uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Martin Scaiano

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Automatic Extraction of a Semantic Representation of English Sentences

TITRE DE LA THÈSE / TITLE OF THESIS

Diana Inkpen

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Stan Szpakowicz

Franz Oppacher

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Automatic Extraction of a Semantic Representation of English Sentences

By

Martin Scaiano

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements

For the Master degree in

Computer Science

School of Information Technology

Faculty of Engineering

University of Ottawa

Your file Votre référence
ISBN: 978-0-494-51855-7
Our file Notre référence
ISBN: 978-0-494-51855-7

Abstract

Many natural language processing tasks are implemented using methods which do not understand language but rather its statistical properties. This thesis presents a system for extracting a semantic frame representation from sentences with the intent of building document-understanding systems.

Many frame extraction systems have been motivated by a competition, such as SemEval 2007 task 19, or as a proof of concept; these systems have not been applied to further work in other tasks. The system presented here is intended as a foundation for developing new algorithms that use a frame-based semantic representation. Frame extraction has two parts: frame identification and role labeling. Frame identification is a fairly simple task, while role labeling is a more difficult task that has received extensive attention in recent years. The system presented here uses two layers of semantic labeling in all tasks: first a general-purpose role labeling done in the initial parse tree and then, using that information as guidance, the roles are labeled with frame-specific information. Several machine learning frameworks are explored for both tasks, usually with varying features and with different divisions of the tasks per classifier. We experimented with three types of classifiers: Naïve Bayes, Decision Trees, and Support Vector Machines.

The system's performance is evaluated by cross-validation on the FrameNet data, and against the SemEval 2007: Frame extraction task data. The system presented here is comparable to other state-of-the-art systems. When considering the intended use of the system and the fact that no optimizations have been done for the SemEval 2007 task, the system's results are promising, especially from the point of view of its precision.

Acknowledgements

I would like to thank Dale, my wife, for supporting me through the difficult times, late nights and cranky moods. Many thanks to Diana Inkpen, my supervisor, who has helped guide me through this work and my Master's degree. Thanks to Collin Baker and the FrameNet team for both access to FrameNet and the SemEval 2007 data.

Table of Contents

List of Tables

List of Figures

1 Introduction

"Let's start at the very beginning

A very good place to start",

The Sound of Music

1.1 Personal Introduction to the Thesis

I have had many discussions with friends, family and strangers who have asked me what I am researching and what my thesis is about. I usually start by organizing my thoughts and targeting the explanation to the audience. Here is one of the simplest, clearest explanations I can give.

I am trying to get computers to understand text. Imagine something out of Star Trek where you could ask the computer a question and get a clear answer. "Computer, given X, Y, Z symptoms, what sickness might I have." Currently, people might go to Google and type some keywords, "headache, fever, hair loss", then they filter through the web pages and content for an answer. It would be much more efficient to have the computer filter through the content and return a direct answer. This is the long-term dream. I am starting towards this goal by simply trying to represent the meaning of language in a uniform way.

Most languages can express the same thing in different ways. For example when purchasing something, you could say:

"I bought something from the store."

"I purchased something from the store."

"The store sold me something."

"I got something at the store."

All the example sentences above express the same basic idea, but how can we represent these as the same thing? This is the goal, to represent the expressions above in a fashion where we can easily recognize that they mean the same thing. Details of the representation will be presented later.

One of the more immediate applications of this type of work could be detecting new content from a stream of content. Newspapers publish articles every day. New articles often contain a lot of information from previous articles about the same topic. One of the first uses may be to compare information or meaning and see if they overlap, then to summarize newspaper articles with only the information new to the reader. The following

is a fictitious example of a series of newspaper articles summarized over time. One may imagine that each article actually repeats most of the previous information but all the reader is concerned with is the new information.

> "A dead body was found by the river on Monday."
>
> "The body has been identified as John Doe. John Doe was a local store owner."
>
> "John Doe is believed to have been murdered and a search for his murderer is underway."
>
> "John Doe's murderer has been captured..."

The example could continue on for months following the trial but we will stop now. While this may be a morbid example, it demonstrates the point well, as new information is regularly discovered in criminal investigations. There are dozens of other applications for a representation of meaning.

Ten years ago FrameNet was barely started. Researchers have been trying to represent and process meaning since computer science started decades ago, but the tools, resources and processing power were not available.

For the purpose of this thesis, my objective is to extract the meaning of language, specifically an English sentence. I expect that further study (in my PhD) will result in some algorithms that can be directly useful to people.

1.2 Formal Thesis Goal

Many current natural language processing algorithms are based on statistical methods, which are powerful and fast. Symbolic methods have traditionally been slower, less robust and lacking in breadth, but symbolic methods also have many processing advantages. The application of a symbolic representation of meaning to many common natural language processing tasks is of interest to us.

Frames (Fillmore, 1976) are an interesting representation, which describes actions, events and objects with a list optional of parameters (for more information about frames see section 1.4.1); FrameNet (Fillmore, Ruppenhofer, and Baker 2004) is a useful resource for working with frames. To apply a frame representation to typical natural language processing tasks would first require a system to automatically extract the representation.

This thesis presents a system that automatically produces a frame representation of sentences. The system automatically identifies and labels frames and their frame elements. This process was done manually for the sentences in FrameNet. The system then uses FrameNet and machine learning to identify frames and label frame elements.

FrameNet and typical SRL systems usually only represent how a single frame or target word relates to the rest of the words in the sentence. In other words, the sentences of FrameNet annotate only one frame, while our system annotates all possible frames for the sentence. The system produces a single representation, which attempts to capture how all frames, frame elements, entities and values interact. The representation produced by this system does not satisfy all the functionality suggested in this thesis. The representations of simple sentences with similar meanings (as in section 1.1 and 1.4.1 for example) should be very similar.

SemEval 2007 task 19, Frame semantic structure extraction (Baker et al., 2007), was a very similar task, which required participant systems to automatically extract a particular frame representation from plain text sentences. This SemEval task provides comparison measures of the precision and recall for the system presented here.

1.3 Outline

This thesis describes the basics and background of this research; including thorough descriptions of the resources used. It also contains a summary of the research as it progressed.

In this *Introduction* chapter, much of the information necessary to understand this work is presented in clear simple terms and using examples. This section introduces some concepts not always found in undergraduate computer science studies but that are

foundational to understanding this document. Most of the details presented in this section may already be known to natural language processing computer scientists.

The next chapter, Related Work, describes research similar to this thesis. It is only a summary of an extensive history of research; for more details please see those specific papers, as referenced. Reading these papers is not required to understand this document.

The *Tools* chapter introduces the tools and resources used for this work. Each tool and resource is described in both enough general detail to understand the document, and enough technical detail to reproduce this work.

The *Representation* chapter analyzes the needs of the representation; then it examines possible shortcomings and advantages of the representation used. This section uses a mix of linguistic, logic, and computer science knowledge. The chapter focuses heavily on frames, and FrameNet, which are introduced in the *Tools* chapter.

Experiments describes many of the different experiments that were run and how they progressed. The experimental results may inspire different ideas or conclusions. Enough information is provided to allow reproduction of the experiments. The experiments are written in series, showing the progression of the research. The most important experiments are listed, but many small, less significant experiments have been omitted from the dissertation.

In the *Evaluation* chapter, the system presented here was tested using the SemEval 2007 Frame Extraction Task data, and its results compared to those of the participants. This evaluation is beneficial by providing understanding of the strengths and weaknesses of the system; and by providing a credible measure of the quality of the system.

The *Future Work* chapter discusses tasks that could be performed to improve the system's performance, improvements to provide more meaningful information to clients, and some ideas on applications of such a system.

The *Appendices* contain a glossary of terms, a description of the implementation, a sample arff[1] file demonstrating some of the advances discussed in later sections and a list of the dependency relations provided by the Machinese Semantics parser by Connexor[2] (Järvinen, 2003).

1.4 Background Information

There are many ways to think about language but generally there seem to be two categories of representing the meaning of text: logical and structured. They are basically two ends of a spectrum, balancing expressibility and extensibility with processing power and clarity.

Logical representations are based on various forms of logic, such as propositional logic, and first order predicate logic. Logical forms usually have strict rules, which often result in awkward ways of representing meanings. Using logic to express human language is effectively expressing language as mathematical statements; it is not a natural thing for humans. Logic forms have the advantage that automated reasoning can be easily done. There are many mature logical representations, which can be used for processing. OWL (see Appendix I)(Harmelen et al., 2003) is a fairly mature logical representation that has several tools for automated reasoning.

Structured representations tend focus on the ability to represent the diversity of language's implied properties: meaning, sentiment, connotation, and perspective. Structured representations are usually more difficult to process because they are more expressive. There are fewer mature[3] algorithms for structured representations.

Logical representations suffer from limited expressibility, awkward handling of some expressions, and poor extensibility, yet they offer mature proven methods of

---

[1] Arff files contain training and testing data used by Weka, which is the machine learning toolkit used in this research. For more details see section 3.1 *Weka* and section 3.2.4 *Arff files.*

[2] This research uses the Machinese Semantic Parser by Connexor. For more details see section 3.3 *Connexor*

[3] When referring to mature algorithms or representations, we mean algorithms or representations which have at least a decade of frequent usage and which have been proven to be effective, consistent, and predictable.

reasoning and processing. Structured representations can often express a lot of languages subtleties and implied meanings, and yet are surprisingly extensible. Structured representations are difficult to process, and can often express the same thing in many different ways. Both representations are simply opposite ends of a scale. Some representations try to gain some middle ground, balancing the advantages and disadvantages of each method. Frames, which are described in the next section, are a good example of a middle ground representation.

1.4.1 Frames

Frames represent events, objects and situations in a parametized way. Consider the following situation, "Bob purchasing a red car from Ford." The frame (event) would be *purchasing*, the parameters (or frame elements) would be a *buyer* (Bob), a *seller* (Ford), a *theme* (the red car). There would undoubtedly be other parameters to the purchasing frame, such as the price and warranty. In language, not all the parameters are given, nor are they always in a single sentence. The following sentences also could represent the sentence and frames described above:

"Bob bought a red car from Ford."

"Bob's red car was purchased from Ford."

"Bob owns a red car which he got from Ford."

"Ford sold a red car to Bob."

"Bob bought a car from Ford. The car is red."

All of these sentences can be expressed with one uniform frame. The frame is graphically represented in Figure 1.

The "car" is also represented as a frame, which has its own parameters. In this case the car has the color red, but other possible frame elements could be the make and model.

The representation used in this work recognizes the people and entities are critical to understanding text, thus they have a special representation; it is a frame with the following elements: first name, last name, age, and sex. Ford is represented as entity though

it could also be represented with frame for companies. Figure 1 represents words which do not have frames and are not entities as atoms.

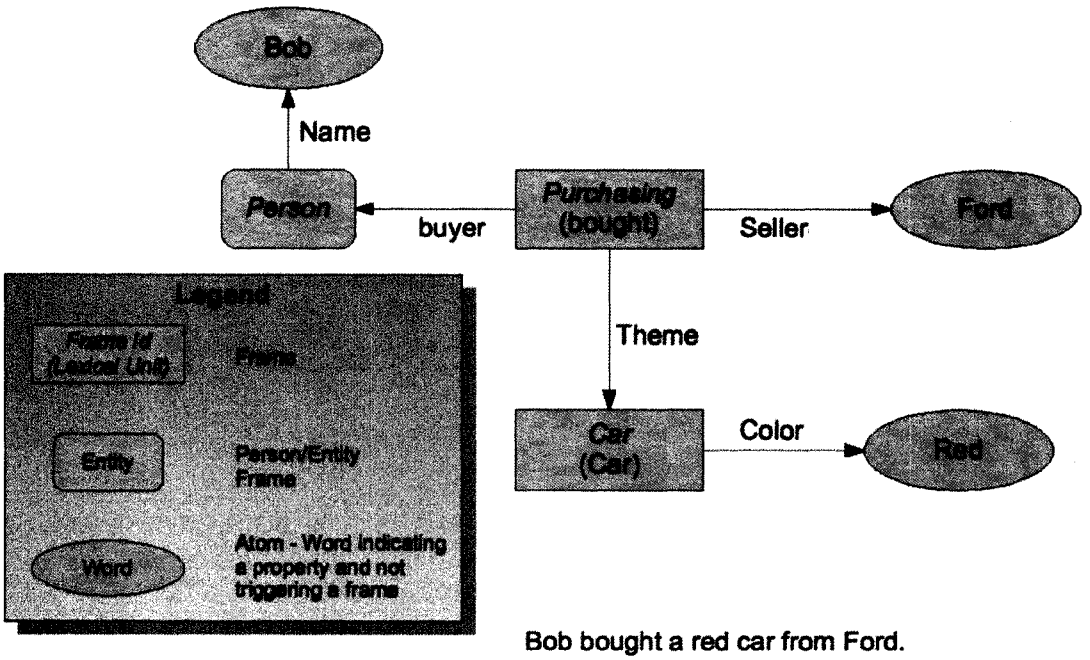


Figure 1 - Example of a frame representation

A benefit of many frame representations is that they can often be converted into logical representations, which can be used for automated reasoning. FrameNet's frames can be converted to OWL, which is a popular and mature logical representation. FrameNet's frames thus provide an effective balance between expressive power and the ability to apply logical processing, that should be beneficial to this research.

Like most representations, in particular logical forms, frames require definitions before they become useful. Without a definition for the purchasing frame, we cannot know that it expects a buyer, or seller, or even a theme. For many years a group of researchers at Berkeley have been working on FrameNet, a collection of definitions and meta data for frames that can be used to represent parts of the English language.

1.4.2 FrameNet

FrameNet is currently the largest and best-known resource for English frame definitions. To some degree frames are language-independent because the parameters of an object or situation are often language-independent; however, not all frames are language-independent, since they may have language-dependant implications. Many of FrameNet supporting resources are language-dependent.

While many different versions of FrameNet exist for different languages: Japanese (Ohara et al., 2003); Spanish (Carlos et al., 2003); German (Boas et al., 2006), none are as complete as Berkeley FrameNet, which provides a lexicon of English frames and examples. Even though there are a few other versions of FrameNet available for different languages, this document will refer to Berkeley FrameNet simply as *FrameNet*.

Some of the resources provided by FrameNet are definitions of frames, ontology[4] of frames, conversions between frames, lists of lexical units (defined below) per frame, and annotated text exemplifying the frames and lexical units.

The current release of FrameNet, 1.3, contains definitions for 795 frames. These frame definitions are critical to building our actual representation. The list of defined frames is small, though it covers many common verbs and nouns.

FrameNet provides information on how to map between two related[5] frames. Frame elements tend to be specific to the frame; to understand the relationship between any two frames, if one exists, we must know how their frame elements are mapped or converted. For example *commerce_goods-transfer* is a specialization of the *transfer* frame. The participants in each frame are buyer/seller and donor/recipient, respectively. To convert between the frames, the frame elements must be correctly mapped.

---

[4] The FrameNet ontology provides several relationships between super-frames and their derived works, such as *commerce_goods-transfer* is a specialization of the *transfer* frame. For more details see section 3.4.4 *Ontology*

[5] Frames can be related through many relationships; the most common relationship is the inheritance relationship, where a derived frame is a specialization of a super (parent) frame. For more details on all the frame relationship see section 3.4.4 *Ontology*

Lexical units are the words or expressions that can suggest the presence of a frame, for example "buy" suggests a purchasing frame. Lexical units are crucial for identifying when a frame should be evoked (used to represent the given situation). A word or expression may suggest multiple possible frames; for example, "bank", may suggest an building for storing money, the side of river, to lean something sideways, or to bet on something (as in "don't bank on it"). In some cases, even when one or more lexical units exist, none of them may represent the current situation. FrameNet 1.3 has about 10 000 lexical units. This does not cover much of the English vocabulary, approximately half a million words; it does include many frequent words, which describe common situations.

FrameNet provides annotated texts, which are sentences with frame-evoking words labeled and their frame elements labeled. These texts can be used to train and test various machine-learning systems to correctly recognize frames and frame elements. FrameNet is described in greater detail in section 3.4.

1.4.3 Machine Learning

Machine Leaning (ML), a sub-field of computer science, has produced a number of algorithms, which can learn to predict patterns from sets of examples. The algorithms used in this research are all classifiers; they learn to classify (label) examples into classes based on the features each example exhibits.

After being trained classifiers typically take an example and classify it into a target class[6]. There are a few algorithms, which can classify multiple examples at the same time and provide multiple classes; for example: Conditional Random Fields (Lafferty, McCallum, and Pereira, 2001) and Hidden Markov Models (Rabiner, 1989). The research presented here uses three classifiers: Naïve Bayes, Decision Trees and Support Vector Machine, all of which provide one class per example (a brief description of each classifier can be found below in sections 1.4.3.1 *Naïve Bayes*, 1.4.3.2 *Decision Tree*, 1.4.3.3 *SVM*).

---

[6] Most classifiers actually provide probabilities for each class and the most probable class is assumed to be the correct class. There are many applications that use the resulting distribution of probabilities. None of experiments described in this thesis leverage this distribution of probabilities.

A typical classification task could be classifying books into genres: science fiction, romance, historic, fantasy, etc. Potential features for each example many include author, publisher and title. The classifier would be trained on a set of examples then the classifier could then be used to classify future examples into the genres present in the training examples.

1.4.3.1 Naïve Bayes

Naïve Bayes (Mitchell, 1997) is a ML algorithm based on calculating the probabilities of each feature value implying each class. To classify an example, the probabilities of each feature being in a particular class are summed up and the class with the highest probability is selected. Naïve Bayes assumes features are independent (do not correlate) but has proven effective on both independent and dependent features.

1.4.3.2 Decision Tree

Decision trees (Mitchell, 1997) are a method for formally representing decision logic; typically they divide a large question into a tree of smaller questions. The answer to each smaller question narrows the possible overall solutions to particular branch of the tree; the process is repeated until leaf node, which contains the final solution, is found. Figure 2 is a graphic example of a decision tree, which decides whether someone's car insurance should be high, medium or low. The nodes (clouds in Figure 2) represent questions, the edges (arrows in Figure 2) represent possible answers to the questions, and the leaf nodes (ovals in Figure 2) represent the final classification: high, medium, low.

