# Automatic semantic tagging of numerical expressions for supporting speech synthesis

*or How to tell a third from a day in March*

*Eira Monstad*

Master thesis in computational linguistics

February 2006


Institutt for Lingvistikk og Litteraturvitenskap

Universitetet i Bergen

# Table of Contents

All scripts, data material and MBL output files are available at http://epistel.no/master/

# Foreword

I have been interested in web development since the birth of the commercially available Internet. I see the Web as a democratic resource, promoting global exchange of information. As such, accessibility is very important, both in terms of internationalization – availability for users of all languages – and in terms of being usable for people with various disabilities.

In my work, I hope to improve the situation for both groups. Speech Synthesis Markup Language, SSML, is a markup language designed to make the Web more accessible for vision impaired users and other users who may benefit from speech synthesis. I will try to enhance the benefit for users of all languages, not just English speakers.

Thanks to:

My supervisor, Koenraad de Smedt, for professional, practical, and moral support

Gisle Andersen, for his help with access to corpus data and syntactical tagging

Tim Altman and Ian Hickson, for proof reading

My fellow students, for invaluable moral support

Helge Dyvik, for helping me get started

Kolbjørn Slethei, for suggesting SSML as an interesting topic to work on

Dag Bakke, for endless encouragement

## Abstract

The topic for this thesis is the semantic categorization of numerical expressions in running text in the context of speech synthesis. It is a problem for speech synthesis engines to determine whether 1/2 is a date, a fraction or something else. If this is not known, the number cannot be spoken correctly.

Using corpus resources, this thesis identifies a number of semantic categories that are important to distinguish between for the purpose of speech synthesis. The categories are defined within the framework of Speech Synthesis Markup Language (SSML), a markup language intended to improve the quality of input to speech synthesis.

The categories are used as classifiers in a syntactically tagged training corpus for memory-based learning. The learning process achieves up to 95% accuracy in determining the correct category for a numerical expression. The memory based learning approach appears promising as a method for automatically improving the quality of speech synthesis, though work remains in subcategorization of the numerical expressions.

# 1 Introduction

## *1.1 Motivation*

Speech synthesis is an invaluable tool for certain population groups, particularly those who are blind or weak-sighted. It is also increasingly popular in applications for a broader audience, such as automated messages that are given over a phone or speaker and GPS driving instructions – in short, any situation where there is no screen to read from or where the recipient needs to keep his eyes somewhere else.

Disambiguation is an important part of speech synthesis. In order to read the text correctly, the synthesis engine must distinguish between the senses of homographs that are not homophones. It would (in most cases) be silly if the speaker explained how to re-string the *bắs*[1] or prepare a delicious meal from a *bā́s*[2]. While English does not have many common words in this class, they occur more frequently in other languages. One disambiguation problem that is present in all languages, though, is the issue of how to handle numbers.

With ten digits we express a vast number of different meanings. Take the sequence 1/2. Is it a fraction? A date? An American date, or a European date? The first object in a set of two? A choice of one or two objects? The list goes on.

Words often do not have to be disambiguated at all. Letters and combinations of letters are mapped to sounds. Even if you have never heard of a "grulzag" before, you can probably pronounce the word just from reading it, and the machine does not have to know whether "rose" is a flower or something Jesus did after he died.

Numbers are not mapped to sounds in such a straightforward way. First, there is the fact that how a number is read depends on where in the sequence it is. 5 is five, but 50 is fifty, and to make it even more complicated, 15 turns it all around and becomes fifteen[3]. This is for all practical purposes a solved problem. But consider the sequence "January 5" or even "George 5". "Five" is suddenly not good enough, we want "fifth". And if we widen it even more, to "January 5 1995", "one thousand nine hundred ninetyfive" doesn't cut it. To make any sense it must be read as "nineteen ninetyfive".

So how do we teach the machine how to tell a third from a day in March? That is the topic for this project.

## *1.2 What I will do*

The markup language SSML, Speech Synthesis Markup Language, provides a set of semantic categories for numerical expressions. In this project I will use corpus resources to find empirical support for each of these categories, as well as empirical

1 Bass, a fish
2 Bass, a musical instrument
3 For more about this, see Gyri Losnegaard – Syntaksen i talluttrykk (Master thesis, 2005)

evidence for whether their definitions need to be changed to support international formats, and whether additional categories are necessary to cover important formats in the target domain.

I will then proceed to investigate how a text can be automatically tagged using these values. Some of the factors that may affect the accuracy are the preprocessing of the input text (syntactic tagging, breaking up the string in smaller components) and the existence of specific words – trigger words – in the context. I will look at how these factors affect automatic tagging, and how an optimal combination of ease, efficiency and accuracy can be obtained.

# 2 Background

## *2.1 Numerals*

### 2.1.1 Definitions

The term "number" has a large number(!) of senses. The simplest one, perhaps, is the mathematical sense: "one of a series of symbols of unique meaning in a fixed order that can be derived by counting". The term is also used to denote the words or symbols referring to numbers – also called numerals. And there is the adjective numeric or numerical, "Of or relating to a number or series of numbers".[4]

In most of this paper, I will use the term "number" in the loose sense, denoting both the abstract mathematical idea of numbers, and the words and symbols used to name them. Where the distinction is important, I will use the more specific term "numeral" to refer to the words and symbols used to name the numbers.

By "numerical expression", I refer to any expression composed of a series of numbers and separator characters – including, but not limited to integers, dates, fractions etc.

### 2.1.2 What are numerals?

Heike Wiese (1997) says:

> "The expression of numbers through numerals (and other linguistic means) is a universal characteristic of natural language, a circumstance that indicates the great importance of number concepts for human thinking. The notion of the number, which is the root of the perception of discrete objects, creates the notion of space – the continuous unity – a basic means of grasping reality."

Clearly, numbers are integral to how we conceptualize the world. Like nouns are used to carve up reality in types of objects, numerals are used to carve up reality in discrete items. In this sense, they have much in common with adjectives – they let us distinguish separate objects. Like nouns, numerals can not only count physical items, but also abstract notions, such as points in time. Hurford (1987) argues that *"numerals are primarily adjectives, and secondarily nouns"*.

Normally we think of numerals in a cardinal sense – we think of them as numbers, so to speak (Wiese, 2003). Cardinal numerals are ordered – four apples are more than three apples, 22.12.2005 is two days earlier than 24.12.2005.

Numerals can also be used nominally. Typical nominal uses of numerals are

4 The American Heritage Dictionary of the English Language

phone numbers or bus lines. In these contexts, the numeric value of the numeral has no meaning. Bus 22 is not inherently any larger or later than bus 21, and they might just as well have been named after their destination.

As a side note, however, the distinction is not as sharp as it may seem. Think of a bank account number. On first sight, it is clearly nominal. If my account number is 1234.55.67899 and yours is 1234.55.67788, it doesn't make sense to order them and say that my account is larger than yours. It is simply a handy way to give each account a unique name, and it might seem that the banking community could just as well have chosen letters or other symbols. But the choice of numerals is not entirely arbitrary, and this is where the cross-over between numeral and cardinal occurs. Bank account numbers, KID numbers[5] and similar unique names used in banking have checksums. Checksums are arithmetic operations that are carried out on parts of the number. The result is attached, usually to the end of the number. When the number is used, the arithmetic operations are run again and matched, to lessen the risk of having a typo cause you to pay your rent to a complete stranger.

Normally, it would not make sense to do arithmetic with a nominal expression. But it does not make sense to think of a numeral as cardinal if its numeric value cannot be ordered related to other numerals in the same set. The distinction is not very important for the task at hand, so I will not investigate it in more detail here.

Grammatically, numerals in the cardinal sense are usually treated as quantifiers. I only look at a subset of the quantifiers – namely, those expressed by numbers rather than by adjectives such as "many" and "some", for reasons outlined in the next section.

Numerals used nominally have much in common with proper names – in many ways, they *are* proper names. In the words of McDonald (1996), understanding proper names *"... is central to the analysis of the extended, unrestricted texts that have become the focus of research in the natural language processing community"*. And indeed, much work has been done on identifying and categorizing proper names over the last ten years. It is not unlikely that much of the work done here is also relevant for recognizing and understanding numbers.

It should not be doubted that numerals deserve the same amount of attention as the proper names have gotten, for exactly the same reasons. Understanding the number is important for understanding the text as a whole. Because numbers, written with different symbols than all other text, are so simple to recognize, it is easy to imagine that the task of understanding them is equally simple. This, we shall see, is not the case.

### 2.1.3 The problem

Magnus Olsson writes about dates in Swedish[6], explaining that years are grouped: "*Swedes realize e.g. the year 1845 as artonhundrafyrtiofem "eighteen-hundredfortyfive" and not as (ett)tusenåttahundrefyrtiofem "(one)thousandeighthundredfortyfive".*" Norwegian and English treat years similarly, as "atten førtifem" or "eighteen fortyfive". For a speech synthesis engine, it is vital to know whether 1845 is a date or not in order to choose between the the two pronunciations "eighteen fortyfive" and "one thousand eight hundred fortyfive".

5 A number that identifies who a bill is paid by and which bill was paid, so that the payment can be registered automatically in the receiver's computer system

6 Magnus Olsson, Swedish numerals in an international perspective (1997), p134

Hurford (1987) says:

> "Numeral systems are in clear ways well integrated with the languages in which they are embedded. In the stream of speech, numerals receive no special attention, making use of the same phonological units (say phonemes) and processes (phonological rules) as the rest of the language."

He goes on to mention that *"The alternative notation can be seen as an efficient shorthand for the longer forms, although it is no doubt significant that such shorthands are especially common for numeral expressions."*

Hurford recognizes that the existence of alternative notation is significant, but to him it is not a very interesting point. To me, however, it is at the core of the problem. In the context of speech synthesis, numerals expressed in letters pose no different difficulties than any other word.

I will look at numerals expressed symbolically, as a series of digits and separator characters – or, in Hurford's words, alternative notation. The alternative notation is not uniquely mapped to a semantic category nor to a phonetical realization. Thus, in order to present the text correctly to the listener, this mapping – disambiguation – must be done by the speech synthesis engine or by preprocessing its input. This is what I will attempt to do – specifically, I will try to do the first step of the mapping, from notation to semantic category. The step from semantics to phonology is highly language- and implementation-dependent and is best left to the particular speech synthesis engine.

## 2.2 Markup languages

"Marking up" a text means to include information about the text interleaved with the text itself. This can be any kind of information, for example about what the text should look like (`<font size="6" color="green">`), or about structure (`<title>Current Issues in Linguistic Theory</title>`). A markup language is the set of elements and rules for writing and reading a document marked up with that language. The early history of markup languages is unclear, but it is known to have been suggested at least as far back as the late 1960s.[7]

The purpose of using markup languages is to ease sharing of documents across multiple programs and platforms. With Tim Berners-Lee's[8] invention of the World Wide Web (WWW or "the Web") and HTML[9] (HyperText Markup Language), markup got a boost. The idea of the WWW was to give anyone, anywhere, with any kind of computer and connection to the Internet, access to the same information. That meant that infomation needed to be coded in a way that would be readable on any computer. (Tim Berners-Lee, 1998[10])

7 Markup language: http://en.wikipedia.org/wiki/Markup_language
8 Time 100: Tim Berners-Lee: http://www.time.com/time/time100/scientist/profile/bernerslee.html
9 HyperText Markup Language (HTML) Home Page http://www.w3.org/MarkUp/
10 The World Wide Web: A very short personal history: http://www.w3.org/People/Berners-

Today, the most widely known markup language is still HTML, though the W3C's[11] XML[12] (eXtensible Markup Language) is gaining popularity. HTML is a simple language designed to present documents in a human-readable way. XML, as the name suggests, is much more than that. In fact, it is not really a language itself, but a meta-language, a framework for creating markup languages. It provides a set of rules about structure and syntax which all languages based on XML must follow.

Both HTML and XML are derived from the SGML[13] (Standard Generalized Markup Language, ISO8879[14]) framework, which itself was based on IBM's Generalized Markup Language[15]. Because SGML is extremely complex, there was a need for a simpler format for small-scale use. HTML achieved this for the purpose of web documents[16], but was not flexible enough to replace SGML as a general framework. XML took this role, and has now almost replaced SGML. It is being used not only for sharing content on the Web, but also for storing settings and data in a large number of applications. One example is Microsoft Office, which will use XML as the default document format from version 12[17].

### 2.2.1 Standards and standards organizations

The World Wide Web Consortium, commonly referred to as the W3C or just the W3, is a consortium of companies and other organizations who have joined together to develop standards related to the Web. Standards are developed by Working Groups, which any member organization can join, provided they are willing to spend the time and money required. Working Groups can also have Invited Experts.

W3 standards are referred to as Recommendations[18]. During the development process, the responsible working group has to respond to all comments to the proposed recommendation. A "note" is a published working document, which may be changed by the Working Group that owns it at any time. There is no obligation for the Working Group to respond to or in any way deal with comments to a note. As such, it should be viewed as a best practice guide rather than a standard.

IETF is another standards body that deals with Internet-related standards, not necessarily confined to the WWW. One of their standards has become very important in the development of other standards, namely RFC 2119[19] which defines the use of the terms "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL". In the sections in this document where I suggest normative additions or changes to the SSML 1.0 say-as attribute values note, I will adhere to the use of these terms as defined

Lee/ShortHistory
11 World Wide Web Consortium: http://www.w3.org/
12 Extensible Markup Language (XML): http://www.w3.org/XML/
13 Standard Generalized Markup Language: http://en.wikipedia.org/wiki/SGML
14 Reference: ISO8879: http://www.w3.org/TR/html4/references.html
15 Generalized Markup Language: http://en.wikipedia.org/wiki/Generalized_Markup_Language
16 On SGML and HTML: http://www.w3.org/TR/html4/intro/sgmltut.html
17 Microsoft Office Open XML Formats Overview: http://www.microsoft.com/office/preview/developers/fileoverview.mspx
18 Recommendation Track Process Maturity Levels: http://www.w3.org/2004/02/Process-20040205/tr.html#maturity-levels
19 Key words for use in RFCs to Indicate Requirement Levels: http://www.ietf.org/rfc/rfc2119.txt

in RFC 2119.

### 2.2.2 XML

XML is a strictly hierarchical markup language. Documents are built using *elements*, *attributes* and *text nodes*. A document consists of an optional prolog[20] and one root element, which contains every other element and text.[21]

Elements are written as *tags* enclosed in angle brackets. An element named "title" would be written as `<title>`. Each opened element has to be closed with a forward slash, either with a separate closing tag: `<title></title>` or by including the slash at the end of the tag: `<title/>`.

An element can be empty, or it can contain other elements and/or text: `<author>`Written by `<name>`Eira`</name></author>`.

Elements can have *attributes* with *values*:

```
<address location="home">
  <name part="first">Eira</name>
  <name part="middle"/>
  <name part="last">Monstad</name>
</address>
```

In the above example, `address` is an element, `location` is an attribute, and `home` is an attribute value. The words "Eira" and "Monstad" are the *data*, while the rest is the *markup*.

The smallest possible XML document contains nothing but the root element, which may be empty:

```
<addressbook/>
```

XML doesn't define the names of the elements or attributes. XML is only a framework, on which markup languages can be built. Those languages define a set of allowed or required elements and attributes, and rules about which elements can contain which

20 Headers describing the document or linking to external resources
21 Erik T. Ray, Learning XML (2003)

other elements, which attributes apply to which elements, and the values the attributes can have.

### 2.2.3 SSML

SSML, Speech Synthesis Markup Language[22], is an XML-based markup language developed by the W3C. It is part of a larger set of voice-related markup languages, all based on XML.

SSML aims to improve the quality of synthesized content. Like the idea behind the Web itself, cross-platform and cross-implementation compatibility is one of the main aims. Traditionally, each speech synthesis application has had its own way of doing things, and authors have had little or no control over the resulting output. By providing "*authors of synthesizable content a standard way to control aspects of speech such as pronunciation, volume, pitch, rate, etc. across different synthesis-capable platforms*"[23], there is hope of higher quality and accuracy in speech synthesis. Authors can now help with disambiguation, provide pronunciation of words that don't follow common pronunciation rules, specify that a certain word or sentence is in a different language than the rest of the text, that a specific word should be pronounced in a special style, slower or with stress, and so on.

SSML is intended to be language independent. However, current versions of the specification does not satisfactorily cater for languages that fall outside the Western European language group. Work is being done to overcome this problem in future versions. In November 2005, a workshop was held on internationalizing SSML[24].

