Natural Language Search of Structured Documents

by

Stephen W. Oney

S.B., Computer Science; S.B., Mathematics M.I.T. (2007)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2008

Author .....
Department of Electrical Engineering and Computer Science
August 22, 2008

Certified by .....
Deb K. Roy
Professor of Media Arts and Sciences
Chair, Academic Program in Media Arts and Sciences
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Natural Language Search of Structured Documents

by
Stephen W. Oney

Submitted to the Department of Electrical Engineering and Computer Science

August 22, 2008

In partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This thesis focuses on techniques with which natural language can be used to search for specific elements in a structured document, such as an XML file. The goal is to create a system capable of being trained to identify features, of written English sentence describing (in natural language) part of an XML document, that help identify the sections of said document which were discussed.

In particular, this thesis will revolve around the problem of searching through XML documents, each of which describes the play-by-play *events* of a baseball game. These events are collected from Major League Baseball games between 2004 and 2008, containing information detailing the outcome of every pitch thrown. My techniques are trained and tested on written (newspaper) summaries of these games, which often refer to specific game events and statistics.

The choice of these training data makes the task much more complex in two ways. First, these summaries come from multiple authors. Each of these authors has a distinct writing style, which uses language in a unique and often complex way. Secondly, large portions of these summaries discuss facts outside of the context of the play-by-play events of the XML documents. Training the system with these portions of the summary can create a problem due to sparse data, which has the potential to reduce the effectiveness of the system.

The end result is the creation of a system capable of building classifiers for natural language search of these XML documents. This system is able to overcome the two aforementioned problems, as well as several more subtle challenges. In addition, several limitations of alternative, strictly feature-based, classifiers are also illustrated, and applications of this research to related problems (outside of baseball and sports) are discussed.

Thesis Supervisor: Deb K. Roy
Title: Professor, MIT Media Laboratory

Acknowledgments

First, I want to give thanks to my family, without whom, none of this would be possible. My parents – Stephenie and Logan Oney – gave me the strength and will to follow my own path and become the first engineer (by training) in the family. My siblings – Theresa, Logan, Christina, and Marcus - provide me with ample positive feedback and encouragement to keep me going. In particular, Logan has always inspired me to build self-confidence and persuaded me to apply to MIT in the first place. I strive to do for others that which my family has done for me.

I would like to thank Professor Deb Roy and the members of the Cognitive Machines Group who helped me shape and direct my research. In particular, I would like to thank Dr. Michael Fleischman and Dr. Nikolaos Mavridis, both of whom have guided my UROPs in the Cognitive Machines Group. Nikolaos introduced me to research and the group my freshman year and Michael and his research helped guide me in ways to solve many of the problems I initially encountered.

Finally, I would like to thank everyone who helped me since my matriculation at MIT, including my academic advisors – Jocelyn Heywood and Professor Ronitt Rubinfeld – and Professor Sekazi Mtingwa.

Contents

List of Figures

List of Tables

Chapter 1 – Introduction

1.1 The Problem

The broad goal of the research behind this thesis is to form a bridge between spoken language and elements easily recognizable by a computer. Although my main focus will be on the English language and Extensible Markup Language (XML) documents of a specific format (illustrated in Appendix A), most of the ideas discussed are still applicable to other languages and document formats.

To native English speakers, the English language and its grammatical rules might, at first glance, seem very natural and well defined. However, in many ways, the English language is very complex, highly context dependent and potentially ambiguous – one sentence might have different meanings under different contexts. Thus, machine processing of English (and every other commonly spoken language) is believed by some to be an AI-Complete[1] problem, making it necessary to build statistical and other models of languages. The field of Natural Language Processing has evolved out of necessity due to these complexities. In addition, several file formats, including XML, and programming languages have been created as intermediaries between written word and computer instructions. Still, XML and similar formats, such as HTML fall much closer to machine language than written word. Much work is still required to bring these languages and formats closer to natural language. This is the problem addressed by this thesis.

I have chosen to focus on XML documents detailing events from Major League Baseball (MLB) games. Sports are a particularly good dataset for language processing

[1] An AI-Complete problem is defined as a problem that is equivalent to making computers as intelligent as humans. [1]

tasks because they are just restrictive enough with their possible contexts. The rules of any sport limit the terminology and information necessary to understand the context of any game event. As previously mentioned, one of the main problems with computers understanding natural language is that language is highly context-dependent. Although there are many efforts to make computers more context-aware, the problem is outside of the scope of this thesis. Thus, I chose a topic with limited context.

Although rules of every major sport are restrictive enough to limit context, they are simultaneously permissive enough to allow for an unlimited variety of possible events and outcomes, which is precisely why they are exciting – many different things could happen. Baseball in particular is a good sport on which to focus. In baseball, plays are cleanly segmented – each play has a minimal effect on the context necessary to understand the rest of the game. In addition, it is easy to identify and segment individual game plays, unlike continuous-play sports such as basketball and soccer.

MLB games in particular are an ideal data source. First, the games are well documented, with accurate records of nearly every pitch thrown in the league since 2004. Second, there is a substantial amount of training and testing data available. Thirty teams play in the MLB, each of which plays 162 regular-season games before an eight-team best-of-seven post-season playoff. This ensures that at least 2,458 games are played each year. Third, each game already has an abundant number of natural language summaries for it. These summaries are often published in local newspapers or online. For each of the (over 2,458) games played, MLB.com has two summaries – one local summary for the home and away teams. Thus, by itself, MLB.com has nearly 5,000 natural language summaries for every year. These summaries, from 2005 to part of 2008, are the testing

and training data for my system, which builds a classifier capable of taking sentences from these summaries and elsewhere, and searching through an XML document, containing events describing what happened during a particular game, for the relevant elements.

The following formalization of the task will introduce notation used throughout this thesis:

Each summary, S, is a sequence of sentences:

$$S = (s_1, s_2, ..., s_n)$$

Where the number of sentences, n, is $|S|$. In addition, define each set of game events, E:

$$E = (e_1, e_2, ..., e_m)$$

Where $|E| = m$. Every event, e, stores the pitcher, batter, a log of each pitch thrown during that at-bat, and a short textual description of that event, as can be seen in Appendix A. Our goal is to, for each sentence $s_i (i = 1,2,...,|S|)$, figure out which subset of E is discussed. For this purpose, define a function that properly maps a sentence, $s \in S$, to the set of events , $F \subset E$, which is discussed in that sentence. This function will be called *DISCUSSED*.

To illustrate, take the following imaginary baseball game:

Event	Inning	Pitcher	Batter	Description
e_1	1 Top	Peter	Betsy	Betsy hits a home run to right field
e_2	1 Top	Peter	Billy	Billy hits a home run to center field
e_3	1 Top	Peter	Bobby	Bobby hits a home run to left field
e_4	1 Top			Marvin, the home team manager, forfeits

Table 1 - Example (Imaginary) Baseball Game

And its matching summary:

Sentence	Text
s_1	It was the shortest game in the history of baseball.

s_2	Betsy, Billy, and Bobby hit three consecutive home runs in the first three plays.
s_3	The home team's manager, Marvin, decided to pull the game in the middle of the inning.
s_4	"It was unbelievable," said Peter.
s_5	"I don't know what hit me."

Table 2 - Example (Imaginary) Summary

This would result in the following definition of *DISCUSSED*:

S	***DISCUSSED*(s)**
s_1	$\{\}$
s_2	$\{e_1,e_2,e_3\}$
s_3	$\{e_4\}$
s_4	$\{\}$
s_5	$\{\}$

Table 3 - Discussion Table for Imaginary Game

Thus, our goal is to find a mapping from S to subsets of E, matching the *DISCUSSED* function as closely as possible. A diagram showing the basic structure of the system is shown in Figure 1 below.

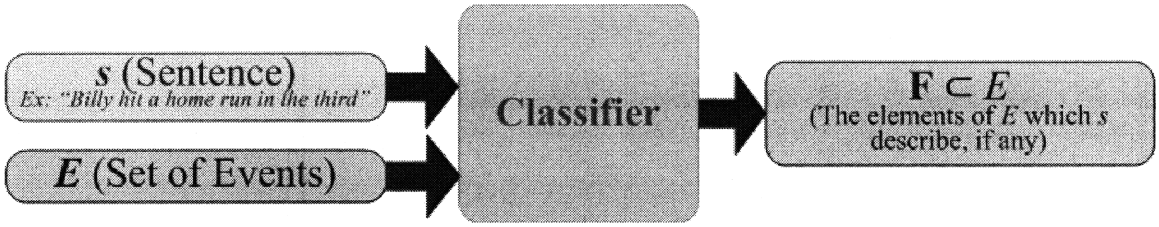


Figure 1 - Basic Classifier Diagram

1.2 Applications

The most obvious and direct application of the work described in this thesis is towards an interface that allows users to easily search for specific parts of any baseball game by inputting an English sentence. In addition, minimal changes would be required to build a database-query system capable of searching across multiple games. Further work could

allow the system to generate statistics based on English queries (Ex: "How many home runs did Manny Ramirez hit in 2007?")

However, the applications of the approach, techniques, and lessons learned from this project are not limited to baseball, sports, or any particular domain. As mentioned in the previous section, the main limiting factor is context – given enough context, a similar system can be built for any domain. This would allow a structured search of any type of structured document on any subject matter with sufficient context understanding.

1.3 Related Work

I did the research for this thesis as an extension of some research done under Dr. Michael Fleischman, who researched language modeling, focusing on sports video [2] for his Ph.D. thesis.

Benjamin Snyder, currently a Ph.D. student, addressed a similar problem in a different way – by using a strictly feature-based approach [3]. Snyder's focus is on generating a set of features to align a database containing game statistics. Unlike the system described in Snyder's research, I assume no access to game statistics, generated or fetched, which were crucial sources of features for Snyder's system. As a result, after some experimentation, I ultimately decided against a strictly feature-based system. In the next chapter, I describe some of the results and difficulties of using a strictly feature-based system similar to Snyder's.

Somewhat related to this work are the efforts to create natural language programming interfaces, all of which, on some level, are doing a search of some sort. In particular, the Chickenfoot plug-in for the Firefox browser [4] stands out because it is

capable of searching HTML (which is similar to XML) pages to allow for easier programming of Firefox plug-in (closer to natural language, but still not written word).