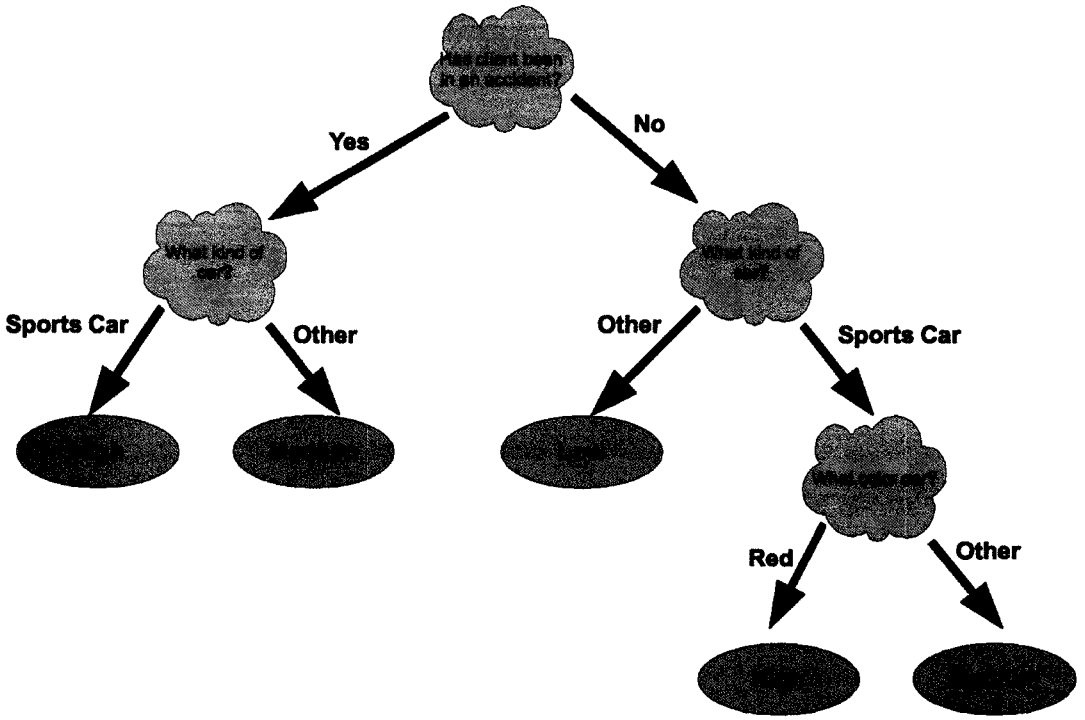


Figure 2 - Example decision tree for deciding if car insurance costs will be high, medium or low

The example above has been simplified; most decision trees will have more than 2 branches from any given node; the answers may be numbers or ranges of numbers; probabilities may also be represented for each branch.

A ML decision tree algorithm tries to infer a tree from a large set of examples. Decision trees are one of very few ML algorithms that produce a model that can be intuitively understood and validated by humans. Theoretically, decision trees can process dependent features, which gives them an advantage over Naïve Bayes.

1.4.3.3 SVM

Support Vector Machines (SVMs) (Cristianini and Shawe-Taylor, 2000) are an advanced ML algorithm. SVMs represent examples in an N-dimensional space where each axis usually corresponds to a feature. The basic functionality of an SVM is to find a dividing hyperplane (separation) between classes of data. Usually there are numerous options for a hyperplane and a SVM chooses the one that maximizes the margin between classes (has the greatest distance from all border examples). Another function of a SVM is to create

additional dimensions to better divide data. These additional dimensions do not alter the original dimensions but are created from the existing dimension.

1.5 Contributions

In this thesis we make the following research contributions:

First, we present new methods for role labeling and frame identification based on initially using a dependency parser on text, and then doing a detailed semantic analysis. The use of a dependency parse and shallow semantic information has only been tried once before (Johansson and Nugues, 2007).

Second, a system, which extracts a semantic frame representation from sentences, is presented. The process of identifying frames and frame elements is normally manual because the correct labeling and creation of frames requires knowledge, which is either not available or usable by computers but is intuitively available to humans. One other system is currently available (a modified version of LTH system from SemEval 2007 is available at http://nlp.cs.lth.se/fnlabeler/) that can complete this task, though it was not available at the beginning of this research. Our system is intended for future research into areas such entailment, novelty detection, question answering, and summarization.

Third, several different configurations for machine learning models are investigated, which can be used for role labeling and frame extraction. The application of negative training[7] instead of rule filtering is quite unique. The most effective configurations can be found in the final sections of each experimental series and in the conclusions.

[7] We use negative training to train our system to identify circumstances where a frame suggesting word does not actually mean or suggest the given frame.

2 Related Work

"Those who cannot remember the past are condemned to repeat it.",

George Santayana

Computer scientists have always been interested in language understanding. Three common methods for representing the meaning of natural language are predicate calculus (Jurafsky and Martin, 2000), frames (Minsky, 1975), and conceptual graphs (Sowa, 1984). All three methods contain overlapping motivation, techniques, and solutions; however each method focuses on a particular need or goal.

Predicate calculus has rigorous rules and structures that facilitate automated reasoning. The rigors of predicate calculus can complicate the way of expressing meaning at times, offsetting the advantages of automated reasoning. Furthermore, predicate calculus is very expressive for mathematical work, but it is not usually expressive enough for work with natural languages and frames.

Conceptual graphs are connected bipartite graphs composed of concepts and concept relations. They are equivalent to predicate calculus, but offer a less formal method of expression. Since conceptual graphs are equivalent to predicate calculus, they share the benefits of easily applying automated reasoning. Conceptual graphs have four basic manipulations: copy, restrict, join, and simplify. These operations assist in working with and reasoning about graphs, though they are not necessarily truth-preserving operations. These operations are the basis of connecting multiple graphs into a single graph.

Frames describe events, situations, or objects and then use slots (frame elements) to give details specific to the instance being described. Frames assume a general understanding of a concept with the specifics, which change from instance to instance, as slots where more information is added. Frames help store information in a hierarchy, which can useful in fields other than natural language processing, such as computer vision (Minsky, 1975). As shown by Minsky, frames can be used for reasoning, and have the benefit of assuming defaults for information that is not always present.

Frames and conceptual graphs share some common ideas with the early semantic network representations of Masterman (1961), Ceccato (1961), Shapiro (1971), and Wilks (1972). Conceptual dependency graphs (Quillian, 1967) were one of the more influential representations of the time. They defined words using a graph composed of other words with their own graph definitions.

Early conceptual dependency graphs did not have standard relations, although many attempts were made to define standard sets of relations (Masterman, 1971; Wilks, 1972; Schank and Colby, 1973; Schank and Nash-Webber, 1975). One of the most ambitious attempts to define conceptual dependencies was by Schank and Rieger (1974). Their representation was uniform for semantically equivalent expressions, but conversion to their canonical form was not computable (Woods, 1985). The foundation of their method was defining atomic relations from which all other actions could be defined. Their work helps demonstrate a method of defining actions in a fashion that enables automated reasoning.

Another useful product of conceptual dependencies were the scripts of Schank and Abelson (1977), which provide a method for assuming default information about the sequencing and requirements of events. A common example is visiting a restaurant; most adults know what to expect during a visit to a restaurant: entering, seating (either self-seating, usually indicated by a sign, or seating by a host), reviewing the menu, ordering food, eating, followed by paying for the meal. Scripts provide a method of encoding the common knowledge into something usable by computers. Scripts are another good example of how to define actions in a computer-friendly manner as they usually include pre- and post-conditions, any transitions that occur, and define sequences of assumed events.

Today, frames, conceptual graphs, and predicate logic are probably the three most-common forms of semantic representation. To successfully apply any of these representations to a task, the data must first be converted into the appropriate form. Our interest is in enhancing tasks with frame information; as such, a method for automatic extraction of frames from text (specifically individual sentences) is needed.

The task of extracting frames from plain text was previously undertaken at SemEval 2007 task 19: Frame Semantic Structure Extraction (Baker and al., 2007). Only three teams

competed and only two completed the entire task[8]. The results show that there is significant room for improvement.

CL Research, a participant in the SemEval 2007 Frame Extraction Task, (Litkowski, 2007) integrated their frame extraction system into an already existing system. Their system used rules, a lexicon and additional processing to extract frames. Our method differs in that we focused on using machine learning with the FrameNet annotated corpus.

The UTD-SRL (Bejan and Hathaway, 2007) applied SVM and Maximum Entropy to the tasks of frame disambiguation and frame element labeling. Their system used established methods for disambiguation (Florian et al., 2002). When their system faced decisions with an inadequate amount of training data (5 sentences or less) the system randomly choose a frame.

The LTH system (Johansson and Nugues, 2007) uses a dependency parser for the initial data structure, as our system does. Before doing disambiguation, their system applies a small set of hand-crafted rules to filter out words that are unlikely to evoke frames and their presence will degrade the performance of the system (in particular prepositions). Frame disambiguation is done using an SVM classifier. The authors have extended the collection of lexical units in FrameNet using WordNet. (Fellbaum, 1998). The addition of new lexical units benefited their system in the task. The frame element labeling is done using words immediately related in the dependency tree.

The task of extracting frames from text can be divided into two main tasks: frame identification and frame element labeling. Frame identification is very similar to the task of sense disambiguation with a lexicon. Frame element labeling is a form of semantic role labeling, which has recently received much attention starting with Gildea and Jurafsky's work in 2002 (Gildea and Jurafsky , 2002). Over the past few years much work has been put into refining feature sets, optimizing classifiers, and evaluating different corpuses (Marquez et al., 2008).

---

[8] The task was a challenge with two parts: frame identification and frame element labeling. All three teams entered systems for frame identification but only two had systems that could label frame elements.

Since semantic role labeling has been successful in improving the results of tasks such as question answering (Narayanan and Harabagiu, 2004), machine translation (Boas, 2002), and summarization (Melli et al., 2005), it is our hope that a frame representation, which provides a deeper semantic representation, would further enhance these tasks.

There are two primary resources used for semantic role labeling: FrameNet (Fillmore, Ruppenhofer and Baker, 2004) and PropBank (Palmer, Gildea, and Kingsbury, 2005). Gildea and Jurafsky's initial work (2002) was implemented using FrameNet. FrameNet provides an annotated corpus of exemplified frames and frame elements (semantic roles specific to given frame). FrameNet provides additional resources for working with frames, such as a frame lexicon and ontology. PropBank, which has been developed more recently, has been the major focus of semantic role labeling. PropBank provides annotated data but uses a smaller set of general semantic roles[9] (unlike the frame-specific roles of FrameNet).

Other resources commonly used in semantic role labeling are: VerbNet (Kipper, Dang, and Palmer 2000), which provides generalizations about verb behavior; WordNet (Fellbaum, 1998), which provides a lexicon of words, synsets[10] and other relations. WordNet and VerbNet can be used to extend the coverage of FrameNet and other semantic resources (Shi and Rada, 2004; Johansson and Nugues, 2007).

[9] PropBank uses general semantic roles, which are numbered arguments. Each verb in a class of verbs may have the arguments appear in different syntactic locations. If verbs of the same class have identical arguments, then the meaning of the verb should be identifical.
[10] WordNet is organized into synsets, which are sets of words that are roughly synonymous in a specific context.

3 Tools

"If the only tool you have is a hammer,

you tend to see every problem as a nail."

Abraham Maslow

A number of tools and resources where used during the course of this research. This section describes Weka (section 3.1), Machinese Semantics (section 3.3), and FrameNet (section 3.4). The majority of this section is devoted to providing information and examples for the purpose of reproducing this work.

3.1 Weka

Weka (Witten and Frank, 2005) comes in two common forms a Java library and a GUI[11] application. Weka provides dozens of machine learning algorithms including classifiers and clustering algorithms, and provides utilities to evaluate the resulting models.

Weka was used both as a library and an application to train, develop, and test our machine learning models. The automated frame extraction system uses Weka library to apply the previously mentioned models to new data.

Problems were encounters using Weka; primarily that Weka would run out of memory; 32 bit applications typically have a memory limitation of about 2GB on standard operating systems. For most of our experiments 2GB was inadequate. When running a 64 bit version of Java, on a 64 bit operating system (which additionally requires a 64 bit computer), more memory was addressable, but the test machine had only 4GB memory available.

More efficient datasets were developed over time. By breaking tasks into smaller tasks, and then removing features and removing classes that could not be present in a particular smaller model, the memory problems were overcome. These improved datasets allowed much faster training and testing (more than 10 times faster).

Only 3 classifiers were tested, J48 decision trees, Naïve Bayes and occasionally SVM, though Weka provides an extensive set of classifiers; these three are representative of 3 types of classifiers. Bayesian methods are known to be effective on text, decision tree are intuitive and human-understandable classifiers, and SVM is know to be effective on data with many features. SVM was memory- and processor-intensive, and thus costly to test. J48

[11] GUI – Graphic User Interface – an application which user can graphically interact with as oppose to strictly interacting with text.

proved to be memory-intensive, but produced good results. Naïve Bayes had significant fluctuations in accuracy; it started with poor results but later was comparable to J48. Naïve Bayes also had the advantage of running quickly and with minimal memory requirements.

More discussions, descriptions, and results of the different classifiers can be found in the experiments section.

3.2 Weka: Usage

Weka provides a graphical user interface (GUI), though most of the research was done by calling Weka from the command line. Usually tests were run in batches with the number of tests often ranging from 500 to 10000 experiments, these were easier to automate using a script from the command line.

We created a Perl script that automatically prepared data and called Weka with the appropriate parameters. Most experiments were run on a 64 bit machine with 64 bit Java to address the maximum possible memory.

To use Weka from the command line one must have weka.jar, which is freely provided on the Weka website (http://www.cs.waikato.ac.nz/ml/weka/). Calling Weka from the command line has the form presented below:

java [Virtual Machine Options] -cp path_to_weka.jar classifier_to_run [options for classifier]

3.2.1 Virtual Machine Options

The -Xmx option was used to set the upper limit of heap memory. The default maximum heap memory for Java is 256 MB.

When using 32 bit Java the limit was set to 2GB with the following command line:

-Xmx2048m

When using 64 bit Java the limit was set to 4GB (the amount of physical memory available) with the following command line:

-Xmx4096m

3.2.2 Classifier to Run

Three different classifiers were used during testing: Naïve Bayes, J48 decision trees and SVM. Each of these algorithms is described on the following pages: Naïve Bayes on page 11, Decision Tree on page 11, and SVM on page 12. Each of the algorithms respectively learns using a progressively more complex and processor-intensive algorithm. To use these classifiers in Weka from the command line one must specify their Java class. Table 1 shows which class to use for which classifier.

Classifier	Java Class
Naïve Bayes	weka.classifier.trees.NaiveBayes
J48 (Decision Trees)	weka.classifier.bayes.J48
SVM	weka.classifier.functions.SMO

Table 1 - Namespaces of Weka classifiers

3.2.3 Options for Classifier

There are a large number of options, some are specific to the classifier and some are general purpose. We used the general-purpose options to control training and testing of the classifiers. Table 2 lists the options used, and a brief description including what settings we used.

Option	Function	Used Settings
-t <file_path>	Set what file to use for training data (implies generating a model)	*.arff
-c <number>	Set which feature should be used as the target class.	first
-d <file_path>	Where to save the resulting model.	*.gz
-l <file_path>	Set which file to load the model from. This cannot be used with -t, which implies generating a model not loading.	*.gz
-x <number>	How many folds to use for cross-validation	2 or 10
-T <file_path>	Sets what file to use for testing data	*.arff

Table 2 - Options used with Weka classifiers

Our early experiments were constantly running out of memory, therefore a two-fold cross validation was used because the training sets were smaller, and thus Weka was less likely to run out of memory.

3.2.4 Arff Files

Arff files are the most common format of data used with Weka. An arff file contains two parts, a header describing the different features and classes, followed by a data section.

The header section defines the different features and their types. If a feature is nominal[12], all possible nominals must be listed; if a feature is a number, it should be specified as such. The following example shows a header section of an arff:

```
@relation Relation_Passing_Grades
@attribute PassingGrade {true,false}
@attribute Grade numeric
@attribute StudentId numeric
```

The @relation line describes the name of the data represented. @attribute lines define features; the first is called PassingGrade and is either true or false; the second is called Grade and is a numeric value; the third feature is a numeric student Id.

The data section starts with a @data line, then each subsequent line contains a value for each feature (in their defined order), separated by commas. Here is a sample data section for the header defined above:

```
@data
true,80,1234
false,45,1235
true,65,3015
```

How arff files are defined can significantly change the performance of the classifiers trained on the data. This is demonstrated in the Experiments section (section 5.1.3 and 5.1.5) and summarized below.

[12] A nominal is a feature which has predetermined set of named values

It is important to use nominals for anything with discrete classes even if the classes are numbers. Frame Ids and lexical unit Ids are numbers but should be treated as nominals because they represent discrete classes. Naïve Bayes performs significantly better under the conditions where frame and lexical unit Ids, of which there are thousands, are represented as nominal values (see section 5.1.3 *Experiment #2 - Super Model, Frame Models, and Lexical Unit Models*).

The header of the previous example may be modified to list all of the student Ids, even if there are thousands. Let "..." represent the Ids of all other students not present in this example. The resulting attribute would be:

@attribute StudentId {1234,1235,3015,...}

Removing extraneous data from the arff file header can make significant improvements to performance and efficiency. This may seem obvious upon reading but one might just as likely assume that features and classes that are never used should not affect the performance of a classifier. Furthermore, when configuring more than one model, it is simpler to define one common arff header, which includes all possible data, but this will also negatively affect performance; instead custom headers should be defined or generated for each model, which only contain definitions for data relevant within the context (for a concrete example see section 5.1.5 *Experiment #4 – Improved Arff Files and Broader Frame Element Classification*). From the previous example, all unused student Ids may be removed to produce the following attribute:

@attribute StudentId {1234,1235,3015}

For an example of one of the arff files generated from model # 2 and used in experiment #4 (section 5.1.5) please see Appendix III – Sample Arff File.

3.3 Machinese Semantics

The Machinese Semantics parser by Connexor Oy (Järvinen, 2003) produces a dependency tree enhanced with semantic information. The addition of semantic information such as the type of dependency relations, or the semantic class of nouns, should make the effort of classifying frame elements easier.

Machinese Semantics runs very quickly even on large sets of documents, and does not require an abundance of memory. The parser is excellent at identifying proper nouns, such as names and places, and some other general semantic information such as whether a noun is an animate, sentient, male/female, plant, or animal. The complete list of semantic information is quite limited, but still very useful.

The dependency relations provided by Machinese Semantics are excellent. They accurately classify the type of relationship between two words. Below is a list of some of the dependency relations defined by the parser (a complete list can be found in Appendix IV).

