Computer Speech and Language (2001) **15**, 287–333
doi:10.1006/csla.2001.0169
Available online at http://www.idealibrary.com on IDEAL®

Normalization of non-standard words

Richard Sproat,†∗ Alan W. Black,‡ Stanley Chen,§ Shankar Kumar,¶ Mari Ostendorf‖ and Christopher Richards∗∗

†AT&T Labs–Research, Florham Park, NJ, U.S.A., ‡Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA, U.S.A., §IBM T. J. Watson Research Center, Yorktown Heights, NY, U.S.A., ¶Electrical and Computer Engineering Dept., Johns Hopkins University, Baltimore, MD, U.S.A., ‖Electrical Engineering Dept., University of Washington, Seattle, WA, U.S.A., ∗∗Department of Computer Science, Princeton University, Princeton, NJ, U.S.A.

Abstract

In addition to ordinary words and names, real text contains non-standard "words" (NSWs), including numbers, abbreviations, dates, currency amounts and acronyms. Typically, one cannot find NSWs in a dictionary, nor can one find their pronunciation by an application of ordinary "letter-to-sound" rules. Non-standard words also have a greater propensity than ordinary words to be ambiguous with respect to their interpretation or pronunciation. In many applications, it is desirable to "normalize" text by replacing the NSWs with the contextually appropriate ordinary word or sequence of words. Typical technology for text normalization 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 the work reported here is to take some initial steps towards addressing deficiencies in previous approaches to text normalization.

We developed a taxonomy of NSWs 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 can lead to better results than have been obtained by the ad hoc treatments that have typically been used in the past. For abbreviation expansion in particular, we investigated both supervised and unsupervised approaches. We report results in 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.

∗Author for correspondence: AT&T Labs–Research, Shannon Laboratory, Room B207, 180 Park Avenue, PO Box 971, Florham Park, NJ 07932-0000, U.S.A. E-mail: rws@research.att.com

0885–2308/01/030287 + 47 $35.00/0

1. Introduction

All areas of language and speech technology must deal, in one way or another, with real text. In some cases the dependency is direct: for instance, machine translation, topic detection or text-to-speech systems start with text as their input. In other cases the dependency is indirect: automatic speech recognizers usually depend on language models that are trained on text. In the ideal world, text would be "clean" in the sense that it would consist solely of fully spelled words and names, and furthermore these spellings would be unambiguous, so that it would be straightforward to reconstruct from the written form which exact word was intended. Unfortunately, written language deviates from this ideal in two important ways. First, in most if not all languages there is ambiguity even for ordinary words: if we write *bass*, it is up to you as the reader to figure out from the context whether we meant *bass* the fish, or *bass* the musical instrument. Second, in most genres of text, many things one finds in the text are not ordinary words. These include: numbers and digit sequences of various kinds; acronyms and letter sequences in all capitals; mixed case words (*WinNT*, *SunOS*); abbreviations; Roman numerals; universal resource locators (URLs) and e-mail addresses. Such "non-standard words"—NSWs—as we shall henceforth call them, are the topic of this paper.

NSWs are different from standard words in a number of important respects. First of all, the rules for pronouncing NSWs are mostly very different from the rules for pronouncing ordinary words. For numbers, for example, one typically needs a specialized module that knows how to expand digit sequences into number names, spelled as ordinary words. For abbreviations such as *Pvt* (*Private*) one needs, in effect, to recover the missing letters and then pronounce the resulting word. Secondly, most NSWs will not be found in dictionaries, so that one cannot expect simply to look up their properties in a list; nor can one derive them morphologically from words that are in a dictionary. What is worse is that even when one does find a dictionary that includes such items—for example, a dictionary of common abbreviations—the entries can often be misleading due to the third property, namely that NSWs have a much higher propensity than ordinary words to be ambiguous. This ambiguity often affects not only what the NSWs denote, but also how they are read. Thus, depending upon the context in which it occurs, the correct reading of *IV* could be *four*, *fourth* or *I. V.* (for *intravenous*); *IRA* could be read as *I.R.A.* (if it denotes the *Irish Republican Army*) or else, for many speakers, *Ira* (if it denotes an *Individual Retirement Account*). *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.

The particular problems these properties present for speech and language systems depend, of course, upon the nature of the system. For text-to-speech (TTS) systems, the primary consideration is how the NSW is pronounced. This is true also for automatic speech recognition (ASR) since ASR systems depend upon language models trained on text, and these models should reflect what people say, not merely what tokens exist in the text. (Note that as techniques for using out-of-domain language model training data improve, methods for extracting from text what people would say—not merely what is in the text—will be increasingly important for utilizing the vast amounts of on-line text resources.) For topic detection, machine translation or information extraction systems, the most important consideration will be what the NSW denotes: in the given context is *1750* a number or a date? Does *IRA* mean *Irish Republican Army* or *Individual Retirement Account*? Note, in particular, that in information extraction, many important pieces of information that one might want to detect, such as dates, currency amounts or organization names will often or even typically be written as

NSWs; it is presumably worth knowing, if one is looking for the organization name IRA, that the particular instance one is looking at in fact denotes the IRA.

For all of these reasons, text normalization—or the conversion of NSWs into standard words—is an important problem. It is also quite a complex problem, due to the range of different kinds of NSWs, the special processing required for each case, and the propensity for ambiguity among NSWs as a class. Unfortunately, text normalization is not a problem that has received a great deal of attention, and approaches to it have been mostly ad hoc: to put the issue somewhat bluntly, text normalization seems to be commonly viewed as a messy chore. In the TTS literature, text normalization is often presented (if at all) in a cursory chapter or paragraph, before going on to address the more interesting issues of unit selection, or intonational modeling. In ASR the issue is rarely discussed at all: text normalization has to be addressed, of course, in order to make use of real text in language model training, but it is typically handled via ad hoc scripts that are not considered worth writing about. One of the consequences of this lack of systematic attention is the fact that we do not even have a good taxonomy of NSWs, so that it may not be immediately clear to someone approaching the problem of text normalization for the first time what the range of problems to be addressed is.

The purpose of the work reported here is to address these deficiencies in previous approaches. We developed a taxonomy of NSWs 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 (in combination) including n-gram language models, decision trees and weighted finite-state transducers to the entire range of NSW types, and demonstrated that a systematic treatment of such cases can lead to better results than have been obtained by the spottier ad hoc treatments that have more typically been used in the past. We also employed more systematic procedures for evaluating performance than has heretofore generally been used in the text normalization literature.

The specific contributions of this research are:

- A proposal for a *taxonomy* of NSWs based on the examination of a diverse set of corpora and the NSWs contained therein.
- Hand-tagged corpora from several specific domains: North American News Text Corpus; real estate classified ads;
  `rec.food.recipes` newsgroup text; `pc110` newsgroup text. Some of these are publicly available under various conditions: see Black, Sproat and Chen (2000) for a current list of what is available.
- An implemented set of methods for dealing with the various classes of NSWs. 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 methods for designing domain-dependent abbreviation expansion modules: the supervised methods presume that one has a tagged corpus for the given domain; the unsupervised methods presume that all one has is raw text.

- A publicly available set of tools for text normalization that incorporate these methods. Again see Black *et al.* (2000) for what is currently available.

The remainder of this paper is organized as follows. In Section 2 we discuss previous approaches to the analysis of NSWs. Section 3 introduces a taxonomy of NSWs. Section 4 describes the corpora we used in our study. Section 5 gives some theoretical background relevant to the text-normalization problem. Section 6 outlines the general architecture of the text normalization system, and describes each of the components in detail, presenting a separate evaluation of each component where appropriate. We focus in that section mostly on *supervised* models of normalization—that is, models that are trained or developed assuming a corpus tagged with classes of NSWs and their expansions. Section 7 focuses on *unsupervised* methods for identifying and handling abbreviations (what we term EXPN), methods that can be applied to a completely untagged corpus to derive expansion models for abbreviations. While much of this discussion would seem to belong in Section 6, we chose to highlight it in a separate section since one of the more novel contributions of this work is the demonstration that one can derive tolerable abbreviation expansion models with minimal or no human annotation of a corpus of texts. Section 8 presents several evaluations of the system on the different corpora, under various training conditions. Finally, Section 9 concludes the paper with a summary and some general discussion.

2. Previous approaches

As we have noted, any system that deals with unrestricted text needs to be able to deal with non-standard words. In practice, though, most of the work that has dealt with text-normalization issues has been confined to three areas, namely text-to-speech synthesis, automatic speech recognition and text retrieval. We briefly consider the techniques applied in these domains in Sections 2.1–2.3 below. Cross-cutting all these domains (though to date only really applied in TTS) is the application of *sense disambiguation* techniques to the problem of *homograph resolution* for NSWs. This is discussed here (Section 2.4) as the only instance of a fairly principled corpus-based technique that has been applied to (a part of) the text normalization problem. Problems with these previous approaches are outlined in Section 2.5.

2.1. 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 & Klatt, 1987; Sproat, 1997; Black, Taylor & Caley, 1999). 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 parentheses 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-1980s 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.

2.2. 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, 1996). As is the case with 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 they are quite domain specific: they are specialized to work on business news text, and do not reliably work outside this domain. For instance, only about 3% of the abbreviations found in our classified ad corpus (Section 4) are found in the LDC tools abbreviation list.

2.3. Text retrieval applications

NSWs cause problems in text retrieval for the obvious reason that they can contribute to a loss in recall. To take a simple example, consider a search over a large text database for texts relating to the Dow Jones Industrial Average. If the user queries using such terms as *Dow Jones* then only texts that have that substring will be retrieved. In particular, if there is a text that only refers to the Dow Jones with the letter sequence *DJIA*, that text would not be retrieved.

Rowe and Laitinen (1995) describe a semiautomatic procedure for guessing the expansion of novel abbreviations in text. Their method depends upon dictionaries of known full words, and dictionaries of known abbreviations and their expansions. Novel abbreviations are dealt with by applying abbreviation rules to known full words in the text being considered. These abbreviation rules include deletion of vowels or truncation of the righthand portion of the word. This procedure will generate a set of candidate expansions for the abbreviation, which are then verified by a user.

The "generate-and-test" procedure and the restriction of candidate expansions to in-domain text is similar to the unsupervised method for abbreviation expansion that we describe below in Section 7, though it differs critically in that the method we describe makes use of n-gram language modeling, which to some degree automates the step of user verification in Rowe and Laitinen's method.