Finally, although the problems might seem disparate, I found many similarities between the task of searching structured documents and that of machine translation[2]. Although the similarities will be detailed in the next section, the basic idea is roughly that we are translating from English to a highly structured language.

1.4 Approaches

I tried two different approaches while trying to solve this problem. The first system I built was strictly feature-based, similar to Snyder's system described in the above section. The second, which uses a few techniques from machine translation, is the main system discussed in the subsequent chapters. My approaches for data gathering will be discussed in the next chapter, followed by a description of the two aforementioned systems in chapters 3 and 4.

Chapter 2 – Data Sources

2.1 Data Collection

There were two types of data to collect – game events and natural language (newspaper) game summaries. From there, I combined the data into a set of XML documents for organizational purposes, which allowed me to leverage existing tools for accessing XML documents. All of the data collected for this project comes from the website MLB.com.

---

[2] Machine translation is defined as the automated translation of a document written in one language to another by a computer.

2.1.1 Events and Game Statistics

The first task is to collect a set of game events in order to create the structured document which will be searched. In other words, I have to build of a set of events, *E*, for every game.

The *E* for every game can be extracted from MLB.com's Gameday feature, which gives a dynamic overview of every game played, with information as detailed as the velocity, spin, and location of each pitch, and statistics of each player (how many hits, scores, etc. they have). The online interface for MLB.com's Gameday feature is shown in Figure 2 below.

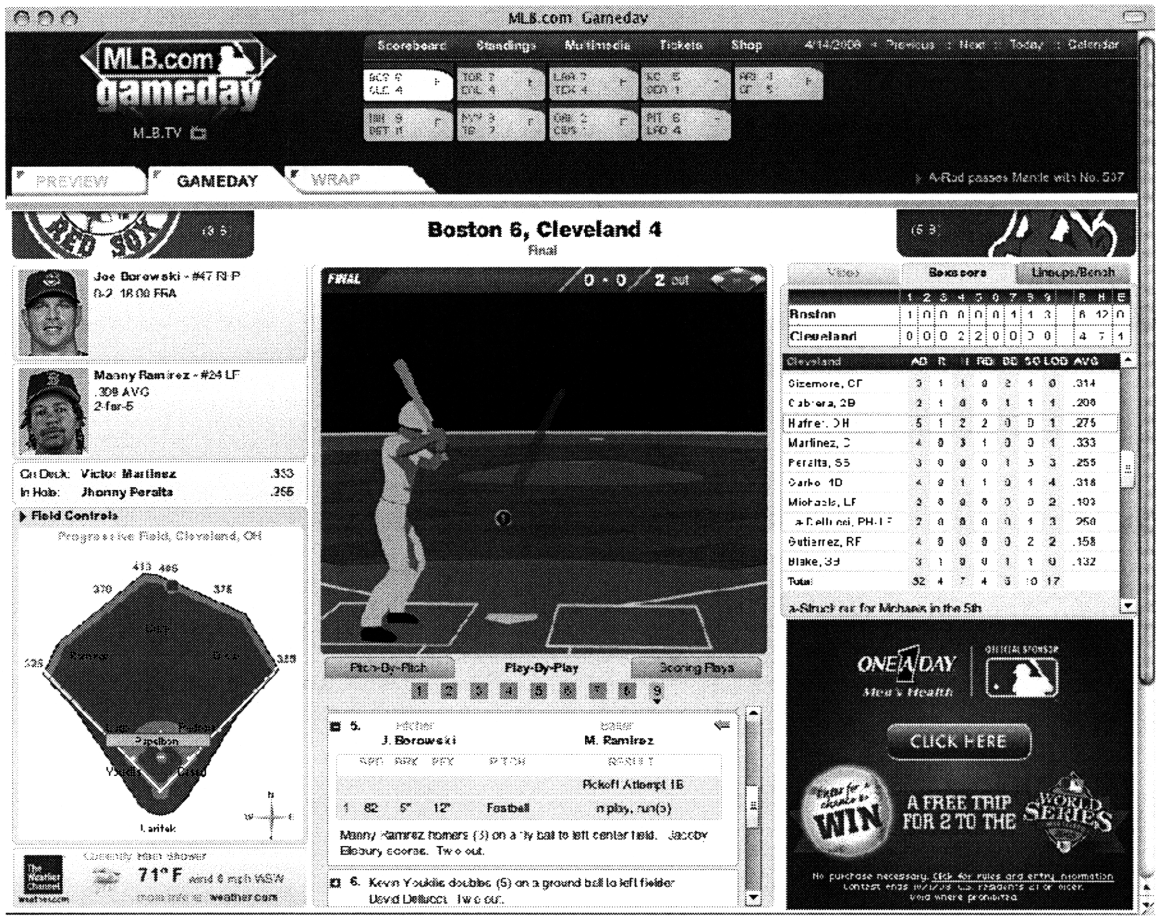


Figure 2 - Sample Game Events

There are several XML files driving this interface. These XML files will subsequently be referred to as "unassembled", in contrast to the "assembled" XML files we are building and will subsequently search. For every game, these unassembled XML files need to be combined into a single XML file with a consistent structure. Two of the unassembled XML files are shown in Figure 3, which shows the pitches of a particular inning (this file is from the ninth inning) and Figure 4, which shows the players involved in the game.

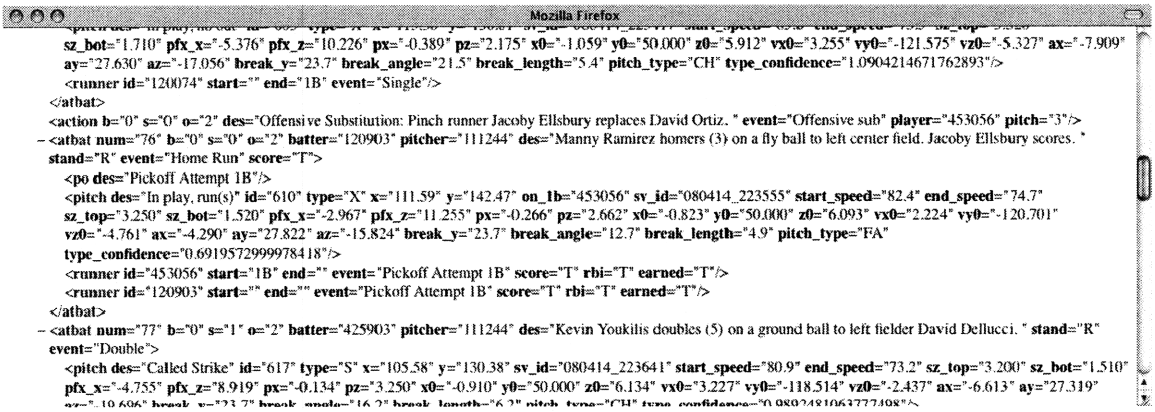


Figure 3 - Sample Unassembled Inning XML File

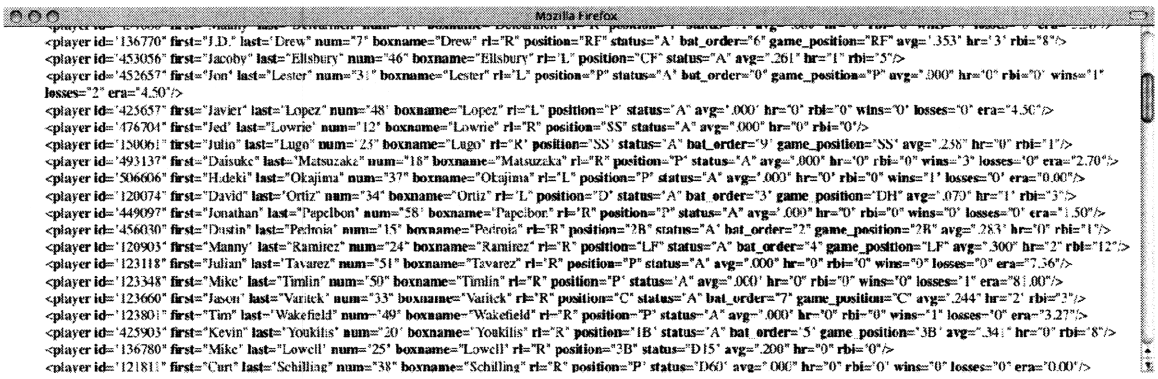
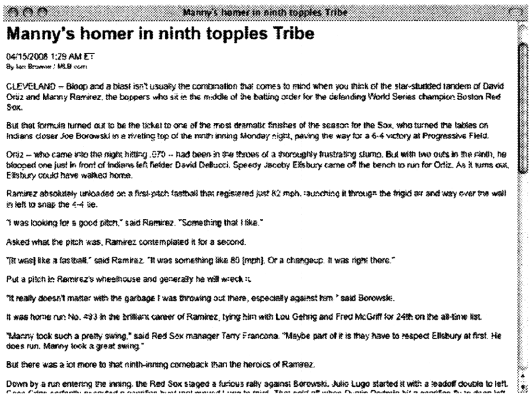


Figure 4 - Sample Unassembled Players XML File

These files and others are assembled to form a set of events, *E*, for that game. A sample assembled XML file is shown in Appendix A.

2.1.2 Game Summaries

In addition to the set of events, *E*, we need a natural language summary, *S*, to train and test the system. For this, I used the summaries available on MLB.com. For each game, there are two summaries – one from the perspective of the home team and one from the perspective of the away team. In addition to having different authors, they also generally discuss different aspects and events of the game. Two such summaries, each describing the Boston – Cleveland game (pictured in Figure 2), are shown below. Figure 5 shows the summary of the game from Boston's perspective, and Figure 6 shows the game from Cleveland's perspective:



Manny's homer in ninth topples Tribe

Manny's homer in ninth topples Tribe

04/15/2008 1:29 AM ET

By Ian Browne / MLB.com

CLEVELAND – Bloop and a blast isn't usually the combination that comes to mind when you think of the star-studded tandem of David Ortiz and Manny Ramirez, the boppers who sit in the middle of the batting order for the defending World Series champion Boston Red Sox.