Subject, Object, Determiner, Modifier, Attribute, Location, Manner, Time, Instrument

The parser does suffer from one major failing: it frequently fails and provides only partial parses. About 30% of our test sentences from the FrameNet annotated corpus could not be parsed into a complete dependency trees. Most of the incomplete parses present themselves with several roots, breaking some of the implied associations.

The following three sentences, which are complex but without any serious grammatical errors, highlight some of the 50 000 sentences which the parser failed to completely parse.

> "It is repeatedly being copied and recopied as the generations go by, like the Hebrew scriptures which were ritually copied by scribes every 80 years to forestall their wearing-out."

> "What commonly seems to happen is that project work comes to consist of the accumulation of large amounts of haphazard information, often copied directly from reference books."

> "Delhi was sacked and many of the Moghul treasures were carried back to Iran, including a Peacock Throne, which was lost en route and then copied by Iranian craftsmen."

In the future, other parsers may be reviewed to replace Machinese Semantics or work in tandem with Machinese Semantics in hopes that multiple dependency trees may provide more information.

3.4 FrameNet

FrameNet is a resource that is difficult to use it to its full potential. FrameNet contains many valuable parts; using each part requires extensive work. FrameNet can also be extended using resources such as VerbNet, WordNet and PropBank[13] (Shi and Mihalcea, 2004).

FrameNet resources used for this research are:

1. Frame structural definitions
2. Annotated sentences
3. Lexical units

FrameNet resources for future use:

1. The ontology
2. Meta-relationships
3. Semantic types (SemTypes)
4. Conversion into OWL[14]

Resources that would be beneficial to have:

1. More frames
2. Frame semantic definitions, in terms of other frames

3.4.1 Frame Structural Definitions

FrameNet provides structural and semantic definitions of frames. The structural definition of a frame defines what frame elements are expected, some possible valences and whether

---

[13] Palmer, Kingsbury, and Gildea, 2005

[14] OWL (Web Ontology Language) is further defined in Appendix I. FrameNet provides resources to convert frames into OWL representations.

a particular semantic type[15] is expected. Frames contain semantic definitions that are plain English explanations of the application and meaning of a frame. There are additionally semantic definitions for each frame element.

The individual frame definitions are thorough though they express a limited vocabulary. We suspect the limited vocabulary will be onerous in many applications, but in our current applications it seem tolerable.

3.4.2 Annotated Sentences

The annotated sentences contain several errors and problems. Most issues were resolvable, while others we simply tolerated. Regardless of the problems, the annotated examples were critical to the application of machine learning to frame identification and frame element labeling.

The names of frame elements in the annotations and the names in the definitions can differ significantly; the names even differ between some annotations. Some simple problems are that the capitalization can differ between frame element names, and the names inconsistently use spaces and underscores.

The most serious problems are that sometimes the annotated frame elements do not even exist in the frame definitions. The table 3 lists 6 examples, which were not easily resolvable.

Frame	Frame Element	Problem
Apply_heat 236	Temperature	Temperature is not a frame element but Temperature_setting is a valid frame element
Apply_heat 236	Food1, Food2	There is a food frame element probably because there are 2 foods present, they were assigned Food1 and Food2.

[15] In FrameNet Semantic Types, or SemTypes, are used to specify what type of value a frame element expects. SemTypes may be specific an entity, a specific expected frame type, such as *locative_relation* or *color*.

Assistance 392	Co_agent	Assistance frame does not have Co_agent nor an Agent frame element. No resolution.
Suspiciousness 531	Entity	Entity does not exist as a frame element, but it could refer to social_actor
Activity_done_state 182	State	There is no state frame element
Inhibit_movement 245	Goal	There is no Goal frame element, but the frame element Holding_location has goal as the SemType.

Table 3 - Inconsistent FrameNet annotations

The problems listed above are infrequent but they do complicate the automated processing of the annotations in preparation for machine learning.

The annotated data in FrameNet is not ideal for training a classifier to "not evoke" frames. At times lexical units should not evoke the frames they are associated with. This kind of negative training is important for a production system, but difficult to obtain because it requires fully annotated training data. FrameNet is still evolving with new Frames being added regularly. It is possible that the addition of a new frame may make previously fully annotated data incomplete; the previously fully annotated data may contain a new lexical unit and frame, which should be labeled. Data that provides inconsistent negative examples would cause machine learning algorithms to make similar inconsistent classifications.

3.4.3 Lexical Units

Each frame contains a list of lexical units, words or expressions, which usually indicate the presence of a particular frame. Lexical units are an effective way of identifying frames. The only two considerations with using lexical units are how to extend the 10000 lexical units already in FrameNet, and how to disambiguate when a word can imply two different frames.

Extending the lexical units for FrameNet is not a new task (Shi and Mihalcea, 2004). Given the large number resources available for processing the English language, such as, WordNet, VerbNet, and Roget's Thesaurus, there is a lot of opportunity to extend FrameNet beyond its current vocabulary.

3.4.4 Ontology

The FrameNet ontology contains multiple types of relations. The subsequent sections list and describe the various relations. Each FrameNet relation defines a mapping between two frames. Many of the relations in the FrameNet ontology could be used to enhance certain NLP tasks, or even to enhance the training and extraction of frames from text. The following sub sections summaries the different relations defined in FrameNet II: Extended Theory and Practice (Ruppenhofer et al., 2006)

3.4.4.1 Inheritance

This is an inheritance relationship where the child frame is a specialization of the parent frame. In this relationship anything that is true of the parent should be true of the child. Most, if not all, of the parents' frame elements should be mapped into the child's frame elements.

3.4.4.2 Using

This is applied when a particular frame refers to actions or results defined in other more general frames. While the frame itself does not directly inherit from the frame being used, it is assumed or referred to by the frame in question. It is likely that a frame A using frame B has some common or related frame elements to frame B. In some cases, what is a single frame element in frame B may be expanded into multiple frame elements in A. For example the *Judgment_communication* frame, evoked by the word "accuse", is not quite a *statement* because it divides the message into two parts, evaluee and reason; *Judgment_communication* is not quite a *judgment* frame because it is not purely a cognitive state; it uses both the *statement* and *judgement* frames to as participants in the action.

3.4.4.3 Perspective

This is a relationship similar to a Using relationship, except that the new frame is actually a point of view of the parent frame. This is best demonstrated with the frame commercial_buy and commercial_sell which are perspectives on the same neutral frame, commercial_goods_transfer. The perspective relationship is important because it encodes subtle but important information from the human communication.

This relationship could be used to create more examples from the training data for some classifiers. By using examples from a parent or child frame and converting them to a frame of interest, we can increase the amount of training data available. For frames with very few examples, leveraging this relationship could improve the results significantly.

3.4.4.4 SubFrames

This is a relationship where one frame represents a complex idea composed of multiple smaller ideas. Some possible complex frames could represent a process, or an event. For example a criminal trial has multiple steps, which translate into sub-frames like presentation of evidence, presentation of witnesses, cross-examination, closing arguments, and the judge's decision. Another example of a complex event is a soccer game, which is composed of goals, free kicks, plays, passing, substitutions, fouls, penalty shots, red and yellow cards, all of which could be sub-frames of soccer.

Sub-frames convert from abstracted or overview frames to more specific detailed frames, and vice versa. Conversion between high-level frames (overview) and sub-frames (specifics) could be advantageous to tasks such as summarization.

3.4.4.5 Precedes

The precedes relationship is used when describing how frames or sub-frames relate to each other in a process. This type of chronological relation occurs very clearly during a process such as criminal trial or the sleep/wake cycle.

This relationship is used only in the definition of frames, not specific instances, and thus should only describe a fixed ordering of frames, as they always occur. These frame

relationships provide obvious benefits when trying to sequence events, which have implied relationships.

3.4.4.6 Causative of and *Inchoative of*

These relationships have to do with changing attributes or properties and their values. It is difficult to understand from FrameNet exactly how to use these relationships. Some examples of frames related these relationships are *Cause change of scalar position, Change position on a scale* and *Position on a scale.*

3.4.4.7 Also See

This relationship merely indicates that two frames are similar or related. While two frames may be related by the also see relationship they may not necessarily share frame elements or have a mapping.

3.4.5 Definitions

FrameNet provides human-readable definitions for each frame, and definitions of each frame element, often with a few short examples. While they are important for humans to understand each frame and how they are used, it would be beneficial to have definitions that are computer-friendly. Previous research has already demonstrated methods for representing definitions of words in a structured way that can be easily processed by computers (Quillian, 1967). These definitions could be used with automated reasoning and possibly contain pre- and post-conditions. Computer-friendly definitions for automated reasoning could be valuable asset for many NLP tasks.

Currently in FrameNet a *commercial transaction* frame is not associated with the *exchange* frame but they should be quite similar except that *commercial transaction* assumes that at least one of the items being exchanged is money (by definition). Defining words with a structured semantic representation could improve a computer's ability to process and reason about statements, facts or questions.

4 Representation

"Language is the source of misunderstandings."

Antoine de Saint-Exupery

Although the representation is based on frames, we have had to modify it somewhat to make it more suitable to the goals of this thesis. This thesis required a fully integrated representation of the meaning of text, unlike frames, which usually are viewed on an individual frame basis and not as a cohesive whole. Each frame is typically related, through frame elements, to segments of text instead of to other frames. For a truly complete representation, the representations of several sentences will need to be "stitched"[16] together into a complete representation. Thus knowledge about a person or event that is spread over several sentences, paragraphs or documents may be connected into a single representation of the person or event.

For planning and evaluation, results from the automated frame extraction system were needed. Since the system was not yet complete, we manually created a few representations for a select set of sentences. The representations were manually created with guidance from the Machinese Semantics dependency tree and the information in FrameNet in an attempt to simulate the automated process this thesis would produce.

All the manually parsed sentences are from the Document Understanding Conference 2007, summarization task. Three sets of sentences with similar meanings were selected; these sets allow evaluation of whether sentences with similar meaning would have similar representations.

The following graphical representations of semantic structure provide a goal for future comparison. The manual representations are not perfect; they demonstrate frame and frame elements that are expected in the output, some that are not expected, and some that are missing from FrameNet. Some representations include comments on difficulties, which were identified while creating these examples.

---

[16] Stitch as in to connect; in reference to stitching multiple photos together into a panoramic shot. Stitching in this sense requires some overlapping knowledge and understanding of implied relationships.

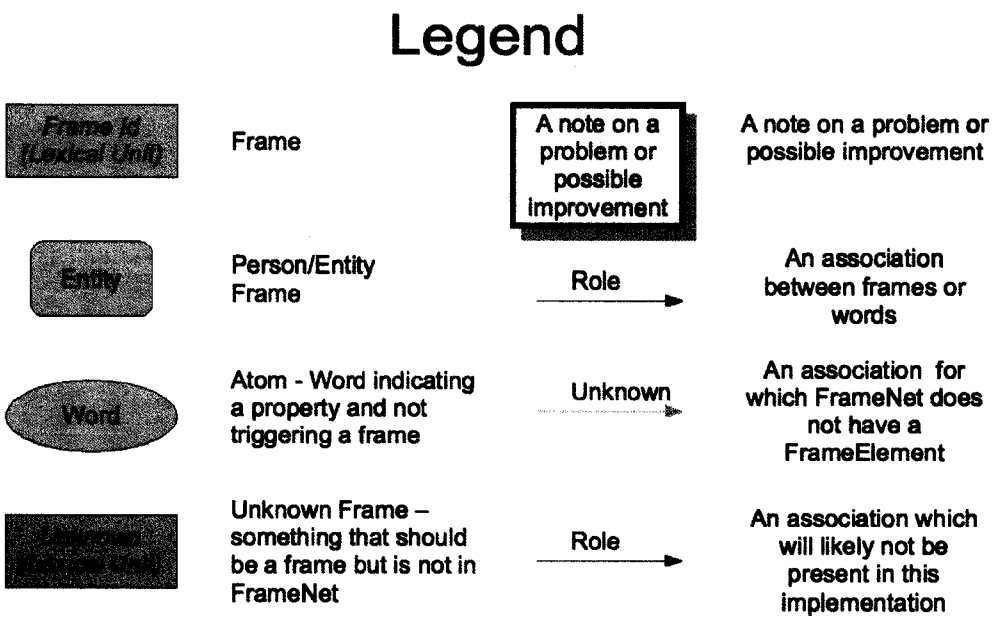
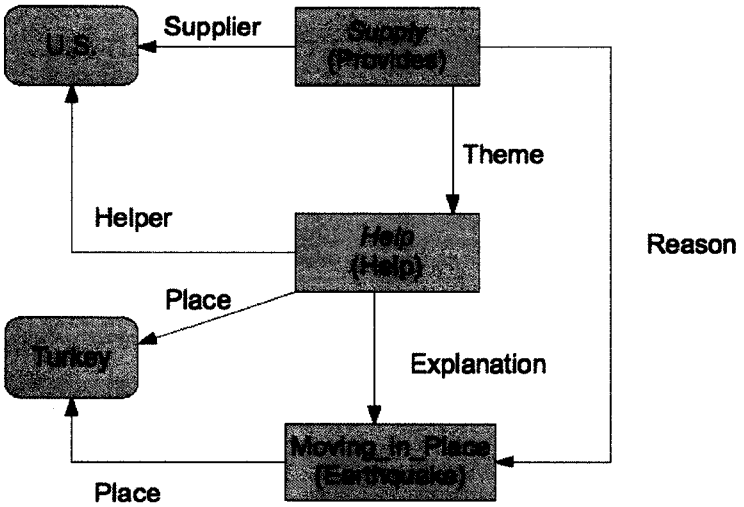


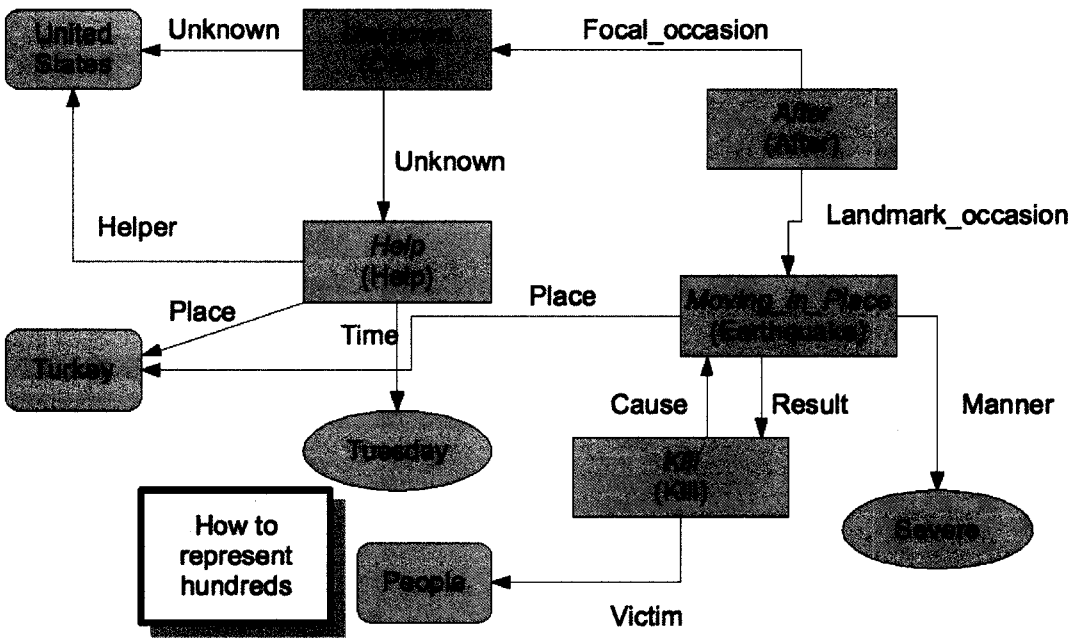
Figure 3 - Legend for manually representations

The first set of examples describes the United States of America offering assistance to Turkey in the wake of an earthquake. The first two examples show the U.S. and Turkey related through the help. The location of the earthquake is shown as Turkey. Thus, a similar core is present, but the representations are not exactly the same. In the second example, there was uncertainty in how to represent a number or group of people; section 4.6 discusses this and explains how it can be achieved using FrameNet. The last example is completely different and contains many implied associations that cannot be added except through the use of automated reasoning. Machinese Semantics had significant problems parsing the last example and thus automated final representation may not be able to assign many of the frame elements.



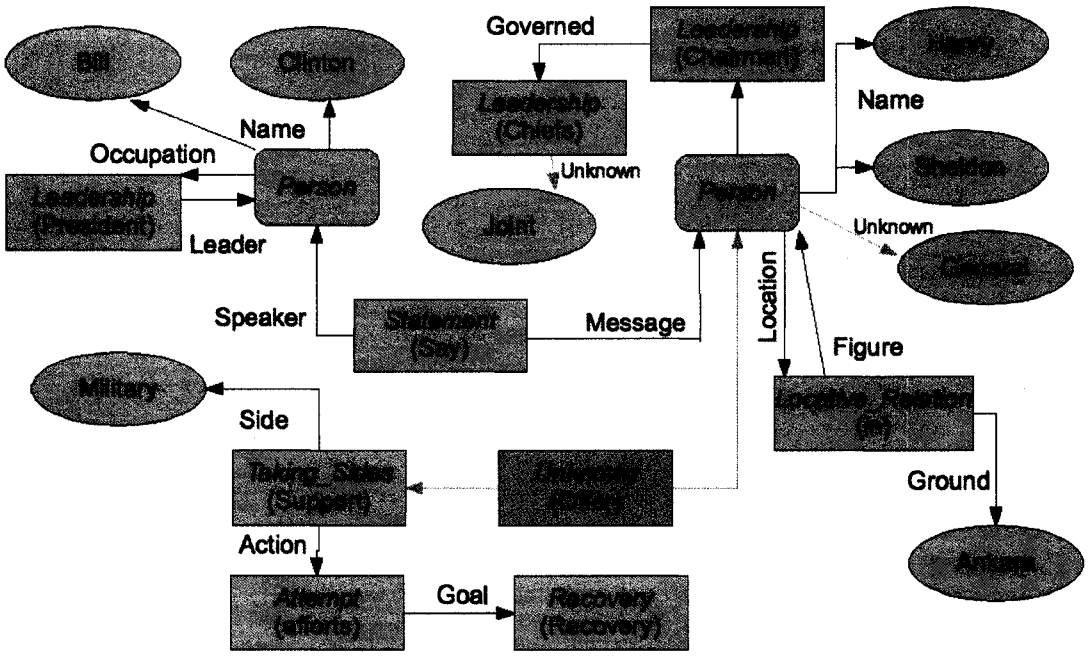
U.S. Provides Help for Turkey's Earthquake.