### 2.2.4 `say-as`

One of the controls offered by SSML is the element `say-as`[25], which tells the speech synthesis processor something about the semantics of the text it contains. A use case for this is to be able to distinguish between a date (10.08.05, August 10 2005) and a time of day (10.08.05, eight minutes and five seconds past ten)[26]. You can also specify the format in more detail, such as distinguishing between 10.08.05 meaning August 10 2005 or October 8 2005. The date August 10 2005 can be marked up like below, where the semantic category is "date" and the date is composed of three fields: day, month and year in that order, "dmy":

```
<say-as interpret-as="date" format="dmy">10.08.05</say-as>
```

22 Speech Synthesis Markup Language (SSML) Version 1.0: http://www.w3.org/TR/speech-synthesis/
23 SSML 1.0 (Abstract)
24 W3C Workshop on Internationalizing the Speech Synthesis Markup Language: http://www.w3.org/2005/08/SSML/ssml-workshop-agenda.html
25 SSML 1.0 S3.1.8
26 In Norwegian, the customary separator character in times is a dot, not a colon as in English

A problem in SSML is that the possible values for the attributes in the `say-as` element are undefined. Everyone who wishes to use this feature needs to define their own values. This hinders consistent implementations across applications and platforms, and thus opposes the very purpose of the specification. As a result of this, there is little support for the `say-as` element so far, even though most larger speech synthesis processors currently support or are in the process of implementing support for SSML. Defining which values are useful and necessary is a challenge. The SSML specification (W3C Recommendation 7 September 2004) says:

> Defining a comprehensive set of text format types is difficult because of the variety of languages that have to be considered and because of the innate flexibility of written languages. SSML only specifies the say-as element, its attributes, and their purpose. It does not enumerate the possible values for the attributes. The Working Group expects to produce a separate document that will define standard values and associated normative behavior for these values. Examples given here are only for illustrating the purpose of the element and the attributes.

The first version of this separate document was published on May 26 2005[27] as a W3C Working Group Note. In the rest of this paper, I will refer to this document as the Note.

In light of the the W3C's definition of a Working Group Note, it can be disputed whether the expectation of *"a separate document that will define standard values and associated normative behavior"* has been fulfilled by the SSML 1.0 Say-as Attribute Values Note. The Note itself states that *"Although the content may of course change before being introduced into a Recommendation-track document, the Working Group believes that the publication of this Note at this time may assist vendors in moving towards a common implementation rather than away",* but also that *"Comments received may be taken into consideration if the material in this Note is used in some form in the creation of a Recommendation-track document."* It is currently unclear whether `say-as` will be formally standardized.

The `say-as` element has three attributes: `interpret-as`, `format` and `detail`:

```
<say-as interpret-as="cardinal" format="," 
detail=".">1.000,50</say-as>
```

The value of the required attribute `interpret-as` specifies the semantic type of the `say-as` element's content – whether it is a date, a cardinal number, a time of day etc. The optional value of `format` suggests a more detailed interpretation for types with ambiguous format, such as the ordering of the fields in a date (for example, American month-day-year vs European day-month-year).

As defined in the SSML specification the `detail` attribute, also optional, specifies the level of detail the contents should be spoken with, for instance whether punctuation should be read out loud or just signaled with changes in prosody. However, in the Note, `detail` is instead used to specify grouping of the contents, such as

27 SSML 1.0 say-as attribute values: http://www.w3.org/TR/2005/NOTE-ssml-sayas-20050526/

specifying the thousands separator in a cardinal number – in other words, *where* instead of *how*. This inconsistency may be resolved in future versions of the Note and/or the SSML Recommendation.

It is important to note that the `say-as` element provides a *hint* about the semantics of the content, not an exact specification of what should be said. Thus, it may not always be spoken exactly the same way by all synthesis engines. The intent is to convey the same meaning, not necessarily the same rendering. The synthesis engine should speak the content according to what is appropriate for the language, locale and the user's preferences.

There is an alternative to `say-as` markup. One could, in an element or directly in the source document, replace the numerical expression with text. This solution, however, is much less flexible when it comes to rendering the text according to the user's preferences. It is also more difficult to automate.

Like the specification it is a part of, the `say-as` element is intended to be language independent. I will note some internationalization issues in the next chapter, but a comprehensive analysis of the Note's suitability for cross-language application falls outside the scope of this thesis.

As a side note, nothing in the SSML specification prevents the `say-as` element from being used to mark up non-numerical content. In fact, the specification explicitly suggests acronyms as a possible use case, though it immediately goes on to suggest other ways of dealing with acronyms. The Note does not give any examples of content without numbers. This project deals with numerical content only.

# 3 Defining the semantic categories

## *3.1 Formats defined in SSML*

Below is a summary of the six values for `interpret-as` defined in the SSML 1.0 say-as attribute values W3C Note of 26 May 2005.

### 3.1.1 Defined values

#### Date

The value `date` indicates that the contained string is a Gregorian calendar date.

Example:

`<say-as interpret-as="date">`10.12.2005`</say-as>` indicates that the contained text is a date. It does not specify what date it is – it may be December 10 or October 12.

The value of the `format` attribute when the value of the `interpret-as` attribute is "date" is built from the three characters `d` (day), `m` (month) and `y` (year), which indicates presence and ordering of the fields. The Note specifies that the speech processor must support at least three separator characters: the hyphen (-), the forward slash (/) and the dot (.). Which separator character is present in the contents is not specified in any of the attributes. The speech processor is expected to detect this automatically. The Note also requires that the same separator character is used to delimit all fields in a date string.

Example:

`<say-as interpret-as="date" format="dmy">`10.12.2005`</say-as>` indicates that the contained text is the date December 10 in the year 2005.

The `detail` attribute is undefined for `date`.

#### Time

The value `time` indicates that the contained string is a time of day. The value does not cover durations or time ranges.

Example:

`<say-as interpret-as="time">`10.15`</say-as>` indicates that the contained text is a time of day. It does not specify whether it is a quarter past ten in the morning or in the afternoon.

The `format` attribute has two possible values when the value of the `interpret-as` attribute is "time": `hms12` and `hms24`. They specify whether the time is in the 12-hour or the 24-hour format. The required separator characters are the colon (:), the dot (.) and the empty string ( ). As in `date`, the speech processor is expected to detect the separator character automatically. However, unlike `date`'s `dmy` format, `hms` does not specify the presence nor the ordering of the fields. The order is expected to always be hour, minute, second. The presence of minutes and seconds fields in the string is optional, but the string cannot contain a seconds field if it does not contain a minute field.

Example:

`<say-as interpret-as="time" format="hms24">`10.15`</say-as>` indicates that the contained text is the time 10.15 in the 24 hour time format, in other words, a quarter past ten in the morning.

The `detail` attribute is undefined for `time`.

## Telephone number

The value `telephone` indicates that the contained string is a telephone number.

Example:

`<say-as interpret-as="telephone">`55 58 00 00`</say-as>` indicates that the contained text is a telephone number. It does not specify what kind of telephone number it is, it may be in a different country or even be an internal number in a company.

The `format` attribute when the value of `interpret-as` is "telephone" is a string of digits corresponding to country codes defined by the International Telecommunication Union in the specification ITU-CC. These codes are commonly known as the country prefixes we dial when making international calls. A country code can be part of the contained string even though it is specified in the `format` attribute, and if they differ, the one contained in the string has priority.

Example:

<say-as interpret-as="telephone" format="47">55 58 00 00</say-as> indicates that the contained text is a telephone number in Norway.

The detail attribute is undefined for telephone.

## Character string

The value characters indicates that the contained string should be spoken as a series of alphanumeric characters, i.e. spelled out.

Example:

<say-as interpret-as="characters">747</say-as> indicates that the contained text should be pronounced "seven four seven" rather than "seven hundred forty seven".

The format attribute when the value of interpret-as is "characters" is one of the values glyphs or characters. The default value is characters.

If format is set to glyphs, the characters should be read with glyph information, that is, whether the character is uppercase or lowercase, accents and diacritics etc. This is important for reading passwords and other text that must be rendered exactly as it is written.

Example:

<say-as interpret-as="characters" format="glyphs">aBc47</say-as> indicates that the contained text should be pronounced as "lower a upper b lower c four seven".

If format is set to characters or is undefined (since characters is the default), the characters should be spelled out without detailed glyph information.

Example:

<say-as interpret-as="characters" format="characters">aBc47</say-as> indicates that the contained text should be pronounced as "a b c four seven".

The detail attribute is a series of digits specifying how the characters are to be grouped. This can be useful e.g. for account numbers, registration codes and other long numbers that are commonly grouped in a special way when spoken. Given a Norwegian

bank account number of 11 digits, the value of the `detail` attribute would typically be "4 2 5". It is up to the synthesis processor how the grouping is to be realized phonetically. If the specified value doesn't add up to the number of characters in the string, the `detail` attribute should be ignored entirely.

Example:

```
<say-as interpret-as="characters"
detail="425">82000176318</say-as>
```

indicates that the contained text should be pronounced "eight two zero zero <pause> zero one <pause> seven six three one eight".

## Cardinal number

The value `cardinal` indicates a regular cardinal number, integral or decimal, with an optional leading + or – to indicate positive or negative numbers.

Example:

```
<say-as interpret-as="cardinal">1995</say-as>
```

indicates that the contained text is the cardinal number one thousand nine hundred and ninety five.

The `format` attribute indicates the character used to separate the integral and fractional parts of the number. In Norwegian, this will usually be a comma, while in English, a dot is the common value. If `format` is not specified, it is up to the processor to decide.

Example:

```
<say-as interpret-as="cardinal" format=".">1.995</say-as>
```

indicates that the contained text is the cardinal number one point nine hundred and ninety five.

The `detail` attribute indicates the character used to group the integral parts of the number. In Western European languages this is usually the thousands separator.

Example:

```
<say-as interpret-as="cardinal" detail=".">1.995</say-as>
```

indicates that the contained text is the cardinal number one thousand nine hundred and ninety five.

### Ordinal number

The value `ordinal` indicates an ordinal integral number. It is up to the synthesis processor to decide how to handle ordinal numbers with separator characters.

Example:

`<say-as interpret-as="ordinal">21</say-as>` indicates that the contained text is the ordinal number twenty-first.

The `format` and `detail` attributes are both undefined for an `interpret-as` value of `ordinal`.

## 3.1.2 Overview

| ***interpret-as*** | ***format*** | ***detail*** | ***separator characters*** |
|---|---|---|---|
| date | d m y | N/D | - / . |
| time | "hms12" or "hms24" | N/D | : . " " (space) |
| telephone | ITU-CC country codes | N/D | N/D |
| characters | "glyphs" or "characters" | grouping digits | N/D |
| cardinal | fractional separator char | integral grouping char | N/D |
| ordinal | N/D | N/D | N/D |

*Table 1: Overview over attribute values specified in the Note. N/D = Not Defined.*

## *3.2 Defining new values*

The Note provides a mechanism for extending the `say-as` element. Backwards compatibility with the values defined in the Note is required, and there should be a clear distinction between the W3C-defined values and the ones defined by vendors. To achieve this, we use namespaces.

Namespaces, defined by W3C for use in XML languages, is a method for qualifying element and attribute names in order to make them unique.[28] XML is a framework for designing new languages, such as SSML. Anyone can design a new language to fit their needs. This leads to problems with naming collisions – two languages may contain elements with the same name, or, if the defined language allows vendor-specific extensions, two vendors may use the same name for their differently-defined extensions. Namespaces allow processors to easily identify the elements they

28 Namespaces in XML: http://www.w3.org/TR/REC-xml-names/

are designed to recognize and process, even when coming across an element with a name it recognizes but which is designed for use with some other processor.

I will not go into the details of namespaces here, except a quick note on how they are used. A namespace is identified by an Uniform Resource Identifier, a URI[29]. The most common way to use a namespace is to declare it in the document's root element, using the attribute *xmlns*, like this:

```
<speak version="1.0"
xmlns="http://www.w3.org/2001/10/synthesis"
xmlns:dasp="http://ling.uib.no/~eira/master/dasp">
```

The prefix is then used to qualify the appropriate names, like this:

```
<say-as interpret-as="dasp:fraction">
```

Using this method, the `say-as` element can be extended if needed.

## *3.3 Corpora*

In order to determine the need for new or changed values, corpus data was gathered and examined. The samples have been collected from the tagged Lancaster/Oslo-Bergen (LOB) corpus, Norsk aviskorpus and the Oslo corpus. The output has been manually sorted to leave only one example of each different format found in each language – for this task, this is all that is needed, since the goal is to find examples of the formats that exist, not to perform a statistical analysis of their frequency or distribution.

A corpus is, broadly speaking, a collection of text[30]. Used in modern linguistics, the term is usually restricted. McEnery/Wilson[31] list four characteristics:

- Sampling and representativeness
- Finite size
- Machine-readable form
- A standard reference

It is important to note that these are *characteristics*, not *requirements*. A corpus' features will vary with its purpose. While corpora intended for quantitative studies usually will have a finite size and carefully balanced sampling, corpora intended to give

29 Uniform Resource Identifiers (URI): Generic Syntax: http://www.ietf.org/rfc/rfc2396.txt
30 Corpus linguistics: http://ling.uib.no/~desmedt/cursus/corpus/syllabus/intro.html
31 Definition of a Corpus: http://bowland-files.lancs.ac.uk/monkey/ihe/linguistics/corpus2/2defin.htm (supplement to the book Corpus Linguistics, see http://bowland-files.lancs.ac.uk/monkey/ihe/linguistics/contents.htm)

a broad scope of the language or to monitor changes will often have a dynamic size, new texts being added on a regular basis.

Corpora are useful sources for collecting empirical knowledge about real-world language use. Because the intention is to as many different examples of numerical expressions as possible, the corpora used here have been selected to cover as wide an area as possible, without a strict criteria about representativeness. The LOB corpus is a well-balanced corpus for the British English language, modelled after the American Brown corpus[32]. The Oslo corpus[33] is the resource that comes closest to a standard corpus for Norwegian. While it has not been designed to be representative, it is quite large, and contains both factual and literary matter. Newspapers are a particularly good source of numerical expressions – dates, money and percentages are often important in news. This made it natural to add Norsk aviskorpus to the selection.

The next sections present the results of the corpus investigations. In 4.4, an overview is given of categories in the Note that were found in the corpus. In 4.5, the categories present in the corpora but not in the Note are listed and discussed.

## *3.4 Values defined in the Note*

Below is an overview of examples found in the corpora. In this section, samples covered by the categories defined in the Note are presented and discussed. In the next section, samples that are not covered by the Note are presented.

SSML aims to be language-independent. For that reason, it is important to investigate how `say-as` works for several languages. I have chosen to look at English and Norwegian. In some cases, examples of a given construct were only found in one language. There may be two reasons for this: either, that construct is not used in that language, or it is used but simply not present in the used corpora. The cases are discussed in more detail as they occur.

It is important to note that the examples found in the corpora are not necessarily comprehensive. If an example of a particular kind of numeral is not found, it may be a coincidence, and does not mean that using a number that way is not possible. And, of course, numerical expressions are a (very) productive class, so that a particular expression has not been used yet does in no way entail that it cannot be used. To distinguish the possibilities, one may broaden the search to other corpora, or utilize introspection[34].

The categories are semantically motivated, but with a bias towards speech synthesis. The purpose is to distinguish between numerical expressions that are pronounced differently even though they may look identical. These differing pronunciations occur because the expressions have different meanings. Thus, the categories have overlapping formats – a numerical expression covered by one category may look just like one covered by another.

This does not mean that the defined categories do not make demands on what the contents should look like – in fact, most of them do. This is a consequence of the requirement that the system should be implementable for speech synthesis purposes. In

32 The LOB Corpus: http://khnt.hit.uib.no/icame/manuals/lobman/LOB1.HTM
33 Oslo-korpuset av taggede norske tekster (bokmålsdelen): http://www.tekstlab.uio.no/norsk/bokmaal/
34 N. Chomsky, Syntactic Structures (2002), 49-50

order to speak the number correctly, the synthesis processor needs to be able to recognize the different parts of a numerical expression. In the case of a date, it needs to know which part is the day, which is the month and which is the year.

### 3.4.1 Ordinal

| |
|---|
| Det kan høres vel optimistisk ut med medalje etter den skuffende **<say-as interpret-as="ordinal">**9**</say-as>** . plassen i fjor |
| five castles , either built by Edward **<say-as interpret-as="ordinal">**1**</say-as>** , occupied by the English for a limited period |
| five different interpretations : *- **<say-as interpret-as="ordinal">**1**</say-as>** . genitive singular when split from some masculin |
| changes in the value of money . **<say-as interpret-as="ordinal">**1,000**</say-as>**th refugee . Britain received last week her 1,000th |

*Table 2: Ordinal numbers*

These examples present some challenges. First, how should trailing punctuation or suffixes that are commonly used to indicate that the numbers are ordinal, be handled? Punctuation usually consists of a dot (.), but obviously, a number followed by a dot in running text is not always an ordinal number. In English, it is common to use suffixes st, nd, rd and th.