2.4. Sense-disambiguation techniques

Sense disambiguation techniques developed to handle ambiguous words like *crane* (a bird, vs. 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*; vs. *bass* "musical range", homophonous with *base*).

As we noted above, many NSWs are homographs, some cases being rather particular, and others 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.

2.5. Problems with previous approaches

Nearly 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 NSWs. For instance the following abbreviations for the term *landfill* occurred in a 1989 *Associated 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.

These examples cannot even remotely be considered to be "standard", and it is therefore unreasonable to expect that the designer of a text normalization system would have thought to add them to the list of known abbreviations.

In some domains, such as real estate classified ads, the set of novel examples that one will encounter is even richer. 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.

Over and above the limitations of predefining which NSWs will be handled, there is the more general problem that we do not have a clear idea of what types of NSWs 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).

3. A taxonomy of NSWs

After examining a variety of data from the corpora described in Section 4, we developed a taxonomy of non-standard words (NSWs), summarized in Table I, to cover the different types of non-standard words that we observed. The different categories were chosen to reflect

TABLE I. Taxonomy of non-standard words used in hand-tagging and in the text normalization models

	EXPN	abbreviation	*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 other)	*$3·45, HK$300, Y20,000, $200K*
	BMONEY	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)
	SLNT	not spoken,	word boundary or emphasis character:
M		word boundary	*M.bath, KENT*RLTY, _really_*
I	PUNC	not spoken,	non-standard punctuation: "***" in
S		phrase boundary	*$99,9K***Whites*, "…" in *DECIDE… Year*
C	FNSP	funny spelling	*sllooooow, sh*t*
	URL	url, pathname or email	*http://apj.co.uk, /usr/local, phj@tpt.com*
	NONE	should be ignored	ascii art, formatting junk

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 6.2 for 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 ordinary 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 cases (BMONEY).

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 that would 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 later 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 either should not be rendered, or where it is at least acceptable not to render them, 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.

Our taxonomy of NSW tags was principally designed before we took on the actual task of investigating automatic recognition of these distinctions. However the design did take into account the fact that we intended the distinctions to be detected automatically and that the distinctions, once made, would aid the rendering of these into standard words. As noted above, we added the PUNC tag relatively late in the process once it became clear that this tag would be useful. (Section 6.2 discusses the issue of actual detection of NSWs.) We defined the taxonomy both to represent the categories we observed and the ones we believed would most easily be automatically identified. However we admit that there are a number of borderline cases where an NSW may fall into more than one category: for instance *CDs* might be viewed as either an LSEQ or a SPLT.

4. Corpora

4.1. Domain descriptions

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 corpus includes data from several sources (New York Times, Wall Street Journal, Los Angeles Times, and two Reuters services). We

TABLE II. Size of different corpora and number of detected non-standard word tokens

Corpus	NANTC	classifieds	pc110	RFR
total # tokens	4·3 m	415 k	264 k	209 k
# NSWs	377 k	180 k	72 k	46 k
% NSW	8·8	43·4	27·3	22·0

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 especially 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 cleaning 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 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 breakdown as shown in Table II. The number of non-standard words given in the table is based on the automatic detection algorithm described in Section 6.2, which does miss some tokens, and does not separately count the sub-components of a SPLT token. The results and evaluation used in the remainder of this paper are based on these automatically detected NSWs as opposed to the actual NSWs in the data; discussion of the accuracy of our automatic detection can also be found in Section 6.2.

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 (every third file was added to the test set) and, although there are distinct sources for parts of the news and classifieds data (i.e. different newspapers), we did not take account of this. Therefore, files from each newspaper appeared in both train and test (except in a few cases). In addition, we held out 10% of the training set for development testing, which includes results reported in Section 6. The

TABLE III. Distribution of frequent alphabetic tags of NSWs in the four corpora. The first three rows are percentages among alphabetic tokens, while the last row is the overall percentage of alphabetics in all tokens

	Domains			
	NANTC	classifieds	pc110	RFR
ASWD	83·49	28·64	64·60	72·36
LSEQ	09·10	03·00	22·60	02·11
EXPN	07·41	68·36	12·80	25·53
All Alphas	54·52	38·65	40·31	31·75

TABLE IV. 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	00·70	00·51	00·27
NORD	09·37	03·37	04·45	00·11
NIDE	02·24	05·83	37·41	00·47
NTEL	01·25	25·92	01·32	00·02
NTIME	01·21	03·28	04·16	01·12
NZIP	00·22	00·29	00·17	00·04
NDATE	00·20	00·13	01·33	00·05
NDIG	00·16	00·00	02·16	00·01
NADDR	00·13	02·20	00·15	00 00
Total Nums	73 005	24 193	7818	18 1950

evaluation test set was only used to test overall system performance, and no error analysis was carried out on those results. The development test data was not folded back into the training set for designing the final evaluation models.

As should be clear from the partial distribution given in Tables III, IV, 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.

4.2. 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, and these were partially augmented (in particular, explicit closing paragraphs were added) in a fully automatic way and extra-

neous characters that interfere with such markup were quoted (that is, "`&`" and "`<`"). For all the other corpora, paragraph boundaries were marked at blank lines. 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 first breaking up the text into whitespace-separated tokens. Then, leading and trailing standard punctuation is removed and saved as features of the token. Tokens are identified as a possible NSW whenever they were not included in a standard on-line dictionary (98K words, CMU dictionary version 0·4 (CMU, 1998)) or were on a list of exceptions. Exceptions include words like *I*, which is in the dictionary but can sometimes be non-standard (the Roman numeral). This process is expedient for labeling, but not perfect, since some abbreviations overlap with standard words (e.g. "so" for *south*). Thus, common abbreviations are also included in the exception list, but there are still some non-standard words that are not labeled, as described in more detail later.

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 value 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.

4.3. Tagging conventions

The labeling task involved looking at an NSW token within a short context (four words on either side) and identifying one of the possible labels for that token. Note that labeling involved (primarily) identifying what words would correspond to the token if the text were read and indicating this symbolically via the tags. Labelers were instructed to type in the expansion explicitly only for tokens where the expansion was 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 due to a non-standard usage (e.g. not *bathroom* or *living room*, respectively).

To speed up the tagging task, labelers were presented only with candidate non-standard words that had been identified using a simple dictionary check. Standard words that were presented as candidate NSWs were simply labeled ASWD. These cases include words that were not in the specific dictionary used, e.g. names, places, and unusual ingredients in recipes. 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 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. The tagging instruction manual, including examples, is available online (Black *et al.*, 2000).

```
<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="with"> w/</WS><WS NSW="MONEY">$750K</WS></W>
down. ROBERT<W NSW="LSEQ">L.</W> TENNEY REALTY<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;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>
```

Figure 1. A sample of XML-markup of NSWs.

4.4. 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 the chance of labeler i assigning class k, i.e. the relative frequency of class k for that labeler), 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 (1998). The kappa statistic is widely used for evaluating labeler consistency (Hirschberg & 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 I, 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 $N = 3$ labelers, the agreement was $\kappa = 0{\cdot}81$. For $D = 622$ NSW tokens from the classified ads and $N = 9$ labelers, the agreement was $\kappa = 0{\cdot}84$. 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 we used, 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).

4.5. Generating "truth" for training and automatic evaluation

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 the expansions in the case of EXPNs). The truth was generated for each marked up XML file in 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. For synthesis, only one value is sufficient, but for language modeling we would ideally produce lattices with probabilities for choices. Since generation of alternatives requires some labeled data of this form and hand-marking alternatives was difficult and expensive, the use of lattices was left for future work.

The labeled data has two sources of noise: missed detection of NSWs in the initial processing of the data, and human labeling errors. Although a number of passes over the database have been made to correct systematic errors, we know errors still exist. The amount of noise in the data is small and would normally not be an issue, but our models have very low error rates compared to, say, speech recognition. Our error rates can be as small as 0·3%, and so even a labelling error rate of a few percent will affect our measurement of error rate.

Two methods have been used to estimate the amount of noise in the labeling. First, we estimated the missed detection rate for NSWs associated with the dictionary look-up algorithm by hand-marking all words in a small subset of each corpus. Missed detection rates ranged from 3% for the RFR corpus to 9% for the PC110 corpus. (A more detailed analysis is included in Section 8.4.) Second, we can count the number of detected NSWs that are problematic for the algorithmic expanders, i.e. when a given token with a given label does not match the expected form of that type, such as when a token marked NYER actually consists of a range of years 1990–95. Although we have been quite liberal in accepting such labelings,

TABLE V. Occurrence of detectably noisy tokens (DNTs) in different corpora

Corpus	NANTC	classifieds	pc110	RFR
total # tokens	4·3 m	415 k	264 k	209 k
# NSWs	377 k	180 k	72 k	46 k
% NSW	8·8	43·4	27·3	22·0
# of DNTs	1546	4583	922	134
% DNTs of tokens	0·03	0·96	0·32	0·06
% DNTs of NSWs	0·41	2·55	1·28	0·29

there are still a number of cases that the algorithmic expanders cannot deal with. For each database, we counted the number of tokens for which some part is detectably not expanded properly in generating "truth". We called the tokens for which our algorithmic expanders could not provide expansion, *detectably noisy tokens*, (DNT). These are a small subset of the identified NSWs including both the few NSWs in classes for which the expander could not expand the token plus all tokens in classes for which we did not provided any expander, namely, misspellings (MSPL), funny spelling (FNSP) and others (OTHERS). The DNTs were not hand corrected and thus they constitute noise in the data, but the percentage of DNTs is quite small relative to the missed detections.

This known amount of noise in our data does put an upper limit on the accuracy of our evaluation of the results.

Note that most, but not all, human labeling errors are detected in the algorithmic expansion. In addition, there is variability in the data associated with having multiple labelers forced to make a single choice in a case when there are different allowable alternatives. We wish to judge how good our generated words are with respect to human acceptability of the expansion, both due to noise in the data and where there exist valid alternatives. For this reason, some human ratings are included in assessing overall system performance, as will be described in Section 8.4.

5. Theoretical models

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 has NSWs marked and split where necessary. The output word sequence corresponds to the desired words to be spoken. Mathematically, the objective is:

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

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

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

where $\mathbf{t} = t_1, t_2, \ldots, t_m$ is the sequence of token tags. In Equation (3), we make the assumption that for a particular output word (or word sequence) 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. In other words,

while there may be multiple tags for a particular observation, there is only one tag that would correspond to the specific word (or word sequence) realization of that observation. This assumption is useful for simplifying the decoding problem, but it is also quite reasonable for most of the cases we are interested in.