But that formula turned out to be the ticket to one of the most dramatic finishes of the season for the Sox, who turned the tables on Indians closer Joe Borowski in a riveting top of the ninth inning Monday night, paving the way for a 6-4 victory at Progressive Field.

Ortiz – who came into the night hitting .070 – had been in the throes of a thoroughly frustrating slump. But with two outs in the ninth, he blooped one just in front of Indians left fielder David Dellucci. Speedy Jacoby Ellsbury came off the bench to run for Ortiz. As it turns out, Ellsbury could have walked home.

Ramirez absolutely unloaded on a first-pitch fastball that registered just 82 mph, launching it through the frigid air and way over the wall in left to snap the 4-4 tie.

"I was looking for a good pitch," said Ramirez. "Something that I like."

Asked what the pitch was, Ramirez contemplated it for a second.

"[It was] like a fastball," said Ramirez. "It was something like 80 [mph]. Or a changeup. It was right there."

Put a pitch in Ramirez's wheelhouse and generally he will wreck it.

"It really doesn't matter with the garbage I was throwing out there, especially against him," said Borowski.

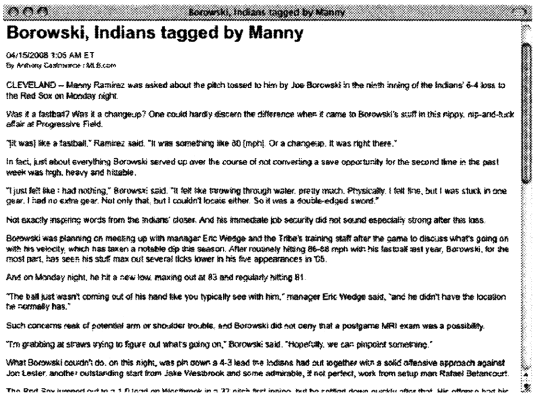
It was home run No. 493 in the brilliant career of Ramirez, tying him with Lou Gehrig and Fred McGriff for 24th on the all-time list.

"Manny took such a pretty swing," said Red Sox manager Terry Francona. "Maybe part of it is they have to respect Ellsbury at first. He does run. Manny took a great swing."

But there was a lot more to that ninth-inning comeback than the heroics of Ramirez.

Down by a run entering the inning, the Red Sox staged a furious rally against Borowski. Julio Lugo started it with a leadoff double to left.

Figure 5 - Boston Summary



Borowski, Indians tagged by Manny

Borowski, Indians tagged by Manny

04/15/2008 1:05 AM ET

By Anthony Castrovince / MLB.com

CLEVELAND – Manny Ramirez was asked about the pitch tossed to him by Joe Borowski in the ninth inning of the Indians' 6-4 loss to the Red Sox on Monday night.

Was it a fastball? Was it a changeup? One could hardly discern the difference when it came to Borowski's stuff in this nippy, nip-and-tuck affair at Progressive Field.

"[It was] like a fastball," Ramirez said. "It was something like 80 [mph]. Or a changeup. It was right there."

In fact, just about everything Borowski served up over the course of not converting a save opportunity for the second time in the past week was high, heavy and hittable.

"I just felt like I had nothing," Borowski said. "It felt like throwing through water, pretty much. Physically, I felt fine, but I was stuck in one gear. I had no extra gear. Not only that, but I couldn't locate either. So it was a double-edged sword."

Not exactly inspiring words from the Indians' closer. And his immediate job security did not sound especially strong after this loss.

Borowski was planning on meeting up with manager Eric Wedge and the Tribe's training staff after the game to discuss what's going on with his velocity, which has taken a notable dip this season. After routinely hitting 86-88 mph with his fastball last year, Borowski, for the most part, has seen his stuff max out several ticks lower in his five appearances in '08.

And on Monday night, he hit a new low, maxing out at 83 and regularly hitting 81.

"The ball just wasn't coming out of his hand like you typically see with him," manager Eric Wedge said, "and he didn't have the location he normally has."

Such concerns reek of potential arm or shoulder trouble, and Borowski did not deny that a postgame MRI exam was a possibility.

"I'm grabbing at straws trying to figure out what's going on," Borowski said. "Hopefully, we can pinpoint something."

What Borowski couldn't do, on this night, was pin down a 4-3 lead the Indians had put together with a solid offensive approach against Jon Lester, another outstanding start from Jake Westbrook and some admirable, if not perfect, work from setup man Rafael Betancourt.

Figure 6 - Cleveland Summary

After fetching these summaries and matching them to games (and their events, *E*), the web styling and other HTML elements are stripped, leaving only the resulting text. The next step is to separate the summary into individual sentences. This is done with the algorithm described by Kiss & Strunk (2006) [5] using the Natural Language Toolkit (NLTK) written in Python [6].

2.2 *File Format*

An example of a fully assembled file is shown in Appendix A. Each file has two portions – a "searchable" portion, which contains all of the events and game information, and a summary portion, which contains the sentences of a particular summary. Put another way, the searchable portion of the assembled file is E and the unsearchable portion is S. However, the fact that they are included in the same file is only a technical detail, done for convenience, but not strictly necessary for any of the techniques used in this project.

The searchable portion of the file contains the names of the home and away teams; the players for both teams; and information for each inning, including pitches, hits, and other game events. Some information is discarded from the unassembled inning file (seen in Figure 3), such as pitch velocity and location, and player statistics (hits, runs, and scores for each player). For the remainder of this document, the specific file format used will be abstracted away, and we will use the terms introduced in Chapter 1 for the summaries, sentences, and game events for the sake of generality.

Chapter 3 – Feature-Based System

My first attempt to create this system was with a system based on a set of "features", similar to that described in Snyder's paper [3]. A feature is any characteristic of a sentence. An example of one possible valid feature is "how many words are in the sentence?" Most features in the system are binary features. A binary feature is a feature that has a yes-or-no answer. An example of a binary feature is "does the sentence contain the word 'pitch'?" These features are combined into a feature vector, Φ.

I implemented two versions of the feature-based system (both of which are entirely distinct from the translation-like system described in Chapter 4). I refer to the first as the "one-stage" system, and the second as the "two-stage" system. The one-stage system tries to directly match each sentence with exactly one event, or with the "null event" (meaning that the sentence does not describe any events). The two-stage system includes most of the features of the one-stage system, but has two stages. The first stage predicts whether or not a sentence is matched to any event, and the second stage predicts which sentence describes which event, for every sentence predicted to describe any event. Both systems are described in the next two sections.

3.1 One-Stage System

Recall from Chapter 1 that there are two data for each game – the sentences ($S = (s_1, s_2, ..., s_n)$) and the events ($E = (e_1, e_2, ..., e_m)$) and that our goal is to determine which sentence discusses which event. The first step is to make a feature vector for every possible pairing between sentences and events (including the null event, $\varnothing$). Each of these feature vectors will be referred to as $\Phi_{s,e}$. This can be put in mathematical terms - $\exists \phi_{s,e} \forall s \in S, e \in (E + \varnothing)$. There are $|S| * (|E| + 1)$ features vectors for each game[3] (S, E pair). Features are split into two types – the "simple" features, and the "generated" features. Simple features are features that are explicitly defined, whereas generated features are generated from a template. Table 4 and Table 5 below show the simple features and generated feature templates respectively.

Feature	Description	Possible Values

[3] For our purposes, a "game" can be described as a set of game events and a summary. Put another way, a game is an (S, E) pair.

Batter Mentioned	True if any part of the name of the batter (first, last, or both, as described by *e*) is mentioned in *s*	{0,1}
Pitcher Mentioned	True if the name of the pitcher (as described by *e*) is mentioned in *s*	{0,1}
Other Players	The number of other players mentioned in *e* that are in *s*	{0,1,2,...}
Common Word Sequences	Looks for phrases and words that appear in *s* and *e*, and assigns a score based on the similarities. For every common sequence *u* between *s* and *e*, the score increases by u^2. This means that longer common sequences have a much larger effect than many smaller common sequences	{0,1,2,...}
Same Inning	Based on the current and previous innings, the system takes a guess at the inning to which *s* refers. This feature is true if the guessed inning of *s* matches the inning of *e*	{0,1}
Is Quotation	True if the sentence either is part of a multi-sentence quotation, or has a quotation in it	{0,1}
Sentence Length	The number of characters in a sentence	{0,1,2,...}
Is Hit	True if *e* describes an event where the batter hit the ball	{0,1}
Is Score	True if *e* describes an event in which any player scored	{0,1}
Is At Bat	True if *e* is an at bat, as opposed to an action, such as a pitcher replacement	{0,1}

Table 4 - One-Stage Simple Features

Note that Table 5 shows the feature *templates*, with which many (hundreds) features are automatically generated.

Feature Template	**Description**	**Possible Values**
Common Words	This feature template generates features that are true if a specified word(s) appear(s) in both *s* and the description of *e*	{0,1}
Words Before and After Names	This feature template generates features that are true if a specified word(s) appear(s) before the name of any players mentioned in *s*	{0,1}

Table 5 - One-Stage Generated Feature Templates

The generated feature templates shown in table 5 come in two varieties each – unigram and bigram versions. The unigram version deals with only one word, whereas the bigram version looks at two words. For example, take the sentence "Billy hit the

ball". The "words before and after names" feature template would generate the following features:

Feature Template	Description	Possible Values
$UWBN_{\emptyset}$ *(Unigram Words Before Names)*	True if no word comes before any given name	{0,1}
$BWBN_{\emptyset,\emptyset}$ *(Bigram Words Before Names)*	True if no two words come before any given name	{0,1}
$UWAN_{hit}$ *(Unigram Words After Names)*	True if "hit" comes after any given name	{0,1}
$BWAN_{hit,the}$ *(Bigram Words After Names)*	True if "hit the" comes after any given name	{0,1}

Table 6 - Features Generated by the Words Before and After Name Feature Template with the sentence "Billy hit the ball"

The generated features, when combined with a plethora of training data, allow us to generate a very descriptive feature vector for every sentence-event pair. The next step is to train the system to learn how strongly each feature tends to correlate with $e \in DISCUSSED(s)$.

