EXTRACTING AND NORMALIZING TEMPORAL EXPRESSIONS

Lois C. Childs, David Cassel

Lockheed Martin Corporation

P.O. Box 8048
Philadelphia, PA 19101

lois.childs@lmco.com

(610) 354-5816

1. INTRODUCTION

As part of our TIPSTER III research program, we have enhanced the NLToolset's[1] capability to extract temporal expressions from free text and convert them into canonical form for accurate comparison, sorting, and retrieval within a database management system.

The date or time that an event occurs is often a critical piece of information. Unfortunately, natural language expressions that contain this information are so numerous and varied that the interpretation of temporal expressions within free text becomes a challenging task for automatic text processing systems.

This paper will look at the nature of the problem, the extraction and computation tasks, the use of a learning program, and the normalization strategy. The concluding section will discuss possible future endeavors related to time extraction.

The NLToolset

The NLToolset is a framework of tools, techniques, and resources designed for building text processing applications. It is a pattern based system which uses world knowledge resident in a lexicon, a location gazetteer, and lists of universal terms, such as first names and the Fortune 500 companies. This knowledge base is extensible with generic, as well as domain-specific, information. It applies lexico-semantic pattern matching in the form of basic structural patterns (*possible-title firstname middle-initial lastname*), as well as contextual knowledge (*possible-name, who is X years old*). The NLToolset has been applied to routing, indexing, name spotting, information extraction, and document management. It is an object-oriented system, implemented in C++ and ODBC to make it portable to both Unix and NT platforms, as well as multiple databases.

[1] The NLToolset is a proprietary text processing product, owned by Lockheed Martin Corporation.

2. PROBLEM DESCRIPTION

The task of automatically extracting temporal information can be divided into four parts:

1) Recognize the temporal expression.

The event happened <u>*Saturday*</u>.

2) Extract its features.

Saturday is a day name and a relative expression.

3) Compute its interval representation.

Based on the reference date of the document and the features of the expression, determine which calendar day is meant. Represent this as an interval:[2] 08291998 - 08291998.

4) Normalize the interval for database use.

Store each part of the interval expression, i.e., day, month, year for start and end points, into an

[2] For the purpose of this paper, the interval will not address smaller units of time than days, i.e. hours, minutes, and seconds. An interval for a day will have identical endpoints.

NLToolset structure. Final output format varies according to application requirements.

Feature Complexity

The greatest difficulty in building an automatic system for interpreting time expressions is the seemingly infinite variety of ways in which human beings express time.

The term "feature" in this context refers to a category of information that can be used to interpret the expression. For example, the feature "unit of time" refers to the terms *month, day, year, century*; "interval endpoint" refers to an explicit reference to at least one end of a time interval, such as *before the end of* or *from June to September.*

Each of the following numbered examples represents a different kind of time expression, based on the features available for its interpretation.

1. *before the end of the year*
2. *next April*
3. *March 1, 1992*
4. *from June to September*
5. *in the 90's*
6. *in two weeks*
7. *the first year*
8. *beginning July 1*
9. *last Summer*
10. *next month*
11. *in the first quarter of fiscal 1992*
12. *the turn of the century*
13. *Saturday*
14. *yesterday*
15. *the previous April*

Table 1 illustrates the relationship between a set of features and the temporal expressions in which they appear. This is often a many-to-many relationship, which makes the manual construction of a decision tree a formidable task.

Feature Available	Example Number
unit of time	1, 6, 7, 10
interval endpoint	1, 4, 5, 7, 8
relative to dateline	1, 2, 4, 6, 8, 9, 10, 13, 14
month name	2, 3, 4, 8, 15
relative direction	2, 9, 10, 15
day number	3, 8
year	3
decade number	5
ordinal	7
relative to event date	7
season name	9
fiscal year	11
fiscal year unit	11
idiom	12
relative to context	5, 12
day name	13
relative day term	4

Table 1: Feature/Expression Relationships

Each expression will require a unique computation function, based on the features present and their interaction. For example, the second expression, *next April*, is different from *April of next year* only if the reference date is within the interval between January 1 and March 31.

There are many possible combinations of features. Additionally, there are many idiomatic temporal expressions, such as *the turn of the century*. These possibilities must be captured within the NLToolset's rule packages so that the expression can be recognized.

Relative Expressions

Some time expressions are specific, e.g. *March 1, 1992*; others are relative expressions, either of a contiguous or non-contiguous nature. For example, expressions like *yesterday* or *next month* are non-contiguous because they are relative to the dateline of the message. But, expressions like *the previous April* or *the following day* usually refer to the immediately preceding time expression, and thus are thought of as contiguous.

The CEO announced his retirement on March 5. The following day, the company's stock price rose.

In this example, *on March 5* is a non-contiguous expression and is calculated from the document reference date, while *the following day* is contiguous and is calculated using the previous temporal expression, *March 5*, as the reference date.

