To avoid errors when further parsing the input in the system, the input must be validated according to a set of rules within the system. For instance, if the system utilizes a lexical database for word lookup, all input arriving at that point should not generate any exceptions at that point. Then, certain words should be filtered out at an early stage. Also, when accepting input from different sources, there may be formatting issues such as different input file formatting or string encoding.

The similarity computation unit can understand the concept that a word represent, thus, it calculates how related two different concepts are. The proposed system must be able to generate such a context in order to evaluate the input correctly. An event is the basis for such a context, as such, we define this as an event context.

We define events as specific phenomenons located at a specific point in time. The length of an event can span across several scenes in a movie, however, for our system, the length is defined as the period of uninterrupted sequence of the same event. They take place on a timeline corresponding

with the actual feed they are representing. For instance, a video feed will consist of several events located at different points in the feed's timeline. The events will typically be a short summary of the complete feed, as of: (1) at point t1, "a man got shot", (2) at point t2 "some were flying an aircraft", (3) at point t3 "there was an accident" etc.

A word, sentence or text is always evaluated based on the context of the person that is evaluating the text. The proposed system must be able to generate such a context in order to evaluate the input correctly. We propose to generate event contexts, such that it is possible to locate events when evaluating the input.

3.1.2 Control Flow

As a high level control flow, the system receives input data and computes a similarity between the two items. However, in order to compute this relatedness, the correct and formatted data will be parsed and an event context must be generated.

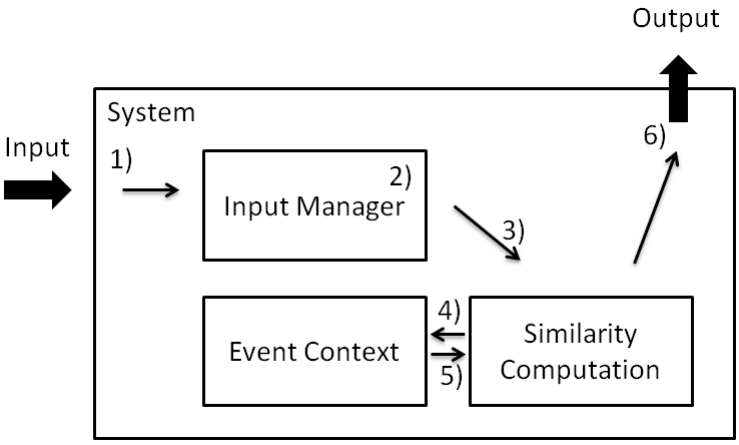


Figure 3.3: High level control flow

Figure 3.3 summarizes the main control flow in our system. To support the calculation of semantic relatedness between input, we need a component that is able to generate a context for a given an event. This includes calculating the relationship within such a context so that there is a defined and clear relatedness between all concepts within the context. In other words, if the original meaning of the base concept is lost when creating the event context, the relatedness between those concepts is below a certain threshold.

The logic of the similarity calculation is contained in a separate component containing its own interface. Hence, it is possible to plug in other algorithms without the need for redesigning the rest of the components based on changes done in the algorithm. Also, additional algorithms can be used

in sequence in order to skew the output result based on the diversity of the algorithms used. This could, for instance, be how the similarity is calculated.

All these components can be considered as separate services that are usable internally through an internal interface. The complexity in usage of all devised components, can be changed based on the said interface. For instance, the complexity regarding the event context generation can be changed to include or exclude the size of the context.

The basic control flow is based on figure 3.3 and defined as the following in our system:

1. Input arrives in the system.
2. The input is parsed and made ready by the input manager.
3. Similarity calculation starts.
4. An event context is generated.
5. Similarity computation is finalized based on the event context.
6. Output is sent from the system.

3.2 System Components

We have defined an event detection system for discovering semantic relatedness between text by detecting events within text. Events are detected based on the discovery of similarity against a generated context. The event detection system is composed of several modules shown in figure 3.4. The main modules are as first shown in figure 3.2, the *Input Manager*, *Similarity Computation* and the *Event Context*.

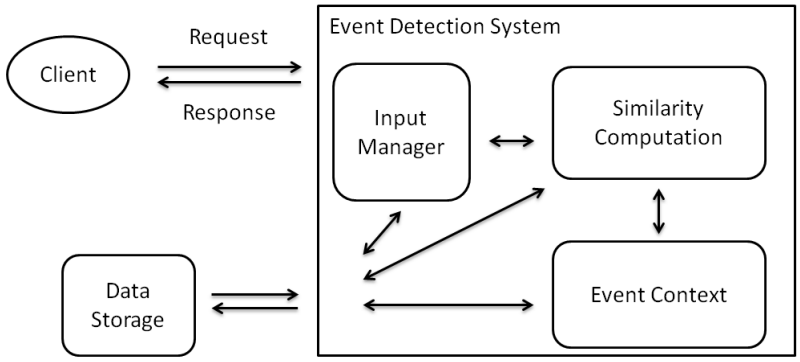


Figure 3.4: Overall event detection architecture

- *Input Manager*: The input manager is responsible for generating valid terms for further use in the processing pipeline. A valid term is defined as a unique term that does not generate any exceptions later in

the system. An exception may be that it is not found in the lexical database, or the term is misspelled. Common tasks for the input manager consists of operations like stemming, lemmatization and stopword removal.

- *Event Context*: Based on simple keywords, this component will generate an event context containing a set of words that relates to the event that is searched for. By using a set of terms associated with the event, the system can, based on certain parameters, expand the set by so-called context exploration in order to increase the context size. Context exploration generates the context based on circular crawling of equal senses with almost similar meaning. This means searching for similar *hypernyms*, *homonyms* and *hyponyms*. Expanding the context size can lead to better chances of discovering the event.

- *Similarity Computation*: This component is responsible for similarity computation. During run-time, the component is able to execute a series of algorithms in order to compute similarity between text and the context representing an event. Computing similarity for a sentence could prove difficult based on several factors, as for instance capturing the essence of a sentence, or even locating the most prominent word. By computing similarity within a sentence, we should be able to capture the essence. Also, by computing the similarity between consecutive sentences, the system should be able to locate the meaning of a series of sentences.

The general program flow is shown in figure 3.4. Typically, an application will ask for a similarity check between input, that is, the relatedness between some text and an event. A context is generated based on the event keyword that could either be received as input, or located in a database, described later in this chapter. The event detection system will utilize a lexical database for semantic inquiry. Also, similarity within and between sentences are computed to generate the best possible result. The computed result is the delivered back to the application.

3.3 Input Manager

The input manager is a tool to prepare the input data for the pipeline processing that awaits later in the system. The system accepts most kind of text, as for instance, single words and short text. For the design and implementation, we will focus on subtitles as input, however, regular words and short text will also work as valid input.

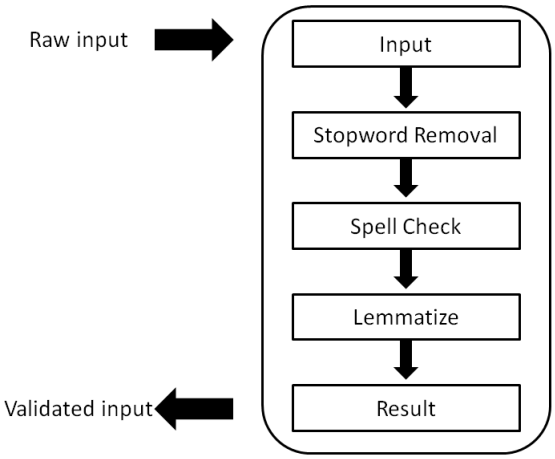


Figure 3.5: Input manager architecture and design

As mentioned earlier, the input manager has the job of preparing the input for further processing in the system. The program flow within the input manager is based on the overall control flow as illustrated in figure 3.3 and more specific in 3.5.

1. Tokenize a short text into terms
2. Remove stopwords and terms without any prominent sense
3. Check for spelling errors
4. Lemmatize all terms

3.3.1 Stopword Removal

Stopwords are in computer science defined as a set of words that is very common and not required for linguistics analysis. Also, they appear frequent and have little or no meaning, thus can safely be removed. Results based on datasets that are not processed for stopwords can make results inaccurate due to words with no or little meaning are being ranked as regular words. Examples of such words are ”thus”, ”then” etc. Stopwords as ”I”, ”a”, ”some” and ”just” are words that when parsed into tokens, make little sense alone. There exist several lists of stopwords on the Internet, some contain up to 10 000 word. When searching a database that have removed the 10 000 most common English words, the execution time of the query and accuracy of the result is far better than without[1]. Some of these words do actually create a structure within the sentence, and could be kept in a shadow sentence for later semantic processing.

[1] http://blog.stackoverflow.com/2008/12/podcast-32/

Stopwords are removed during the linguistic processing of the data. Since we are expecting heavy processing during the semantic analysis, it is better to remove stopwords at the very first in the processing line.

3.3.2 Stemming and Lemmatization

This component is responsible for locating the root of a given word. Stemming is used within linguistics for the purpose of reducing a word to its root. Take for instance the word "jumping". Its root is "jump" and it has many forms and endings when used in speech, as for instance "jumped". For better comparison between words, all words should have the same form and tense. The first stemmer published was by Julie Beth Lovins in 1968 [42], though published early and regarded as important for the linguistic field, the Porter Stemmer is now among the most used and best known stemmers. Published by Martin Porter [43] in 1980 its algorithm is ported to most programming languages today. In his paper, the word connect has been written with several endings, but still contains the same meaning as a base. "Connect", "connected", "connecting", "connection" and "connections" still use connect as its root and all these words should be treated as "connect". It is however not possible for a stemmer to find relationships like "good" and "better".

Lemmatization is a more advanced form of stemming, where you allow for these links to be discovered. While stemmers use grammatical rules for word reduction, lemmatizers allows for dictionary lookups in lexical databases as Wordnet. This means that lemmatizers can understand the context and therefore find the part-of-speech used with the lemma.

Words like "jump", "jumping" and "jumped" have all the same lemma which is "jump". By removing the suffix in "jumping" and "jumped", all these words will have the same base and therefore can be compared to each other. However, if we decide not to include any information of the different terms, they might actually semantically mean different things, as one term could be in the present state, while the other, indicated by the ending, happened several years ago.

3.4 Event Context

The event context component generates a context for a specific event. It is by this context, sentences are evaluated against based on word patterns. For a given scene within any media, there exist a context for it. In order to evaluate any discovery, we need to evaluate towards the correct context. We define an event context as a pattern of words that are representing the event. The context is "expertly" chosen terms special for the event we are computing against. For soccer, it would be for instance "keeper", "forward", "back" etc.

We think of using a keyword based approach to generate the context that ultimately will be the base for the similarity evaluation. Several non-lexical algorithms utilize a corpus and the generated information content in order to generate a probability for a given term within a sentence. If we are able to construct an event context consisting of similar words as the keyword, we may be able to generate higher precision in our event detector, based on a larger and more specialized comparable context.

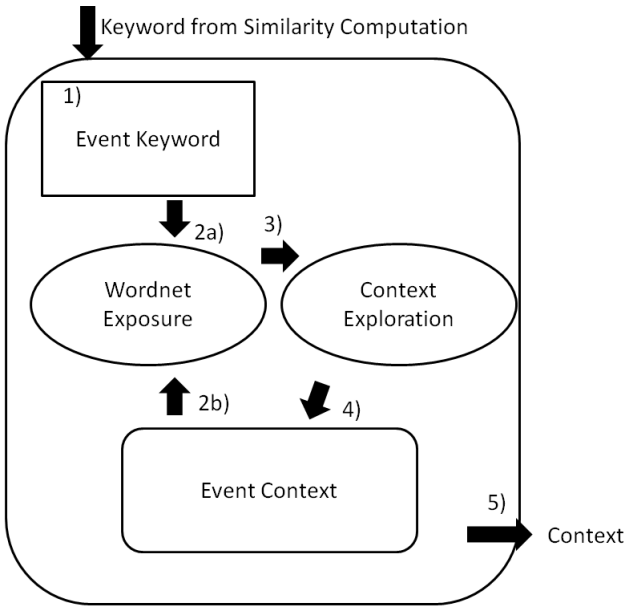


Figure 3.6: Architecture and design of the event context generator

Based on the overall control flow in figure 3.3 and 3.6, a specific control flow has been created.

1. A base context is established
2. The keyword is exposed to wordnet
3. A context exploration is performed
4. Based on certain metrics and criteria, an event context is created
5. The context is completed and sent to the similarity computation module.

Through the use of Wordnet, each word can be associated with a synonym set. This is a collection of words where there are defined relationships between them. We can, by performing a term expansion process, hopefully increase the possibility of finding terms that generate a probability hit when

searching for events. By a context expansion, we try locating all synonyms that are relevant for the actual unique term. Also all synonym sets that are in relation to our word's synonym set should also be evaluated in order to generate the best matching list for our search. Li et al. [44] also proposes such a solution by scaling the similarity based on depth in the taxonomy.

