Formatting Time-Aligned ASR Transcripts for Readability

Maria Shugrina*
Google Inc.
New York, NY 10011
shumash@google.com

Abstract

We address the problem of formatting the output of an automatic speech recognition (ASR) system for readability, while preserving word-level timing information of the transcript. Our system enriches the ASR transcript with punctuation, capitalization and properly written dates, times and other numeric entities, and our approach can be applied to other formatting tasks. The method we describe combines hand-crafted grammars with a class-based language model trained on written text and relies on Weighted Finite State Transducers (WFSTs) for the preservation of start and end time of each word.

1 Introduction and Prior Work

The output of a typical ASR system lacks punctuation, capitalization and proper formatting of entities such as phone numbers, time expressions and dates. Even if such automatic transcript is free of recognition errors, it is difficult for a human to parse. The proper formatting of the transcript gains particular importance in applications where the user relies on ASR output for information and where information-rich numeric entities (e.g. time expressions, monetary amounts) are common. A good example of such application is a voicemail transcription system. The goal of our work is to transform the raw transcript into its proper written form in order to optimize it for the visual scanning task by the end user. We present quantitative and qualitative evaluation of our system with a focus on numeric entity formatting, punctuation and capitalization (See Fig. 1).

Apart from text, the ASR output usually contains word-level metadata such as time-alignment and confidence. Such quantities may be useful for a variety of applications. Although simple to recover via word alignment after some types of formatting, word-level quantities may be difficult to preserve if the original text has undergone a significant transformation. We present a formal and general augmentation of our WFST-based technique that preserves word-level timing and confidence information during arbitrary formatting.

Raw Transcript:
hi bill it's tracy at around three thirty P M just got an apartment for one thousand three thirty one thousand four hundred a month my number is five five five eight eight eight eight extension is three thirty bye
Our Result:
Hi Bill, it's Tracy at around 3:30 PM, just got an apartment for 1,330 1,400 a month. My number is 555-8888 extension is 330. Bye.

Figure 1: An example of a raw transcript with ambiguous written forms and the output of our formatting system.

The problems of sentence boundary detection and punctuation of transcripts have received a substantial amount of attention, e.g. (Beeferman et al., 1998; Shriberg et al., 2000; Christensen et al., 2001; Liu et al., 2006; Gravano et al., 2009). Capitalization of ASR transcripts received less attention (Brown and Coden, 2002; Gravano et al., 2009), but there has also been work on case restoration in the context of machine translation (Chelba and Acero, 2006; Wang et al., 2006). Our work does not propose competing methods for transcript punctuation and capitalization. Instead, we aim to provide a common framework for a wide range of formatting tasks. Our method extends the approach of Gravano et al. (2009) with a general WFST formulation suitable for formatting monetary amounts, time expressions, dates, phone numbers, honorifics and more, in addition to punctuation and capitalization.

To our knowledge, this scope of the problem has not been addressed in literature. Yet such formatting can have a high impact on transcript readability. In this paper we focus on numeric entity format-

*Thank you to Michiel Bacchiani, Martin Jansche, Michael Riley and Cyril Allauzen for discussion and support.

Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL, pages 198–206,
Los Angeles, California, June 2010.

ting. In general, context independent rules fail to adequately perform this task due to its inherent ambiguity (See Fig. 1). For example, the spoken words "three thirty" should be written differently in these three contexts:

- meet me at **3:30**
- you owe me **330**
- dinner for **three 30** minutes later

The proper written form of a numeric entity depends on its class (time, monetary amount, etc). In this sense, formatting is related to the problem of named entity (NE) detection and value extraction, as defined by MUC-7 (Chinchor, 1997). Several authors have considered the problem of NE value extraction from raw transcripts (Huang et al., 2001; Jansche and Abney, 2002; Béchet et al., 2004; Levit et al., 2004). This is an information extraction task that involves identifying transcript words corresponding to a particular NE class and extracting an unambiguous value of that NE (e.g. the value of the date NE "december first oh nine" is "12/01/2009"). Although relevant, this information extraction does not directly address the problem of proper formatting and ordinarily requires a tagged corpus for training.