Figure 4 - Manual Sample Frame Parse 1



The United States offered help to Turkey on Tuesday after a severe earthquake there killed hundreds of people.

Figure 5 - Manual Sample Frame Parse 2

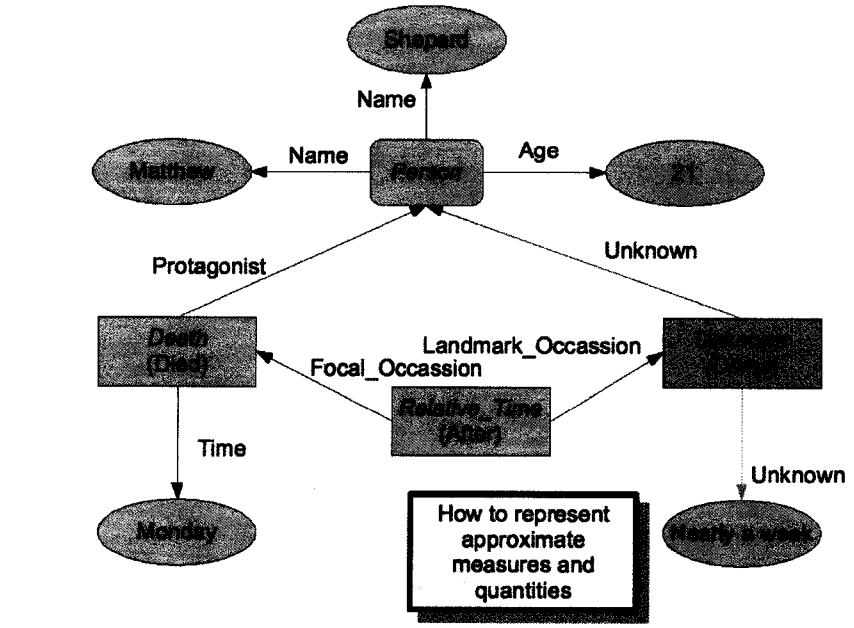


President Bill Clinton said Chairman of the Joint Chiefs of Staff Gen. Henry Shelton was in Ankara for military meetings, and he offered military support for recovery efforts.

Figure 6 - Manual Sample Frame Parse 3

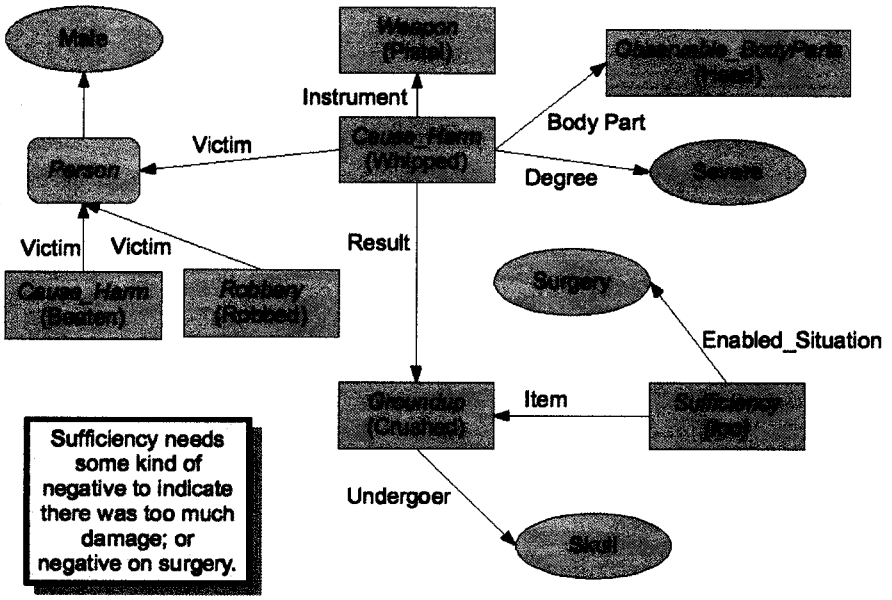
The next set contains two examples, which describe the victim of a violent crime, who died in hospital. In the first example, the representation of approximate time is not defined. This is seen when considering the statement "nearly a week", which should somehow be quantified and associated with the coma frame. Additionally, the Relative_Time frame may not be the best method of representing the passage of time and how events relate. Time lines and the ambiguous time measures will need to be improved for this representation to be complete. Section 4.3 discusses representing the progression of time.

The second example does not show significant overlap with the first even though they are about the same event. The second example shows a need to be able to negate relations to imply the opposite of the relation. This is demonstrated when "crushed" is related "sufficiency", and "surgery", for which the "item" or condition is not sufficient for surgery.



Matthew Shepard, 21, died early Monday after being in a coma for nearly a week.

Figure 7 - Manual Sample Frame Parse 4



He had been robbed, beaten and pistol-whipped in the head so severely his skull was too crushed for surgery.

Figure 8 - Manual Sample Frame Parse 5

The previous parses visually represent a reasonably effective semantic representation; they also help reveal some problems. The following sections will discuss some problems then offer some potential solutions. This thesis does not implement solutions to all the issues and requirements proposed here.

4.1 Quotations

In some circumstances a literal quote is more important than its literal meaning, because the quote maybe an allusion, metaphor or some other abstract expression. The following series of examples demonstrate statements that should not be interpreted literally:

1. The document "Automatic Extraction of a Semantic Representation of English Sentences" can be a tedious read at times.

Example 1 refers to this document by name, *Automatic Extraction of a Semantic Representation of English Sentences*, while in some circumstances semantically understanding the title is important, in this case it more important to understand the title as a name.

2. Who was it that said "Language is the source of misunderstandings"?

Example 2 inquires who is accredited with exclaiming a particular quote. In response to this circumstance the exact text should be matched and not simply the meaning of the expression.

3. "Silence. I kill you."

Example 3 is in reference to Jeff Dunham in Spark of Insanity presented in 2007. In context the statement raised a chorus of laughter. Subsequently, viewers of the presentation would repeat the quote to silence people and remind them of the humorous presentation. A literal interpretation of this quote has profound implications.

When the meaning of a quotation is represented as frames and fully integrated into a large knowledge representation structure it may be hard to identify which frames and frame elements were derived from the quote. There may be circumstances or types of processing for which finding the meaning of the text is important and other times where

the exact quotations is needed. Thus the original text and the semantic representation should be easy accessible and cross-referenced.

The ability to easily cross-reference both semantic representation and original text may have more benefits than just working with quotations. Readers sometime make the wrong assumptions about the text they are reading. Normally when they discover their misinterpretation, they re-read previous texts to correct their previous interpretation. Perhaps the same could be true of a system building a semantic representation of text. There may be circumstances where an error is identified, and previously processed information should be re-processed in a new context.

The ability to cross reference text and semantic representation should not be limited to frames but should also include frame elements. It is possible for two frames that were not derived from the quote to have frame elements that were derived from the quote.

4.2 Optional or Conditional and Future Information

A number of auxiliary verbs suggest possible futures, things that may occur, are likely or even are occurring, but not presently completed. Verbs like *will, would, could, might* or *may have* all suggest possible events, or information of which is not known whether it has happened. This type of optional or conditional information must somehow be clearly denoted as conditional or optional in a knowledge representation.

Consider a sentence like:

> Bob may paint his car red.

In this example we want to encode the possibility that the car will be red, but we do not know its current color; thus, if a semantic representation is being processed, it should be able to know that the car being red is a possibility but not a fact.

Many of the possible applications of a semantic representation will involve an evolving knowledge base which will need to be able to convert possible events into either actually occurred events (now facts), waiting (still possible) events, or not happening events (any possibility of the event has been cancelled).

One last use of this type of conditional representation could be when representing a hypothetical situation or example, which are never meant to be facts but are a part of knowledge model. Expressions like "imagine ...", "suppose ...", or "for example ..." suggest the subsequent information is not fact and should perhaps be stored as conditional.

4.3 Time

The previous issue of possible futures, pasts, or presents, already began to deal with the idea of time and an evolving representation. For any semantic representation, this is a difficult issue, but even static models (non evolving) will need to represent information in order of occurrence or relative times.

Any representation of the following sentences and their meaning should also encode the time information.

Tom previously had a cat named Tiger but now has a cat named Aslan.

The representation must make the distinction between the present cat and the previous cat. They have different names and Tom had them at different times. Likewise, in an evolving representation the present naturally becomes the past and new information should be associate with the present, while things that were in the present, should now be moved into the past.

When discussing quotations, we have seen the need to make any frame or frame element traceable to its source; likewise, each frame and frame element will need to be traceable to its source time, or what time it is relative to.

Being able to identify when information was added and in what relative time may be useful for tasks such as update summarization, where time is evolving and the knowledge base is changing. Tagging the semantic representation with time information may also help in the construction of time lines. Time lines may have many uses in many NLP tasks.

Some frames already contain some basic information about time; currently no special processing is being done to support time lines.

4.4 Metaphors and Poems

Representation of metaphors, poems and other non-literal text is a difficult area of study. This thesis makes no efforts towards a representation that supports metaphors or poems. This thesis will restrict its focus to text that can be processed literally.

4.5 Implications

Some expressions may have implied meanings, which are obvious or common-sense to humans, but computers will not be able to understand. These problems seem to be problems of reasoning about a semantic representation, and not problems of the representation itself; the representation should facilitate the use of automated reasoning algorithms. Automated reasoning can be applied once an initial representation is constructed.

The following is a simple example that demonstrates the subtlety of implied meanings:

Bob was murdered by a tall man.

It is stated that Bob was murdered, additional reasoning is required to understand that he is now dead. While this is a simple form of reasoning, which only requires understanding of the definition of murder, it should not be done during the initial creation of the representation. Ideally, any representation should provide some resources to assist in processing implied information; this information could be computer-readable definitions for the frames being used.

No automated reasoning has been implemented in this thesis. In the future, the ability to parse the definitions of words from a dictionary could be an effective way of extracting such simple implications.

A resource such as DIRT (Lin and Pantel, 2001) provides inference rules, which may be used to identify implied actions and relationships. Future work may include testing entailment using this representation a resource such as DIRT.

4.6 Conjunctions, Masses and Repeated Patterns

While simple plural forms of words are easy to represent, some conjunctions, groups, generalizations, and other repeated patterns are difficult to represent. The difficulties stem from how a group relates to claims made about the group. For example:

They all had a gun.

This could be a statement about a group of people who shared a single gun, or it could be statement that they each had a gun. A good representation should be able to support both interpretations, but should also provide some method of representing the ambiguity. Ambiguities may be removed with automated reasoning which may implement some common sense. The representation and implementation presented here does not support ambiguities.

FrameNet primarily represents multiplicity through two frames: *Cardinal_Number* and *Quantity*. *Cardinal_Number* frame allows us to associate a pattern or frame, with multiplier. *Quantity* is similar but is used for vague quantities, like a few or several, and can be used to select a subset of a larger group. Selecting a subset of a larger group can be effective for expressions like:

20 people were playing soccer but only some of them had cleats.

The previous example has a subset of players who have cleats. This example might be visually represented with the frames below:

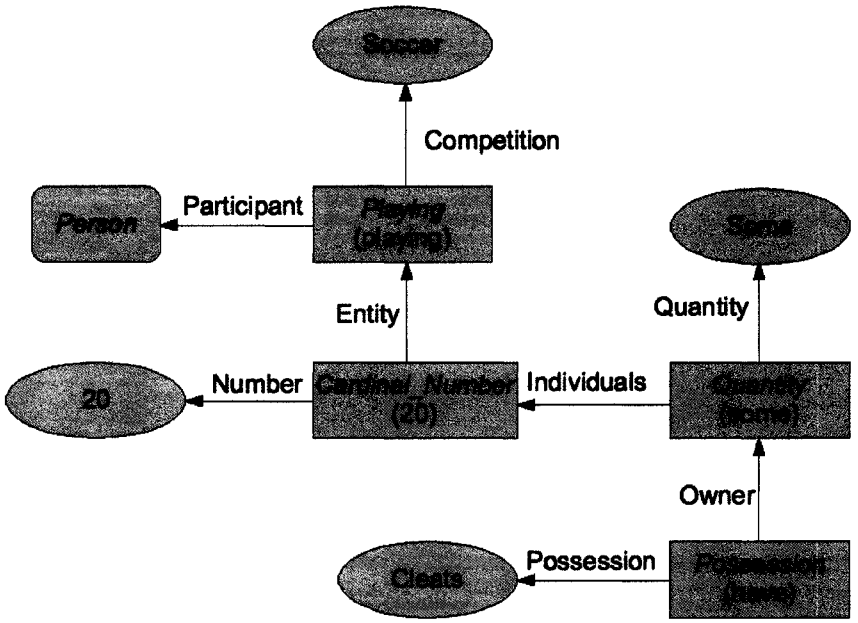


Figure 9 - Example of multiplicity and quantities*

There is still uncertainty as to whether this method of representing multiplicity is effective. The application of this representation to modern tasks will help determine its effectiveness.

4.7 Negation

Negative statements, which use words like "no" or "not", are difficult to represent because of their varying scopes. A negation may refer to a relation, a single frame, or a group of frames. How negations are represented will affect how they are processed. The effectiveness of the representation may be measured by effectiveness of the processing. No representation is proposed here because there is no method of evaluation for the representation at this time.

* To the best of my knowledge this is the method for representing quantities but this does not seem to work. In particular quantities should not be owning cleats, furthermore there should be 20 people playing in one soccer game.

5 Experiments

"Do something. If it doesn't work,
do something else. No idea is too crazy.",
Jim Hightower

Two series of the experiments were run: frame identification and frame element classification. The majority of the effort was on correctly classifying frame elements. Classification of frame elements required weeks of CPU time to run the series of experiments. Frame identification had fewer and faster iterations, but achieved reasonable results. One other work which might be considered a short experiment can be found in section 6 *Evaluation*: the evaluation of the system on SemEval 2007 frame extraction task.

5.1 Frame Element Classification

This series of experiments has the goal of correctly assigning frame elements, for an already extracted frame. This assumes that the frame has already been correctly extracted. Each experiment after experiment #0 focuses on sending data about words to a machine-learning model that then classifies whether the word is a frame element and which frame element.

5.1.1 Experiment #0

This Experiment was an attempt to build a rule-based system for extracting a frame representation. This system tried to apply the most frequent patterns found in the FrameNet annotated data to frame element identification. The rule-based system used the dependency relations and parts of speech produced by Machinese Semantics and then attempted to match those with parts of speech and grammatical information provided in the FrameNet annotated data. No formal measure was taken of its performance, since it became clear early on that it did not perform well. Upon review, this method required large manual efforts to develop and maintain the mappings between the dependency information and FrameNet's data. Furthermore, manual effort was required to tweak and develop rules for fringe cases.

Instead of using a rule-based approach we decided to use machine learning for the task of frame element identification. Machine learning had the advantage of not requiring manual effort to maintain rules, standard evaluation methods, and numerous ML algorithms could be interchanged and tested. A popular tool for machine learning tasks is Weka, which is written in Java. The rest of the system was written in Java to facilitate

working with Weka. The use of Java is advantageous because there are numerous other resources available with Java interfaces.

The baseline for frame elements classification experiments can be considered the most common frame element (theme), which produces an accuracy of 4%.

5.1.2 Experiment #1 – One Super-Classification Model

5.1.2.1 Introduction

This is the first experiment in a long series to find effective machine learning conditions, features, and models for classifying frame elements. FrameNet specifies that frame elements with the same name represent similar semantic relations. This experiment attempts to use one model to classify all frame elements for all frames. This should provide the maximum number of training examples for each frame element, while contextual information such as frame Id and lexical unit Id are features of the model.

5.1.2.2 Data

The data was collected from the FrameNet annotated data. For each annotated frame, the matching word was found in the Machinese Semantics parse tree then all the immediate children were processed to identify their frame element. At this point only immediate children of the frame word in the parse tree are assumed to be frame elements, In Section 5.1.5 *Experiment #4 – Improved Arff Files and Broader Frame Element Classification*, all relatives within two relation of the frame word were processed making almost all frame element identifiable. Further in section 5.1.7 *Experiment #6 Frame and Frame Element Location Model* the distribution of frame elements by location is examined.

The output of the ML model is the frame element name. There are about 1000 different frame elements in all of FrameNet. The input features were chosen because other researchers (Gildea and Jurafsky, 2002) were using similar features. Table 4 shows two of the features used; these features differ from other features used (table 5) because they do not describe the relation or state of either word but define the frame being used and how it was evoked.

Input	Description
Frame Id	The Id of the frame being processed
Lexical Unit Id	The Id of the lexical unit which invoked the frame

Table 4 - Frame Element Model Input Features

Table 5 contains features that are directly related to the words being tested (frame word and possible frame element word). These features were used twice, with different contexts, for each frame element. The features were used for the frame word (the word which evoked the frame) and for the frame element word currently being classified.

Input	Description
Dependency Relation	These are dependency relation defined by the Machinese Semantics parser. Some examples are main, object, condition, location, and coordination. (See the appendix for a complete list).
Part of Speech	These are basic parts of speech, such as, verb, noun, adjective, adverb, and article
Leading Preposition	This is the preposition which introduces a phrase. The preposition (in a Machinese Semantics parse) is a child of the headword for the expression.
Semantic Type	These are semantic types provided by Machinese Semantics; there are very few and we only used: human, male, female, plant, animal, and nationality.
Voice of Verb	This indicates voice of the verb (passive, active) if the word is a verb otherwise it is set to none.
Is Animate	Indicates if the word is an animate object.

Table 5 - Frame Element Model Repeated Input Features

All the features were nominal except the frame Id and the lexical unit Id. The Ids were treated as numbers because there are about 1000 frame Ids to list and 10,000 lexical unit Ids to list.

The arff file was generated using Java code, which read the annotated FrameNet files and the Machinese Semantics parses to extract the features for each possible frame element word. Each example in the arff file represents one frame word, and a frame element relationship classification. The experiment was run using Weka at the command line.

The data set contained about 232,000 examples, with 92,000 of them being negative examples (sample words, children of the frame word that were not frame elements). Only about 700 of the 1000 frame elements actually had examples. This is partially because the extraction of frame elements from the annotated text and semantic parse had problems. In particular, not all the expected relations could be found, either because the word for the frame element was not an immediate child of the frame word or because an incomplete parse separated the two words.

5.1.2.3 Experimental Setup

The experiment took more than half a day and failed because Weka was running out of memory. On a 32 bit computer, using 32 bit Java, the limited memory of any single process is about 2GB, though there are some configurations of 32 bit operating systems which support 3GB of memory per process.