3.4.1 Generating Context

For a given scene within any media, there exists a context within it. In order to evaluate any results, we need to evaluate the results to the right context. It is possible to generate a context based on keywords as proved by [6], building a bag-of-words based on similarity towards some keyword. Typically, we can perform a context expansion utilizing Wordnet as a lexical database and generate larger set of words within the context. The set of terms can have several properties as of a maximum distance from a term in the Wordnet hierarchy. Also, this generation allows for several metrics, calculating for instance, the distance and edge-weighting using the lexical hierarchy. A proper solution can utilize edge counting and weighting as base probability.

For each event, some keywords regarding this event must be defined in advance. For instance, all relations to the word "kill" is located through the lexical database. The list contain both *hypernyms* and *hyponyms*. While *hypernyms* are words of a less specific nature like "termination", *hyponyms* are more specific instances of the word. For the case of "kill", more specific words are "neutralize", "dispatch", "lynch" etc. Based on the total number of senses available after this exposure, we have the ability to repeat the step. An important part of the process is to discover the part-of-speech of the keywords. If a noun is discovered, this must be taken into account when exposing the keyword to Wordnet as the lexical database is POS sensitive for difference between noun and verbs, thus must be taken into account.

The generated context consists of several connected senses, defined via part-of-speech tagging and word sense disambiguation. For instance, a word could have several meanings based on the POS. The word "kill" would have some defined senses if it is a verb. A "kill" would be the termination of somebody, the actual action while the noun POS would be the event.

Unique words are sent through a pipeline process where the lemma is first found, then both POS tagging and WSD are used to find the word type and its sense. The reason we use both POS tagging and WSD is that though we can find the part-of-speech, the part-of-speech may consist of several senses that are only found by a WSD classifier. It is the WSD that based on the text or sentence tries finding the right sense that would be appropriate in the given context.

3.5 Similarity Computation

It is within the similarity computation component that all the similarity is computed. Semantic similarity is finding the probability that the concept the term is representing is actually similar to the comparing concepts. The internal design of the similarity computation component is illustrated in figure 3.7.

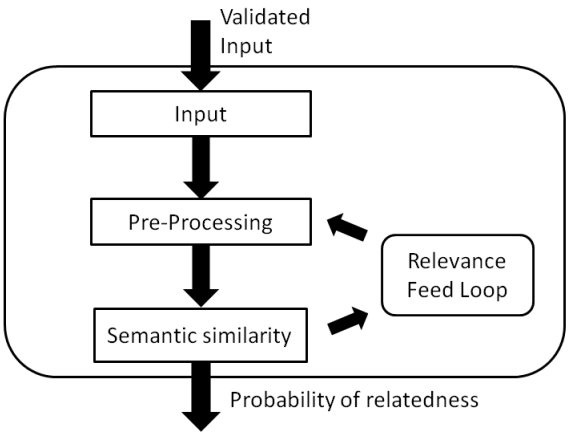


Figure 3.7: Architecture and design of the semantic similarity computation module

We find three different calculation units within. They are: (1) similarity calculation towards an event, (2) similarity calculation within a sentence and (3) similarity calculation in consecutive sentences. Based on figure 3.7, we have designed the flow of control within the similarity computation unit:

1. Receive list of tokens for further processing
2. Pre-process for similarity comparison that typically is Wordnet access
3. Generate similarity probability

3.5.1 Determine Semantic Links

By locating possible semantic relationships between a selection of words, it might be easier to discover the true meaning and content of a given sentence. A way of exploring the semantic relationships between words is to use an ontology. Since verbs and nouns are each set up in a hierarchy we can determine the distance between senses, thus estimate how two words are related. A computation of semantic relatedness can only give an indication of the similarity between two words and their meaning. Jiang and Conrath [36] describes an approach for discovering semantic similarity by combining

lexical taxonomy and corpus statistics. By using a taxonomy for defining relationships, they define a topology which bases similarity on edge counting.

A single word can have several different meanings or senses, and for each word, a calculating process will have to determine the similarity of a large number of words and their meanings. By using part-of-speech (POS) tagging and word sense disambiguation (WSD), it is possible to compute the similarity towards the given word in its correct sense. As there are a notable computational time there can be an issue when computing similarities against many sentences, therefore, we have created database support for the similarity calculation, such that context are calculated for each sentence when we import them into the system. This allows for both a full computation on demand as when a client asks for similarity and database retrieval minimizing the computation needed before returning a result.

3.5.2 Similarity Within A Sentence

To discover the context of a sentence, the system need to understand the meaning of each word located within that sentence. We will look at possibilities to relate different senses within a sentence toward each other. By performing such a similarity search, we hope to locate and discover the meaning and pattern between a set of words in a sequence. As described in section 2.1, each word has a sense related to it. The Wordnet lexical database have constructed relationships between all words in the hierarchy. This allows for using synonym sets. By using the lexical database, we are able to compute the similarity between synonym sets.

There exist several algorithms for discovering relationships between words, each emphasizing on different attributes. The regular PATH described in section 4.4, is based on edge counting within a hierarchy of words. This algorithm exists in many different variants though most are based on counting edges between terms. Leacock-Chodorow is another algorithm that also utilizes edge counting. It does however, emphasize more on the taxonomy that is used. For instance, the relationship between two senses are based on the negative logarithmic scale of the shortest path divided on the taxonomy depth. Wu-Palmer uses the least common subsumer of the two senses in the taxonomy, that is the lowest most common term within the hierarchy. The main difference between these algorithms the use of the least common subsumer. While Leacock-Chodorow computes the maximum depth of the taxonomy where the sense occurs, Wu-Palmer calculates the similarity based on the most specific ancestor in the hierarchy. Based on the difference, we related the most the the Wu-Palmer version of edge counting, based on the least common subsumer between two senses.

3.5.3 Similarity in Consecutive Sentences

In order to evaluate a sentence towards an event context, the context must be correct. That is for instance, a context generated based on the correct sense of a word. Detecting the main context of a sentence can be difficult depending on, for instance, the length of the sentence. Based on the length of a sentence, the context may change based on the known words. For instance, if we encounter a scene boundary in a movie or TV-show, the content often changes or have a different meaning in another scene. Else, a new context can be created and new relationships may be found. By creating separate contexts for each scene boundary, we avoid floating boarders, where possible events may not be recognized due to floating contexts.

As our stab at increasing the chances of generating the correct context, we look at locating similarities between consecutive sentences. If similar words or concepts are present in for instance two to three sentences, that word may contain a prominent meaning across several sentence, thus is a base for the event context.

Chapter 4

Implementation

This chapter describes the design and implementation of our system. The chapter is organized in the following way. First, section 4.1 details the tools and environment used for our system. Then, section 4.2 describes the pipeline used for preparing each sentence for further processing. Context generation and similarity computation are described in section 4.3 and 4.4. The storage unit is elaborated in section 4.5 while the different applications are described under section 4.6. Finally, we present a summary of this chapter in section 4.7

4.1 Technologies Used

For this thesis the following software development platform was chosen.

- Windows 7
- Silverlight
- Python
- NLTK
- Wordnet

As we developed the application in Silverlight, the Windows 7 operating system was chosen for this purpose. Silverlight is an application framework from Microsoft for writing and running browser plugins and desktop applications. Unix systems were also used for developing the actual event detection system in Python. The language was chosen for easy data structures and the use of the framework for accessing the Wordnet application for lexical database access. The accessing framework for Wordnet was NLTK which is an academic project [45] and is currently used as a de facto standard for this type of framework application. Several Python modules where used

and include for instance, the srtParser and other standard libraries. The srtParser was used to parse subtitles by creating each line as a accessible object with properties like start-time, end-time and text. Numpy is also used for mathematical computations.

4.2 Input Manager

We based the implementation of the input manager on the design elaborated in section 3.3. The basic implementation is shown in figure 4.1. First, the system receives text as input from, for instance, an application. The input is then (1) tokenized into a list for easier processing. Then, for each token within the list, (2) stopwords are removed before (3) spell checking is performed. During the last two phases, the token may be discarded based on the system's findings. Afterwards, the system (4) perform lemmatization of the given token, and is stored in a list. When all tokens are processed, the validated input are sent to the similarity computation unit as described in figure 3.3.

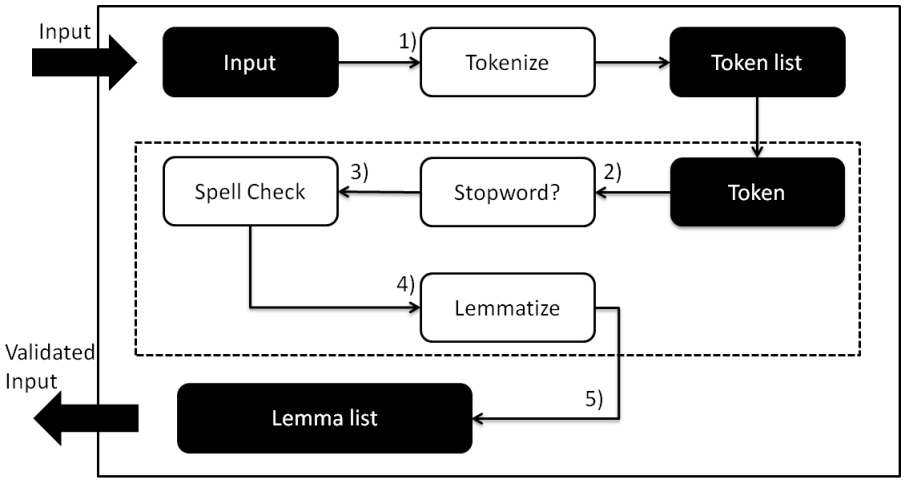


Figure 4.1: Implementation of the input manager

4.2.1 Stopword Removal

We have chosen to use a stopword list containing 126 words as provided by the NLTK package which includes the most known stopwords. Also, we filter out all words with length less that 3 characters. These are the most common words used in English today and when removed, only meaningful words make out the sentence.

Each word within the sentence is compared against the list and removed if found equal. Also, most words that have the length of two characters or

less have little on no value and therefore removed. By implementing a too large stopword list, the result will be that almost all words are filtered out before the actual processing begins. However, if a too small list is chosen, the result would be that too much of the sentence is left when processing for semantic similarity. Therefore, we have chosen a smaller subset of an English brown-stopword list.

As point 2 in figure 4.1 illustrates, tokens arrive at the stopword removal process. If the token exists in the stopword list, the token is disposed of. However, if not found, the token is then passed on in the pipeline to step 3.

4.2.2 Syntax and Spell Checker

All edit distance algorithms performs well, though some are more complicated than others. We have chosen to implement a simple version of the Levenshtein algorithm [33]. The implementation is straight forward. A term arrives at the process and is checked against a dictionary provided in Wordnet. If it exists, the word is passed on for further processing. If not, we locate the k-nearest words and execute the algorithm. The lowest score by the algorithm is chosen as the word and passed on. The basic algorithm is as below.

Listing 4.1: Levenshtein Implementation

```
1 def Levenshtein(base, test):
2         distance = 0
3         for char1 in enumerate(base):
4                 for char2 in enumerate(test):
5                         if char1 == char2:
6                                 continue
7                         else:
8                                 distance += 1
```

Testing the difference between two words is in this example as easy as comparing each character for equality.

4.2.3 Stemming and Lemmatization

Based on the discussion is section 3.3.2, we have chosen to use a lemmatizer in order to discover the root of each word. This allows our system to discover relationships that a stemmer cannot detect, such as locating the lemma of the word better is the word good. The general form of lemmatizing a word is to apply grammatical rules until the word it is not possible to preform such actions or it is located in a dictionary. This could for instance be, checking the suffix for known endings such as -ion, -ing or -sses.

The generic lemmatization process usually involves the following steps:

1. Select word to lemmatize: lets say "meeting".
2. Grammatical rules to find the lemma or root word which is meet.
3. Analyse the context, using a word sense disambiguation.
4. Based on the context, meet could be: (a) noun - sports meeting (b) noun - gathering (c) noun - encounter (d) verb - meeting someone (e) verb - satisfy requirements, to mention some of the senses found.

Based on the analysis of the context, performed by WSD and POS, the lemmatizer will make an informative guess of the sense. We use the lexical database of Wordnet to find possible senses. As the word is now lemmatized, it is also possible to further process the words by extracting the actual sense or concept the word is representing.

4.3 Event Context

The implementation of the event context is based upon the design from section 3.4. Context representing the event is generated based on two algorithms that will be presented below.

4.3.1 Generating Context

Each comparable context could be generated differently based on attributes set by a user or temporary results within the generation process. The context should be between a minimum and a maximum number of sense. If too many senses are present, it is highly possible that the number of positive results may drop in percentage due to the number of false positives rising. This is because we can relate incoming sentences with too many and widespread senses. If too few senses are present, we may not be able to generate scores above given threshold and therefore not be able to detect events.

In section 3.4, we have defined the fanout of senses as a context exploration. Pseudo code for the fanout is given below.