The assumption of high probability mass on particular combinations of tags and word sequences also makes it possible to segment the word string into an m-length sequence $\mathbf{w} = \mathbf{w}_1, \mathbf{w}_2 \ldots \mathbf{w}_m$, where the sequence notation $\mathbf{w}_i$ is used to indicate that o_i may be expanded to multiple words. In other words, for cases where single tokens in the observation space (after splitting) map to multiple words in the output space (e.g. *wbfplc* maps to *wood burning fireplace*), we represent the multiple word sequence as a single "word". This alignment simplifies training for the models representing dependence between words and tags, and it need not complicate the language model, since the probability of a "multi-word" is simply the product of conditional probabilities of the individual words in the sequence.

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.

5.1. 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 $\mathbf{w}$ 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 $\mathbf{o}$ according to the sequence of channel models $p(\mathbf{t}|\mathbf{w})$ and $p(\mathbf{o}|\mathbf{t}, \mathbf{w})$. Mathematically, this corresponds to:

$$\hat{\mathbf{w}} \approx \operatorname*{argmax}_{\mathbf{w}} \max_{\mathbf{t}} p(\mathbf{w}, \mathbf{t}|\mathbf{o}) = \operatorname*{argmax}_{\mathbf{w}} \max_{\mathbf{t}} p(\mathbf{o}, \mathbf{t}, \mathbf{w}) \tag{4}$$

$$= \operatorname*{argmax}_{\mathbf{w}} \max_{\mathbf{t}} p(\mathbf{o}|\mathbf{t}, \mathbf{w})\, p(\mathbf{t}|\mathbf{w})\, p(\mathbf{w}) \tag{5}$$

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 smoothing (Kneser & Ney, 1995; Chen & Goodman, 1999).

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

$$p(\mathbf{t}|\mathbf{w}) = \prod_{i=1}^{m} p(t_i|\mathbf{w}_i). \tag{6}$$

Given the 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.

$$p(t_i|\mathbf{w}_i) \rightarrow p(t_i|c(\mathbf{w}_i)) \rightarrow p(t_i)$$

where the word class might be defined in terms of part-of-speech (or semantic class) label, word length, or word frequency in the target domain.

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 $\mathbf{w}_i$. 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:

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

and the key problem is to find $p(o_i|t_i, \mathbf{w}_i)$. For cases where the pair $(t_i, \mathbf{w}_i)$ are observed frequently, maximum likelihood estimates or simple rules are probably sufficient. For infrequent words, or word-tag pairs, we need to use general word properties, in which case decision trees will prove useful.

5.2. 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. While the work reported here does not implement a full direct model, it does use components and the approach is of interest for future work, so we describe it briefly below.

With the direct model, the overall objective (including the tags) is:

$$\hat{\mathbf{w}} \approx \operatorname*{argmax}_{\mathbf{w}} \max_{\mathbf{t}} p(\mathbf{w}, \mathbf{t}|\mathbf{o}) \tag{7}$$

$$= \operatorname*{argmax}_{\mathbf{w}} \max_{\mathbf{t}} p(\mathbf{w}|\mathbf{t}, \mathbf{o})\, p(\mathbf{t}|\mathbf{o}). \tag{8}$$

In this approach, 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, one can 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 a decision tree ($T[\cdot]$) to simplify the prediction space:

$$p(\mathbf{t}|\mathbf{o}) = \prod_{i=1}^{m} p(t_i|t_{i-1}, \mathbf{o}) = \prod_{i=1}^{m} p(t_i|T[t_{i-1}, \mathbf{o}]). \tag{9}$$

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 possible to separately explore tag predictors designed for subclasses of NSWs such as the numbers (NDIG, NUM, NORD, NIDE, etc.). Examples of some questions that may be useful in the decision tree include whether the token is mixed case, has numbers and/or non-alphanumeric characters, is in an abbreviation

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

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

$$p(\mathbf{w}|\mathbf{t}, \mathbf{o}) = \prod_{i=1}^{m} p(\mathbf{w}_i|\mathbf{w}_{i-1}, t_i, \mathbf{o}). \tag{10}$$

The probability $p(\mathbf{w}_i|\mathbf{w}_{i-1}, t_i, \mathbf{o})$ is impractical to estimate using standard language modeling techniques based on smoothed relative frequency estimates, 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 $p(\mathbf{w}_i|\mathbf{w}_{i-1})$, $p(\mathbf{w}_i|t_i)$ and $p(\mathbf{w}_i|o_i)$ in an exponential model. In addition, the exponential model offers a framework for easily incorporating triggers and/or other predictors based on long distance statistics $s(\mathbf{o})$ as in $p(\mathbf{w}_i|s(\mathbf{o}))$.

6. System description

In this section we turn to a description of the system developed here, including the overall architecture and the details of the particular modules.

6.1. Architectural overview

The system used in these experiments implements a subset of the ideas described in the theoretical discussion in the previous section. The architecture of the system is diagrammed in Figure 2 and includes:

(1) Tokenization and NSW detection;
(2) Detection and splitting of compound (SPLT) tokens, which gives the observation sequence $\mathbf{o} = \{o_1, \ldots, o_n\}$;
(3) Prediction of the best tag sequence $\mathbf{t}^*$, where $\mathbf{t}^* = \text{argmax}_{\mathbf{t}}\, p(\mathbf{t}|T[\mathbf{o}])$, using a decision tree T;
(4) Expansion of tokens to possible word sequences given the tag labels, which results in a word lattice with each branch in the lattice annotated with $p(o_i, t_i|\mathbf{w}_i)$; and
(5) Search for the best path in the lattice according to

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

which incorporates an n-gram language model.

Each of the modules will be described in detail below, but a few things should be noted here, given the preceding discussion. The implementation is largely consistent with the source–channel approach, and could easily be generalized to use a lattice at the tag level with the most probable tags according to the decision tree. Note that, in using the tag decision tree, we have also borrowed from the direct model. The primary motivation for mixing models here is to reduce the decoding cost of the source–channel model. This is not a problem theoretically, so long as the tag prediction model is used to prune the search space and not to annotate the tag branch probabilities (which should instead be marked with $p(t_i|\mathbf{w}_i)$). In effect, the joint maximization over t and w in Equations (3) and (5) is approximated by separate maximizations over t and w to cut computation and storage costs, with the posterior distribution used for t to reduce pruning errors.

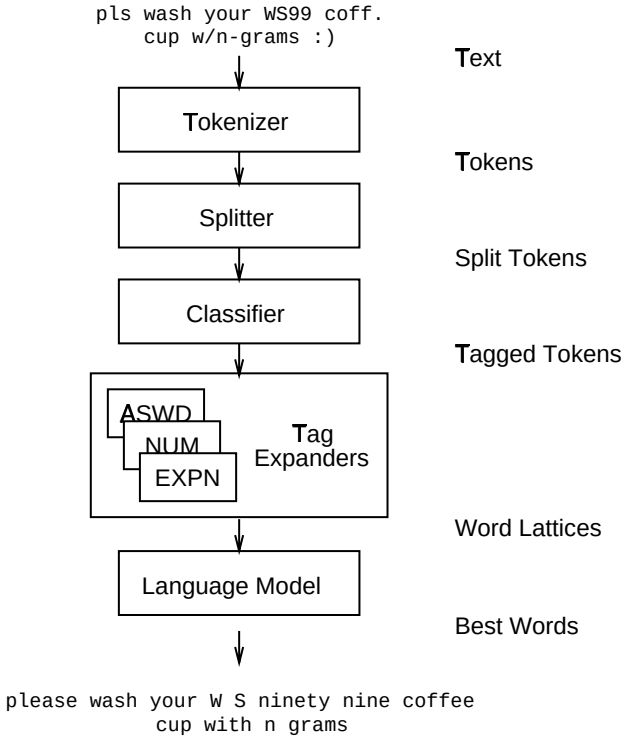

Figure 2. Overall architecture of the text normalization system, with a hypothetical input sentence *pls wash your WS99 coff. cup w/n-grams :-).*

Since there are multiple system components that make use of decision trees, weighted finite-state transducers, and n-gram models, these techniques are reviewed here briefly. Decision trees (Breiman, Friedman, Olshen & Stone, 1984; Quinlan, 1993) are used for tag prediction and for abbreviation probability models in the case of unsupervised training (described in Section 7). The decision trees use CART-style training, which involves iteratively partitioning the training data according to the binary questions that minimize the entropy of the leaf node distributions. The trees are grown to maximize accuracy over held-out data from the whole training set. Weighted finite-state transducers (WFSTs) are used in the splitter (one implementation) and in the expansion module (see, e.g. Mohri, Pereira & Riley, 1997). Finally, n-gram sequence models (Jelinek, 1997) are used for improving discrimination of the different alphabetic tag types (letter language model) as well as for disambiguating potential expansions (word language model). Standard n-gram training techniques are used, as surveyed in Chen and Goodman (1999).

The following subsections describe in detail each of the modules introduced above and diagrammed in Figure 2. All results reported in this section are on the development test set and, with the exception of the NSW detection and splitting modules, reflect results from supervised domain-dependent training.

TABLE VI. Precision and recall rates for detecting NSWs, as measured in a hand-corrected subset of the training data

Domain dependent?	Detection algorithm	Precision/Recall NANTC	classifieds	pc110	RFR
No	non-lexical	55/79	96/79	80/65	76/82
No	+ sct + abbrevs	44/93	95/91	70/90	73/96
Yes	++ abbrevs	39/93	92/92	60/91	46/97

6.2. Tokenization and NSW detection[1]

The text is first broken up into whitespace-separated tokens, and leading and trailing standard punctuation is removed and saved as features of the token. Initially, tokens were marked as an NSW using only the simple dictionary criterion, i.e. if the token was not in the dictionary, it was marked as an NSW. However, in analysis of a small sample of fully hand-marked data, we found that the detection rate for NSWs was not sufficiently high, as shown in the recall figures of the first row of Table VI. Therefore, the data was reprocessed by marking any single character token (sct), tokens with punctuation (e.g. *I.B.M.*) and common abbreviations (that were in the lexicon) as an NSW. This significantly improved the detection rate with some loss in precision, as shown in the second row of Table VI. Since falsely detected NSWs can be tagged with ASWD, the lower precision was not deemed to be a problem. Finally, we revised the tagging once more to add NSW tags for common abbreviations that also correspond to standard words (e.g. *sun*, *Jan*), as well as a few domain-specific abbreviations (e.g. *kit* and *named*). Again, recall was improved at the cost of reduced precision; see the third row of Table VI. In the interest of generating "truth" with as little noise as possible, we used the version with the highest recall rate.