For training, I use the Weka Data Mining Software [7] from the University of Waikato. In particular, I used the C4.5 algorithm [8] (named J48 in Weka) to build a decision tree. The system is then trained with examples of (s, e) pairs, by specifying whether or not $e \in DISCUSSED(s)$ with a 1 if it is, and 0 otherwise. The tree is generated by taking a subset of the trained pairs and repeatedly training on this subset, while testing on the rest of the trained examples. The result is a tree-based classifier capable of taking a feature vector, $\Phi_{s,e}$, and intelligently guessing if $e \in DISCUSSED(s)$. One sample tree is shown in Figure 7 below.

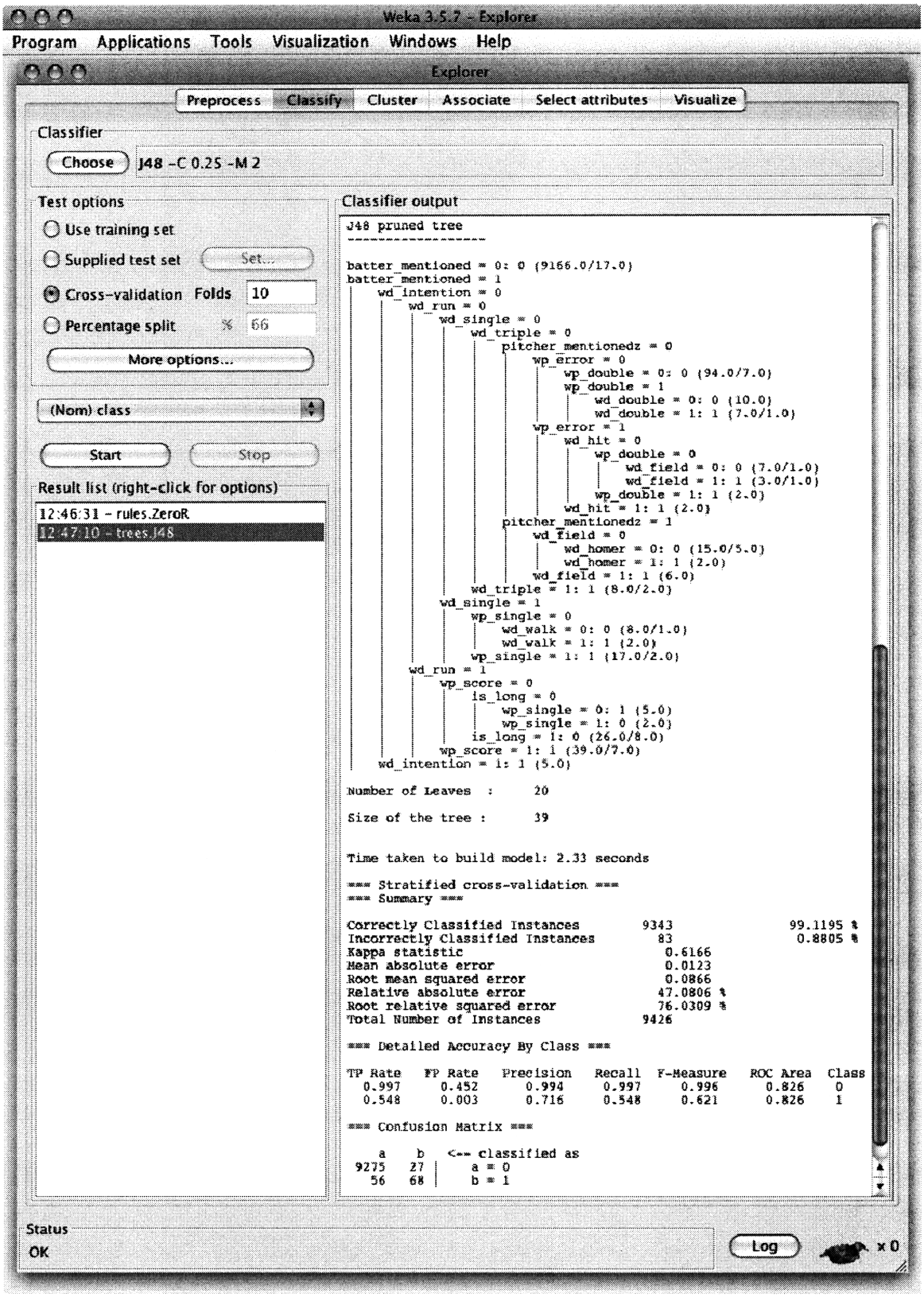


Figure 7 - Sample Weka Classifier Tree

The overall system is diagrammed in Figure 8 below.

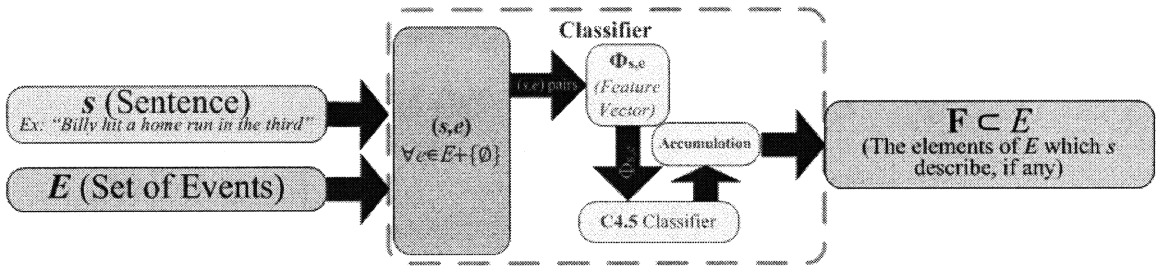


Figure 8 - One-Stage Diagram

The main problem with this system is that $e \notin DISCUSSED(s)$ is the case for the vast majority of all possible (*s, e*) pairs. Out of thousands of (*s, e*) pairs, $e \in DISCUSSED(s)$ holds in only about 20 pairs. Thus, the classifier often gets its best result by building a tree which guesses $e \notin DISCUSSED(s)$ for the vast majority of (*s, e*) pairs, missing the few times that $e \in DISCUSSED(s)$. This problem necessitated the creation of the two-stage system described in the next section.

3.2 Two-Stage System

One way to alleviate the sparse data problem of the one-stage system is to reduce the number of (*s, e*) pairs where $e \notin DISCUSSED(s)$ in the training and testing data. We can do this by reducing the number of (*s, e*) pairs that we train on, automatically eliminating pairs where we know $e \notin DISCUSSED(s)$.

Thus, for the two-stage system, every (*s, e*) pair where it seems that $DISCUSSED(s) = \{\}$ is discarded. In other words, any sentence that is assumed to not talk about any specific event in the document is thrown out. After this, the remaining

sentences go through the same process as the one-stage system described in the previous section. Figure 9 below diagrams the two-stage system.

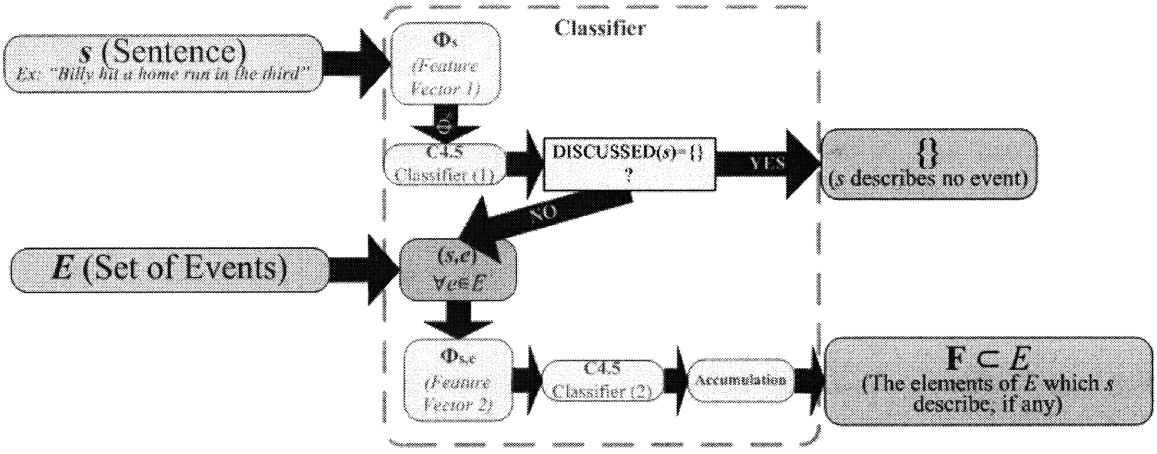


Figure 9 - Two-Stage Diagram

Now, rather than thousands of (*s, e*) pairs, there are hundreds, alleviating the major problem with the one-stage system. However, the results were still not where I wanted them to be. This led to the creation of the system outlined in the next chapter.

Chapter 4 – Translation-Like Model

The shortcomings of the one and two-stage feature-based models led me to search for a new way to classify sentences. I decided to look at the problem in different ways, and realized that it shared some similarities with the problem of machine translation (MT). The goal of MT is to translate from one natural language to another – an example of an MT system is one that is able to take sentences from French and translate them to English. The goal of this project can be thought of as translating from a natural language to a different, structured language.

When looking at the problem from this perspective, other similarities to MT appeared. Take the following MT problem: you are translating a French sentence to English, and are trying to find the probability that French word w_f translates to English word w_e, which is written as $P(w_e \mid w_f)$. If you already have a model of translation the opposite way (from English to French) and a decent model of English word usage frequencies, which is sometimes the case, it would be helpful to use Bayes' Rule and model the probability as:

$$P(w_e \mid w_f) = \frac{P(w_e)P(w_f \mid w_e)}{P(w_f)} = P(w_e)P(w_f \mid w_e)$$

Equation 1 – MT Word Probabilities

Similarly, when looking at the problem of finding the events that correlate with a particular sentence, we can relate the problem in terms of MT. The summary, *S*, is analogous to the French sentence we are trying to translate. Each sentence, *s*, is analogous to a particular French word, w_f. Each event, *e*, is analogous to an English word, w_e. It follows that if we want to find the probability $P(e \in DISCUSSED(s))$, which can be written as $P(e \mid s)$, we could instead find:

$$P(e)P(s \mid e)$$

Equation 2 - Translation-Like System Sentence Probability

That is, we can model the probability of a sentence *s* discussing a given event *e* as the probability of *e* being discussed, $P(e)$, multiplied by the probability of *s* being mentioned if *e* is discussed, $P(s \mid e)$.[4]

[4] With an infinite number of ways to describe *e* with sentences, it might initially appear that $P(s \mid e) = 0$. However, we are looking at each sentence *s* as part of a whole summary, *S*, limiting the number of possible sentences to |*S*|, which allows the possibility of $P(s \mid e)$ being greater than zero.