Listing 4.2: Context Exploration (Fanout)

```
1 def Fanout(keyword):
2         senseList.append(keyword.getSense())
3         senseList.append(sense.getHypernyms())
4         senseList.append(sense.getHyponyms())
```

A problem related to the context exploration is that words can loose the original meaning of the keyword. As this may not be a problem for *hyponyms* where words are more specialized instance of the high-level concept. However, as *hypernyms* are less specific than the current term, they may lead to unrelated terms in the sense of a concept looses its specific attribute. An example would be the noun "kill". Its *hypernym* is "termination". While "termination" is a good term that is interchangeable with "kill", the *hypernym* of "termination" is "change of state". The term "change of state" has in this context lost all the meaning of the original word. *Hyponyms* of "termination" includes "destruction", "demoralization" and "liquidation" to mention a few. All these terms are good examples that the term has not lost its original meaning. If too many senses are generated by the context exploration, we simply calculate the relatedness for each new sense to the original sense. To calculate the relatedness between two words, we calculate the difference between their synonym set, as each word belongs in such a set: $S = [s_0, s_1, \ldots, s_{n-1}]$. Based on the score, some senses will be removed as they do not longer posses the same attributes as the original sense. The pseudo code for the removal process is given below and is based on the Path algorithm explained in section 2.5.2.

Listing 4.3: Path Implementation

```
1 def Path(synset1, synset2):
2         # returns score between 0 and 1
3         numEdges = CountEdges(synset1, synset2)
4         if score < 0.85:
5                 disposeOfSynonym(synset2)
```

We use two sets of metrics for calculating a sense's surrounding context. The first metric describes the selection of senses into the context, while the second metric describes the relatedness between similar senses. The first is:

(1)
$$\mathsf{Context} = \mathsf{Fanout}(\mathsf{keyword})$$

$$\mathsf{score} = \mathsf{Path}(\mathsf{s1}, \mathsf{s2})$$

To select senses for context generation, we perform a circular crawling of the hierarchy and the specific tree the sense belongs too. By discovering both hypernyms and hyponyms we are able to generate a context where similar senses are present. Based on the selected senses in to context, each of them are checked for similarity relatedness towards the original sense (keyword).

(2)

$$similarity = \frac{Path(s1, s2)}{\sum_{j=0}^{n} depth(s1, s2)}$$

In the second metric, we use both the PATH similarity and Wu-Palmer similarity, both described in section 2.5.2. By using the PATH algorithm, we calculate the distance between the two senses the is compared. If the score calculated in the first metric is higher than the similarity calculated in the second metric, the sense is thrown away. Based on the distance that is calculated based on edges between them within the Wordnet hierarchy, we know their basic relatedness. This allows us to discover the relatedness based on common senses and concepts in the hierarchy. By comparing the score of each algorithm we have a basis for knowing the actual sense to sense relatedness. The use of these algorithms will be described in the following paragraph.

4.4 Similarity Computation

It is within the similarity computation component that all similarity is computed. The implementation is based upon both the discussion in section 3.5 and figure 3.7. To produce similarity between two concepts, there must exist a relationship between them. These are often defined in lexical databases such as Wordnet. As discussed earlier, we propose locating both the most prominent word within a sentence and the discovery of the base context that consecutive sentences represent.

4.4.1 Similarity Within A Sentence

By computing the similarity between all words within a sentence, we hope to better locate the essence of the sentence in such way that it is easier to compute the similarity between the sentence and an event. For this we use a metric.

The metric computes the similarity based on the depth of each word within the current taxonomy. The depth is calculated based on the depth of the taxonomy used. That is, the distance between the word in question and the root of the taxonomy based on the distance in the Wordnet hierarchy. By computing the fraction of 2 times the depth divided by the sum of the two different depths each word + two times the depth the score will be at most 1. The depth function calculates the offset from the entity regarded as top in the Wordnet hierarchy.

(3)

$$score = \frac{2 \times depth_{max}}{(depth_1 + depth_2 + 2 \times depth_{max})}$$

Our implementation of the similarity within a sentence is based on calculating each word against every other word within the sentence, given that they are not filtered out by the pre-precessing. This includes both stopword removal and lemmatizing as described regarding the input manager. A total score will be generated based on the implementation of the algorithm used. Since a score of one only indicates basic similarity between words, we have chosen to set the threshold higher than one. By encountering such a similarity, we have the ability to later emphasize on the relationship between those words. Again, this may result in an extra enquiry of finding the sense that most relates to the similar words as the possibility of that sense to be the most prominent and thus indicating a possible context for the sentence.

The enquiry is done by a simple edge count. Also, we have modified the counting to also take into account the number of *hyponyms* and *hypernyms* each sense have. If a sense have many *hypernyms*, it means that it is one of many different senses originates from the parent, thus is more specific that senses with less *hypernyms*. This is however not the case with *hyponyms*. As a sense with several *hyponyms* will have many children meaning that it is a important concept, that is often related to and have properties that we may look after. Based on a simple scoring algorithm with attributes like the number of *hypernyms* and *hyponyms*, each sense is graded, thus ranked. The ranked sense list is the based of the concept or context the sentence is containing.

4.4.2 Similarity in Consecutive Sentences

The algorithm for calculating similarity based on the discussion in section 3.5.3. We underlined the problem of not successfully locating the correct context that sentences were evaluated against. For optimal use of similarity calculation towards an event, we need an algorithm that captures the context based on certain criteria.

We have developed a metric that calculates the sum of the path based on the synonym set of the first word divided by the number of paths. It is multiplied by the constant $\mathfrak{a}$ divided on the sum of similarity of the synonym sets $\mathfrak{s}1$ and $\mathfrak{s}2$. The path is calculated to the root and could maximum be 1, making the left most side equal to 1. The constant $\mathfrak{a}$ is calculated and based on the currently found similarity in the consecutive sentences. Similarity between synonyms are calculated based on shortest path between them in the hierarchy by the metric displayed below.

(4)
$$\frac{\mathfrak{path}(\mathfrak{s}1, \mathfrak{s}2)}{\mathfrak{n} \times \mathfrak{syn}(\mathfrak{s}1, \mathfrak{s}2)}$$

The $\mathfrak{path}$ calculates the shortest path between the different synonym sets, while $\mathfrak{syn}$ calculates the similarity of the synsets based on the least

common subsumer as described in section 2.5.2. n is the number of words within the sentences. We have implemented the sentence consecutive similarity by comparing each sentence towards each other. As the score is tied to a specific set of senses or concepts, we simply calculate the score for each sense against the score of each other sense within the compared sentences.

Listing 4.4: Syn Implementation

```
1 def Syn(keyword, synonym):
2         ic = generateInformationContentFromCorpus()
3         keyword_lcs = informationContent(keyword, ic)
4         synonym_lcs = informationContent(synonym, ic)
5         subsumers = keyword_lcs.common_hypernyms(
6         synonym_lcs)
7
8         lcs = subsumers.filter(keyword_lcs,
9          synonym_lcs)
```

If similar senses are discovered, we calculate a simple PATH algorithm, finding the similarity between the two (or more) senses. Based on the algorithm, we should get a score above 1 if there are some similarities between the two sentences, or less if no obvious similarity is found. As elaborated in section 4.4.1, we will perform an enquiry of the most prominent sense within all sentences. This would be the context the consecutive sentences are promoting.

4.4.3 Detecting Events - Similarity Towards a Context

To discover relationships between a sentence and an event, the whole sentence is evaluated against the generated context. The context is found earlier and explained in section 4.3.1. Sentences have been through a pipeline of processing, ranging from regular stopword removal and lemmatizing to similarity calculation within sentences and between them. Now, we evaluate all sentences against the generated context in order to detect possible events.

We have chosen to implement a version of the Jiang-Conrath similarity. The algorithm bases it self on an information content described in section 2.1.4. It returns a score that represents the equality of two senses based on the information content or the most specific ancestor node that both have in common. A maximum score based on this algorithm will be between zero to one. The relationship is given by the metric:

(5)
$$score = \frac{1}{ic(s1) + ic(s2) - 2 \times ic(lcs)}$$

We divide one on the sum of the information content of both synsets minus two times the information content of the least common subsumer. This means that if both senses are equal, one is returned. The IC call generates the probability of the synonym set is present in the information content. The metric returns an error if the two senses are not connected by an ancestor. It is important to remember that though every sense is connected within the Wordnet hierarchy based on the part-of-speech, that they may not be in the information content that is generated from a corpus.

For the duration of subtitles, each lemma that reaches the similarity process are compared towards the contexts. The score generated is the total score of every lemma from a given text string towards the entire context.

Scoring Schema

To make sure that our system does not differentiate between short, medium or long sentences, we have devised a scoring schema. This schema first computes the average score of the similarity for the given sentences as elaborated in section 4.4.3. The scoring algorithm will then take into account the size of the sentence compared, and the context compared against, meaning that though a short sentence calculated against a context will not receive a high total score even if all the words are in some way related to the context. It will, however, have a high average score ranking it on the top list when further processed. In the opposite case, the system encounters a long sentence with only a few words related to the context, though the average score would be high enough for passing the threshold. These two attributes (short and long sentences) have been marked as the most problematic encounters for the system, as either one of them can be falsely produced as an event or not.

4.5 Storage

For storage and data retrieval, we have chosen to implement support for a database. A database support storing of structured and organized data that are logical related within a database model. Data models have evolved and today, we operate with approximately 3 common models: (1) relational model, (2) entity-relationship model and (3) object model. The relation model's primary focus is to store data in tables with defined relationships between them. This is based on first-order predicate logic, meaning it uses quantifiers that range over a given domain. For instance, "for all items in" and "there exist items in" are examples of first-order logic. Whereas the entity-relationship model uses an abstract and conceptual representation of data. That is, modelling relationships and a conceptual schema utilizing a top-down approach. Typical, we find entities, relationships and attributes in these schemas. The object model relates differently to data, as it is represented by objects and classes with attributes. It more powerful that

the other two models, but more complex and is conceptually harder and more complicated to use.

We chose to implement a database using Microsoft SQL server, integrated with Visual Studio 2010. This convenience allows us to easily manipulate both data and the actual database. We created 5 different tables for usage in this thesis. These are *video*, *subtitle*, *term*, *termCollection* and *UniqueTerm*. Keep in mind that the database schema was design and tailored to handle text for our text event detection system, as modelled in section 3.1. to the event detection system, using a recommender system as application.

The database is used for storing subtitles and other text processed for the semantic text event detector. Also, the context of an event is stored in case of often usage, meaning that it context will not be regenerated every time.

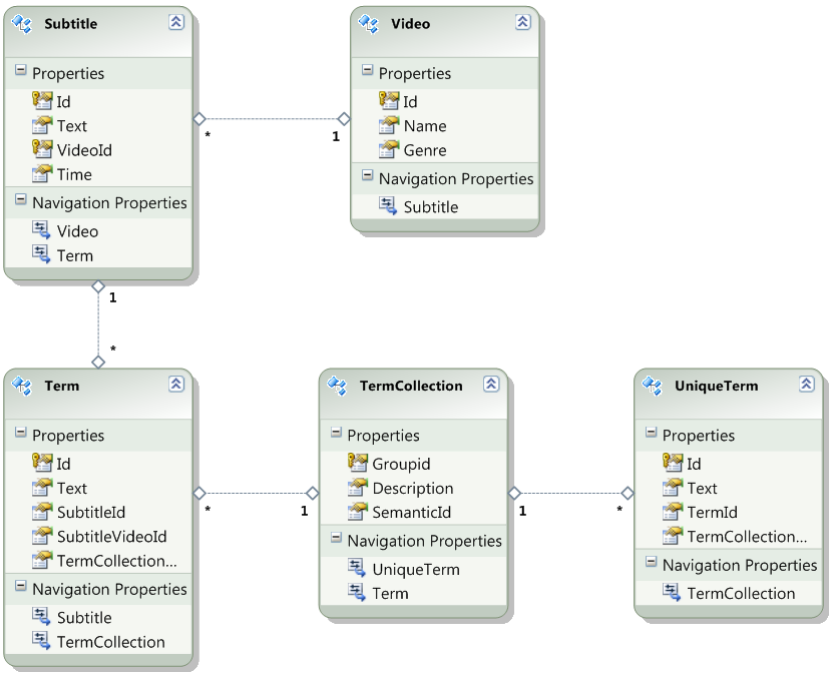


Figure 4.2: Database schema used for storage and extraction of data

The video table contain an unique id, a name and the genre of a given media file. Each video table can access several subtitles. A subtitle table contain the subtitle's id, the given text, a foreign key to access the media file and the display time of the subtitle. The subtitle table can access the term table, whereas all terms are described. Here we find an id, the term itself, a subtitleId, and a foreign key to the termCollection.

4.6 Applications

To test our text event detection system, we have implemented several applications. These include a recommender system that uses ranking based on time, votes, and a similarity score as input and an event discovery application that locates all defined events in text.

4.6.1 Recommender System

Our recommender system provides a couple of features regarding the algorithm used. We have implemented both a collaborative filtering algorithm and one based on scoring events based on similarity. The collaborative filtering is based on the early work regarding the group lens project [29]. As we have implemented a scoring algorithm based on the social news media site reddit.com we use a self implemented semantic text event detector. This allows us to register new events within our database or network and thus finding suitable references to TV-shows based on e.g. time-stamps.