A parallel corpus containing raw transcriptions and the corresponding formatted strings would facilitate the solution to the transcript formatting problem. However, there is no such corpus available. Therefore, we follow the approach of Gravano et al. and provide an approximation that exploits readily available written text instead. In section 2 we detail our method, provide a probabilistic interpretation and present a practical formulation of the solution in terms of WFSTs. Section 3 shows how to augment the WFST formulation to preserve word-level timing and confidence. Section 4 presents both qualitative and quantitative evaluation of our system.

2 Method

First, handwritten grammars are used to generate all plausible written forms. These variants are then scored with a language model (LM) approximating probability over written strings. To overcome data sparsity associated with written numeric strings, we introduce numeric classes into the LM. In section 2.1 we give a probabilistic formulation of this approach. In section 2.2 we comment on the handwritten grammars, and in section 2.3 we discuss the class-based language model used for scoring. Section 2.4 provides the WFST formulation of the solution.

2.1 Probabilistic Formulation

The problem of estimating the best written form $\hat{w}$ of a spoken sequence of words s can be formulated as a Machine Translation (MT) problem of translating a string s from the language of spoken strings into a language of written strings. From a statistical standpoint, $\hat{w}$ can be estimated as follows:

$$\hat{w} = \underset{w}{\operatorname{argmax}}\{P(w|s)\} \approx \underset{w}{\operatorname{argmax}}\{P'(s|w)P'(w)\},$$

where $P(\cdot)$ denotes probability, and $P'(\cdot)$ a probability approximation. The probability over written strings $P(w)$ can be estimated by training an n-gram language model on amply available written text. The absence of a parallel corpus containing sequences of spoken words and their written renditions makes the conditional distribution $P(s|w)$ impossible to estimate. An approximation $P'(s|w)$ can be obtained by defining handwritten grammars that generate multiple unweighted written variants for any spoken sequence. For a given s, a collection of grammars encodes a uniform probability distribution across the set of all written variants generated for s and assigns a zero probability to any string not in this set. Such grammar-based modeling of $P(s|w)$ combined with statistical estimation of $P(w)$ takes advantage of prior knowledge, but does not share the disadvantages of rigid, fully rule-based systems.

2.2 Handwritten Grammars

Handwritten grammars $G_1 ... G_m$ are used to generate unweighted written variants for a raw string s. In Gravano's work (Gravano et al., 2009) the generated variants include optional punctuation between every two words and an optional capitalization for every word. Our system supports a wider range of variants, including but not limited to multiple variants of number formatting.

The handwritten grammars can be very restrictive or very liberal, depending on the application requirements. For example, a grammar we use to generate punctuation and capitalization only generates sentences with the first word capitalized. This enforces conventions and consistency, which the best scoring variant could occasionally violate. On the other

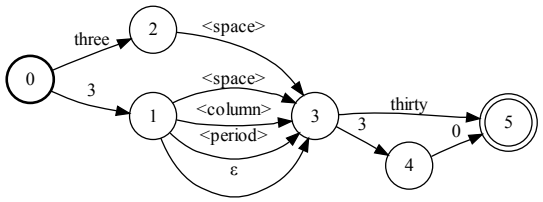


Figure 2: An FSA encoding all variants generated by the number grammar for a spoken string "three thirty".

hand, the grammar for number formatting could be very liberal in producing written variants (See Fig. 2). Jansche and Abney (2002) observe that handwritten rules deterministically tagging numeric strings of certain length as phone numbers perform surprisingly well on phone number NE identification in voicemail. If appropriate to the task, deterministic grammars can be incorporated into the grammar stack. The unweighted written variants generated by applying $G_1 ... G_m$ to s are then scored with the language model.

2.3 Language Model

The probability distribution over written text $P(w)$ can be approximated by a Katz back-off n-gram language model trained on written text in a domain semantically similar to the domain for which the ASR engine is deployed. Unlike some of the approaches used for NE identification (Jansche and Abney, 2002; Levit et al., 2004) and sentence boundary detection (Christensen et al., 2001; Shriberg et al., 2000; Liu et al., 2006), LM-based scoring cannot exploit a larger context than n tokens or prosodic features. The advantage of the LM approach is the ease of applying it to new formatting tasks: no new tagged corpus, and only trivial changes to the preprocessing of the training text would be required.