Equation 2 is crucial - it is the basis for the "translation-like" system described in the following sections. Figure 10 below diagrams the structure of the translation-like system.

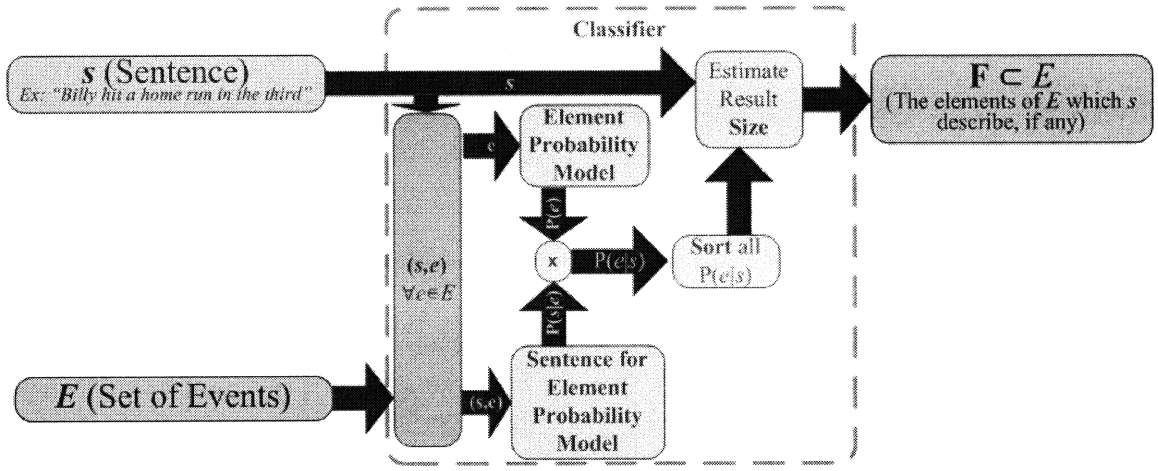


Figure 10 – Translation-Like Model Diagram

As shown in Figure 10, there are three main components that must be created. The "Element Probability Model" will find $P(e)\forall e \in E$. The "Sentence for Element Probability Model" will find $P(s \mid e)\forall(s,e) \in S \times E$. By multiplying these two results for any (s, e) pair, we get $P(e \mid s)$ for every sentence. Next, we can sort $P(e \mid s)\forall e \in E$ from highest to lowest probability. Finally, the third component will decide how many events s is discussing, and take that many elements from the front of the sorted list. Unlike most translation systems, in which most words in French are mapped to an English word, and vice-versa, most sentences are not mapped to any events. This is what necessitates this component. All three of these components are described fully in the following three sections. The last section of this chapter will briefly describe the interconnection between these components.

4.1 Element Probability Model

The element probability model is the most straightforward of the three main components in the translation model. At its root, it finds $P(e)$ using a simple count with a few improvements. First, event types are split up by characteristics. Each unique set of characteristics is denoted as a *type* of event, or type(e). Then, for each type of event, a probability is roughly generated as:

$$\frac{\text{count}\big(\text{type}(e) \text{ where } (\exists s \text{ s.t. } e \in DISCUSSED(s))\big)}{\text{count}(\text{type}(e))}$$

Equation 3 - Approximate Typed Event Probability

That is, the total amount of events which are in *DISCUSSED* for any sentence over the total amount of sentences of that types.[5] This is not the complete equation - there are more factors, as discussed in the "Add-One Smoothing" section below. But first, the methodology for splitting events into types based on features is discussed.

4.1.1 Event Features

In order to be able to split events into types, we must first find features of events which distinguish it from other events. However, we don't want too *many* event types – as the number of types of events increases, we approach a system which would predict that $P(e) = 0$ for nearly any e it hasn't yet encountered, or which has not been discussed yet. We also don't want too *few* types of events, or it will reduce the effectiveness of this

---

[5] An alternative (and problematic) way to define $P(e)$ is as $\frac{\text{count}\big(\text{type}(e) \text{ where } (\exists s \text{ s.t. } e \in DISCUSSED(s))\big)}{\text{count}\big(e \text{ where } (\exists s \text{ s.t. } e \in DISCUSSED(s))\big)}$. That is, by dividing by the total number of discussed events of any type, rather than the total number of that type. The problem with this definition is that there are some events which are rare, but always discussed when they happen – for instance, a grand slam (when 4 runs are scored at the same time). If a grand slam, which happens very infrequently, is discussed every time one happens, we want $P(e)$ to be near 1 if e is a grand slam. This is why we set $P(e) = \frac{\text{count}\big(\text{type}(e) \text{ where } (\exists s \text{ s.t. } e \in DISCUSSED(s))\big)}{\text{count}(\text{type}(e))}$.

method. For instance, if we placed each event into a single type, we would end up with $P(e) = \frac{\text{count}(e \text{ where } \exists s \text{ s.t. } e \in DISUSSED(s))}{\text{count}(e)}$ for every e, giving a uniform distribution of probabilities. This is not what we want – a home run, for instance, is much more likely to be discussed than a regular hit.

Thus, part of the challenge of devising good features by which to split events is finding the right number of features to distinguish events. In my system, having limited my feature-types to binary features, if $|f|$ is the number of features, the number of event types is $2^{|f|}$.[6] I chose to have five features to describe each event. They are described in Table 7 below.

Feature	Description
Score?	True if there was a score during the event, false otherwise
Hit?	True if the ball was hit during the event, false otherwise
Out?	True if an out was recorded during the event, false otherwise
At Bat?	Only some events are at-bats. Others are "actions", such as a pitcher change, manager ejection, etc. This feature distinguishes between at-bats and actions
Pitcher Change?	True if a pitcher was changed during the event. False otherwise.

Table 7 - Binary Event Categorization Features

Although I have five features, which theoretically allows 32 event classifications, only approximately 10 of these actually show up in baseball play, due to the rules and nature of baseball. Now, given an event e, I can find the feature vector for that event, give it a type, and a probability. However, there is a minor revision to Equation 3 discussed in the next section.

[6] This is only always true if the features are independent. In my system, the features are not independent, but this equation still roughly holds.

4.1.2 Add-One Smoothing

There are many factors which control the number of features which is good to use. The foremost in my mind when deciding to use five features was the number of training examples I provided. The more training examples, the more types of features one is allowed to have, because for any given type of feature, it is more likely that it will have appeared at least once in the training data.

Still, no matter how many training data there are, there is always the possibility of an event whose type was never discussed in training, being discussed on the test data. If we stick to Equation 3, $P(e) = 0$ in this case, which would make $P(s|e) = 0$, which we don't want in the vast majority of cases. Instead, it would be better to have $P(s|e)$ be some small number. To solve this, I use add-one smoothing.

Add-One smoothing is common in NLP, due to the "sparse data" problem.[7] One example of the sparse data problem in NLP is as follows. Suppose you are searching for the probability of a word, w_b, following another word, w_a. If you have a corpus, or a set of texts, the simplest way to do this is by setting the probability to $\frac{\#\text{ of times } w_b \text{ follows } w_a}{\#\text{ of times } w_a \text{ appears}}$. However, if w_b never follows w_a in the corpus (there are, after all, many possible two-word combinations), this probability will be 0, which we don't want. A simple way of correcting this is to simply add 1 to the numerator of each probability. In order to maintain a proper probability (where everything adds to 1) a factor must also be added onto the denominator. After add-one smoothing, the resulting equation for $P(e)$ is:

[7] This is distinct from the problem with sparse data that the feature-based model had.

$$P(e) = \frac{1 + \text{count}\left(\text{type}(e) \text{ where } \left(\exists s \text{ s.t. } e \in DISCUSSED(s)\right)\right)}{\text{count(type)} + \text{count}\left(\text{type}(e)\right)}$$

Equation 4 – P(e)

Where count(type) is the number of theoretically possible event types, 2^5. Thus, add-one smoothing is a simple way to smooth out the probabilities and be prepared for new event types outside of the training data. It should be noted, however, that for the particular example in NLP used above, better alternatives exist [9]. However, the sparse data problem is much less of an issue in our system than it is in NLP, so add-one smoothing is enough.

4.2 Sentence for Element Probability Model

The system which determines $P(s|e)$ is discussed in this section. It is much more complex than the $P(e)$ system discussed in the previous section, out of necessity – natural language work is much harder than work with structured elements. There are many components in the $P(s|e)$ model. Each component returns a score. The sum of all of these scores is taken, then divided to form a proper probability. Each of these components are described in the sections below, after which, their interconnection is discussed.

4.2.1 Word Stemming

There are many words which are very telling in linking sentences and events. Take the following example of a possible event-sentence pair:

Event Description	Sentence
David Ortiz doubles (1) on a line drive to right fielder Gary Sheffield.	It was no surprise that it was David Ortiz -- the certified Yankee destroyer -- who got things rolling with a leadoff double to right.

Table 8 - Possible Event-Sentence Pair

To us, it is apparent that $e \in DISCUSSED(s)$. There are many clues – the batter's name is the same, and both mention hitting doubles to right field. However, the event description uses the active voice ("David Ortiz doubles") while the sentence uses the passive voice ("got things rolling with a leadoff double"). Although the root word ("double") is the same in both cases, they are used in very different senses.

This is often the case in matching summaries and events. Thus, we need a general way to stem words. For this, I used the Porter Stemmer [10] provided with the Python Natural Language Toolkit. When provided with the words "double", "doubles", or "doubled", the Porter stemmer produces the stem "doubl". Although an in depth definition of the stemmer is outside of the scope of this paper, the original paper describing the algorithm can be found in the references.