There are four ways to handle these instances:

1. Add SSML markup for all the ordinal numbers, as in the above examples
2. Add SSML markup for ordinal numbers with trailing punctuation, but not with suffixes
3. Add SSML markup for ordinal numbers with suffixes, but not with punctuation
4. Do not add SSML markup for ordinal numbers with suffixes or trailing punctuation

Handling the two cases differently does not make a lot of sense, since they are in essence the same. There seems to be no real advantage to not adding markup. The processor should be expected to handle the instances with suffixes correctly, but as punctuation is ambiguous, adding markup can be of important help to the synthesis processor.

`Say-as` with an `interpret-as` value of `ordinal` cannot contain punctuation or letters. I believe the following procedure would be the most reasonable way for a synthesis processor to handle the case of ordinals with trailing suffixes or punctuation:

A processor that supports the `say-as` element should, in the case of suffixes, speak the contained number as an ordinal, and ignore a known suffix immediately following (i.e. no whitespace) the markup. In the case of punctuation, processor-specific heuristics should be applied to determine whether the dot marks the end of a sentence or should be seen as part of the ordinal number and thus be ignored. This kind of heuristics would also be needed if the number was not marked up, so the SSML markup imposes no extra requirements on the processor.

Since the specification only allows integral ordinal numbers, the processor may automatically treat relevant punctuation inside the number as grouping characters or simply ignore it[35].

### 3.4.2 Cardinal

| |
|---|
| I forhold til 2003 økte salget i fjor med ca . 25 prosent , og vil for hele året ende på nærmere **<say-as interpret-as="cardinal" detail=" ">**115 000**</say-as>** . [36] |
| **<say-as interpret-as="cardinal">**29**</say-as>**-åringen kom søndag kveld til familiens hus på Gause |
| Den kostet minst **<say-as interpret-as="cardinal" detail=".">**5.100**</say-as>** personer livet |
| Her var bølgen opptil **<say-as interpret-as="cardinal" format=",">**10,5**</say-as>** meter høy |
| Senterets første måling indikerer at skjelvet måler **<say-as interpret-as="cardinal" format=".">**8.0**</say-as>** på Richters skala . |
| Shirov har de siste ti årene vært inne på sjakkens topp-**<say-as interpret-as="cardinal">**10**</say-as>**-liste |
| in the potential range **<say-as interpret.as="cardinal" format=".">**-0.55**</say-as>** to -0.80 V . in both these papers one meets again the cu |
| Mr K's latest speech scared **<say-as interpret-as="cardinal" detail=",">**1,157**</say-as>** East Germans to cross into west Berlin's receptio |
| as background to other activity . a **<say-as interpret-as="cardinal">**14**</say-as>**-year-old boy confesses , *' you can neck and kiss your gir |
| is just under .5 compared with **<say-as interpret-as="cardinal" format=".">**.8**</say-as>** for the midland region and 1.2 for the country as |

35 as noted in the say-as note of 26 May 2005, section 3.6, first paragraph: http://www.w3.org/TR/2005/NOTE-ssml-sayas-20050526/#S3.6

36 SPACE is a valid separator character - http://www.w3.org/TR/REC-xml/#AVNormalize states that "All attributes for which no declaration has been read SHOULD be treated by a non-validating processor as if declared CDATA.", meaning that the whitespace in the value should not be normalized but be preserved as-is when passed from the XML parser to the processor.

Most of these examples conform to the defined standard in a straightforward way. The only content that might pose difficulties is the last, ".8". The specification does not handle this directly, but as it is common practice to omit the integral part of decimal numbers below 1, processors should be expected to handle this according to the custom in the spoken language. In English, it would typically be read as "point eight", whereas in Norwegian, it is common to insert the implied zero, reading it as "null komma åtte".

### 3.4.3 Time

| |
|---|
| Tirsdag 04.01.2005 , **<say-as interpret-as="time" format="hms24">**08:05**</say-as>** |
| Da den første navnelista ble publisert klokka **<say-as interpret-as="time" format="hms12">**11.30**</say-as>** , var antallet savnede 279 . |
| I dag **<say-as interpret-as="time" format="hms24">**0800**</say-as>** offentliggjorde etaten en ny liste . |
| centred ESE of Newfoundland at **<say-as interpret-as="time" format="hms24">**00**</say-as>** GMT 26 February 1959 ( fig 1 ( a ) ) moved ra |
| from 2200-0200 hr ; and 1st , from **<say-as interpret-as="time" format="hms24">**0200**</say-as>**-0600 hr . similar ontogenetic differences are appare |
| was reported missing by her parents at **<say-as interpret-as="time" format="hms12">**1 a.m**</say-as>** on Sunday July 9 . a search was made , but |
| in tonight's *' lifeline **' ( BBC , **<say-as interpret-as="time">**10.15**</say-as>** ) . they will be asked to comment on the design o |
| C04 federation of Rhodesia and Nyasaland ( **<say-as interpret-as="time" format="hms12">**10.30 p.m**</say-as>** ) . say Granada TV , the producers : *' w |

The Note does not mandate support for a.m and p.m without the final dot. The corpus results show several examples of a.m and p.m used without the final dot. Searches in more recent corpora[37] do not give any results without the final dot, which may indicate that people have moved away from this way of writing these abbreviations. However, the processor should be expected to read both old and new texts as close as possible to how a human would read them. Thus, there may be good reasons to allow common mispunctuation.

The Note does not require content to follow the suggested lexical token definitions, but it does expect of conforming processors that they should support the lexical tokens defined in the Note. In other words, content that differs from the defined lexical tokens is allowed but may not be correctly interpreted by the processor.

37 FROWN untagged, 90's, FLOB untagged, 90's, Australian untagged, 80's

When marking up the text, I will allow a.m and p.m without final punctuation for the above reason. I will also allow using the number 24 for end-of-day midnight[38]. How to handle this when speaking will be up to the processor.

The Note is ambiguous when it comes to which time separator characters processors are required to support. The introduction states that "At least one separator character must be supported: the colon (':')." However, under the heading "Basic tokens of a time string", three separator characters are listed; colon (":"), dot (".") and empty string (""). If say-as is to work internationally, all three are needed. Thus, I will assume processor support for all three.

### 3.4.4 Date

| |
|---|
| Ronny Deila kunne juble for mål for Odd mot Viking **<say-as interpret-as="date" format="d">**16**</say-as>** . mai . |
| Frode Olsen i aksjon for Viking mot Bryne i **<say-as interpret-as="date" format="y">**2003**</say-as>** |
| Selv om Kripos uomtvistelig er vår fremste ekspertise på dette området , slapp de først til torsdag **<say-as interpret-as="date" format="dm">**30.12**</say-as>** . , fire døgn etter katastrofen . |
| Publisert : **<say-as interpret-as="date" format="dmy">**03.01.2005**</say-as>** - 19:49 |
| AV/DN96/01 : -96 () Forfatter : Knut Asbjørnsen Dato : **<say-as interpret-as="date" format="dmy">**21-06-96**</say-as>** 13 : 09 |
| Årsrapporten for forsikringsselskapet Skogbrand for perioden 1/10 -93 til 30/9-94 forteller derfor om et rolig forsikringsår |
| other words the brunt of the attack in **<say-as interpret-as="date" format="y">**1381**</say-as>** fell on those who were , either professionally or |
| , via Steyning , opened on July **<say-as interpret-as="date" format="d">**1**</say-as>** , 1861 . he got to work immediately . he and the |
| Salisbury on the morning of the **<say-as interpret-as="date" format="d">**11**</say-as>**th , was the thought that *' Lord Lytton ( was ) goi |

Note that one of the lines above does not contain a markup suggestion. That is because the Note requires that *"The same character must be used to delimit all fields in a date string"*. In Norwegian, "d/m-y" is a common format. I will get back to this problem.

In a sentence such as "on the morning of the 11th", 11 is both an ordinal number and a date. For the purpose of reading the text out loud, which is the intended application of SSML, knowing that it is an ordinal number is more useful. At least in Western European languages, ordinal numbers are pronounced identically whether they

38 See discussion at http://lists.w3.org/Archives/Public/www-voice/2005AprJun/0032.html and http://lists.w3.org/Archives/Public/www-voice/2005AprJun/0046.html

are dates or any other semantic type. I am not aware of languages where this is not the case, though they may of course exist. In light of this information, I will choose to give ordinal priority over date when marking up the text.

The timestamp format "01012005" is not supported. However, this way of writing dates is uncommon in texts meant to be read by humans, and so falls outside the scope of `say-as`.

The English corpus material did not contain dates on compact format (e.g. dd.mm.yyyy). This is not surprising, since in English it is a lot more common to spell out the month name, either in full or abbreviated. The Norwegian examples are sufficient to demonstrate the use of SSML for such cases.

### 3.4.5 Character string

| |
|---|
| HJULENE BRAKK : Dette Boeing **<say-as interpret-as="characters" format="characters">**737**</say-as>**-lasteflyet ble stående på rullebanen på flyplassen i Banda Aceh |
| Vogntog-velt på **<say-as interpret-as="characters" detail="1 2">**E39**</say-as>** i Høyanger |
| Bank : Kontonummer 12 eller **<say-as interpret-as="characters" format="characters" detail="4 1 2 1 5">**8380.08.07800**</say-as>** hvis du benytter nettbank eller kontofon . |
| Ett døgn senere var det første flyet fra **<say-as interpret-as="characters">**335**</say-as>**-skvadronen med personell av gårde . |
| 2b . hence , o(t) must admit both **<say-as interpret-as="characters" format="characters">**2a**</say-as>** and 2b as periods . if A , B , and C are any thre |

How to realize the grouping phonetically is up to the processor. A central problem is whether to read each number separately, using the grouping value to insert pauses, or whether to use the grouping to read more complex numbers. This is not explicitly defined in the Note. Exemplified by the account number, two sensible ways to read the number in Norwegian would be:

```
<say-as interpret-as="characters" format="characters"
detail="4 1 2 1 5">8380.08.07800</say-as>
```

*Åtte tre åtte null <pause> punktum <pause> null åtte <pause> punktum <pause> null sju åtte null null*

Or:

```
<say-as interpret-as="characters" format="characters"
detail="2 2 1 2 1 2 3">8380.08.07800</say-as>
```

*Åttitre åtti punktum null åtte punktum null sju åtte hundre*

Note that the value of the `detail` attributes are different. If they were not, this decision could just be left up to the processor, as suggested by the Note. But we need to make a conscious decision when marking up the text, and the choice will depend on the expected rendering.

Reading simple numbers is a "safer" decision than reading complex numbers. It will work in all cases, and is more compatible with the idea of character string as a semantic value. Thus, I will assume that character strings will always be read one character at a time. That means when there is a strong preference for reading a number as complex, it should be marked up as cardinal if possible, even if it is part of a larger word. An example of this above is the road number E39, which should then be marked up as:

```
E<say-as interpret-as="cardinal">39</say-as>
```

### 3.4.6 Telephone

| |
|---|
| Telefon **<say-as interpret-as="telephone" format="47">**22 31 05 57**</say-as>** |
| Ring da **<say-as interpret-as="telephone">**02800**</say-as>** , og du blir automatisk satt over til din lokale politistasjon . |
| Ringer du fra utlandet , er nummeret **<say-as interpret-as="telephone" format="47">**+ 47 23 20 87 00**</say-as>** |

The English corpus material contained no telephone numbers. In order to include some common English formats, I have added the US Postal Service's phone numbers, as listed on their web site:

| |
|---|
| General Information: **<say-as interpret-as="telephone">**1-800-ASK-USPS**</say-as>** |
| or **<say-as interpret-as="telephone">**(800) 275-8777**</say-as>** |
| Domestic Package Tracking **<say-as interpret-as="telephone">**1-800-222-1811**</say-as>** |

The Note does not define a way to group digits in phone numbers. The processor is expected to do this automatically based on whitespace and separator characters in the contained text, and language-specific customs. The Note imposes no restrictions on the contained string, the content is only restricted by the synthesis processor's abilities.

## *3.5 Categories found in corpora, but not covered in the Note*

Some findings are not covered by the categories defined in the Note. Some are specific semantic categories that for most practical purposes can be marked up as cardinal numbers, including percentages and currencies. Bible passages are compounds of various types of numbers, and cannot be satisfactorily marked up as cardinal numbers. I will get back to this.

The goal must be to have the smallest number of categories that cover the field well and do not overlap. This is necessary to achieve ease and elegance of implementation and authoring. Pros and cons must be weighed to decide where the limits go. Categories that are only needed for number formats that are not regularly found in the target group of texts should be avoided.

I have defined two main conditions that should be met for a new category to be made:

- Distribution: The format must have a non-negligible distribution in the target texts, both in number and in being present in a variety of domains.
- Relevance: The format must be semantically well delimited

I will go through some uncovered categories in detail:

### 3.5.1 Date

| |
|---|
| Årsrapporten for forsikringsselskapet Skogbrand for perioden 1/10 -93 til 30/9-94 forteller derfor om et rolig forsikringsår |

This sentence shows two common variations on the date format d/m-y, one with a space before the year and one without. The first case can be trivially handled by the `date` format as defined in the Note, simply by marking up d/m and -y separately. This is also required simply because the Note does not allow whitespace in a date string. The representation would be:

```
<say-as interpret-as="date" format="dm">1/10</say-as>
<say-as interpret-as="date" format="y">-93</say-as>
```

The Note does not explicitly allow a leading date field separator for the format `y`. Supporting this would be up to the synthesis processor. Even if the processor does not explicitly handle this case, any synthesis processor with sane error handling should simply ignore the leading hyphen.

The second case is harder to handle. One could follow the above procedure and mark up the date as two separate elements. This does however seem like a hack – the restriction is imposed by limitations of the markup language rather than mandated by the semantics of the text itself. This is also the case for the first date in the example. Thus, I propose a new definition:

> Definition: A `dasp:date` consists of the three optional fields `m`, `d` and `y`, in any order, separated by separator characters. The presence and ordering of the fields is specified in the `format` attribute, as for `date`. The synthesis processor should support at least the following tokens as separator characters: - (hyphen), / (forward slash), . (dot) and whitespace. Whitespace preceding or following another separator character should be ignored.

We then end up with the following markup for the two dates in the example:

```
<say-as interpret-as="dasp:date" format="dmy">1/10 -93</say-as>
<say-as interpret-as="dasp:date" format="dmy">30/9-94</say-as>
```

### 3.5.2 Range

| |
|---|
| 30-40 utlendinger drev business på øya . |
| 21.-23 . januar |
| 700.000-800.000 personkunder |
| I sesongen 1992/1993 |
| mellom 26-28 timer |
| same as the C.F.O forecasts . the 1,000-500 mb thicknesses and the 500 mb heights are muc |
| . 7 . repeat ( 5 ) . then repeat ( 1-4 ) , plug will be wetter than when you started by |
| ( p 5 ) , points to the period c 1280-1300 for the date of the group of pottery from Lesnes |
| or what parents want . **' a number of 14/15-year-olds seem to like the serials on television but it is |

| table tops ; ( b ) . this lasted from 4 1/2 - 7 years of age roughly . at first the child |
|---|

As we can see, the ranges are composed of numbers of various categories: cardinal numbers, dates and fractions (not defined in the Note). It is also quite likely that one will find ordinal numbers ("1.-3. place") and times ("12.00-14.00").

What all of these examples of ranges have in common is that they consist of two (and always two) numbers of the same semantic category, usually a hyphen, "-". Forward slash, "/", seems to occur if the second number immediately follows the first in the domain. In cases other than this the "/"as separator character often suggests that the expression is not a range, but two separate examples for separate cases. One example could be a recipe that lists the needed ingredients for two versus four servings. We would then find "2/4 eggs" without it being the range "two to four", but rather "two or four" (depending on which mode of the recipe you're following). And last but not least, the choice of separator character may be language specific.

How to speak the ranges is obviously language specific, and varies a lot more than the values defined in the Note. The first five examples above will typically be read in Norwegian as "tretti til førti", "tjueførste til tjuetredje", "sjuhundretusen til åttehundretusen", "nittennittito nittennittitre" and "tjueseks og tjueåtte". This depends on context and must be handled by the synthesis processor. Handling these differences is beyond the scope of `say-as`.

SSML specifies that *"The say-as element can only contain text to be rendered."*[39] In light of this, there are three possible ways to handle the case of ranges. All three involve the creation of a new value for `interpret-as`, `range`, using the `detail` attribute to specify the separator character. This use of `detail` is in line with how it has been used in the Note.

The first possible approach is to advocate a change to the SSML specification to allow `say-as` to be recursive:

```
<say-as interpret-as="range" detail="-">
     <say-as interpret-as="ordinal">21</say-as>. -
     <say-as interpret-as="ordinal">23</say-as>.
</say-as> januar.
```

Changing a W3C Recommendation is a process that takes years, and the proposed change might not be accepted. In any case, changing the SSML specification is beyond the scope of this thesis.