If the LM is to score written numeric strings, care must be taken in modeling numbers. Representing each written number as a token (e.g. tokens "1,235", "15") during training results in a very large model and suffers from data sparsity even with very large training corpora. An alternative approach of modeling every digit as a token (e.g. "15" is comprised of tokens "1" and "2") fails to model sufficient context for longer digit strings. A partially class-based LM remedies the drawbacks of both approaches, and has been used for tasks such as NE tagging (Béchet et al., 2004). The generalization provided by classes eliminates data sparsity, and is able to model sufficient context.

Class Set A	
Numeric range	*Interpretation*
2-9	single digits
10-12	up to hour in a 12-hour system
13-31	up to the largest day of the month
32-59	up to the largest minute in a time expression
other 2-digit	all other 2-digit numbers
other 3-digit	all 3-digit numbers
1900 - 2099	common year numbers
other 4-digit	all other 4-digit numbers
10000-99999	all 5-digit numbers; e.g. US zip-codes
≥ 100000	all large numbers

Class Set B	
Numeric range	*Interpretation*
0-9	one digit string
10-99	two digit string
...	...
$10^9 - (10^{10} - 1)$	ten-digit string
$\geq 10^{10}$	longer digit string

Table 1: Two sets of number classes used in our system. Each sequence of consecutive digit characters is mapped to the appropriate class. For example, "$1,235.12" would become "⟨dollar⟩ 1 ⟨comma⟩ ⟨num_100_999⟩ ⟨period⟩ ⟨num_10_12⟩" in Class Set A and "⟨dollar⟩ ⟨num_1D⟩ ⟨comma⟩ ⟨num_3D⟩ ⟨period⟩ ⟨num_2D⟩ in Class Set B.

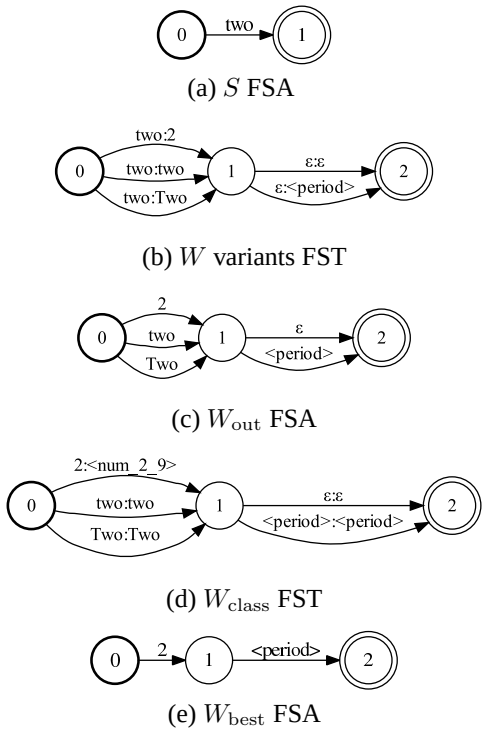
We experiment with two sets of classes (See Table 1). Class Set B, based on (Béchet et al., 2004), marks strings of n consecutive digits as belonging to an n-digit class, assuming nothing about the number distribution. Class Set A is based on intuition about number distribution in text (See Table 1, *Interpretation*). In section 4.4 we show that Class Set A achieves better performance on number formatting. Now that it is established that the choice of classes affects performance, future research could focus on finding an optimal set of number classes automatically. Clustering techniques, often used to derive class definitions from training text, could be applied.

Although more punctuation marks could be considered, we focus on periods and commas. Similarly to Gravano et al. (2009), we map all other punctuation marks in the training text to these two. In many formatting scenarios (e.g. spelled out acronyms, numeric ranges), spaces are ambiguous and significant,

and it is therefore important to consider whitespace when scoring the written variants. Because of this, we model space as a token in the LM.

2.4 WFST Formulation

The one-best[1] ASR output s can be represented by a Finite State Acceptor (FSA) S. We describe a series of standard WFST operations on S resulting in the FSA W_{best} encoding the best estimated formatted variant $\hat{w}$. Current section assumes familiarity with WFSTs; for background see (Mohri, 2009).



(a) S FSA

(b) W variants FST

(c) W_{out} FSA

(d) W_{class} FST

(e) W_{best} FSA

Figure 3: An example showing transducers produced during formatting.

