IMPROVED LANGUAGE MODELING FOR CONVERSATIONAL APPLICATIONS USING SENTENCE QUALITY

Mark Epstein, Bhuvana Ramabhadran, Rajesh Balchandran

Human Languages Technology Department
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

ABSTRACT

In this paper, we propose a new approach to build language models for conversational systems using a corpus of text as opposed to a live or a Wizard-of-Oz collection. Each sentence in the corpus is assigned a “quality” that reflects the developer’s intuition for how likely that sentence is to be spoken by a real user to the live system. Language Models (LM) are built for each sentence quality, and these are subsequently interpolated to produce the final model. We also have built a classifier that assigns sentence qualities to the data, and whose subsequent language models achieve similar improvements in word and turn error rate.

Index Terms— speech recognition, natural language interfaces

1. INTRODUCTION

Statistical Language Models (LM) for automatic speech recognition systems are usually trained using a corpus of sentences collected by the LM designers. The larger the corpus and the more the corpus resembles the actual sentences real users will subsequently speak to the system, the better the language model will be [1]. For most Automated Speech Recognition (ASR) and Natural Language Understanding (NLU) applications, it is possible to use text-based techniques to assemble the language model corpus. It is hard to collect data to build the initial models without having the initial model already built [2].

There are many approaches to bootstrapping a prototype application with minimal initial user data. One could search the web [3], [4], [5] to find appropriate sentences. Another approach is to start from an out-of-domain corpus and modify words and phrases to morph the corpus into an in-domain corpus [2]. Both of these approaches will collect lots of noisy data, which require filtering to improve the quality of the data. Filters have been proposed using maximum entropy models [6], Bleu [3], and user simulation [2].

In our approach, we also use a text-based data collection to which we apply a filter. Users are provided scenarios corresponding to various application functions and are asked to give a few textual sentences for each scenario. Each of these sentences is then assigned a “sentence quality” that reflects the importance that the application correctly process the sentence. This helps solve many of the problems associated with text-based data collection: mismatches between typed-in and spoken commands, data that regurgitates n-grams in the the colletion scenarios, and in-domain but out-of-scope commands. By weighting the “good” sentences higher in the final language model, we achieve a reduction in perplexity, word-error rate, and turn error rate.

2. SENTENCE QUALITY

Consider these sentences given by users visiting our web page:

- Make the volume louder
- Audio level is too low
- Need to volume to be higher
- Can you raise the volume of the TV by three to seven

We use these examples to explain the rationale behind tagging sentences with a sentence quality value. The first sentence is natural and direct and the ASR/NLU system is expected to decode it correctly. Such a sentence is tagged as $Q_1$ (highest quality/importance). The second sentence is less direct and less likely to be spoken by an actual user. These sentences are tagged as $Q_2$ (slightly lower quality/importance). The third example is unlikely to be spoken, so is rated $Q_3$. The last example is so unlikely to ever be spoken that it is rated $Q_4$. While most of the N-grams in this sentence are useful, there are several that could bias the conditional probabilities “away” from more likely N-grams. As shown in Section 4.1, $Q_4$ sentences do not occur that often.

Many factors impact sentence quality: sentence length, directness, grammaticality, awkwardness, frequency of N-grams, and the appropriateness of the response to the state of the dialog system. As a first step in understanding the impact of sentence quality on ASR/NLU performance, we manually assigned these four quality levels to our training data, and use these to build the language model, as described in the next section. Annotators were told to rate a sentence as $Q_1$...$Q_3$ based on whether a system “must”, “should”, or “hopefully” recognize this sentence if spoken by a user. They were also told that if they would “definitely”, “likely”, “possibly” or “never” speak this sentence, then that could also be used to judge how to assign a sentence quality. Syntactic, semantic, or lexical considerations all impact the sentence quality. Most of the data was annotated by one annotator as part of the first DICIT prototype (section 4). During the annotation of data for the second prototype, 20% of the qualities changed. This indicates that ambiguity and subjectivity exist when assigning sentence qualities.

3. CONSTRUCTING LANGUAGE MODELS USING SENTENCE QUALITY

We build our language model using techniques learned from topic-based language models [7], [8]. We try two different approaches mentioned in [9]:

- Interpolated Model - We build three individual language models $LM_{Q1}$, $LM_{Q2}$, and $LM_{Q3}$ from data for each sen-
tence quality, and interpolate:

\[ LM = \lambda_1 LM^Q_1 + \lambda_2 LM^Q_2 + \lambda_3 LM^Q_3 \]  

(1)

where, \( \lambda_1, \lambda_2 \) and \( \lambda_3 \) are interpolation weights and \( \lambda_1 + \lambda_2 + \lambda_3 = 1 \).