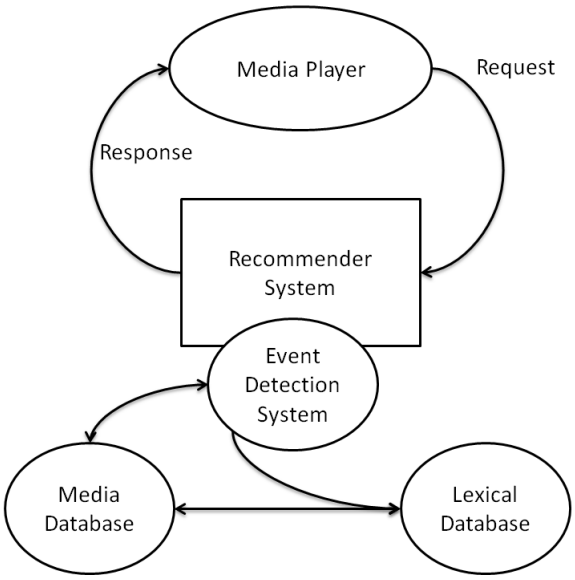


Figure 4.3: Recommender system utilizing the event detection system.

For any recommender system they are evaluated after certain properties, it need to propose the best, most relevant, up-to-date and comprehensiveness for any given user. We predict that by modifying several parameters for the said reddit news algorithm, we are able to provide up-to-date and relevant recommended items to a user. The system operates with a data source feeding relevant text and sentences into the system. Each incoming video feed is first ranked based on the time since last event. This allows us to give a feed an initial ranking based on its freshness. Then, we utilize

semantic relatedness between a given property and all the feeds. The idea is that the system is are able to find several equal events based on the semantic evaluation. A new score is added to the each video feed within the system, that is the probability of the feed to contain the specific event we are evaluating for. Note that all video feeds participating within the recommender system will have their score altered every few interval. This is to ensure freshness and precision in the results generated by the system.

Ranking

We have implemented a version of the reddit news ranking algorithm [32]. This algorithm utilizes attributes such as length of availability, total score, and votes into account when scoring an object. When a new item enters the system, an initial score is given. Later, when the score is updated, these are then altered based on, for instance, the length of which the item has been in the system. Also, the 100 first votes counts more that the 100 next votes scored on a time axis.

Listing 4.5: Hot Ranking

```
1 def Hot(votes_up, votes_down, time):
2         timeDifference = time - epoch
3         tmp_score = log(max(abs(votes_up - votes_down),
4          1), 10)
5         score = tmp_score * timeDifference / 45000
```

The function *Hot* first defines the time period at which the item was first published. Then, a score is created based on the logartimic scale with base 10, based on the absolute value of the votes given. Lastly, the final score is based on the temporary score and the time difference from the time the item was published divided by a constant.

By using this scoring schema, we achieve a ranked list of the different items in the system.

4.6.2 Media Client

We have implemented a media client that is capable of playing a video feed from the local drive. This client will in turn request similarity scores from the event detector which will comply with numbers based on the current action level within several different sources. The media client's use is primarily for the recommender system, as a working prototype. The implementation is done in Silverlight and it uses local media files on disk opposed to streaming. This is because it is only a proof of concept and this thesis focuses on event detection based on text and not streaming solutions.

The media client is a desktop application capable of playing given media files such as video. The application initiates contact with the recommender system and is given a list of video feeds in choose from. Communication is done asynchronously as Silverlight imposes restrictions to the desktop application to always be responsive to the user. When a user initiates a change in the watching schema, the recommender system utilizes the event detector by discovery of similar events sorted by closeness in time. The most appropriate video feed is then returned to the user.

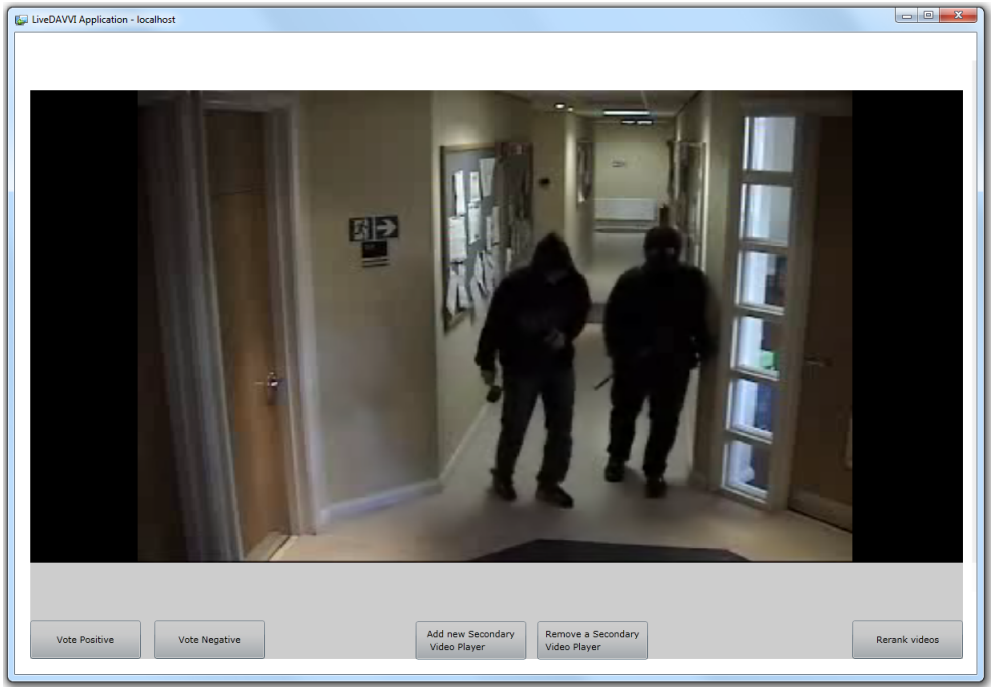


Figure 4.4: A screen shot of the media player.

The recommender system will have the opportunity of altering the result list received by the event detector. By utilizing concepts such as collaborative filtering and news hotness, the list may be rearranged before presented to the user.

4.6.3 Event Discovery

It is well known that two different users or two different commentators can describe the same event using different words. It is hard for a regular search engine to find relationships between two terms describing the same concept by simply using syntactic search. We have proposed a solution to the problem by introducing a lexical database in a semantic search. Here it is possible to draw relations between different terms describing a concept.

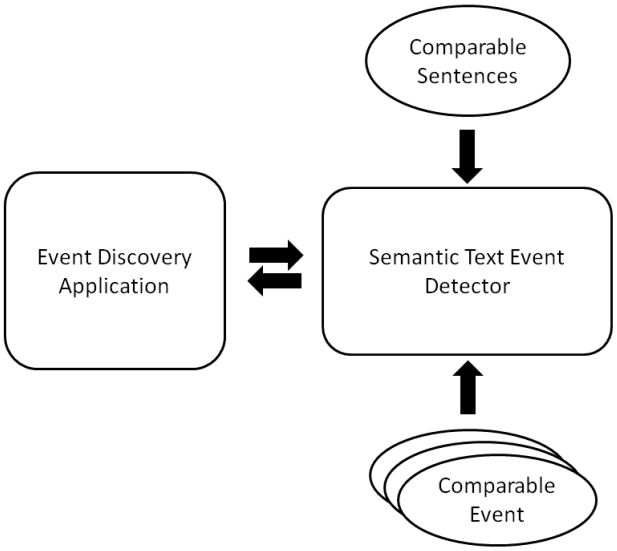


Figure 4.5: Event Discovery Application using our semantic text event detector.

The system allows for systematically event discovery as for finding all events that are pre-defined. Events must, as described earlier be defined through a specific schema. These are then searched for through regular processing. A list of possible detected events are created based on a threshold for the minimum score. This type of application apply for several domains, for instance sports and video summary. For sporting events, commentary can be scraped off a live commentary site and parsed through the system, possibly finding all related events of interest. By analysing a video, it is possible to find events throughout the entire video, generating short video summary of the action.

4.7 Summary

In this chapter, we have presented the implementation of our text event detection system. The system currently runs on Windows and is implemented in Python. We have elaborated the functionality of our three main components, the input manager, the context generation and similarity computational unit.

The input manager generates validated input from given text based on search engine principals, such as removing stopwords and lemmatize words. Then, the context generation module generates an event context that presents the event. We presented the concept of context exploration where the algorithm analyse the input and generate the context systematically by relate each new word to the original words. Lastly, the similarity

computational module have several algorithms implemented that can be used for detecting similarity relatedness between text. Our system offers the ability to compute similarity between several consecutive sentences to better capture the essence of the sentences in addition to compute similarity towards the specific event.

Chapter 5

Experiments

We will during this chapter, evaluate our system and its algorithms based on a set of experiments. First, we describe our experiment setup, along with out test plan, then, the results of the experiments are presented. In light of our context, scope and problem definition, we will evaluate these findings objectively. We propose three different sets of experiments, the first will evaluate the event detection mechanism, while the second set of experiments will evaluate the different algorithms and how they perform based on accuracy. Lastly, the different algorithms and their approaches will be timed.

5.1 Experiment Setup

All our experiments were performed on the same set of servers. We used a server with two Quad-Core Xeon E5430 2.66 GHz processors running on 1.995 GHz with a front side bus running at 1333MHz. There were 2x6MB L2 cache (per processor), and a total of 16GB of main memory. The servers hosting the clients were running Intel Xeon E5540 processors at 2.26GHz. The front side bus was running at 1666MHz with 2x8MB L2caches and 8 GB of main memory. All servers were connected in the same network by a switch. The experiments done are computational heavy as most NLP problems are.

The client side had Python and Silverlight installed. Libraries like *cherrypy*, *routes*, *bottle* and *pySrt* were also used. To avoid unwanted interference on the different servers such as the current load, all experiments took place late at night. This off-peak run also removed the possible unwanted network load by administrative tasks such as backups running at that hour.

Several scripts were created in order to reduce the interference from the researcher running the actual experiments. These scripts handled the start-up of the different experiments. Since the similarity between two terms do not change at all based on static use of lexical databases, experiments

regarding the similarity calculation have only been executed once. However, the timing experiments in section 5.7 and 5.8 were completed several times in order to calculated the average and mean value.

5.1.1 Test Plan

In this chapter, we will conduct a series of tests, analysing our implementation of the prototype. First, we will test the usability and an argument to use a system such as our prototype and the algorithms its using. Secondly, discovering related senses will be evaluated in light of the usage experiment, while the third experiment will analyse the specific event detection system. Then, we will evaluate the context size described in section 4.3, before we look at sentence relatedness, based on finding prominent senses based on consecutive sentences. Lastly, we will evaluate the application domains.

5.2 Initial Proof of Idea

As described in section 1.2, the scope and limitation of the problem solved by this thesis was to create a working prototype of a system capable of detecting events based on semantic analysis of text for use in a recommender system. A prototype that is able to detect events based on subtitles has been created.

We will conduct an experiment to see the potential of our idea regarding the discovery of events based on semantic analysis on text. In this first experiment, we will outline the need for semantic evaluation of words in order to detect events. We conjecture that if all words are transformed to the concepts they represent by semantic analysis, we will see overlapping concepts of words containing similar meaning. When the overlapping concepts are removed, as done in regular systems handling text, such as search engines, the only way of discovering relationships between the remaining words, is by semantic analysis. By locating all words, unique words, concepts and unique concepts, we can discover the usability of such a system we are proposing.

This experiment will analyse the complete subtitle from a television show, discovering the number of words, number of unique words, lemmas, concepts, unique concepts and the number of overlapping concepts within. First we will create a baseline for the experiment by using unaltered text as input, before we do a lemma based filtering as described in section 4.2.3. Lastly, we will do a complete filtering of the input text by for instance removing stopwords and wrongly spelled words before we lemmatize them as described in section 4.2.1 and 4.2.2. The last way of processing subtitle is typically how text processing is done today in order to achieve optimized search through storage.

The graph in figure 5.1 shows three different types of columns. The *original* column creates the baseline for these experiments, while *lemma* and *stopword* are modifications to that. The baseline discovered a total of

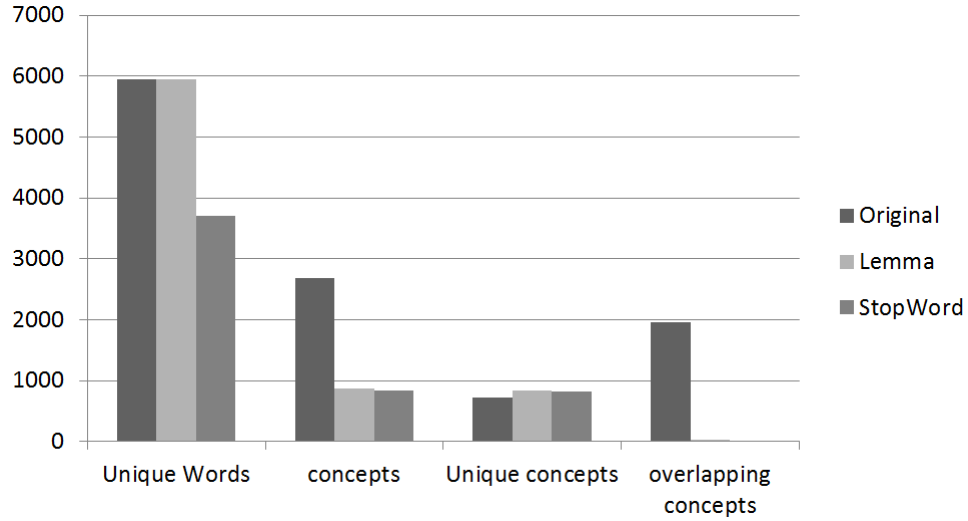


Figure 5.1: Distribution of unique words and concepts

5954 different words. These words mapped down to 2680 different concepts, however, only 716 of these concepts were unique, meaning an overlap of 1964 concepts. By using lemmatization as a filter in the second run, discovered 5954 different words. Of these, only 864 different concepts where extracted and filtered as 841 unique concepts, meaning that only 25 concepts where overlapping. Last, we applied the stopword filter and the spell check filter along side with the lemmatizer. During the experiment, the system found 3710 unique words, translating them into 842 different concepts. From that pool of concepts, only 21 of them were overlapping.