We encode each grammar G_i as an unweighted FST T_i that transduces the raw transcript to its formatted versions. The necessity to encode them as FSTs restricts the set of grammars to regular grammars (Hopcroft and Ullman, 1979), sufficiently powerful for most formatting tasks. The back-off n-gram LM is naturally represented as a weighted deterministic FSA G with negative log probability weights (Mohri et al., 2008). The deterministic mapping of digit strings to number class tokens can also be accomplished by an unweighted transducer K, which passes all non-numeric strings unchanged.

Composing the input acceptor S with the grammar transducers T_i results in a transducer W with all written variants on the output. Projected onto its output labels, W becomes an acceptor W_{out}. W_{class}, the result of the composition of W_{out} with K, has all formatted written variants on the input side and the formatted variants with digit strings replaced by class tokens on the output. The output side of W_{class} can then be scored via composition with G to produce a weighted transducer W_{scored}. The shortest path in the Tropical Semiring on W_{scored} contains the estimate of the best written variant on the input side. This algorithm can be summarized as follows (See Fig. 3):

1. $W = S \circ T_1 \circ T_2 ... \circ T_m$
2. $W_{\text{out}} = \text{Proj}_{\text{out}}(W)$
3. $W_{\text{class}} = W_{\text{out}} \circ K$
4. $W_{\text{scored}} = W_{\text{class}} \circ G$
5. $W_{\text{best}} = \text{Proj}_{\text{in}}(\text{BestPath}(W_{\text{scored}}))$

where $\circ$ denotes FST composition, Proj_{in} and Proj_{out} denote projection on input and output labels respectively, and $\text{BestPath}(X)$ as a function returning an FST encoding the shortest path of X. The key Step 2 ensures that the target written variants are not consumed in the consequent composition operations. For efficiency reasons it is advisable to apply optimizations such as epsilon removal and determinization to the intermediate results. [2]

3 Preserving Word-Level Metadata

We extend the WFST formulation to preserve word-level timing and confidence information.

3.1 Background

A WFST is a finite set of states and transitions connecting them. Each transition has an input label, an output label and a weight in some semiring $\mathbb{K}$. A semiring is informally defined as a touple $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$, where $\mathbb{K}$ is the set of elements, $\oplus$ and $\otimes$ are the addition and multiplication operations, $\overline{0}$ is the additive identity and multiplicative annihilator, $\overline{1}$ is the multiplicative identity (See (Mohri,

[1] This WFST formulation can also be applied to the ASR lattice or n-best list with some modification to the scoring phase.

[2] Our system implements proper failure transitions available in the OpenFST Library (Allauzen et al., 2007).

2009)). By defining new semirings we can use standard FST operations to accomplish a wide range of goals.

3.2 Timing Semiring

In order to formulate time preservation within the FST formalism, we define the *timing semiring* $\mathbb{K}_t$ where each element is a pair (s, e) that can be interpreted as the start and end time of a word:

$$W_t = \left\{(s, e) : s, e \in \mathbb{R}^+ \cup \{0, \infty\}\right\}$$

$$(s_1, e_1) \oplus (s_2, e_2) = (max(s_1, s_2),\ min(e_1, e_2))$$

$$(s_1, e_1) \otimes (s_2, e_2) = (min(s_1, s_2),\ max(e_1, e_2))$$

$$\overline{0} = (0, \infty) \qquad \overline{1} = (\infty, 0)$$

Intuitively, the addition operation takes the largest interval contained by both operand intervals, while multiplication returns the smallest interval fully containing both operand intervals. [3] This definition fulfills all the semiring properties as defined in (Mohri, 2009). Note that encoding only the duration of each word is not sufficient, as there may be time gaps between the words due to the segmentation of the source audio. Let $\tilde{S}$ denote the Weighted Finite State Acceptor (WFSA) encoding the raw ASR output with the start and end time stored in the weight of each arc.

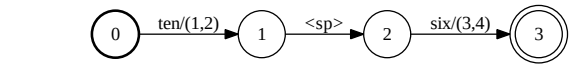
In order to preserve word-level confidence in addition to timing information, a Cartesian product of $\mathbb{K}_t$ and the Log semiring can be used to store both time and confidence in an arc weight.