The second approach is to use the `format` attribute to specify the semantics of a range's contents:

```
<say-as interpret-as="range" format="ordinal"
detail="-">21. - 23.</say-as> januar.
```

39 SSML 1.0 S3.1.8

As SSML 1.0 states that *"the optional format attribute can give further hints on the precise formatting of the contained text for content types that may have ambiguous formats"*, this solution may be defended. It is, however, not very elegant, and is harder to implement.

The third approach is to drop the inner level of semantics, giving "range" as the only semantic information:

```
<say-as interpret-as="range" detail="-">21. - 23.</say-as>
januar.
```

This seems a suboptimal solution, since more information is lost than gained as far as phonetic realization is considered.

A relevant question is whether it is useful or necessary to add markup for ranges. As seen above, the result is either added complexity or lost information. The cost may be higher than the benefit. The question is to what degree the synthesis processor is able to utilize the added information to separate ranges from similarly formatted numerical content, such as "30 minus 40".

It would be interesting to compare a synthesis processor's results with and without tagging of ranges before deciding whether the benefit outweighs the cost. Unfortunately there is a bootstrapping problem. Until the value has been properly defined, existing synthesis processors will not implement support for them.

In light of the discussion above, I will not use specific markup for ranges in this project.

### 3.5.3 Fraction

| |
|---|
| I den nye filmen " 37 1/2 " spiller teatersportekspert Helén Vikstvedt en Dagbladet-journalist i midtlivskrise . |
| Morton's very fierce 3-carburetter 4 1/2-litre Bentley and Morley's drastically lowered 4 1/2-li |
| put opposite one another so that r = 3/4 . the values of X3 and X4 are put toget |

Fractions are important in texts meant to be human-readable. In such texts, you will often find 1/2 instead of 0.5. Unfortunately, the Note does not handle fractions. Ideally, the speech synthesis should not hinder or disrupt the listener's comprehension of the text. In order to maintain flow and ensure correct understanding, it is important that the synthesis processor reads "a half" instead of "one slash two", or worse, mistakes it for a date.

Fractions are semantically well defined, and adding this as a new category will not overlap already defined categories. In conclusion, the benefits outweigh the cost,

and a new category should be defined.

There are two possible ways to define this new category. Consider the example "1 1/2", normally read as "one and a half". Should the integral number be contained in the markup, or left outside?

> Definition 1: A fraction consists of an optional integral number with following whitespace, followed by an integral number, a separator character, and an integral number:
>
> ```
> <say-as interpret-as="dasp:fraction">1 1/2</say-as>
> ```
>
> Definition 2: A fraction consists of an integral number, a separator character, and an integral number.
>
> ```
> 1 <say-as interpret-as="dasp:fraction">1/2</say-as>
> ```

The latter definition, where only the fraction itself is marked up, seems to be the cleanest and least overlapping solution. The synthesis processor should utilize this information to speak preceding numbers according to context – in Norwegian, that means adding the word "og", rendering the expression "en og en halv", completely marked up as:

```
<say-as interpret-as="cardinal">1</say-as> <say-as
interpret-as="dasp:fraction">1/2</say-as>
```

In other words, no information will be lost by choosing the simpler solution.

The `format` attribute could be used to specify the separator character, similar to how it is used for the value `cardinal` defined in the Note. The alternative is to leave the separator character undefined, leaving it up to the processor to transparently support likely separator characters. This is analogous to how the Note handles separator characters for `time`.

There seems to be no real benefit to specifying the separator character explicitly in the case of fractions. If the content is a valid fraction, there will only be one separator character, which should be easily identifiable for the processor. In conclusion, both `format` and `detail` are left undefined for `fraction`. The synthesis processor is expected to ignore any values in these attributes that it does not support, as defined in the SSML Recommendation[40].

40 SSML 1.0 S3.1.8

### 3.5.4 Ratio

| |
|---|
| survival ratio was high , at around 1 : 100 . my gravest error was in the choice of the |
| femoral nerve . blood pressure was 115/70 . the haemoglobin was 11.4 g per 100 ml . a d |
| for a horse at the short odds of 12-1 *- I believe he will last out only on the best of |
| low interstage pressure ratios ( 3/1 or less ) and by the use of lubricating oils of h |

Ratios and fractions have a lot in common, both semantically and in terms of syntax. What is the difference between a ratio and a fraction?

In simple terms, a ratio is a relationship between two or more sets (which may or may not be in a subset/superset relation), while a fraction is a relationship between a subset and its superset. If you have three apples and five bananas, the ratio is 3:5. A fraction is a part of a whole, which means the proportion of apples is 3/8. The example with blood pressure is perhaps even clearer: The first number is the maximum pressure (during ventricle contraction), and the other is the minimum pressure (between ventricle contractions). It is quite clear that this relationship is not a fraction. The survival ratio in the first example, on the other hand, can easily be interpreted as a fraction. This shows that fractions are a subset of ratios. In the following text I will for simplicity use "ratio" and "fraction" as mutually exclusive categories, where "ratio" will mean all ratios that are not fractions.[41]

The question is whether ratios and fractions can be handled by the same category, and whether a joint category would be a suitable solution. From the examined material it seems that "pure" fractions are significantly more common than the looser ratio, also across domains, though a proper statistical analysis was not performed.

Ratios and fractions can be pronounced in the following ways in Norwegian:

- Ratio: hundreogfemten *over* sytti
- Ratio: tolv *til* en
- Brøk: tre fjerdedeler
- Brøk: tre *av* fire

The corresponding English pronunciations are:

- Ratio: onehundredandfifteen *over* seventy
- Ratio: twelve *to* one
- Fraction: three fourths
- Fraction: three *out of* four

41 References:
Ask Dr. Math: http://mathforum.org/library/drmath/view/63884.html
Ratio. Fraction. What's the difference? http://www.sci.tamucc.edu/txcetp/cr/math/rf/RatioFraction.pdf
Ratio: http://en.wikipedia.org/wiki/Ratio

Choosing the appropriate pronunciation is up to the synthesis processor. By extending the `fraction` category to include generic ratios, the choice is made much harder by at least doubling the possibilities. If the processor chooses the less appropriate of the two ways to speak a fraction in a given context, it is not a big problem. The meaning is not changed. The consequences are worse if the wrong choice between ratio and fraction is made, because in this case the meaning of the expression is changed. A situation like this is important to avoid.

There are three possible ways out:

1) Have separate categories for ratios and fractions
2) Have a generic ratio category, but use the `format` attribute to specify ratio or fraction
3) Ignore non-fraction ratios altogether.

Because ratios are limited in frequency and cross-domain use, and because adding ratios is expensive to implement, I will go with the third option of ignoring non-fraction ratios. The benefits do not outweigh the costs for a generic-purpose system.

### 3.5.5 Index

| |
|---|
| 18 . we therefore had to consider : ( 1 ) whether any new advice could be brought effecti |
| . **' measures agreed so far include : 1 . a mass call-out of police , special constables |
| of shots within reasonable limits . 1 . introduction . about a hundred million shots a |
| Huxley , the flame trees of Thika . 1 : impartiality . Mr Corfield has a distinguishe |
| header tanks , as indicated in figures 7-10 . with the simple cross-flow layout in figure 7 , |
| also seen: 7:10 |

These numbers are in themselves covered well by the value `cardinal`. What makes them special is that they are often written the same way as ordinal numbers or sometimes even ranges. In some contexts and/or languages list indices may be more appropriately spoken as ordinals than cardinals, and should be marked up accordingly. This is an obvious problem for automatic tagging. There is, however, no significant advantage to creating a specific `interpret-as` value for lists and other indices.

In conclusion, no new category is created to handle these examples.

### 3.5.6 Mathematical expressions

| |
|---|
| we impose are ( 1.3 ) , for some b 0 , together with the requirement that y(x) be $L^2$ |
| ourselves to l 0 . the case l = 0 , with q(r) continuous , is just the Sturm-Liouvi |
| H so obtained was such that $p_h$ 0.05 . 4 . discussion . the following discussion deals |
| located on the x axis , between 0 and 1 , and g(ch) also on the x axis , between 1 |
| with s = 1/2 we have from *?13 10.6 equation ( 83 ) |
| resistance the current will be : *- 3 x $10^{-14}$ coulombs / sec . the charge on 0 |
| : x thousand pounds will produce x or x/2 or x/4 or 2x or 4x votes . |
| , H is distributed approximately as $xe^2$ with d.f = k-1 , |
| respectively . the proportion between $z_1$ and $z_2$ is therefore 0.4732 - 0.2734 = 0.1998 |

The numbers are intrinsically cardinal numbers. The challenge for the synthesis processor is to use the context to decide how to speak the operators. This is a domain-specific task that lies outside the scope of `say-as`.

### 3.5.7 Score

| |
|---|
| after the sinewy Scots had led 6-0 for nearly an hour . five sparkling minutes of fl |
| 1- 0 til Sør-Afrika |
| Gustad har likevel en bedre følelse nå enn i september , da det ble 22-25 i seriestarten . |

Sports results and similar scores are very common in newspaper articles and other media. Different sports have different conventions on how to read the scores, though the meaning will usually be kept even if the reading does not follow the expected convention. When it comes to format, scores have a lot in common with ranges.

Scores do not strictly meet the cross-domain criterion. However, given the proportion of sports articles in the media and importance of sports in modern society, a separate `score` category may still be warranted based on the size of the domain rather than number of domains. The category is well delimited and does not add significant complexity for implementors or authors.

Definition: The content of a `say-as` element with the value of `score` for `interpret-as` consists of a cardinal number, optional whitespace, followed by a separator character, optional whitespace, followed by another cardinal number. The hyphen ("-") must be supported as a separator character. Other separator characters

may be supported. `format` and `detail` are undefined for `score`.

```
<say-as interpret-as="dasp:score">6-0</say-as>
```

### 3.5.8 Guns

| |
|---|
| between two fellows . **' he hauled a .44 from a back pocket and laid it on his lap . |

Bullet sizes are often found in crime reports and similar contexts. Special conventions for reading them apply in most languages. However, these instances are relatively infrequent when all domains are viewed together, and are hardly ever found outside a very limited set of domains. In conclusion, a new category should not be made to handle these cases. A synthesis processor may be programmed to recognize the relatively small set of common bullet sizes in order to speak them correctly, or, if one does not want to rely on the processor, the `phoneme` element may be used.

### 3.5.9 Issue numbers

| |
|---|
| to acknowledge receipt of your note nd 115/12 of the 18th of November confirming that it is the |
| a paragraph from the ministry circular 5/61 which is the present practice and which is now be |

The challenge when reading issue numbers is to avoid confusing them with e.g. fractions. Realizing that something should be read as an issue number will in most cases require advanced interpreting of semantic context, though trigger words may be of some help. Generic synthesis processors cannot be expected to handle these cases reliably.

Issue numbers are not common enough to warrant a separate `say-as` category to assist processors in this process. The issue and volume could be marked up separately as cardinal numbers. In this case it will be up to the synthesis processor how to speak the separator. Presence of `cardinal` markup should prevent the synthesis processor from rendering the content as a fraction or other complex categories.

### 3.5.10 Biblical references

| |
|---|
| moral impasse . the quest for wisdom . 1 . 12-18 12 . I the preacher have been king over I |

| |
|---|
| . ) the search for the supreme good . 1 . 12-2 . 26 under the pseudonym of Solomon , Qohe |
| of all the seed , but John 1 : 14 reveals that he was made flesh so that of hi |
| . Lit *' asked **' . we may compare 1 John 2.16 for the *' lust **' ( desire ) of the e |
| that maketh not ashamed ( Rom 5 : 4,5 ) . the scripture speaks more than once of a *' b |

Biblical references are basically composed of cardinal numbers and ranges. As they are relatively infrequent, and in most cases domain-specific, a new category should not be created to handle them. The synthesis processor may implement heuristics to recognize these cases directly. As these expressions are quite complex, attempting to mark them up with existing `interpret-as` values may do more harm than good. My suggestion is therefore to leave them undefined, or alternatively specifying pronunciation with the `phoneme` element.

### 3.5.11 Summary

Below is a table describing the six interpret-as values defined in the Note, as well as the three new values suggested in the discussion above. No new values were defined for the categories range, ratio, index, mathematical expressions, guns, issue numbers and biblical references, for reasons outlined in the discussion.

| ***interpret-as*** | ***format*** | ***detail*** | ***separator characters*** |
|---|---|---|---|
| date | d m y | N/D | - / . |
| time | "hms12" or "hms24" | N/D | : . " " (space) |
| telephone | ITU-CC country codes | N/D | N/D |
| characters | "glyphs" or "characters" | grouping digits | N/D |
| cardinal | fractional separator char | integral grouping char | N/D |
| ordinal | N/D | N/D | N/D |
| dasp:fraction | N/D | N/D | / |
| dasp:score | N/D | N/D | - |
| dasp:date | d m y | N/D | - / . |

*Table 3: Overview over all attribute values specified in the Note as well as in this document. N/D = Not Defined.*

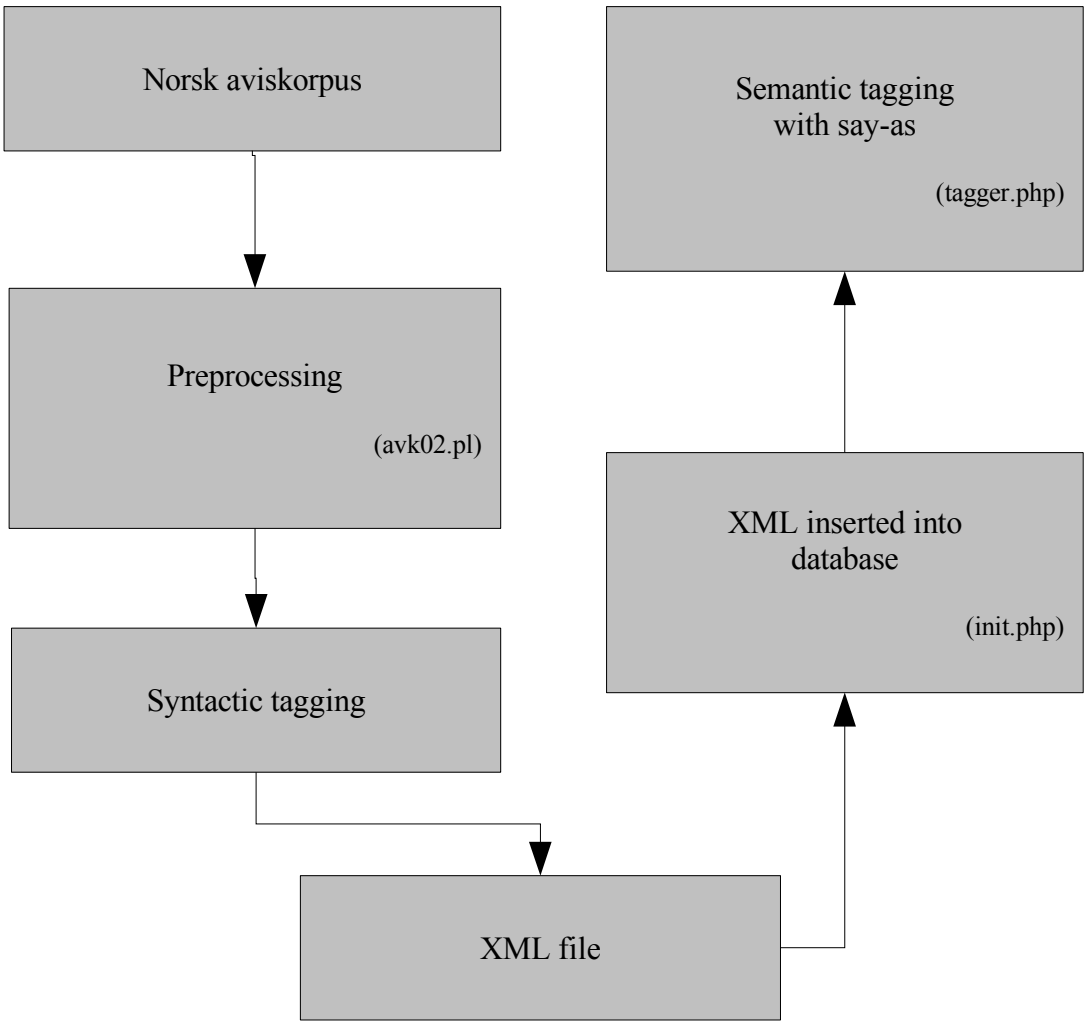
Obviously, not all numerical expressions will be covered by these categories. These expressions will simply be left untagged in a final text. For the purpose of further investigations in this paper, numerical expressions that fall outside the defined categories will be tagged with a dummy `interpret-as` value of "unknown".