4.2.2 Stem similarities

With the Porter stemmer, we can find the stem of every word in the sentences and event descriptions. We can see from Table 8 that some common stems seem to be significant, like "doubl" or "right" (as in "right field"). However, other shared stems are insignificant. For example, "a" and "to" are also shared stems, but don't contribute much. How can we automatically measure the significance of stems that are shared by event descriptions and sentences?

First, we have to look at the common words in the set of training examples. For each common stem, c_s, we assign a "co-occurrence probability" of $\frac{\text{count}(c_s \text{ in } e \text{ and } c_s \text{ in } s \text{ and } e \in DISCUSSED(s))}{\text{count}(c_s \text{ in } e \text{ where } \exists s \text{ s.t. } e \in DISCUSSED(s))}$. Now, given any potential (*s, e*) pair, the sum of

the co-occurrence probability of every stem is added for every common stem.[8] In addition, for every stem in e, but not in s, the co-occurrence is subtracted from the score. Because we are searching for $P(s|e)$, the co-occurrence score is found for every potential s. This means that the lowest score is usually negative. If we call l the lowest score, then (l+1) is added to each score to ensure that all resulting scores are positive.

4.2.3 Named Entity Recognition

In the introduction, I mentioned that one of the benefits of working with baseball is that it is a domain which doesn't require too much context. To contrast, look at any given newspaper article. It will likely mention named entities such as corporations, people, or places without giving much context. By contrast, the vast majority of named entities mentioned in a baseball game are players, teams, or locations which are mentioned in the XML document.

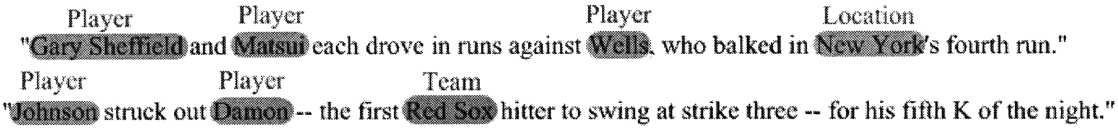


Figure 11 - Named Entity Recognition

Recognizing mentioned players is very useful. One of the most telling features of events is the names of the pitcher and batter. For each event, the pitcher and batter are mentioned. Often, sentences which discuss specific events will mention the pitcher or batter. Thus, whenever reading a baseball XML document (like the one shown in Appendix A), the player names are stored. This way, when reading a summary, player

[8] The probabilities are added, to form a score, rather than multiplied for various reasons. First of all, there is almost guaranteed to be a 0 co-occurrence probability for every potential pair, which would always render the result as zero. Secondly, this component is meant to output a score, rather than a real probability, at first. The scores will later be mapped into probabilities.

names can be easily identified. This is done for each player, by searching for and tagging any times where the full name (first and last) is mentioned, after which the players list is traversed again, and the summary is tagged by the last name.

Unlike the players, cities and teams (of which there are 30 in the MLB) were simply entered manually. They only come into play as a factor when tagging sentences, which is discussed in the "Result Size Predictors" section.

4.2.4 Inning Matches

One potentially very telling feature of any potential (*s*, *e*) pair is whether or not *s* seems to be discussing the inning which *e* is in. Thus, the system has a mechanism to deduce which innings any particular sentence might be discussing. This can be thought of as one way of increasing the context-awareness of the system. One consistent pattern across summaries which can be exploited is the tendency to use phrases like "in the first", or "in the bottom of the second".

This system takes advantage of this pattern by searching for manually entered phrases such as "the first", and "the second" as part of the sentence. Sometimes, however, the phrase "the first", or similar phrases are mentioned in sentences well before any specific game event is discussed. Take the summary in Figure 12 – the first sentence ("Johnson came out dealing in the first inning, …") lets us know that the rest of the paragraph is discussing something that happened in the first inning.

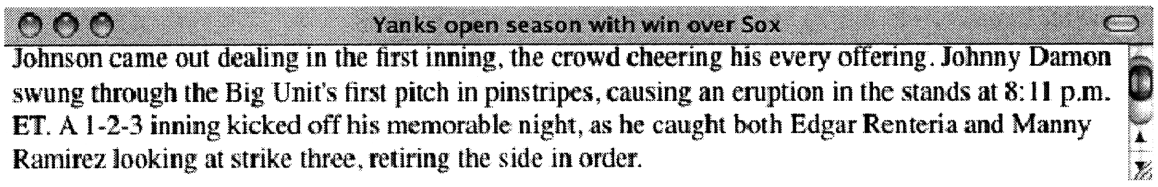


Figure 12 - Inning Inference

Thus, in order to deduce which inning a particular sentence is discussing, it is not enough to simply search for phrases like "the first", etc. It is also necessary to look back at previous sentences. Put another way, in addition to searching for these phrases in sentence $s_i \in S$, we also have to search through s_{i-1}, s_{i-2}, etc. until we find a clue as to which inning is being discussed (if any).

4.2.5 Final Scoring

The final score of a potential (s,e) pair is a linear combination of several other scores. These scores are listed in Table 9 below.

Name	Description	Possible Values
$v_{pitcher}$	1 if s mentions the pitcher of e, 0 otherwise	{0,1}
v_{batter}	1 if s mentions the batter of e, 0 otherwise	{0,1}
$v_{other\_players}$	The number of players mentioned in e that are also mentioned in s	{0,1,2,…}
v_{stems}	The score from stem similarities, as described in 4.2.2	{0,1,2,…}
v_{inning_match}	1 if s seems to be discussing the same inning that e is in, 0 otherwise	{0,1}

Table 9 - Scoring for P(s|e)

Thus, the final score for each potential (s,e) pair is:

$$score_{s,e} = (\rho_{pitcher} * v_{pitcher}) + (\rho_{batter} * v_{batter}) + (\rho_{other\_players} * v_{other\_players}) + (\rho_{stems} * v_{stems}) + (\rho_{inning\_match} * v_{inning\_match})$$

Equation 5 - Final Score for P(s|e)

Where each ρ represents a multiplier correlating with the significance of that particular score. If, for example, v_{batter} is found to be particularly significant in predicting if $e \in DISCUSSED(s)$, then ρ_{batter} will be a large number. These multipliers are optimized automatically, as described in 4.4.1. Finally, for each event e, $P(s|e)$ is defined as:

$$P(s|e) = \frac{score_{s,e}}{\sum_{s'} score_{s',e}}$$

Equation 6 - P(s|e)

4.3 Result Size Predictors

Now that we have models to find $P(e)$ and $P(s|e)$ for any potential (s, e) pair, we can measure $P(e|s)$ and accurately rank the potential of every event to be discussed in any particular sentence. The next step is to decide where to cut this list off and decide exactly which events are discussed by a sentence.

4.3.1 Probable Contexts

Although there are multiple authors in each summary, when reading a few of the summaries, a pattern seems to appear. The first and last parts of the summary almost invariably discuss things outside of the context of the events of the game. As a result, an estimate of the probability of any sentence discussing a particular event can be improved by taking into account *where* that sentence is in the summary.

This idea can be generalized by thinking in terms of contexts. In any natural language document, contexts often shift throughout the document. In some cases, patterns might emerge. A function capable of estimating this pattern is very useful in estimating the number of events which are in $DISCUSSED(s)$ for any sentence s.

For this system, there are only two contexts we care about – "pertaining to game events" and "not pertaining to game events". Thus, one way to generate a function describing the probability of a sentence describing any particular event is, for every training summary S, for each sentence, s, in that summary, measure where s is as a

fraction of $|S|$ and assign that to x (for example, the first sentence will have $x = \frac{1}{|S|}$, the second $x = \frac{2}{|S|}$, the last $x = 1$, and so on). Now, for each s, set y to $|DISCUSSED(s)|$. All of these data points are combined for each training example. The result is shown in Figure 13.

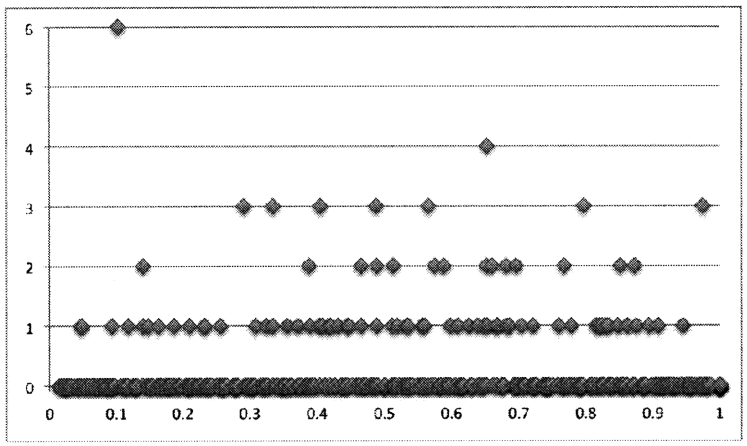


Figure 13 - Context Data Points (Sentence Location vs. |*DISCUSSED*|)

The next step is to find a function to fit these data points. This is done with linear regression. When choosing the number of factors to use with linear regression, it is important to not choose too many, which would over fit the data, or to few, which would not be very descriptive of the data. I chose to use five factors. The results are shown in Figure 14 below.

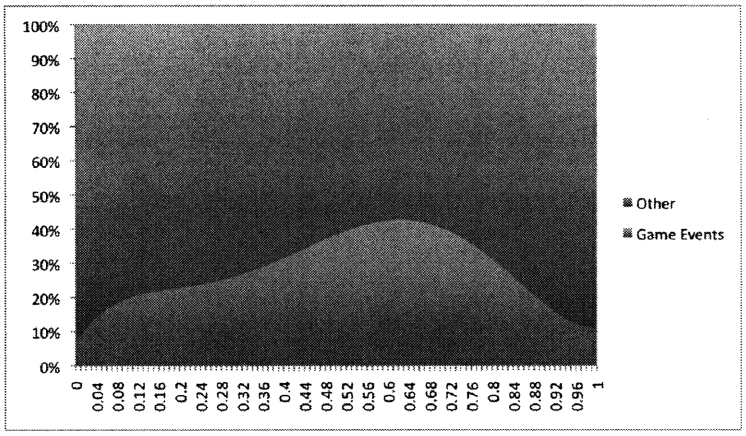


Figure 14 - Probable Context Graph (by Sentence Location)

As Figure 14 shows, a summary is most likely to discuss specific game events about 65% of the way through the summary, and rarely discusses specific events in the beginning or end of the summary.