The test computer was only a 32 bit system and was thus limited to 2GB of memory available. The amount of data Weka needed to process was reduced; frame elements with less than 100 examples were removed, furthermore two fold cross-validation was used to reduce the memory required by Weka during training.

5.1.2.4 Results

Naïve Bayes returned a two fold cross-validated accuracy of 32% while J48 decision tree had an accuracy of 81%. The large difference in results was surprising because in our experiences with Naïve Bayes and J48 decisions tree tend to have significantly smaller performance differences. Modeling the frame Ids and lexical unit Ids as numeric values instead of nominal values, even though they actually represent discrete classes, may have caused the performance discrepancy.

Examination of the results suggests that many of the missing frame elements were siblings of the current word. The following sentence, which has a visual parse on page 35 (Figure 4 - Manual Sample Frame Parse 1), exemplifies this problem.

The U.S. provides help for Turkey's earthquake.

In this sentence the U.S. is both the provider and the helper but *U.S.* can only be the child of *provide* or *help*, not both. In this parse it is a child of *provide* and thus for the *U.S.* to be classified as the *helper* more than immediate children will need to be processed. This issue is not addressed until experiment #3.

Notably there was not enough memory available for Weka to train on the complete data set. One solution is to use 64 bit Java on a 64 bit computer with more memory.

5.1.3 Experiment #2 – Super Model, Frame Models, and Lexical Unit Models

5.1.3.1 Introduction

Experiment 2 attempted to address two issues from the previous experiment: memory usage and poor performance of Naïve Bayes.

5.1.3.2 Data

The frame Ids and lexical unit Ids being modeled as numeric values instead of as nominals may have caused the poor performance of Naïve Bayes. The header of the arff file for this experiment contained all frame Ids and lexical units Ids listed as nominal values.

Additionally, between the Experiment #1 and Experiment #2, the Java code which extracts and aligns annotated data and the semantic parses was improved. Each of FrameNet's annotated sentences also contains the sentence in plain text. The plain text format used by FrameNet is not parsed well by Machinese Semantics. FrameNet uses spaces to divide a number of words and tokens, for example the contracted word "isn't" would actually be represented as "isn 't". Machinese Semantics did not handle these extra spaces well, which lead to may fragmented parses[17]. By removing these extra spaces, more

[17] When the Connexor Machinese Semantic Parser has trouble parsing a sentence it will create multiple parse tree fragments that are not connected.

sentences were completely parsed. This increased the number of examples in the data set to 252,000, with 737 frame elements exemplified and 99,000 negative examples.

5.1.3.3 Experimental Setup

Two efforts were made to address the memory usage issues. The experiments were run on 64 bit machine using 4GB memory. Additionally, two different machine learning configurations were used, which divided the task in smaller more manageable parts and should require less memory. The three machine learning configurations are explained in table 6.

Number	Model	Description
1	Super Model	One model is used to classify all frames, frame elements, lexical units. The one model takes all details as input features. Same as Experiment #1, including ignore frame elements with less than 100 examples.
2	Frame Model	One model per frame. For each frame a model was trained to classify its frame elements. The lexical Id is passed as nominal input feature.
3	Lexical Model	One model per lexical unit. For each lexical unit a model was trained to classify its frame elements. Neither the frame Id nor the lexical unit Id are passed in, because they are assumed.

Table 6 - Different Frame Element Model Approaches

5.1.3.4 Results

Each experiment had runtime near to or in excess of 24 hours on a four core Xeon processor running at 3.20 GHz. Testing was long with mistakes and out of memory errors adding delays.

The results are listed below, in table 7, for each model, as numbered above. Models 2 and 3 subdivided the task into more manageable parts, yet some of the smaller parts

continued to exhaust memory. J48 represents the results of the J48 decisions tree while NB, represents the results of Naïve Bayes.

The accuracies presented below are not the average accuracies across all frames, but instead the sum of correct classifications divided by total examples. The "accuracy" column represents the number of correct classifications divided by the number of classified examples (not including those that failed due running out of memory). The "overall accuracy" column represents the number of correct classifications divided by the total number of examples (including those that failed to be tested due to running out of memory).

Experiment	J48 Accuracy	J48 Overall Accuracy	NB Accuracy	NB Overall Accuracy
Model # 1	50%	-	42%	-
Model # 2	79%	58%	73%	73%
Model # 3	69%	67%	64%	64%

Table 7 - Results of Experiment # 2

5.1.3.5 Analysis

Upon examination of the results from model #2 (which is composed of about 500 sub parts), it seems that J48 consistently outperform Naïve Bayes by at least 2 or 3 percent. The difference was shown to be significant with a confidence of 0.999, using the Friedman test (Friedman, 1937) (see Appendix I).

With the changes to the arff file, Naïve Bayes results were closer to the J48 results, consistently within 10%. Naïve Bayes required less memory during training and thus was able to cover more of the frame elements. Model #2 produced the best results. In model 2, J48 produced the best results when memory was not a concern, while Naïve Bayes was almost able to process the entire dataset.

These results do not account for any frame elements that were not immediate children of the frame word.

5.1.4 Experiment #3 – Improved Arff Files and Broader Frame Element Classification

5.1.4.1 Introduction

This experiment attempted to address two issues:

- Reduction in memory usage through streamlining of the arffs files.
- Processing of all frame elements by adding all frame elements within two relations of the frame word.

5.1.4.2 Data

The data was modified in two ways: for each frame a custom arff file header was generated instead of using a common general-purpose header, and all words in the parse tree within 2 relations of a frame word were processed.

Experiment #2 used a common arff header for each frame specific arff file. This common header referred to frame elements and lexical units that were not relevant to a particular frame. By listing only the valid frame elements and lexical units for each frame, Weka's memory usage should be reduced. (Also see section 3.2.4 Arff Files on page 23)

The results of experiment #1 indicated that many frame elements were missing from the data set, and many were specifically siblings of the frame word. To describe better the relative location of the examples, more features were also added. The features are almost the same as the previous features but they include a relative location feature (described in table #9) and a set of features for any intermediate words (described in table #9).

Feature	Description
Frame Element	The frame element that this word is associated with, *None* if the word is not associated with a frame element
Lexical Unit	The Id of the lexical unit that was used to invoke this frame element
Relative Location	The location of the word relative to the frame word. Possible values are grandparent, parent, sibling, child, grandchild

Frame Word Properties	The same properties used in experiment #2, such as dependency relation and leading preposition, for the frame word.
Frame Element Word Properties	These are the properties of the word being classified as a frame element. These are the same properties as in experiment #2, such as dependency relation and leading preposition.
Intermediate Word Properties	The same properties used in experiment #2, such as dependency relation and leading preposition, for an intermediate word. Intermediate words exist for locations like grandparent or sibling where the intermediate word would be the common parent. The grandchild location would have an intermediate word, the child location.

Table 8 - Experiment #3 - Features Description

The new data set contained almost 1,000,000 examples and about 250,000 positive examples (frame element examples). All of the new frame element examples were previously unprocessed relatives of the frame word, like parents and siblings. Previously only child nodes were considered as possible frame elements. The majority of the new examples were not frame elements (about 650,000 were relatives that were not frame elements), though the number of frame elements has doubled (to 250,000). Now only 2000 examples of frame element in the FrameNet annotated data are not detectable.

5.1.4.3 Experimental Setup

The Perl script, which separates the dataset into arff files per frame, was modified to generate an arff header, which is frame-specific. The Perl script needed access to some of the FrameNet data, which was re-formatted using XSLT (see Appendix I) so that for each frame the lexical unit list and frame element list were easily accessible to the script in a tab-delimited format.

5.1.4.4 Results

The system never ran out of memory, not even when running on a 2GB system and the runtime was between one and two hours, down from about 24 hours. The reduction in memory requirements meant that both two fold cross-validation and ten fold cross-

validation could be run. Since all tests ran without failure due to insufficient memory, the separation between the *Accuracy* and *Overall Accuracy* is no longer required.

Algorithm	2 Fold Cross-Validation	10 Fold Cross-Validation
J48	87%	88%
Naïve Bayes	84%	85%

Table 9 - Experiment #3 – Cross Validated Accuracies

5.1.4.5 Analysis

As mentioned in the results, the runtime of the experiment was drastically reduced, and memory usage has become a non-issue; thus, the improvements proposed in this experiment have been a complete success.

The dataset has been significantly improved; it now covers more frame elements than before, 250,000 up from 153,000 in previous experiments. There are about 2000 frame element examples from the annotated data, which are not detectable by within 2 relations; all of the undetectable frame element examples are from sentences with incomplete parse trees. This high correlation between undetectable frame elements and incomplete parses suggests that perhaps the incomplete parses are the reason for the system being unable to detect these 2000 frame elements. The assumption that most frame elements are within 2 relations of a frame word is true.

The data set includes more frame elements than before but there has also been significant increase in the number of negative examples, from 92,000 to 750,000. Previously, less than half the data were negative examples, now about three quarters are negative examples. The dataset now lends itself to achieving an accuracy of 75% just by always assuming there is no frame element. Additionally the models were tested on the positive class, shown in table 11, which is the recall or the percentage of actual frame elements correctly labeled. Table 11 also includes the recall of Experiment #2, for comparison purposes.

Algorithm	Experiment #2	Experiment #3
J48	17%	30%
NB	27%	32%

Table 10 - Experiment #3 – Recall for Experiment #2 and for Experiment #3

Table 11 shows poor recall though there has been improvement since Experiment #2. There are several standard methods for improving the recall of a classifier. Experiment #4 considers and tests several possible methods to improve the recall of this experiment.

5.1.5 Experiment #4 – Improving Results on the Positive Class

5.1.5.1 Introduction

Experiment #3 showed the classifiers have surprisingly poor recall, which represents poor performance on the positive class (frame elements). Since this series of experiments is focused on identifying frame elements (the positive class), the low recall must be addressed. This experiment was a collection of several attempts to improve each classifier's performance on the positive class (the recall), and to provide insight into the results of experiment #3.

5.1.5.2 Data

Several different modifications to the data were tested. Oversampling was attempted; each positive example was added to the training data set twice, with the objective of increasing the weight of the positive class.

Some experiments were run using a minimal set of features to observe how this would impact recall. One experiment used only the dependency relation of the frame word, frame element word and intermediate word; another experiment used relative location and the previously listed dependency relation; yet another experiment included lexical unit Ids, relative locations, and dependency relation.

5.1.5.3 Experimental Setup

The experimental setup was similar to experiment #3, except that the Perl script, which creates arff files and runs the tests, was modified to make the changes described above: double positive class, or remove features.

Since the experiments ran so quickly and the computer did not run out of memory, we added the SVM (SMO in Weka) classifier to the testing.

5.1.5.4 Results

Using the SVM algorithm during testing increased the runtime to a few days. With so many experiments planned and SVM's significant increase to experimental times, SVM was dropped from these experiments. The preliminary SVM results did not show significant improvements over the other algorithms and thus did not justify the long run times.

	Accuracy			**Precision**[18]			**Recall**		
Test	J48	NB	SVM	J48	NB	SVM	J48	NB	SVM
Doubling Positive Examples	88%	82%	-	0%	0%	-	7%	16%	-
Using only dependency relations	85%	83%	-	82%	67%	-	45%	52%	-
Using dependency relations and location	87%	84%	86%	61%	81%	100%	39%	40%	39%
Using dependency relations, location and lexical units	87%	84%	-	100%	100%	-	36%	39%	-

Table 11 - Experiment #4 – Comparing the Overall Accuracy to Precision and Recall.

SVM did not show a significant improvement over the other algorithms, but did require almost 100 times longer to run, thus SVM was not used for intermediate testing. After features and configurations have been finalized, SVM will be included in the evaluation of classifiers.

The results were surprising: while the overall accuracy is fairly consistent, recall fluctuated greatly. Also surprising was that doubling the number of positive examples did

[18] The precision shown in this table was calculated after the experiments and is not exact. It was derived from the accuracy and recall; accuracy was cross-validated but recall was not. The equations to calculate precision from accuracy and recall can be found in Appendix I under the section Confusion Matrix.

not increase the recall; in fact it decreased recall. Perhaps the models were overfitted because the repeated data created the impression of false patterns.

The models with reduced features showed improvements to recall compared to oversampling and experiment #3. The model with only dependency relations performed best, with a significance of 0.99 over the next best results.

The results of this experiment showed that accuracy is fairly consistent or stable through various changes while recall fluctuates significantly. The results suggested that the addition of features increases the accuracy on the negative class.

5.1.6 Experiment #5 – Frame and Frame Element Location Model

5.1.6.1 Introduction

This experiment examined the effect that the location of a word relative to the frame word had on classification. The relative location might influence the understanding of a relationship between the two words. The previous experiment showed a lower recall when the location feature was added to the dataset. This experiment made the importance of the relative location feature explicit by creating one classifier per location (per frame).

Additionally, this experiment examined the distribution of the positive and negative classes across the different locations (grandparent, parent, sibling, child, grandchild). Information about the distribution of frame elements across locations assisted in understanding the classifiers' results and dataset.

With the arff files separated by location, to create separate classifiers, the intermediate word features maybe removed for the two locations (parent and child), to which they were not relevant because they did not have intermediate words.

5.1.6.2 Data

The data was the same as the previous experiment, but sub-divided into arff files specific to frame and location. Since the location was now implied by the choice of the arff file being used, the relative location feature was removed.

5.1.6.3 *Experimental Setup*

The Perl script that ran experiment #3 and experiment #4 was modified to subdivide the frame arff files into arff files per frame per location. The script also removed the values related to the intermediate word for locations that did not have intermediate words (parent and child).

The results of this experiment are presented subdivided by location; for reference the results of the previous experiment are also presented divided by location.

5.1.6.4 *Results*

The results are presented in three different tables: table 13 shows the distribution of the positive and negative examples by location; table 14 shows the accuracy of the classifiers divided by location; table 15 shows the recall of the classifiers. The table headings use the following abbreviations: GP = Grandparents, P = Parents, S = Siblings, C = Children, GC = Grandchildren

Table 13 demonstrates the distribution of positive and negative examples. This agrees with the earlier assumption that children of frame words are the most important relatives for frame elements. Additionally, the results show that 38% of frame elements are not immediate children of the frame word. The second most important location for labeling frame elements are the siblings of the frame word which account for 22% of frame elements and are 22% of the time frame elements. Parents, which do not account for many of the total number of frame elements, are frame elements 21% of the time.

Counts	GP	P	S	C	GC	Totals
All	51396	94186	275090	295333	262175	978180
Positive	1763	16323	50484	146404	10555	225529
Negative	49633	77863	224606	148929	251620	752651

Table 12 - Distribution of positive and negative examples across locations

Table 14 shows the 10-fold cross-validated accuracies of several classifiers divided by location. The first three models were trained for a specific frame, for a specific location (grandparent, parent, sibling, etc.) and they differ by what features were used. The first

used all features available, the second used only dependency relations, and the third used only the dependency relations relevant to the location (the introduction of this experiment details how parent and child relations do not require the intermediate word features, and thus do not require the intermediate dependency relation). The last two results presented are from the previous experiment but divided by location. Best results are indicated in bold.

While the results presented in table 14 are quite good, most of those values can be mostly achieved through correctly classifying the negative class. Table 15 shows the recall of the classifiers presented in table 14.

It should be noted that results in table 15 are not cross-validated and are thus trained and tested on overlapping data sets. Automatic calculation of a cross-validated recall was difficult to script and extract from Weka because this would have required using hundreds of confusion matrices with several values on each axis (of for each frame element) and scripting the working of extracting the recall. Given that these results are for analytical purposes and not for evaluation purposes these non-cross-validated results should be acceptable. Future plans already include improvements to calculate recall using cross-validation.

The overall column in tables 14 and 15 is the accuracy and recall respectively calculated by summation of examples and is not an average of results. The number of successful results for a given location can be calculated by multiplying the given percentage by the total number of results for that location (table 13). The overall percentage is calculated as the sum of the number of successful results over all locations divided by the total number of examples.

Experiment	Algorithm	GP	P	S	T	OC	Overall
All Features, subdivided by location	J48	**97%**	**93%**	**90%**	80%	**96%**	**89%**
	NB	**97%**	**93%**	86%	77%	**96%**	87%
Only dependency relations, subdivided by location	J48	**97%**	91%	89%	76%	**96%**	88%
	NB	**97%**	91%	85%	75%	**96%**	86%
Only relevant dependency relations, subdivided by location	J48	**97%**	91%	89%	76%	**96%**	87%
	NB	**97%**	91%	85%	76%	**96%**	86%
All Features (From previous experiment)	J48	**97%**	**93%**	**90%**	80%	**96%**	**89%**
	NB	**97%**	**93%**	86%	77%	**96%**	87%
Only dependency relations (From previous experiment)	J48	86%	86%	86%	**85%**	85%	85%
	NB	84%	83%	83%	83%	83%	83%

Table 13 - Overall accuracy distributed across locations

Experiment Recall	Algorithm	GP	P	S	C	Cv	Overall
All Features, subdivided by location	J48	81%	22%	38%	18%	84%	27%
	NB	73%	20%	42%	27%	81%	33%
Only dependency relations, subdivided by location	J48	**98%**	37%	46%	28%	90%	36%
	NB	78%	30%	60%	33%	87%	43%
Only relevant dependency relations, subdivided by location	J48	**98%**	38%	46%	29%	**94%**	37%
	NB	86%	38%	59%	32%	**94%**	37%
All Features (From previous experiment)	J48	81%	22%	38%	18%	84%	27%
	NB	73%	20%	40%	27%	81%	33%
Only dependency relations (From previous experiment)	J48	84%	52%	46%	40%	85%	45%
	NB	86%	**60%**	**61%**	**45%**	89%	**52%**

Table 14 - Recall distributed across locations

5.1.6.5 Analysis

Tables 14 and 15 show that the classifiers with the best accuracy do not have the best recall. In particular the classifier with the highest accuracy uses all the features, but the classifier with highest recall uses the least number of features.

Using the Friedman test on the classifiers with high recall from the current experiment, Naive Bayes with only dependency relations, and the previous experiment, Naive Bayes with only dependency relations, show a significant difference with a confidence level of more than 99%. In about 90% of the frames tested the classifier from

the previous experiment outperformed the classifier with the highest recall from this experiment.

The hypothesis of this experiment was false: dividing by location does not help accuracy or recall. Though the hypothesis is false, the distribution frame elements by locations has yielded some interesting information.