3.3 Weight Synchronization

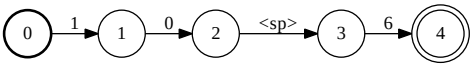
The goal is to associate the timing/confidence weights of $\tilde{S}$ with the word labels of W_{best}, the best formatted string (See Sec. 2.4). Because the weight of each transition in $\tilde{S}$ already expresses the timing/confidence corresponding to its word label, it is sufficient to associate the labels of $\tilde{S}$ with the labels of W_{best}. This is equivalent to identifying the output labels to which each input label is transduced during Step 1 in section 2.4. However, in general WFST operations may desynchronize input and output labels and weights, as the FST structure itself does not indicate a semantic correspondence between them. To alleviate this, we guarantee such a correspondence in our grammars by enforcing that for all paths in any grammar FST T_i:

- an input label appears before any of the corresponding output labels, and
- output labels corresponding to a given input label appear before the next input label.

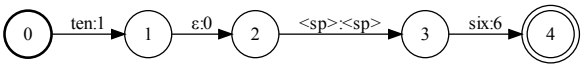
In practice, these assumptions are usually met by handwritten grammars. Even if these assumptions are violated for a small number of paths, only small word-level timing discrepancies will be incurred. Each path in W can be thought of as a sequence of subpaths with only the first transition containing a non-ϵ input label. We say that the input label of each such subpath corresponds to that subpath's output labels.



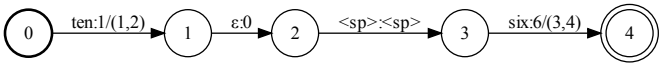
(a) $\tilde{S}$ FSA



(b) W_{best} FSA



(c) $W_{\text{raw:best}}$ FST



(d) $\tilde{W}_{best}$ FST

Figure 4: A small example of time preservation section of the algorithm. Arcs with non-unity timing weights show parenthesized pair of start and end time.

The best path that has input labels corresponding to the raw ASR output can be obtained by composing the variants FST W with the best formatted FSA W_{best} and picking any path. The timing weights are restored to by composing the weighted $\tilde{S}$ with this result. To preserve timing we add two more steps to Steps 1–5 in section 2.4:

6. $W_{\text{raw:best}} = \text{RmEps}(\text{AnyPath}(W \circ W_{\text{best}}))$
7. $\tilde{W}_{\text{best}} = \tilde{S} \circ \text{Map}_{\text{t}}(W_{\text{raw:best}})$

where $\text{RmEps}(X)$ applies the epsilon-removal algorithm to X (Mohri, 2009), and $\text{Map}_{\text{t}}(X)$ maps

[3] Note that this is just a Cartesian product of min-max and max-min semirings. The elements of $\mathbb{K}_t$ are not proper intervals, as it is possible for s to exceed e.

all non-zero weights of X to the unity weight in the *timing semiring*. Because $\tilde{S}$ is an epsilon-free acceptor, the result $\tilde{W}_{\text{best}}$ will contain the original weights of $\tilde{S}$ on the arcs with the corresponding input labels (See Fig. 4 for an example). The space-delimited words and the corresponding weights can then be read off by walking $\tilde{W}_{\text{best}}$.

4 Evaluation

Section 4.1 presents our datasets and an evaluation metric specific to number formatting, and section 4.2 describes our experimental system. We present quantitative evaluation of capitalization/punctuation performance and number formatting performance separately in sections 4.3 and 4.4. Because the ultimate goal of our work is to improve the readability of ASR transcripts, we also present the result of a user study of transcript readability in section 4.5.

4.1 Data and Metrics

The training corpus contains 185M tokens of written text normalized to contain only comma and period punctuation marks. A set of 176M tokens (TRS) is used for training and a set of 7M tokens (PTS) is held back for testing punctuation and capitalization (See Table 3). To obtain a test input (NPTS) for our system, PTS is lowercased and all punctuation is removed.

	words	commas	periods	capitals
TRS	176M	10.6M	11.8M	24.3M
PTS	7M	420K	440K	880K

Table 3: Training set TRS and test set PTS.