Discussion

The result is somewhat as expected, generally underlining our hypothesis that when processed, concepts that the words represent are not syntactically equal. This shows the potential of our system, since relatedness between the words left in the process, can only be found by semantic analysis of the text. The experiment baseline illustrates that a regular television show has about 6000 words mapping down to 2680 concepts with almost 2000 overlapping concepts. The number of unique words and overlapping concepts is an argument for a solution to semantic analysis text in order to find similar concepts, as our implemented prototype does.

A simple regular syntactic search could not find relatedness between words, thus having access to a lexical database for further semantic processing is needed. By improving the input filter as explained in the implementation of the system, we were able to further filter out unwanted words and finding the core word and lemma. Then, executing calculation based on the

lemma removed most of the overlapping concepts. Last, the graph illustrates the potential of reducing the pool of words by removing stopwords, thus having a smaller subset of words to test. By using regular filtering techniques, we find little overlapping between the different concepts, meaning that regular search will have small chances to locate any relatedness between words.

This experiment has shown the potential of performing semantic analysis of text. By finding the core lemma of a word, it can easily be translated to the concept representing the word by using a lexical database such as Wordnet. Further, by removing stopwords etc. we have managed to remove most of the overlapping concepts originally found, thus meaning that almost 2000 concepts in related and we have the implementation to find their relatedness.

5.3 Detecting Specific Events

In section 3.1, we defined an event as a specific phenomenon located at a specific point in time. To find such events, we analyse text in order to locate specific patterns whereas the system can make an informative guess of what type of event the text represents. This could, for instance, be an action sequence, an emotional scene or more specific events such as a murder or crying.

A sentence is compared against a context, that is, the essence of a given event. This context defines the event as described in section 4.3. The context is made up by a series of related words based on the equality in its sense, that is, words with similar meaning. To capture the essence of an event we do a hyponym and hypernym *fanout* to better find related concepts with the same or a related meaning. The *fanout* is described in section 4.3 and called a context exploration. However, as described in section 4.3.1 each new sense added to the context is checked for relevance in order to remove unrelated such as senses that are related but without the specific meaning the original word contained. This experiment is conducted as a baseline experiment using a context with a zero context exploration, that is, only the event keyword present in the context.

The experiment will evaluate several television shows against a set of events that are defined in advance. Each sentence that is evaluated is sent through our text event detection system defined in sections 4.2, 4.3 and 4.4. The immediate results are written to a file which is later accessed to determine the actual result based on a set of criteria. The criteria defines a result as either positive, uncertain, or false and are determined two computer science graduate students. A sheet of paper containing the complete subtitle that the system was evaluating were handed out to the students. They were then required to mark each processed line from the system as either positive,

uncertain or false. A positive result is defined as a positive likelihood of an event to actual take place during the text evaluated against. If an event is not certain to take place, though the system has flagged the event, the human judges must use their own intuition to determine whether or not this event should be flagged as uncertain or false. If no event can be discovered and matched towards the context by the human judges, the result is counted as a false result.

Based on the similarity computation described in section 4.4, our system computes the total score that a sentence is related towards the event. An average score is computed and checked against a set threshold. This threshold defines how similar each word within the sentence must be, on average towards the event, in order to flag the event as positive. If the score is above the threshold, it is marked as a positive result with the score accompanying the found event for further processing outside the system, as for instance ranking and recommendation.

We calculate the score as percentage of the total number of found events. For instance, if the system discover 10 events whereas 5 are positive, 2 are uncertain and 3 are defined as false, the percentage share would respectably be 50%, 20% and 30% for the positive, uncertain and false rating.

Our conjecture is that it is possible to detect events based on semantic analysis of text. By examining a sentence, the system should be able to locate the most dominant word and its corresponding concept for further processing by the event detection system. However, we do not expect to locate a huge number of events since the context that words are compared against, only contain the actual word and sense that the system is looking for. By including a larger vocabulary of context we think it will be easier to find events.

As seen in figure 5.2 the system is able to discover events by analysing text. It does however, based on the scoring and rating schema only achieve 14.8% positive rating on the discovered events, that is those events also recognized by the human judge. 19.4% of the events discovered got an uncertain rating, meaning that a human judge verified that the sentence compared to the context, contained some words that also were found in the related context. However, we see that the system achieves a false rating of 65.7% of the events found, meaning that the human judges were not able to locate any substantial evidence of relatedness between the sentences and the context.

Discussion

By examining the results more closely, we look at the text behind the results in order to get a better understanding of how the system actually evaluate a sentence. We can see from the result files that *"You're the only person on this planet i'm totally honest with."* gets a similarity score of 60.2% towards

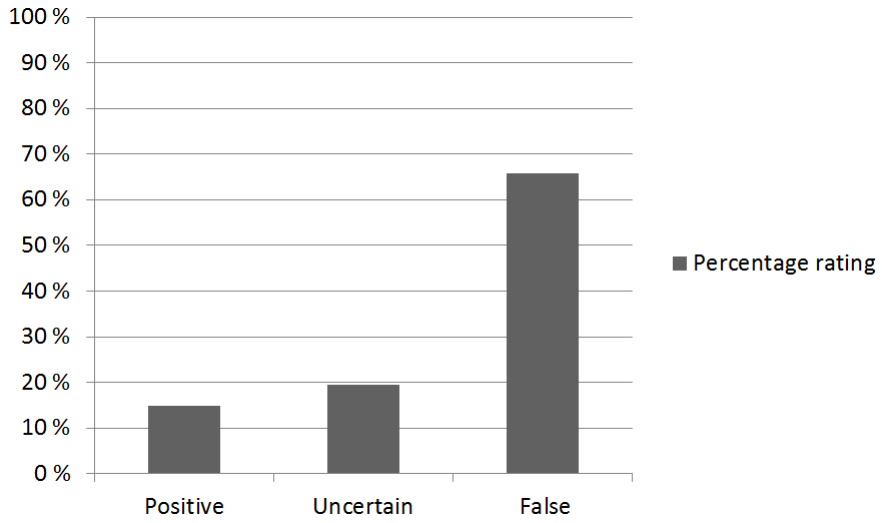


Figure 5.2: Accuracy of a simple event detection

the event *emotion*. For a human, this makes sense in the order of that event would have a high possibility of happening. Consider the two following sentences: *"In czech republic, too, we love pork."* and *"Now, listen, artie's business..."*. The first sentence received a score of 20.65% while it does not have any substantial meaning towards the event, however, it contains a word that belong in the search pattern, namely *"love"*. As we can clearly see, the text event detector has mistakenly thought that the word *"love"* belonged to the event pattern and marked it as an event. This is the case of an uncertain rating. The second sentence has no useful meaning towards the event *(emotion)* and clearly is a false match from the event detection system. However, if we closely examine the similarity computed between the second sentence and the event, we can see how the event detector could flag this as a match. By using the NLTK to access Wordnet, we find that the word "business" and "love" from the event context have these words in common: "undertaking", "work", "activity" and "act". Not easily thought of by a human judge, the computational unit found the similarity that was marked false.

5.4 Context Size

As shown by the previous experiment, locating events within a piece of text can prove difficult. Since a similarity match between a term and an event is based upon the a hierarchy computation of similarity using a corpus to determine the possibility of two term coexisting in the same hierarchy branch. Using a larger context will affect the similarity results since the information content will change. We define the context size as a context

level, meaning that with no context exploration, we have a context level of zero.

Based on the description of the two different algorithms used for similarity calculation between sentences and within them (section 4.4), we have conducted an experiment for discovering whether the event detection algorithm performs better or worse when exploiting sentence similarity. For each run within the experiment, the size of the context is changed in order to discover the impact the context size has on the event detection. As mentioned previously, we utilize context exploration, that is, a fanout based on the word's hypernyms and hyponyms. Then, based on the similarity for each new sense towards the original event keyword, it is added to the context pool.

Our conjecture on utilizing context exploration is that we will see an increase in the positive rating for the event detection. As more senses are added to the context pool, the greater are the chances for a sentence to be related to the event if it contains a word in the vicinity of the context. However, when the context pool grows beyond a certain size, the percentage of positive rating compared to the false rating will decrease based on the fact that the context now contain too many relations that are not relevant for the event compared too.

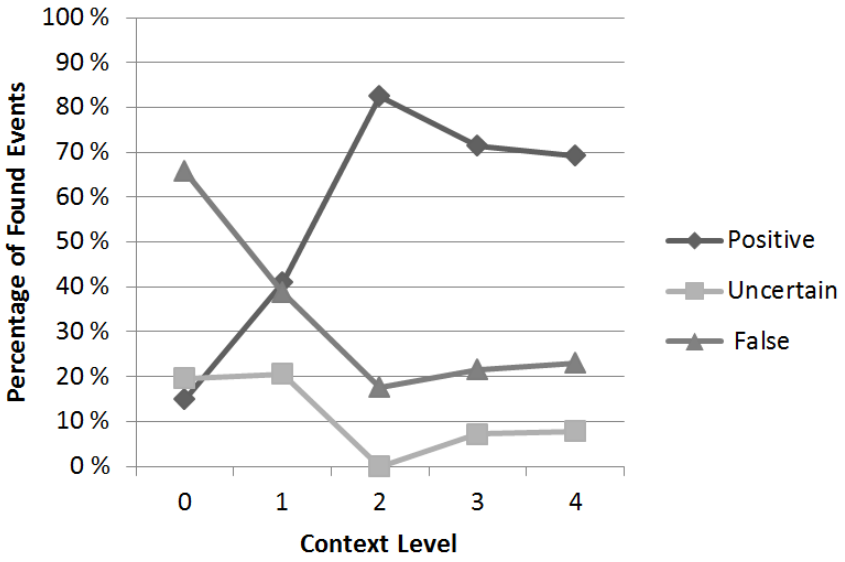


Figure 5.3: Accuracy of event detection based on context size

Interestingly, the graph in figure 5.3 illustrates that when increasing the context level, the positive rating of the result increases. At a context level of two, the positive rating is far better than in the previous experiment. For a context level of three and four, the rating decreases.

From figure 5.3 we can see that by increasing the context size we achieve

a better result for our semantic text detector. At a context size of zero we achieve a positive rating of only 14.8%. The uncertain rating is at 19.4%, while the false rating is at the 65.7% level. When the context size is zero, the system will only evaluate on the exact sense of the event, for instance will the word "kill" only be represented by the concept "kill" being a verb or noun. At context size 1, the system has made a concept expansion by locating the hypernyms and hyponyms of the given event. For the word "kill", these are "termination", "slaughter", "destroy", "assassinate" and "death blow" to mention a few. Now, all words are compared against a larger context, thus having the possibility for relating themselves to a larger set. Again, we can see in the graph that the result for the positive, uncertain and false rating are respectively 40.9%, 20.5% and 38.6%. Then, for a context expansion of 2 we have the result of 82.3% for positive, 0% for uncertain and 17.7% for the false rating. At expansion 3 we have 71.4% for positive, 7.1% for uncertain and 21.5% for false. At last, we have 69.2%, 7.8% and 23.0% for the different ratings.

Discussion

For the result of the zero expanding context we have a huge false rating at 65.7%. We think that it is because of all words entering the event detection system will be matched only against the specific event keyword. As most sentences will have some words with relatedness towards an event as explained previously in section 5.3. We can see that at context expansion one, the positive rating has gone from 14.8% to 40.9% and likewise, the false rating has gone down from its 65.7% margin to 38.6%. By expanding the context based on the implemented sense relatedness explained in section 4.3.1, an increase in the positive result is measured. This is also the case with our second context expansion, as the positive rating crosses the 82.3% line. However, we find a 0.0% uncertain rating and a 14.8% false rating. This illustrates the usage of related context exploration as an event will typically increase the similarity relatedness discovered, thus making events easier to detect. We reckon the drop in the positive rating and thus an increase in the uncertain and false ratings when going from two to three and four context expansions. Currently the level two context expansion is the limit from where adding more words to the event has a positive effect towards the end result. We see that when adding too many word to an event, the context gets flooded and it is hard to differentiate between the best terms and the second best terms to evaluate from. The best way of dealing with this is to use a different scoring mechanism. For instance, per expansion outwards, the score generated by the similarity computation will fall in order to preserve the original state of term weighting.

5.5 Sentence Relatedness

Calculating similarity against a single sentence may prove inaccurate for discovering events. The context of a scene may span across several seconds, or even minutes in a movie or television show. Therefore, we perform a similarity calculation between consecutive sentences in order to locate the context. This is a different context than the event context / context level discussed earlier. The computation may range from one to four sentences as described in section 4.4.2. Also, for locating the most prominent words within sentences, we use the implementation described in section 4.4.1.

