Normalization of Non-Standard Words: WS '99 Final Report

Richard Sproat Alan Black Stanley Chen
Shankar Kumar Mari Ostendorf Christopher Richards

September 13, 1999

Abstract

All areas of language and speech technology must deal, in one way or another, with real text. Real text is messy: many things one finds in text — numbers, abbreviations, dates, currency amounts, acronyms ... — are not standard words in that one cannot find their properties by looking them up in a dictionary or deriving them morphologically from words that are in a dictionary, nor can one find their pronunciation by an application of "letter-to-sound" rules. For many applications, such non-standard words — NSW's — need to be normalized, or in other words converted into standard words. Since the correct normalization of a given token often depends upon both the local context and the type (genre) of text one is dealing with, "text-normalization" is in general a very hard problem. Typical technology for text-normalization mostly involves sets of ad hoc rules tuned to handle one or two genres of text (often newspaper-style text), with the expected result that the techniques, do not usually generalize well to new domains.

The purpose of this project was to take some initial steps towards addressing deficiencies in previous approaches to text normalization. We developed a taxonomy of NSW's on the basis of four rather distinct text types — news text, a recipes newsgroup, a hardware-product-specific newsgroup, and real-estate classified ads. We then investigated the application of several general techniques including n-gram language models, decision trees and weighted finite-state transducers to the range of NSW types, and demonstrated that a systematic treatment will lead to better results than can be obtained by the more ad hoc treatments that have more typically been used in the past. For

abbreviation expansion in particular we investigated both supervised and unsupervised approaches, and we report results for both of these. Note that the unsupervised approach allows one to find and posit expansions for abbreviations for a domain given only a raw corpus from that domain, so that in real estate classified ads, for instance, the method will discover that the abbreviation *BR* means *bedroom* or *FP* means *fireplace*.

The overall results that we will report will be terms of word-error rate, which is standard in speech recognition evaluations, but which has only occasionally been used as an overall measure in evaluating text normalization systems.

Contents

1 Motivation

All areas of language and speech technology — be they machine translation, automatic speech recognition, or topic detection — must deal, in one way or another, with real text. In some cases the dependency is direct: machine translation systems and topic detection systems usually work on textual input. In other cases the dependency is indirect: automatic speech recognizers usually depend on language models that are trained on text. But in any case, systems must deal with real text, and real text is messy: many things one finds in text — numbers, abbreviations, dates, currency amounts, acronyms ... — are not standard words in that one cannot find their properties by the trivial procedure of looking them up in a dictionary, or their pronunciation by an application of "letter-to-sound" rules. Such *non-standard words* (NSW's) must typically be *normalized*, or in other words converted into standard words. Since the correct normalization of a given token often depends upon both the local context and the type (genre) of text one is dealing with, "text-normalization" is in general a very hard problem.

Unfortunately text-normalization is rarely considered to be interesting per se, let alone worthy of serious study. As a result typical technology mostly involves sets of ad hoc rules tuned to handle one or two genres of text (often newspaper-style text), with the expected result that the techniques, such as they are, do not usually generalize well to new domains. Note that this statement correctly describes the state of affairs even in text-to-speech synthesis, where one would have expected that the matter would have received more serious attention.

In addition to the ad hocity of most approaches, an additional problem with the lack of systematic work on the topic of non-standard words, is that we do not have a clear idea of the range of types of NSW's that must be covered. Anyone who has worked on text-normalization for a particular domain will be very familiar with the range of possible NSW's for that domain, and will have developed some ways of dealing with them, but there has been little or no work on developing a broader picture of the range of cases one could expect to find if one looks over a wider set of text types.

The purpose of this project was to take an initial stab at addressing these deficiencies in previous approaches. We developed a taxonomy of NSW's on the basis of four rather distinct text types — news text, a recipes newsgroup, a hardware-product-specific newsgroup, and real-estate classified ads. We then investigated the application of several general techniques including n-gram language models, maximum entropy models, decision trees and

weighted finite-state transducers to the entire range of NSW types (rather than a selected subset), and demonstrated that a systematic treatment of such cases will lead to better results than can be obtained by the spottier ad hoc treatments that have more typically been used in the past. We also employed a more standard and systematic procedure for evaluating performance than has heretofore generally been used in the text normalization literature.

This is not an end, but a beginning: it is hoped that the preliminary results we have obtained will inspire others to take this field of inquiry seriously, and to contribute to its future improvement.

2 Introduction: The Problem

Real text contains a variety of token types that are non-standard in the sense that they are either: not ordinary words (dictionary words, or their derivatives) or names; or *are* ordinary words but are typographically abnormal in some way. Typographic abnormality includes distinctive use of capitalization to indicate emphasis, or *non*-distinctive use of capitalization when the entire region of text is capitalized.

Such Non-Standard Words — NSW's — include: digit sequences; words, acronyms and letter sequences in all capitals; mixed case words (*WinNT*, *SunOS*) to be read in various ways; abbreviations; roman numerals; URL's and e-mail addresses. Many of these kinds of elements are pronounced according to principles that are quite different from the pronunciation of ordinary words or names; furthermore there is a high degree of ambiguity in pronunciation (higher than for ordinary words) so that many items have more than one plausible pronunciation, and the correct one must be disambiguated from context: *IV* could be *four*, *fourth*, *the fourth*, or *I.V.*; *IRA* could be *I.R.A.* or *Ira*; *1750* could be *seventeen fifty* as a date or building number, or *seventeen hundred (and) fifty* (or *one thousand seven hundred (and) fifty*) as a cardinal number.

Such items are problematic for language and speech technology systems. For Text-to-Speech systems (TTS) there is the problem of rendering them appropriately in speech. This is particularly important for the goal of *universal access*, since web-based and email documents are more likely to have NSW's than newspaper text, for example.

For Automatic Speech Recognition (ASR) they cause problems for training language models on text, since the language models (LM) must model what people will say, not merely what tokens exist in the text. Hence, it

is typical in preparing text for LM training to apply a variety of ad hoc scripts — e.g. the Linguistic Data Consortium's "Text Conditioning Tools" described below — to normalize NSW's in the text. As techniques for using out-of-domain language model training data improve (Iyer and Ostendorf, 1997), more sophisticated text normalization will be an important tool for utilizing the vast amounts of on-line text resources.

In addition to normalizing text for language modeling in general it is worth noting that many important parts of the text — dates, organization names, money values, etc. — are exactly those which will typically be represented by non-standard words. Thus language models from normalized text are likely to be of specific benefit in information extraction applications, which involve identifying such expressions.

3 Previous Approaches

In principle any system that deals with unrestricted text needs to be able to deal with non-standard words. In practice most work on this topic has been done in the context of two particular applications, namely text-to-speech systems, and text "conditioners" for automatic speech recognition applications. In the first application, obviously, the problem is to decide how an automatic system should say the token; in the second the problem is to predict how a person reading the text would say the token. We briefly consider the techniques applied in these domains below.

Cross-cutting both of these domains (though to date only really applied in TTS) are the application of *sense disambiguation* techniques to the problem of *homograph resolution* for NSW's. This is discussed here (Section 3.3) as the only instance of a fairly principled corpus-based technique that has been applied in this domain.

Problems with these previous approaches are outlined in Section 3.4.

3.1 Text-to-Speech Synthesis Systems

The great bulk of work on "text normalization" in most TTS systems is accomplished using hand-constructed rules that are tuned to particular domains of application (Allen, Hunnicutt, and Klatt, 1987; Black, Taylor, and Caley, 1998; Sproat, 1997).

For example, in various envisioned applications of the AT&T Bell Labs TTS system, it was deemed important to be able to detect and pronounce (U.S. and Canadian) telephone numbers correctly. Hence a telephone number detector (which looks for seven or ten digits with optional parenthe-

ses and dashes in appropriate positions) was included as part of the text-preprocessing portion of the system. On the other hand, although e-mail handles were commonplace even in the mid-80's, when this system was designed, nobody thought of including a method to detect and appropriately verbalize them. This kind of spotty coverage is the norm for TTS systems.

Expansion of non-standard words is accomplished by some combination of rules (e.g. for expanding numbers, dates, letter sequences, or currency expressions) and lookup tables (e.g. for abbreviations, or roman numerals). Ambiguous expansions — e.g. *St.* as *Saint* or *Street* — are usually handled by rules that consider features of the context. In this particular case, if the following word begins with a capital letter, then it is quite likely that the correct reading is *Saint* (*Saint John*), whereas if the previous word begins with a capital letter, the correct reading is quite likely *Street*. Simple rules of this kind are quite effective at capturing most of the cases that you will find in "clean" text (i.e., text that, for instance, obeys the standard capitalization conventions of English prose); but only of course for the cases that the designer of the system has thought to include.

3.2 Text-Conditioning Tools

In the ASR community, a widely used package of tools for text normalization are the Linguistic Data Consortium's (LDC) "Text Conditioning Tools" (Linguistic Data Consortium, 1998). Like most TTS systems, these text-conditioning tools depend upon a combination of lookup tables (e.g., for common abbreviations); and rewrite rules (e.g. for numbers).

Disambiguation is handled by context-dependent rules. For instance there is a list of lexical items (*Act, Advantage, amendment . . . Wespac, Westar, Wrestlemania*) after which Roman numerals are to be read as cardinals rather than ordinals.

Numbers are handled by rules that determine first of all if the number falls into a select set of special classes — U.S. zip codes, phone numbers, etc., which are usually read as strings of digits; and then expands the numbers into number names (*1,956* becomes *one thousand nine hundred fifty six*) or other appropriate ways of reading the number (*1956* becomes *nineteen fifty six*).

The main problem with the LDC tools, as with the text normalization methods used in TTS systems, is that it is quite domain specific: it is specialized to work on Broadcast News-type text material, and does not reliably work outside this domain. For instance, only about 3% of the abbreviations found in our classified ad corpus are found in the LDC tools abbreviation

list.

3.3 Sense-Disambiguation Techniques

Sense disambiguation techniques developed to handle ambiguous words like *crane* (a bird, versus a piece of construction equipment), can be applied to the general problem of homograph disambiguation in TTS systems (e.g. *bass* 'type of fish', rhyming with *lass*; versus *bass* 'musical range', rhyming with *base*).

Many NSW's are homographs, some particular cases, and some more systematic. A particular case is *IV*, which may be variously *four* (*Article IV*), *the fourth* (*Henry IV*), *fourth* (*Henry the IV*), or *I. V.* (*IV drip*). More systematic cases include dates in *month/day* or *month/year* format (e.g. 1/2, for *January the second*), which are systematically ambiguous with fractions (*one half*); and three or four digit numbers which are systematically ambiguous between dates and ordinary number names (*in 1901*, *1901 tons*).

Yarowsky (1996) demonstrated good performance on disambiguating such cases using *decision-list* based techniques, which had previously been developed for more general sense-disambiguation problems.

Once again though, such techniques do presume that you know beforehand the individual cases that must be handled.

3.4 Problems with Previous Approaches

All of the previous approaches to the problem of handling non-standard words presume that one has a prior notion of which particular cases must be handled. Unfortunately this is often impractical, especially when one is moving to a new text domain. Even within well-studied domains — such as newswire text — one often finds novel examples of NSW's. For instance the following abbreviations for the term *landfill* occurred in a 1989 *Associate Press* newswire story:

> Machis Bros **Lf** (S Marble Top Rd) , Kensington, Ga .
> Bennington Municipal Sanitary **Lfl**, Bennington, Vt .
> Hidden Valley **Lndfl** (Thun Field), Pierce County, Wash .

None of these examples can even remotely be considered to be "standard", and it is therefore unreasonable to expect that the designere of a text normalization system would have thought to add them to the list of known abbreviations.

When one moves to certain new domains, such as real estate classified ads, the one often finds a quite rich set of novel examples Consider the example below taken from the *New York Times* real estate ads for January 12, 1999:

> 2400' REALLY! HI CEILS, 18' KIT,
> MBR/Riv vu, mds, clsts galore! $915K.

Here we find *CEILS* (*ceilings*), *KIT* (*kitchen*), *MBR* (*master bedroom*), *Riv vu* (*river view*), *mds* (*maids (room)* (?)) and *clsts* (*closets*), none of which are standard abbreviations, at least not in general written English.

A more general problem is that we do not in fact have a clear idea of what types of NSW's exist, and therefore *need* to be covered: there is no generally known taxonomy of non-standard words for English, or any other language, though there have been many taxonomies of particular subclasses (Cannon, 1989; Römer, 1994).

4 Overview of the Team's Contribution

To our knowledge, the work described here is the first *systematic* study of NSW's and algorithms for handling them.

Our specific contributions are:

- A proposal for a *taxonomy* of NSW's based on the examination of a diverse set of corpora and the NSW's contained therein.
- Publicly available hand-tagged corpora from several specific domains: North American News Text Corpora; Real Estate Classified Ads; `rec.food.recipes` newsgroup text; `pc110` newsgroup text.
- An implemented set of methods for dealing with the various classes of NSW's. These include:
  - A splitter for breaking up single tokens that need to be split into multiple tokens: e.g. *2BR,2.5BA* should be split into *2 BR , 2.5 BA*.
  - A classifier for determining the most likely class of a given NSW.
  - Methods for expanding numeric and other classes that can be handled "algorithmically".
  - Supervised and unsupervised corpus/domain-dependent methods for dealing with abbreviations: the supervised methods presume that you have a tagged corpus for the given domain; the unsupervised methods presume that all you have is raw text.

- A publicly available set of tools for text normalization that incorporate the above-mentioned methods.

Each of these issues is described in detail in the sections below.

5 A Taxonomy of NSW's

After examining a variety of data, we developed a taxonomy of non-standard words (NSWs), summarized in Table 1, to cover the different types of non-standard words that we observed. The different categories were chosen to reflect anticipated differences in algorithms for transforming (or expanding) tokens to a sequence of words, where a "token" is a sequence of characters separated by white space (see Section 8.2 for a more on defining tokens).

Four different categories are defined for tokens that included only alphabetic characters: expand to full word or word sequence (EXPN), say as a letter sequence (LSEQ), say as a standard word (ASWD) and misspelling (MSPL). The ASWD category includes both standard words that are simply out of the vocabulary of the dictionary used for NSW detection and acronyms that are said as a word rather than a letter sequence (e.g. NATO). The EXPN category is used for expanding abbreviations such as *fplc* for *fireplace*, but not used for expansions of acronyms/abbreviations to their full name, unless it would be more natural to say the full expansion in that genre. For example, *IBM* is typically labeled as LSEQ (vs. EXPN for *International Business Machines*), while *NY* is labeled as EXPN (*New York*). Similarly, *won't* is not labeled as an expansion, but *gov't* should be. Of these four categories, the problem of expanding the EXPN class of tokens is of most interest in our work, since pronouncing words and detecting misspellings has been handled in other work.

Several categories are defined for tokens involving numbers. We identified four main ways to read numbers: as a cardinal (e.g. quantities), an ordinal (e.g. dates), a string of digits (e.g. phone numbers), or pairs of digits (e.g. years). However, for ease of labeling and because some categories can optionally be spoken in different ways (e.g. a street address can be read as digits or pairs), we defined categories for the most frequent types of numbers encountered. We chose not to have a separate category for roman numerals, but instead to label them according to how they are read, i.e. as a cardinal (NUM, as in World War II) or an ordinal (NORD, as in Louis XIV or Louis the XIV). For the most part, once a category is given, the expansion of numbers into a word sequence can be implemented with a straightforward set of rules. The one complicated case is money, where *$2 billion* is spoken

as *two billion dollars*, so the *dollars* moves beyond the next token. Allowing words to move across token boundaries complicates the architecture and is only necessary for this special case, so we define a special tag to handle these case (BMONY).

Sometimes a token must be split to identify the pronunciation of its subparts, e.g. *WinNT* consists of an abbreviation *Win* for *Windows* and the part *NT* to be pronounced as a letter sequence. To handle such cases, we introduce the SPLT tag at the token level, and then use the other tags to label sub-token components. In some instances, the split tokens include characters that are not to be explicitly spoken. These are mapped to one of two categories – PUNC or SLNT – depending on whether or not the characters are judged to be a non-standard marking of punctuation (i.e. correspond to a prosodic phrase break). Both tags can also be used for isolated character sequences (i.e. not in a split). The PUNC class was not in the original taxonomy, but was introduced at the start of the workshop after experience with labeling suggested it would be reliable and useful.