Also to be factored in as a consideration in relative expressions is the tense of the verb.

The ship sailed on Saturday.

The ship will sail on Saturday.

Computation of the correct interval depends on whether the date is meant to indicate past or future.

Ambiguity

Some expressions are simply meant to be ambiguous, indicating a general vicinity of time, but not meant to be exact. When the expression, *next week*, is used, does that mean the seven days beginning on Sunday, or does it mean the five days of the business week? There definitely is information contained within the expression, but the problem is capturing the information without overstating the accuracy of its representation. The following example further illustrates this point.

Baseball season begins next week.

In this case, what is meant is that the season will begin at some point during the interval that is next week; however, the exact time is ambiguous.

The NLToolset's current implementation will arbitrarily decide what the interval of *next week* is. It will make no attempt to resolve the ambiguity, nor to note that such ambiguity exists. This is an area for future research.

Specialized Calendars

Information extraction systems are often developed for specialized domains. The following examples illustrate the problem of specialized calendars. The first example is from a business domain from which the system must extract information about joint ventures.

Profits during the first year reached $5 million.

In this example, the reference point is the date that the joint venture began operations. This is used to calculate the interval represented by *the first year*.

The second example is from the automotive domain.

Since the 1990 model year began on October 1, Buick sales have plunged.

Introduction of world knowledge to the system would be necessary to have it understand that the start of the model year was in 1989.

The third example might appear in an agricultural domain.

During the current crop year, Brazil will produce 7 million tons of sugar.

This time period would depend on the crop grown and the growing location.

3. EXTRACTION AND COMPUTATION

The NLToolset has a rule package that can recognize common temporal expressions, both absolute and relative; its accuracy has been measured at above 90%. An important feature of the NLToolset is the ability it affords the developer to add variables to the rule patterns. In the case of temporal expressions, the pattern variables capture the features, such as month, day, or year, that make up the expressions. These values are used in the computation of the interval representation.

Computing the Interval

The computation stage involves determining the reference point and using it, plus the feature information and the information from the expression's context to compute the interval. For example, if the expression is *next year*, the system would find the reference year and then add one; the interval would extend from January 1 until December 31 of that year.

If the expression is *Saturday*, the system must decide whether it refers to next Saturday or last Saturday, based on the sentence tense. It must then ascertain the weekday name of the reference date and add or subtract the appropriate number of days to reach the proper calendar date.

Arithmetic of calendar days across months can be problematic. To avoid this problem, the NLToolset converts each calendar day into a Julian day number form.[3] This number is the count of days,

[3] The Julian day number was introduced in 1581 by the French scholar Joseph Justus Scaliger to define a number from which all time could be reckoned. As a starting point, Scaliger chose the last year that the following cycles began simultaneously: the 28 year-long Sun cycle in which the calendar dates repeat on the same weekdays, the 19 year-long Metonic cycle in which the phases of the Moon repeat on almost the same calendar dates, and the 15 year-long cycle for tax collection and census that was used in the Roman

starting with the day 0 on the 1st of January, 4713 BC. After the calculation is completed, the NLToolset converts the Julian day back to its original time scale.

For the majority of cases, it is a simple matter to write a computation for a specific pattern that takes into consideration all of the relevant features and then determines the interval; however, the many-to-many relationship between features and expressions, coupled with a context dependency, complicates the overall process.

Algorithm Complexity

The simplest approach would be to write a package of rules, each of whose left hand side matches a certain time expression and whose right hand side is the relevant computation function. This method, while simple to implement, would bog down our pattern matcher by giving it too many possible paths to check. The following example illustrates this point.

Straightforward mapping of patterns to functions

```
< monthname > >> Function-1
< monthname day > >> Function-2
< monthname day year > >> Function -3
< monthname year > >> Function-4
```

In this example, if the pattern matcher finds a monthname, it must check each of these patterns to see which one is applicable. If, instead, we construct one non-deterministic pattern, we can eliminate this problem. The curly brackets indicate optional elements.

Collapse of four patterns into one

```
< monthname { day } { year } > >> Call-correct-function
```

In this case, the complexity migrates to the right side of the rule. The Call-correct-function function now must compute the interval based on the features that have matched. The difficult part, with a variety of candidate features, is constructing a decision tree that is efficient, and then, when new cases are added, reconstructing the decision tree, while maintaining its efficiency.

Identifying the Interval Type

Because the NLToolset represents dates as intervals, the NLToolset must decide how to fill the start and end points of each interval. A starting or ending point could be unknown, a part of the date that is being interpreted, or the dateline (or other reference date). The decision as to what will fill each point of the interval is based partly on the prepositions and context, and partly on the date being interpreted. For instance, *next week* will have a start date at the beginning of the week following the dateline, and an end date at the end of that week. However, *by next week* will use the dateline as the start date.