# 4 Tagging

## *4.1 Introduction*

This chapter describes the process of collecting raw data and tagging it with syntactic and semantic information. The next section introduces the corpus the data was collected from, and explains why this particular corpus was chosen. Section 4.3 describes the preprocessing done before tagging could start. Section 4.4 describes the syntactic tagging, and section 4.5 describes the semantic tagging.

The process in this chapter is described in the flowchart below:



## 4.2 Corpus data

Norsk aviskorpus[42] - Norwegian newspaper corpus – is an Aksis[43] project collecting data daily from a large number of Norwegian newspapers online. The corpus is intended both for regular empirical studies of the Norwegian language and for the study of construction and development of new words in the language and other linguistic changes[44].

My choice of corpus was based on expected content as well as adaptation to expected target usage of SSML. Newspapers typically contain a lot of numbers – dates, times, statistics, finance, sports scores et cetera, which means it is a good source of information. Newspaper articles are also a typical use-case for SSML and the `say-as` element, and perhaps especially the automated tagging with such markup. Accuracy is important, giving a need for disambiguating, but the volume and rapid updates of the information means that it is not a viable task to do hand-tagging or make recordings of humans reading each text, as is often a better choice for making more static text available for listening.

## 4.3 Preprocessing

The first step was collecting the needed data and preprocessing it. I picked a random set of days from Norsk aviskorpus, spread out over several years, and concatenated the files. This data had already been preprocessed, with a word to each line. I had to reverse this process in order to build normal sentences appropriate for syntactic tagging. This was done with a Perl[45] script[46].

The main problem was that punctuation was on separate lines. Words on each line could simply be put next to each other with a space between, but if this was applied to punctuation, the result would be sentences such as "Hurry up , he said ." where the original text most likely was "Hurry up, he said."

The file was read, and the following was applied, using regex[47] search-and-replace:

- Lines not containing a number were removed
- Lines consisting of aviskorpus codes or patterns matching typical headings were

42 Norsk aviskorpus: http://avis.uib.no/
43 Avdeling for kultur, språk og informasjonsteknologi (Section for culture, language and information technology) at Bergen University
44 Prosjekthistorikk og finansiering: http://avis.uib.no/project.page
45 Perl is a high-level interpreted programming language especially well suited for text manipulation. It was originally written by Larry Wall and developed further by him and a large number of volunteers. Perl is open source and free software. See http://www.perl.org/
46 Script: avk02.pl
47 Regular expressions, or regexes, are text strings describing search patterns. A large number of different types of wildcards and similar operators are available for finding such things as the start of a string, the end of a string, a pattern of exactly *x* characters, a pattern limited to certain types of characters and so on.

removed

- Remaining codes were removed from the line
- Space was removed from after a hyphen if the hyphen did not have a space character before it
- Space was removed from after (, [ and {
- Space was removed from before the following characters: ) ] } . , ! : ;
- Space was removed after " and before " in a balanced pair
- Line-final whitespace was removed

10.000 random lines were then printed to a file and run through the Oslo-Bergen tagger.

## *4.4 Syntactic tagging*

The syntactic tagging was done using the Oslo-Bergen tagger[48]. This is a constraint based tagger, based on the Constraint Grammar framework originally written by Fred Karlsson[49].

A constraint grammar consists of a set of declarative constraints on words and syntax. It works bottom-up, and takes morphologically analyzed word forms as input. The constraints can operate on the word itself or on the features from the morphological analysis, such as part-of-speech, tense and case. The output is a linear structure of words annotated with syntactical and morphological information. Ideally the output is fully disambiguated, but in case of unresolvable ambiguity each word may have more than one annotation.

This structure is especially well suited for the task at hand. The other prevalent output structure from parsers is the parse tree, realized as a set of possible trees, one for each possible reading of the sentence. Using such a structure would result in redundancy and a more complicated task in adding `say-as` markup to the tagged text.

The Oslo-Bergen tagger was originally written by Lingsoft, but the tagger is available for use in research[50]. The Norwegian version was developed by Dokumentasjonsprosjektet and Tekstlaboratoriet at UiO and Aksis at UiB, but large parts of the tagger has since been rewritten by Aksis.

The tagger consists of three parts: The preprocessor, the multitagger and the disambiguator. The preprocessor is a tokenizer, finding word and sentence limits. The multitagger tags the words with morphological information, using information from Norsk ordbank and a compound analyzer. The disambiguator then removes incompatible tags based on morphological and syntactic declarative constraints.

48 En grammatisk tagger for norsk (bokmål): http://www.hf.uio.no/tekstlab/tagger2.html
49 Fred Karlsson – Constraint grammar as a framework for parsing running text, 1990
50 Oslo-Bergen-taggeren (for bokmål og nynorsk): http://decentius.hit.uib.no:8005/cl/cgp/test.html

## 4.5 Semantic tagging

The semantic tagging with `say-as` was done by the help of PHP[51] and a MySQL[52] database. The first step was getting the data into the database. The database had a single table with five fields:

```
CREATE TABLE sentences (
id int(5) NOT NULL auto_increment,
sentence text,
done tinyint(1),
skip tinyint(1),
newsentence text,
PRIMARY KEY (id)
);
```

*id* is a unique number assigned to each sentence, in order to tell them apart. *Sentence* is the original sentence as it appears in the corpus. *Done* is a boolean value that is set to true when the sentence has been tagged with `say-as`. If the sentence turned out to be useless (not containing any numbers, or containing control codes etc. that slipped through the preprocessing stage) s*kip* is set to true. *Newsentence* contains the sentence complete with `say-as` tags.

The input format was an XML file. The file was processed by a PHP script taking each sentence and putting it into a row in the database, with all XML markup intact[53].

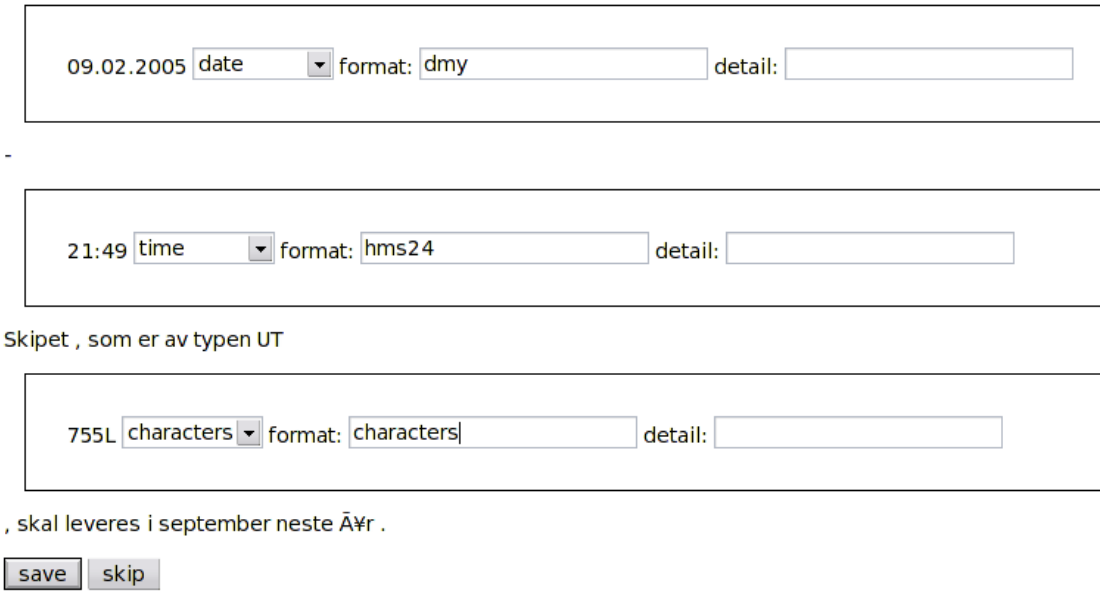
The database was then utilized by another program[54], giving a graphical user interface for the manual part of the process:

51 PHP is a recursive akronym which stands for PHP: Hypertext Preprocessor (originally Personal Home Page tools). As the name suggests, it is a scripting language designed for integration with HTML and other SGML and XML languages. It is well suited for developing web applications, which made it an obvious choice for the web-based user interface of my tagger. See http://www.php.net/

52 MySQL is a very popular open source database. It has a dual license – commercial and GPL. If you use MySQL in a closed source application, you may purchase a commercial license. MySQL is available for free if you use it in an application licensed with a GPL compatible license, which the software developed in this project will be. One of the main advantages of using MySQL today is that nearly all relevant tools has integrated support for it, making it straightforward to use with programming languages such as PHP. See http://www.mysql.com/

53 Script: init.php

54 Scripts: tagger.php and saver.php




| interpret-as | format | detail | separator |
|---|---|---|---|
| date | d m y | | - / . |
| time | hms12 hms24 | | : . " " |
| telephone | ITU-CC country codes | | |
| characters | glyphs characters | grouping digits | |
| cardinal | fraction char | grouping char | |
| ordinal | | | |
| dasp:fraction | | | / |
| dasp:score | | | - |
| dasp:date | d m y | | - / . |

Given that the input was still XML, it would seem sensible to use an XML parser for the remaining processing. However, after some deliberation, I realized that for my purposes this was not needed and would do nothing but add complexity to the script. Instead, the sentence was split into an array with one element or text node per item, keeping the markup intact.

For each text node that contains numbers, the script prints a line in an HTML form, offering a dropdown to choose between the nine possible values of `interpret-as`, as well as a tenth option, "unknown". Two text input fields are also printed, one for the `format` attribute and one for `detail`.

In order to automate the tagging as much as possible, the most likely choice in the dropdown is pre-selected, based on the format of the numerical expression:

1) If it matches a date format with / as the separator, guess date
2) If it contains / and did not match the date format, guess fraction
3) If it matches a date format with – or . as separators, guess date
4) If it contains digits, then a hyphen, then digits, guess score

5) If the number starts with +, guess telephone
6) If it matches a time format with : or . as separators, guess time
7) If it contains am, pm, a.m or p.m, guess time
8) If it contains a . followed by end-of-string, guess ordinal
9) If it contains alphabetic characters, guess characters
10) If it is a four-digit number between 1920 and 2050, guess date
11) If it only contains digits with optional . or , separator characters, guess cardinal
12) Failing all of the above, select unknown

JavaScript could have been used to fill in the most likely values of the format and detail fields based on the current numerical expression and the selection of interpret-as. However, that would introduce a higher risk of errors because it would require too little manual intervention and thus less attention. Such a mechanism would however be trivial to implement should it be desired.

At the bottom of the form are two buttons: "Save" and "Skip". Clicking Skip simply writes a value of 1 to the database field Skip for the row in question. This is used for numerical expressions with no context and sentences which did not contain any numerical expressions – something that happened because the Oslo-Bergen tagger uses a different way of splitting a line into separate sentences than the method used in Norsk aviskorpus.

Clicking Save sends the form data along with the sentence's id to a new script. The data is assembled, combining the values for each row in the submitted form with the numerical expression they describe. The new entries are then inserted into the complete sentence, and the sentence is written to the "newsentence" field in the appropriate row in the database, along with a value of 1 for "done". The browser is redirected back to the tagger script, which fetches the next sentence which does not have either of the done or skip flags set.

The tagging process does not output the say-as markup exactly as it would be used in an SSML document intended for end-users. The main deviation is that the tag is inserted around the entire word, so that "1980-tallet" ends up as `<say-as interpret-as="date" format="y">1980-tallet</say-as>` rather than the more correct `<say-as interpret-as="date" format="y">1980</say-as>`-tallet. Given that the markup will be extracted from the expression in order to run it through memory learning, I chose not to spend time on this detail – it is simply not relevant in the context.

The two most common types of expressions marked up as "unknown" were ranges and durations of time, common in sports results.

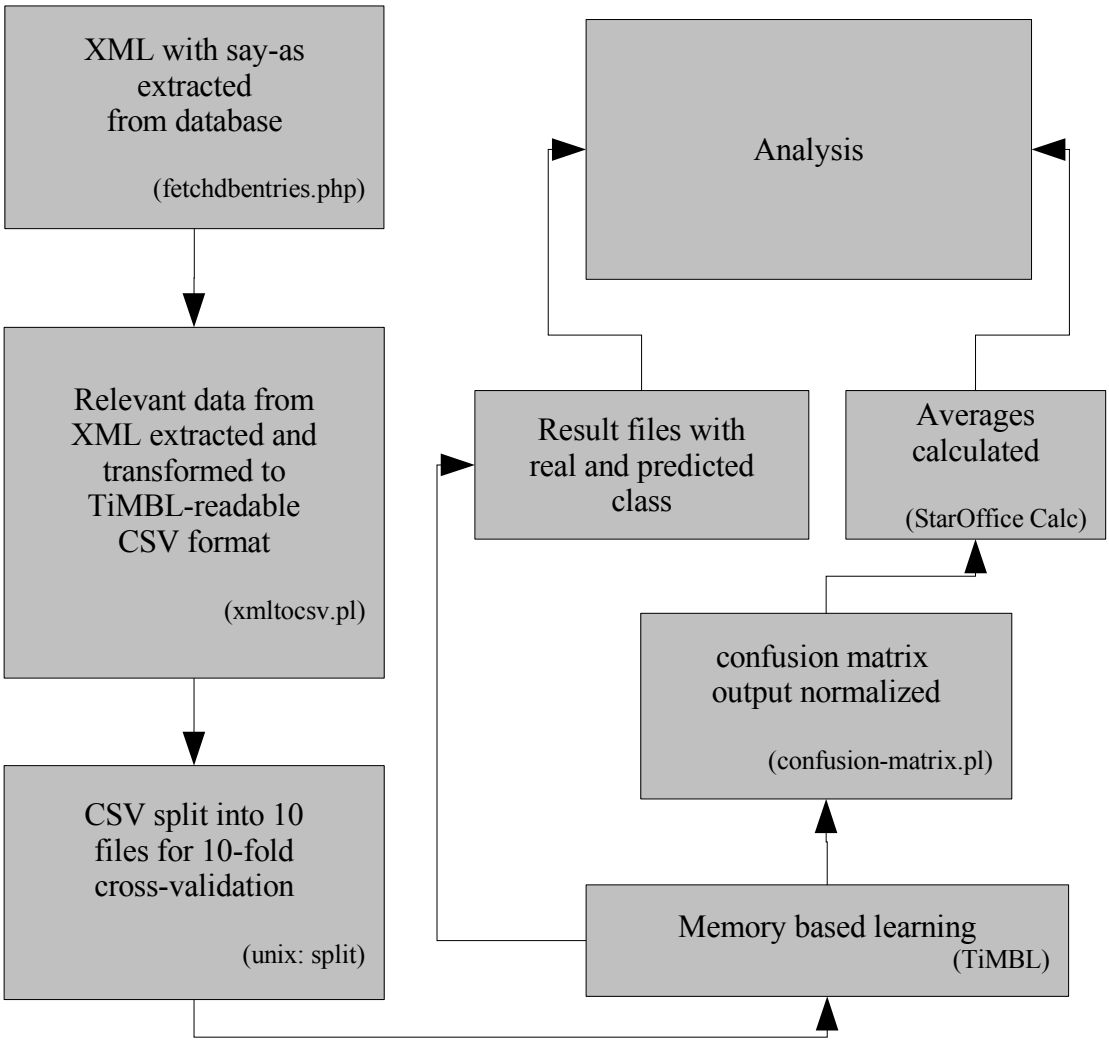
# 5 Memory based learning

## *5.1 Introduction*

This chapter describes how a memory based learning algorithm is applied to the collected, tagged data. The first section gives an introduction to memory based learning. Subsequent sections describe the data and methods used, and the results are presented.

The main aim of this memory learning experiment is to find out whether memory learning is a viable method for use in predicting the semantic category of numerical expressions for the purpose of speech synthesis. In this process, I will also look at what kind of information is necessary and optimal for predicting the category.

The process in this chapter is outlined in the flowchart below:



## *5.2 About MBL*

### 5.2.1 Basics

> Memory based learning is founded on the hypothesis that performance in cognitive tasks is based on reasoning on the basis of similarity of new situations to *stored representations of earlier experiences*, rather than on the application of *mental rules* abstracted from earlier experiences (Daelemans et al. 2004)

Many natural language processing applications are based on rules and heuristics trying to mimic the decision processes of the brain. Some researchers argue that this is not a viable method for processing unrestricted text, because it is not the way the brain works. Memory based learning, they hypothesize, may be better at capturing the processing in the brain and thus provide better accuracy and adaptability to new situations. With MBL, it is not necessary to discover and write rules for every little sub-regularity and exception. This is, however, not only a strength. While MBL may provide us with better natural language processors and let us study some linguistic phenomena from new and interesting angles, other linguistic phenomena are obscured. It is not easy to study what the learner discovers. Thus, it is hard to study and learn about the very sub-regularities that MBL helps us handle[55].