Three additional categories were included to handle phenomena in electronic mail: funny spellings of words (presumed intentional, as opposed to a misspelling), web and email addresses, and NONE to handle ascii art and formatting characters. The category NONE is assumed to include phenomena that would not be spoken and is mapped to silence for the purpose of generating a word sequence, but it also includes tokens that we simply do not know how to render at this point, such as the quoting character ">" and smiley faces ":)" in email, computer error messages, and stock tables in news reports.

Although not included in the table below, an additional OTHER tag was allowed for rare cases where the labelers could not figure out what the appropriate tag should be. The OTHER category was not used in the word prediction models.

6 Corpora and Tagging Conventions

6.1 Corpora Chosen

In order to ensure generalizability of the tag taxonomy and algorithms developed here, we chose to work with four very different data sources, described below.

NANTC: The North American News Text Corpus (NANTC) is a standard corpus available from the Linguistic Data Consortium (LDC). The

Table 1: Taxonomy of non-standard words used in hand-tagging and in the text normalization models.

	EXPN	abbreviation, contractions	adv, N.Y, mph, gov't
alpha	LSEQ	letter sequence	CIA, D.C, CDs
	ASWD	read as word	CAT, proper names
	MSPL	misspelling	geogaphy
	NUM	number (cardinal)	12, 45, 1/2, 0.6
	NORD	number (ordinal)	May 7, 3rd, Bill Gates III
	NTEL	telephone (or part of)	212 555-4523
	NDIG	number as digits	Room 101,
N	NIDE	identifier	747, 386, I5, pc110, 3A
U	NADDR	number as street address	5000 Pennsylvania, 4523 Forbes
M	NZIP	zip code or PO Box	91020
B	NTIME	a (compound) time	3.20, 11:45
E	NDATE	a (compound) date	2/2/99, 14/03/87 (or US) 03/14/87
R	NYER	year(s)	1998 80s 1900s 2003
S	MONEY	money (US or otherwise)	$3.45 HK$300, Y20,000, $200K
	BMONY	money tr/m/billions	$3.45 billion
	PRCT	percentage	75%, 3.4%
	SPLT	mixed or "split"	WS99, x220, 2-car
			(see also SLNT and PUNC examples)
O	SLNT	not spoken, word boundary	word boundary or emphasis character:
T			M.bath, KENT*REALTY, _really_, ***Added
H	PUNC	not spoken, phrase boundary	non-standard punctuation: "..." in
E			DECIDE...Year, "***" in $99,9K***Whites
R	FNSP	funny spelling	sllooooooww, sh*t
	URL	url, pathname or email	http://apj.co.uk, /usr/local, phj@teleport.com
	NONE	token should be ignored	ascii art, formating junk

corpus includes data from several sources (New York Times, Wall Street Journal, LA Times, and two Reuters services). We used a small random sample from these five sources taken from the 1994-1997 period. This corpus was chosen because it represents clean well-edited text data of the form often used in existing text analysis tools. Such data is already used for training language models in speech recognition and for training existing TTS systems. Although the percentage of NSWs is relatively small, we include this to allow easier comparison of our results with existing text analysis techniques.

Classifieds: A corpus of classified ads was collected for this work by the LDC. It contains real estate ads from three sources (Boston Globe, Washington Post, and the FindItOnline Classified Network). These were collected during the first half of 1999. This corpus was chosen because of the high frequency of NSWs particularly EXPN tokens, which pose particularly difficult problems for text normalization.

pc110: The pc110 corpus was collected from a public mailing list on the IBM pc110 palmtop computer (pc110@ro.nu). It consists of daily digests from the list from 1998-1999. Messages are usually quite technical though still in a chatty email style. It contains many abbreviations, misspellings, unusual capitalization and lots of machine and part identifiers. Unlike general newsgroups, however, it has very few off-topic articles. No significant tidy up of the data has been done except automatic detection of mail headers, so it is very noisy compared to NANTC. It was selected to represent email, but from a forum that allows us to publicly distribute it.

RFR: The RFR corpus includes recipes from the rec.food.recipes electronic newsgroup. Although the submissions are via electronic mail, the data is relatively clean because the list is carefully moderated. This means that there is little of the discussion and quoting that is typical of many email lists (including the pc110 corpus), although there is a large number of URL and email addresses. The data was collected during in the first half of 1999. It was chosen because it represents a non-technical but very specialized style of text that was easy to collect.

In total there are about 5.5 million tokens in the databases, with a break down as shown in Table 2. The number of non-standard words given in the table is based on the automatic detection algorithm described in

Table 2: Size of different corpora and number of detected non-standard word tokens.

Corpus	NANTC	Classifieds	pc110	RFR
total # tokens	4.3m	415k	264k	209k
# NSWs	377k	180k	72k	46k
% NSW	8.8%	43.4	27.3	22.0

Table 3: Distribution of frequent alphabetic tags in the four corpora.

	Domains			
	NANTC	Classifieds	pc110	RFR
ASWD	83.49	28.64	64.60	72.36
LSEQ	9.10	3.00	22.60	2.11
EXPN	7.41	68.36	12.80	25.53

Section 8.2, which does miss some tokens, and does not separately count the sub-components of a SPLT token.

As should be clear from the partial distribution given in Tables 3–4, these corpora are very different in nature. For the main three alphabetic labels, there are a large percentage of ASWD tokens in all domains, which are primarily out-of-vocabulary words. These include names of people and places in news, names of streets and towns in the classified ads, and unusual ingredients in recipes. In addition, the classified ads have a large number of EXPN abbreviation tokens, as expected; the pc110 domain has a large percentage of letter sequences because of the technical jargon (e.g. PCMCIA); and the recipes have a large number of EXPN tokens corresponding to measurement abbreviations. Looking at the distributions of numbers, it is not surprising to see that years are frequent in the news domain, telephone numbers are frequent in classified ads, identifiers are frequent in the pc110 corpus (equipment IDs), and the main use of numbers in recipes is to indicate quantities.

6.2 Initial Processing of Corpora

In each case the raw data was converted to a simple XML based markup format. The NANTC data from the LDC already has some SGML based tags

Table 4: Distribution of frequent number tags in the four corpora.

	Domains			
	NANTC	Classifieds	pc110	RFR
NUM	66.11	58.26	43.77	97.90
NYER	19.06	0.70	0.51	0.27
NORD	9.37	3.37	4.45	0.11
NIDE	2.24	5.83	37.41	0.47
NTEL	1.25	25.92	1.32	0.02
NTIME	1.21	3.28	4.16	1.12
NZIP	0.22	0.29	0.17	0.04
NDATE	0.20	0.13	1.33	0.05
NDIG	0.16	0.00	2.16	0.01
NADDR	0.13	2.20	0.15	0
Total Nums	73005	24193	7818	18195

these were partially augmented (in particular, explicit closing paragraphs were added) in a fully automatic way and extraneous characters that interfere with such mark up were quoted (that is "`&`" and "`<`"). For the other corpora, paragraph boundaries were marked at blank lines and for the pc110 and RFR, which came from electronic mail and usenet data, article headers were marked up and ignored in later processing. Where the article content contains quoted headers (effectively only in pc110) this data remained as part of the training data.

After markup only those tokens appearing within paragraphs were considered for analysis, all other tokens were ignored (primarily mail headers and NANTC story headers and footers).

Once in a standard XML form we automatically extracted tokens that were NSWs by the definition given in Section 8.2. Each NSW was extracted with surrounding context (three tokens on each side) and a guess at the NSW type was given. These tokens alone were presented for hand labeling as discussed in the next section.

Each token identified as an NSW was given the tag `W` with an attribute `NSW` and value as given by the labelers. When the NSW tag is EXPN, an additional attribute `PRON` is given whose value is a string of space separated words representing its expansion.

Additionally where a token was identified as being split, the split tokens

were marked with a `WS` tag within a `W` marking the whole unsplit token. Whitespace is preserved in the marked up files containing the additional NSW tags (and possibly pronunciations) without losing any information from the original text.

An example of marked-up XML text is given in Figure 1.

6.3 Tagging Conventions

The labeling task involves looking at a token within a short context (four words on either side) and identifying one of the possible labels for that non-standard word. Note that labeling involves (primarily) identifying what words would correspond to the token if the text was read and indicating this symbolically via the tags. Labelers are instructed to type in the expansion explicitly only for tokens where the expansion is unclear or ambiguous, such as *sunny* for *sun* (vs. *Sunday*). Expansions that are frequent for a particular corpus, such as BA and LR in the classified ads, are expanded automatically unless explicitly expanded by the labeler because of having a non-standard usage (e.g., not *bathroom* or *living room*, respectively).

To speed up the tagging task, labelers are presented only with candidate non-standard words that have been identified using a simple dictionary check. As a result, some of the non-standard words are not labeled, but the percentage is small. Standard words that are presented as candidate NSWs are simply labeled ASWD. Also to speed up labeling, the intermediate *tag* SCORE was introduced for tokens such as *5-7*, which are later automatically re-labeled as SPLT and split into a *NUM to NUM* sequence. (The "-" is an EXPN that expands to the word *to*.)

The labelers used a tagging tool, actually a special mode in the Emacs editor, that presents each token on a new line surrounded by its context. A guess at the label is given at the start and the labeler must either accept the guess or provide an alternative. In the following example, which illustrates the presentation, the corrected tags are given in the first column and the initial guess is in the second column.

```
NORD  NUM             for Bosnia by Oct * 15 * he would go to 109
NORD  NUM             no later than Nov * 15 * The United States along $
NUM   NUM             begin the sale of * 12 * million barrels of oil  $
ASWD  ASWD   possibility of doing this * multilaterally * 0 0 0 0 358
LSEQ  LSEQ    The Washington Post says * U.S * relations with its alli$
EXPN  ASWD Rosenblatt Stadium in Omaha * Neb * they have never seen
```

The first two NSWs are not simple numbers but ordinals, since they are dates and hence must be labeled NORD. The third line is correct, a simple

```
<DOC>
<TEXT>
<P>
AAA INVESTMENTS SO SHORE,<W NSW="SPLT"><WS NSW="NUM"> 40</WS><WS
NSW="EXPN" PRON="plus">+
</WS></W> modern<W NSW="EXPN" PRON="brick"> brk</W><W NSW="EXPN"
PRON="apartments"> apts</W> on<W NSW="SPLT"><WS NSW="NUM"> 4</WS><WS
NSW="EXPN" PRON="plus">+</WS></W> acres,<W N
SW="EXPN" PRON="individual"> indiv</W><W NSW="EXPN" PRON="heating">
ht.</W> Income<W NSW=
"MONEY"> $400K.</W> Ask<W NSW="MONEY"> $2,975,000</W><W NSW="SPLT"><WS
NSW="EXPN" PRON="w
ith"> w/</WS><WS NSW="MONEY">$750K</WS></W> down. ROBERT<W NSW="LSEQ">
L.</W> TENNEY REAL
TY<W NSW="PUNC"> (</W><W NSW="NTEL">617</W><W NSW="PUNC">)</W><W
NSW="NTEL"> 472-0629472-
0630</W>
</P>
</TEXT>
</DOC>
<DOC>
<TEXT>
<P>
AAA INVESTMENTS<W NSW="EXPN" PRON="north west"> N.W.</W> OF BOSTON,<W
NSW="NUM"> 17</W><W
 NSW="EXPN" PRON="modern"> mod</W> brick<W NSW="EXPN"
PRON="apartments"> apts,</W> new<W
NSW="EXPN" PRON="roof"> rf</W><W NSW="EXPN" PRON="and">&amp;</W>
boiler,<W NSW="EXPN" PRO
N="included"> inc</W><W NSW="MONEY"> $126K,</W> ask<W NSW="MONEY">
$815K</W><W NSW="SPLT"
><WS NSW="EXPN" PRON="with"> w/</WS><WS NSW="MONEY">$200K</WS></W>
down
</P>
</TEXT>
</DOC>
<DOC>
<TEXT>
<P>
AAA INVESTMENTS<W NSW="EXPN" PRON="north west"> N.W.</W> OF BOSTON,<W
NSW="NUM"> 17</W><W
 NSW="EXPN" PRON="modern"> mod</W> brick<W NSW="EXPN"
PRON="apartments"> apts,</W> new<W
NSW="EXPN" PRON="roof"> rf</W><W NSW="EXPN" PRON="and">&amp;</W>
boiler,<W NSW="EXPN" PRO
N="included"> inc</W><W NSW="MONEY"> $126K,</W> ask<W NSW="MONEY">
$825K</W><W NSW="SPLT"
><WS NSW="EXPN" PRON="with"> w/</WS><WS NSW="MONEY">$200K</WS></W>
down
</P>
</DOC>
```

Figure 1: A sample of XML-markup of NSW's.

number. The fourth *multilaterally* is a standard word that appears because it is not in our lexicon; however, it is correctly guessed as a word (ASWD). The next line is correctly labeled as a letter sequence. The last line *Neb* is an abbreviation for *Nebraska* and hence should be marked as EXPN.

6.4 Inter-Labeler Reliability Measures

A portion of the data was marked by multiple labelers to assess transcriber reliability. The level of agreement between the different labelers for the categories described above was measured using the kappa statistic, which is the ratio

$$\kappa = \frac{P_o - P_c}{1 - P_c},$$

where P_o is the percent agreement measured between labelers and P_c is the agreement that would be predicted by chance. Assuming that all N coders label all D data points with one of C classes, the specific formulas for computing these quantities for the multi-class, multi-labeler case are

$$P_c = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1,\neq i}^{N} P_c^{ij},$$

where

$$P_c^{ij} = \sum_{k=1}^{C} p_i(k) p_j(k)$$

is the chance agreement for coders i and j ($p_i(k)$ is chance of labeler i assigning class k), and

$$P_o = \frac{\sum_{l=1}^{D} \sum_{k=1}^{C} n_{kl}(n_{kl} - 1)}{DN(N-1)}$$

where n_{kl} is the number of coders that labeled datum l with class k. The kappa score is computed using publicly available software developed by Flammia (Flammia, 1998). The kappa statistic is widely used for evaluating labeler consistency (Hirschberg and Nakatani, 1996; Carletta et al., 1997; Jurafsky et al., 1997; Flammia, 1998), and it is generally agreed that a kappa score of greater than 0.7 indicates good agreement.

We measured inter-labeler agreement on two subsets of data for a set of 25 NSW tags, including the 23 tags in Table 1, the OTHER tag and the intermediate SCORE tag. In both cases, the NSW type SPLT was regarded as a separate category by itself, so if a token A was split by the labeler as

$[A \rightarrow\ a_0\ a_1\ a_2]$, the categories for a_0, a_1 and a_2 were not taken into account. For $D = 2268$ NSW tokens from the news data and three $N = 3$ labelers, the agreement was $\kappa = 0.81$. For $D = 622$ NSW tokens from the classified ads and $N = 9$ labelers, the agreement was $\kappa = 0.84$. Both results indicate good reliability.

In looking at the data to understand the disagreements that are there, we find that the main problems are labeling errors and lack of specific examples in the labeling guide, rather than real ambiguities. Labeling errors include unnoticed misspellings (labeled ASWD), simple typing errors, and labeler misspellings in expansions. The main problem with unclear guidelines was for the use of the split command, and labeling unspoken tokens with NONE vs. SLNT. For future use, the labeling guidelines have been expanded to address these problems. For the data used in the workshop, the amount of noise that these errors introduce is relatively small. However, it is significant, given that the accuracy level of our algorithms is relatively high (at least in the supervised learning cases). For that reason, there was some effort made to detect and correct errors, particularly simple errors where correction could be automated (e.g. missing splits).

7 Theoretical Models and Overall Architecture

7.1 Theoretical Preliminaries

We pose the problem of predicting the expanded form of non-standard words as one of finding the most likely word sequence $\mathbf{w} = w_1, w_2, \ldots w_n$ given the observed token sequence $\mathbf{o} = o_1, o_2, \ldots o_m$. The observed token sequence corresponds to the input text which is marked as to whether each token is an NSW, and the output word sequence corresponds to the desired words to be spoken. Mathematically, the objective is:

$$\hat{\mathbf{w}} = \operatorname*{argmax}_{\mathbf{w}} p(\mathbf{w}|\mathbf{o}) \quad (1)$$

$$= \operatorname*{argmax}_{\mathbf{w}} [\sum_{\mathbf{t}} p(\mathbf{w}, \mathbf{t}|\mathbf{o})] \quad (2)$$

$$\approx \operatorname*{argmax}_{\mathbf{w},\mathbf{t}} p(\mathbf{w}, \mathbf{t}|\mathbf{o}) \quad (3)$$

where $\mathbf{t}$ is the sequence of token tags, which may be at the output word level (n-length) or the input token level (m-length). In equation 3, we make the assumption that for a particular output word and a particular observed token, there is usually only one tag that would be appropriate. Thus, most

of the probability mass in the joint word-tag conditional distribution is associated with a single tag. This assumption is useful for simplifying the recognition decoding problem, but it is also quite reasonable for most of the cases we are interested in.