J48 classifier with only relevant dependency relations performed exceptionally on the two fringe cases, grandparent and grandchildren. The results are nearly perfect, though since these two fringe cases account for very few frame elements its application is not clear.

5.1.6.6 Final Results for Frame Extraction

These results represent the best results of all configurations: one frame element labeling model per frame with only grammatical functions as features for each word.

This was the final experiment and configuration, the results in table 16 show the performance of all three classifiers Naïve Bayes, Decision Trees, and SVM.

	Precision	Recall	F-measure
Naïve Bayes	74%	52%	61.0%
Decision Trees	82%	45%	58.1%
SVM	82%	43%	56.8%

Table 15 - Results of Naive Bayes, Decision Trees and SVM on the final frame element labeling configuration

5.2 Frame Identification

This series of experiments has the goal of classifying a word as a particular frame. Words may suggest many possible frames, and one or none of them should be evoked. This series begins by examining simple methods.

5.2.1 Experiment #1 – Matching by Frequencies

5.2.1.1 Introduction

This experiment attempts to classify frames using three simple methods based on word lookup. When a word that suggests at least one possible frame is found, one of the

following methods is applied to determine which frame should be evoked. The three methods are:

- Most frequently matching lexical unit – select the frame that is most often evoked by this word. The frequencies used are from the FrameNet annotated data.
- First matching lexical unit – select the first frame in the database that matches for this word.
- Longest matching lexical unit – if there are several matching lexical units, use the one with the most lexemes (words). If there are still multiple options for which frame to evoke, use one of the previous methods.

These three methods are simple and take advantage of the lexical units provided by FrameNet. These methods will never be able to detect when a frame should not be applied. To improve the overall results of this method we have also included some special rules to assist in the identification of the frames.

- Person – If the word is classified (by Machinese Semantics) as a human then represent it as person (which is a special type of frame).
- Location – If Machinese Semantics tags a word as a location, we mark the word with a special frame indicating it is a named location, including what type of location: city, country, continent.
- To be – If the word is the verb "be" then do not create a frame. The verb "be" is frequently used but rarely carries an associated frame, such as "Bob is a tall man". The word "is" does not evoke a frame; only Bob, tall and man need representation. FrameNet contains one frame evoked by the word "is": Perform_Role. This frame relates to an actor or actress playing a particular role in performance, such as a character in a play. The verb "to be" is extremely frequent and the application of the Perform_Role frame rare, thus no frames are ever applied to the verb "to be".

5.2.1.2 Data

The dictionary of frequencies and words evoked by frames comes directly from FrameNet's dictionary of frames. Testing was done using the annotated examples provided by FrameNet for each lexical unit.

5.2.1.3 Experimental Setup

Each of the methods described above was tested on FrameNet's annotated data. Each annotation set was compared the expected frame to the frame that each method suggested. This measured how often each method selected the correct frame (assuming one is supposed to be evoked); this was the recall of the method.

5.2.1.4 Results

All three of the methods gave results of 70% (they were distinguishable by fractions of a percent).

5.2.1.5 Analysis

A performance of 70% is acceptably high and can be considered a baseline for any future experiments since each method is fundamentally quite simple.

5.2.2 Experiment #2 – Machine Learning with POS and Dependency Relation

5.2.2.1 Introduction

The results of the previous experiment were good and to fully test their success those methods were applied to the SemEval 2007: Frame Extraction Task data (this task is discussed in more detail in the next section, Evaluation on SemEval 2007 Data, along with our final results). SemEval 2007: Frame Extraction Task was a competition where participants were provided plain texts, which their system had to annotate with frame and frame element information. The output of each system was evaluated using precision and recall of the expected frames and frame elements.

The method and system presented in this thesis were developed after the competition, but the competition data and evaluation still provide a method of evaluating the methods described here. The methods described in experiment 1 produced a frame precision and recall of about 20%. The large difference between the evaluations was probably due to the fact that frames and lexical units in the SemEval 2007 task have different frequencies than those in FrameNet. The low frame recall and precision also negatively affected the frame element labeling component of the task, which depends on

the correct frames being identified. The quality of frame identification must be improved. Machine learning will be used to help improve frame identification results.

5.2.2.2 Data

The machine learning models were trained using both the lexical unit annotated examples from FrameNet, the full annotations for FrameNet, and the training data provided for SemEval 2007. The evaluation was run using the SemEval 2007 Intro to Dublin file. The features used by the machine learning algorithm were the part of speech, the dependency relation of the frame word, and the dependency relation of the immediate children. The part of speech and dependency relation of the frame word were represented as nominal lists. For the child dependency relations, one boolean feature was added for each dependency relation, and was set to true when the relation was present. For example, there was a feature for the dependency relation "object" and its value would be either true, if present, or false, if not present.

5.2.2.3 Experimental Setup

This experiment was run using Weka but called directly using Java code. We cross-validated the training results and measured the results on one test file from the SemEval 2007 task. The classifiers tested were J48, Naive Bayes and SVM. Two slightly different feature sets were used; both sets included the part of speech and the dependency relation of the frame word, but the second set included the dependency relation of the children. The goal of the second set was to determine if knowledge about child relations would help improve the classification results.

The classifiers used may indicate that a word is a frame or that the word is not a frame (frame Id is none). To make possible the detection of frame Ids that are none, we included some training data that was fully annotated and used examples of suspect words (words that could indicate a frame) but did not evoke a frame, in the training data.

The system produced one model per lemmatized word. Machinese Semantics was used to lemmatize words. All words with the same lemma will use the same model for frame identification.

In the evaluation of this experiment we considered examples which evoke frames to be the positive class, and examples which do not evoke frames to be the negative class. Weka provided the confusion matrix from which precision and recall were derived; precision representing the percentage of words that were assigned frames and were in fact correct; recall representing what percentage of the expected frames were identified.

5.2.2.4 Results

Table 17 presents the results for all the experiments. The evaluation contains the results of the new methods for identifying frames on the "Intro to Dublin" text from SemEval 2007: Frame Extraction task because it was previously observed that the cross-validated results and the SemEval results were quite different.

Test:		Cross-Validated		SemEval 2007 (Dublin)	
	Measure:	Recall	Precision	Recall	Precision
1. Frame Word Features					
Naïve Bayes		72%	51%	32%	61%
J48		71%	50%	33%	61%
SVM		71%	52%	31%	62%
2. Including Children					
Naïve Bayes		75%	60%	33%	61%
J48		75%	62%	33%	61%
SVM		75%	62%	32%	63%

Table 16 - Precision and recall of different frame identification models

5.2.2.5 Analysis

The results show improvement on the frame identification of the SemEval 2007 data compared to our previous 20% recall and 20% precision. Recall seems particularly low on the SemEval 2007 data; Some of the low recall can be justified as poor identification of special frames used in SemEval 2007, though this only counts for a small part of the recall. A detailed analysis of the causes of our poor frame recall can be found in section 6.1.

SemEval 2007 used a few special frames; the special frames are used to identify named entities. These frames give a general classification of what the named entity represents, person or location.

Another possible cause of our poor recall is that there are lexical units in FrameNet with no or very few annotated examples. Without sufficient annotated examples our machine learning algorithms may not be able to properly learn to identify a particular frame. To solve this problem a method to provide supplementary data to these classifiers should be considered.

The addition of the child dependency relation seems to have been successful on the cross-validated data but there was little impact on the SemEval 2007 data. The results suggest there may be value in using the child dependency relation in classification, but since it did not improve SemEval 2007 performance, this is not yet conclusive. This thesis will continue to use the child dependency relations because their cross-validated results showed improvement over the cross-validated results without child dependency relations.

5.2.3 Experiment #3 – Lexical Unit Validation and Supplementary Examples

Because incorrect identification of a frame leads to incorrect frame element labeling, frame identification needs further improvement. Frame identification has significantly impact on this system's performance.

5.2.3.1 Introduction

Many lexical units in FrameNet have fewer than five training sentences; in fact some have no training examples. This experiment increases the amount of training data available for each lemma, by assuming that words that evoke similar frames will have similar properties.

Lexical units may be composed of more than one word; the frame disambiguation models are selected because a particular lemmatized word matches the headword of a lexical unit. While the headword of a lexical unit may match the rest of a lexical unit may not. The machine learning models that identify frames are given no information regarding the validity of each lexical unit. This experiment introduces a validation for each lexical unit being considered by a machine learning model.

5.2.3.2 Data

As in the previous experiment there are arff files and models for each lemma. Each lexical unit evokes a particular frame and that the lexical unit may have a small set of examples; to increase the number of training examples for the given lexical unit, all examples from all lexical units that evoke the same frame are pooled together. This should have the effect of making the frame identification models, which are per lemma, into models which disambiguate between frames and between lexical units. This is based on the assumption that examples that evoke the same frame will have overlapping features.

In this experiment each example contains a few new features. Each example contains a boolean list of frames that might be evoked by the lemma. The value of each boolean indicates if at least one lexical unit associated with the frame is valid in this particular case.

5.2.3.3 Experimental Setup

The experimental setup is identical to the one used in the previous experiment 5.2.2, with the exception of the data changes described above in section 5.2.3.2.

5.2.3.4 Results

Table 18 shows the ten-fold cross-validated precision, recall and F-measure for the new models. The results on the SemEval task 19 data showed significant improvement in precision using these models, a 5% or more improvement on all texts. Table 19 contrasts the old results on the SemEval data with the new results using J48. When tested on the SemEval data, Dublin text, J48 performed the best.

	Precision	Recall	F-measure
NB	96%	99%	97.5%
J48	96%	99%	97.5%
SVM	97%	99%	98%

Table 17 - Precision, Recall and F-measure of Frame Identification with Lexical Unit Validation

Text	Text	Precision	Recall	F-measure
Dublin	Previous	0.56250	0.22685	0.32331
	Current	0.63507	0.22027	0.32710
China	Previous	0.53837	0.26428	0.35452
	Current	0.56323	0.26245	0.35806
Work	Previous	0.61512	0.26519	0.37060
	Current	0.71053	0.28000	0.40170

Table 18 - Improved SemEval 2007 results with new Frame Identification

5.2.3.5 Analysis

This modification significantly improved precision and recall when cross -validated. The SemEval results showed significant overall improvement with increased precision while the recall showed no significant improvement. This change offered significantly improved results but further efforts may still focus on improving the recall on the SemEval 2007 task 19 data.

6 Evaluation on SemEval 2007 Data

"You always pass failure on the way to success."

Mickey Rooney

The SemEval 2007 task 19 was to extract frames and identify frame elements from plain text. Three texts were used during the evaluation, and Perl scripts were used to compare participants' results with the gold standard. The expected output format was similar to FrameNet's annotation format.

FrameNet's annotation format has a few significant differences from our systems output. The primary difference is that our output is a graph of all frames and how they interact with each other, while FrameNet labels each frame independently of other frames; thus each frame element is a selection of text, which may or may not contain frames.

There were numerous challenges for this system's output to match the format required by the task; many of the challenges were successfully overcome, but a few were not completed; these challenges were directly related to the differences in representation.

The following *Challenges* section will outline many of the problems encountered during evaluation of this system on the SemEval 2007 data. The subsequent *Results* section, briefly discusses the results and their significance.

6.1 Challenges

There were numerous challenges and difficulties to correctly evaluate this system on the SemEval 2007 data. The following list highlights each the issues and subsequently a brief section is dedicated to discussing each issue and the resolution.

- Use of names instead of Ids for frames and frame elements
- Mismatching frames and frame element names
- Evaluation script contained a divide by zero error
- Differences in tokenization methods, thus comparison of tokens sometimes failed.
- Differences in selection of frame element boundaries

6.1.1 Use of Names Instead of Ids for Frames and Frame Elements

Section 3.4.2 - Annotated Sentences contains a list of some of the inconsistencies in FrameNet. In particular the capitalization of any particular name can change between the definitions and the usages.

The system described in this thesis resolved some of the frame naming and frame element naming inconsistencies by normalizing the names: all frames were stored and processed in lowercase. Since this system stores names in lowercase, all names were output in lowercase. Furthermore even if names could be output with capitalization, as listed in the frame or frame element definitions, there was no guarantee that the same capitalization was used in the gold standard.

Since the annotated data already used names instead of Ids and outputting identical capitalization was questionable, the evaluation script was modified to compare frame and frame element names in a case insensitive manner. Should this have been during the original challenge, this option would not have been available, but since the goal was to measure the quality of this system, the adjustment would make the results more representative and accurate of the systems quality. The correct best solution would have been if the gold standard and evaluation script uses Ids (which are numeric) instead of names so that there were no possible inconsistencies.

6.1.2 Mismatching Frame and Frame Element Names

This issue is discussed in section *3.4.2* - Annotated Sentences where frames with incorrectly matching frame element names are listed. This system compensates for the issue by applying manually encoded correction when appropriate; otherwise this system ignores annotations that cannot be corrected. A common inconsistency involves whether to use spaces or underscores in names; this system uses strictly spaces.

The gold standard was not tested for issues of mismatching names, because the data contained new frames and frame elements, which were not in previous releases of FrameNet, which would have been detected as mismatches. It would have been hard to detect if a name was a mismatch or new frame.

The system outputs names as they have been internally normalized and allows the evaluation script process them as such. This means there may be some misrepresentation of this system's quality; to adjust for this issue, one of the texts has been manually evaluated.

6.1.3 Evaluation Script Contained a Divide by Zero Error

The evaluation script contained a divide by zero error when there were no frames or frame elements to recall. The precision and recall are calculated both per sentence and for the overall text; thus if a single sentence caused this divide by zero error then the entire evaluation would fail. To correct this issue the evaluation script was modified, not to divide by zero, but to give a default value for precision and recall in these circumstances.

6.1.4 Differences in Tokenization Methods

Comparison between participants' output and gold standard results was done through comparison of the tokens. If the tokens do not match the gold standard tokens then the script does not even compare frames and frame elements. This systems tokenization was done using Machinese Semantics, and thus its tokens had to be mapped to the gold standard tokenization.

A module was created which could consistently reproduce the tokenization required for SemEval 2007 frame extraction task (and coincidently FrameNet); a separate module used the results from both Machinese Semantics and SemEval 2007 tokenizations and mapped the resulting tokens from one format to another; this results in the occasional errors. For example, possessive words with an "'s" are consider two words in the FrameNet parse (the owner and the apostrophe s) while Machinese Semantics treats them as one word. Any text that is labeled with the "'s" is considered incorrectly labeled by the SemEval gold standard.

Another difficultly was that Machinese Semantics tends to combine multiple words into a single token, usually expressions or verb chains. To map between both tokenization methods required separating the compound tokens into smaller tokens that could be matched with SemEval 2007's tokens.

Ultimately the automatic mapping between the two tokenization methods was highly successful but initially required hours of refining. There are occasionally errors caused by the differences in tokenization, but these issues are infrequent.

6.1.5 Differences in Selection of Frame Element Boundaries

The problem of differences in frame element selection boundaries stems from the differences in representations. The SemEval 2007 task represented each frame as independent of other frames, with frame elements referring to sequence of text. The representation used in this thesis models all frames in one graph. The frame elements in this thesis represent associations between two nodes (usually frames). Frame elements from this representation refer to only one word for each frame element, the headword or frame evoking word of the frame element.

This system outputs the headword for each frame or frame element being labeled. In many circumstances these frames and frame elements match the gold standard because most often frames and frame elements are only a single word, but any circumstances requiring more than one word will fail. An example that fails is "The Irish", which this system outputs as "Irish", while others scenarios require complete selection of noun phrases or verb phrases.

While the SemEval 2007 task 19 is being used to evaluate the system, optimizing for this task was not the goal. Several improvements could have been made to maximize results on the task. The results should be considered a guide to the effectiveness of the system. Because of the many difficulties involved in the evaluation, a manual evaluation was done. The manual evaluation was meant to correct for issues of frame element boundary selection.

6.2 Results

The results are considered in two parts: first how the system performs on the SemEval 2007 task through the use of the evaluation script; second how well the system performs when the results are manually evaluated on the "Intro to Dublin".

6.3 Evaluation by Script

Table 20 represents the results of this system and participants' systems (Baker et al., 2007) in SemEval 2007 on the frame identification task. Both precision and recall are a little lower than other participants' results. It should be noted that poor performance in the

frame identification causes the results of frame element labeling to suffer, because frame element labeling is dependent on selecting the correct frame. This system is listed as the University of Ottawa.

Text	System	Recall	Precision	F-Measure
Dublin	University of Ottawa	0.3238	0.6211	0.4257
	UTD-SRL	0.4188	0.7716	0.5430
	LTH	0.5184	0.7156	0.6012
	CLR	0.3984	0.6469	0.4931
China	University of Ottawa	0.4261	0.6401	0.5116
	UTD-SRL	0.5498	0.8009	0.6457
	LTH	0.6261	0.7731	0.6918
	CLR	0.4621	0.6302	0.5332
Work	University of Ottawa	0.4132	0.7336	0.5286
	UTD-SRL	0.5251	0.8382	0.6457
	LTH	0.6606	0.8642	0.7488
	CLR	0.5054	0.7452	0.6023

Table 19 - SemEval 2007 Results - Frame Identification

Table 21 shows the results for combined frame identification and the frame element labeling. When considered in light of this system weaker frame identification performance, the results suggest that this system excels at frame element labeling. This system generally has the lowest recall but is superior in precision. The results posted in table 21 are using J48 models.

Text	System	Recall	Precision	F-Measure
Dublin	University of Ottawa	0.22027	0.63507	0.32710
	UTD-SRL	0.26238	0.53432	0.35194
	LTH	0.36345	0.54857	0.43722
China	University of Ottawa	0.26245	0.56323	0.35806
	UTD-SRL	0.31489	0.53145	0.39546

	LTH	0.40995	0.57410	0.47833
Work	University of Ottawa	0.28000	0.71053	0.40170
	UTD-SRL	0.30641	0.61842	0.40978
	LTH	0.45970	0.67352	0.54644

Table 20 - SemEval 2007 Results - Frame and Frame Element identification

6.4 Manual Evaluation

The manual evaluation on the SemEval 2007 data was done on for the "Intro to Dublin" text. The quality of the system and problems with the system were better understood after the manual evaluation, which involved individually examining each incorrect classification. The most consistent problem with the system was the misidentification of frames.

While this system identifies people and locations effectively, it does not identify all of the sub-classifications used in the SemEval 2007 data. It would be beneficial to add support for identification of other special classes that FrameNet and SemEval support, such as, dates.

The manual evaluation corrected for labels considered incorrect due to minor labeling differences, such as minor frame element boundaries. The manual evaluation of the system produced corrected values which increased the precision by 3%, and increased the recall by 1.5%.