There are, basically, two modes of machine learning: supervised and unsupervised[56]. Memory based learning is a supervised method. You start with a set of pre-defined answers, carving up the domain. Unsupervised learning, on the other hand, starts only with the input data and no pre-defined categories, though it may be given hints about the kind of models it should form. An unsupervised learning algorithm typically finds clusters of input patterns that belong together, organizing itself rather than grouping data according to a pre-defined organization[57].

For the task at hand, control over the output categories is wanted. In order to implement speech synthesis systems that support semantically tagged numerical expressions, the semantic categories need to be clearly defined. Therefore, supervised learning is a good choice.

As mentioned, a memory learning algorithm needs to be provided with material to learn from – examples of the input with the correct answer included. When the learning phase is over, you feed it input in the same format, but with the answers removed. Based on a measure of similarity, the memory learning application predicts the correct answer by comparing the input to the instances it has learned from.

The Tilburg Memory-Based Learner (TiMBL)[58] was chosen as the memory

55 G. Skantze - Transformation-based and Memory-based Learning for Detecting Speech Recognition Errors (year?)
56 Jurafsky & Martin – Speech and Language Processing (2000), p118
57 Peter Dayan, Unsupervised Learning
58 Tilburg Memory-Based Learner

based learning application for this project. It was chosen over The University of Waikato's WEKA[59] because TiMBL can read the C4.5 data format, which requires no headers describing the relations. This makes it more elegant and easier to handle than WEKA's ARFF format, especially when features have large numbers of possible values. TiMBL runs on the command line.

There are many ways of measuring similarity, and what is the most accurate measure varies with the input data. TiMBL provides several different *distance metrics*, described in the TiMBL Reference Guide, chapter 5.1. They are briefly presented below.

The most basic is *Overlap*, where the distance between two patterns is the sum of the differences between the features. It simply counts the number of matching values and chooses the best match. This is a simple measure, and error-prone when the frequencies of the classes to be predicted are uneven.

The *Information Gain* metric attempts to overcome this problem by computing how much information each value contributes to the knowledge about the class label. Thus, some values are weighted heavier than others and are considered more important when the values are matched. This prevents values with low frequency but high informativity from drowning in the average.

However, Information Gain has problems with generalization. The importance of features with many different values is overestimated compared to features with few different values. A feature with unique values will be very closely linked to the classifier in each instance, but useless for predicting new instances. The *Gain Ratio* metric attempts to overcome this by normalizing the measure for features with different number of values. The *Chi-square* metric improves on Gain Ratio by correcting for number of degrees of freedom.

The above distance metrics all do exact matching on feature-values. This method does not necessarily detect all similarities in the data – some possible values may be closer than others. In the numerical expressions domain, not all numbers are used in a cardinal sense, and the value may in fact be an empty string, so it does not make sense to treat the feature as numerical. But it is still useful to be able to say that 1 and 2 are closer than 1 and 1400. The *Modified Value Difference Metric*, MVDM, looks at co-occurrence of values with target classes in order to determine the similarity of a feature's values.

Memory based learning can be applied both for practical purposes, in language engineering applications, and theoretically, for exploring theories in linguistics and psycholinguistics.

### 5.2.2 Memory based learning in language engineering applications

Why are memory based learning models interesting for natural language processing applications?

Human language is in its nature fuzzy. It is hard to find good rules to cover all

59 Weka Machine Learning Project: http://www.cs.waikato.ac.nz/~ml/

cases, and especially hard to generalize to new, previously unseen content. Because language changes continuously, this is a serious shortcoming of rule-based systems. Memory based learning is one way to overcome this problem.

The main feature of memory based learning is perhaps its robustness. MBL is designed to handle new, unknown content and make best guesses based on what it already knows. This makes it well suited to deal with new situations.

It is also easy to get started with. Though the system needs to be presented with a reasonably large selection of examples with answers to learn from, this process is substantially less work-intensive (and knowledge-intensive) than the careful crafting of hundreds or even thousands of rules to cover all possible cases. The learning process is very fast, since it is "lazy" – no processing is done, the examples are simply stored in memory for future reference[60].

If there is a large amount of data, the computational price of classification may be high. This is because the input needs to be compared to every instance stored in memory. This problem can partly be overcome by using more efficient indexing instead of just storing each instance in a flat file[61].

Depending on the nature of the data to be classified, a memory based system may be less accurate than a rule-based one. The accuracy of MBL depends on the amount of data it has to learn from, but also on how well the data can be grouped. The more fragmented the data is, the more exceptions there are, the less accurate the MBL system will typically be.

Memory based learning has been applied to a large number of different tasks within linguistics, such as part-of-speech tagging[62], named entity recognition[63], morphological analysis[64] and more.

### 5.2.3 Supporting linguistic theories with memory learning results

Memory based learning can also be used to investigate theories in linguistics. Based on the regularities and sub-regularities found in the data, a theory may be supported, rejected or altered. A good example of this is the work done on acquisition of stress in Dutch by Daelemans et al[65]. They compare the learning results to metrical theory of stress assignment. Based on their work I have performed similar experiments for English in a past project[66].

The interesting factor is that the learning algorithm has no a priori knowledge about the domain or any specific theory. Thus, it is interesting to compare the learner's results to the "rules" typically found in a theoretical framework. One can experiment with different encodings, leaving out certain parts of the data etc. in order to find what

60 TiMBL Reference Guide, 19
61 TiMBL Reference Guide, 27pp
62 A memory-based part of speech tagger generator, Daelemans et al. (1996)
63 Memory-based named entity recognition using unannotated data. De Meulder et al. (2003)
64 Memory-based morphological analysis, Van den Bosch et al. (1999)
65 The Aquisition of Stress: A Data-Oriented Approach, Daelemans, W., Gillis, S., & Durieux, G. (1994)
66 Computer learning of stress assignment in English: http://epistel.no/dasp/eira/dasp303/

information is important and what is irrelevant in order to determine the category. This information might again be used to alter the rules to better match the data – new factors may be discovered, or known factors rethought.

It is of course vital that the data being tested is not unnecessarily simplified. In order to be a meaningful complement to a theoretical framework, irregularities must be taken into account. If the data is simplified before the learning process, they are in effect being adapted to a known theory, and so cannot be trusted to provide an unbiased look at the domain. Daelemans mention this as a problem of the research on stress assignment by Dresher & Kaye (1990).

## *5.3 Data*

As noted earlier, the data is based on corpus data from Norsk aviskorpus. A newspaper based corpus was chosen for several reasons. First of all, newspapers are a very realistic use case for speech synthesis. Additionally, the text is rapidly produced and updated, which means that the more of the process that can be automated, the better.

Numerical expressions also have a more prominent place in newspaper text than in many other genres, such as fiction. News commonly contains dates, times, sports scores and similar, which are essential to the text.

However, the choice of a limited domain does, almost inevitably, lead to a skewed data set. The frequency of dates, times and sums of money ( here treated as cardinal numbers) will be high, and the dates and times will often have a more formal and uniform format than they would in a wider domain. One example is the lack of `dasp:date` in the dataset. The `dasp:date` format (*d/m -y*) was found in the initial corpus data, which was based on a wide domain, but was not found in the newspaper texts. `dasp:fraction` was also very infrequent.

Since the data is limited to a single language, some formats will be significantly more common than others. This need not be a very important drawback, since the language of a text to apply memory learning on in a real-world application can be assumed to always be known. The data has not been sorted or edited beyond the preprocessing described in the previous chapter. It contains all the irregularities present in the underlying corpus data.

The underlying data contains three classifiers: the main classifier `interpret-as`, and the two sub-classifiers `format` and `detail`, whose values partly depend on `interpret-as`. For this experiment, I chose to omit format and detail. This decision has an empirical and a technological reason. The empirical reason is to keep the data simple in the first pass, in order to find basic problems before trying to make a more fine-grained division. The technological reason is that current memory learning software lack the option to learn more than one classifier. There are ways around this problem – one possible approach is to use the output of one learning phase as input to another. Exploring such approaches is left to later experiments.

### 5.3.1 Encoding

The data was retrieved from the database[67] and encoded into 31 fields plus the classifier[68]. The data forms four logical blocks:

1. The numerical expression as a string, and its syntactic properties
2. The numerical expression split into fields and separator characters
3. The preceding word and its syntactic properties
4. The following word and its syntactic properties

The complete list of fields is as follows:

1. Numerical expression as string
2. POS
3. Gender
4. Number
5. Definite
6. Case
7. Time
8. Additional modifiers (kvant/prop)
9. Numerical value 1
10. Separator 1
11. Numerical value 2
12. Separator 2
13. Numerical value 3
14. Separator 3
15. Numerical value 4
16. Word-1 as string
17. POS
18. Gender
19. Number
20. Definite
21. Case

67 Script: fetchdbentries.pl
68 Script: xmltocsv.pl

22. Time
23. Additional modifiers
24. Word+1 as string
25. POS
26. Gender
27. Number
28. Definite
29. Case
30. Time
31. Additional modifiers
32. Interpret-as (classifier)

The most problematic aspect of the encoding is the set of numerical fields, fields 9 to 15. Memory learning is done by comparing the same fields in each line. This comparison will break when the date "10.12.2005" is encoded with "10" in the first field, and the date "2005" is also encoded into the first field, though we would have wanted to compare it to the first date's fifth field.

Solving this problem is not straightforward. Sorting the content into the correct fields based on content type (the classifier) cannot be done, since the idea is to learn the classifier from the data, not the other way around.

One option is to copy the contents of the first numeric field into all four numeric fields if the expression is a single number[69]. This would mean that years would be compared to years, but it also introduces noise. There is still a difference between these lines and the regular lines with multiple number fields, since the lines with copied fields do not contain separator characters. But in the above example, the advantage of getting to compare 2005 to 2005 is gained at a cost of also having to compare 12 to 2005 – and, of course, we also still have to compare 10 to 2005. Some tests with this encoding were performed.

Another option is to copy lines, instead of fields. For the single number 2005, four lines would be generated – one with "2005" in field 9, one using field 11, one using field 13 and one using field 15. This solution has the drawback of severely skewing the frequency of lines with single fields, in addition to introducing three "wrong" lines for every line with the number in the correct field. The advantage is that the correct line itself would have no noise. Tests with this encoding were left to a later experiment.

The table below shows TiMBL's statistics for the encoded file. The first column shows the number of the field, and corresponds to the list of fields above. The second column, Vals, shows the number of unique values that field has. The third and fourth columns show a measure of how informative the field is with, respectively, the Information Gain and Gain Ratio weighting methods. The higher the number, the more informative the field is interpreted to be.

69 Script: copysingles.pl

| Feats | Vals | InfoGain | GainRatio |
|---|---|---|---|
| 1 | 2537 | 2.0017238 | 0.20652592 |
| 2 | 6 | 0.44968515 | 0.38505992 |
| 3 | 4 | 0.028095013 | 0.073623341 |
| 4 | 3 | 0.12044934 | 0.14310504 |
| 5 | 3 | 0.014706972 | 0.038337164 |
| 6 | 2 | 0.0035038892 | 0.091731388 |
| 7 | 4 | 0.0010653701 | 0.15917935 |
| 8 | 3 | 0.28514022 | 0.29927714 |
| 9 | 576 | 0.96192783 | 0.13840452 |
| 10 | 6 | 0.70372139 | 0.46513637 |
| 11 | 197 | 0.59254435 | 0.22338933 |
| 12 | 6 | 0.087776969 | 0.33483462 |
| 13 | 58 | 0.10097836 | 0.27243386 |
| 14 | 5 | 0.0048867542 | 0.35566297 |
| 15 | 3 | 0.0011720665 | 0.33813560 |
| 16 | 1317 | 1.1151112 | 0.15314819 |
| 17 | 10 | 0.45437288 | 0.16947170 |
| 18 | 4 | 0.047171246 | 0.053541333 |
| 19 | 3 | 0.055570293 | 0.057815802 |
| 20 | 3 | 0.027930550 | 0.042386289 |
| 21 | 4 | 0.0046398567 | 0.042805680 |
| 22 | 6 | 0.043281271 | 0.067761941 |
| 23 | 3 | 0.068763485 | 0.096547666 |
| 24 | 1428 | 1.2303397 | 0.16730194 |
| 25 | 12 | 0.23052458 | 0.11166742 |
| 26 | 4 | 0.26663026 | 0.20088102 |
| 27 | 3 | 0.26485994 | 0.22280589 |
| 28 | 3 | 0.24336509 | 0.23309929 |
| 29 | 4 | 0.0051361137 | 0.055019774 |
| 30 | 6 | 0.024189264 | 0.063598423 |
| 31 | 3 | 0.12467842 | 0.20701825 |

## 5.4 Tests

Below is an overview over the tests run in TiMBL, and a short introduction to them. The outcomes and more detailed analysis of the tests are described in the next section.

| Test # | Description |
|---|---|
| 1 | All features on, overlap metric |
| 2 | All features on, MVDM metric |
| 3 | String, syntax and numerics on, context off, MVDM |
| 4 | String, numerics and context words on, all syntax off, MVDM |
| 5 | String, numerics and context words on, all syntax except POS off, MVDM |
| 6 | Numerics and syntax on, string and context words off, MVDM |
| 7 | All fields on except numerics, MVDM |
| 8.1 | Only numerics on, MVDM |
| 8.2 | Only numerics on, overlap |
| 9.1 | Only the string, MVDM |
| 9.2 | Only the string, overlap |
| 10 | Only the string and its syntax, MVDM |
| 11 | The string, its POS and numerics, MVDM |
| 12 | The string's POS and numerics, no context, MVDM |
| 13.1 | As 5, but with single numbers copied to all number fields |
| 13.2 | As 11, but with single numbers copied to all number fields |

The MVDM metric was chosen for most of the tests, because important parts of the data is believed to benefit from a measure of similarity that is more fine-grained than exact matching. Dates and times have more or less fixed value boundaries, and in order to separate these categories from others, it is useful to treat 10 as more similar to 12 than to 90, instead of seeing all the values as completely separate entities. Some tests were performed using the default overlap metric as a control.

The natural first step is to run a test with all features on. That was done in tests 1 and 2. The subsequent tests have certain fields turned off, to investigate how the presence and absence of various kinds of information, such as context and syntactical details, affect the outcome. With 30 different features it is of course not a viable solution to test all possible combinations. The features form partly overlapping groups of related information, and tests were run with various combinations of these groups.

In test 3, all context is turned off. In test 4, the context words are turned on, but all syntactical details are off. Test 5 is identical except that part-of-speech is turned on. In Test 6, all syntactical details are on, but the words are off, including the numerical string itself. However, the numerical data is still present in the numerical value fields.

In test 7, the numerical value fields are turned off, but all other fields are on.

Tests 8 and 9 have quite minimal amounts of data, respectively only the numerical string and only the numerical value fields. In test 10 the syntactical details of the string is added. Test 11 combines the numerical string, the numerical value fields and the string's part-of-speech, while test 12 leaves out the string itself, leaving only the part-of-speech and the numeric value fields.

Tests 13.1 and 13.2 are identical to previous tests, but performed on slightly different data. In these tests, if only the first numerical value field contained data, and the other numerical value fields did not, the contents of the first field were copied to the other.

## *5.5 Results*

In this section the results from the memory based learning are presented. I will present some different ways of looking at the data, and go into more detail about some parts of the results.

### 5.5.1 Averages

Below is a summary of the results, with averages for each test run.

| Test # | % | Description |
|---|---|---|
| 1 | 93 | All features on, overlap metric |
| 2 | 95 | All features on, MVDM metric |
| 3 | 94 | String, syntax and numerics on, context off, MVDM |
| 4 | 94 | String, numerics and context words on, all syntax off, MVDM |
| 5 | 95 | String, numerics and context words on, all syntax except POS off, MVDM |
| 6 | 94 | Numerics and syntax on, string and context words off, MVDM |
| 7 | 92 | All fields on except numerics, MVDM |
| 8.1 | 85 | Only numerics on, MVDM |
| 8.2 | 84 | Only numerics on, overlap |
| 9.1 | 86 | Only the string, MVDM |
| 9.2 | 86 | Only the string, overlap |
| 10 | 87 | Only the string and its syntax, MVDM |
| 11 | 94 | The string, its POS and numerics, MVDM |
| 12 | 93 | The string's POS and numerics, no context, MVDM |
| 13.1 | 95 | As 5, but with single numbers copied to all number fields |
| 13.2 | 94 | As 11, but with single numbers copied to all number fields |

*Table 4: Summary of memory learning averages*

The scores range from 84 to 95 percent accuracy, forming two distinct groups: One group with results ranging from 84 to 87 percent, and the other ranging from 92 to 95 percent. The distance between the groups (5 points) is larger than the internal distance between the results in each group (3 points in both).