The basic problem posed in equation 3 can in principle be solved using two different approaches, which we will refer to as the source-channel model (or noisy channel model) and the direct model, as described respectively in the two sections to follow.

7.1.1 Source-Channel Model

The source-channel model is analogous to the approach used widely in speech recognition. That is, we view the desired word sequence that $\mathbf{w}$ as being generated by some source with probability $p(\mathbf{w})$, and transmitted through a noisy channel which randomly transforms the intended words to the observed character sequence $\mathbf{w}$ according to the channel model $p(\mathbf{o}|\mathbf{w})$. Mathematically, this corresponds to:

$$\hat{\mathbf{w}} = \operatorname*{argmax}_{\mathbf{w}} p(\mathbf{w}|\mathbf{o}) = \operatorname*{argmax}_{\mathbf{w}} p(\mathbf{o}|\mathbf{w})p(\mathbf{w}).$$

Adding the NSW tags to this picture, the problem becomes

$$\begin{aligned} \hat{\mathbf{w}} &\approx \operatorname*{argmax}_{\mathbf{w},\mathbf{t}} p(\mathbf{o},\mathbf{t}|\mathbf{w})p(\mathbf{w}) && (4) \\ &= \operatorname*{argmax}_{\mathbf{w},\mathbf{t}} p(\mathbf{o}|\mathbf{t},\mathbf{w})p(\mathbf{t}|\mathbf{w})p(\mathbf{w}) && (5) \end{aligned}$$

where we have decomposed the overall probability distribution into three components: a language model $p(\mathbf{w})$ (as in speech recognition), a tag model $p(\mathbf{t}|\mathbf{w})$, and a tag-dependent observation model $p(\mathbf{o}|\mathbf{t},\mathbf{w})$. Assumptions behind the implementation of the different models are described below.

Language Model. The language model $p(\mathbf{w})$ serves the same function as in speech recognition, so it is natural to borrow speech recognition techniques for modeling and estimation here: in our particular implementation we use trigram language models with modified Kneser-Ney backoff (Kneser and Ney, 1995; Chen and Goodman, 1998).

Tag Model. In the source-channel framework, we represent one tag per word, and assume that tags depend only on the current word:

$$p(\mathbf{t}|\mathbf{w}) = \prod_{i=1}^{n} p(t_i|w_i). \quad (6)$$

In order to train this model, we need a tag sequence that is expanded to match the word sequence one-to-one, i.e. using sub-token labels in split tokens. For cases where single tokens in the observation space map to multiple words in the output space, we can either repeat tags or represent the multiple word sequence as a single "word". Given the two parallel tag and word sequences, the tag model can be trained using standard n-gram back-off techniques. However, one might want to introduce an intermediate stage of back-off depending on the word class, i.e.,

$$p(t_i|w_i) \rightarrow p(t_i|c(w_i)) \rightarrow p(t_i)$$

where the word class might be defined in terms of part-of-speech (or semantic class) label or word frequency in the target domain. For cases where the observation o_i contains only alphabetic characters but is labeled as an NSW because it is out of vocabulary, it may be useful to compute features of the word to predict the tag probability, as in (Bikel et al., 1997).

Observation Model. For purposes of simplifying the discussion, assume that the hypothesized word sequence can be reliably "parsed" so that there is a one to one mapping between observation tokens o_i and words w_i. Next, we assume that the observed realization of a word will be conditionally independent from word to word, given the tag and possibly statistics about the domain. Thus the observation model becomes:

$$p(\mathbf{o}|\mathbf{t}, \mathbf{w}) = \prod_{i=1}^{n} p(o_i|t_i, w_i),$$

and the key problem is to find $p(o_i|t_i, w_i)$.

7.1.2 Direct Model

An alternative to the source-channel model is to represent the posterior probability of words given observations directly; hence, this approach is often referred to as the direct model. The overall objective, including the tags, is:

$$\hat{\mathbf{w}} \approx \underset{\mathbf{w},\mathbf{t}}{\operatorname{argmax}}\, p(\mathbf{w}, \mathbf{t}|\mathbf{o}) \quad (7)$$

$$= \underset{\mathbf{w},\mathbf{t}}{\operatorname{argmax}}\, p(\mathbf{w}|\mathbf{o}, \mathbf{t})p(\mathbf{t}|\mathbf{o}). \quad (8)$$

In the direct model, there are two main component models: the tag sequence model and the tag-dependent word sequence model, both of which

are conditionally dependent on the observed token sequence. Using the direct modeling framework has the advantage that it is practical to use the full observation sequence for prediction, as opposed to only the local observation in the source-channel model. However, the disadvantage is the potentially large parameter space, because of the large number of factors that get incorporated into the models, particularly the word-sequence model. To simplify this problem, we will use decision tree and maximum entropy techniques, as well as the standard Markov assumptions on the word sequence.

Tag prediction model. In the tag sequence model, we will start by assuming that tags are conditionally independent of all but the most recent tag given the observation sequences, and then use decision trees to simplify the prediction space:

$$p(\mathbf{t}|\mathbf{o}) = \prod_{i=1}^{m} p(t_i|t_{i-1}, \mathbf{o}) \quad (9)$$

$$= \prod_{i=1}^{m} p(t_i|T[t_{i-1}, \mathbf{o}]) \quad (10)$$

(Note that, in this case, it is simplest to use an m-length tag sequence, i.e. that has a one-to-one mapping with the observed token sequence.) In fact, it may be sufficient to simply condition on the observation space and not the previous tag, which simplifies the search and makes it possibly to separately explore tag predictors designed for subclasses of NSW's such as the numbers (NDIG, NUM, NORD, NIDE, etc.). The types of questions that may be useful in the decision tree include:

- is the word mixed case?
- does the word have a number in it?
- does the word have non-alphanumeric characters in it?
- does the word have a "$" in it?
- is the word in an abbreviation list?
- ....

Unlike in the source-channel case, it is straightforward to evaluate this tag sequence model apart from other component of the system, since we have tag-labeled test data.

ME word sequence model. The second model component is the word sequence model:

$$p(\mathbf{w}|\mathbf{o},\mathbf{t}) = \prod_{i=1}^{n} p(w_i|w_{i-1},t_i,\mathbf{o}). \tag{11}$$

The probability $p(w_i|w_{i-1},t_i,\mathbf{o})$ is impractical to estimate using standard maximum likelihood techniques, because of the large size of the conditioning space. However, it is well suited to using maximum entropy techniques as a way of combining separately estimated statistics for $p(w_i|w_{i-1})$, $p(w_i|t_i)$ and $p(w_i|o_i)$, as well as trigger models and or predictors based on long distance statistics as in $p(w_i|s(\mathbf{o}))$.

7.2 Architectural Overview

The implemented system implements a subset of the ideas described in the theoretical discussion in the previous section. The architecture of the implemented system is diagrammed in Figure 2. Each of the modules will be described in detail in Section 8, but a few things should be noted here, given the preceding discussion. The classifier, which decides upon the most likely tag or tags for a given NSW token, is based upon a (CART-style) decision tree, which is an instance of what we have termed the direct model in the previous discussion. The language model in the currently implemented architecture, an n-gram language model, is consistent with the source-channel approach. The tag expansion models, corresponding to what we previously termed the *observation model*, range from algorithmic — e.g. in the case of an NDIG tag, where one merely wants to produce a string of digit names; to more probabilistic, as in the case of EXPN's where one is especially interested in estimating $p(o_i|t_i,w_i)$, as previously described.

8 Description of Individual Modules

The following subsections describe in detail each of the modules introduced in Section 7 and diagrammed in Figure 2.

8.1 Lattice Format

The format for the lattices is simple and is described briefly in Appendix C.

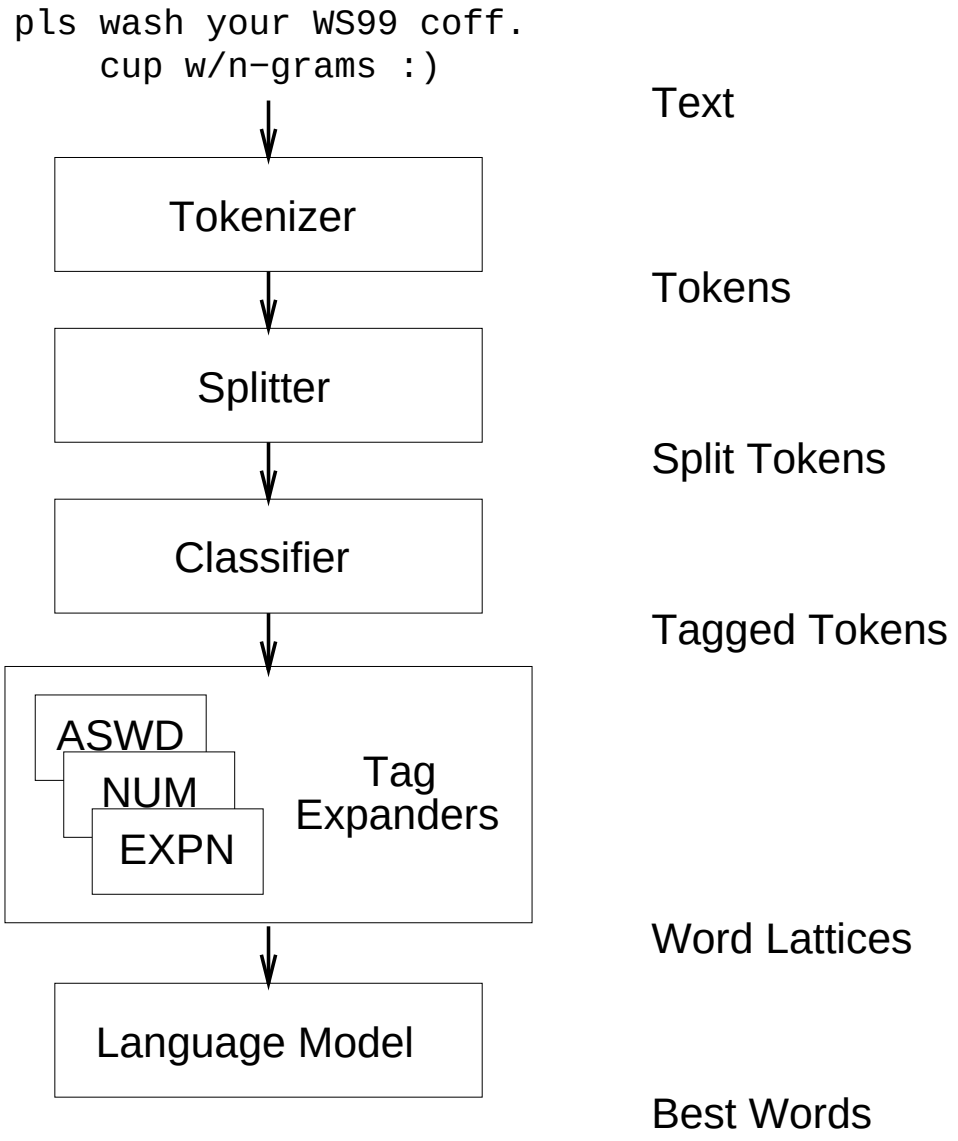

Figure 2: Overall architecture of the implemented system. Input text is passed through a tokenizer, which produces a stream of (basically) whitespace-separated tokens. Then splittable tokens are split by the splitter into further tokens. The resulting token stream is then passed through the classifier, which outputs a sequence of tokens tagged with their NSW class. The tag expanders produce a lattice of possible outputs for each class of NSW. The language model is then used to disambiguate the output.

NANTC	classifieds	pc110	RFR
4.36	16.77	14.80	7.00

Table 5: Percent of split NSW tokens in the August 9 training set.

8.2 Tokenizer

The text is broken up into whitespace-separated tokens, and leading and trailing standard punctuation is removed and saved as features of the token.

8.3 Splitter

After the tokenizer resolves the original input text into whitespace separated tokens, these are passed on to the splitter where they are further broken down into subtokens. In the following sections we discuss the motivation for splitting tokens; the algorithms and approaches we developed for doing so; and finally, we present an evaluation of the splitter's performance. The reader should note that in the following sections, references to *tokens* and *all tokens* do not include those tokens labeled as NONE.

8.3.1 Motivation

The problem of expanding single non-standard words is exacerbated by the various phenomena which result in the effective deletion of whitespace between certain tokens. This deletion process has several avenues of expression: some are unintentional, as is the case for typo- and scanographical errors; and others, intentional, as is often true in the classifieds domain. In the latter case, in printed media, one often saves money by omitting spaces where possible.

Furthermore, "truth" as expressed by the hand-labeled data indicates that not an insignificant portion of the "non-standard" tokens in each corpus should be split. As shown by Table 8.3.1, The precise number varies by corpus, from less than 4.4% of NSW tokens in NANTC, to approximately 16.8% of NSW tokens in the classifieds.[1] It is worth noting that in the classifieds domain, where NSWs account for 43.4% of all tokens, 7.29% of all tokens should be split.

Examples In most text, there will be several forces at work which result in aggregate tokens, hence necessitating a splitter. Some examples from

[1] These numbers are relative to the August 9 training set.

the various corpora will help to illustrate the problem at hand. Take, for instance, the following table. Each example demonstrates a different class of

	NANTC	classifieds	pc110	RFR
Token	`RVing`	`4BR`	`xjack`	`11/2`
Realization	`RV ing`	`four bedrooms`	`X jack`	`one and a half`

Table 6: Examples of tokens to split.

tokens which require splitting. From left to right, we have a letter sequence with a morpheme affix; a concatenation of a number and an abbreviation; a brand name subject to English phonotactics; and an ambiguous numeric token, which must be split to remove ambiguity.

8.3.2 Method

The essential observation in both of these approaches is that splitting tokens is a two-part process. First, one must identify which tokens or subtokens (i.e., internal character sequences) must be grouped in order that they *not* be split. This is important, because many of the same cues which indicate a likely split point (e.g. commas, hyphens, slashes) also serve as joining points in certain common entities (e.g. numbers, telephone numbers, and dates, respectively).

While there are a plethora of special cases to cover while grouping, the second phase of splitting is relatively straightforward. While the former included, among others, rules to capture dates, letter sequences, telephone numbers, monetary expressions, and so forth, a mere four to six sufficed to hypothesize split points (other than the implicit ones resulting from the grouping process). In particular, the most productive split points were the following: at transitions from lower to upper case; after the penultimate upper-case character in transitions from upper to lower case; at transitions from digits to alphabetic characters; and at punctuation.

During the course of the workshop, we developed two rule-based approaches to hypothesizing the subparts of aggregate tokens. The first of these was implemented in the Perl text processing language; this version of the splitter is the one from which our reported results are derived. The second method makes use of weighted finite-state transducers.

Perl The component responsible for designating strings to group and ignore was written as a boolean-valued function of a single argument, where a

return value of "true" indicated that the argument, a string, was not to be subject to the splitting step. The splitting step took the following recursive form. For each splitting rule in an ordered list, ignore each token in the given list of tokens if possible. Otherwise, split them with the current rule, return the split pieces as input to the same function, and advance to the next splitting rule in the list. Finally, return the resulting list of subtokens when there are no more splitting rules to apply.

Weighted Finite-State Transducers The WFST splitter, constructed with the AT&T *lextools* and *fsm tools*, provides an alternate approach to the recursive, sequential method used by its Perl counterpart. Rather than consider the string and substrings of the given token sequentially and atomically, the WFST splitter considers all strings as members of the transitive closure of groupable and splittable character sequences. Specifically, each possible grouping and splitting of the token is considered in parallel, and that configuration with the least resulting weight is chosen as the result. Due to this property, weights are chosen to minimize the number of splits made, and hence it tends to preserve the structure of the groupable entities.

The WFST splitter has a couple of advantages over its previously described sibling. For one, it has the (yet-unused) ability to produce output in the form of a lattice, to cover cases where several reasonable splits are possible. Moreover, the WFST paradigm lends itself to models which are more modular and easier to manipulate in arbitrary ways. On the other hand, it is significantly slower, and and the increased wall-clock time is not offset by a comparable increase in performance.

8.3.3 Evaluation

We evaluated the splitter in two stages. The first, "coarse-grained" stage yields a measurement of how well the splitter decides *whether* the given token should be split; these measurements are reported in terms of precision and recall.