4.3.2 Part of Speech Tagging

It is often very useful to figure out the subjects, verbs, and objects of a particular sentence to reason about how many events it is discussing. In order to do this, we have to figure out and tag the part of speech in every word of the summary.

As mentioned in the "Named Entity Recognition" section, players, teams, and cities are already recognized, which takes care of tagging for most proper nouns that appear in the summary. For the rest, I used a bigram tagger, with a unigram tagging back off, trained on the Brown Corpus (from the Python NLTK). Some examples of tagged sentences are shown in Figure 15[9].

[9] Note that there are still some inaccurate labels, in part due to the style with which baseball terminology is used. For example, "left" is labeled as a verb in the first sentence of Figure 15 while it should be an adjective.

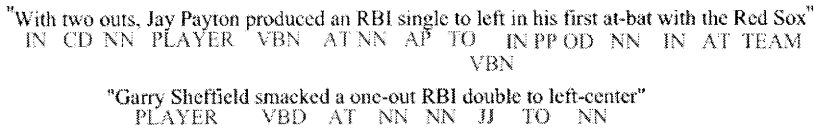


Figure 15 - Tagged Sentences

When training the classifier, the stem of each verb is then mapped to the average size of $DISCUSSED(s)$ for every training sentence s which has a player as either the subject or object of that verb.

4.3.3 Final Result Size Estimation

Just like the result for $P(s|e)$, the final estimation for the size of $DISCUSSED(s)$ is the result of a linear combination of the two systems defined above:

$$|DISCUSSED(s)| = (\rho_c * c) + (\rho_v * v)$$

Equation 7 - |DISCUSSED(s)|

Where c is the value of the function shown in Figure 14 and v is the average mapped to the stem of the verb in s, as mentioned in 4.3.2 above. In addition, the value of Equation 7 is rounded to the nearest non-negative integer to give an unambiguous value as the result.

4.4 Putting it Together

Now, we have the system roughly described in Figure 10. As is shown in Figure 10, when a sentence, s, and a set of events, E, are fed into the system, $P(e|s) \forall e \in E$ is deduced by taking $P(e) * P(s|e)$. Each event is then sorted by $P(e|s)$ from highest to lowest. Then, the system described in 4.3 which estimates the size of $|DISCUSSED(s)|$ is run, and the first $|DISCUSSED(s)|$ elements from the sorted events are returned. However, there are a few more steps.

4.4.1 Parameter Optimization

First, recall that in the preceding section, there were several ρ values which were unspecified multipliers. The method by which these parameters are set is described in the following section.

Each parameter, ρ, of the system is optimized individually. First, the parameter is set arbitrarily to 0. Next, the following algorithm is run:

```
for i=0,1,2,3:
      inc = 1/10^i
      best_offset = 0
      for offset = -10*inc,-9*inc,…,9*inc,10*inc:
            if score(ρ+offset) > score(ρ+best_offset):
                  best_offset = offset
      ρ = ρ+best_offset
```

Figure 16 - Parameter Optimization Algorithm

Where `score` is the function we are trying to maximize (in this case, the accuracy of the predictor). This algorithm is run repeatedly parameter by parameter, and repeated to find the optimal value for each ρ to within three decimal places.

4.4.2 Performance Optimizations

To maximize the performance and usability of the system, the events mapped to each sentence are predicted ahead of time. There are also various other optimizations to the algorithm shown in Figure 16 to allow the parameters to be optimized quickly, such as breaking out of the loop when it appears a particular ρ is at its optimal value.

Chapter 5 – Results

In order to test my system, I completed event-matching for 20 summaries. I then test by cross-validating my training examples, meaning I train my classifier on about 20 games,

and then test the results on the remaining 10 (although cross-validation is done automatically in the Feature-Based system by Weka). The *precision* and *recall* of the system are then measured for positive and negative examples. For example, with positive training data in the one-stage system, precision is the fraction of guessed events, e' that are actually in $DISCUSSED(s)$. The recall is the fraction of $DISCUSSED(s)$ which is in the set of guessed events. From the precision and recall, an F-score[11] is generated. The F-score is defined as:

$$\frac{2 * precision * recall}{precision + recall}$$

Equation 8 - F-Score

An F-Score is generated for the positive and negative examples, and these two F-Scores are weighted according to the number of examples. The accuracy of the system is measured from the composite F-Score it generates. In addition, it is the `score` mentioned in Figure 16 for which the parameters are optimized.

5.1 Feature-Based System

One Stage:

The one stage system, as mentioned before suffers from the problem that the vast majority of the training data are negative examples. This results in the classifier rarely classifying potential (s,e) pairs as positive. As a result, the precision and recall of this system on negative examples ($e \notin DISCUSSED(s)$) are fairly high – on the rare occasion when it does guess a positive (s,e) pair, it is usually right. However, the recall is poor on positive examples ($e \notin DISCUSSED(s)$) because most positive (s,e) pairs are missed. These are reflected in the results below.

Type	Count	Precision	Recall	F-Score
$e \notin DISCUSSED(s)$	9,302	0.994	0.997	0.996
$e \in DISCUSSED(s)$	124	0.716	0.548	0.621
Total	9,426	0.990	0.991	0.990

Table 10 - One-Stage System Results

Two Stage:

Recall the two parts of the two stage system. The first stage determines, for each s, whether or not s discusses any game events (whether or not $DISCUSSED(s) = \{\}$). The precisions, recalls, and F-Scores of that stage are shown in Table 11.

Type	Count	Precision	Recall	F-Score
$DISCUSSED(s) = \{\}$	455	0.932	0.884	0.932
$DISCUSSED(s) \neq \{\}$	91	0.689	0.560	0.618
Total	546	0.892	0.884	0.888

Table 11 - Two-Stage System Results – Stage One

After the sentences which don't discuss any events are ruled out, a system almost identical to the one-stage system is run. Because this system is not trained on as many negative examples as the one-stage system, however, the precision and recall for negative examples ($e \notin DISCUSSED(s)$) are improved, although positive results decrease.

Type	Count	Precision	Recall	F-Score
$e \notin DISCUSSED(s)$	7,799	0.992	0.995	0.993
$e \in DISCUSSED(s)$	135	0.725	0.612	0.688
Total	7,934	0.988	0.988	0.988

Table 12 - Two-Stage System Results – Stage Two

Finally, the two stages are put together to form a single system. The final results of the two-stage system are shown in Table 13. As the data shows, although the score is roughly equivalent to the one-stage results, the performance on positive examples, where $e \in DISCUSSED(s)$, is greatly improved.

Type	Count	Precision	Recall	F-Score
$e \notin DISCUSSED(s)$	9,436	0.993	0.996	0.994
$e \in DISCUSSED(s)$	135	0.725	0.612	0.688
Total	9,571	0.989	0.991	0.990

Table 13 - Two-Stage System Results – Aggregate

5.2 Translation-Like System

The translation-like system, discussed in chapter 4, is more versatile and accurate than the feature-based system. Its results are shown in Table 14 below. Due to limitations of the initial implementation of the feature-based system, the translation-like system had to be trained and tested on different training data than the two feature-based systems. Still, it was trained and tested on the same number of games and in the same way as the previous two systems.

Type	Count	Precision	Recall	F-Score
	8,783	0.997	0.998	0.997
	118	0.780	0.701	0.738
Total	8,901	0.994	0.994	0.994

Table 14 - Translation-Like System Results

The "bottleneck" of this system's performance is the estimation of the number of events discussed ($|DISCUSSED(s)|$).

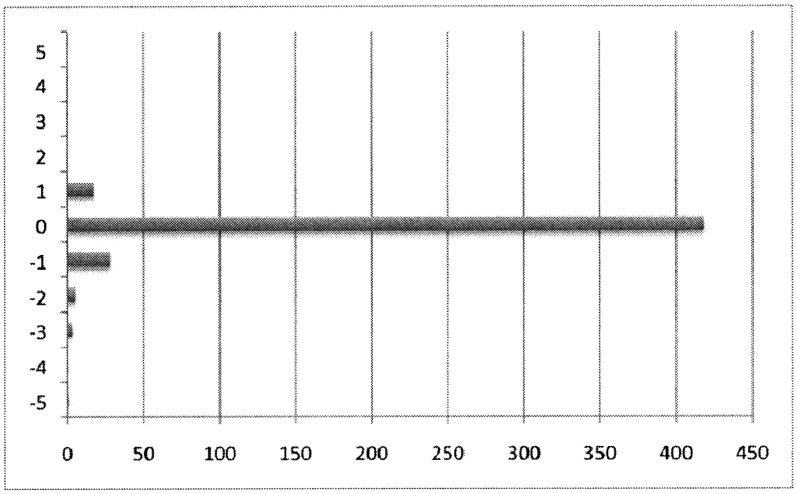


Figure 17 - Error in Estimating Number of Events Discussed (x Axis is actual – estimated, y Axis is count)

Improving this portion of the system would be the first step in improving the system performance as a whole.

Chapter 6 – Conclusions

There were several limitations of strictly feature-based systems which made the translation-like system necessary. First, it can be the difficult to give the feature-based system the ability to compare and rank different possible events – most learning algorithms and trees are only capable of giving a classification, rather than a probability. In addition, it is very difficult to create features which are interdependent with other features or classifications, without exponentially increasing the number of features used, which can be problematic.

The translation-like model helps alleviate some of these problems, but some parts require somewhat problem-specific and specialized solutions, such as the factor added whenever an inning of a particular sentence seems to match that of an event. However, as mentioned in the Future Work chapter, there are many components can be generalized so that they may be generated automatically for any problem type.

Chapter 7 – Future Work

This thesis addresses what I feel is a very interesting problem which warrants future research. One thing that would likely be very helpful in improving the accuracy of the system is to improve the component which estimates the size of results. There are many ways to do this. For example, the sentence structure could be analyzed on a deeper level

with chunking, allowing the core elements of any sentence to be reasoned about. In addition, sentences could be further simplified by finding synonyms for words using simple dictionary services like WordNet. [12] This would be the most immediate way of improving the performance of the system, as shown in the "Results" chapter. In addition, if we could give the system more context awareness in the summary, its performance might be improved.