6.3. Splitter

After the tokenizer resolves the original input text into whitespace separated tokens and marks NSWs, the NSWs are passed on to the splitter where they are further broken down into subtokens. 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 typographical 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 a significant portion of the "non-standard" tokens in each corpus should be split. As shown by Table VII, 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 classified ads. This means that, in the classifieds domain where NSWs account for 43·4% of all tokens, 7·29% of all tokens should be split.

In most text, there will be several forces at work which result in aggregate tokens, hence necessitating a splitter. Some examples from the various corpora will help to illustrate the problem at hand. Consider the examples in Table VIII. Each example demonstrates a different class of tokens which require splitting. From left to right, we have a letter sequence with a

[1] Since this section is rather short, we forego the division into a "Method" and "Evaluation" subsection that we use in subsequent sections.

TABLE VII. Percent of NSW tokens in the training set that are labeled with SPLT

NANTC	classifieds	pc110	RFR
4·36	16·77	14·80	7·00

TABLE VIII. Examples of tokens to split

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

morphemic affix; a concatenation of a number and an abbreviation; a brand name; and an ambiguous numeric token, which must be split to remove ambiguity.

In the following sections, we discuss the algorithms and approaches we developed for splitting, and we present an evaluation of the splitter's performance.

6.3.1. Method

Many of the character sequences we would like to emit as single tokens contain within them cues that would ordinarily cause those sequences to be split. For example, while commas, hyphens, and slashes usually indicate subtoken boundaries, they should not be so construed when appearing in numbers, telephone numbers, and dates, respectively. We will refer to these problematic single-token character sequences as *groups*.

Given the presence of groups, we use two passes in our approach to splitting tokens. In the first pass we search the input for groups and bracket those we find. This bracketing serves two purposes. First, the presence of brackets around a group causes the second pass of the splitter to consider the group indivisible. Second, the begin- and end-bracket markers themselves induce split points at their respective locations. In this way we attempt to keep groups coherent and independent from the surrounding text.

In the second pass, we hypothesize split points without regard to any underlying structure in the text except for the bracketed groups. In contrast to the first pass, which requires a plethora of rules to cover a correspondingly numerous set of special cases, the second pass is relatively straightforward: a mere four to six rules sufficed to hypothesize split points. 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.

We developed a rule-based approach to hypothesizing the subparts of aggregate tokens, which was implemented in the Perl text processing language for use in the overall system described here. The rules can also be implemented using weighted finite-state transducers.

6.3.2. Splitter 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 decides *where* to split a given token, where accuracy is at

TABLE IX. Results for the splitter. Precision and recall indicate accuracy of detecting NSW token to be split, and the "correct" figures indicate accuracy of split location for the subset of known SPLT tokens and all tokens, respectively

	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

the token level, i.e. a "split" is correct only if all generated subtokens exactly match those in truth. These measurements are reported in terms of "split correct" and "total correct"; the former considers only those tokens which are known to be split (analogous to recall), while the latter considers all NSW tokens regardless (analogous to precision).

The results of both of these stages are found in Table IX. In the above table, the relatively poor precision in each domain is quite striking. It is, however, in many cases an overly pessimistic measure of the system's quality. While the results indicate that the splitter does over-generate splits, there are some mitigating factors. First, it is often the case that a hyphen used in a delimiting context, as in *12:00-3:00* and *$975K-$1,595,000*, is not correctly used by the human labelers to split the token. In many cases, the splitter is more reasonable than the human specified form (or at least more consistent). Further over-generative errors are not problematic because it is unclear whether the split form would be pronounced differently from the "correct" form, as is the case with many tokens (such as hyphenated words) whose internal punctuation was the impetus for the split.

6.4. A tag classifier for non-standard words

The purpose of the NSW classifier is to categorize a token from the output of the splitter into any one of the predefined NSW types, e.g. 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.

6.4.1. Method: overall tag classifier

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

We use 136 features in training the decision tree which can be separated into two classes:

(1) Simple domain-independent features.
These look at properties of the individual token, and require no domain-specific information to calculate. These are features about the character content (all alphabetic, numeric, or mixture), vowel/consonant content, casing (all upper, or lower, or mixed), and if it contains some specific punctuation symbols (slash, dot or dash). These features are provided for the current token and tokens within a window of two before and two after.

(2) Domain-dependent features.

As the range of alphabetic tokens changes over the different domains, we also calculate a set of features that are based on domain-specific information. These features constitute 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 section.

6.4.2. Method: 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 main categories:

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

Tokens hand-tagged as MSPL, FNSP and OTHER may also fall into the class of alphabetic tokens, but they are rare in our data and so we do not attempt to explicitly model them.

The alphabetic token classifier problem can be given a statistical formulation as follows. The probability of assigning NSW tag t to observed string o can be estimated using the familiar Bayes approach as:

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

where $t \in \{\text{ASWD, LSEQ, EXPN}\}$. The terms of this equation are derived as follows:

(1) The probability $p(o|t)$ can be described by a trigram letter language model (LLM) for predicting observations of a particular tag t.

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

where $o = (l_1, l_2, \ldots, l_N)$ 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 & Witten, 1990) and estimation of language entropy (Brown, Pietra, Pietra, Lai & Mercer, 1992). The language model used is the most widely adopted n-gram (in our case trigram) formulation (Jelinek, 1997).
(2) The probability $p(t)$ is the prior probability of observing the NSW tag t in the text.
(3) The probability of the observed text or the normalization factor is given by

$$p(o) = \sum_t p(o|t)\,p(t).$$

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

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

(1) $p(t|o), t \in \{\text{ASWD, LSEQ, EXPN}\}$;

TABLE X. Examples of alphabetic tokens with LLM features and correct tag

Token	$p(\text{ASWD}\|\mathbf{o})$	$p(\text{LSEQ}\|\mathbf{o})$	$p(\text{EXPN}\|\mathbf{o})$	$p_{\max}$	$t_{\max}$	*diff*	Correct
mb	0·0001	0·0038	**0·9962**	0·9962	EXPN	0·9924	EXPN
Grt	0·0024	0·0000	**0·9976**	0·9976	EXPN	0·9952	EXPN
NBA	0·0017	**0·9983**	0·0000	0·9983	LSEQ	0·9966	LSEQ
Cust	**0·5456**	0·0000	0·4544	0·5456	ASWD	0·0912	EXPN

TABLE XI. Accuracy of the three-way LLM classifier compared to a baseline of choosing the most probable class (supervised training paradigm)

Domain	NANTC	classifieds	pc110	RFR
Baseline	83·90	80·53	63·77	69·98
LLM Classifier	95·72	98·74	92·27	97·92

(2) $p_{\max} = \max_t p(t|o)$ (maximum probability of an alphabetic category);
(3) $t_{\max} = \text{argmax}_t\ p(t|o)$ (most probable alphabetic tag);
(4) *diff* = Difference between highest and second highest probabilities $p(t|o)$.

Some examples of tokens with the six LLM features are displayed in the Table X.

In supervised training of the letter language model, the alphabetic tokens for each of the tag categories [ASWD, LSEQ or EXPN] are partitioned into separate sets, one for each domain. If there are multiple examples of a particular token in a set (e.g. multiple instances of *kit* labeled as EXPN), then all instances are used in training the LLM for that tag type. A trigram LLM is trained for each case using the modified Kneser–Ney smoothing (Kneser & Ney, 1995; Chen & Goodman, 1999).

6.4.3. Evaluation of the LLM and overall tag classifier

To obtain an estimate of the sub-classifier accuracy, an intermediate token error rate can be calculated on alphabetic tokens. The (supervised) domain-dependent alphabetic LLM classifier is compared to a baseline where all alphabetic tokens are labeled with the most frequent tag for that domain, which is ASWD in the NANTC, pc110 and RFR domains and EXPN in the classifieds domain. The accuracy of the classifier in these experiments is presented in Table XI.

The decision tree for the overall classifier was built with and without LLM features in order to judge the effect of these extra features on the overall accuracy. The corresponding accuracies are presented in Table XII, showing that there is a small gain in performance due to the LLM, with the exception of the RFR corpus. It is possible that the RFR LLM suffered from sparse training problems, since that corpus was smallest and had the fewest alphabetic tokens. However, the result seems surprising given the high accuracy reported in Table XI.

The domain-dependent LLM that has been trained on each on the three alphabetic tag categories, was used to compute perplexity of the test data. These perplexity values are given in Table XIII.

TABLE XII. Overall NSW tag classifier accuracy (supervised training paradigm)

	NANTC	classifieds	pc110	RFR
no LLM features	97·7	92·7	90·9	97·3
all LLM features	98·1	93·5	91·8	96·8

TABLE XIII. Perplexity of the domain-dependent LLM on alphabetic NSW tokens

Tag Category	NANTC	Classifieds	pc110	RFR
ASWD	22·7	19·0	16·9	13·9
LSEQ	86·9	56·0	62·6	50·3
EXPN	28·7	14·4	31·8	25·6

6.5. Word expansions

As discussed in Section 6.1, our system implementation involves choosing between possible expansions of tokens according to

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}}\, p(\mathbf{o}, \mathbf{t}^*|\mathbf{w})\, p(\mathbf{w}),$$

where $\mathbf{t}^*$ is the most likely tag sequence and the lexical model has the form

$$p(\mathbf{o}, \mathbf{t}^*|\mathbf{w}) = \prod_{i=1}^{m} p(o_i, t_i^*|\mathbf{w}_i).$$

Given a hypothesized tag t_i^*, the word expansion module provides the set of possible expansions $\{\mathbf{w}_i\}$ and the associated conditional probability of the observed token $p(o_i, t_i^*|\mathbf{w}_i)$. For the cases where the expansion is algorithmic, described in Section 6.5.1, a single observation is generated and the observation probability is trivial. For the EXPN case, discussed in Section 6.5.2, there is genuine ambiguity and the observation probability is an important component.