The second, "fine-grained" stage yields a measurement of how *well* the splitter splits a given token, where a split is either correct (all generated subtokens exactly match those in truth) or incorrect. These measurements are reported in terms of "split correct" and "total correct"; the former considers only those tokens which are known to be split, while the latter considers all tokens regardless.

The results of both of these stages are found in Table 8.3.3. Recall that the performance of the WFST splitter was nearly the same, and is hence

	NANTC	classifieds	pc110	RFR
Recall	98.89	94.96	87.66	98.88
Precision	74.41	87.32	81.68	89.51
Split Correct	92.54	85.99	74.11	89.54
Total Correct	98.45	95.19	92.97	98.40

Table 7: Results for the Perl-based m4 splitter.

omitted.

In the above table the relatively poor precision in each domains is quite striking. It is, however, in many cases an unfair measure of the system's quality. While the results indicate that the splitter does over-generate splits, there are a number of mitigating factors. The most prevalent of these is that even though labeler agreement is high with respect to whether a given token should be split, it is low with respect to where to split it. This is compounded, moreover, by a high degree of labeler error; for instance, it is very often the case that a hyphen used in a delimiting context, as in *12:00-3:00* and *$975K-$1,595,000* , is not used to split the token as it should. Further over-generative errors are suspect because it is unclear whether the split form would be pronounced differently from the "correct" form, as is the case with many tokens whose internal punctuation was the impetus for the split.

8.4 A Classifier for Non-Standard Words

8.4.1 Introduction

The purpose of the NSW classifier is to categorize a token from output of the splitter into any one of predefined NSW types Eg. NUM, MONEY, ASWD, EXPN, LSEQ, NYER etc. This predicted category is then used as an input for the tag expander that determines the expansion of the token to words.

8.4.2 The Overall NSW Classifier for All Tokens

The role of the NSW classifier is to assign a tag to each NSW token. This overall classifier is implemented by a CART model. The CART model makes use of a number of features for the classification.

The features used in the CART tree fall into two classes:

1. Simple domain independent features:
   These look at properties of the individual token, and require no do-

main specific information to calculate. Examples of such features are: token length, contains numbers (and type), contains vowels, all capitals. Such features are used over a window of current and previous and next 2 tokens.

2. Domain dependent features:
   As the type of alphabetic tokens changes over the different domains we also calculate a set of features that are based on domain specific information. These features consitute a subclassifier for alphabetic tokens. Features from this subclassifier are used within the overall classification tree. This sub-classifier is discussed in detail in the following sections.

8.4.3 Sub-classifier for Alphabetic tokens

Alphabetic tokens are those that consist of strings of alphabetic characters with optional periods between characters. They also consist of those tokens with both alphabetic characters and characters such as apostrophes (' or ') and slashes(/). These alphabetic tokens can be classified into three fundamental categories characteristic of alphabetic tokens [also seen from the distribution of hand labeled tag categories for the different domains Table 10] :

1. ASWD : The tokens to be treated as words. eg. NATO, Kinshasa
2. LSEQ : The tokens which are to be treated as sequences of letters. eg. I.B.M, USA
3. EXPN : The tokens that have to be expanded using the abbreviation expander. eg km, blvd

8.4.4 Formulation of the Problem

The Non-Standard word Classifier problem can be given a statistical formulation as follows : The probability of assigning NSW tag $\mathbf{t}$ to observed string $\mathbf{o}$ can be estimated using the familiar Bayes approach as :

$$p(\mathbf{t}|\mathbf{o}) = \frac{p_t(\mathbf{o}|\mathbf{t})p(\mathbf{t})}{p(\mathbf{o})},$$

where $\mathbf{t} \in [ASWD, LSEQ, EXPN]$. In this equation:

1. The probability $p_t(\mathbf{o}|\mathbf{t})$ can be described by a trigram Letter Language Model (LLM) for predicting observations of a particular tag $\mathbf{t}$.

$$p_t(\mathbf{o}|\mathbf{t}) = \prod_{i=1}^{N} p(l_i|l_{i-1}, l_{i-2}),$$

   where $\mathbf{o} = (l_1 l_2, ...l_N)$ is the observed string made up of N characters. Such Letter Language Models have been used earlier for applications such as text compression(Bell, Cleary, and Witten, 1990) and estimation of language entropy (Brown et al., 1992).The language model used is the most widely adopted n-gram (in our case trigram) formulation. (Jelinek, 1997)

2. The probability $p(\mathbf{t})$ is the prior probability of observing the NSW tag $\mathbf{t}$ in the text.

3. The probability of the observed text or the normalization factor is given by

$$p(\mathbf{o}) = \sum_t p_t(\mathbf{o}|\mathbf{t})p(\mathbf{t})$$

This model assigns higher probability to shorter tokens in comparison to longer ones. However,the probabilities $p(\mathbf{t}|\mathbf{o})$ which are compared always correspond to the same token, compensating for the length factor.

8.4.5 Sub-classifier Features for Alphabetic Tokens

The sub-classifier for alphabetic tokens outputs the following letter language model based features for the full CART tree which performs the overall classification.

1. $p(\mathbf{t}|\mathbf{o}), t \in ASWD, LSEQ, EXPN$
2. $p_{max} = \max_t p(\mathbf{t}|\mathbf{o})$ (maximum probabilty of an alphabetic category).
3. $t_{max} = \text{argmax}_t\, p(\mathbf{t}|\mathbf{o})$ (most probable alphabetic tag).
4. *diff* = Difference between 1-best and 2-best probabilities $p(\mathbf{t}|\mathbf{o})$

Some samples of tokens with the 6 LLM features is displayed in the Table 8.

Token	p(ASWD|**o**)	p(LSEQ|**o**)	p(EXPN|**o**)	p_{max}	t_{max}	*diff*
mb	0.0001	0.0038	**0.9962**	0.9962	EXPN	0.9924
Grt	0.0024	0.0000	**0.9976**	0.9976	EXPN	0.9952
NBA	0.0017	**0.9983**	0.0000	0.9983	LSEQ	0.9966
Cust	**0.5456**	0.0000	0.4544	0.5456	ASWD	0.0912

Table 8: Samples of alphabetic tokens with LLM features

Domain	NANTC	ads	pc110	RFR
%alphabetic	54.52	38.65	40.31	31.75

Table 9: Distribution of alphabetic tokens in NSWs across domains

8.4.6 Distribution of NSW tokens in labeled data

If we look at the distribution of the alphabetic NSWs among the NSWs in the domains from Table 9, we find that percentage of alphabetic NSWs is substantial across all domains, ranging from 30% to 50%.We can get a prior distribution of different NSW tags for these alphabetic NSW tokens by looking at the hand labeled data in the training set of these domains [Table 10]

Percentages	NANTC	ads	pc110	RFR
ASWD	81.50	29.73	64.33	70.61
LSEQ	12.30	12.46	20.88	2.25
EXPN	5.68	55.60	11.33	25.88
MISC	0.52	2.21	3.47	1.26

Table 10: Distribution of NSW tags in the Alphabetic NSW tokens across Domains

8.4.7 Supervised Training and Evaluation Paradigm for the NSW Alphabetic Classifier

The Algorithm. As a first approach we train the NSW classifier in a supervised manner. For this approach to work we require hand labeled NSW data to know the NSW tag t assigned for the observed token o in each of the 4 domains.

The overall algorithm for the supervised training of the tag classifier for the 3 way alphabetic distinction is as follows:

The hand labeled training data for a specific domain is separated into a development(devtrain) section and a held-out(devtest) section. For each of the tag categories [ASWD, LSEQ or EXPN] we obtain a list of the tokens from the training data. This gives us 3 training sets which are used for training the Letter Language Models for the 3 major NSW tags.

The 3 Letter Language models are evaluated on the devtest data to obtain the test probabilities $p_t(\mathbf{o}|\mathbf{t})$ for each NSW token $\mathbf{t}$. The prior probabilities $p(\mathbf{t})$ for each NSW tag is obtained by the relative frequencies of the tag $\mathbf{t}$ in the devtrain data for the domain. These results are used to calculate the LLM features which are fed into the overall CART tag classifier.

Results. To obtain an estimate of the sub-classifier accuracy, an intermediate token error rate can be calculated on the alphabetic tokens. The accuracy of the classifier in predicting tags of alphabetic tokens is evaluated on the devtest sections of each domain. Three cases are examined:

1. Baseline : All tokens are labeled with the most frequent tag [obtained from the distribution of the NSW tags in the devtrain section of the domain] for that domain. So we label all tokens as ASWD in NANTC,pc110 and RFR domains and as EXPN in classifieds domain.

2. Uniform: Here Letter Language Model is used to obtain $p_t(\mathbf{o}|\mathbf{t})$ for each domain while $p(\mathbf{t})$ is kept same (uniform) across all tags. In other words we do not make any assumptions about the distribution of tags for the domain.

3. Unigram: Here Letter Language Model is used to obtain $p_t(\mathbf{o}|\mathbf{t})$ for each domain while $p(\mathbf{t})$ is based on the unigram frequencies of tags in that domain.

The accuracy of the classifier on the devtest in these 3 experiments is presented in the Table 11

8.4.8 Unsupervised Training and Evaluation Paradigm for the NSW Classifier

The Algorithm. In this approach we do not require any labeled NSW data thus making it useful for applying the technique to any new domain.The overall approach is described below:

Domain	NANTC	ads	pc110	RFR
Baseline	83.9[ASWD]	80.53[EXPN]	63.77[ASWD]	69.98[ASWD]
Uniform	88.92	98.5	90.83	97.36
Unigram	95.72	98.74	92.27	97.92

Table 11: Accuracy of the classifier:Supervised Training paradigm

We use the unlabeled data in a given domain to extract possible EXPN and LSEQ tokens by using some simple heuristics.These heuristics allow us to come up with a EXPN and LSEQ list of tokens on a domain dependent basis without need for labeled NSW tokens.The unsupervised lists serve as the training data for the ASWD,LSEQ and EXPN Letter Language Models.

1. *ASWD* list: The CMU dictionary[98k] with words with more than 4 characters is used as the training data.
2. *LSEQ* list: These tokens are extracted by searching for tokens which have alternate characters[which are lower or upper case] and periods in them. The LSEQ tokens from NANTC domain are considered to be fairly standard and can be used in other domains also.
3. *EXPN* list : The heuristics consist of some specific text patterns
   (a) Alphabetic tokens with no vowels.
   (b) Tokens followed by a period and then a token starting with a lowercase character.This is intended to search for abbreviations ending with a period and not followed by a sentence boundary. Example: frplc. ba in which frplc is an abbreviation that has be expanded.
   (c) Plural forms of the tokens falling in the previous 2 categories.

The number of occurrences of each of these tokens in the devtrain section of domain is also incorporated in these lists.

Evaluation. The NSW tokens from the test data (which is from the same domain as the training data) are extracted. The alphabetic tokens are filtered from these NSW tokens for the classifier input. The Letter Language Models for ASWD, LSEQ and EXPN types are evaluated on these alphabetic tokens to obtain $p_t(\mathbf{o}|\mathbf{t})$ for each of them. Based on these, the six Letter Language Model features are output to the CART model for the overall tag classification.

Classifier Accuracy for Unsupervised Training paradigm. The accuracy of the NSW classifier on alphabetic tokens in the unsupervised evaluation paradigm is presented in the Table 12

Domain	NANTC	ads	pc110	RFR
Accuracy	92.98	87.90	68.90	92.06

Table 12: Accuracy of the classifier:Unsupervised Training paradigm

Classifier results: Accuracy on all NSW tokens. The CART model for the overall classifier is built in two different ways: with and without Letter Language Model features in order to judge the effect of these extra features on the overall accuracy. In each case it is trained on the devtrain and tested on the devtest setion of the training data for each of the 4 domains. These accuracy values of the Overall NSW classifier are presented in Table 13

Accuracy	NANTC	ads	pc110	RFR
supervised:No LLM Feats	97.7	92.7	90.9	97.3
supervised:All LLM feats	98.1	93.5	91.8	96.8

Table 13: NSW Classifier Results : Accuracy on all NSWs

8.4.9 NSW Classifier: Cross Domain Testing

A set of experiments were carried out to examine the effect of training a classifier on one domain and testing the resulting CART model on the other domain. News(NANTC) domain was chosen as the training domain. The LLM features for the training domain were obtained using the supervised paradigm and the LLM features for the test domains were obtained in an unsupervised manner. We obtain 2 extra LLM features for both training and test domains that indicate if the NSW token is present in a LSEQ or a EXPN lookup list for that domain. The lookup list for the training is obtained from the labeled data while it corresponds to the unsupervised hypothesised list (Section 8.4.8) in the test data. The results from this experiment are presented in Table 14.

Domain	NANTC	ads	pc110	RFR
Accuracy	96.5	78.4	58.8	90.3

Table 14: NSW Classifier Results : Cross Domain Testing

8.5 Algorithmic Expansions

For most of the tags the expansion to a word sequence is algorithmic. That is although there may possibly be some choices in the pronuncation, the expansion algorithm is deterministic and not dependent of the domain or context.

The expander for EXPN, because it is the only expander that contains interesting content is discussed in the next section.

Even within these algorithmic expanders some are more trivial that others. The tagged tokens are treated as follows

NONE expanded to no words.

SLNT also expanded to no words.

PUNC expanded to itself.

ASWD expanded to itself.

LSEQ expands to token to a list of words one for each letter.

NUM expanded to string of words representing the cardinal number. This covers integers, float and roman forms.

NORD expanded to string of words representing the ordinal number. The token may be simply numeric, numeric appended *th*, *st* or *rd* included or may also be roman.

NDIG expanded to string of words one for each digit.

NYER expanded to words as in a pronunciation of a year. That is each pair of digits is pronounced as NUM except where the following two are *00*, where the group of four are pronounced as a whole.

NADDR expanded as words using the same algorithm as NYER.

NZIP spoken as string of digits. with silence for any dash within the token.

NTEL expanded as string of digits with silence for punctuation.

NIDE expanded as string of letters of pairs of numbers (as in NYER).

MONEY expands to string of words to say the number; deals with various currencies.

BMONY expands to string of words to say the number; deals with various currencies. The pronunciation for the following token (if it is *million*, *billion*, or *trillion*) is included within the expansion of this token, before the money identifier. Thus tokens which are preceded by a B-MONY token (*million*, *billion* or *trillion*) or are ASWD are expanded to nothing.

NDATE expanded as a number sequence.

NTIME expanded as a numbers with *hours*, *minutes* and *seconds* as appropriate.

PRCT pronounced as number with '%' sign removed and the word *percent* appended.

EMAIL written as is with no expansion.

URL written as is with no expansion.

OTHER written as is with no expansion.

MSPL written as is with no expansion: spelling correction is outside the domain of this project.

FNSP again written as is as there no record of the "proper" word. If fact there often isn't a proper word as in tokens like *Hmmmm* and *Arghhhh*.

8.6 Abbreviations

As we discussed in Section 7 one way of viewing the general problem to be solved in NSW expansion is an attempt to maximize the following quantity:

$$\hat{\mathbf{w}} \approx \operatorname*{argmax}_{\mathbf{w},\mathbf{t}} p(\mathbf{o},\mathbf{t}|\mathbf{w})p(\mathbf{w}) \quad (12)$$

$$= \operatorname*{argmax}_{\mathbf{w},\mathbf{t}} p(\mathbf{o}|\mathbf{t},\mathbf{w})p(\mathbf{t}|\mathbf{w})p(\mathbf{w}) \quad (13)$$

We assume $p(\mathbf{w})$ is given by the language model and that it is the task of the NSW expander to provide $p(\mathbf{t}|\mathbf{w})$ and $p(\mathbf{o}|\mathbf{t},\mathbf{w})$. Let us term the first two terms of the equation — $p(\mathbf{o}|\mathbf{t},\mathbf{w})p(\mathbf{t}|\mathbf{w})$ — the *lexical terms*.

In the case of abbreviations, which are often highly corpus-dependent (different abbreviations occur in classified ads from what one finds in pc110 netnews), the lexical terms are best estimated on a particular corpus. How one does this depends upon whether or not one has training data for the particular domain. In case one does, one can do *supervised* estimates of $p(\mathbf{o}|\mathbf{t},\mathbf{w})p(\mathbf{t}|\mathbf{w})$; in case one does not one must resort to *unsupervised* techniques. We describe each of these approaches in the next two subsections. In Section 8.6.3 we present a decision-tree model for predicting the probability of a particular abbreviation given the word — $p(\mathbf{o}|\mathbf{t},\mathbf{w})$.