7 Conclusions

"Be yourself and think for yourself; and while your conclusions may not be infallible, they will be nearer right than the conclusions forced upon you."

Elbert Hubbard

The automated frame extraction system presented here can be considered near the cutting edge when cross validated and evaluated on the SemEval 2007 data. The system will be used as a foundation for research in developing algorithms that leverage this semantic representation.

Numerous configurations for machine learning models have been tested; generally the application of different algorithms (NB, J48, and SVM) has little impact on the overall results, even when applied to independent tests, such as SemEval 2007 as opposed to cross validation. The most successful machine learning configurations were one model per frame for frame element labeling. The features used for frame identification were effective when cross-validated and precision on the SemEval data.

The cross-validated results differed significantly from the SemEval 2007 results. Cross-validation creates independent sets of data from a pool of data, but usually the test sets have similar distributions and properties as the training sets; instead future testing should be cross-validated using hold out topics or domains (such as the SemEval 2007 files or parts of the fully annotated FrameNet texts). The tests will be more representative of the system's behavior on data with different properties and distributions that the original training data.

While this system clearly has room for improvement, the foundation is solid enough to begin building applications and processes which use it. A clearer understanding of the requirements of the system and weakness should be developed as the system is applied to new tasks.

8 Future work

"The best way to predict the future is to invent it."

Alan Kay

Many improvements to the system and the resulting automatically extracted representation may be applied to several tasks. Tasks that may benefit from application of such a representation are: summarization, novelty detection, entailment, and question answering. Entailment and novelty detection provided the original motivation for the system. If frames (as defined by FrameNet) are a fairly uniform representation of meaning then tasks such as entailment and novelty detection should benefit greatly from this representation. There are several resources that could assist in these tasks, such as the FrameNet ontology.

Before doing a task such as novelty detection, it would be beneficial if the system could connect, "stitch", multiple sentences into a complete model. Stitching can in part be done through a simple method like connecting common elements, named entities, specific nouns, and references together. More complex relationships that may be implied must be resolved through the application of automated reasoning. The simplest forms of stitches would be connecting references to a specific named entity, connecting a pronoun with the referenced noun, connecting nouns with specific determiners, like "the", or connecting any generalized claims on the same topic, such as "cats have four legs" and "cats have claws".

Future work is also needed to extend FrameNet's support of the English language, increasing the number of frames and lexical units, and increasing the number of mappings between frames. As previously mentioned, there has been some research done on integrating numerous resources to extend FrameNet's vocabulary and coverage of the English language. Additional work may focus on building a system that reads a dictionary and automatically extracts new frames or frame relations from the definitions.

There are extensive improvements that could be made to our frame identification and frame element labeling system. Frame element labeling could be improved through the use of semantic types (semtypes) provided by FrameNet. This could improve frame element labeling. A more difficult goal, which could improve frame element labeling, would be applying a more complete approach to labeling. Using one model to label all frame elements at the same time. As other research has shown this requires machine learning

more advanced than classifiers, methods like Conditional Random Fields and Hidden Markov Models are required.

If FrameNet's coverage is extended using resources such as VerbNet or WordNet, new models would be needed for disambiguation of new lexical units. New lexical units from these resources would likely not contain any training examples, thus disambiguation would need examples that might be supplemented through examples of the same frames from different lexical units.

Appendix I – Glossary

Accuracy

A measure of success or correctness where the number of correct answers is divided by the total number of answers.

Accuracy = True Positive + True Negative / Total

Total = True Positive + True Negative + False Positive + False Negative

Compiler

A program that converts human readable code into instructions, which a computer can execute.

Cross-Validation

A method of testing used often in machine learning. The technique consists of dividing the training data in N folds of equal size. For each of the N folds, train the machine learning model on all the other folds and then test the system on the current fold. This allows for independence between the training and testing sets during evaluation when only one set was originally available.

Confusion Matrix

A way of representing the evaluation of classification problem. Two classes are required: a positive class (class of interest), a negative class (irrelevant class). The confusion matrix is a simple 2x2 matrix, which represents the classes add shows the number of correct (true) or incorrect (false) classifications.

	Actually Positive	Actually Negative
Classified Positive	True Positive	False Positive
Classified Negative	False Negative	True Negative

The equations presented below show how to obtain the confusion matrix from the accuracy and recall. These equations can easily be applied to the results from tables 13, 14 and 15

Calculating Confusion Matrix:

TruePositive = Recall * ActuallyPositive	(From table 15 and table 13)
TrueNegative = Accuracy * Total - TruePositive	(Table 14 and table 13)
FalsePositive = ActuallyNegative - TrueNegative	(Table 13)
FalseNegative = ActuallyPositive - TruePositive	(Table 13)

F-measure

A single measure which represents the balance of both recall and precision. The F-measure presented here is the harmonic F-measure where Precision and Recall carry the same weight.

F-Measure = 2 * (Precision * Recall) / (Precision + Recall)

Frame

A representation of an object or event in parametized way. See section 1.4.1 Frames for more details.

Friedman Test

A test used to measure whether a difference in statistical results is significant. It functions similar to ANOVA but without the parametric assumptions. Like ANOVA, the Friedman test is effective on multiple sampling and does not rely on the mean or standard deviation to summarize the distribution. (Friedman, 1937)

Summary of method:

Let N be the number of data sets

Let M be the number of algorithms being tested

1. For each data set rank each algorithm for 1 to M where 1 performed best and M worst. If any algorithms results are equal, give them all the average of the appropriate rankings.

2. Calculate the average ranking for each algorithm.

Let i be each of the data sets

Let j be each of the algorithms

$R_j = 1 / N * Sum(ranking_{ij})$

3. Compare the results using the Chi square

$X^2_F = 12 * N / (M * (M + 1)) * (Sum(R_j^2) - M * (M +1)^2 / 4)$

Lemma, Lemmatization

Lemmatization is the process finding the lemma; the root form of a word.

Lexeme

A word, which is part of a lexical unit.

Lexical Unit

A word or expression which may evoke a particular frame. Lexical units are made of one or more lexemes.

Named Entities

A proper noun referring to an entity (usually person or location). For example: Martin or George Bush.

Ontology

A hierarchal relationship between elements. The most common example would be the taxonomy of living things.

OWL (Web Ontology Language)

A set of standard languages for defining ontologies. OWLs are often used in logic processing. There are automated reasoning tools available for data represented in OWL form.

Perl (Practical Extraction and Report Language)

A programming language used primarily for processing text.

Precision

A measure similar to accuracy but tests how often a "yes" is a correct yes. The term is usually used in information retrieval and is defined as:

Precision = True Positives / (True Positives + False Positives)

Recall

A measure similar to accuracy, in terms of search, it represents how many of the relevant or interesting results were returned. The term is usually used in information retrieval and is defined as:

Recall = True Positives / (True Positives + False Negatives)

Role Labeling

The operation of assigning roles (descriptions) to a verb, or frame. Roles may be general, such as subject, object, and modifier, or specific frame elements, such as buyer, instrument, and containing_event.

XML (Extensible Markup Language)

XML is a meta-language or a specification for defining new markup languages. XML has rigid requirements, which make easier for automated systems to process. There are several tools for working with XML.

XSLT (Extensible Stylesheet Language Transform)

A special XML language used for transforming XML from one format to another. Often XSLT is used to transform a particular XML format into another XML format.

Appendix II – Implementation

The purpose of this section is to provide some limited organizational information for researchers considering improving or altering the system. The system is primarily written in Java, though frame element labeling arff files are generated using a Perl script and a master feature file. The master feature file is generated using Java code, which reads the FrameNet database and the annotated examples.

8.1 fe.2.1.arff

fe.2.1.arff is the arff file created during experiment #3 in section 5.1.5. It contains all examples of frame elements from the FrameNet annotated data. This file contains examples of all words within 2 relations of the frame word. This file is subdivided for all frame element experiments by the superScript.pl Perl script, which runs all the experiments. This file contains about 1000000 training examples, as described in section 5.1.5. The file contains all features described during in Experiment #3, #4, #5 (sections 5.1.5 – 5.1.7).

The creation of this file was a lengthy and semi-manual process with multiple steps. The process may soon be replaced by a fully automated process built into framer.jar.

8.2 superScript.pl

The frame element models are created and cross-validated using "superScript.pl", a Perl script which calls weka.jar. Before using the script, two variables in the script must be set: $weka_path and $java_call. Both can be found near the beginning of script. $weka_path must point to the location of the weka.jar file and $java_call must call the desired java runtime environment, and any parameters. The current settings are listed below:

```
my $weka_path = "~/dev/weka/weka.jar";
my $java_call = "/usr/local/java/64-bit/jdk1.6.0/bin/java -Xmx4096m";
```

Take note that in this example $java_call is calling a 64 bit version of java and setting the runtime environment memory to 4GB. These settings are valid from the mscai056 account on tamalesx1 at the University of Ottawa.

When calling the script to run experiments or generate frame element labeling models, the script takes one parameter, which is the arff file to use. The script runs 3 major steps:

1. The system reads the frames_lex.csv file, which contains of list of valid lexical units and frame elements for each frame. The file was generated using an XSLT. For each frame the file contains three lines, each line starts with an identifier, followed by a comma and then the associated comma delimited data. Below is an excerpt from the file. The first line indicates that this data is for frame Id 6. The second line lists all valid lexical units for frame Id 6. The final line, which is line wrapped on this page, lists all the valid frame elements. The data from this file is used to generate the custom headers for each arff.

```
Frame,6
LexicalUnits,5,4630,10151,10152,13211,13212,13270,13272,13326
FrameElement,Communicator,Addressee,Message,Topic,Medium,Amount_of_informatio
n,Depictive,Manner,Means,Time,Duration
```

2. The system then creates an arff file for each frame exemplified in the training data. Each example in the master arff file, which is referenced as input parameter to the script (usually fe.2.1.arff), is processed. Processing of each example involves:
   a. If the frame associated with the example has not yet been processed create a custom arff header. The custom header contains only the lexical units and frame elements for the current frame (these were extracted from frames_lex.csv) and all features and features values described in the experimental section.
   b. The frame Id is removed the example and the data is appended to arff file for this frame.
3. The system processes all of the frame arff files: modifying features set (usually reducing features), creating models, cross validating, and calculating recall for Naïve Bayes, J48 and SVM. This part of the processing is done in the "ProcessArffHash" subroutine. The subroutine is customized for each experiment. There are a number of helpful functions called from this routine:

a. CreateReducedFeatureArff, which will create a new arff file with a reduced set of features
b. BuildClassifier, which will build and cross validate a classifier
c. CountArffFile, which counts the number of examples in the arff file
d. BuildTestFile, which creates an arff file containing only positive examples
e. RunTest, which runs constructed models on a specific test file

All models and arff files are outputted to the "frame.arffs" directory. This directory can be changed by changing the $directory variable in the script.

Most of this process will be re-written into framer.jar. The new implementation will support proper cross-validated recall and will be more customizable.

8.3 Framer.jar

The main body of the work is written in Java and contained in framer.jar. The framer project is divided in several different modules for specific functions. They are as follows:

msem – for processing, representing, and abstracting the xml output files from Machinese Semantics.

fnet – for processing, and accessing FrameNet data including the frame definitions, annotated examples and SemEval 2007 formats.

utils – contains generic processing utilities like the configuration manager and logging class.

semeval – contains classes for exporting to the Semeval format and utilities. Some of the classes in this namespace are obsolete.

framer – contains functionality for extracting the frame representation. It uses weka.jar, fnet and msem. It contains classifiers, hard coded rules and a basic frame representation.

run – contains classes used to run certain functions of framer.jar. It contains classes for generating frame identification arffs, generating models from arffs and running SemEval 2007 experiments.

wnet – contains an abandoned attempt using WordNet to enhance frame element labeling. The code provides examples of how to interface with WordNet through jaws-bin-1.0.jar.

fnet.semtye – contains more code from the abandoned attempt to use WordNet to identify semtypes (semantic types).

8.4 utils

Module is a collection a few simple common functions that are used by other modules. Any text in this `font`, represents actual class names or text from the code. The module contains only four classes:

1. `BaseSaxInterface` – SAX (Simple API for XML) is one of two standard methods for parsing XML. This implementation of SaxInterface converts the `characters` method to return a `String` instead of character array and adds a method to assist in accessing the attributes of XML elements.
2. `ConfigurationManager` – This class is a singleton, which provides access to name-value pair stored in an XML configuration file. Framer.jar stores configuration values in "config.xml". These value pairs allow some customization of path and features on each computer that the system is used.
3. `Logger` – This class is singleton, which provides an interface for logging errors, warning and messages from different sub-systems. The logged data is stored in an XML file for later examination. The purpose of this class if to keep all logged information organized running through a single API.
4. `Utils` – this class only contains static methods, which are simple helper functions that are used in multiple files through the code. Most of the functions manipulate strings or collections.

8.5 msem

The msem module is used to parse, represent and abstract the output from Machinese Semantics. To parse the output of the semantic parser the `msem.SaxInterface` class is used. An example is given below of how to parse a file:

```
import java.io.*;

File msemFile = new File("");
msem.SaxInterface msemSax = new msem.SaxInterface();

SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating( false );

try
{
        SAXParser saxParser = factory.newSAXParser();
        saxParser.parse( msemFile, msemSax );

        msem.Sentence sentence = msemSax.getSentence( 0 );
}
catch( Throwable err )
{
}
```

The first line should be modified to create a new file, which refers the semantic parse output file. Most of the code above configures the XML parser. The actual parsing is done by the "`saxParser.parse(...)`" call. Each sentence represented in the original file can be accessed from the `msemSax` object, which is a `msem.SaxInterface`, by calling `getSentence( i )` where "`i`" is the zero based index of the sentence in question. The number of sentences can be accessed using `countSentences`.

The output of Machinese Semantics is a very deep XML tree with two node types; the document is loaded with the exact same document tree and is composed of the `FSNode` and `ValueNode` classes. Because of the depth and layout of the XML document, the nodes are actually quite difficult to work with; a more practical abstraction layer was created that can access all the needed information but through a cleaner and simpler interface.

Each sentence in the document is represented with the `Sentence` class, which provides sentences level methods and data. Each word in the document is represented using the `Word` class. In the original format words and their associated data are spread over

several nodes under different branches, the `Word` class provides a standard interface for working with these words.

8.6 fnet

The fnet module contains all methods and classes required to access and work with the FrameNet database, which is stored in several XML files. The main interface to FrameNet data is the `FrameDatabase`, which can be used as a singleton or an instantiated class.

The `FrameDatabase` class is capable of loading the main frames.xml and representing most of the contained data. The class also supports saving the data, though not all the information from the original file is saved. For this reason the saved data is stored in a separate file. The SemEval 2007 task 19 data included frames not in FrameNet 1.3, thus the database was updated and saved with the new data.

The frame database supports access to frame definitions through a number of methods, these methods return a `Frame` class which contains the properties and definition of the frame. The frame element for a given frame can be accessed through the `Frame` class and returned as the `FrameElement` class. LexicalUnits are accessible through the `FrameDatabase` and are returned using the `LexicalUnit` class.

The `HackFixer` class encodes all known errors and corrections to the FrameNet annotated data, as described in section 3.4.2.

The `AnnotationSaxInterface` can parse the FrameNet annotated data, annotated lexical unit examples, and the SemEval 2007 task 19 data. The data is stored in `AnnotationSentence` instances and `AnnotationSet` instances. The `AnnotationSentence` contains information about the sentence while the `AnnotationSet` contains information about an individual frame and its annotated frame elements. An example of these classes being used can be found in `framer.BuildCompositeClassifier`.

`AnnoSaxInteface` and `processMsemXML` classes are obsolete and should not be used.

8.7 framer

The `framer` module contains the main implementation of the system. The module depends on `utils`, `msem`, `fnet` and weka.jar. The main class in this module is `FrameBuilder`, which given a sentence parsed by Machinese Semantics and represented with `msem.Sentence` can automatically identify frames and label frame elements. The `FrameBuilder` also provides access to the finally constructed frame representation. To see an example of the class being used see `run.CreateFrames.main`.

The `FrameBuilder` manages and abstracts the identification of frame and labeling of frames elements at a high level, each individual word is actually processed by the `FrameComposer` class. The `FrameComposer` class abstracts all the logic involved in determining if a word is a frame and if so what are its frame elements. Furthermore, the `FrameComposer` creates and connects all of frame and frame elements. The actual frame instances are represented with the `FrameInst` class.

The `Classifier` interface represents an abstract frame element classifier. There are currently 3 implementations `Classifier_v1`, `Classifier_v2`, and `Classifier_v3`; only `Classifier_v3` is now used. Given two words the classifiers loads the ML model associated with the frame and can determine what frame element should exist between the words. The classifiers are called from `FrameComposer`.

The `FrameClassifier` interface represents an abstract frame classifier. There are currently 4 implementations `FrameRelativeClassifier`, `FrameRuleClassifier`, `FrameGFClassifier` and `FrameGF2Classifier`. Only the `FrameGF2Classifier` is now used, as it implements the latest and most successful frame classifier described in section 5.2.3. `FrameRelativeClassifier` is a remnant from an experiment involving classifying frames use not just words properties but other nearby frames.

`BuildCompositeClassifier` is used to build models for the `FrameClassifiers`. An example of it being used can be found in `run.CreateArffs`.

`WekaHelper` is a utility to simplify the loading of weka models. `WekaClassifierFactory` is an abstract interface for creating a specific classifier. This

interface allows most of the code to create and use a classifier without knowledge of the actually classifier type (Naïve Bayes, J48, SVM). There are three concrete implementations of this class: `NBFactory`, `J48Factory`, `SVMFactory`.

`SynchronizeParses` is used synchronize and map tokens between Machinese Semantics tokenization and the FrameNet/SemEval tokenization.

8.8 run

Contains high-level classes, which can run many of the basic functions required to work with the automatic frame extraction system. There are only 5 classes in this module: `CreateArffs`, `CreateModels`, `CreateFrames`, `SemEval` and `UI`.

1. `CreateArffs` processes all annotated frame examples using the `BuildCompositeClassifier` class, which then uses the `FrameClassifier` interface to extract all features and data to create arff files for identifying and disambiguating frames.

   The `CreateArff` class can be called directly from the command line with no parameters. The behavior of the class is controlled with the following variables store in config.xml

   SemEval07, which should point to the directory containing the SemEval2007 data.