The experiment conducted calculates the possibility for given events in a larger setting. This setting is currently between one and four sentences long. For each sentence in the context, the most prominent words is discovered. Based on the result of the first context calculation, the calculation of similarity between the different sentences are conducted. The generated result will contain a list of words linked with a modifier score as of how prominent they are. These are compared against the event context for locating the relatedness towards the event.

We have conducted an experiment to explore the possibility of better calculating relatedness towards an event context based on knowing the essence of a set of consecutive sentences. Our conjecture is that based on a number of consecutive sentences, we are able to increase our results based on a better understanding of the underlying context. However, we think that if the similarity generated for the context grows too large, events will be zeroed out due to too many events possible within that context. The implementation used is described in section 4.4.1 and section 4.4.2.

An interesting notice, is that the positive rating when calculating similarity between consecutive sentences, is that the positive rating is almost stable at around 40%, while the uncertain and false rating interleave each other for each context level. Figure 5.4 displays the result of the first experiment where we calculated the relatedness between sentences and events based upon a consecutive sentence length on two. That means that every two pairs of sentences were evaluated against the event context based on the similarity score between the two sentences as described in section 4.4.2. The y-axis display the percentage of detected events that where either positive, uncertain or false. The x-axis shows at what context level the results are based on.

For a context level of two, the positive rating received a score of 35.5%, the uncertain rating was a 4.4% while the false rating hit 60.0% of the total rating. A context level of three increased the positive results to 41.1%, while the uncertain and false rating were at 23.5% and 35.2% respectively. At context level four, the positive rating was at 37.8% while the uncertain rating dropped to 5.4%, leaving the false rating to 56.7%. When calculating the results of using a context level of five, our system achieved 41.1%, 23.5%

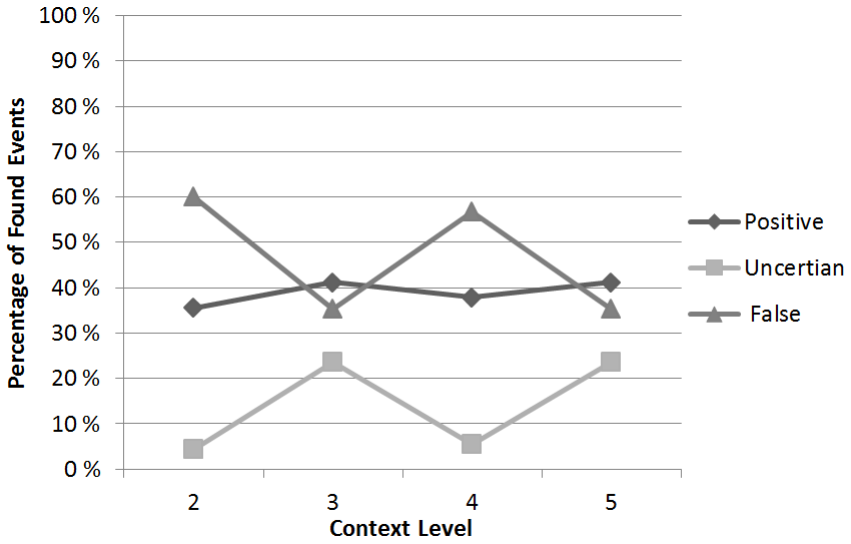


Figure 5.4: Accuracy of event detection using consecutive sentence similarity with length 2

and 35.3% for the different ratings.

The results from the second experiment regarding use of similarity calculations in consecutive sentences are displayed in figure 5.5. The size of consecutive sentences are three in this experiment meaning that every three sentences are compared and the most prominent words are selected and used for similarity calculations between the sentences and the constructed event. The y-axis display the percentage of detected events that where either positive, uncertain or false. The x-axis shows at what context level the results are based on. By calculating the similarity between three sentences, the algorithm achieves a positive rating around 55% for all context levels.

At a context level of two, the positive rating was calculated to a score of 52.1% while the uncertain rating received a score of 17.3%. The false rating at this context level was 30.4%. The result of the calculations of context level three, was for the positive rating 58.3% and for the uncertain and false rating, our system achieved a score of 16.6% and 25% respectively. With a context level of four, the following result were achieved: 50% for the positive rating resulting in 16.7% and 33.3% for the uncertain and false rating. At context level five, 60% and two times 20% rating were achieved.

Figure 5.6 displays the result of the first experiment where we calculated the relatedness between sentences and events based upon a consecutive sentence length on four. That means that every four pairs of sentences were evaluated against the event context based on the similarity score between the two sentences as described in section 4.4.2. The y-axis display the percentage of detected events that where either positive, uncertain or false. The

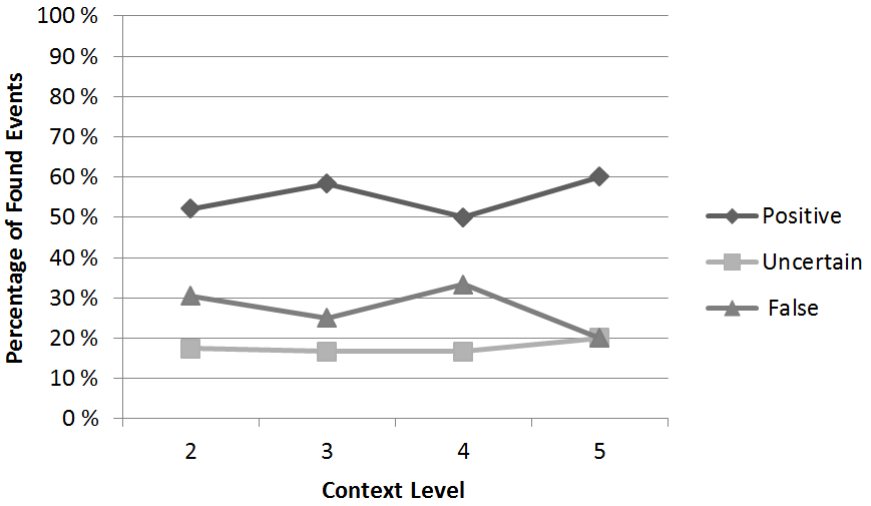


Figure 5.5: Accuracy of event detection using consecutive sentence similarity with length 3

x-axis shows at what context level the results are based on.

For a context level of two, the positive rating received a score of 64.3%, the uncertain rating was a 28.6% while the false rating hit 7.1% of the total rating. A context level of three increased the positive results to 100%, leaving the uncertain and false rating to 0.0%. This was also the case for context level four. At context level five, there were no events discovered by the algorithm, meaning that there were no results.

Discussion

The graph in figure 5.4 illustrates that our system actually discovers more false events than positive events. At context level two, the false rating is at 60.1% while the positive rating is only at 35.5%. When increasing the context level further, our system is able to locate 41.1% positive senses, staying on almost the same level as before, while the uncertain rating increases and the false rating decreases. For a context level of three, uncertain decreases and false increases again, while it is opposite from the level five context level.

As for the consecutive sentences similarity of three, most of the events discovered are of a positive nature for all context levels. We think that by first calculating the similarity between three sentences, locating the most prominent words and comparing them to the event context, reduces the number of false positives since there are less prominent words to compare.

Our argument of removing unwanted words from the similarity computation increases the number of true positives is illustrated by the last experiment. In figure 5.6, there are only positive events located at a context level four and higher. Therefore, by discovering the essence of the sentence,

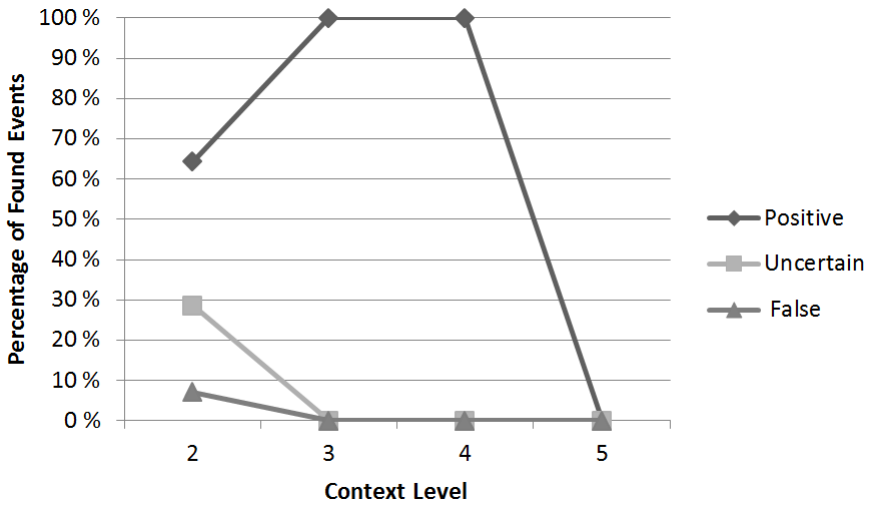


Figure 5.6: Accuracy of event detection using consecutive sentence similarity with length 4

we are able to produce far more true positives. However, we conjecture that if similarity between too many consecutive sentences are computed, there will be less total events discovered, even though those that are found, are correct.

5.6 Context Evaluation

As we experienced during the two last experiments, we have seen an increase in the accuracy of detecting events, however, we like to know the number of discovered positive events for each of the proposed methods. The score of each algorithm is presented as a percentage of the total number of events discovered by the human judges.

The graph display the results of the different algorithms in use. We present the precision of the five different uses of the algorithms. First with no consecutive sentence similarity, then the algorithms where the consecutive algorithms were in use calculating similarity between two and five consecutive sentences. The x-axis present the context level, under which the results were retrieved. The y-axis displays the percentage in how good an algorithm performed.

The key points from the graph in figure 5.7 are that the baseline algorithm performs best overall, with the consecutive two algorithm as closest competitor. Also, all instance of the algorithms perform best at a context level of two.

At context level two we see the best overall result. With no consecutive similarity a total of 85.71% of the positive events were discovered. By

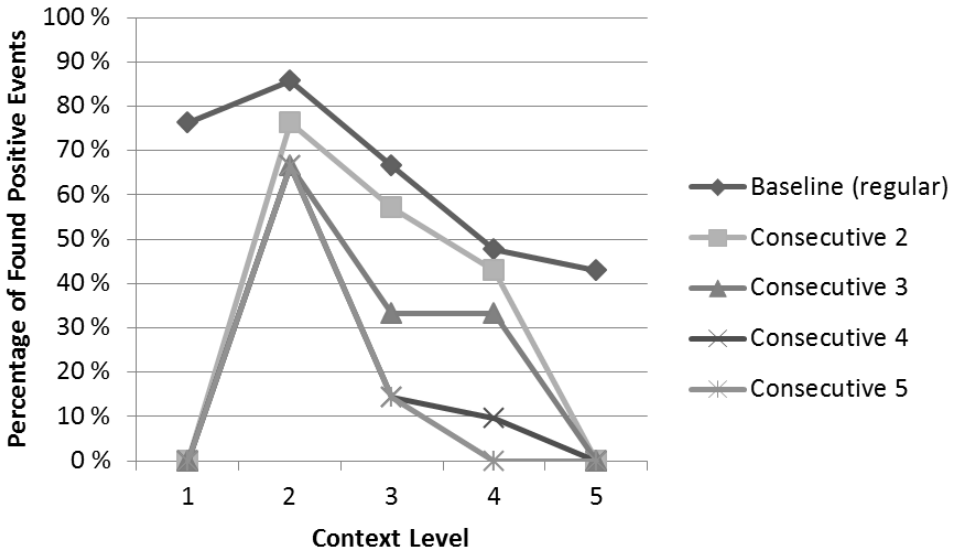


Figure 5.7: Percentage of discovered positive events when increasing the context

first calculating consecutive similarity between two sentences, the result was at 76.19%. Calculating consecutive similarity between three, four, and five sentences all achieved the same result of 66.67%. By increasing the context level to three the overall results are degraded. Best, is still the algorithm not using any consecutive similarity between sentences and achieves a rating of 66.6%. Calculating a consecutive similarity of two and three will respectively result in a score of 57.14% and 33.33%. The worst result is achieved when calculating the similarity between four and five consecutive sentences which are at 14.26%. At a context level of four, the result is again falling. With a consecutive similarity of one, that is without any similarity calculation between sentence, the score is a 47.62%. Close behind with an achieved rating of 42.86% is the calculated consecutive similarity of two sentences. The results of calculating consecutive similarity on three, four and five sentences are respectively 33.33%, 9.52% and 0.0%. At context level five, all approaches using consecutive similarity calculations fails and receive 0.0%. With no such calculation, the system received a the score of 42.86% discovery.

Discussion

With this experiment, we showed that the base algorithm had the best overall performance. It discovered 85.71% of all events marked by the human judges. As the graph illustrates, all instances of the similarity algorithm peaked at context level two. For the current implementation, the algorithms

performs best at a context level of two. A context level of zero and one make all the consecutive algorithms fail due to a too little context, therefore too many events are marked as positive by our system, though marked false by the human judges. All instances of the algorithm achieves a lower score when further increasing the context level from two and upwards. Best again is the base algorithm with the consecutive 2 instance as number two.

Though the base algorithm performs best by accuracy of positive events, our next experiment will measure the computational time for each of these algorithms.