Finally in Section 8.6.4 we describe experiments on both the supervised and unsupervised methods for classified ad data.

8.6.1 Supervised method.

If one has an appropriately tagged training corpus for a particular domain, one can compile a list of abbreviations from that corpus. This is equivalent, of course, to the traditional approach used in TTS systems or the LDC text-conditioning tools, of hand-constructing a list of abbreviations for a particular domain. The main difference is that we also estimate the probability $p(\mathbf{o}|\mathbf{w}) = p(\mathbf{o}|\mathbf{t},\mathbf{w})p(\mathbf{t}|\mathbf{w})$ (in our case using the maximum likelihood estimate), given the distributions of words and their abbreviations in the training data. Note that since in this case since we know the word and its abbreviation(s), we do not need to separately estimate $p(\mathbf{t}|\mathbf{w})$ (the probability that a word will be abbreviated in any form) and $p(\mathbf{o}|\mathbf{t},\mathbf{w})$ (the probability of a particular abbreviation given that we know the word and that it will be abbreviated). Rather we can estimate their product directly.

8.6.2 Unsupervised method.

In the unsupervised method we assume that we have the following three things:

1. An automatically inferred list of "clean" words from the corpus;
2. An automatically inferred list of potential abbreviations from the corpus;
3. A procedure for aligning clean words and n-grams of clean words in the corpus with their potential abbreviations.

The first two items can be inferred using the classifier described Section 8.4. Given the first two items, in order to produce the alignment required by

3 we need an abbreviation model that for a given full word or phrase, can predict the set of possible abbreviations.

For example, suppose we have the abbreviations *kit* and *livrm* in our corpus as in the example

eat-in kit livrm dinrm 17x25 famrm

and that elsewhere in the corpus we find examples like the following:

... eat-in kitchen ...
... living room ...

We would like to guess that *kit* might be an abbreviation of *kitchen* (among other possibilities) and that *livrm* is a possible abbreviation of *living room* (among other possibilities). We would also like to assign some nominal probability estimate for $p(\mathbf{o}|\mathbf{t},\mathbf{w})$ — the probability, given that one wants to abbreviate, e.g., *kitchen*, that one would do it as *kit*. Note that at this stage we do not have any methods for estimating $p(\mathbf{t}|\mathbf{w})$ — the probability that one would abbreviate *kitchen* in any form, though as we shall see, it is possible to reestimate the combined term $p(\mathbf{o}|\mathbf{w}) = p(\mathbf{o}|\mathbf{t},\mathbf{w})p(\mathbf{t}|\mathbf{w})$ in some conditions.

8.6.3 Tree-Based Abbreviation Model.

By and large abbreviations involve removal of letters, though there are some cases where one finds (usually pronunciation-influenced) letter substitutions (e.g. *lgstx* for *linguistics*). There are various obvious generalizations on how one may form abbreviations:

- Delete vowels and possibly sonorant consonants (*hdwd* for *hardwood*).
- Delete all but the first syllable (*ceil* for *ceiling*).
- Delete all but the first letter (*N* for *north*).

However it is in practice difficult to formulate a set of rules to handle abbreviation, much less provide an estimate of how likely a given abbreviation is.

We therefore decided to train a decision-tree model using the classification and regression tree (CART) (Breiman et al., 1984) code by Mike Riley. The system was trained on 854 pairs of abbreviations and their expansions — taken from the classified ads, as this is the domain richest in productive deletions. The features chosen were as follows:

- The class of letter two to the left (−2), one to the left (−1), one to the right (+1) and two to the right (+2) of the current letter, as well as the class of the current letter itself. Class was here defined (somewhat solecistically) as obstruent, sonorant, vowel and "y" (containing just the letter "y").
- The boundary to the immediate left and right of the consonant: word-boundary, morpheme-boundary or no boundary.
- The fate of the −2 letter and −1 letter: deleted or not deleted.
- The fate of the +1 letter and +2 letter: deleted or not deleted.

The last feature — which clearly depends upon knowing the future output — was shown nonetheless to produce better results, cross-validated on the training data: without this feature the tree performed at 85% and with the feature it performed at 88% in predicting deletion. Note that the baseline for this task is 54%.

Information about word-internal morpheme boundaries was obtained using a crude morphological analyzer, that allows a concatenation of one or two three-or-more-letter words from the CMU dictionary, possibly followed by *-s* or *-es*.

The tree-based model was compiled into a weighted finite-state transducer (WFST) using an algorithm related to that reported in (Sproat and Riley, 1996). In practice it was found that the model had too great a predilection to allow deletion of word-initial letters: this in part had to do with errors in alignment in the training data, but was also affected by the small training sample size and the fact that word-initial deletion of some letters is occasionally found (e.g. *xpwy* for *expressway*). We therefore arranged for the compiled transducer to disallow word-initial deletions. We also had little data on added characters — e.g. ".", "/" — which one finds in abbreviations, usually as indications that one is dealing with an abbreviation ("."), or that there are some deleted characters ("/" as in *w/* for *with*). In the WFST we optionally allowed a sequence of deleted characters to surface as one of these characters.

Of course, by restricting the abbrevation operation to deletions one is ruling out some abbreviations such as *sociolx* for *sociolinguistics*, which involve letter substitutions, or the use of non-alphabetic characters such as "+" for *plus*. A restricted set of the former transductions (e.g. "cs" becoming "x"), could be added to the model. The latter kind is impossible to infer anyway: one simply must depend upon a lexicon that tells you that

SO	SO	SO
SHORE	SHORE	SHORE
</s>	</s>	</s>
40	@NUM+PL	@NUM+PL
+	@EXPN	@EXPN
MODERN	MODERN	MODERN
BRK	BRICK	<UNK>
APTS	APARTMENTS	<UNK>
ON	ON	ON
</s>	</s>	</s>
4	@NUM+PL	@NUM+PL
+	@EXPN	@EXPN
ACRES	ACRES	ACRES

Table 15: Sample text and annotation for abbreviation expansion experiments.

one of the readings of "+" is *plus*: this is unlikely to be amenable to purely unsupervised inference.

8.6.4 Experiments.

Overview of Experimental Paradigm. All experiments described here were run a portion of the classified ad corpus divided into 307,735 tokens of training data, and 76,676 tokens of test data.

From the initial classification we have:

- An estimate of which words are *standard*.
- Non-alphabetic or mixed *non-standard* words.
- An estimate of which words are *alphabetic non-standard*: i.e. EXPN, LSEQ or ASWD.

Note that the classifier assumed here was the initial pre-workshop classifier, rather than the classifier developed in the context of the workshop. This is expected to affect the number false positives (see the discussion at the end of this section), and the number of true EXPN's that were missed by the method: but the overall trends in the data are probably not affected.

The language model used in all cases was a trigram-language model using modified *Kneser-Ney* backoff (Kneser and Ney, 1995; Chen and Goodman, 1998). In the case of supervised training, the language model was trained

on the training data, with all ordinary words as themselves, all EXPN's as their expansion, and all non-alphabetic or mixed NSW's as their tag, with the additional stipulation that plural NUM's were distinguished from *1*. The kind of training data given to the language modeling tools in the supervised case is shown in the second column of Table 8.6.4.

For unsupervised learning, the language model is not supposed to know the true expansion of an abbreviation, since for an untagged corpus this would be unknown. On the other hand, we do not want to train the language model with text that includes the abbreviations. So in the unsupervised case we transduce all EXPN's to the unknown word tag (<UNK>), with everything else being the same as in the supervised case. This is shown in the third column of Table 8.6.4.

For the supervised lexical expansion models, we simply collect the set of EXPN's and their hand-annotated expansions, and compute the statistics for each case.

For the unsupervised lexical models, all we know is the set of potential EXPN's and the set of clean words. We can also compute the set of clean word pairs that occur in the training corpus. These lists can be further pruned in various ways, e.g. to eliminate low-frequency words. Call the thus-derived lexicon of EXPN's NSW and the lexicon of clean words and clean word pairs SW. Denote with A the abbreviation model implemented as a WFST, as described in Section 8.6.3. In order to compute our list of potential abbreviation-word pairs, we need to compute the composition of SW with A and NSW. For the testing phase, we need to model the expansion of EXPN's into words or word pairs, rather than the other way around, so we actually want to compute the inverse of this composition:

$$[SW \circ A \circ NSW]^{-1} \tag{14}$$

In the testing phase we assume that, since we are dealing with unseen data, we do not know exactly which words are EXPN's, but only which words are alphabetic NSW's and which are therefore *potential* EXPN's. (Once again, we were using data tagged with the pre-workshop classifier.) All such potential EXPN's are submitted to the expander which will produce a weighted lattice of alternatives. This results in errors (which are reflected in the error rates reported here) where true alphabetic EXPN's were missed because they were not classified as EXPN's, as well as "false positives" (which are *not* reflected in the error rates) of tokens that were treated incorrectly as EXPN's; the latter point is discussed in the discussion at the end of this

section. The error rate is computed by computing the *token error rates* (see Section 9.4) on the true EXPN's.

Finally, note that in these experiments we uniformly map characters to upper case, since the language modeling tools prefer case-uniform text. This results in some loss of information. For example w in the classified ads is usually *with*, whereas W is usually *west*, something that the supervised methods could take advantage of. On the other hand, the unsupervised methods have no way of taking advantage of case information: we can predict that either *west* or *with* could be abbreviated as w (or W), but we currently have no basis for predicting a preference for upper case in one instance and lower case in the other (though in this case this is surely related to the content-word/function-word distinction, plus the fact that *west* is frequently capitalized). Hence for the sake of uniformity across the two training conditions we ignore case distinctions.

Supervised Method. The supervised method was tested in two conditions:

1. Without the language model, in which case we pick the expansion that maximizes $p(\mathbf{w}|\mathbf{o})$. In other words, the expander is behaving as a unigram language model.

2. With the language model, in which case we pick the expansion that maximizes $p(\mathbf{o}|\mathbf{w})$.

The error rate in the no-language-model case was 6.7%, and in the with-language-model-case 4.8%.

The relatively low error rates mean that the training corpus is covering most of the cases that one encounters in the test corpus, though about 5% of the cases in the test corpus are still unseen.

The relatively small difference between the with-language-model and no-language-model cases reflects the fact that most abbreviations are unambiguous in this domain, though some (e.g. *DR* for *drive* or *dining room*) are ambiguous. Interestingly most of the difference for the particular run done in this experiment has to do with a bug in the database labeling: the abbreviation *SF* was tagged as meaning *south facing* in most instances, though a few correct instances of *square foot/feet* were found. Without the language model the expansion with the highest lexical prior — *south facing* — was chosen. With the language model, the correct choice was allowed and preferred.

Unsupervised Method: Experiment 1. In all the unsupervised methods tested, language modeling was used.

The first unsupervised method constructed *SW* out of:

- All singleton clean words.
- All clean-word bigrams occurring in the data, occurring a minimum of 3 times.

The three-word minimum was felt to be a reasonable cutoff, since anything occurring fewer than three times is probably unreliable. This first test had an error rate of 33%.

Unsupervised Method: Experiment 2. It was observed that some of the errors introduced in the first experiment involved proposed expansions where the target was a rare word. For example, *SF* was expanded as *SURF*, which occurred only twice in the training data. Thus in the second experiment we tried removing low-frequency unigrams from the set of clean words. *SW* then consisted of:

- All singleton occurring more than 10 times
- All SW bigrams occurring in the data filtered with the above list.

Here the error rate increased to 34.5%.

Unsupervised Method: Experiment 3. In experiment 3, we returned to the original way of compiling *SW*, and added a third component, namely a short list of arguably standard abbreviations:

> *aug* (*August*); *av* (*Avenue*); *blvd* (*Boulevard*); *ext* (*extension*); *ft* (*foot*, *feet*); *inc* (*incorporated*); *l* (*left*); *n* (*north*); *r* (*right*); *rd* (*road*); *st* (*street*); *w* (*west*); *w/* (*with*); *x* (*extension*); *sf* (*square foot*, *square feet*); *etc* (*etcetera*); *n.w* (*northwest*)

Here the error rate reduced substantially, to 24%.

One might be inclined to think this as cheating but it is reasonable to put a more positive spin on this: it gives some indication of how much of a reduction in error rate one can do if one is willing to do a small amount of handwork.

Unsupervised Method: Experiment 4. In experiment 4 we had the same setup as experiment 3, but returned to investigating further purely automatic methods. Given *SW* defined as for experiment 3, we:

- Reran the *expander* and the *language model* on the *training data.*
- Selected the most frequent expansion found for each potential EXPN in the training data as "truth".
- Used these single items to predict the expansion of potential EXPN's on the test data.

Here again the error rate was reduced substantially, to 19.9%.

Unsupervised Method: Experiment 5. An obvious unfortunate property of experiment 4 is that any given abbreviation can only have one expansion, whereas we know that at least some EXPN's are ambiguous. We tried allowing for such ambiguity by following the same general procedure as in experiment 4, but this time selecting two alternatives, if the second most frequent alternative occurred at least half as many times as the most frequent alternative. In such cases we retain both alternatives, and weight them according to their frequency: more specifically, we take the distribution from the training-data run to be truth, and we estimate $p(\mathbf{o}|\mathbf{w}) = p(\mathbf{o}|\mathbf{t}, \mathbf{w})p(\mathbf{t}|\mathbf{w})$ as we would in the supervised case.

Unfortunately this resulted in no reduction of error rate: 19.9%. However we decided to retain the ability to allow for alternatives since it at least has the potential to do better than a method that only allows one alternative, as we have seen in the supervised case.

Unsupervised Method: Experiment 6. In experiment 5, although we treated the guessed expansions on the training data as truth, we still used the same language model on the test data as had been trained on the original unexpanded training data. The problem with this is that for some words we have very poor estimates of their true probability of occurrence from their occurrence in the raw training data: this is because they are highly likely to be abbreviated. Thus *bedroom* is about twice as likely to occur abbreviated (usually as *BR*) in the classified ads as it is to occur fully spelled out. Experiment 6 addresses this deficiency by retraining the language model on the expanded training data. Once the language model was retrained, we once again rescored the lattice of possible expansions for the training data,

Experimental Condition		Token Error rate (%)
Supervised	no language model	6.7
Supervised	with language model	4.8
Unsupervised	Experiment 1	33.4
	Experiment 2	34.5
	Experiment 3	24.2
	Experiment 4	19.9
	Experiment 5	19.9
	Experiment 6	19.5

Table 16: Summary of experimental results: abbreviation expansion.

and reestimated $p(\mathbf{o}|\mathbf{w}) = p(\mathbf{o}|\mathbf{t}, \mathbf{w})p(\mathbf{t}|\mathbf{w})$ for each of up to two expansions for each abbreviation.

This method resulted in a modest reduction in error rate, to 19.5%.

The results of each experiment are summarized in Table 8.6.4.

8.6.5 Brief Synopsis of Abbreviation Expansion Tools

The released version of the abbreviation expansion tools contain tools for constructing abbreviation expanders for both the supervised and unsupervised scenarios.

For the supervised case, the tools (as opposed to the test versions we described above) assume case sensitivity in constructing the tables so that, e.g., w might expand by default to something other than W. Two such models are constructed, one appropriate to the situation where you have no language model (maximizing $p(\mathbf{w}|\mathbf{o})$) and the other which is appropriate to the situation where one has a language model (maximizing $p(\mathbf{o}|\mathbf{w})$).

For the unsupervised case, the setup outlined in Experiment 5 is provided — i.e., with no retraining of the language model on the tagged data. (The user can of course also retrain the language model on this retagged data.)

8.6.6 Discussion

The work discussed in this section has shown that if you are unwilling or unable to tag data from a novel domain, and if that domain provides enough "clean" text to allow one to make inferences about possible expansions for abbreviations, and if one is willing to tolerate approximately four times the

error rate of a supervised method, then it is possible to automatically infer models that handle abbreviations.

A post-hoc analysis of the errors in experiment 6 leaves even more room for optimism: fully half of the "errors" were either not errors at all because the hand-labeling was wrong, or were acceptable alternatives. A common instance of the latter was *bath* rather than *bathroom* as an expansion of *BA*, which in the context of real estate descriptions is perfectly acceptable, or even preferred. Thus the true error rate may be closer to about 10%.

One serious problem is "false positives" — cases where a token was expanded that should not have been. These were not counted in the errors described above, but were nonetheless quite substantial: in experiment 6, for instance, there were about 80% as many false positives as counted errors. This of course relates to the reliability with which we can detect a potential abbreviation.