Number formatting is evaluated on a manually formatted test set. We manually processed the set of raw manual transcripts (NNTS) from the LDC Voicemail Part I training set (Padmanabhan et al., 1998) to obtain a reference number formatting set (NTS). All numeric entities in NTS were formatted according to the following conventions:

- all quantities under 10 are spelled out
- time is written in a 12-hour system as "xx:xx" or "xx"
- dollar amounts are written as "$x,xxx.xx" with cents included if spoken
- US phone numbers are written as "(xxx) xxx-xxxx" or "xxx-xxxx"
- other phone numbers are written as digit strings
- decimals are written as "x.x"
- large amounts include commas: "x,xxx,xxx"

All contiguous sequences of words in NTS that could be a target for number formatting were marked as *numeric entities*, whether or not these words were formatted by the labeler (for example "six" is a *numeric entity*). To evaluate number formatting performance, we process NNTS with our full experimental system, then remove all capitalization and inter-word punctuation. This result is aligned with NTS, and each entity is scored separately as totally correct or totally incorrect (See Table 2), yielding:

$$\text{Numeric Entity Error Rate} = 100 \cdot \frac{I}{N}$$

where I is the count of entities that did not match the reference entity string exactly and N is the total entity count. This error rate is independent of the numeric entity density in the test set. The errors are broken down into three types:

- incorrect formatting - when the system incorrectly formats an entity that is formatted in the reference
- overformatting - when the system formats an entity that stays unformatted in the reference
- underformatting - when the system does not format an entity formatted in the reference

Out of 1801 voicemail transcripts in NTS, 1347 contain at least one entity for a total of 3563 entities, signifying a frequent occurrence of numeric entities in voicemail. There is an average of 7 raw transcript words per entity, suggesting that in many cases entity formatting is non-trivial.

4.2 Experimental System

The experimental system includes a 5-gram LM trained on TRS with spaces treated as tokens. Number evaluation is performed with two sets of number classes, listed in Table 1. System A contains LM with classes from set A, and System B contains LM with classes from set B. The experimental setup also includes the following grammars:

- G_{phone} - deterministically formats as a phone number any string spoken like a US 7 or 10 digit phone number
- G_{number} - expands all spoken numbers to a full range of variants, with support for time expressions, ordinals, decimals, dollar amounts
- $G_{\text{cap\_punct}}$ - generates all possible combinations of commas, periods and capitals; always capitalizes the first word of a sentence

Raw:	for	**six**	people at	**five five thirty**	cost is	**eleven hundred dollars**
Ref:	for	**six**	people at	**5 5:30**	cost is	**$1,100**
Hyp:	for	**6**	people at	**5 5:30**	cost is	**11 $100**
Score:	-	incorrect	-	correct	-	incorrect

Table 2: A example of a raw transcript, reference transcript with number formatting and the hypothesis produced by the system. The entities (bold) in reference and hypothesis are aligned and scored.

4.3 Evaluation of Punctuation

To evaluate the performance of capitalization and punctuation we run System A on NPTS with only the $G_{\text{cap\_punct}}$ (in order not to introduce errors due to numeric formatting). The precision, recall and F-measure rates for periods, commas and capitals are computed using PTS as reference (See Fig. 5).

	Precision	Recall	F-Measure
Capitals	0.7902	0.5356	0.6385
Comma	0.5527	0.3129	0.3996
Period	0.6672	0.6783	0.6727

Figure 5: Punctuation and capitalization results.

It should be noted that a 5-gram language model that treats spaces as words models the same history as a 3-gram model that omits the spaces from training data. When this is taken into account, our results with a much smaller training set are comparable to Gravano et al. (2009). The F-measure scores for commas and periods are also comparable to the prosody-based work of (Christensen et al., 2001), with the precision of the period slightly lower, but compensated by recall. Thus, our system can perform additional formatting, while retaining a reasonable capitalization and punctuation performance.

4.4 Evaluation of Number Formatting

We evaluate number formatting performance of Systems A and B, which use different sets of classes for the language modeling (See Table 1). We process NNTS with both systems and score against the reference formatted set NTS to obtain Numeric Entity Error Rate (NEER). Class Set B naively breaks numbers into classes by digit count. System B using this class set performs worse than System A by 1.7% absolute (See Table 4). In particular, the overformatting rate (OFR) is higher by 1.2% absolute in System B than in System A. An example of overformatting is the mis-formatting of the English impersonal pronoun "one" as the digit "1". Such overformatting errors are much more noticeable than the underformatting errors, which are higher by 0.4% absolute in System A. This result shows that the choice of classes for the class-based LM significantly impacts number formatting performance. Superior overall performance of System A suggests that prior knowledge in the choice of classes favorably impacts performance.