As we can see, the scores for test runs 8 and 9, numerics-only and string-only, are very close and make up most of the lower-score group. The similar score is to be expected since the two tested field groups contain roughly the same data, only encoded differently. However, for strings that contain other symbols than numbers and separator characters, those other symbols (normally alphabetic characters) are not included in the numeric encoding. This may account for the slightly better accuracy of the string-only test run. That these tests only have about 85% accuracy is a clear sign that some kind of context is important for correctly classifying numerical expressions. Looking only at the numerical expression itself is not enough.

Two tests achieve a score of 95 percent – tests 2 and 5. Test 2 includes all features, while test 5 excludes syntactical details except part-of-speech. This suggests that other syntactic data than part-of-speech are irrelevant. Given that test run 5 ties for

the first place based on less information than its competitor, it will be considered the best test result.

Including POS only improves accuracy a little bit compared to no syntax, test run 4. This is good news for use in applications, because it eases the task significantly if one does not have to parse the text syntactically to determine the semantics of numbers.

Good results were also achieved when ignoring all context, provided that the part-of-speech was included, test run 11. In fact, the score is the same when including POS but excluding context, as when excluding POS and including context. Including both only marginally improves the results, but excluding both leads to a significant drop in accuracy.

The format of the string does play a significant role when context is ignored. Comparing test runs 3 and 10 shows that just by ignoring the features containing the string split in numbers and separator characters, accuracy drops from 94% to 87%. However , if both syntax and context are present, accuracy is back to 92% (test run 7). In other words, the importance of the three feature groups context, syntax and string format are highly interrelated.

In test run 5, 2825 out of 7913 correctly classified instances were exact matches. That is a relatively high number (36%, approximately one third), but it is important to notice that two thirds of the correctly classified instances were not exact matches. This shows that the learner is able to generalize.

### 5.5.2 Weighting

The best and one of the second-best options were tested with various weighting methods:

| % | Weighting method |
|---|---|
| 95 | Gain ratio |
| 95 | No weighting |
| 94 | Info gain |
| 94 | Chi square |

*Table 5: String, numerics, context words and POS, various weighting (based on test run 5)*

| % | Weighting method |
|---|---|
| 94 | Gain ratio |
| 94 | No weighting |
| 94 | Info gain |
| 94 | Chi square |

*Table 6: String, POS and numerics, no context, various weighting (based on test run 11)*

The lack of variation in accuracy reveals that the data are not particularly sensitive to the weighting method.

### 5.5.3 Confusion matrix

A confusion matrix is an overview over predicted and real classes, displayed in a table[70]. This is the matrix for the best test run:

| | time | cardinal | date | dasp:score | ordinal | characters | telephone | unknown | dasp:fraction |
|---|---|---|---|---|---|---|---|---|---|
| time | **755** | 13 | 10 | 0 | 0 | 0 | 0 | 4 | 0 |
| cardinal | 23 | **4423** | 34 | 0 | 49 | 25 | 4 | 5 | 0 |
| date | 6 | 34 | **1258** | 3 | 1 | 1 | 1 | 3 | 2 |
| dasp:score | 0 | 0 | 0 | **334** | 0 | 2 | 0 | 11 | 0 |
| ordinal | 0 | 45 | 0 | 0 | **700** | 6 | 0 | 4 | 0 |
| characters | 1 | 47 | 4 | 1 | 5 | **129** | 0 | 6 | 0 |
| telephone | 0 | 12 | 5 | 0 | 0 | 0 | **98** | 0 | 0 |
| unknown | 3 | 24 | 25 | 9 | 5 | 3 | 0 | **213** | 0 |
| dasp:fraction | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **3** |

*Table 7: Confusion matrix for test run 5 (accuracy: 95%)*

The table shows the real class horizontally, and the predicted class vertically. Ideally, they should match perfectly, giving a diagonal from top left to bottom right with numbers, and zeros in all the other fields. When the learner makes a mistake, the error can be found in the intersection between the real and the predicted class.

The confusion matrix is useful for finding out which classes are particularly problematic for the learner. For instance, the table above shows that ordinals are often mistaken for cardinals. This is not surprising, since ordinal numbers that lack a dot look exactly like cardinal numbers. Information about part-of-speech and format is not

70 The script confusion-matrix.pl was written to format the output from TiMBL to a table that could be imported into a spreadsheet

enough to disambiguate.

From this information, a hypothesis can be made that the recognition of ordinals is more accurate with context on than off, since context is the factor that differs the most between cardinal and ordinal. Below are the confusion matrices for test runs 2 and 3, with and without context. These test runs were chosen because they are maximal, with no data group other than context manipulated. This is important because, as shown earlier, leaving out more than one group of information causes a drop in performance regardless of which two groups are removed.

| | time | cardinal | date | dasp:score | ordinal | characters | telephone | unknown | dasp:fraction |
|---|---|---|---|---|---|---|---|---|---|
| time | 755 | 14 | 9 | 0 | 0 | 0 | 0 | 4 | 0 |
| cardinal | 17 | 4438 | 29 | 0 | **47** | 23 | 4 | 5 | 0 |
| date | 6 | 33 | 1262 | 3 | 1 | 0 | 1 | 3 | 0 |
| dasp:score | 0 | 1 | 1 | 337 | 0 | 0 | 0 | 8 | 0 |
| ordinal | 1 | **42** | 0 | 0 | 704 | 5 | 0 | 3 | 0 |
| characters | 1 | 51 | 3 | 0 | 6 | 125 | 1 | 6 | 0 |
| telephone | 0 | 14 | 4 | 0 | 0 | 0 | 97 | 0 | 0 |
| unknown | 3 | 25 | 24 | 9 | 2 | 8 | 0 | 211 | 0 |
| dasp:fraction | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |

*Table 8: Confusion matrix for test run 2 : context included (accuracy: 95%)*

| | time | cardinal | date | dasp:score | ordinal | characters | telephone | unknown | dasp:fraction |
|---|---|---|---|---|---|---|---|---|---|
| time | 728 | 42 | 11 | 0 | 0 | 1 | 0 | 0 | 0 |
| cardinal | 8 | 4480 | 27 | 0 | **4** | 26 | 6 | 12 | 0 |
| date | 3 | 57 | 1235 | 2 | 0 | 2 | 4 | 6 | 0 |
| dasp:score | 0 | 0 | 1 | 330 | 0 | 0 | 0 | 16 | 0 |
| ordinal | 1 | **73** | 0 | 0 | 675 | 3 | 0 | 3 | 0 |
| characters | 2 | 53 | 2 | 0 | 2 | 128 | 0 | 6 | 0 |
| telephone | 0 | 40 | 3 | 0 | 0 | 0 | 72 | 0 | 0 |
| unknown | 2 | 25 | 12 | 14 | 3 | 5 | 0 | 220 | 1 |
| dasp:fraction | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |

*Table 9: Confusion matrix for test run 3: context excluded (accuracy: 94%)*

Comparing these two tables, we can see that when context is included, 42 numbers that are really ordinal are mispredicted as cardinal, whereas 47 numbers that are really cardinal are mispredicted as ordinal. When context is excluded, 73 numbers that are really ordinal are mispredicted as cardinal, while only 4 numbers that are really cardinal are mispredicted as ordinal. This supports the hypothesis. The number of ordinals

mispredicted as cardinal went down from 73 to 42 if context was included. However, this is at a cost: Cardinals misinterpreted as ordinal went up from 4 to 47!

As mentioned, the numerical fields are a source of trouble. One proposed solution was to copy single values to all numerical fields, thus ensuring that corresponding parts of a number are compared. The tables below show the results of these experiments:

| | time | cardinal | date | dasp:score | ordinal | characters | telephone | unknown | dasp:fraction |
|---|---|---|---|---|---|---|---|---|---|
| time | 755 | 13 | 10 | 0 | 0 | 0 | 0 | 4 | 0 |
| cardinal | 23 | 4423 | **34** | 0 | 49 | 25 | 4 | 5 | 0 |
| date | 6 | **34** | 1258 | 3 | 1 | 1 | 1 | 3 | 2 |
| dasp:score | 0 | 0 | 0 | 334 | 0 | 2 | 0 | 11 | 0 |
| ordinal | 0 | 45 | 0 | 0 | 700 | 6 | 0 | 4 | 0 |
| characters | 1 | 47 | 4 | 1 | 5 | 129 | 0 | 6 | 0 |
| telephone | 0 | 12 | 5 | 0 | 0 | 0 | 98 | 0 | 0 |
| unknown | 3 | 24 | 25 | 9 | 5 | 3 | 0 | 213 | 0 |
| dasp:fraction | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |

*Table 10: Confusion matrix for test run 5 (accuracy: 95%)*

| | time | cardinal | date | dasp:score | ordinal | characters | telephone | unknown | dasp:fraction |
|---|---|---|---|---|---|---|---|---|---|
| time | 714 | 54 | 12 | 0 | 0 | 1 | 0 | 1 | 0 |
| cardinal | 6 | 4454 | **35** | 0 | 31 | 22 | 3 | 12 | 0 |
| date | 4 | **62** | 1227 | 2 | 0 | 4 | 4 | 6 | 0 |
| dasp:score | 0 | 0 | 1 | 330 | 0 | 2 | 0 | 14 | 0 |
| ordinal | 0 | 85 | 0 | 0 | 666 | 2 | 0 | 2 | 0 |
| characters | 1 | 78 | 2 | 1 | 5 | 101 | 0 | 5 | 0 |
| telephone | 0 | 41 | 2 | 0 | 0 | 0 | 72 | 0 | 0 |
| unknown | 2 | 24 | 13 | 14 | 4 | 1 | 0 | 223 | 1 |
| dasp:fraction | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |

*Table 11: Confusion matrix for test run 13.1 (same parameters as test run 5, but with copied fields. Accuracy: 95%)*

| | time | cardinal | date | dasp:score | ordinal | characters | telephone | unknown | dasp:fraction |
|---|---|---|---|---|---|---|---|---|---|
| time | 724 | 45 | 12 | 0 | 0 | 0 | 0 | 1 | 0 |
| cardinal | 5 | 4468 | **31** | 0 | 20 | 21 | 6 | 12 | 0 |
| date | 4 | **57** | 1232 | 2 | 0 | 4 | 4 | 6 | 0 |
| dasp:score | 0 | 0 | 1 | 330 | 0 | 2 | 0 | 14 | 0 |
| ordinal | 0 | 78 | 0 | 0 | 673 | 2 | 0 | 2 | 0 |
| characters | 1 | 57 | 2 | 1 | 3 | 123 | 0 | 6 | 0 |
| telephone | 0 | 41 | 2 | 0 | 0 | 0 | 72 | 0 | 0 |
| unknown | 2 | 24 | 13 | 14 | 4 | 1 | 0 | 223 | 1 |
| dasp:fraction | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |

*Table 12: Confusion matrix for test run 11 (accuracy: 94%)*

| | time | cardinal | date | dasp:score | ordinal | characters | telephone | unknown | dasp:fraction |
|---|---|---|---|---|---|---|---|---|---|
| time | 727 | 47 | 7 | 0 | 0 | 0 | 0 | 1 | 0 |
| cardinal | 5 | 4469 | **31** | 0 | 18 | 22 | 6 | 12 | 0 |
| date | 6 | **56** | 1233 | 2 | 0 | 1 | 5 | 6 | 0 |
| dasp:score | 0 | 0 | 1 | 328 | 0 | 2 | 0 | 16 | 0 |
| ordinal | 1 | 77 | 0 | 0 | 674 | 1 | 0 | 2 | 0 |
| characters | 1 | 57 | 2 | 1 | 3 | 123 | 0 | 6 | 0 |
| telephone | 0 | 41 | 2 | 0 | 0 | 0 | 72 | 0 | 0 |
| unknown | 2 | 23 | 14 | 16 | 3 | 2 | 0 | 222 | 0 |
| dasp:fraction | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |

*Table 13: Confusion matrix for test run 13.2 (same parameters as test run 11, but with copied fields. Accuracy: 94%)*

By looking closer at the date/cardinal intersections, we can determine whether the dates were less likely to be misinterpreted as cardinal numbers if single fields were copied. In test run 5, we see that 34 dates were incorrectly classified as cardinal, and the same number of cardinals were incorrectly classified as dates. Comparing this result to a test run with the same parameters on a data file with copied fields, we see that 62 dates were incorrectly classified as cardinal, while 35 cardinals were incorrectly classified as dates – in other words a slight drop in accuracy, rather than the expected increase. The second test pair shows similar results, comparing 57/31 to 56/31.

The only likely explanation for these results is the added noise, which is significant. For each value in the correct field, there are three values in the wrong fields.

The most common mistake is that other classes are mistaken to be cardinals. Given that the frequency of cardinals is much larger than the other classes, this is not unexpected. The most frequent class will often be most prominent when resolving ties

in classification. Additionally, cardinal numbers are often simple numbers with no separator characters. All the other classes overlap with this format.

In the overview over averages, we saw that the results formed two groups. So far we have looked at confusion matrices for the good results, but it is also interesting to take a closer look at the bad results. Below are matrices for test runs 8 (numeric fields only) and 9 (string only):

| | time | cardinal | date | dasp:score | ordinal | characters | telephone | unknown | dasp:fraction |
|---|---|---|---|---|---|---|---|---|---|
| time | 709 | 57 | 12 | 0 | 1 | 0 | 0 | 3 | 0 |
| cardinal | 7 | 4370 | 22 | 0 | 119 | 21 | 3 | 21 | 0 |
| date | 4 | 93 | 1196 | 2 | 0 | 1 | 5 | 6 | 2 |
| dasp:score | 0 | 0 | 1 | 326 | 0 | 5 | 0 | 15 | 0 |
| ordinal | 1 | **588** | 0 | 0 | 150 | 16 | 0 | 0 | 0 |
| characters | 1 | 123 | 1 | 2 | 33 | 27 | 0 | 5 | 1 |
| telephone | 0 | 41 | 2 | 0 | 0 | 0 | 72 | 0 | 0 |
| unknown | 5 | 27 | 14 | 14 | 1 | 0 | 0 | 221 | 0 |
| dasp:fraction | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 |

*Table 14: Confusion matrix for test run 8 (accuracy: 85%)*

| | time | cardinal | date | dasp:score | ordinal | characters | telephone | unknown | dasp:fraction |
|---|---|---|---|---|---|---|---|---|---|
| time | 322 | **454** | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| cardinal | 2 | 4532 | 20 | 0 | 0 | 3 | 6 | 0 | 0 |
| date | 2 | 207 | 1094 | 1 | 0 | 0 | 4 | 1 | 0 |
| dasp:score | 0 | 50 | 0 | 296 | 0 | 0 | 0 | 1 | 0 |
| ordinal | 0 | 99 | 0 | 0 | 656 | 0 | 0 | 0 | 0 |
| characters | 1 | 98 | 0 | 0 | 0 | 92 | 0 | 2 | 0 |
| telephone | 0 | 42 | 1 | 0 | 0 | 0 | 72 | 0 | 0 |
| unknown | 0 | 188 | 2 | 6 | 0 | 1 | 0 | 85 | 0 |
| dasp:fraction | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |

*Table 15: Confusion matrix for test run 9 (accuracy: 86%)*

Both perform equally badly, having too little information available to distinguish between different classes. But the errors made are noticeably different. The test run with only numeric fields is very confused about ordinals, misinterpreting 588 ordinals as cardinals. As already demonstrated, this task is a difficult one when context is not available. Thus, the result is not surprising. However, the test run with only the string performs much better classifying the ordinals. Instead, it mistakes times for cardinals. A possible reason for this is that the fields in times are not separated into several fields,

but instead presented to the learner as a single chunk. This theory is supported by the fact that the second most frequent error is that dates are mistaken for cardinals. Dates and times are the categories that most frequently consist of several fields.

### 5.5.4 Specific errors

So far we've looked at the accuracy averages, showing overall performance, and the confusion matrices showing the amount of errors per category. It can be useful to dig even deeper and take a closer look at the specific errors that occur, especially those that are frequent. TiMBL provides result files for each test run, showing each input line with the predicted class at the end. The result file from test run 5 was filtered to leave only the errors[71], and sorted by category. A typical error looks like this:

*4-tallet,subst,noyt,ent,be,,,,4,,,,,,,tror,verb,,,,,pres,,er,verb,,,,,pres,,cardinal,characters*

There are several similar examples, with words like "21-tiden" and "2-åringen" – all of them cardinals that are misclassified as character strings. This is an easily understandable error, given that these words contain many alphabetic characters, a common feature of character strings. It is not easy for the learner to distinguish between character strings and cardinals that happen to be connected to a regular word. In a tagging process, this mistake would cause the tags to be inserted in the wrong place in addition to having the wrong values. What should be tagged as `<say-as interpret-as="cardinal">4</say-as>`-tallet would instead be tagged as `<say-as interpret-as="characters">4-tallet</say-as>`.