The approach described here relates to a couple of other strands of research. One is automatic spelling correction, where the problem is to find the closest and contextually most appropriate correctly spelled target word to a misspelled token. The typical approach is to assume that the correct target is within some small edit distance of the misspelled token, and then use some form of language modeling technique to select the correct one given the surrounding words (see, e.g., (Golding and Roth, 1999)). There are three differences between the present work and the previous work on spelling correction, however.

First of all, as has already been mentioned, spelling correction algorithms limit the target words to those known clean words that are within a small edit distance of the token. In contrast, we made no assumptions about how distant in terms of edit distance the abbreviation could be from its target. Secondly, the target correctly spelled word corresponding to a single misspelled token is itself assumed to be a single token. For abbreviations, we must at least allow that a single abbreviation come from potentially two original words. Thirdly, most work on spelling correction systems assume you know that you are dealing with an error. Indeed, it is typical to evaluate such systems by demonstrating performance on a few selected spelling errors. In contrast, one of the tasks that we attempt to address in this work is detection of potential expansions, in addition to prediction of their actual expansion.

There is also a similarity between the current work and work reported in (Taghva and Gilbreth, 1995) on using approximate string matching methods to induce interpretations of acronyms or letter sequences from their full word expansions found somewhere in the immediate context of the given

letter sequence. Thus from a text such as *today, leaders of the North Atlantic Treaty Organization (NATO) met in Brussels ...*, one would infer *North Atlantic Treaty Organization* as a possible interpretation of *NATO*. The present method, however, does not presume that the answer is in the immediate context, or even particularly close by.

8.7 Language Modeling

As mentioned earlier, we used trigram language models constructed with modified Kneser-Ney smoothing. In terms of perplexity, modified Kneser-Ney smoothing has been found to consistently outperform all other popular n-gram smoothing methods (Chen and Goodman, 1998). Language models were trained separately for each of the four domains, and separately for the unsupervised and supervised systems in each domain. The vocabulary for each n-gram model was constructed by collecting every token that occurred in the corresponding training set. The method by which the unsupervised and supervised training sets are created is described in Section 8.6.4. Some tokens are mapped to a tag label, in which case the tag label is added to the vocabulary and treated like any other token in the vocabulary.

While we use language models within a source-channel framework as is done in speech recognition, there are several significant differences between language modeling for text normalization and language modeling for speech recognition. The most notable difference is the bootstrapping issue: the most useful language model for text normalization is presumably one built on normalized text, which by assumption we do not possess. As shown in Section 8.6, we can still get useful performance from language models built on unnormalized text, *e.g.*, when an abbreviation occurs in its expanded form in some portion of the text. However, if a word is consistently abbreviated in text, a language model will not help determine its expansion unless trained on text, such as hand-labeled text, that includes the expansion. Because of the need of hand-labeled text, the language models we used were constructed on substantially smaller data sets than used in many speech recognition systems. Because of the sparse data issue, the language models constructed were unable to discriminate abbreviation expansions as well as they might have.

9 Performance Measures

This section describes some issues in choosing an automatic scoring method for calculating the performance of the models we built as well as comparing

these models to pre-existing systems that attempt the same task.

9.1 Truth

In order to evaluate our performance it was necessary to generate "truth" in the sense of the words that would be said given "perfect" knowledge. This "perfect" knowledge consists of the hand labeled NSW tags (and in the case of EXPNs, the expansions given). The "truth" was generated for each marked up XML file in the each corpus. For the purposes of the results presented here only one value for truth was given even though there may in reality be more than one reasonable way to say a token.

However we do not wish to exclude the possibility that truth should allow for alternatives. For synthesis obviously only one value is sufficient but for language modeling we wish to produce lattices with probabilities for choices, though so far we have not yet done that.

9.2 Noise

Given that labeling was done by a number of different humans the labeling contains a certain amount of noise. This noise includes mislabeling (i.e. mistakes) as well as alternative labeling in the same context but different occurrence (i.e. genuine choice). The alternatives in labeling may be because there are genuinely different options in saying a token or just that different labeler chose different tags. Although a number of systematic passes over the database have been made to correct errors, we know errors still exist.

Although the amount of noise in the data is small and would normally not be an issue, our models have very low error rates compared to say speech recognition. Our error rates can be as small as 0.30% or less, and so even an error rate of 0.1% will affect our measurement of error rate.

Two methods have been used to estimate the amount of noise in our labeling. First in generating truth the algorithmic expanders can flag when a given token with a given label doesn't match the expected form of that type. For example where a token marked NYER actually consists of a range of years *1990-95*. Although we have been quite liberal in accepting such labeling there are still a number of cases that the algorithmic expanders can't deal with. For each database we counted the number of tokens for which some part is detectably not expanded properly. These DNTs (detectably noisy tokens) include labeled misspellings, and funny spellings and tokens labeled OTHER. (It does not include URLs and EMAIL tags).

	NANTC	ads	pc110	RFR
total # tokens	4.49m	477k	289k	220k
# of DNTs	1546	4583	922	134
% DNTs	0.034	0.96	0.32	0.061

To put these figures in perspective, as we will see later these detectable noisy tokens are contributing to around 10-15% of our token error rate of our best predictive models. But these only cover the detectable errors, and we also wish to judge how good our generated words are with respect to human acceptability of the expansion both due to undetectable noise in the data and where there exist valid alternatives.

9.3 Manual Evaluation

In this section, we discuss the methodology used to manually evaluate the performance of our system relative to other existing systems as well as the results of this evaluation. Briefly, we examined the output of several normalization systems and manually judged the acceptability of the output to estimate token error rates. While automatic evaluation is more desirable, this requires the presence of reference data describing all of the acceptable normalized forms of some raw text. The hand-labeled text collected in this project does not meet this standard for two primary reasons:

- The hand-labeled text only specifies a single acceptable alternative, not all acceptable alternatives. There are many different types of tokens that can be expanded in multiple ways.
- The text was not completely labeled, in the sense that some ambiguities still exist. For example, the expansion of common abbreviations such as *BR* in classified ads is not manually specified but is instead performed automatically. Thus, we do not know if the correct expansion is *bedroom* or *bedrooms*. Another ambiguity is the presence of punctuation after abbreviations. For example, the manual labeling does not specify whether there should be end-of-sentence punctuation after an abbreviation such as "*St.*".

Due to resource limitations, it was impractical to create reference data describing all acceptable alternatives in sufficient amounts for all four domains to precisely estimate error rates during the workshop; thus, we chose a manual evaluation scheme.

The basic procedure we used is as follows: We randomly sample a space-separated token from the unnormalized version of our test set. We then

present that space-separated token and surrounding tokens and the normalized version of these tokens to the adjudicator, who determines the acceptability of the normalization of the original token. Through repeated sampling, we can estimate the token error rate of a normalization system.

The guidelines given to adjudicators for determining the acceptability of a normalized token is included as an Appendix. The most notable points are that:

- All of the normalized tokens that should be produced from the original raw token must be present and correct (and no extra tokens must be present) for the token to be judged correct. There is no partial credit.
- The presence of punctuation must be predicted correctly, but not the identity. For example, it is acceptable to substitute a sequence of dashes for a single dash. This decision was motivated by the fact that the presence of punctuation is useful for guiding speech synthesis and language model training, but that the identity of punctuation is less important.

To reduce the number of samples needed to achieve a certain accuracy in estimating error rate, we did not sample tokens uniformly from the test set but instead used the technique of *importance sampling*. Intuitively, by sampling more frequently from the tokens that are likely to be errorful, we can emphasize the differences in performance between various systems. We partition tokens into several catgories, sample from each category depending on the error rate and the frequency of the category, and scale the number of errors found in each category appropriately to get an estimate of the overall error rate.

More precisely, we partitioned the (unnormalized) space-separated tokens in the test sets into fourteen categories according to typographic information. Example categories include: tokens containing only lower-case letters optionally followed by a comma; tokens containing only lower-case letters followed by a period; tokens containing a digit; and tokens containing a hyphen or slash. On training data, we manually evaluated the LDC tools on thirty tokens from each category in each of the four domains to get a rough estimate of the error rates we would find. Using these estimates and the frequency of each token category in the training data, we calculated the proportion of samples to take from each category in each domain to minimize the standard error of our overall estimate of error rate. A simple analysis reveals that the number of samples to take from a given category to

satisfy this condition should be proportional to $f\sqrt{E(1-E)}$, where f is the frequency of the category and E is the error rate on tokens in the category.

We selected a total of about 500 samples to evaluate each system in each domain, using the above calculation to determine the number of samples taken from each category, but taking a minimum of thirty samples from each category. For each sample token, we presented the output of each of the evaluated systems in sequence but in random order. As each system is evaluated on the same tokens and as the output of each system is presented to the adjudicator in succession, paired significance tests can be meaningfully carried out.

Because of the manual labor required to evaluate each additional system using this methodology, it was only practical to do a limited number of such evaluations. Thus, we evaluated only one of the many systems developed during the workshop in this manner. To evaluate how the many developed systems compare against each other, we used the collected hand-labeled data as reference text and used automatic means to score system output against the hand-labeled data. This method proved to be quite useful and effective for internal evaluation.

9.3.1 Aligning Raw and Normalized Text

As mentioned earlier, for each token to be judged we present that token and surrounding tokens as well as the normalized versions of these tokens to the adjudicator. To do this, we need to know which raw tokens align with which normalized tokens, and this alignment information cannot easily be recovered from all of the systems evaluated. In particular, the LDC tools are a collection of complex Perl scripts from which alignment information is difficult to extract. Consequently, we developed an algorithm to automatically align raw and normalized text, given a large corpus of normalized text.

This alignment is not trivial because the tokens in normalized text can be substantially different from the tokens in the raw text. For example, classified ads can be almost entirely composed of abbreviations and numbers, so that normalized ads have very few tokens identical to those in the original ad. In addition, the normalized version of a token may have a very different length than the original; *e.g.*, a single numeric token often expands to many words.

This alignment task can be viewed as a simple version of the bilingual word alignment task faced in machine translation, and we use similar techniques as those used in bilingual alignment (Brown et al., 1990). The bilin-

raw token	normalized token
rm	*room*, *family*, *colonial*
flrs	*floors*, *hardwood*
Rd	*to*, *road*, *route*
flr	*floor*
LR	*with*, *room*, *kitchen*, *fireplace*, *living*, *dining*

Table 17: Examples of token translations found for unnormalized/normalized text alignment in the classified ads domain

gual word alignment task consists of determining which words in a bilingual corpus are translations of each other, a bilingual corpus being two corpora containing the same text but in different languages. Our task can be considered an instance of this task, where instead of text in two different languages we have an unnormalized text corpus and its normalized version. Our task is somewhat easier than the usual bilingual alignment task because of the presence of a large number of *cognates*, or tokens with the same spelling in both languages, and because there is very little word-order rearrangment in normalized text, unlike between distinct languages.

A strategy used in bilingual word alignment is to first construct a bilingual dictionary, or list of which words in one language translate to which words in the other language. This can be done by first aligning higher-level units (such as sentences) in the two corpora using some other means, such as aligning sentences with similar length (Brown, Lai, and Mercer, 1991), and then collecting word pairs that tend to co-occur in aligned sentences. These words will usually be translations of each other.

In our scenario, we have text that has been segmented into paragraphs, and this segmentation is undisturbed by normalization. By counting how often token pairs tend to co-occur in corresponding unnormalized and normalized paragraphs, we can approximately reconstruct which tokens are expanded to which tokens by a normalization system.

More precisely, we segment unnormalized text into tokens by keeping together alphabetic characters and separating all other (non-space) characters into their own token. (This has the advantage of letting us induce the expansions of numbers on a digit-by-digit basis.) Normalized text is segmented into tokens using spaces. Then, we take all token pairs that co-occur in aligned paragraphs significantly more often than one would expect given the total number of paragraphs each occurs in to be "translations" of each other. We use a χ^2 test with significance threshold of 10^{-50}. Some examples

```
3300+                                sf     10       Rm,
three thousand three hundred PLUS SQUARE FOOT ten ROOM ,

5BR,           2BA,           library,  DR,
five bedroom , two bathroom , library , dining room

HW       flrs,    gas FP's,        2C    gar,  yrd. Grt loc.
HARDWOOD FLOORS , gas FIREPLACES , two C gar ,          YARD
```

Table 18: Examples of text alignment in the classifieds ads domain

of translations discovered for the classifieds domain are given in Table 17. While the translation lists have many extraneous entries, this does not affect performance unduly due to the highly constrained nature of the task.

To calculate the alignment between the unnormalized and normalized versions of a paragraph given a set of translation pairs, we use the dynamic programming algorithm for calculating word edit distance between two word sequences. However, instead of using a cost of 1 for each insertion, deletion, and substitution, we use a cost of 3 for insertions and deletions and 4 for substitutions as is done by NIST's `sclite` tool for calculating speech recognition word-error rates. Furthermore, a cost of 0 is assigned to substitutions between tokens with identical spelling and a cost of 1 is assigned to substitutions between nonidentical tokens that are "translations" of each other. These cost assignments are arbitrary and were found to yield adequate performance. As some paragraphs are extremely long and the edit distance algorithm is quadratic in time and space, we used beam search to prevent excessive computation. Some example alignments from the classifieds domain are presented in Table 18. While not perfect, these alignments are mostly correct.

In the actual evaluation process, we present a total of about 120 characters of context around the token to be judged and around the normalized token that we decide is aligned with the original token. The alignment process has done its job adequately if the correctness of the original token can be judged given the amount of context presented. We found that less than 0.3% tokens were misaligned in each domain according to this criterion, where many of these "misalignments" were not the fault of the alignment algorithm. For example, some tokens expand to more than 120 characters when normalized, and thus cannot be judged when presenting only 120 characters. Thus, the alignment algorithm implemented was entirely adequate

	classifieds	news text	pc110 news	recipes
Festival	34.5% ± 1.8%	0.7% ± 0.7%	5.9% ± 1.1%	4.2% ± 1.0%
LDC	32.4% ± 1.8%	0.2% ± 0.6%	6.3% ± 1.1%	3.0% ± 1.0%
m4	12.3% ± 1.5%	0.4% ± 0.7%	3.4% ± 1.0%	2.6% ± 0.9%
hand-label	9.6% ± 1.4%	0.3% ± 0.6%	2.2% ± 0.9%	2.6% ± 0.9%

Table 19: Token error rates of various systems on various domains as estimated through manual evaluation

for its purpose. When a token could not be judged from 120 characters of context, the whole surrounding paragraph is presented to the adjudicator.

9.3.2 Results

Using the methodology described above, we estimated the token error rate of three different normalization tools: the Linguistic Data Consortium normalization tools, the normalization tools provided with the Festival speech synthesis toolkit, and the system we developed that is referred to as *m4* elsewhere. We also evaluated the quality of the hand-tagged data described in Section 6. Performance was evaluated separately in each of the four domains considered.

The results of this manual evaluation are presented in Table 19. We also performed Student's t-test for paired samples to test the statistical significance of performance differences found. All performance differences in the table are significant at the 2% level except for the following: in the news text domain, *no* differences are significant except for that between LDC and Festival; in the *pc110* domain, the difference between LDC and Festival is not significant; and in the recipes domain, the differences between LDC, *m4*, and the hand-labeled data are not significant.

For each domain, we see that *m4* performs at least as well as the LDC and Festival systems, and for some domains it performs substantially better. Furthermore, it is not much worse than the hand-labeled data in each domain. While the token error rate of the hand-labeled data is quite low in three of the domains, it is near 10% for the classifieds domain. To achieve performance better than this level, it is likely that the labeling of the classifieds data must be improved. The *m4* system is trained on hand-labeled data, and its performance is probably limited by the quality of the training data used. All of the systems evaluated perform quite well on the news text.

To make further distinctions between the performance of different systems on this domain would require a substantially larger sample set than was actually used.

We perused the errors of the hand-labeled data in the four domains, and found that the errors came from a variety of sources over the different domains. Examples of the more common errors are:

- In the classifieds domain, punctuation was not correctly placed after abbreviations. As mentioned earlier, this information is not provided by labelers.
- In the classifieds domain, many non-standard words were not correctly identified as non-standard and were thus not presented to labelers, *e.g.*, *OH*, *SE*, *PH*, *ac*.
- In the *pc110* domain, E-mail addresses and URL's were not expanded correctly. This information was not generally provided by labelers.
- In the recipes domain, whether an abbreviation expands as plural or singular (*e.g.*, *3g*) is often incorrect. Again, this information was not provided by labelers.

9.4 Measurement criteria

Each database was split into train and test sets with approximately one third for test and two thirds for training. Splits were done on a per-file basis and although there are distinct sources for parts of the news and classifieds data (i.e. different newspapers) we did not take account of these, and so files from each newspaper appear in both train and test (except in a few cases).