   FrameNet, which should point to the directory containing FrameNet data

   LexicalUnits, which should point to the directory containing the annotated lexical unit data.

   msemXML, which should point to the directory containing of the sentences, from the annotated examples listed above, parsed by Machinese Semantics. If the *msemXML* directory does not contain the parsed results for each sentence then the "script/call_msem.sh" script will be called with two parameters: a text file containing the sentence to be parsed and a file where the results of the parse should be placed. The "script/call_msem.sh" should be customized for each user's environment.

2. `CreateModels` processes all of the arff files generated in the `CreateArffs` step. Processing involves creation of models and cross-validation. This class can called directly from the command line and requires no parameters. `CreateModels` will create Naive Bayes, J48 and SVM models and then output the cross-validated results for each classifier.
3. `CreateFrames` is stub piece of code that can used as an example of how to process a sentence into the frame representation and output the representation as XML.
4. `SemEval` class is used to test models on the SemEval 2007 task 19 data. The class can be called directly from the command line but often requires some modification to the implementation depending on the experiment (change of classifier, models, or features). There are 2 basic experiments that the class is usually used for: comparison of Naïve Bayes, J48 and SVM on a single text or running a single classifier on all text to obtain results compared to the gold standard.
5. `UI` provides a graphical user interface for experiments. The interface can call Machinese Semantics using the "script/call_msem.sh" script for a sentence or a file. The results of the Machinese Semantics parse can be visualized using the work of Darren Kipp (2006). The resulting frame representation is output to an XML file. The user interface can also launch a visualization of the result in Graphviz. The Graphviz input file is generated by applying an XSLT to the XML frame representation file.

Appendix III – Sample Arff File

@relation JFramer-ML-data
@attribute FrameElement
{none,creator,original,copy,source,goal,characterization,means,manner,result,depictive,degree,purpose,time,place,reason,instrume
nt,duration,fidelity,iteration,frequency}
@attribute LexicalID {8,9,10,11,12,13,14,15,16,17,5816,5817}
@attribute WordRelation {kNone,kParent,kGrandParent,kSibling,kChild,kGrandChild}

@attribute FrameGrammarFunction
{kUnknown,kSubject,kObject,kDeterminer,kModifier,kAttribute,kPreposition,kAdverbial,kNounGroup,kMain,kGoal,kLocation,kSource,
kContingency,kSubclauseModifier,kComplement,kManner,kCoordination,kPostModifier,kClauseAdverbial,kTime,kCondition,kFreque
ncy,kObjectComplement,kGenitive,kDuration,kQuantity,kComitative,kQuantifier,kIndirectObject,kInstrument,kTheme,kPath,kCopredi
cative,kVocative,kAdAdverbial,kPredeterminer,kTagQuestion,kPreMarker,kVerbChain,kCoordMarker,kAuxiliary,kPhrasalPart,kNegP
art}

@attribute FramePartOfSpeech
{kUnknown,kVerb,kNoun,kPreposition,kDeterminer,kAdjective,kInifinitiveMarker,kPronoun,kNumeral,kAdverb,kCoordinator,kSubordi
nator}

@attribute FrameLeadingPreposition
{female,none,in,from,to,until,onto,into,by,that,if,for,round,around,at,about,after,upon,with,behind,without,on,across,as,of,since,via,bef
ore,down,past,through,during,over,as_if,inside,like,between,towards,outside,against,but_for,although,within,because,near,though,w
hether_or_not,whether,above,up,throughout,up_and_down,along,alongside,under,beside,among,as_though,even_if,along_with,belo
w,beneath,all_over,according_to,from_under,prior_to,because_of,regardless_of,amongst,beyond,as_to,as_in,so,despite,off,apart_fr
om,once,besides,underneath,worth,in_order_that,amid,than,unless,ahead_of,toward,in_between,aboard,round_and_round,instead_
of,unto,halfway_to,nearer,except_for,more_like,even_though,thanks_to,per,in_spite_of,de,amidst,till,as_well_as,as_for,astride,unlik
e,lest,opposite,athwart,as_against,in_line_with,as_from,long_before,whereas,save,short_of,albeit,akin_to,v.,above_all,as_of,except,
as_to_whether,out-
of,v,re,notwithstanding,for_or_against,owing_to,as_opposed_to,atop,en,upwards_of,versus,aside_from,except_that,cos,to/from,but,
con,cum,vis-a-vis,a_la,ante,sans,aka}

@attribute FrameVoice {kNone,kActive,kPassive}

@attribute FrameSemanticClass {false,none,human,male,female,plant,animal,nationality}

@attribute FrameAnimate {true,false}

@attribute FrameElementGrammarFunction
{kUnknown,kSubject,kObject,kDeterminer,kModifier,kAttribute,kPreposition,kAdverbial,kNounGroup,kMain,kGoal,kLocation,kSource,
kContingency,kSubclauseModifier,kComplement,kManner,kCoordination,kPostModifier,kClauseAdverbial,kTime,kCondition,kFreque
ncy,kObjectComplement,kGenitive,kDuration,kQuantity,kComitative,kQuantifier,kIndirectObject,kInstrument,kTheme,kPath,kCopredi
cative,kVocative,kAdAdverbial,kPredeterminer,kTagQuestion,kPreMarker,kVerbChain,kCoordMarker,kAuxiliary,kPhrasalPart,kNegP
art}

@attribute FrameElementPartOfSpeech
{kUnknown,kVerb,kNoun,kPreposition,kDeterminer,kAdjective,kInifinitiveMarker,kPronoun,kNumeral,kAdverb,kCoordinator,kSubordi
nator}

@attribute FrameElementLeadingPreposition
{female,none,in,from,to,until,onto,into,by,that,if,for,round,around,at,about,after,upon,with,behind,without,on,across,as,of,since,via,bef
ore,down,past,through,during,over,as_if,inside,like,between,towards,outside,against,but_for,although,within,because,near,though,w
hether_or_not,whether,above,up,throughout,up_and_down,along,alongside,under,beside,among,as_though,even_if,along_with,belo
w,beneath,all_over,according_to,from_under,prior_to,because_of,regardless_of,amongst,beyond,as_to,as_in,so,despite,off,apart_fr
om,once,besides,underneath,worth,in_order_that,amid,than,unless,ahead_of,toward,in_between,aboard,round_and_round,instead_
of,unto,halfway_to,nearer,except_for,more_like,even_though,thanks_to,per,in_spite_of,de,amidst,till,as_well_as,as_for,astride,unlik
e,lest,opposite,athwart,as_against,in_line_with,as_from,long_before,whereas,save,short_of,albeit,akin_to,v.,above_all,as_of,except,
as_to_whether,out-

of,v,re,notwithstanding,for_or_against,owing_to,as_opposed_to,atop,en,upwards_of,versus,aside_from,except_that,cos,to/from,but,con,cum,vis-a-vis,a_la,ante,sans,aka}

@attribute FrameElementVoice {kNone,kActive,kPassive}

@attribute FrameElementSemanticClass {false,none,human,male,female,plant,animal,nationality}

@attribute FrameElementAnimate {true,false}

@attribute IntermediateFunction {kUnknown,kSubject,kObject,kDeterminer,kModifier,kAttribute,kPreposition,kAdverbial,kNounGroup,kMain,kGoal,kLocation,kSource,kContingency,kSubclauseModifier,kComplement,kManner,kCoordination,kPostModifier,kClauseAdverbial,kTime,kCondition,kFrequency,kObjectComplement,kGenitive,kDuration,kQuantity,kComitative,kQuantifier,kIndirectObject,kInstrument,kTheme,kPath,kCopredicative,kVocative,kAdAdverbial,kPredeterminer,kTagQuestion,kPreMarker,kVerbChain,kCoordMarker,kAuxiliary,kPhrasalPart,kNegPart}

@attribute IntermediatePartOfSpeech {kUnknown,kVerb,kNoun,kPreposition,kDeterminer,kAdjective,kInifinitiveMarker,kPronoun,kNumeral,kAdverb,kCoordinator,kSubordinator}

@attribute IntermediateLeadingPreposition {female,none,in,from,to,until,onto,into,by,that,if,for,round,around,at,about,after,upon,with,behind,without,on,across,as,of,since,via,before,down,past,through,during,over,as_if,inside,like,between,towards,outside,against,but_for,although,within,because,near,though,whether_or_not,whether,above,up,throughout,up_and_down,along,alongside,under,beside,among,as_though,even_if,along_with,below,beneath,all_over,according_to,from_under,prior_to,because_of,regardless_of,amongst,beyond,as_to,as_in,so,despite,off,apart_from,once,besides,underneath,worth,in_order_that,amid,than,unless,ahead_of,toward,in_between,aboard,round_and_round,instead_of,unto,halfway_to,nearer,except_for,more_like,even_though,thanks_to,per,in_spite_of,de,amidst,till,as_well_as,as_for,astride,unlike,lest,opposite,athwart,as_against,in_line_with,as_from,long_before,whereas,save,short_of,albeit,akin_to,v.,above_all,as_of,except,as_to_whether,out-
of,v,re,notwithstanding,for_or_against,owing_to,as_opposed_to,atop,en,upwards_of,versus,aside_from,except_that,cos,to/from,but,con,cum,vis-a-vis,a_la,ante,sans,aka}

@attribute IntermediateVoice {kNone,kActive,kPassive}

@attribute IntermediateSemanticClass {false,none,human,male,female,plant,animal,nationality}

@attribute IntermediateAnimate {true,false}

@data
creator,10,kChild,kMain,kVerb,none,kActive,none,false,kSubject,kPronoun,none,kNone,female,false,kUnknown,kUnknown,none,kNone,none,false
copy,10,kChild,kMain,kVerb,none,kActive,none,false,kObject,kNoun,none,kNone,none,false,kUnknown,kUnknown,none,kNone,none,false
none,10,kGrandChild,kMain,kVerb,none,kActive,none,false,kGenitive,kPronoun,none,kNone,female,false,kObject,kNoun,none,kNone,none,false
source,10,kChild,kMain,kVerb,none,kActive,none,false,kLocation,kNoun,in,kNone,none,false,kUnknown,kUnknown,none,kNone,none,false
none,10,kGrandChild,kMain,kVerb,none,kActive,none,false,kDeterminer,kDeterminer,none,kNone,none,false,kLocation,kNoun,in,kNone,none,false

Appendix IV – Complete List of Dependency Relations

AdAdverbial
Adverbial
Attribute
Auxiliary
ClauseAdverbial
Comitative
Complement
Condition
Contingency
Coordination
CoordMarker
Copredicative
Determiner
Duration
Frequency
Genitive
Goal
IndirectObject
Instrument
Location
Main
Manner
Modifier
NegPart
NounGroup
Object
ObjectComplement
Path
PhrasalPart
PostModifier
Predeterminer
PreMarker
Preposition
Quantifier
Quantity
Source
SubclauseModifier
Subject
TagQuestion
Theme

Time
VerbChain
Vocative

References

Collin Baker, Michael Ellsworth and Katrin Erk. 2007. SemEval-2007 Task 19: Frame Semantic Structure Extraction. *In Proceedings of the Fourth International Workshop on Semantic Evaluations* (*SemEval-2007*), pages 99-104, ACL 2007.

Cosmin Adrian Bejan and Chris Hathaway. 2007. UTD-SRL: A Pipeline Architecture for Extracting Frame Semantic Structures. *In Proceedings of the Fourth International Workshop on Semantic Evaluations* (*SemEval-2007*), ACL 2007.

Hans C. Boas. 2002. Bilingual FrameNet dictionaries from machine translation. In *the proceedings of the Third International Conference on Language Resources and Evaluation* (LREC), pages 1364-1371.

Hans C. Boas, Elias Ponvert, Mario Guajardo, and Sumeet Rao. 2006. The current status of German FrameNet. *SALSA workshop at the University of the Saarland.* Saarbrücken, Germany.

Carlos, Petruck, Miriam. 2003. Surprise: Spanish FrameNet. *International Congress of Linguists. Workshop on Frame Semantics.* Prague, Czech Republic.

S. Ceccato. 1961. *Linguistic Analysis and Programming for Machine Translation.* New York: Gordon & Breach.

Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods.* Cambridge University Press, 2000

Christiane Fellbaum (editor). 1998. WordNet: An electronic lexical database. MIT Press.

Charles J. Fillmore. 1976. Frame semantics and the nature of language. *In Annals of the New York Academy of Sciences: Conference on the Origin and Development of Language and Speech*, 280: 20-32.

Charles J. Fillmore, Josef Ruppenhofer, and Collin Baker. 2004. FrameNet and representing the ink between semantic and syntactic relations. *In Churen Huan and Winfried Lenders, editors, Frontiers in Linguistics, volume I of Language and Linguistics Monograph Series B. Institute of Linguistics, Academia Sinica, Taipei*, pages 19-59.

Radu Florian, Silviu Cucerzan, Charles Schafer, and David Yarowsky. 2002. Combining classifiers for word sense disambiguation. In *Natural Language Engineering.*

Milton Friedman, 1937. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *In Journal of the American Statistical Association*, Vol 32, No 200, pages 675-701

Daniel Gildea and Daniel Jurafsky: 2002. Automatic Labeling of Semantic Roles. In *Computational Linguistics* 28(3): pages 245-288.

Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. Guinness, Peter F. Patel-Schneider, and Lynn Andrea Stien. 2003. OWL Web Ontology Reference. www.w3c.org/TR/2003/WD-owl-ref-20030331/

Richard Johansson and Pierre Nugues, 2006. Automatic Annotation for All Semantic Layers in FrameNet. In *EACL – Demos*

Richard Johansson and Pierre Nugues. 2007. LTH: Semantic Structure extraction using nonprojective dependency trees. *In Proceeding of the 17th Inter-national Workshop on Semantic Evaluations (SemEval-2007)*, pages 227-230, Prague, Czech Republic.

Daniel Jurafsky, and James H. Martin. 2000. "First Order Predicate Calculus", in Daniel Jurafsky, and James H. Martin, Speech and Language Processing, Prentice-Hall, Inc, 2000, pp. 513.

Dan Jurafsky, Sameer Pradham, Wayne Ward, Kadri Hacioglu, James H. Martin, 2004. Shallow Semantic Parsing using Support Vector Machines. *In Proceedings of the Human Language Technology Conference/North American chapter of the Association of Computational Linguistics (HLT/NAACL)*

Timo Järvinen. 2003. Multi-layered annotation scheme for treebank annotation. In *Joakim Nivre and Erhard Hinrichs, editors, TLT 2003. Proceedings of the Second Workshop on Treebanks and Linguistic Theories*, pages 93 -104, Växjö. Växjö University Press.

Karin Kipper, Hoa Trang Dang, and Martha Palmer. 2000. Class based construction of a verb lexicon. *In Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000), Austin, TX.*

John Lafferty, Andrew McCallum, Fernando Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *In Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001),* 2001.

Dekang Lin and Patrick Pantel. 2001. *DIRT – Discovery of Inference Rules from Text,* In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. Pages 323-328.

Ken. Litkowski. 2007. CLR: Integration of FrameNet in a Text Representation System. *In Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 113-116, ACL 2007.

Lluis Márquez, Xavier Carreras, Kenneth C. Litkowski, and Suzanne Stevenson. 2008. Semantic Role Labeling: An Introduction to the Special Issue. In *Computational Linguistics.* Vol 34, No 2, pp. 145-156.

M. Masterman. 1961. *Semantic message detection for machine translation, using Interlingua.* In Proceedings of the 1961 International conference on machine translation.

G. Melli, Y. Wang, Y. Liu, M. M. Kashani, Z. Shi, B. Glu, A. Sarkar, and F. Popowich. 2005. Description of SQUASH, the SFU question and answering summary handler for the DUC-2005 Summarization Task. In *Proceedings of the HLT/EMNLP Document Understanding Conference* (DUC), Vancover, Canada.

Marvin Minsky. 1975. *A Framework for Representing Knowledge.* In P. H. Winston (Ed.), The Psychology of Computer Vision. New York: McGraw-Hill, pp. 211-277.

T. Mitchell, "Decision Tree Learning", in T. Mitchell, *Machine Learning*, The McGraw-Hill Companies, Inc., 1997, pp. 52-78

T. Mitchell, "Bayesian Learning", in T. Mitchell, *Machine Learning*, The McGraw-Hill Companies, Inc., 1997, pp. 160-206

Srini Narayanan and Sanda Harabagiu. 2004. Question Answering based on semantic structures. In *Proceeding of the 20th International Conference on Computational Linguistics* (COLING), pages 693-701, Geneva, Switzerland.

Kyoko Hirose Ohara, Seiko Fujii, Hiroaki Saito, Shun Ishizaki, Toshio Ohori, and Ryoko Suzuki. 2003. The Japanese FrameNet Project: A Preliminary Report. In *Proceedings of Pacific Association for Computational Linguistics* (PACLING'03), pages 249-254. Halifax, Canada.

Martha Palmer, Paul Kingsbury, and Daniel Gildea. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics 31* (1): 71–106.

M. Ross Quillian. 1967. *Word concepts: A theory and simulation of some basic capabilities.* In Behavioral Science, 12:410-430

Lawrence R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE*, 77 (2), p. 257–286, February 1989

Josef Ruppenhofer, Michael Ellsworth, Miriam R. L. Petruck, Christopher R. Johnson, and Jan Scheffczyk. 2006. FrameNet II: Extended Theory and Practice. In *The Book.* http://framenet.icsi.berkeley.edu/

R.C. Schank and K.M. Colby. 1973. Computer Models of Thought and Language. San Francisco: Freeman.

R.C. Schank and B.L. Nash-Webber. 1975. Using Knowledge To Understand. In *The Primitive ACTs Of Conceptual Dependency.*

R.C. Schank and C.J. Rieger. 1974. Inference and the computer understanding of natural language. In *Artificial Intelligence* 5(4):373-412

S.C. Shapiro. 1971. *A net structure for semantic information storage, deduction and retrieval.* In Proceedings of the Second International Joint Conference of Artificial Intelligence. pp. 512-523.

Lei Shi and Rada Mihalcea, Semantic Parsing Using FrameNet and WordNet. *In Proceedings of the Human Language Technology Conference*, companion volume (HLT/NAACL 2004), Boston, May 2004.

John F. Sowa. 1984. *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, MA.

Ian H. Witten and Eibe Frank. 2005. *Data Mining: Practical machine learning tools and techniques*, 2nd Edition, Morgan Kaufmann, San Francisco

Y.A. Wilks. 1972. *Grammar, Meaning and the Machine Analysis of Language.* London: Routledge & Kegan Paul.

W. Woods. 1985. *What's in a Link: Foundations for Semantic Networks.* In Representation and Understanding: Studies in Cognitive Science, p. 35-82, Academic Press, New York, N.Y.