6.5.1. Method: 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 pronunciation (i.e. *one hundred one* vs. *one hundred and one*), the expansion algorithm is not dependent on the domain or context. In this work, the labelers were not asked to choose a specific pronunciation for efficiency reasons, so it was not possible to estimate observation distributions and a single choice is made. As mentioned earlier, for many domains and especially for text-to-speech synthesis applications, such an approach is quite reasonable.

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

NONE: expands to no words.
SLNT: expands to no words.
PUNC: expands to itself.
ASWD: expands to itself.

LSEQ: expands to a string of words, one for each letter.
NUM: expands to string of words representing the cardinal number. This covers integer, decimal and Roman forms.
NORD: expands to string of words representing the ordinal number. The token may be simply numeric, numeric appended with *th*, *st* or *rd*, or it may also be Roman.
NDIG: expands to string of words, one for each digit.
NYER: expands to words as in a pronunciation of a year. That is, each pair of digits is pronounced as NUM except where the last two are *00*, where the group of four are pronounced as a whole.
NADDR: expands as words using the same (digit pairs) algorithm as for NYER.
NZIP: expands as string of digits, with deletion of dashes within the token.
NTEL: expands as string of digits with deletion of punctuation.
NIDE: expands as string of letters and digit pairs (as in NYER).
MONEY: expands to string of words to say the number and currency; deals with various currencies.
BMONEY: 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 BMONEY token (*million*, *billion* or *trillion*) and are ASWD expand to nothing.
NDATE: expands as a number sequence.
NTIME: expands as a number sequence with *hours*, *minutes* and *seconds* as appropriate.
PRCT: pronounced as a number with the '%' sign removed and the word *percent* appended.
URL: written as is with no expansion; not addressed in this work.
OTHER: written as is with no expansion.
MSPL: written as is with no expansion: spelling correction is outside the scope of this work.
FNSP: again written as is, since there is no record of the "proper" word. In fact, there often is not a proper word as in tokens like *Hmmmm* and *Arghhhh*.

6.5.2. Method: the EXPN *model*

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 $p(o|\mathbf{w}) = p(o, t|\mathbf{w})$, given the distributions of words and their abbreviations in the training data. (The equality is because an abbreviated observation can have only the EXPN tag.) In our case, the maximum likelihood estimate (i.e. relative frequency) is used. Note that since we know the word and its abbreviation(s), we do not need to separately estimate $p(t|\mathbf{w})$ (the probability that a word will be abbreviated in any form) and $p(o|t, \mathbf{w})$ (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.

In this work, we do not attempt to expand cases where the observation o has never been observed in the training corpus, but in principle that could be handled by the generative model described in Section 7. Evaluation of these techniques is given in Section 8. In our data the only examples of EXPN tokens for which the expanded form did not appear in the training set were *etc* and *Mrs*, which we can reasonably assume as being standard. However in the classified ads, which contain much more productive abbreviations, new data from a different

(geographical) area is very likely to contain EXPNs for which we have not previously seen their full expansion.

6.6. Language modeling

In many cases, the best choice between different expansions can be determined by local lexical context. For example, when *St.* is preceded by a name it is typically *street*, and when it is followed by a name it is typically *saint*. Since n-gram language models are good at representing local context, they are potentially very useful for disambiguation. In particular, since an n-gram language model is easily trained from data, it can pick up dependencies that a human writing rules might not notice.

6.6.1. Method

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 & Goodman, 1999). Language models were trained separately for each of the four domains. The vocabulary for each n-gram model was constructed by collecting every token that occurred in the corresponding training set. Some tokens (particularly non-alphabetic NSWs) 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. This is to make the language model less sparse for the cases where knowing the identity of the token rather than the tag will not help much in disambiguation. For example, a number might be mapped to NYER or NUM rather than expanded to the associated word sequence. In effect, this gives us a partially class-based language model. It also means that the actual implementation is slightly different than the architecture described in Section 6.1, in that the algorithmic expansions actually come after the lattice scoring, though this is not a requirement of the class-based language model.

Note that, because of the use of hand-labeled NSW tags and expansions, the language models 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. A more extensive use of class-based language modeling techniques might be useful in future work, which would help with cases such as the *St.* example above.

6.6.2. Evaluation of expansion with LM disambiguation

Testing was done on the classified ads data only, under two conditions:

(1) without the language model, in which case we pick the expansion for an abbreviation that maximizes $p(\mathbf{w}_i|o_i)$ separately for each i; and
(2) with the language model, in which case we pick the expansion that maximizes $p(\mathbf{o}|\mathbf{w})\,p(\mathbf{w})$.

Performance is evaluated only on the true EXPNs.

The error rates were 4·8% and 6·7%, respectively with and without the language model. 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, and informatively, most of the difference between the with and without language model cases resulted from a bug in the database labeling involving a single NSW type: the abbreviation *SF* was tagged as meaning *south facing* in most instances, though a few correct taggings 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 preferred. Thus, the language model was correcting for annotator error.

7. Unsupervised training

The previous discussion has focussed on supervised methods for treating NSWs. As we have seen, the distribution of NSWs is very different across the different corpora. As one might expect, using a model trained on one domain for another gives significantly degraded performance. Thus, model parameters are best estimated within the domain that one wishes to apply the model to. This applies not only to alphabetic NSWs—abbreviations, letter sequences and acronyms—but to NSWs of other classes too: a sequence like *110* is much more likely to be read as *one ten* in text from the pc110 newsgroup than it is in more general text. However, the richest source of variability among NSWs across domains is found in alphabetic NSWs. Classified ads, for example, contain many abbreviations—*BR* (*bedroom*), *DR* (*dining room*), *W* + *D* (*washer and dryer*)—that are unique to this domain.

When one has a hand-built abbreviation list for a particular domain, one naturally can achieve good coverage for that domain, but there are many different text domains, and it is therefore simply impractical to expect that one will have a totally hand-constructed abbreviation list for every new domain that one encounters. The question naturally arises whether one can bootstrap an expansion model and how well one can do with such a model. This is the topic of this section. Since alphabetic NSWs pose the most challenging examples and make up a large fraction of the total NSWs (ranging from 30% to 55%), we decided to focus on these in investigating unsupervised training of NSW normalization models.

The treatment of alphabetic NSWs includes two aspects. The first is classification: we need to know whether a given alphabetic NSW should be treated as a letter sequence (LSEQ), a word (ASWD) or an abbreviation (EXPN). If it is one of the first two, then the reading is more or less algorithmic, but if it is an EXPN, then we also need to predict how to expand it. We treat each of these issues, classification and expansion, in the next two sections, followed by a discussion of the expansion results and related work.

7.1. Unsupervised NSW tag classification

The alphabetic NSW sub-classifier first introduced in Section 6.4.2 can also be trained in an unsupervised paradigm that does not require any labeled NSW data for the domain. Rather, 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 list of EXPN and LSEQ tokens on a domain-dependent basis without the need for labeled NSW tokens. The supposed LSEQ tokens are extracted by searching for tokens that are comprised of lower or upper case characters alternating with periods. In addition, the known LSEQ tokens from the NANTC domain can be used to augment the domain-specific lists. The heuristics used to construct the EXPN lists consist of selecting tokens with the following text patterns:

- Alphabetic tokens with no vowels.
- Tokens followed by a period, which are followed by a token starting with a lowercase character. This heuristic is intended to capture abbreviations ending with a period and

TABLE XIV. Accuracy of the three-way LLM classifier (unsupervised training paradigm)

Domain	NANTC	classifieds	pc110	RFR
Baseline	83·90	80·53	63·77	69·98
Unsupervised	92·98	87·90	68·90	92·06
Supervised	95·72	98·74	92·27	97·92

not followed by a sentence boundary. (Example: *frplc. ba* in which *frplc* is an abbreviation that has to be expanded.)

- Plural forms of the tokens falling in the previous category.

The two unsupervised lists serve as the training data for the LSEQ and EXPN letter language models, respectively. As in the case of supervised LLM training, these two lists included multiple instances of tokens when they occur more than once in the respective data sources. Words in the CMU dictionary (CMU, 1998) with more than four characters are used as the training data for the ASWD letter language model. Once these lists have been compiled, letter language models are built separately for each list as before.

We evaluated the unsupervised method on the four domains as before: NSW tokens from the test data for each domain are extracted, and alphabetic tokens are filtered from these NSW tokens for input to the classifier. The accuracy of the three-way NSW classifier on alphabetic tokens in the unsupervised evaluation paradigm is presented in Table XIV, with the prior baseline and supervised results included for reference. Significant improvements are made in the RFR domain, but the gain is small for the pc110 domain. Note that the model for the NANTC domain is not strictly unsupervised because of the knowledge used in building the LSEQ training list.

7.2. Unsupervised treatment of abbreviations

For training an unsupervised abbreviation expansion model, we assume that one has an untagged corpus from the domain of interest, plus a method, such as the one described in Section 7.1, for deciding if a word should be expanded.

We also crucially assume that one's untagged corpus gives evidence for the correct expansion for each of the abbreviations in question. To see what we mean by this, consider a corpus of classified ads in which we have the abbreviations *kit* and *livrm* as in the example:

> eat-in kit livrm dinrm 17x25 famrm

This is a sample from a highly abbreviated ad, but one may hope that elsewhere one can find ads that are less highly abbreviated that contain examples such as:

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

The problem is to identify that *kit* is a plausible abbreviation of *kitchen* and that *livrm* is a plausible abbreviation of *living room*. More precisely, one would like to estimate the probability $p(o, t|\mathbf{w})$, where in this case o is, say, *kit*, t is (by definition) EXPN, and $\mathbf{w}$ is *kitchen*: in other words, the probability that one wants to abbreviate, e.g. *kitchen*, and that one would do it as *kit*.

Making a reasonable guess about a possible expansion of an abbreviation requires modeling the abbreviation process. We present a decision-tree-based model of this process in

Section 7.2.1. The procedure for training and testing an unsupervised model is given in Section 7.2.2, and then experimental results are described in Section 7.2.3.

7.2.1. Method: a 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 heuristics 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 to predict whether or not a letter would be deleted. The system was trained on 854 pairs of hand annotated abbreviations and their expansions—taken from the classified ads, as this is the domain richest in productive deletions. The alignments between the full form of the word and the abbreviated form in the training data were produced by composing the full form with a WFST that allows letters either to delete at a cost, or to be mapped to themselves; the resulting transducer is then composed with the abbreviated form, and the cheapest path selected. This will result in an alignment where full-form letters match in a left to right fashion with the letters in the abbreviated form, and are otherwise deleted.