- **Count Merged Model** - We build a single LM by using all data, but after scaling the count for each N-gram, by quality-specific scale factors \( s_Q \), as shown below:

\[
p(w_n | w_{n-1}, w_{n-2}) = \frac{\sum_{Q=1}^3 s_Q \cdot c_Q(w_n, w_{n-1}, w_{n-2})}{\sum_{Q=1}^3 s_Q \cdot c_Q(w_{n-1}, w_{n-2})}
\]

(2)

where, \( w_n \) is the \( n^{th} \) word, \( p \) is the conditional probability, \( c_Q \) is the N-gram count for the specified N-gram.

Interpolated models are commonly used for combining topic-specific language models [7], [10], [11], whereas count-merged models are more suited when all data is from a single domain. The counts for the lower-order models overlap and this impacts most smoothing algorithms [12]. In addition, data with examples of N-grams with high count occurring for only one quality level, might be poorly modeled by an interpolated model if \( \lambda \) is very low for that quality level. A count-merged model may better represent this type of data.

4. THE DICIT APPLICATION

The DICIT system [13] provides a conversational interface to enable users to interact with a television without using a close-talking microphone. It employs far-field microphone array and echo cancellation technologies to enable voice interaction from a distance [14]. The presence of residual TV noise in the audio makes it a very challenging task for speech recognition. The DICIT application supports natural-language speech recognition and dialog management.

In this application, there are two main dialog states. The first dialog state defines the state when the user is watching a program, and is called the TV-Screen state. While in this state, the user can change the channel, change the volume, or request the electronic program guide (EPG) from which they can browse and select a program. The second dialog state is the EPG state, during which the user can browse the program guide, apply filter criteria to search the guide, and eventually select the show or channel they want to watch.

We only present results for the TV-Screen model. The results for the EPG model were similar.

4.1. Text-based Data Collection

The training data was collected using a web-application. Users were presented with a sample TV screen shots for 90 different scenarios, and were asked to give sentences they would “speak” if this were a conversational system. In total, 21,069 sentences were collected and used for training language models. The test set was collected from non-native European speakers of English at various DICIT partner sites (Amuser, FBK, and Elektrobit) using actual prototype systems. The test set comprises of 609 sentences.

The training sentences were annotated with named entities. Sentences like “I want to watch CNN” and “I want to watch BBC” both map to the same classed sentence “I want to watch CHANNEL-NAME”. The named entities are modeled as grammars embedded within the N-gram language model. The words within each embedded grammar are implemented by probabilistic finite state grammars.

The N-gram language model score and the embedded grammar probabilities are multiplied together to compute the Viterbi LM score for a specific sentence [15], [16].

Of the 21,069 training sentences (Total Training), 2497 were provided by developers of the DICIT application (Training D) and 18,572 were obtained from non-developers (Training ND). The distribution of sentence quality values for each of these data sources and for the test data is shown in Table 1. It can be seen that the distribution of sentence quality in the test set closely matches the data given by the developers and doesn’t match the data given by non-developers. The test set has an artificially high percentage of Q4 sentences as many of the far-field DICIT recordings were incomplete as they were chopped midway by the acoustic front-end. The number of “truly” (untruncated) Q4 sentences is approximately 1%.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Sentence Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1 (%)</td>
</tr>
<tr>
<td>Training Total</td>
<td>39.2</td>
</tr>
<tr>
<td>Training D</td>
<td>64.2</td>
</tr>
<tr>
<td>Training ND</td>
<td>35.8</td>
</tr>
<tr>
<td>Test</td>
<td>62.4</td>
</tr>
</tbody>
</table>

5. RESULTS (MANUAL QUALITY TAGGING)

We built several different trigram models, smoothed using Kneser-Ney-Mod smoothing [12]. The test utterances were decoded using IBM’s Embedded ViaVoice. The NLU system is a maximum entropy multistage action classifier that assigns one of 21 different meanings (actions) to the whole sentence [17]. We report results for perplexity, WER, and Turn Error Rate (TER). The TER measures the accuracy of the NLU action on the decoded sentence. For an utterance to be correct, all the named entities found by the ASR and the most likely NLU meaning have to match. This is a strict interpretation of turn correctness; no “partial credit” is given for near-misses.

5.1. Baseline Experiments

We first built trigram baseline models for the DICIT TV-Screen state using 85% of the collected data for training, and 15% for smoothing. This model has a perplexity of 17.3, a WER of 26.4%, and a TER of 22.3%. We also built a baseline model using all the training to train the language model and half the test data as smoothing data. This improves the results to a perplexity of 16.8, a WER of 25.3%, and a TER of 21.4%. As expected, using test data for smoothing improves results. There are other adaptation techniques to adapt an LM built from text-based collection methods to realistic data spoken by actual users. However, given our limited test data size and goal to improve on text-collected data, we did not pursue these techniques.