5.7 Computation Throughput

In this section we will present timing experiments conducted with our prototype system. The results will be compared with the findings of the different accuracy experiments performed earlier. Our system will be evaluated based on its performance and according to the previous experiments.

Earlier experiments, namely those described and evaluated in section 5.3 to 5.5 shows that different results were achieved for the event detection based on the use of different attributes. As some experiments increased the event detection percentage, we wonder if performance took a hit for this. As known from the use of typical natural language processing techniques and algorithms, there may be huge time differences in the execution time. Based on our input being subtitles, we would measure the time performance in sentences per second.

All previously done experiments will be evaluated again, but with the focus to timing. First, we will perform the 'clean' experiment of calculating the total similarity towards an event by calculating the relatedness for each and every word within that sentence. This is done for a context level of zero to five, and will be the same for every algorithm in this experiment. Then, we time the algorithm that uses the discovered similarity between consecutive sentences as a first line filter before relating towards the event. The consecutive sentence similarity will be calculated between two to five sentences.

Figure 5.8 displays the result of our timing experiment. The graph shows how many sentences each instance of the used algorithm is able to process each second. Used as label on the y-axis is the number of sentences processed each second, while the x-axis displays the different context levels used during the experiment.

Note that the regular algorithm is far better than its competitors at context level zero, then at context level one, consecutive two and three performs as well as the baseline. At a context level of three, the baseline algorithm is not even close to the performance of all the other consecutive algorithms.

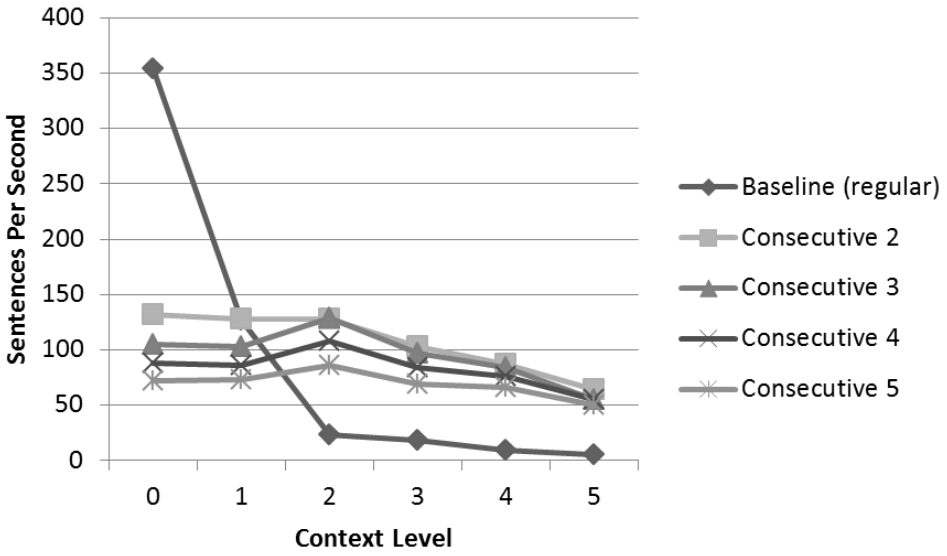


Figure 5.8: Time used evaluating each sentence using several algorithms

The graph is based on the following results: For regular calculated similarity, 353 sentences per second are processed towards the context and the zero context exploration. Then, when increasing the context level by one, sentences process falls to 127 sentence per second. Additionally, when increasing the context level, it has the possibility to process 23, 18, 9, and 5 sentences per second respectively towards context level two, three, four, and five.

The results for the consecutive sentence calculations all follow a distinct patter in figure 5.8. First, when calculating the similarity between two consecutive sentences, the algorithm described in section 4.4.2 is able to process 131 sentence per second. However, when calculating similarity between three and four consecutive sentences, the algorithm is able to process 128 sentences. Then, for three sentences, the result is 102 sentences, and 86 for four consecutive sentences. When calculating similarity between five different consecutive sentences, it is able to process 64 sentences per second.

By first calculating the similarity between three consecutive sentences, the system achieved the following timing: 105 sentence per second were calculated towards context level zero. Then, at context level one, 102 sentence were processed each second. At context level two, the system were able to increase the possible computation by processing 129 sentence per second. However, the sentence per second processed dropped to 97, 84, and 55 sentences per second respectively for context level three, four, and five.

When locating the similarity between four sentence, our system was able to process 87 sentence per second when evaluating against context level zero. When increasing the context level to one, the system processed 86 sentences

per second. Also, our system was able to process an additional 21 sentence as context level two, increasing the number to 107 sentence per second. At context level three and four, our system processed 84 and 75 sentences. For context level five, the algorithm preform at the level as the previous algorithm instance, processing 55 sentence a second.

By computing the similarity between five sentence, our system achieved the following results: 72 sentences were processed each second when computing similarity against a context of level zero. Then, our system were able to increase the processing by computing similarity between 73 and 85 sentences for context level one and two. At context level three, the algorithm handled 69 sentence each second that dropped to 65 sentence per second at context level four. 49 sentences were processed each second against the level five event context.

Discussion

This is the baseline for our evaluation, since by this calculation method, every word within the sentence is related towards the context. By first performing the consecutive sentence analysis, only the most important words within the sentences are calculated towards the event context. The difference between the regular algorithm and calculation, is that we experience a huge drop in sentences per second processing. By only computing the similarity between the most important words within the sentences, thus fewer, the computational time does not suffer that badly when increasing the context size. Common for the algorithm first calculating the consecutive similarity, is that the computational ability increases at context level two, opposed from *sim2*. Notice that all experiments using the consecutive sentence computation algorithm performed better than the regular computing algorithm at context level two and towards the end. Also, calculating the similarity between two sentences performed equal at context level one.

5.8 Computation Throughput with Database Access

We explained in section 4.5 that we could use the database to store preprocessed sentences. We thought that if, for instance, the similarity between consecutive sentences were already computed when the subtitles were imported into the system, our system would perform a lot better. Instead of using time computing similarity between consecutive sentences, the event detection system will now be able to extract the preprocessed sentences from the database and compute the similarity against the event context instantly.

The earlier timing experiment from section 5.7 did not use the enhanced version of the consecutive algorithm with database access. In this exper-

iment, we will use the enhanced algorithm that make use of a database. Since the result will be affected by the number of related senses between sentences, the experiment were conducted for five different events, and the result is based on the average result.

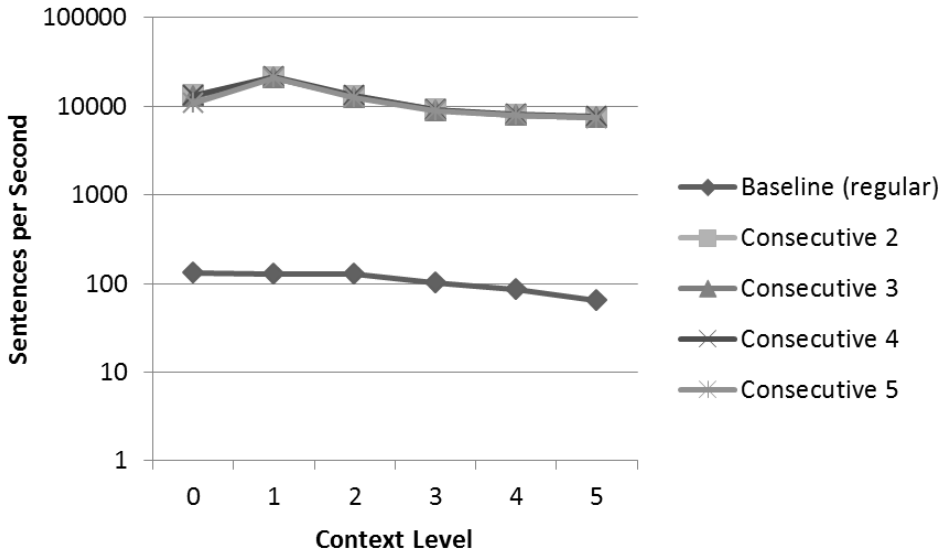


Figure 5.9: Time used evaluating each sentence based on the consecutive algorithm with database access.

Figure 5.9 displays the result of the timing experiment, when computing the similarity towards the event context based on preprocessed consecutive similarity between sentences. The graph shows how many sentences each instance of the used algorithm is able to process each second. Used as label on the y-axis is the number of sentences processed each second and is in a logarithmic scale. The x-axis displays the different context levels used during the experiment.

As seen in the figure, the first line illustrates the best performance achieved by the same algorithm, though without the use of preprocessed sentences in the database. In this case, it is the algorithmic instance of first computing the similarity between two sentences before evaluating against the event context.

The results achieved when using the database were pretty even for all instances of the algorithm. When using a consecutive similarity between two sentence, the algorithm was able to process 13230 sentence each second at context level zero. At context level one, the number increased to 21200 sentences. For context level two, it achieved 12794 sentences and dropped below 10000 sentence to 9173 at context level three. Our system managed to process 7872 and 7407 sentences each second at context level four and five.

For the instance where similarity were computed between three sentences, the result was almost equal as the first instance. 13000 sentences at context level zero, 21061 sentences were processed at context level one. At context level three, the number was 12719 with a difference on 23 sentence from the previous instance with two consecutive sentences. The algorithm instance achieved 7872 and 7407 sentences processed each second at context level four and five.

We had 13454, 21549, 13219, 9261, 8167, and 7682 sentences processed each second when using preprocessed sentences.

The number of sentences processed each second was at consecutive similarity between five sentences 10769 sentence at context level zero and 21079 sentence at level one. For context level two and three, the numbers were 12669 and 8993 sentences. 7874 and 7383 sentences per second were processed at context level four and five.

Discussion

As shown by earlier experiments, the algorithm described in section 4.4.2 reduces the number of words within a sentences. Preprocessing sentences and temporary storing the result in a database allows for further decreasing the computational time the algorithm needs for a successfully run. We have shown that our system is able to process far more sentences per second when using preprocessed sentences as input. This means that the algorithm computing and choosing the most prominent words from sentences when computing similarity for consecutive sentences actually is a much tougher job than computing the similarity for the remaining words towards the event context.

This approach is promising, since all major systems in for instance media delivery networks such as DAVVI [2] pre-process input when it first arrive in the system. Although the overall accuracy of this algorithm is not the best as shown in the experiment in figure 5.8. For the instance of first locating the most prominent words in two consecutive sentences proved to discover 76.19% of all events, less than 10% lower than the base algorithm was only able to process 23 sentences per second opposed to 128 sentences that was the case for the algorithm using consecutive sentence similarity. The algorithm should be developed further to increase the accuracy, but without increasing the computational time.

5.9 Summary

In this chapter we have presented the results from our experiments and evaluated the system based on our problem definition, scope and limitation. The first experiment concluded that our prototype was able to solve the problem of relating concepts that are not syntactical equal to each other. In

our second experiment we showed how our implemented algorithm worked by analysing a television show and comparing the sentences towards a generated event context. The result was rather disappointing, as the system only located 14.8% of the events marked as positive. However it matched our conjecture that by analysing towards a small context, it is difficult to discover relationships. Therefore, in the next experiment, we chose to look at how the algorithm performed when increasing the size of the event context. By increasing the context size, the overall results increased. At context level two, the algorithm was able to discover 82.3% of the positive result.

To further increase the understanding of the sentences, we introduced an algorithm that first calculated the similarity between a set of sentences, in order to locate the most prominent words and only calculate the similarity for them against the event context. This was to better capture the essence of the sentences. While the experiment showed that the algorithm now was able to locate a less percentage of the positive marked events, the next experiment illustrated that the result was not as bad as first thought. The experiment calculated the accuracy of locating the positive events. Here, all instances of the algorithm performed well at a context level of two. Best was the base algorithm discovering 85.71% of all the positive results while calculating the similarity between two consecutive sentences and the event context were able to locate 76.19% of the positive events.

We then performed two sets of timing experiments to find the time differences for using the different context sizes and algorithm instances. In the first experiment, the base algorithm was able to process 353 sentences per second at a context level of two. However, at context level one the algorithm was only able to process 127 sentences whereas the algorithm based on two consecutive sentences processed 128 sentences at context level one. At context level two, the base algorithm performed worst all all instance, only processing 23 sentences a second while the consecutive two algorithm still processed 128 sentences beat by the consecutive three algorithm that performed at 129 sentences each second. From here on and out, all algorithm instances decreased their sentences per second rate. To test our database access we used an enhanced version of the consecutive sentences algorithm. By retrieving the consecutive sentences similarity from the database, the algorithms were now able to process an astonishing 21549 sentences each second set by the consecutive three algorithm instance at context level one.

Chapter 6

Conclusion

We will in this chapter conclude this thesis. We have described and implemented a semantic text event detector that is able to analyse text and detect events based on sentence relatedness.

6.1 Achievements

The thesis is based on the problem definition, scope and limitation in section 1.1 and 1.2 that states:

> *This thesis shall design, implement and evaluate a similarity calculation service, based on probabilistic approaches. The focus will be on constructing algorithms that supports semantic similarity calculations between text. The basis for evaluating the algorithms will be by using human judges.*

This thesis has focused on the use of semantic analysis of text, as a base of the discovering interesting events. We have reviewed existing frameworks, algorithms and lexical databases that are currently used in academia. Based on our findings, we chose to implement a module that computes similarity based on semantic relatedness between a sentence and a context. The actual computation and context generation are described in chapter 4. Key components were the *input manager*, *context generation*, *similarity computation* and *storage*.