In addition we split the training set into a devtrain and devtest sets with approximately a 90/10% split. The test set was only used to test what we considered final models and no error analysis was ever carried out on those results. The devtrain and devtests however were investigated fully, though models were only trained from data in devtrain.

In order to provide day-to-day measurements of how our models improved and to show relative accuracy over different models we required an automatic scoring method. Using the user based scoring method discussed above would be too expensive. Thus, despite the known noise in our data and in truth itself, we choose two figures to measure the accuracy of our models.

token error rate: the percentage of original unsplit tokens whose expansion to words does not completely match the expansion to words in the truth.

word error rate: the percentage of wrong words in an expansion (including insertions deletions and substitutions) with respect to truth.

The first of these is a good measure because the number of original unsplit tokens is the same for almost all of the models we present results for (The LDC text conditioning tools do not preserve token boundaries over their expansions so we can only report word error rates.) The word error rate, although highly correlated with token error rate is not so straightforward, if a model mistakenly identifies an abbreviation as a letter sequence the number of word errors will be greater than if it wrongly identifies it as a ASWD.

Another measure that would be interesting is the split token error rate. This however can't be calculated as our splitter does not necesssarily match the split tokens in truth.

However looking at the actually erroneous tokens we feel these figures do adequately represent the relative accuracies of these models.

9.5 Baseline systems

One of the purposes of this project is to introduce the idea of measuring the accuracy of a text analysis system. Thus in order to place our own models in context we have choosen two publicly available pre-existing text analysis systems to show what the current state is.

The first is the LDC text conditioning tools. These are standard tools used in normalizing text in order to build language models for automatic speech recognition. They consist of a set of simple rule driven scripts that have been augmented with rules to expand text as used by various DARPA evaluations. Although not inetended to be domain dependent they are heavily biased towards news type data.

The second system is Festival, a publicly available text-to-speech synthesis system. The text analysis part consists of both rule driven and statistical prediction models for number and homograph disambiguation. Festival was primarily trained and tested on news-type data though an email database was also used. Although we are using Festival as the framework for building our new NSW models this test is on the 1.4.0 release without any benefits from new models produced in this project.

	LDC tools		Festival	
	TER	WER	TER	WER
NANTC	–	2.88	1.00	1.38
classifieds	–	30.81	30.09	33.48
pc110	–	22.36	14.37	32.62
RFR	–	9.06	6.28	16.19

As the mechanism used to generate the truth that these systems are compared against is using some of the same mechanism that the Festival text analyser is using there is probably a slight bias in these results towards Festival. Looking closely at "errors" between the LDC NANTC results we feel that perhaps a truer error rate would be close to 1.5, as many of the "errors" fall into punctuation, hyphenation type problems, although we tried to account for many of these. Also given the results of the use oriented measurement is seems we can't make any strong statement about the different between Festival and the LDC tools performance on NANTC.

However we feel that the above table shows that these existing systems do reasonably on NANTC-type data and perform miserably on any of the other domains.

9.6 NSW based models

There are five threads in the presentation of these results. First we'll present the best domain dependent model. This uses the training data as much as possible and gets our best results. The second thread is modifying our best model by removing parts of it to indicate how important each part is. The third thread is to incrementally add oracles at each stage that look at truth. This is another indication of how well each component works.

The fourth thread discusses the building of domain independent models, where the data to be run on is not part of the training set and does not fall into the same domain. The final thread discusses domain dependent but unsupervised models. These models are of use when data for a new domain is available but it has no labeling.

m4 consists of the following parts

- domain independent token splitter
- NSW tags predicted by a CART tree trained on the NSW labeled tags in devtrain. The features used include letter language model based features for alphabetic NSWs.

- Tokens classified as EXPN are expanded using a WFST, build from domain dependent expansions, that produce a list of potential expansions.
- A language model trained on domain depenedent labeled data which chooses between the different expansions.

The results over the domains are as follows

	m4	
	TER	WER
NANTC	0.39	0.82
classifieds	7.00	9.71
pc110	3.66	9.25
RFR	0.94	2.07

Next, to investigate how important each component is we removed parts to see how this affects results.

m4 as above

m4.nolm as in m4 but we take the most probable EXPN expansion and does not use the language model to choose between options.

m4.noef as in m4 but tag prediction tree don't use any features dependent on the letter language model.

m4.noeflm as in m4, with no letter language model features, most probable exansion and no language model.

	m4		m4.nolm		m4.noef		m4.noeflm	
	TER	WER	TER	WER	TER	WER	TER	WER
NANTC	0.39	0.82	0.39	0.81	0.38	0.78	0.38	0.78
classifieds	7.00	9.71	6.82	9.70	7.55	10.39	7.41	10.42
pc110	3.66	9.25	3.63	9.25	3.93	10.90	3.90	10.90
RFR	0.94	2.07	0.93	2.06	0.88	2.07	0.88	2.07

As one can see, always using the most probable expansion for EXPNs without using a langauge model is overall better than letting the language model choose. Looking at actual errors it is not the case that using the language model always makes correct choices wrong, it does get some tokens right that are wrong without it, but unfortunately it also makes some right things

wrong. There are, in fact, only a few places where expansions have valid alternatives and in those cases its not clear how a general n-gram language model would help disambiguate them, so perhaps this result should not be surprising. But in removing the possibility of choice will guarantee that the less probable expansions will never get selected, which seems deficient.

When the letter language model features are deleted they certainly make both the classifieds and pc110 domains worse. These two domains have a much higher occurrence of domain specific acronyms and abbreviations and it is probably those that are better predicted with the extra features. The other two domains, NANTC and RFR seem to suffer from using these extra features.

The next set of experiments show what happens when we give the model truth through an oracle. This can only be realistically done in two places.

m4.nosplt In this case we use the original split token as labeled rather than rely on our own splitter.

m4.nost Here we use the original split and the hand labeled tags. Thus the only part left is the expansion expander at the language model.

The third possible test is to also use the correct expansions but in that case the language model would have no choices and therefore the answer would be truth. Again m4 is shown for comparison

	m4		m4.nosplt		m4.nost	
	TER	WER	TER	WER	TER	WER
NANTC	0.39	0.82	0.20	0.44	0.03	0.06
classifieds	7.00	9.71	5.40	6.35	3.15	4.24
pc110	3.66	9.25	2.58	4.61	0.49	0.75
RFR	0.94	2.07	0.59	1.11	0.16	0.24

This shows that there is a difference in our splitter compared to the original hand label splits that is affecting performance. The CART predictors were trained from the original hand split tokens and may benefit if they were trained on the type of tags that come from our own splitter, though this could require re-labeling. In many cases we feel our splitter is more reasonable that the human specified form (or at least more consistent).

When we give the model actual tags it naturally does better, but at the same time the performance shows there is still a substantial part of the problem left, particularly in classifieds, which is the domain with the most abbreviations.

The fourth line of experiments were to find out how well we can build a domain independent model. At first we considered building models based on three of our labeled domains then testing on a third, but our initial experiments (based on the performance on the CART trees build in this manner) was that the NANTC domain is about as generic as we can get and it performs as well on the other domains as a combined model probably would.

For comparison we include Festival's results and m4 results before giving m4.domin which is m4 with all domain dependent parts fixed to use NANTC based models. m4.dominE again uses NANTC domains models but for EXPNs it uses a list of the most frequent abbreviations taken from the data, i.e. it does use domain dependent data. The rationale for this test is to find out what would happen in you at least created a new list of domain dependent expansions for the model, which is considered much less work than labeling data

	festival		m4		m4.domin		m4.dominE	
	TER	WER	TER	WER	TER	WER	TER	WER
NANTC	1.00	1.38	0.39	0.82	0.39	0.82	0.39	0.82
classifieds	30.09	33.48	7.00	9.71	25.20	29.11	19.69	21.18
pc110	14.37	32.62	3.66	9.25	12.35	18.69	12.09	18.07
RFR	6.28	16.19	0.94	2.07	2.71	4.66	2.32	4.14

Although the results for our NANTC model compare favorably with Festival's they are still some what poor. In the classifieds domain we are still getting around 1 in 4 words wrong. The addition of a list of known abbreviations in the domain helps but it probably still isn't useful enough to consider using the output for any real task.

The final set of experiments that we report here are on building domain dependent models on unlabeled data. That is we assume there is some example data from the domain but don't expect it to be labeled. We consider scenario a likely one in the case of text condition building language models. A database of text will be provided so it may be analysized to build models automatically. So far we have only done this for the classified domain. The steps involved are

- Use the NANTC model to generate tags for the new domain, dumping the features for each new NSW with its predicted tag.
- Instead of taking the NANTC prediction as correct, for all alphabetic tokens we take the best based on the letter language model features

built from unsupervised data, as described above. All other tokens we just use the NANTC guess.

- We then build a prediction tree based on these new labeled features in our unknown domain.
- With this new classifier we expand the data this time generating words where we can and the literal EXPN for EXPN tokens as we don't yet know what it expands to.
- A WFST built to automatically expand abbreviations, as described above, is then used to find the expansions.
- We then re-expand the the data giving the most probable expansion.
- A language model is then build on the expanded data
- The final model runs by predicting a number of possible expansions and the language model is used to decide between them

The NANTC tree on classifieds for NSW tags gives 67.44% correct (on devtest). When devtrain is marked with the NANTC prediction and alphabetic tokens are modified to have the tag best predicted by the letter language model, a tree built from that data give 86.72% tag correct on devtest.

The results here first show the fully domain dependent supervised (trained on labeled data) model from m4. The second is us1.nolm, is the unseupervised domain depened model taking the mode probable expansion. us1.lm allows multiple expansions and uses a domain dependent language model. us2 uses an explicit list of expansions which assumes that a labeler has given the expansion, though what is an abbreviation has been been automatically detected in the same way as in us1. This third test is to show the results if at least some time is taken to explicitly specify expansions.

	TER	WER
m4	7.00	9.71
us1.nolm	12.64	13.50
us1.lm	12.50	13.40
us2	10.58	13.51

The results are reasonable and significantly better than the domain independent models though we still are requiring no labeling of the data. The most common mistake this model makes is failing to identify an easily pronounceable word as an abbreviation (e.g. *kit* for *kitchen*). This type of mistake however doesn't detract from understanding in spoken output.

10 Discussion

The work reported here represents, we believe, a significant advance in the state of the art in text normalization. We have provided not only *supervised* models that perform well on four distinct domains, but have also provided methods that allow one to build text normalizers for new domains given that one has raw text from that domain.

Of equal importance is the fact that we have provided performance measures as a whole for the various text-normalization approaches on the different domains. This contrasts with the more normal practice of reporting error rates (if at all) on selected text-normalization problems, such as the problem of distinguishing ordinary numbers from dates. Such microscopic evaluations *are* important of course: it is certainly useful to have finer-grained information on errors. However in the absence of overall statistics it is hard to put such finer-grained measures in context.

As we said in the introduction, this work represents not an end, but a beginning. There is substantially more work to be done in the area of text normalization. Towards this end the tools and the databases created for this project will be publicly available, and this will hopefully encourage others to improve upon the work we have done here.

One weak link in the work we have done here is language modeling. The trigram language model was critical only in the generation of unsupervised abbreviation lists: in the final runs on the test data, language modeling was not of significant help. This result may be surprising given that we know that some abbreviations are ambiguous, and their ambiguity is typically resolvable from the immediate context. On the other hand, it is likely that a trigram model based on *words* is too impoverished to provide much help in resolving the many cases (e.g. *St* for *Saint* versus *Street*) that are better be cast in terms of features of the context. This suggests the use of "direct models", such as decision lists, decision trees or maximum entropy methods, something that we did not have time to adequately investigate in the context of this workshop.

Finally, our work has focused exclusively on English, and one important area to investigate is the application of these and related techniques to languages besides English. We expect that many of the techniques would carry over, mutatis mutandis, to other Western languages (broadly construed). Complexities will arise in languages like Russian where even seemingly innocuous abbreviations like *kg* can be read in various ways depending upon the case, number, gender and other properties of words in the context; the best approaches to handly such cases currently involve hand-constructed

rules (Sproat, 1997). Some languages, such as Chinese present additional problems, including the lack of space delimiters for words (see, e.g., (Sproat et al., 1996)); on the other hand there seem to be an almost total lack of abbreviations, in the technical sense used here in Chinese (see, e.g., (Sproat, 2000)).

11 Acknowledgments

We would like to thank Kevin Walker, Chris Cieri and Alexandra Canavan of the Linguistic Data Consortium for their work on obtaining our Classified Ad data. We also thank Michael Riley and David Yarowsky for useful discussion.

REFERENCES

Allen, Jonathan, M. Sharon Hunnicutt, and Dennis Klatt. 1987. *From Text to Speech: the MITalk System.* Cambridge University Press, Cambridge.

Bell, T. C., J. G. Cleary, and I. Witten. 1990. *Text Compression.* Prentice Hall, Englewood Cliffs.

Bikel, D, S. Miller, Richard Schwartz, and Ralph Weischedel. 1997. Nymble: a high-performance learning name-finder. In *Applied Natural Language Processing Conference*, pages 194–201.

Black, Alan, Paul Taylor, and Richard Caley. 1998. The Festival speech synthesis system. http://www.cstr.ed.ac.uk/projects/festival.html.

Breiman, Leo, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. 1984. *Classification and Regression Trees.* Wadsworth & Brooks, Pacific Grove CA.

Brown, Peter, John Cocke, Stephen Della Pietra, Vincent Della Pietra, Frederick Jelinek, John Lafferty, Robert Mercer, and Paul Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, June.

Brown, Peter, S. A. Della Pietra, V. J. Della Pietra, J. C. Lai, and R. L. Mercer. 1992. An estimate of the upper bound of the entropy of English. *Computational Linguistics*, 18:31–40.

Brown, Peter, Jennifer Lai, and Robert Mercer. 1991. Aligning sentences in parallel corpora. In *Proceedings 29th Annual Meeting of the Association for Computational Linguistics*, pages 169–176, Berkeley, CA, June.

Cannon, Garland. 1989. Abbreviations and acronyms in English word-formation. *American Speech*, 64:99–127.

Carletta, Jean, Amy Isard, Stephen Isard, Jacqueline Kowtko, Gwyneth Doherty-Sneddon, and Anne Anderson. 1997. The reliability of a dialogue structure coding scheme. *Computational Linguistics*, 23(1):13–31.

Chen, Stanley F. and Joshua Goodman. 1998. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University.

Flammia, G. 1998. *Discourse segmentation of spoken dialogue: an empirical approach.* Ph.D. thesis, MIT.

Golding, Andrew and Dan Roth. 1999. A Winnow-based approach to spelling correction. *Machine Learning.*

Hirschberg, Julia and Christine Nakatani. 1996. A prosodic analysis of discourse segmentation in direction-giving monologues. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics.*

Iyer, Rukmini and Mari Ostendorf. 1997. Transforming out-of-domain estimates to improve in-domain language models. In *Proceedings of Eurospeech*, volume 4, pages 1975–1978.

Jelinek, Frederick. 1997. *Statistical Methods for Speech Recognition.* MIT Press, Cambridge.

Jurafsky, Dan, R. Bates, N. Coccaro, R. Martin, M. Meteer, K. Ries, Elizabeth Shriberg, Andreas Stolcke, Paul Taylor, , and C. Van Ess-Dykema. 1997. Switchboard discourse language modeling project final report. Summer Research Workshop Technical Reports 30, Johns Hopkins University, Baltimore, MD.

Kneser, Reinhard and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 181–184.

Linguistic Data Consortium. 1998. 1996 CSR Hub-4 language model. http://morph.ldc.upenn.edu/Catalog/LDC98T31.html.

Römer, Jürgen. 1994. Abkürzungen. In Hugo Steger and Herbert Ernst Wiegand, editors, *Schrift und Schriftlichkeit/Writing and its Use*, volume 2. Walter de Gruyter, Berlin, chapter 135, pages 1506–1515.

Sproat, Richard, editor. 1997. *Multilingual Text to Speech Synthesis: The Bell Labs Approach.* Kluwer Academic Publishers, Boston, MA.

Sproat, Richard, editor. 2000. *A Computational Theory of Writing Systems.* Cambridge University Press, Stanford, CA.

Sproat, Richard and Michael Riley. 1996. Compilation of weighted finite-state transducers from decision trees. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 215–222, Santa Cruz, CA. Association for Computational Linguistics.

Sproat, Richard, Chilin Shih, William Gale, and Nancy Chang. 1996. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3).