5.2. Interpolated Model Experiments

We built separate language models for Q1, Q2, and Q3 using 85% of the data for each sentence quality as training and 15% as held out smoothing. We interpolated these three models in two ways. We first used an empirical approach, varying \( \lambda_1, \lambda_2 \) and \( \lambda_3 \) from 0 to 1.0 in steps of 0.05, subject to the constraint that \( \sum_{i=1}^3 \lambda_i = 1 \). Figure 1 shows the turn error rate plotted for the interpolation weights \( \lambda_1 \) and \( \lambda_2 \). Since \( \lambda_3 = 1.0 - \lambda_1 - \lambda_2 \), the closer you move to the origin or either axis, the larger \( \lambda_3 \) is, and in general, the larger the...
TER. The results for several models are shown Table 2. The model that uses interpolation weights that most closely matches the prior distribution of the training data are shown in the line “Train Prior”, of the developer training data are in the line “Dev. Prior”, and of the test data are in the line “Test Prior”. The results that minimize the perplexity were computed using the test data in a two-way cross-validation. The model that uses the easily computed “Dev. Prior” gives a 1.8% improvement in WER and a 3.4% improvement in TER, but has a 10% worse perplexity. The worst model (from Figure 1), not surprisingly, uses $\lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 1.0$. The best relative improvement of WER was 5.7% and TER was 5.3% and use $\lambda_{Q_1} = 0.75$ and $\lambda_{Q_2} = 0.1$ or 0.15 respectively. These are not shown in the table, as the $\lambda$’s were determined by looking at the best experimental test results.

![Fig. 1. TV-Screen Interpolated Turn Error Rate](image)

### Table 2. Interpolated Model Results

<table>
<thead>
<tr>
<th>Model</th>
<th>$\lambda_1/\lambda_2/\lambda_3$</th>
<th>WER</th>
<th>TER</th>
<th>Perp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Prior</td>
<td>0.4/0.35/0.25</td>
<td>25.4%</td>
<td>22.3%</td>
<td>19.1</td>
</tr>
<tr>
<td>Dev. Prior</td>
<td>0.65/0.35/0.0</td>
<td>25.9%</td>
<td>21.6%</td>
<td>19.0</td>
</tr>
<tr>
<td>Test Prior</td>
<td>0.65/0.3/0.05</td>
<td>25.6%</td>
<td>21.7%</td>
<td>19.1</td>
</tr>
<tr>
<td>Min. Perp.</td>
<td>0.62/0.14/0.24</td>
<td>25.0%</td>
<td>21.8%</td>
<td>17.1</td>
</tr>
</tbody>
</table>

The row corresponding to “Test Prior” shows the results that could be obtained had the quality distribution of the test sentences been known. One could also use the test data in an unsupervised fashion to learn the interpolation weights (such as in a two-pass approach), and hopefully get gains closer to the best accuracies attained.

### 5.3. Count-Merged Model Experiments

In the count-merged experiments, the training and smoothing counts for each sentence quality were accumulated separately, and then scaled. Integer scale factors were used from 0 to 20, constrained to sum to 20. This impacts the Kneser-Ney-Mod smoothing, as separate discounts are used for counts 1 and 2 [12]. Figure 2 illustrates the results.

Results for the models are shown in Table 3. By merging counts, the perplexity, WER and TER gains also improve upon the baseline, and are slightly better than the interpolated models. The scale factors shown scale the number of sentences used for each quality so the percentage of each quality most closely matches the specific prior percentage. The best WER improves the baseline by 6.2% and uses scale factors $s_{Q_1} = 9$, $s_{Q_2} = 5$ and $s_{Q_3} = 6$. The best TER improves the baseline by 8.7% and uses scale factors $s_{Q_1} = 15$, $s_{Q_2} = 4$ and $s_{Q_3} = 1$. These are not shown in the table. We plan to explore means to learn these scale factors as a constrained optimization problem which minimizes the WER or TER directly.

![Fig. 2. TV-Screen Weighted Turn Error Rate](image)

### Table 3. Count Merged Model Results

<table>
<thead>
<tr>
<th>Model</th>
<th>$s_{Q_1}/s_{Q_2}/s_{Q_3}$</th>
<th>WER</th>
<th>TER</th>
<th>Perp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Prior</td>
<td>7/6/7</td>
<td>25.6%</td>
<td>22.0%</td>
<td>17.7</td>
</tr>
<tr>
<td>Dev. Prior</td>
<td>13/7/0</td>
<td>25.5%</td>
<td>21.1%</td>
<td>19.2</td>
</tr>
<tr>
<td>Test Prior</td>
<td>12/6/2</td>
<td>25.5%</td>
<td>21.4%</td>
<td>17.0</td>
</tr>
<tr>
<td>Min. Perp.</td>
<td>12/4/4</td>
<td>25.4%</td>
<td>20.9%</td>
<td>16.7</td>
</tr>
</tbody>
</table>