The experiments analysed the different parts of the implementation. First we ran an experiment illustrating the need for such an text event detection system based on textual analytics. The results discovered that of 2700 concepts found within a subtitle, 2000 of them were overlapping. By removing stopwords and lemmatize the words as done in most systems and search engines, only 21 concepts were overlapping. Using a regular syntactic search on that result would not have found any relatedness between those words. We showed that by utilizing semantic analysis, these concepts and

related words were discovered. Then, the actual performance of the different algorithms were tested, discovering the accuracy and computational throughput of each algorithm. By introducing the event context with several levels, we were able to improve the result of discovering correct events from 14.8% to 82.3% using the exact same algorithm computing against the second context level. Based on the computational time, the algorithm calculating consecutive sentence similarity performed best when computing similarity against the second context level. It was able to locate 76.2% of all the positive events, however, those only made out 35.5% of its total event hits. By preprocessing the input and store it in a database, we saw a tremendous increase in the computational throughput when using the consecutive similarity based algorithm.

Based on the results in our experiments, we state that semantic analysis of text as a type of event detection mechanism is a promising approach. By further improving the implementation of the algorithms used, this type of analysis should be able to reach even better results. The prototype is generic enough to accept all kind of text, such as words and sentences as input. We have pointed out several times its useful areas, such as summary generation and event discovery.

6.2 Related Work

Previous work from Varelas et al. [6] propose what they call a term expansion. A lack of similar words in two documents does not mean that they are unrelated, thus performing a term expansion allows for increasing the number of possible similarities. They generate a bag-of-words, containing important synonyms for later processing. We relate our context exploration model to this work, and how to rank the different available synonyms in the event context. Such an approach is also done by Li et al. [44], which scales the similarity based on the depth found in the taxonomy used. However, Mihalcea et al. [8] proposes a similarity calculation based on a second-order word relations in a hierarchy that prove adoptable to different application domains since it is based on corpus statics.

We relate our semantic text event detector mainly towards this work, as they also exploits relationships between the given words based on term expansion for better similarity calculations.

6.3 Concluding Remark

We have successfully implemented a set of algorithms for discovering semantic relatedness between words, thus being able to discover events. By comparing sentences towards an event context, the level of success rating increased drastically. The base algorithm increased its accuracy from 14.8%

to 82.3% by the use of context exploration. All results were validated and confirmed by human judges.

Our prototype system consists of three different modules, computing the similarity towards events that defines text. The algorithms used are based on word relationships defined in lexical databases and their probability of occurrence in text. To analyse text, we used natural language processing.

6.4 Future Work

In this section, we present issues that should be further explored in order to improve the implementation of the prototype and the algorithms used.

- Develop the algorithms further. This is at least the case for the consecutive similarity algorithm since its perform quite well for locating the number of positive senses, and computation time. It did however struggle with the accuracy, as it discovered most false events. There are numerous parameters that can be changed and tweaked within that algorithm and it should be evaluated better.

- Implement a better scoring algorithm when used in context exploration as the algorithm used today does not differentiates between the different levels of hypernym / hyponym. Such an implementation could for instance contain different levels of scoring a word when evaluating against the event context, so that hyponyms are better scored for similarity that hypernyms since hyponyms are more specific that hypernyms.

- Add better support for applications. Today, applications can only access the similarity service through a webserver serving as a host. Based on the application's query, our service should be able to process the similarity on an algorithm specified by the user, or at least change the parameters within the algorithm.

- Use feature extraction in addition to NLP semantic similarity. From the background, it is known that the current models used today are not good enough and is computational heavy. Due to time constraints of creating and training such a model, it was not taken into account when designing this prototype.

- Use two stage processing by combining the use both types of similarity algorithms for better accuracy. By first using the consecutive algorithm (section 4.4.2), discovering a broad spectre of events, though most of them false for the current implementation, the discovered events can trigger a second level processing for a more accurate search. Remember that the best results (section 5.6) were achieved

by a context level of two. Here, the full processing of the consecutive algorithm was still far better than the regular processing. The second level processing can use the base algorithm described in section 4.4.3 which is more accurate. By creating this two stage processing, we can save computational processing power by not process every sentence that thoroughly.

References

[1] A. Ulges, M. Koch, D. Borth, and T. M. Breuel, "Tubetagger - youtube-based concept detection," in *Proceedings of the 2009 IEEE International Conference on Data Mining Workshops*, ser. ICDMW '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 190–195. [Online]. Available: http://dx.doi.org/10.1109/ICDMW.2009.41

[2] D. Johansen, H. Johansen, T. Aarflot, J. Hurley, A. Kvalnes, C. Gurrin, S. Zav, B. Olstad, E. Aaberg, T. Endestad, H. Riiser, C. Griwidz, and P. Halvorsen, "Davvi: a prototype for the next generation multimedia entertainment platform," in *Proceedings of the seventeen ACM international conference on Multimedia*, ser. MM '09. New York, NY, USA: ACM, 2009, pp. 989–990. [Online]. Available: http://doi.acm.org/10.1145/1631272.1631482

[3] V. Snasel, P. Moravec, and J. Pokorny, "Wordnet ontology based model for web retrieval," in *Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 220–225. [Online]. Available: http://portal.acm.org/citation.cfm?id=1105926.1106251

[4] J. Castillo and M. Cardenas, "Using sentence semantic similarity based on wordnet in recognizing textual entailment," in *Advances in Artificial Intelligence IBERAMIA 2010*, ser. Lecture Notes in Computer Science, A. Kuri-Morales and G. Simari, Eds. Springer Berlin / Heidelberg, 2010, vol. 6433, pp. 366–375, 10.1007/978-3-642-16952-6_37. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16952-6_37

[5] P. Achananuparp, X. Hu, and X. Shen, "The evaluation of sentence similarity measures," in *Proceedings of the 10th international conference on Data Warehousing and Knowledge Discovery*, ser. DaWaK '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 305–316. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85836-2_29

[6] G. Varelas, E. Voutsakis, P. Raftopoulou, E. G. Petrakis, and E. E. Milios, "Semantic similarity methods in wordnet and their application to information retrieval on the web," in

Proceedings of the 7th annual ACM international workshop on Web information and data management, ser. WIDM '05. New York, NY, USA: ACM, 2005, pp. 10–16. [Online]. Available: http://doi.acm.org/10.1145/1097047.1097051

[7] A. Budanitsky and A. Budanitsky, "Lexical semantic relatedness and its application in natural language processing," Tech. Rep., 1999.

[8] R. Mihalcea and C. Corley, "Corpus-based and knowledge-based measures of text semantic similarity," in *In AAAI06*, 2006, pp. 775–780.

[9] D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young, "Computing as a discipline," *Commun. ACM*, vol. 32, pp. 9–23, January 1989. [Online]. Available: http://doi.acm.org/10.1145/63238.63239

[10] A. M. Turing, "Computing machinery and intelligence," pp. 433–460, 1950. [Online]. Available: http://cogprints.org/499/

[11] S. J. DeRose, "Grammatical category disambiguation by statistical optimization," *Comput. Linguist.*, vol. 14, pp. 31–39, January 1988. [Online]. Available: http://portal.acm.org/citation.cfm?id=49084.49087

[12] R. Girju, P. Nakov, V. Nastase, S. Szpakowicz, P. Turney, and D. Yuret, "Semeval-2007 task 04: classification of semantic relations between nominals," in *Proceedings of the 4th International Workshop on Semantic Evaluations*, ser. SemEval '07. Stroudsburg, PA, USA: Association for Computational Linguistics, 2007, pp. 13–18. [Online]. Available: http://portal.acm.org/citation.cfm?id=1621474.1621477

[13] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," *COMPUTATIONAL LINGUISTICS*, vol. 19, no. 2, pp. 313–330, 1993.

[14] Brown corpus. Brown University. [Online]. Available: http://khnt.aksis.uib.no/icame/manuals/brown/

[15] P. Resnik, "Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language," *Journal of Artificial Intelligence Research*, vol. 11, pp. 95–130, 1999.

[16] ——, "Using information content to evaluate semantic similarity in a taxonomy," in *In Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995, pp. 448–453.

[17] M. Sahami and T. D. Heilman, "A web-based kernel function for measuring the similarity of short text snippets," in *Proceedings of the*

15th international conference on World Wide Web, ser. WWW '06. New York, NY, USA: ACM, 2006, pp. 377–386. [Online]. Available: http://doi.acm.org/10.1145/1135777.1135834

[18] T. Pedersen, "Information content measures of semantic similarity perform better without sense-tagged text," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, ser. HLT '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 329–332. [Online]. Available: http://portal.acm.org/citation.cfm?id=1857999.1858046

[19] K. Sparck Jones, *A statistical interpretation of term specificity and its application in retrieval*. London, UK, UK: Taylor Graham Publishing, 1988, pp. 132–142. [Online]. Available: http://portal.acm.org/citation.cfm?id=106765.106782

[20] C. Kiddon and Y. Brun, "That's what she said: Double entendre identification," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT11)*, 2011.

[21] R. Mihalcea, "Making computers laugh: Investigations in automatic humor recognition," in *In Proc. of the Joint Conference on Human Language Technology / Empirical Methods in Natural Language Processing (HLT/EMNLP*, 2005.

[22] (2010, November) Cleverbot. amix. [Online]. Available: http://singularityhub.com/2010/01/13/cleverbot-chat-engine-is-learning-from-the-internet-to-talk-like-a-human/

[23] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl. Acquis.*, vol. 5, pp. 199–220, June 1993. [Online]. Available: http://portal.acm.org/citation.cfm?id=173743.173747

[24] S. L. Reed and D. B. Lenat, "Mapping ontologies into cyc," 2002.

[25] Sensus. isi.edu. [Online]. Available: http://www.isi.edu/natural-language/resources/sensus.html

[26] P. Resnick and H. R. Varian, "Recommender systems," *Commun. ACM*, vol. 40, pp. 56–58, March 1997. [Online]. Available: http://doi.acm.org/10.1145/245108.245121

[27] M. Nathan, C. Harrison, S. Yarosh, L. Terveen, L. Stead, and B. Amento, "Collaboratv: making television viewing social again," in *1st International Conference on Designing Interactive*

User Experiences for TV and Video, vol. vol 291, ACM. Silicon Valley, CA: ACM, 10/22/2008 2008, pp. 85–94. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1453805.1453824

[28] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[29] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, "Grouplens: applying collaborative filtering to usenet news," *Commun. ACM*, vol. 40, pp. 77–87, March 1997. [Online]. Available: http://doi.acm.org/10.1145/245108.245126

[30] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," in *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '98. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1998, pp. 668–677. [Online]. Available: http://portal.acm.org/citation.cfm?id=314613.315045

[31] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," 1999.

[32] (2010, November) The reddit algorithm explained. amix. [Online]. Available: http://amix.dk/blog/post/19588

[33] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966. [Online]. Available: http://sascha.geekheim.de/wp-content/uploads/2006/04/levenshtein.pdf

[34] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Commun. ACM*, vol. 7, pp. 171–176, March 1964. [Online]. Available: http://doi.acm.org/10.1145/363958.363994

[35] W. E. Winkler, "String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage," in *Proceedings of the Section on Survey Research Methods, American Statistical Association*, 1990, pp. 354–359.

[36] J. Jiang and D. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," in *Proc. of the Int'l. Conf. on Research in Computational Linguistics*, 1997, pp. 19–33. [Online]. Available: http://www.cse.iitb.ac.in/ cs626-449/Papers/WordSimilarity/4.pdf

[37] D. Lin, "An information-theoretic definition of similarity," in *In Proceedings of the 15th International Conference on Machine Learning*. Morgan Kaufmann, 1998, pp. 296–304.

[38] Z. Wu and M. Palmer, "Verbs semantics and lexical selection," in *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, ser. ACL '94. Stroudsburg, PA, USA: Association for Computational Linguistics, 1994, pp. 133–138. [Online]. Available: http://dx.doi.org/10.3115/981732.981751

[39] C. Leacock and M. Chodorow, *Combining local context and WordNet similarity for word sense identification.* In C. Fellbaum (Ed.), MIT Press, 1998, pp. 305–332.

[40] G. A. Miller, "Wordnet: A lexical database for english," *Communications of the ACM*, vol. 38, pp. 39–41, 1995.

[41] R. Richardson, A. F. Smeaton, R. Richardson, and A. F. Smeaton, "Using wordnet in a knowledge-based approach to information retrieval," Tech. Rep., 1995.

[42] J. B. Lovins, "Development of a stemming algorithm." Jun. 1968. [Online]. Available: http://stinet.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=AD0735504

[43] M. F. Porter, *An algorithm for suffix stripping.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 313–316. [Online]. Available: http://portal.acm.org/citation.cfm?id=275537.275705

[44] Y. Li, D. McLean, Z. A. Bandar, J. D. O'Shea, and K. Crockett, "Sentence similarity based on semantic nets and corpus statistics," *IEEE Trans. on Knowl. and Data Eng.*, vol. 18, pp. 1138–1150, August 2006. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2006.130

[45] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. O'Reilly Media, Inc., 2009.