Taghva, Kazem and Jeff Gilbreth. 1995. Recognizing acronyms and their definitions. Technical Report 95–03, Information Science Research Institute, University of Nevada, Las Vegas, June.

Yarowsky, David. 1996. Homograph disambiguation in text-to-speech synthesis. In Jan van Santen, Richard Sproat, Joseph Olive, and Julia Hirschberg, editors, *Progress in Speech Synthesis*. Springer, New York, pages 157–172.

A Appendix 1: Labeling Guide for NSWs

A.1 Background

Although we may think text is made up of words, actually there are often tokens within text that are not simply "words." For example, numbers, abbreviations etc are surprisingly common. What is more the pronunciation of these tokens is not always trivial. Consider the digit string *1985*. This will be pronounced differently depending on its context. As a year it is *nineteen eighty-five*, while as a number as in *1985 pages* it is *one thousand nine hundred (and) eight-five*, while as a telephone number it could be *one nine eight five*. Abbreviations are also common: e.g. *20GB 50Mhz*. Depending on the type of text the percentage of *non-standard words* with non-trivial pronunciations in the text can be as much as 50%. As part of a project to investigate the relationship between written text and the its pronunciation we wish to label large amounts of text from at least four different domains. These are: news stories from press wires; some USENET/email data; classified ads; and IRC (internet relay chat). The project is to design statistical models to predict the pronuncation of such words for both speech synthesis and for building language models in speech recognition. This project will run at Johns Hopkins University from mid July to the end of August this year.

A.2 The labeling task

The labeling task itself involves looking at a number of words within a short context (three words at either side) and identifying one of around twenty possible labels for that non-standard word. To aid this, the presentation method only presents tokens which might be NSWs though the heuristic for finding them is slight over general such that some identified NSWs are actually just words that aren't in our lexicon. Sometimes (more often in some text types) the token must also be split to identify its pronunciation of its subparts: e.g. *WinNT* consists of an abbreviation *Win* for *Windows* and the part *NT* to be pronounced as a letter sequence.

A.3 A Simple Example

For example the labeling tool, actually a special mode in the Emacs editor, presents each token on a new line surrounded by its context. A guess at the label is given at the start and the labeler must either accept the guess or provide an alternative.

```
      NUM              for Bosnia by Oct * 15 * he would go to 109
      NUM              no later than Nov * 15 * The United States along $
      NUM              begin the sale of * 12 * million barrels of oil  $
      ASWD     possibility of doing this * multilaterally * 0 0 0 0 358
      LSEQ      The Washington Post says * U.S * relations with its alli$
      ASWD   Rosenblatt Stadium in Omaha * Neb * they have never seen
```

The first two NSWs are not simple numbers but ordinals, since they are dates and hence must be labeled NORD. The third example is a simple number. The fourth, *multilaterally*, is a standard word but because it is not in our lexicon it appears as a potential NSW; however it is guessed to actually be a regular word (ASWD). The next example is a letter sequence. Finally, the last example *Neb* is an abbreviation for *Nebraska* and hence should be marked as EXPN; thus the guessed tag (ASWD) is wrong. Thus after labeling, the above will look like.

```
NORD  NUM              for Bosnia by Oct * 15 * he would go to 109
NORD  NUM              no later than Nov * 15 * The United States along $
NUM   NUM              begin the sale of * 12 * million barrels of oil  $
ASWD  ASWD     possibility of doing this * multilaterally * 0 0 0 0 358
LSEQ  LSEQ      The Washington Post says * U.S * relations with its alli$
EXPN  ASWD   Rosenblatt Stadium in Omaha * Neb * they have never seen
```

A.4 A More Complex Example

Some NSWs have internal structure. such as *PCCard*, *64MB*, *LviewPro*, and these need to be identified more fully. For such NSWs you may select the split option (the character '/') and the labeler will prompt you with the token, you insert spaces at the appropriate boundaries, then HIT return and the sub parts can be labeled. For example:

```
      ASWD         down 110 Preferably a * 4MB * unit with no HD        $
```

requires splitting into *4* and *MB* giving

```
SPLT  ASWD         down 110 Preferably a * 4MB * unit with no HD        $
        4
        MB
```

which are labeled as NUM and EXPN. For every token that is labeled EXPN only one expansion should exist. Such examples are *mg*, *kg*, *N.Y.*, *Capt.* Note the EXPN label may sometimes be used for things which could also be split: *D-Mass*, *R-TX* (identifying party and state of US senators and representatives).

A.5 Tagging Chart

The labeler runs as a special mode in the Emacs editor, single key strokes add labels to the NSW on the current line.

Key	Label	Explanation	Example
m	MSPL	misspelled word	geogaphy
e	ESPN	abbreviation/contraction	adv, N.Y, mph
l	LSEQ	letter sequence	CIA, D.C, CDs
a	ASWD	read as word	CAT, proper names
f	FNSP	funny spelling	sllloooww, sh*t
x	NONE	token should be ignored	ascii art, formating junk
s	SLNT	not pronounced	punctuation in compounds
n	NUM	number (cardinal)	12, 45, 1/2
o	NORD	number (ordinal)	May 7, 3rd, Bill Gates III
t	NTEL	telephone (or part of)	212 555-4523
d	NDIG	number as digits	Room 101,
i	NIDE	indentifier	747, 386, 8086
,	NADDR	number as street address	5000 Pennsylvania, 4523 Forbes
z	NZIP	zip code or PO Box	91020
c	NTIME	a (compound) time	3.20, 11:45
C	NDATE	a (compound) date	2/2/99, 14/03/87 (or US) 03/14/87
u	URL	url/pathname	http://slashdot.org /usr/local
y	NYER	year(s)	1998 80s 1900s 2003
$	MONEY	money (US or otherwise)	$3.45, HK$300, Y20,000
b	BMONY	money tr/m/billions	$3.45 billion
S	SCORE	scores, ranges (not dates)	NN-NN
%	PRCT	percentage	75% 3.4%
.	OTHER	unknown (use sparingly)	
SPACE		Selects the guessed token	
r		prompts for user specified token	
/		prompt for split of token	

Note that labeling should (primarily) identify how you would pronounce the token. Note if the guess is ROM, a roman numeral, identify its uses as a NUM (as in World War II) or NORD as in Louis XIV or Louis the XIV. For unusual abbreviations, or ones where the token itself might be ambiguous it is neccesary put the expansion in the label itself. All labels starting with lower case letters are treated as in-line expansions. This seems particular useful with split NSWs.

A.6 How run the labeler

The labeler is an special mode in Emacs. To run it you need the script *toklabel* and the Emacs Lisp file *toklab.el*. Download these files and save them in a new directory. Edit your copy of toklabel so the value of *TOKLABDIR* is the name of the directory that contains both the *toklabel* script and the *toklab.el* file. You will be given files like *example.feats* which after labeling should be saved to *example.done*. You do this by downloading the file (for example into the same directory as the scripts) and type

```
./toklabel example.feats
```

This presents a screen (you may wish to make the window wider) with the tokens in context. Pressing any of the single characters described above will add the appropriate token in column 1. The space key will select the default. For some texts the default will often be right, for some tokens the default will almost always be right, but note for the occasional weird forms, for example numbers that look like years, letter sequences that are really words etc. The list of labels and examples may be obtained in Emacs itself with the command *C-h m* (described-mode). When you type something you didn't mean to and things become *strange* you can use Emacs' undo feature, available from the Edit menu and as *C-_* (that's control underscore). Also you can override the special characters to type your own by preceding them with *C-q* (though the r key will usually be sufficient).

B Appendix 2: Guide for the Manual Evaluation of Text Normalization Performance

Evaluating text normalization quality involves editing a simple text file. A sample excerpt of such a file is:

```
...en boot without *your config.sys     (&lt;F5>       d...
...en boot without  your config dot sys ( < F. five >. d...
1

...ngers >of   the *747               airplane,   may be...
...ngers >. of the  seven forty seven airplane ,  may be...
1

...   >>    >> How *did      we determine there is a USB...
...>. >. >. >. How  D. I. D. we determine there is a U. ...
0

     <<<I  am Type *II  now... PC110              strand...
     <<<I. am Type  two now    P. C. one one zero strand...
1

... to look at the *IBM      as you can compare the Type...
... to look at the  I. B. M. as you can compare the Type...
1

...   off, pop the *12,      and then put it back.>>>
...ro off  pop the  twelve , and then put it back .>>>
1
```

Text comes in pairs of lines, the first line being the original raw text and the second line being normalized text. For each pair, the labeler is supposed to evaluate the correctness of a single space-separated token. The particular token to evaluate is marked with an asterisk to the left and always starts in the same column. The number below the pair of lines is the judgement of correctness of that example; it is set to 1 originally, which denotes correct. The labeler's job is to edit that character to be *1* for correct, *0* for incorrect, and *m* for *misalignment*. To be rated as correct, *all* tokens that should be generated by that token must be correct, *e.g.*, `747` must generate all three tokens `seven forty seven` correctly, and in the last example both the `twelve` and the comma must be present.

The *misalignment* judgement means that it is not possible to make a judgement from the context presented, which may be because the automatic alignment between pairs is wrong or because the generated tokens run off the screen. For example, the following should be labeled as a misalignment

```
...(At  *http://www.cadex.com/cfm/index.cfm                ...
...( At  h. t. t. p. colon slash slash w. w. w. dot cadex ...
```

because we cannot tell if the whole token is correct without more context to the right. However, the following example should be marked as correct

```
...mAh,  *they're $35.00                          (sometimes)  ...
...mAh ,          they're thirty five dollars ( sometimes )...
```

because even though the automatic alignment is not completely accurate, there is adequate context to judge the correctness of the token.

The `<<<` symbol signals the beginning of a paragraph, `>>>` signals the end of a paragraph, and `...` denotes that the paragraph continues outside of the context.

To decide whether a space-separated token is correct, use the following guidelines:

- As mentioned above, all tokens in the normalized text corresponding to that raw token must be correct to be considered correct.
- As mentioned above, do not count minor alignment errors as wrong; *i.e.*, if the normalized token corresponding to a raw token does not occur directly underneath, that is not cause to mark something incorrect.
- **Grading punctuation:** punctuation does not have to be gotten exactly correct; the only thing that is important is whether the *presence* of punctuation is predicted correctly. For example, expanding a comma as a semi-colon is fine, or expanding a left-quote as a right-quote is fine. However, having punctuation where there should be none or not having punctuation where there should be some is an error. For example, expanding "`The US came`" as "`The U. S. . came`" is an error, as is expanding "`in the US. In other news`" as "`in the U. S. In other news`". Furthermore:
  - It is OK to substitute a single punctuation mark for multiple consecutive marks, or vice versa. For example, expanding "`- - - -`" as "`-`" is fine.

  - Extraneous or missing punctuation at the beginning or end of a *paragraph* (not *sentence*) is fine.
  - These rules are motivated by the principle that in language modeling or speech synthesis, we need punctuation to identify things such as segment boundaries or locations to place pauses, but the identity of the punctuation is less important. In addition, segment boundaries or pauses at the beginning or end of a paragraph are clear.

- **Rule of correct intention:** if it is clear that the given text normalizer "understood" the given token, but just didn't output exactly the correct tokens according to our grading specifications because it didn't know what the grading guidelines were beforehand, then the token should be judged correct. Examples:

  - Expanding `IBM` as `I B M` is technically incorrect because it should really be `I. B. M.`, but should be marked as correct.
  - Expanding `http://yahoo.com` as `h. t. t. p. : slash slash yahoo dot com` is technically incorrect because the colon as written would be treated as punctuation instead of read as the word *colon*, but this case should be marked correct.
  - Expanding `call (808)425-1345` as `call ( eight oh eight ) four two five - one three four five` is technically incorrect because there probably shouldn't be punctuation after *call*, but this should marked as correct.
  - Expanding `----------` as `line of hyphens` is fine.

- **Principle of acceptability:** any expansion that is acceptable *either* for language modeling conditioning or speech synthesis is acceptable, and *only* such expansions.

  - There may be many ways to read a token. For example, `105 dogs` could be normalized as `one hundred five dogs` or `a hundred and five dogs`.
  - If you do not know for sure that an expansion is incorrect, mark it as correct. For example, if you see `Ex Michael 222-2178`, expanding `Ex` as itself is acceptable (unless you know what it really is) even though it may very well be some sort of abbreviation.
  - Expanding "garbage" is not acceptable. For example, the string `charset="iso-8859-1"` found in E-mail headers should not be

pronounced in speech synthesis or be in language model training text. Such garbage should expand to nothing.

- *Gritty details:*
  - Grading hyphenated words: all hyphenated words should be split into space-separated words, unless the split words are not words by themselves (or at least not the intended words). For example, `two-sided` should be expanded as `two sided`, while `re-jumpering` should be expanded as is.
  - The tokens `Mr.`, `Ms.`, and `Mrs.` may be left as is and considered correct (though expansion to full word is also OK).
  - Mismatches in capitalization are not errors.
  - The tokens `I` and `I.` (and `A` and `A.`) should be correctly differentiated. The former token is the word *I*, and the latter is used to spell out letter sequences containing the letter *I*. This is not an issue for the other letters since they are not ambiguous. Also, the rule of correct intention applies when expanding acronyms, see above.
  - Misspellings are errors unless corrected in the expansion.
  - Concatenations of words are errors unless separated, *e.g.*, `garrisonsquare` should be expanded `garrison square`.
  - Plural and singular forms must be distinguished correctly if spelled differently, *e.g.*, `3g` should be expanded as `three grams`, and the expansion `three gram` is incorrect.
  - Plural and possessive forms must be distinguished correctly if spelled differently, *e.g.*, `the wtrfrnts. best area` should be expanded as `the waterfront's best area`; if the apostrophe is excluded, it is an error.
  - All punctuation characters (*e.g.*, `&`) should be assumed to be punctuation and not read as a word, except where the rule of correct intention applies.
  - For strings that should be read verbatim, such as computer commands (*e.g.*, `format c: /ix=5`) or WWW addresses, every (nonredundant) character must be read correctly to be correct (*e.g.*, `format c. colon slash i. x. equals five` or even better, `format space c. colon space slash i. x. equals five`).

- Do not punt on any token, even though we may not be considering processing all tokens in our own system (*e.g.*, expanding WWW addresses).
- There may be ambiguous cases in assigning an error to a raw token. For example, if the text `three eggs two cups milk one apple` is expanded as is, there are two errors because of missing punctuation between `eggs` and `two`, and `milk` and `one`. It is ambiguous which of these raw tokens to assign the errors to. Use the following guideless to resolve the ambiguity:
  * For money, getting the word `dollars` correct is the responsibility of the raw token containing the `$`, unless it is absent in which case it is the responsibility of the number.
  * Otherwise, resolve ambiguity by choosing the leftmost raw token of the possibilities to be responsible for the error.

C Appendix 3: Lattice Format

This section describes the file format that we use to represent the lattice of possible expansions for utterances. Lattices for multiple utterances can be stored in the same file.

At the beginning of the lattice for each utterance, the following line must be included:

```
FSM-ID: <unique-ID>
```

The ID is arbitrary; it is used to guide later processing. At the end of the lattice for each utterance, the following line must be used:

```
END
```

To list an alternative for the nth word in a sentence, use the line:

```
<n-1>  <n>  <raw-word>  [<expanded-word>]  [<cost>]  [<tag>]
```

The fields should be separated by exactly a single tab. Multiple words in `<expanded-word>` should be separated by spaces. The last three fields are optional. If `<expanded-word>` is missing or empty, it is assumed to be the same as `<raw-word>`. To specify an empty `<expanded word>`, use the token `<sil>`. `<cost>` should be a log probability, base 10 (and will thus usually be negative). `<tag>` is the NSW tag associated with the raw token, e.g., `ASWD`.

Alternatives must be listed for words in the order they occur in the sentence, i.e., alternatives for the $(n+1)$st word must follow those of the nth word.

The following is a sample file:

```
FSM-ID: a034c1
0       1       NATO    NATO    -0.1    ASWD
0       1       NATO    N. A. T. O.     -0.4    LSEQ
1       2       LIVES   LIVES   0       ASWD
2       3       ###     <sil>   0       SLNT
3       4       ON
4       5       AND
5       6       ON
END
FSM-ID: a034c2
0       1       NATO    NATO    -0.1    ASWD
0       1       NATO    N. A. T. O.     -0.4    LSEQ
1       2       LIVES   LIVES   0       ASWD
```

```
2       3       ###     <sil>   0       SLNT
3       4       ###     <sil>
END
```