### 6. CLASSIFYING SENTENCE QUALITY STATISTICALLY

One obstacle in using sentence qualities in practice is that the training data needs to be manually tagged with sentence quality values. For this approach to be more practical and widely accepted, it is desirable to have an automatic method of assigning sentence quality values. By creating a feature vector for each training sentence using features that help discriminate sentence quality, one could then cluster the vectors. Since we manually annotated our data for the baseline experiments, in this paper we report results using a maximum-entropy classifier. In the future, we plan to try various clustering approaches and compare them to our classifier.

Even though there are only 4 quality values to choose from, given the significant overlap between sentences, it is quite challenging to come up with a consistent and accurate classification. Even among humans, there can be disagreement on what constitutes Q1 vs Q2 or Q2 vs Q3 (20% of the qualities assigned during the building of prototype one were later changed).

Building an automatic classifier to assign sentence qualities is similar to that of topic-clustering for which there has been considerable research [18], [19], [20], [21]. There are two significant differences. First, the training data is from a single domain, with significant overlap between words and n-grams in each quality. Second, we are using a small corpus with only about 20K sentences.

Selecting a good set of features to learn the sometimes subtle (and capricious) differences that humans use when assigning sentence qualities is rather challenging. We noticed that humans, when assigning qualities, tended to use heuristics like sentence length, the number of embedded grammars in the sentence, grammaticality, the presence of a rare but important n-gram, the presence of an “awkward” n-gram, and so forth. We capture the concept of “important” and “awkward” n-grams by examining the n-gram and feature weights of the LM and NLU baseline models. This leads to 3 classes of features: lexical, lm-score, and nlu-weight. The feature vector has 38 different dimensions. In order to facilitate making our maximum-
entropy model, we quantized each dimension into 5-20 values. This led to 431 binary features.

7. RESULTS (AUTOMATIC QUALITY TAGGING)

We ran the classifier using 4-way cross validation to annotate the training data with sentence qualities. The automatic classifier annotated the training corpus with 77.4% accuracy. An automatic classifier built with all training data annotated the held-out test set with 61% accuracy. 60.6% of the data was assigned the training corpus with 77.4% accuracy. An automatic classifier annotated the training corpus with 77.4% accuracy. An automatic classifier annotated the training corpus with 77.4% accuracy. An automatic classifier annotated the training corpus with 77.4% accuracy.

We then built language models following the procedures listed in Sections 3 and 5. The models built from the automatically classified data are slightly better than those built using human-assigned sentence quality, as shown in Table 4. The model using the weights that best matched the developer-given prior reduced the WER by 2.6% and the TER by 3.4% relative to the baseline. The count-merged model with the best WER and TER reduced the baseline WER by 7.6% and the baseline TER by 9.7%, which is also slightly better than the best models built from human-assigned quality. We believe that automatically classified data leads to better LM and NLU models because of the LM features. 15 of the 16 highest scoring weights in our trained classifier used one of the LM features.

Table 4. Auto-Classified Best Model Results

<table>
<thead>
<tr>
<th>Model</th>
<th>$s_{Q1} / s_{Q2} / s_{Q3}$</th>
<th>WER</th>
<th>TER</th>
<th>Perp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Prior</td>
<td>7 / 6 / 7</td>
<td>24.8%</td>
<td>21.2%</td>
<td>17.6%</td>
</tr>
<tr>
<td>Dev. Prior</td>
<td>13 / 7 / 0</td>
<td>25.7%</td>
<td>21.6%</td>
<td>18.5%</td>
</tr>
<tr>
<td>Test Prior</td>
<td>12 / 6 / 2</td>
<td>24.8%</td>
<td>20.8%</td>
<td>17.5%</td>
</tr>
<tr>
<td>Min. Perp.</td>
<td>10 / 5 / 5</td>
<td>25.5%</td>
<td>21.4%</td>
<td>17.4%</td>
</tr>
</tbody>
</table>

8. CONCLUSION

Using sentence qualities to filter data collected from potential users appears to be a very promising line of research. Our hope is to be able to build language models for conversational applications that require very little end-user data, and to maximize the benefit of using data given by application-aware users. By using an automatic classifier to assign qualities, then scaling counts to match the priors that either developers or real users give in their data, we’re able to reduce the WER and TER by 3-7%. We plan to continue working with sentence-quality based language models. In particular, we would like to 1) try clustering methods to build a classifier from unsupervised data, 2) learn scale factors automatically, 3) see if we can create a domain-independent quality classifier, and 4) see whether or not the approach works for higher-perplexity language models.

9. REFERENCES