	NEER	IFR	OFR	UFR
System A				
exact	16.1%	9.7%	5.4%	1.0%
ignore space	11.2%	4.9%	5.4%	1.0%
System B				
exact	17.8%	10.6%	6.6%	0.6%
ignore space	13.2%	6.0%	6.6%	0.6%

Table 4: The total NEER score, NEER due to incorrect formatting (IFR), NEER due to overformatting (OFR) and NEER due to underformatting (UFR); NEER rates with whitespace errors ignored are also listed.

In order to estimate the error rate not caused by whitespace errors, we also compute the NEER with whitespace errors ignored. It turns out that between 4 and 5% absolute of the errors are whitespace errors. Even if all whitespace errors are significant, the 83.9% of perfectly formatted entities suggests that the proposed formatting approach can achieve good performance on the number formatting task.

	entities	:	.	$	,
Reference totals	3563	310	50	39	17
System A correct	3161	232	36	33	10
System B correct	2923	204	35	31	5

Table 5: The count of formatted entities in NTS containing various formatting characters; the counts of these entities correctly formatted by the systems A and B.

To estimate how well the systems perform on specific number formatting tasks we count the number of reference entities containing certain formatting characters and compute the number of these entities correctly formatted by Systems A and B (See Table 5). The count of different formatting characters in NTS is small, but still provides an estimate of the number formatting performance for a real appli-

cation like voicemail transcription. System A performs significantly better on the formatting of time expressions containing a colon, getting 74.8% correct. The NEER of System A for entities containing special formatting characters is under 28% for all formatting characters except comma, which is used inconsistently in training text.

4.5 Qualitative Evaluation

In addition to quantitative evaluation we have conducted a small-scale study of transcript readability. The study aims to compare raw ASR transcripts, ASR transcripts formatted by our system and raw manual transcripts. We have processed LDC Voicemail Part 1 with our ASR engine achieving an error rate of 30%, and have selected 50 voicemails with error rate under 30% and high informational content. Messages containing names, addresses and numbers were preferred. The word error rate on the selected voicemails is 20%. For each voicemail we have constructed three semantic multiple-choice questions, aimed at information extraction. We have asked each of 15 volunteers to answer all 3 questions about half of the voicemails. The questions were shown in sequence, while the transcript remained on the screen. The transcript for each voicemail was randomly selected to be ASR raw, ASR formatted or manual raw. The response time was measured individually for each question.

The analysis of the responses reveals a statistically significant difference in response time between formatted and raw ASR transcripts ($p = 0.02$, even allowing for per-item and per-subject effects; see also Fig. 6) and comparable accuracy. The response times for formatted ASR were comparable to the response times for manual unformatted transcripts. This suggests that for transcripts with low error rates the formatting of the ASR output significantly impacts readability. This disagrees with a similar study (Jones et al., 2003), which found no significant difference in the comprehension rates between raw ASR transcripts and capitalized, punctuated ASR output with disfluencies removed. This could be due to a number of factors, including a different type of transformation performed on the ASR transcript, a different corpus, and a lower word error rate of transcripts in our user study.

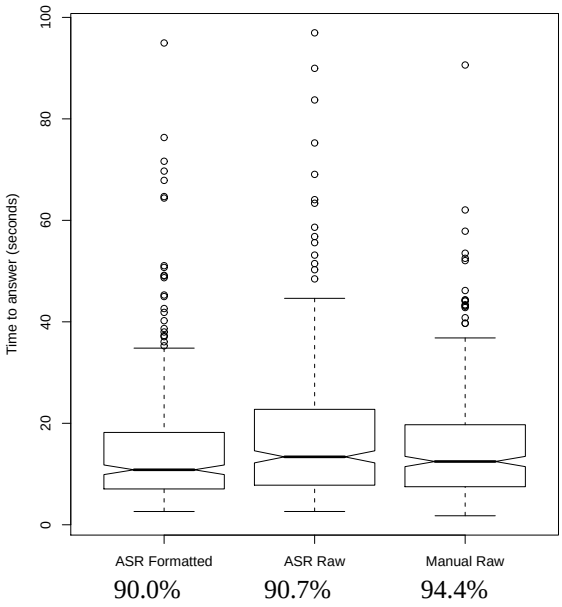


Figure 6: The standard R box plot of the response time for different transcript types and the corresponding accuracy.