There are, by our reckoning, twelve ways to fill in the start and end dates. By examining the context in which the date appears, we can select one of these ways rather than trying to work with the contextual information directly as we fill in the interval. Table 2 enumerates the possibilities.

START	END	EXAMPLE
beg		before last week
unk	end	through last week
unk	dl	until today
beg	unk	as of this week
beg	end	(during) next week
beg	dl	beginning last week
end	unk	after next week
end	dl	since last week
dl	unk	after today
dl	beg	until next week
dl	end	through next week
dl	dl	today

KEY:

unk = unknown

beg = beginning of the interpreted date

end = end of the interpreted date

dl = dateline (or other reference date)

Table 2: Interval Type Algorithm

4. LEARNING THE DECISION TREE

We decided to try using machine learning to help generate the Call-correct-function code. We chose Quinlan's C4.5 software because it has been successfully applied to many problems requiring decision trees. C4.5 uses training examples to build a classification system, which, in this case, will comprise a decision tree which lays outs a feature-based path to each correct computation. As new cases

empire. This starting year for the Julian day was 4713 BC.

are added to the rule package, the tree can be quickly regenerated by adding more training examples.

Using C4.5

We will describe our experiment with C4.5. For a complete description of C4.5, see Quinlan's own publication.[4]

To use C4.5, the developer specifies: 1) the classes of interest; these will become the leaves of the decision tree and 2) the features and their possible values; these are the nodes of the tree. A set of training examples is provided and, when the tree has been generated, each path can be considered a rule.

The C4.5 specification builds a description space whose dimensions correspond to the number of features describing the problem. Each training example is a point within the space. The decision tree is a classifier that divides the description space into regions, each one labelled with classification type. C4.5 decides which feature is the best one to use as a first discriminator, and then starts to divide the region based on that feature. This is a key element of C4.5. It provides the most efficient tree that it can discover. It also includes heuristics for simplifying the tree. In general, C4.5 generates a decision tree by ordering the testing of features according to how much information each feature will provide. Each decision splits the region into smaller pieces, until finally the classification is reached.

According to Quinlan's guidelines, the best classifier will have few classes, few regions per class, many training cases relative to the volume of the regions, and no misclassification of the training cases.

Failed Attempt

Our first attempt at describing the problem in C4.5 syntax resulted in something like the following model.

Classes: one class for each
computation function

Features and Values:

Month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec)

Day (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31)

Year (continuous values)

This approach failed because it does not abide by Quinlan's guidelines. We are trying to classify into many categories, one for each computation function. Our preliminary working set consists of fifteen classes. We also have many features with many possible values, and not all of the features are relevant in every case. In fact, in all cases, only a subset of the features is relevant. As a result, C4.5 has difficulty in generating a good decision tree, even with several hundred training examples.

Different Approach

To remedy this situation, we transformed the description space by converting the feature values to boolean -- Y or N -- because the value of the feature does not matter as much to the decision as whether the feature is present.

Classes: one class for each
computation function

Features and Values:

Month (Y, N)

Day (Y, N)

Year (Y, N)

This change, although it maintains the large number of classes, allows us to reduce the volume of the regions and avoid the fragmentation of the previous model. Additionally, this model produces a binary tree, which is a simple if-then-else algorithm to implement. In fact, we can automatically convert the generated decision tree to C++ code, using a Perl script.

This is an unusual use of C4.5 in that it does not follow Quinlan's guidelines for developing a good classifier; however, it does work for our purposes. It has alleviated the tedious and time-consuming problem of generating and re-generating an efficient decision tree in C++ code.

5. NORMALIZATION

The NLToolset gives a temporal expression an interval representation. The temporal interval currently abides by the time standard of the original temporal expression; however, in future, the temporal interval will be normalized into Coordinated Universal Time (UTC), which is considered the modern implementation of Greenwich Mean Time.

[4] Quinlan, J. Ross. C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, 1993.

This time standard is used worldwide and will allow for greater interactivity between databases and within visualization tools.

The interval representation is stored within an NLToolset structure in its component parts; that is, the year, month, day, hour, minute, and second for the beginning and endpoint of each interval are stored separately. The original values of the text are also stored. This affords flexibility as the NLToolset is applied to various domains. The application requirements can dictate which parts of the time representation will be stored and displayed.

6. CONCLUSIONS AND FUTURE WORK

This paper has examined the task of extracting and normalizing temporal expressions, and has described the NLToolset's approach to accomplishing this task. It has also described the use of a learning program to deal with the complexity of developing such a system, as well as the methodology for normalizing temporal information for database use.

As the time extraction process is exercised across applications, it will be expanded to cover more and more cases.

Future research work may address the issue of ambiguous temporal expressions. Statistical means may be appropriate for representing the uncertainty of an interval representation. Comparisons across languages may also prove enlightening. In the near future, an existing prototype application will be translated into the Spanish language.