Another interesting potential project is to generalize some of the elements which were coded in. For example, features (for translation and feature-based models) which could be automatically generated would make the system more applicable to many more domains. The context graphs could also be generalized to improve performance and versatility.

Another interesting direction to take this research would be to extend it beyond XML documents and into other kinds of structured documents and elements, similar what Chickenfoot [4] and a few other projects have done for allowing programming languages to become more natural.

Chapter 8 – Bibliography

[1] D. Shahaf, E. Amir. *Towards a Theory of AI Completeness.* Association for the Advancement of Artificial Intelligence Spring Symposium Series, 2007

[2] M. Fleischman, D. Roy. *Grounded Language Modeling for Automatic Speech Recognition of Sports Video.* Human Language Technologies - North American Chapter of the Association for Computational Linguistics, 2008

[3] B. Snyder, R. Barzilay. *Database-Text Alignment via Structured Multilabel Classification.* International Joint Conference on Artificial Intelligence, 2007

[4] M. Bolin, M. Webber, P. Rha, T. Wilson, R. Miller *Automation and Customization of Rendered Web Pages.* ACM Conference on User Interface Software and Technology (UIST), 2005

[5] T. Kiss, J. Strunk. *Unsupervised Multilingual Sentence Boundary Detection.* Computational Linguistics, 2006, 32(4), 485-525

[6] S. Bird, E. Loper. *NLTK: The Natural Language Toolkit*. Proceedings of the Association for Computational Linguistics Demonstration Session, 2004, 214-217

[7] I. Witten, E. Frank *"Data Mining: Practical machine learning tools and techniques",* 2nd Edition. Morgan Kaufmann, San Francisco, 2005.

[8] R. Quinlan. *Data Mining from an Al Perspective*. 15th International Conference on Data Mining, Issue 23-26, 1999 p.186

[9] J. Zavrel, W. Daelemans. *Memory-based learning: using similarity for smoothing.* European Chapter Meeting of the ACL, Proceedings of the 8th conference on European chapter of the Association for Computational Linguistics, 1997 p. 436 – 443

[10] C.J. Rijsbergen, S.E. Robertson, and M.F. Porter. *New models in Probabilistic Information Retrieval.* London: British Library. (British Library Research and Development Report, no. 5587). 1980

[11] J. Makhoul, F. Kubala, R. Schwartz, R. Weischedel. *Performance measures for information extraction.* Proceedings of DARPA Broadcast News Workshop, Herndon, VA, 1999

[12] C. Fellbaum et. al. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge MA. 1998

Appendix A – Sample Training Data

Below is a sample training game, with most elements cut out to save space.

```
<test_data>
	<teams>
		<away abbrev="BOS" code="bos" losses="1" name="Boston" wins="0"/>
		<home abbrev="NYY" code="nya" losses="0" name="NY Yankees" wins="1"/>
	</teams>
	<players>
		<away>
			<player box_name="Machado" first_name="Alejandro" id="425472" last_name="Machado"/>
			<player box_name="Nixon" first_name="Trot" id="119811" last_name="Nixon"/>
			<player box_name="Damon" first_name="Johnny" id="113028" last_name="Damon"/>
			<!-Other Players cut to save space-->
		</away>
		<home>
			<player box_name="Williams" first_name="Bernie" id="124288" last_name="Williams"/>
			<player box_name="Rodriguez, A" first_name="Alex" id="121347" last_name="Rodriguez"/>
			<!-Other Players cut to save space-->
		</home>
	</players>
	<game day="3" location_city="New York" location_state="NY" month="4" stadium="Yankee Stadium" time="08:05 PM"
year="2005">
		<stats>
			<away errors="2" hits="6" runs="2"/>
			<home errors="1" hits="15" runs="9"/>
		</stats>
		<innings count="9">
			<inning num="1" /> <!-- (CUT) -->
			<inning num="2">
				<top runs="1"> <!-Most Plays cut to save space-->
					<at_bat balls="1" batter="120074" des="David Ortiz doubles (1) on a line drive to
right fielder Gary Sheffield." outs="0" pitcher="116615" strikes="0">
						<pitches>
							<pitch des="Ball"/>
							<pitch des="In play, no out recorded "/>
						</pitches>
						<hit des="Double" type="H"/>
					</at_bat>
					<at_bat balls="1" batter="132788" des="Kevin Millar flies out to left fielder
Hideki Matsui." outs="1" pitcher="116615" strikes="1">
						<pitches>
							<pitch des="Foul"/>
							<pitch des="Ball"/>
							<pitch des="In play, out(s) recorded"/>
						</pitches>
						<hit des="Fly Out" type="O"/>
					</at_bat>
					<at_bat balls="1" batter="123660" des="Jason Varitek grounds out, shortstop Derek
Jeter to first baseman Jason Giambi. David Ortiz to 3rd." outs="2" pitcher="116615" strikes="1">
						<pitches>
							<pitch des="Called Strike"/>
							<pitch des="Blocked Ball in Dirt"/>
							<pitch des="In play, out(s) recorded"/>
						</pitches>
						<hit des="Ground Out" type="O"/>
					</at_bat>
					<at_bat balls="1" batter="134341" des="Jay Payton singles on a line drive to left
fielder Hideki Matsui. David Ortiz scores." outs="2" pitcher="116615" strikes="2">
						<pitches>
							<pitch des="Ball"/>
							<pitch des="Foul"/>
							<pitch des="Foul"/>
							<pitch des="In play, run-scoring play"/>
						</pitches>
						<hit des="Single" type="H"/>
						<score away_score="1" home_score="0" points="1"/>
					</at_bat> <!-Most Plays cut to save space-->
					<at_bat balls="1" batter="110840" des="Mark Bellhorn called out on strikes."
outs="3" pitcher="116615" strikes="3">
						<pitches>
							<pitch des="Swinging Strike"/>
							<pitch des="Ball"/>
							<pitch des="Called Strike"/>
							<pitch des="Called Strike"/>
```

```
                                        </pitches>
                                </at_bat>
                        </top>
                        <bottom runs="1">
                                <at_bat balls="3" batter="425686" des="Hideki Matsui singles on a fly ball to
left fielder Manny Ramirez." outs="0" pitcher="124071" strikes="2">
                                        <pitches>
                                                <pitch des="Swinging Strike"/>
                                                <pitch des="Called Strike"/>
                                                <pitch des="Ball"/>
                                                <pitch des="Ball"/>
                                                <pitch des="Foul"/>
                                                <pitch des="Ball"/>
                                                <pitch des="In play, no out recorded "/>
                                        </pitches>
                                        <hit des="Single" type="H"/>
                                </at_bat>
                                <at_bat balls="0" batter="120691" des="Jorge Posada flies out to right fielder
Jay Payton." outs="1" pitcher="124071" strikes="1">
                                        <pitches>
                                                <pitch des="Called Strike"/>
                                                <pitch des="In play, out(s) recorded"/>
                                        </pitches>
                                        <hit des="Fly Out" type="O"/>
                                </at_bat>
                                <at_bat balls="1" batter="114739" des="Jason Giambi singles on a ground ball to
right fielder Jay Payton. Hideki Matsui to 3rd." outs="1" pitcher="124071" strikes="1">
                                        <pitches>
                                                <pitch des="Ball"/>
                                                <pitch des="Foul"/>
                                                <pitch des="In play, no out recorded "/>
                                        </pitches>
                                        <hit des="Single" type="H"/>
                                </at_bat>
                                <at_bat balls="1" batter="124288" des="Bernie Williams out on a sacrifice fly to
left fielder Manny Ramirez. Hideki Matsui scores." outs="2" pitcher="124071" strikes="2">
                                        <pitches>
                                                <pitch des="Called Strike"/>
                                                <pitch des="Ball"/>
                                                <pitch des="Foul Tip"/>
                                                <pitch des="In play, run-scoring play"/>
                                        </pitches>
                                        <hit des="Fly Out" type="O"/>
                                        <score away_score="1" home_score="1" points="1"/>
                                </at_bat>
                                <at_bat balls="0" batter="124523" des="Tony Womack grounds into a force out,
shortstop Edgar Renteria to second baseman Mark Bellhorn. Jason Giambi out at 2nd." outs="3" pitcher="124071"
strikes="1">
                                        <pitches>
                                                <pitch des="Called Strike"/>
                                                <pitch des="In play, out(s) recorded"/>
                                        </pitches>
                                        <hit des="Ground Out" type="O"/>
                                </at_bat>
                        </bottom>
                </inning>
                <inning num="9" /> <!-- (CUT) -->
        </innings>
    </game>
    <sentences summary_name="MLB_BOS"><!-Most Sentences cut to save space-->
        <sentence contents="For weeks, perhaps months, Red Sox manager Terry Francona had been telling anyone
who would listen that the historic magic of last year won't win a single game for the 2005 Red Sox." state="complete"/>
        <sentence contents="He now has a perfect case in point, as the defending World Series champions opened
their title defense by being soundly beaten, 9-2, by the Yankees in Sunday's Opening Night game at Yankee Stadium."
state="complete"/>
        <sentence contents="The Yankees had no trouble shaking away the bad memories from last October, when the
Red Sox became the first team in baseball history to overcome a 3-0 deficit in a postseason series." state="complete"/>
        <sentence contents="Both of the rivals have added new faces that will give this year an identity of its
own." state="complete"/>
        <sentence contents="By no means are we deflated." state="complete"/>
        <sentence contents="&amp;quot; The Sox temporarily derailed the momentum of Johnson's Big Apple unveiling by
rallying for a run in the second." state="complete">
                <match xpath="/test_data/game/innings/inning[@num='2']/top/*[4]"/>
        </sentence>
        <sentence contents="It was no surprise that it was David Ortiz -- the certified Yankee destroyer -- who
got things rolling with a leadoff double to right." state="complete">
                <match xpath="/test_data/game/innings/inning[@num='2']/top/*[1]"/>
        </sentence>
        <sentence contents="It looked as if the Sox were about to take a 2-0 lead when the next batter, Kevin
Millar, ripped a deep drive to left." state="complete">
                <match xpath="/test_data/game/innings/inning[@num='2']/top/*[2]"/>
        </sentence>
</test_data>
```