5 Conclusion

We present a statistical approach suitable for a wide range of formatting tasks, including but not limited to punctuation, capitalization and numeric entity formatting. The average of 2 numeric entities per voicemail in the manually processed LDC Voicemail corpus shows that number formatting is important for applications such as voicemail transcription. Our best system achieves a Numeric Entity Error Rate of 16.1% on the ambiguous task of numeric entity formatting, while retaining capitalization and punctuation performance comparable to other published work. Our algorithm is concisely formulated in terms of WFSTs and is easily extended to new formatting tasks without the need for additional training data. In addition, the WFST formulation allows word-level timing and confidence to be retained during formatting. In order to overcome data sparsity associated with written numbers, we use a class-based language model and show that the choice of number classes significantly impacts number formatting performance. Finally, a statistically significant difference in question answering time for raw and formatted ASR transcripts in our user study demonstrates the positive impact of the transcript formatting on the readability of errorful ASR transcripts.

References

C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. 2007. Openfst: A general and efficient weighted finite-state transducer library. In *Proceedings of CIAA*, pages 11–23.

F. Béchet, A. Gorin, J. Wright, and D. Hakkani-Tür. 2004. Detecting and extracting named entities from spontaneous speech in a mixed-initiative spoken dialogue context: How may i help you? *Speech Communication*, 42(2):207–225.

D. Beeferman, A. Berger, and J. Lafferty. 1998. Cyberpunc: A lightweight punctuation annotation system for speech. In *Proceedings of ICASSP*, pages 689–692.

E. Brown and A. Coden. 2002. Capitalization recovery for text. In *Information Retrieval Techniques for Speech Applications*, pages 11–22, London, UK. Springer-Verlag.

C. Chelba and A. Acero. 2006. Adaptation of maximum entropy capitalizer: Little data can help a lot. *Computer Speech and Language*, 20(4):382–399.

N. Chinchor. 1997. Muc-7 named entity task definition. In *Proceedings of MUC-7*.

H. Christensen, Y. Gotoh, and S. Renals. 2001. Punctuation annotation using statistical prosody models. In *ISCA Workshop on Prosody in Speech Recognition and Understanding*.

A. Gravano, M. Jansche, and M. Bacchiani. 2009. Restoring punctuation and capitalization in transcribed speech. In *Proceedings of ICASSP*, pages 4741–4744. IEEE Computer Society.

J. Hopcroft and J. Ullman, 1979. *Introduction to automata theory, languages, and computation*, pages 218–219. Addison-Wesley.

J. Huang, G. Zweig, and M. Padmanabhan. 2001. Information extraction from voicemail. In *Proceedings of the Conference of the ACL*, pages 290–297.

M. Jansche and S. P. Abney. 2002. Information extraction from voicemail transcripts. In *In EMNLP*.

D. Jones, F. Wolf, E. Gibson, E. Williams, E. Fedorenko, D. Reynolds, and M. Zissman. 2003. Measuring the readability of automatic speech-to-text transcripts. In *Proceedings of EUROSPEECH*, pages 1585–1588.

M. Levit, P. Haffner, A. Gorin, H. Alshawi, and E. Nöth. 2004. Aspects of named entity processing. In *Proceedings of INTERSPEECH*.

Y. Liu, E. Shriberg, A. Stolcke, D. Hillard, M. Ostendorf, and M. Harper. 2006. Enriching speech recognition with automatic detection of sentence boundaries and disfluencies. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1526–1540.

M. Mohri, F. Pereira, and M. Riley. 2008. Speech recognition with weighted finite-state transducers. In *Handbook on Speech Processing and Speech Communication*. Springer.

M. Mohri. 2009. Weighted automata algorithms. In *Handbook of Weighted Automata. Monographs in Theoretical Computer Science.*, pages 213–254. Springer.

M. Padmanabhan, G. Ramaswamy, B. Ramabhadran, P. Gopalakrishnan, and C. Dunn. 1998. Voicemail corpus part i. Linguistic Data Consortium, Philadelphia.

E. Shriberg, A. Stolcke, D. Hakkani-Tür, and G. Tür. 2000. Prosody-based automatic segmentation of speech into sentences and topics. *Speech Communications*, 32(1-2):127–154.

W. Wang, K. Knight, and D. Marcu. 2006. Capitalizing machine translation. In *Proceedings of HLT/ACL*, pages 1–8. Association for Computational Linguistics.