The features chosen for training were as follows:

(1) 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").
(2) The boundary to the immediate left and right of the current letter: word-boundary, morpheme-boundary or no boundary.
(3) The fate of the -2 letter and -1 letter: deleted or not deleted.
(4) The fate of the $+1$ letter and $+2$ letter: deleted or not deleted.

The last feature (which clearly depends upon knowing the future output) produced better results: without this feature the tree performed at 85% (measured by cross-validation) and with the feature it performed at 88% in predicting deletion. Note that the baseline for this task (always predict deletion) 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 (CMU, 1998), possibly followed by *-s* or *-es*.

The model implemented by the tree can be described formally as follows. Given an observed abbreviation $\mathbf{o}$ of word w, we assume that there is a unique mapping to an $\mathbf{o}'$ the same length as w, where deletions are marked as ϵ. This uniqueness assumption is reasonable since:

- We are only modeling deletions. It would be more complex if we modeled substitutions, but we decided to simplify the model since the vast majority of abbreviations can be handled by deletions only.

- In cases where a letter in an abbreviation might in principle have been derived from more than one full-form letter—e.g. *excel* from *excellent*, where there are two potential sources for the *l*—we can for the sake of simplicity assume a "greedy" left-to-right mapping.

Then the model can be given as

$$p(\mathbf{o}|w) = p(\mathbf{o}'|w) = \prod_{i=1}^{d} p(I_i(\mathbf{o}')|T(\mathbf{o}'_1, \ldots, \mathbf{o}'_{i-1}, w))$$

where d is the length of o' and w, $I_i(\mathbf{o}')$ is the deletion indicator function ($I_i(\mathbf{o}') = 1$ for $\mathbf{o}_i{}' = \epsilon$ and 0 otherwise), and T is the decision tree that operates on features as already described. The product can be viewed as a probability estimate as long as *future* values of $I_i(\mathbf{o}')$ (e.g. the future fates described in 4) are not used as features. If future values of $I_i(\mathbf{o}')$ are used, then the resulting product is not strictly a probability, but it does provide a useful score for use in a minimum cost search.

The tree-based model was compiled into a weighted finite-state transducer (WFST) using an algorithm related to that reported in (Sproat & 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 abbreviation 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 one of the readings of "+" is *plus*: this is unlikely to be amenable to purely unsupervised inference.

7.2.2. Method: unsupervised training

The unsupervised abbreviation expansion model starts with an untagged corpus from which we assume we can infer the set of ordinary ("clean") words and the set of abbreviations. The abbreviations can be identified, for example, using the classifier described in Section 7.1, and the set of ordinary words can be identified by matching to a dictionary.

The basic procedure for training then requires two components, described below:

Abbreviation model. We collect the set of potential EXPNs. We also collect the set of clean words and clean word pairs occurring in the corpus: note that abbreviations may expand to more than one word (cf. *livrm* for *living room*), and expansions into two words are particularly common in some genres such as classified ads. Call the thus-derived lexicon of EXPNs 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 7.2.1. Then, the set of possible abbreviations can be derived by composing SW with A and composing the result

with NSW:

$$SW \circ A \circ NSW \tag{11}$$

where "$\circ$" denotes the composition operation (see, e.g. Mohri *et al.*, 1997). This will give a transducer that maps between ordinary words or word pairs and potential abbreviations, but of course for tagging raw text, we need a transducer that computes the inverse of this relation. This *expander* can be trivially derived by inverting the transducer above:

$$expander = [SW \circ A \circ NSW]^{-1}. \tag{12}$$

This expander can then be used to expand tokens classified as EXPNs.

Language model. In the ideal case, we train the language model on perfectly clean text from the domain of interest, i.e. containing no abbreviations. This is of course possible if we have hand-annotated data, since we know what the correct expansion of abbreviations is. In the unsupervised case, by assumption, all we have is text containing some unknown number of abbreviations, for which we do not know the expansion. Since the language model is intended to be a model of the "clean" version of the text, we at least want to exclude the abbreviations from the parameters of the language model by treating them as unknown words: this we do by mapping all alphabetic tokens identified as EXPNs to the unknown word tag ($\langle$UNK$\rangle$).

Figure 3 gives an example of what the language model is trained on in both supervised and unsupervised situations. The first column represents the raw text. The second column is the fully expanded text, such as we might get from hand annotation, and illustrates the data that would be used in supervised training. The third column represents the text that the language model would be trained on in the unsupervised case. As in the supervised case, we ignore the expansion of non-alphabetic NSWs, but rather simply substitute for them their tag (automatically predicted, in the unsupervised case). Thus *40* is represented in all cases as @NUM+PL (plural number). Again, this gives us a partially class-based language model. (Note the tagging error associated with a missed NSW detection: *SO* should be expanded to *south* in column 2 but not in column 3.)

The language model used in all cases was a trigram language model using modified Kneser–Ney smoothing (Kneser & Ney, 1995; Chen & Goodman, 1999), as in the supervised training experiments.

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 next, 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.

7.2.3. Experimental evaluation

All experiments described here were run on a portion of the classified ad corpus divided into 307 735 tokens of training data, and 76 676 tokens of test data, taken from the *training*

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

Figure 3. Sample text and annotation for unsupervised language model training: column 1 shows the source tokens, column 2 shows the non-alphabetic tokens replaced by an automatically generated NSW tag, and column 3 changes all alphabetic tokens marked as NSWs to unknown words.

portion of the data.[2] For these experiments only—*not* for the final evaluation discussed in Section 8—we used a simpler classifier than the one described in Section 7.1. We simply classified alphabetic tokens as "clean" if they were in the dictionary, and as potential EXPNs otherwise.

The error rates reported here correspond to *token error rates* (see Section 8.1) on the true EXPNs. Since only automatically classified potential EXPNs are considered for possible expansion, we find errors—reflected in the error rates reported here—where true alphabetic EXPNs were missed because they were not classified as EXPNs. We also get "overgeneration"—*not* reflected in the error rates—for tokens that were classified incorrectly as EXPNs; the latter point is discussed at the end of this section.

Finally, note again that in these experiments we uniformly mapped characters to upper case. This results in some loss of information. For example, *w* in the classified ads is usually *with*, whereas *W* is usually *west*, something that our supervised list-based methods could take advantage of. On the other hand, our unsupervised methods currently do not take 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, since the language model ignores case distinctions. Hence, for the sake of uniformity across training conditions, we ignore case for both supervised and unsupervised methods. However, with more complex language models, it may be possible to learn such preferences, as illustrated by the above example, where the preference is surely related to the content-word/function-word distinction, plus the fact that *west* is frequently capitalized.

Note that in the unsupervised experiments described below, language modeling may be used in two possible ways. In all but experiment 4, the language model is used for disam-

[2] As a reviewer of this paper points out, it would be interesting to see how sensitive the performance reported here is to training set size. Unfortunately we did not vary the size of the training data in our experiments, so we cannot address that question here.

biguating possible expansions in the test data. In experiments 4–6, the language model is used on the training data to provide better expansion training data.

In order to interpret the experimental results for unsupervised training described next, it is useful to recall that the performance of the algorithms based on supervised training were at 6·7% error without the language model for disambiguation, and 4·8% with the language model.

Finally note that in this section we report only token error rates: while different from word error rates, the two are of course highly correlated, so it is sufficient for the purpose of evaluating refinements of the unsupervised abbreviation methodology to concentrate on token error rates.

Unsupervised method: experiment 1. In the first experiment we constructed SW out of:

- All singleton clean words.
- All clean 2-word sequences occurring in the data a minimum of three times.

The three-word minimum was felt to be a reasonable cutoff, since anything occurring fewer than three times is probably unreliable anyway. The language model was built on the training data as described above in Section 7.2.2. During testing, automatically detected EXPNs were submitted to the expander ($[SW \circ A \circ NSW]^{-1}$) and the various alternatives scored by the language model. This first test had a token error rate of 33%.

Unsupervised method: experiment 2. It was observed that some of the errors introduced in experiment 1 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 clean words occurring more than ten times.
- All clean two-word sequences occurring in the data filtered with the above list.

Here, the error rate increased to 34·5%, clearly indicating that information was lost by the filtering process.

Unsupervised method: experiment 3. In experiment 3, we returned to the method for constructing SW from experiment 1, 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%.

Since we are providing some hand annotation in this experiment, one may be inclined to think of this as cheating. A more charitable interpretation is that this approach 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, something that is not unreasonable to expect.

Unsupervised method: experiment 4. In experiment 4, we had the same setup as experiment 3, but returned to investigating 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 EXPNs on the test data.

In this setup, there is clearly no need to use the language model during testing, since there is only one expansion allowed for each EXPN. Here again, the error rate was reduced substantially, to 19·9%.

Unsupervised method: experiment 5. An obviously unfortunate property of experiment 4 is that any given abbreviation can only have one expansion, whereas we know that at least some EXPNs 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: the reason for this minimal count was to eliminate noise. 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 $p(o|\mathbf{w}) = p(o, t|\mathbf{w})$ as we would in the supervised case.

Unfortunately, this resulted in no reduction of error rate: 19·9%. However, it was 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.

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, and reestimated $p(o|\mathbf{w}) = p(o, t|\mathbf{w})$ for up to two expansions for each abbreviation. This method resulted in a modest reduction in error rate, to 19·5%.[3]

The results of each experiment are summarized in Table XV. Note that the best error rate—19·5%—is roughly four times the error rate of the supervised method tested on the same data.[4]

7.3. Discussion

The work discussed above has shown that if one is unable to tag a lot of 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

[3]One reviewer asks: "what would be the impact on token eror rate of using a language model trained on normalized text". Note that this experiment answers that question to some extent since although the "normalized" text in this case is automatically inferred, and not the same as "truth", it does show that there is certainly some sensitivity to having a language model that is trained on text that is closer to unabbreviated text.

[4]The results reported here cannot be directly compared with the results in Section 8: the current results are for EXPNs only, whereas the results reported in Section 8 pertain to performance of all the components. Furthermore, the test set for evaluation in this section was different from that used in the evaluations in Section 8.

TABLE XV. Summary of experimental results on abbreviation expansion

Experimental condition		Token error rate (%)
Supervised	no language model	06·7
	with language model	04·8
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

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 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 & 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, since any such assumption would be impractical. 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.