Among the cardinals misclassified as dates, four-digit numbers between 1900 and 2000 are prevalent. The opposite error is also quite common. This, too, is an obvious pitfall. Telling a year from a cardinal in the same range is difficult, and depends heavily on context. Below are some examples:

*2000,det,,fl,,,,kvant,2000,,,,,,,maks,,noyt,ent,ub,,,,tegn,subst,noyt,fl,ub,,,,cardinal,date*

*1944,det,,fl,,,,kvant,1944,,,,,,,i,prep,,,,,,,bare,,,,,,,,date,cardinal*

Most of the cardinals misclassified as ordinals and the other way around are simple numbers without a trailing dot, as the ones below. These are nearly identical, and cannot be distinguished with the information given here. This suggests that accuracy may be improved by including a wider context window.

*8,det,,fl,,,,kvant,8,,,,,,,c,,,,,,,),,,,,,,,,cardinal,ordinal*[72]

71 Script: finderrors.pl

72 A single c in a field means that the original text contained a comma in that place, which had to be changed in order to use it in a comma-separated file

*6,det,,fl,,,,kvant,6,,,,,,,c,,,,,,,,),,,,,,,,ordinal,cardinal*

The cardinals misclassified as times are typically two-digit numbers within the range of a time of day expressed as an hour, such as the one below:

*13,det,,fl,,,,kvant,13,,,,,,,Apollo,subst,mask,ent,ub,,,,|,,,,,,,,cardinal,time*

An investigation of character strings misclassified as cardinals reveals several strings of digits only, which is an understandable mistake. But there is also a surprisingly high number of digit/alphabetic character mixes such as "mp3", "TV2", "3-D" etc. which show that some research is needed to find out how accuracy could be improved with regard to acronyms like these.

Among the unknowns, there were a lot of sports times on the minute.second.hundredth-of-a-second format. These look like times of day, even though many of them are out of the normal range for times of day:

*14.48.97,det,,,,,,kvant,14,.,48,.,97,,,c,,,,,,,,c,,,,,,,,unknown,date*

## *5.6 Summary*

We looked at the data, the tests performed, and analyzed results from various angles, looking at average performance, confusion matrices and specific errors.

As seen, accuracy up to 95% was achieved. Removing the most computationally expensive part of the process, the syntactic tagging, does not have a significant impact on the result. Accuracy up to 94% was achieved without any syntactical information.

As expected, the most problematic aspect for the learner is where formats overlap between categories. Analysis shows signs that more context may be necessary to improve the accuracy.

# 6 Discussion and conclusion

## *6.1 Discussion*

This project had a double aim. The first aim was to investigate and improve on the semantic categories suggested in a W3C Working Group Note for numerical expressions in speech synthesis. The other was to explore memory based learning as a method for applying the above categories to numerical expressions in running text.

Norwegian and English corpus data was examined to give an empirical basis for the categorization. We saw that the six pre-defined categories for a large part covered the data they were designed for, though some problems were noted. A significant amount of data was not covered by the pre-defined categories. These examples were categorized and discussed. A selection was made of which categories to keep in the further work, based on distribution and relevance, as well as the technical limitations imposed by SSML.

Some limitations in the selection process should be noted. Only two languages, Norwegian and English, were covered. As broad international coverage is an aim of SSML, more languages should be investigated. A related problem is that the data is limited. It is based on the available numerical expressions in three corpora, and cannot claim to be comprehensive, though it does in all likelihood give a good indication of which numerical expressions are most common. Finally, the selection and categorisation is a subjective process. Though some guidelines were set, the decisions reflect the opinions and intuitions of one person. This introduces an aspect of "armchair linguistics" into an otherwise data-driven project. Further studies might benefit from a more detailed statistical analysis of the frequencies and distribution of various categories.

The data was further processed before applying memory based learning. There are potential sources of error in this process. Some assumptions were made when reversing the one-word-per-line format of Norsk aviskorpus. It is possible that some of the data was not reversed to it exact original form, especially if the original data had irregularities in the punctuation. Thus, the data may be slightly more normalized than the original input to Norsk aviskorpus. It is my belief that this problem is marginal, and that it does not affect the outcome significantly.

The data was syntactically tagged using the Oslo-Bergen tagger. The result is only as good as the rules in the automatic tagger. Little documentation of the tagger is available, and it is outside the scope of this thesis to evaluate the quality of the tagging process. Given that the results with and with syntactical data included are so similar, and that for application use it is preferred not to include syntactical data because of the computational cost, this is not seen as a big problem. The results could theoretically be improved with better syntactical tagging, but this might in fact be a circular problem. It is not unthinkable that the syntactical tagging could be improved if the tagger had knowledge of the semantics of each numerical expression – the knowledge this project is trying to gain.

Close to ten thousand lines were semantically tagged by hand. Ideally, the

training set would be even larger. Ten thousand lines were considered a reasonable trade-off between practical feasibility and the desire to have a base for generalizing that is as large as possible.

Occasional errors in such a process are almost inevitable. A closer inspection of the detailed results file with real and predicted class reveals some lines that are almost certainly wrongly tagged, and that the learner in fact correctly classified. Some examples are presented below.

*15,det,,fl,,,,kvant,15,,,,,,,klokken,subst,mask,ent,be,,,,den,det,mask,ent,,,,,cardinal,time*

This line is tagged as cardinal, though if we combine the context and the numerical expression we get "... klokken 15 den ...", which indicates that the number is probably a time. The learner classified it as a time.

*12/09/2004,subst,,,,,,,12,.,9,.,2004,,,,,,,,,,,-,,,,,,,cardinal,date*

The sequence "12/09/2004" is almost certainly a date, and definitely not a cardinal number as it was tagged. The learner classified it as a date.

*2000,det,,fl,,,,kvant,2000,,,,,,,på,prep,,,,,,,kroner,subst,mask,fl,ub,,,,date,cardinal*

"... på 2000 kroner ..." indicates that the content is a cardinal number, which is what the learner classified it as. It was however tagged as a date.

The vast majority of inspected lines seem to be correctly tagged. That some errors occur in the material is unlikely to affect the overall performance significantly.

There is little literature about recognizing and categorizing numerical expressions in text. Thus, there is not much previous work to refer to as a theoretical basis for the work done in this thesis. The problem has much in common with the recognition and categorization of proper names, a topic that has been the focus of research from several angles. Evaluating the relationship between numerical expressions and proper names has not been a focus in this thesis, but it is not unthinkable that research in the numerical expressions domain could benefit from knowledge about proper name recognition.

It would be interesting to know how existing speech synthesis engines deal with numerical expressions internally. Several makers of popular synthesis software were contacted with requests for information, but none replied. Conversations with users of such software suggested that handling is very basic, and that a string such as "1/2" is typically read as "one slash two". This means that users of synthesis engines would benefit from more detailed input data. The work done in this thesis shows that this is a goal that can be achieved.

## 6.2 Further work

The main problem that remains to be solved is subcategorization. In the context of SSML, this means inclusion of format and detail. Future projects should look at ways to include these attributes where applicable. Currently, only the main category is known. Using dates as an example, it is desirable to also know which parts of the date are present and how they are ordered. 10.12 could be either day-month or month-day. This data was included when tagging the texts semantically and is readily available for future experiments.

This experiment did not include learning of the format and detail attributes. Learning multiple classifiers in one pass is not an available feature in current memory based learners. The ideal solution would be to implement a memory learner, or modify an existing one, to include this feature. That is a complex task, and there may be easier ways around the problem. A simpler approach is to use output from one round of learning as input to another. Inclusion of format and detail is a problem that must be solved before the automatic tagging can be used in real-world applications.

The range category was excluded for technical reasons. As ranges of various kinds are quite common, it is desirable to take a closer look at this problem. It is necessary to determine whether the category is really needed, preferably with a wide selection of languages as a reference. If the decision is that the category is needed, it must be implemented in a suitable way. As shown in this thesis, this is not straightforward. The most viable solution is perhaps to modify the SSML specification to allow nesting of the `say-as` element under certain, well-defined conditions.

As seen from the learning results, the learner does not always have enough context available to distinguish between different categories, in particular ordinal vs. cardinal. Future work should experiment with larger context windows and investigate whether accuracy can be improved in these areas.

One interesting approach would be to combine memory based learning with some manually introduced biases. Trigger words, such as "klokken" for times, is one possibility, though with a sufficiently large training set the memory learner should have no problems learning these automatically. A more interesting area to include biases for is the normal ranges typically found in dates and times. Even when using the MVDM metric that should minimize this problem, the learning results showed that the learner had difficulties distinguishing between times of day and strings that look like times of day but really are sports results.

A human would in most cases be able to tell them apart simply because the sports results are out of range of a normal time of day. A time of day is restricted – hours range from 00 to 24, minutes and seconds from 00 to 59. A sports result is usually also restricted, but contains other fields with different limits. A common format is minutes and seconds both ranging from 00 to 59, and hundredths of a second ranging from 00 to 99. If the learning algorithm could be told to prefer values within certain ranges when assigning the time classifier, accuracy could be improved. One would need to experiment with weighting of the bias to maximize the gain while avoiding losing too much robustness when it comes to handling unknown content.

Combining memory based learning and manually introduced biases is not supported in a straightforward way in current memory based learning implementations.

Several technical challenges must be met before the idea of introducing biases can be pursued.

Result files from the preferred test run were analyzed to find patterns in the specific errors that occurred. These files can also be used to compare different test runs, in order to find out which combinations of information cause which types of errors. This information can be used to improve future experiments, and such a comparison should be considered before using the material for further testing.

Though the work done here is specifically aimed at the domain of speech synthesis, it is not unlikely that the results can contribute to knowledge about the semantic classification of numbers in a wider context. Substantial amounts of work remain before a complete theoretical framework for the domain can be constructed.

## *6.3 Conclusion*

The first aim of this project was to investigate and improve on the semantic categories suggested in a W3C Working Group Note for numerical expressions in speech synthesis. It was found that, for Norwegian and English, the suggested categories largely covered the types of numerical expressions they were designed to cover. It was also found that the categories did not cover the whole domain of numerical expressions. New categories were defined to cover additional parts of the domain.

The resulting set of categories must be seen as an improvement over the previously defined set. The investigation has added to the knowledge of numerical expressions in an international perspective, by providing examples an analysis of cases where the W3C defined categories did not cover content in a non-English language, in this case Norwegian.

The second aim of this project was to explore memory based learning as a method for applying the above categories to numerical expressions in running text. Accuracy of up to 95% was achieved, and a number of problem areas were found and analyzed.

The high accuracy shows that memory based learning is promising as a method for automatically finding the correct semantic category of a number in running text. The ability to tag texts automatically means that a much larger amount of texts can be prepared for speech synthesis with accurate reading of numerical expressions, than if this work was to be done manually. The main benefit of memory based learning is that the system can deal with previously unseen content in a more robust way than a rule-based system would be able to.

# 7 References

## *7.1 Publications*


Chomsky, N. 2002. Syntactic Structures. Second Edition. Mouton de Gruyter, Berlin.

Daelemans, W., J. Zavrel, K. van der Sloot, A. van den Bosch, 2004. TiMBL: Tilburg Memory-Based Learner Reference Guide. ILK Technical Report – ILK 04-02.

Daelemans, W., J. Zavrel, P. Berck, and S. Gillis. 1996. MBT: A memory-based part of speech tagger generator, Daelemans et al. In E. Ejerhed and I. Dagan (editors). Proceedings of Fourth Workshop on Very Large Corpora, pages 14-27. ACL SIGDAT.

Daelemans, W., S. Gillis, and G. Durieux. 1994. The Aquisition of Stress: A Data-Oriented Approach. Computational Linguistics, 20(3):421-451.

Dayan, P. 1999. Unsupervised Learning. In Wilson, R. A. and Keil, F. C. (editors). The MIT Encyclopedia of the Cognitive Sciences. Cambridge, MA: MIT Press.

De Meulder, F. and W. Daelemans. 2003. Memory-based named entity recognition using unannotated data. In W. Daelemans and M. Osborne (editors). Proceedings of CoNLL-2003, pages 208-211. Edmonton, Canada.

Hurford, J. 1987. Language and Number – The Emergence of a Cognitive System. Basil Blackwell Ltd.

Jurafsky, D and J. Martin, 2000. Speech and Language Processing – An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Prentice Hall.

Karlsson, F. 1990. Constraint grammar as a framework for parsing running text. In Hans Karlgren (editor). Papers presented to the 13th International Conference on Computational Linguistics, Vol. 3. Helsinki, pages 168-173.

Losnegaard, G. 2005. Syntaksen i talluttrykk. Master thesis, University of Bergen.

McDonald, D. 1996. Internal and external evidence in the identification and semantic categorization of proper names. In B. Boguraev and J. Pustejovsky, editors, Corpus processing for lexical acquisition, pages 21—39.

Monstad, E. 2003. Computer Learning of Stress Assignment in English. http://epistel.no/dasp/eira/dasp303/

Olsson, M. 1997. Swedish numerals in an international perspective. Lund University Press.

Ray, E. 2003. Learning XML. O'Reilly.

Skantze, G. Transformation-based and Memory-based Learning for Detecting Speech Recognition Errors. Available at http://w3.msi.vxu.se/~nivre/teaching/gslt/ml_papers.html

The American Heritage Dictionary of the English Language, Fourth Edition. 2004. Houghton Mifflin Company.

Van den Bosch, A. and W. Daelemans. 1999. Memory-based morphological analysis. In Proceedings of the 37th Annual Meeting of the ACL, pages 285-292, San Francisco, CA. Morgan Kaufmann.

Wiese, H. 1997. Zahl und Numerale – Eine Untersuchung zur Korrelation konzeptueller und sprachlicher Strukturen. Akademie Verlag GmbH, Berlin.

## *7.2 Web sites*

*All sites last visited January 30 2006.*

Ask Dr. Math: http://mathforum.org/library/drmath/view/63884.html

Corpus linguistics: http://ling.uib.no/~desmedt/cursus/corpus/syllabus/intro.html

Definition of a Corpus: http://bowland-files.lancs.ac.uk/monkey/ihe/linguistics/corpus2/2defin.htm (supplement to the book Corpus Linguistics, see http://bowland-files.lancs.ac.uk/monkey/ihe/linguistics/contents.htm)

En grammatisk tagger for norsk (bokmål): http://www.hf.uio.no/tekstlab/tagger2.html

Extensible Markup Language (XML): http://www.w3.org/XML/

Generalized Markup Language:
http://en.wikipedia.org/wiki/Generalized_Markup_Language

HyperText Markup Language (HTML) Home Page http://www.w3.org/MarkUp/

Key words for use in RFCs to Indicate Requirement Levels:
http://www.ietf.org/rfc/rfc2119.txt

Markup language: http://en.wikipedia.org/wiki/Markup_language

Microsoft Office Open XML Formats Overview:
http://www.microsoft.com/office/preview/developers/fileoverview.mspx

Namespaces in XML: http://www.w3.org/TR/REC-xml-names/

Norsk aviskorpus: http://avis.uib.no/

On SGML and HTML: http://www.w3.org/TR/html4/intro/sgmltut.html

Oslo-Bergen-taggeren (for bokmål og nynorsk):
http://decentius.hit.uib.no:8005/cl/cgp/test.html

Oslo-korpuset av taggede norske tekster (bokmålsdelen):
http://www.tekstlab.uio.no/norsk/bokmaal/

Ratio: http://en.wikipedia.org/wiki/Ratio

Ratio. Fraction. What's the difference?
http://www.sci.tamucc.edu/txcetp/cr/math/rf/RatioFraction.pdf

Recommendation Track Process Maturity Levels: http://www.w3.org/2004/02/Process-20040205/tr.html#maturity-levels

Reference: ISO8879: http://www.w3.org/TR/html4/references.html

Speech Synthesis Markup Language (SSML) Version 1.0:
http://www.w3.org/TR/speech-synthesis/

SSML 1.0 say-as attribute values: http://www.w3.org/TR/2005/NOTE-ssml-sayas-20050526/

Standard Generalized Markup Language: http://en.wikipedia.org/wiki/SGML

The LOB Corpus: http://khnt.hit.uib.no/icame/manuals/lobman/LOB1.HTM

The World Wide Web: A very short personal history:
http://www.w3.org/People/Berners-Lee/ShortHistory

Time 100: Tim Berners-Lee:
http://www.time.com/time/time100/scientist/profile/bernerslee.html

Uniform Resource Identifiers (URI): Generic Syntax:
http://www.ietf.org/rfc/rfc2396.txt

W3C Workshop on Internationalizing the Speech Synthesis Markup Language:
http://www.w3.org/2005/08/SSML/ssml-workshop-agenda.html

Weka Machine Learning Project: http://www.cs.waikato.ac.nz/~ml/

World Wide Web Consortium: http://www.w3.org/

www-voice@w3.org Mail Archives: http://lists.w3.org/Archives/Public/www-voice/