We have already noted the similarity of the current work to that of Rowe and Laitinen (1995), discussed in Section 2.3. 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. Overall system performance

This section compares the overall performance of the systems we built to that of pre-existing systems that attempt the same task. Methods for evaluating text normalization systems have not been well-established. Since we have hand-annotated data in each of our domains (which we refer to as "truth"), a natural metric is to calculate a token or word error rate with respect to this "truth". We use this metric for most of our evaluations, as described in Sections 8.2 and 8.3. However, this type of evaluation falls short of the ideal in two main respects. First, there are often multiple acceptable normalizations of a given piece of text, while our "truth" only specifies a single alternative. (To create manually annotated data containing all acceptable alternatives is terribly costly.) Second, the truth was generated in a semiautomatic fashion, and the automatic processes used may have introduced systematic errors. As one example, only NSWs that were automatically detected as such were manually annotated. However, it was shown in Section 6.2 that our detection algorithm is imperfect. Another example is that common abbreviations such as *BR* in classified ads were expanded automatically. Thus, we do not know for sure whether the correct expansion in each case is *bedroom*, *bedrooms*, or something else entirely.

To address these issues, we also performed another type of evaluation. Instead of automatically comparing system output to reference text, we manually judged the acceptability of system output to estimate error rates. Due to the labor-intensive nature of this method, we only performed a few evaluations of this type. These evaluations are described in Section 8.4.

8.1. Measurement criteria

We chose 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 for the LDC tools we can only report word error rates. Although some dynamic programming method could be used to reconstruct the token boundaries, we did not do that since we believed that further errors could be introduced that way. 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 an ASWD.

While neither measure is ideal, looking at the erroneous tokens, we feel these figures adequately represent the relative accuracies of these models.

TABLE XVI. Performance relative to truth of existing text analysis systems on the different corpora

	LDC tools		Festival	
	TER	WER	TER	WER
NANTC	—	02·88	01·00	01·38
Classifieds	—	30·81	30·09	33·48
pc110	—	22·36	14·37	32·62
RFR	—	09·06	06·28	16·19

8.2. Baseline systems

One of the purposes of this work was 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 chosen two publicly available pre-existing text analysis systems to show what the current state is.

The first is the LDC text conditioning tools, described previously in Section 2.2. The second system is Festival (Black *et al.*, 1999), 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.

The performance relative to truth of the two systems on the different domains is given in Table XVI. As the mechanism used to generate the truth that these systems are compared against uses some of the same mechanisms that the Festival text analyser uses, there is probably a slight bias in these results that favors Festival. Looking closely at "errors" in the LDC NANTC results, we estimate that a truer error rate would be close to 1·5, as many of the "errors" are actually trivial: for instance, hyphenated words are sometimes not split by the LDC tools although they are in our reference expansion. Also, given the results of the manual evaluation, it seems that we cannot make any strong statement about the difference 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. Note that these error rates are not directly comparable with the manual evaluation results to be presented in Section 8.4, both because of inaccuracies in "truth" and because the manual evaluation allows any acceptable alternative to be correct instead of just one.

8.3. Performance of the present NSW system

In this section, we present a series of experimental results that demonstrate the overall performance of the full system and the relative contribution of the different components. First, we give results for the full system using domain-dependent training, with experiments on variations that remove the language models to investigate the relative contribution of these parts. Next, we incrementally add oracles (i.e. truth) at different stages to better understand performance of other components. Then, we describe results for domain-independent models, where the training and test data do not fall into the same domain. Finally, we look at

TABLE XVII. Token and word error rates of text normalization systems with supervised training, comparing the full system to versions without using the language model for disambiguation (No LM) and/or without using the letter language model in the tag prediction tree (No LLM)

	Festival		Full system		No LM		No LLM		No LM/LLM	
	TER	WER	TER	WER	TER	WER	TER	WER	TER	WER
NANTC	01·00	01·38	0·39	0·82	0·39	0·81	0·38	00·78	0·38	00·78
Classifieds	30·09	33·48	7·00	9·71	6·82	9·70	7·55	10·39	7·41	10·42
pc110	14·37	32·62	3·66	9·25	3·63	9·25	3·93	10·90	3·90	10·90
RFR	06·28	16·19	0·94	2·07	0·93	2·06	0·88	02·07	0·88	02·07

domain-dependent but unsupervised models, which use data for a new domain without hand labeling tags or expansions, again assessing variations with and without the language model.

8.3.1. Domain-dependent models

The full system using domain-dependent supervised training consists of the following parts:

- a domain-independent token splitter;
- a domain-dependent decision tree for predicting NSW tags, trained on the hand-labeled tags and including letter language model based features for alphabetic NSWs;
- a domain-dependent WFST for expanding tokens classified as EXPN, built from hand-labeled data and producing a list of potential expansions; and
- a domain-dependent language model that chooses between the different expansions, again trained on hand-labeled tags and expansions.

The results over the different domains are given in Table XVII, showing that the full system outperforms the comparison baselines (LDC tools and Festival, the latter given here for comparison) on all domains, with error reductions of more than a factor of three in many cases.

In addition, the table gives results for cases that exclude one or the other, or both, of the two language models. Excluding the word language model implies using the most probable expansion for a given token independent of context. The results show that the language model does not impact performance except for the classifieds data, where the error rate actually increases with the language model. Looking at the test data, we find that there are only a few places where expansions have valid alternatives, and in those cases it is not clear how a general n-gram language model would help disambiguate them. Removing the possibility of choice guarantees that the less probable expansions will never get selected. Excluding the letter language model implies using a tag prediction tree that was trained without those features. Deleting the letter language model negatively impacts both the classifieds and pc110 domains, both of which have a much higher occurrence of domain-specific acronyms and abbreviations. The RFR domain seems to have better performance without the LLM features, but this may be due to the sparse LLM training data for that domain as noted earlier.

8.3.2. Adding truth through oracles

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: the splitter and the NSW tagger. When the hand-labeled token splits and NSW tags are both used, the only part left is the

TABLE XVIII. Token and word error rates of text normalization systems based on supervised training, comparing the full system to versions with an oracle token split and tagger, where the oracle is the hand-labeled data

	Full system		Oracle			
			Split		Split & tag	
	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

TABLE XIX. Comparison of domain-dependent (DD) and domain-independent (DI) NSW systems with Festival and with a domain-independent system augmented with expansions for the most frequent abbreviations in that domain

	Festival		DD NSW		DI NSW		DI NSW / DD abbr	
	TER	WER	TER	WER	TER	WER	TER	WER
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	06·28	16·19	0·94	2·07	02·71	04·66	02·32	04·14

expander and the language model disambiguation. The results, summarized in Table XVIII, show that while a significant portion of the errors is due to the splitter, a much larger component is due to tag prediction errors. The table also shows that the expansion step is still a substantial part of the problem, particularly in the classifieds, which is the domain with the most abbreviations.

8.3.3. Domain-independent models

The third set of experiments was carried out 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 fourth, but our initial experiments (based on the performance of the decision trees built 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. The results are summarized in Table XIX, with Festival's results for comparison. (Since the results on the NANTC corpus would not be considered domain-independent in this case, they are omitted.) In addition, we include results for a system that is partially domain dependent in that it uses NANTC domain tag and language models, but for EXPNs it uses a list of the most frequent abbreviations taken from the data. The rationale for this test is to find out what would happen if one at least created a new list of domain-dependent expansions for the model—something that is surely much less work than exhaustively labeling a large portion of the data. Although the results for our NANTC model compare favorably with Festival's they are still somewhat poor, particularly for the classifieds and pc110 data. The addition of a list of known abbreviations in the domain helps substantially for the classifieds domain, but it probably still is not useful enough to consider using the output for any real task.

TABLE XX. System performance using different strategies for supervised and unsupervised training for domain-dependent NSW processing of the classified data, compared to the domain-independent result

Training	LM disambiguation?	TER	WER
Domain independent	Yes	25·20	29·11
Unsupervised	No	12·64	13·50
Unsupervised	Yes	12·50	13·40
Unsupervised + abbrevs.	Yes	10·58	13·51
Supervised	Yes	07·00	09·71

8.3.4. Unsupervised models

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 we do not expect it to be labeled. We consider this scenario likely in the case of text conditioning for building language models. So far, we have only done this for the classified domain. The steps involved are:

(1) Use the NANTC model to generate tags for the training data in the new domain;
(2) Instead of taking the NANTC tree prediction as correct, for all alphabetic tokens we take the best prediction based on the letter language model features built from unsupervised data;
(3) We then build a new tag prediction tree based on these new labeled training tokens in our unknown domain;
(4) With this new classifier we expand the data, this time generating words for ASWD and LSEQ. EXPN remains as a tag (as we do not know its expansion) and other tags are left as tags as they are useful as class categories.
(5) A WFST built to automatically expand abbreviations, as described in Section 7.2, is then used to generate the expansions;
(6) We then expand the training data giving the most probable expansion;
(7) A language model is then built on the expanded data; and
(8) The final model runs by predicting a number of possible expansions and deciding between them using the language model.

Using the NANTC tag prediction tree as is on classifieds gives 67·44% tags correct on the development test data, which increases to 86·72% tags correct when the alphabetic tokens are reclassified according to the unsupervised letter language model and a new tree is built (after step 3).

The results in Table XX first show, for reference, the results of the domain-independent model. The second and third rows correspond to domain-dependent unsupervised training, using the most probable expansion vs. the language model for disambiguation, respectively. The fourth row corresponds to a system that uses a list of domain-dependent expansions, which assumes that a labeler has given the expansion, though EXPN tokens are automatically detected in the same way as in the other unsupervised training conditions. This test shows the potential gain if at least some time is taken to explicitly specify expansions. Finally, for reference, the result of the fully domain-dependent supervised model is given.

The striking fact about these results is that they are significantly better than (supervised) domain-independent models applied to this domain. In other words, while one will certainly do better on text such as classified ads if one is prepared to spend some time labeling data, one can use the unsupervised methods described here on unlabeled data, and still achieve much better performance than one would obtain if one were merely to use an "off-the-shelf" text normalizer designed for "general" text.

Note, finally, that the most common type of mistake the unsupervised model makes is failing to identify an easily pronounceable word as an abbreviation (e.g. *kit* for *kitchen*). This type of mistake does not detract from understanding in spoken output, though it would be problematic for a language model training application.

8.4. Manual evaluation

In this section, we discuss the methodology used in, and results of, manually evaluating the performance of our system relative to other existing systems. Briefly, we examined the output of a normalization system and manually judged the acceptability of the output to estimate token error rates.

8.4.1. Evaluation procedure

The basic procedure for the manual evaluation is as follows. We randomly sample a space-separated token from the unnormalized version of our test set, as described in Section 8.4.2. The normalized text associated with that token and its neighbors are determined by a string alignment procedure described in Section 8.4.3. We then present that space-separated token and surrounding tokens along with 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 were, in brief, that words associated with the original raw token must be complete and correct with no extra tokens present. In addition, 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 often less important.

8.4.2. Random sampling

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 categories, 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.

8.4.3. 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, but this alignment information cannot easily be recovered from all of the systems (and the hand labels) evaluated. 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. Consequently, we developed an algorithm to automatically align raw and normalized text, given a large corpus of normalized text.

The 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 bilingual 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 rearrangement in normalized text, unlike between distinct languages.

A first step is to construct a bilingual dictionary, or list of which words in one language translate to which words in the other language. In our scenario, the dictionary is determined by counts of how often token pairs co-occur in corresponding unnormalized and normalized paragraphs. 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}. While the translation lists have many extraneous

```
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
```

Figure 4. Examples of text alignment in the classifieds ads domain, where "-" indicates a deletion in the normalized text.

TABLE XXI. Token error rates of various systems on various domains as estimated through manual evaluation

	NANTC	Classifieds	pc110	RFR
Festival	0·7% ± 0·7%	34·5% ± 1·8%	5·9% ± 1·1%	4·2% ± 1·0%
LDC	0·2% ± 0·6%	32·4% ± 1·8%	6·3% ± 1·1%	3·0% ± 1·0%
NSW system	0·4% ± 0·7%	12·3% ± 1·5%	3·4% ± 1·0%	2·6% ± 0·9%
Hand-label	0·3% ± 0·6%	09·6% ± 1·4%	2·2% ± 0·9%	2·6% ± 0·9%

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, with costs of 0 for identical tokens, 1 for substitutions between tokens that are "translations" of each other, 3 for insertions and deletions, and 4 for other substitutions. These cost assignments are somewhat arbitrary, but 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 Figure 4.

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, which includes cases that were not actually misaligned but simply corresponded to cases where a token expanded to more than 120 characters when normalized. Thus, the alignment algorithm implemented was entirely adequate for its purpose. When a token could not be judged from 120 characters of context, the whole surrounding paragraph is presented to the adjudicator.

8.4.4. Results

Using the methodology described above, we estimated the token error rate of three different systems: the Linguistic Data Consortium normalization tools, the normalization tools provided with the Festival speech synthesis toolkit, and the full NSW system we developed using

domain-dependent supervised training. We also evaluated the quality of the hand-tagged data described in Section 4. Performance was evaluated separately in each of the four domains considered.

The results of this manual evaluation are presented in Table XXI. We used 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, the NSW system, and the hand-labeled data are not significant.

For each domain, we see that the NSW system 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, the labeling of the classifieds data must be improved. The NSW system is trained on hand-labeled data, and its performance is 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 classified ads 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. The labelers were not asked to provide this information, since it was not within the scope of this study.
- In the recipes domain, the expansion of an abbreviation as plural or singular (e.g. *3g*) is often incorrect. Again, the labelers were not asked to provide this information.

9. 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 *unsupervised* techniques that allow one to build text normalizers for a new domain given only 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.

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 did not afford a significant improvement. 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 cast in terms of features of the context. Class-based language models and feature-based back-off mechanisms (Bikel, Schwartz & Weischedel, 1999) offer possible solutions for the approach based on a source–channel model. Alternatively, previous work such as that of Yarowsky (Yarowsky, 1996) applying decision lists to particular instances of these problems suggests that direct models might be an effective solution, which would likely require maximum entropy techniques. For the unsupervised training paradigm, it may be that cache-based language models are useful.

As we have hinted at various places in the paper, the generation of a lattice of alternatives at each stage of the processing rather than a single answer, would probably improve results. In the system reported here, this was only done at the expansion stage feeding into the language model. But one could imagine having lattices at even earlier stages: for example, the splitter could return a lattice of possible splits, which could then be fed into the classifier, and so forth. This would allow the classifier to influence the decisions made by the splitter, something that is not possible in the system that we actually implemented. As a reviewer has noted, Table XVIII clearly shows that the system is sensitive to classifier and splitter performance, so a means to improve performance by incorporating classifier decisions into the splitter would surely be helpful.

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). In particular the techniques for tokenization, NSW classification, expansion and language modeling should all carry over to many other languages, though of course the training material and (in the case of hand-constructed portions such as number expansion) the rules will be language particular. Even the splitter (Section 6.3), which clearly has many English-particular characteristics, is expected to work with some changes, for other languages that use scripts (such as Latin, Cyrillic or Greek) that distinguish capitalized from non-capitalized letters, and use capitalization roughly as it is used in English.

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 handle 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, Shih, Gale & Chang, 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.) However, see Olinsky and Black (2000) for an extension of some of the work presented in this paper to Chinese.

The work reported here represents not an end, but a beginning: there is substantially more work to be done in the area of text normalization. Towards this end some of the tools and the databases created for this project are, as noted in Section 1, publicly available, and this will hopefully encourage others to improve upon the work we have done here.

The work reported in this paper was supported by the National Science Foundation under Grant No. #IIS-9820687, and carried out at the 1999 Workshop on Language Engineering, Center for Language and Speech Processing, Johns Hopkins University. We would like to thank Fred Jelinek and

the CSLP for hosting this event. Much of the basic labeling and original data preparation was done at University of Edinburgh's Centre for Speech Technology Research.

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.

Finally we thank two anonymous reviewers for *Computer Speech and Language* for useful comments.

References

Allen, J., Hunnicutt, M. S. & Klatt, D. (1987). *From Text to Speech: the MITalk System*. Cambridge University Press, Cambridge.

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

Bikel, D., Schwartz, R. & Weischedel, R. (1999). An algorithm that learns what's in a name. *Machine Learning*, **34**, 221–231.

Black, A., Sproat, R. & Chen, S. (2000). Text normalization tools for the Festival speech synthesis system. `http://festvox.org/nsw`.

Black, A., Taylor, P. & Caley, R. (1999). The festival speech synthesis system. `http://www.cstr.ed.ac.uk/projects/festival.html`.

Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth & Brooks, Pacific Grove, CA.

Brown, P., Cocke, J., Pietra, D. S., Pietra, V. D., Jelinek, F., Lafferty, J., Mercer, R. & Roossin, P. (June 1990). A statistical approach to machine translation. *Computational Linguistics*, **16**, 79–85.

Brown, P., Pietra, S. D., Pietra, V. D., Lai, J. & Mercer, R. (1992). An estimate of the upper bound of the entropy of English. *Computational Linguistics*, **18**, 31–40.

CMU Carnegie Mellon Pronuncing Dictionary. `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`, 1998.

Cannon, G. (1989). Abbreviations and acronyms in English word-formation. *American Speech*, **64**, 99–127.

Carletta, J., Isard, A., Isard, S., Kowtko, J., Doherty-Sneddon, G. & Anderson, A. (1997). The reliability of a dialogue structure coding scheme. *Computational Linguistics*, **23**, 13–31.

Chen, S. & Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, **13**, 359–393.

Flammia, G. (1998). *Discourse segmentation of spoken dialogue: an empirical approach*. PhD Thesis, MIT.

Golding, A. & Roth, D. (1999). A Winnow-based approach to context sensitive spelling correction. *Machine Learning*, **34**, 107–130.

Hirschberg, J. & Nakatani, C. (1996). A prosodic analysis of discourse segmentation in direction-giving monologues. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Jelinek, F. (1997). *Statistical Methods for Speech Recognition*. MIT Press, Cambridge.

Jurafsky, D., Bates, R., Coccaro, N., Martin, R., Meteer, M., Ries, K., Shriberg, E., Stolcke, A., Taylor, P. & Van Ess-Dykema, C. (1997). Switchboard discourse language modeling project final report. In *Summer Research Workshop Technical Reports 30*, Johns Hopkins University, Baltimore, MD.

Kneser, R. & Ney, H. (1995). Improved backing-off for n-gram language modeling. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pp. 181–184.

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

Mohri, M., Pereira, F. & Riley, M. (1997). A rational design for a weighted finite-state transducer library. In *Second International Workshop on Implementing Automata*. pp. 43–53. Ontario, Canada.

Olinsky, C. & Black, A. (2000). Non-standard word and homograph resolution for Asian language text analysis. *ICSLP2000*, Beijing. China.

Quinlan, J. R. (1993). *C4.5 Programs for Machine Learning*. Wadsworth and Brooks, Pacific Grove, CA.

Römer, J. Abkürzungen (1994). *Schrift und Schriftlichkeit/Writing and its Use*, Chapter 135, volume 2, (Hugo Steger and Herbert Ernst Wiegand, eds), pp. 1506–1515. Walter de Gruyter, Berlin.

Rowe, N. & Laitinen, K. (1995). Semiautomatic disabbreviation of technical text. *Information Processing and Management*, **31**, 851–857.

Sproat, R. ed (1997). In *Multilingual Text to Speech Synthesis: the Bell Labs Approach*. Kluwer Academic Publishers, Boston, MA.

Sproat, R. (2000). *A Computational Theory of Writing Systems*. Cambridge University Press, Stanford, CA.

Sproat, R. & Riley, M. (1996). Compilation of weighted finite-state transducers from decision trees. In *34th Annual Meeting of the Association for Computational Linguistics*. pp. 215–222. Association for Computational Linguistics, Santa Cruz, CA.

Sproat, R., Shih, C., Gale, W. & Chang, N. (1996). A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, **22**, 377–404.

Taghva, K. & Gilbreth, J. (June 1995). *Recognizing acronyms and their definitions*, Technical Report 95–03, Information Science Research Institute, University of Nevada. Las Vegas.

Yarowsky, D. (1996). Homograph disambiguation in text-to-speech synthesis. In *Progress in Speech Synthesis*. (J. van Santen, R. Sproat, J. Olive and J. Hirschberg, eds), pp. 157–172. Springer, New York.

(Received 14 March 2001 and accepted for publication 24 May 2001)