Revisiting the Predictability of Language: Response Completion in Social Media

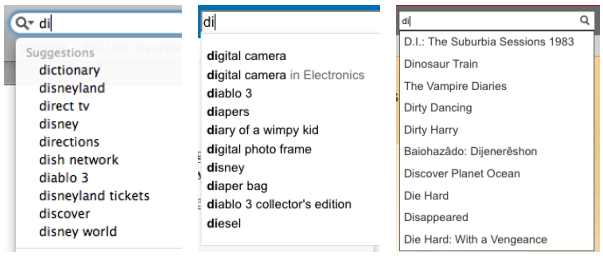
Bo Pang **Sujith Ravi**
Yahoo! Research
4401 Great America Parkway
Santa Clara, CA 95054, USA
bopang42@gmail.com sujith_ravi@yahoo.com

Abstract

The question "how predictable is English?" has long fascinated researchers. While prior work has focused on formal English typically used in news articles, we turn to texts generated by users in online settings that are more informal in nature. We are motivated by a novel application scenario: given the difficulty of typing on mobile devices, can we help reduce typing effort with message completion, especially in conversational settings? We propose a method for automatic response completion. Our approach models both the language used in responses and the specific context provided by the original message. Our experimental results on a large-scale dataset show that both components help reduce typing effort. We also perform an information-theoretic study in this setting and examine the entropy of user-generated content, especially in conversational scenarios, to better understand predictability of user generated English.

1 Introduction

How predictable is language? As early as 1951, long before large quantities of texts (or the means to process them) were easily available, Shannon had raised this question and proceeded to answer it with a set of clever analytical estimations. He studied the predictability of printed English, or "how well can the next letter of a text be predicted when the preceding N letters are known" (Shannon, 1951). This was quantified as the conditional entropy, which measures the amount of information conveyed from statistics over the preceding *context*. In this paper, we discuss a novel application setting which mirrors the predictability study as defined by Shannon.



(a) Google (b) Amazon (c) Netflix

Figure 1: Query completion as users type into the "Search using Google" box on a browser, as well as the search box in Amazon and Netflix.

Text completion for user-generated texts: Consider a user who is chatting with her contact or posting to a social media site using a mobile device. If we can predict the next word given the preceding words that were already typed in, we can help reduce the typing cost by offering users suggestions of possible completions of their partially typed messages (e.g., in a drop-down list). If the intended word is ranked reasonably high, the user can select the word instead of typing it. Assuming a lower cost associated with selections, this could lead to less typing effort for the user.

An interface like this would be quite familiar to Web users today. Providing suggestions of possible completions to partially typed queries, which we will refer to as *query completion*,[1] is a common feature of search boxes (Figure 1). In spite of the similarity in the interface, the underlying technical challenge can be quite different. Query completion does not necessarily rely on language models: can-

[1]Note that this feature is often tagged as "query suggestion" in the user interface; we avoid that terminology since it is often used to refer to query re-formulation (of a completely entered query) in the literature, which is a very different task.

Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 1489–1499, Jeju Island, Korea, 12–14 July 2012.

didate completions can be limited to popular queries that were previously submitted to the site or entries in a closed database of available objects, and ranking can be done by overall popularity. In contrast, our scenario requires generation of unseen texts.

Given the difficulty of generating full-length text, we consider a more realistic setting, where we perform completion on a word-by-word basis. Each time, we propose candidate completions at the word-level when the user is about to start a new word, or has partially entered the first few letters; once this word is successfully completed, we move on to the next one. This predict-verify-predict process exactly mirrors the human experiment described by Shannon (1951), except we do this at the word-level rather than the letter-level: having the user examine and verify predictions at the letter level would not be a practical solution for the intended application.

The response completion task: In addition, our task has another interesting difference from Shannon's human experiment. Consider the mobile-device user mentioned previously. If the user is *replying* to a piece of text (e.g., an instant message sent by a contact), we have an additional source of contextual information in the *stimulus*, or the text which triggered the *response* that the user is trying to type. Can we learn from previously observed stimulus-response pairs (which we will refer to as *exchanges*)? That is, can we take advantage of this conversational setting and effectively use the information provided by stimulus to better predict the next word in the response? We refer to this task as the *response completion* task.

Our task is different from "chatter-bots" (Weizenbaum, 1966), where the goal is to generate a response to an input that would resemble a human conversation partner. Instead, we want to complete a response as the replier intends to. Recently, Ritter et. al (2011) experimented with automatic response generation in social media. They had a similar conversational setting, but instead of completion based on partial input, they attempted to generate a response in its entirety given only the stimulus. While many of the generated responses are deemed possible replies to the stimulus, they have a low chance of actually matching the real response given by the user: they reported BLEU scores between 0 and 2 for various systems. This clearly shows the difficulty of the task. While we are addressing a more modest setting, would the problem prove to be too difficult even in this case?

In this paper, we propose a method for automatic response completion. Our approach models the generic language used in responses, as well as the contextual information provided by the stimulus. We construct a large-scale dataset of user-generated textual exchanges, and our experimental results show that both components help reduce typing effort. In addition, to better understand predictability of user generated English, we perform an information-theoretic study in this conversational setting to investigate the entropy of user-generated content.

2 Related work

There has been previous work in the area of human-computer interaction that examined text entry for mobile devices (MacKenzie and Soukoreff, 2002). In particular, one line of work looked into *predictive text input*, which examined input effort reduction by language prediction. Previous work in predictive text input had very different focus from our study. Oftentimes, the focus was to model actual typing efforts using mobile device keypads, examine the speed and cognitive load of different input methods, and evaluate with emprical user studies in lab settings (James and Reischel, 2001; How and Kan, 2005), where the underlying technique for language prediction can be as simple as unigram frequency (James and Reischel, 2001), or restricted to narrow domains such as grocery shopping lists (Nurmi et al., 2009). In addition, to the best of our knowledge, no previous work in predictive text input addressed the conversationl setting.

As discussed in Section 1, the response generation task (Ritter et al., 2011) also considered the converstational setting, but the MT-based technique was not well-suited to produce responses as intended by the user. There has been extensive previous research in language modeling (Rosenfeld, 2000). While previous work has explored Web text sources that are "better matched to a conversational speaking style" (Bulyko et al., 2003), we are not aware of much previous work that has taken advantage of information in the stimulus for word predictions in responses.

Previous work on entropy of language stems from the field of information theory (Shannon, 1948), starting with Shannon (1951). An extensive bibliography covering early related work (e.g., insights into the structure of language via information theory, entropy estimates via other techniques and/or for different languages, as well as a broad range of applications of such estimates) can be found in (Cover and King, 1978). More recently, Brown et. al (1992) computed an upper bound for the entropy of English with a trigram model, using the Brown corpus. Some other related works on this topic include (Teahan and Cleary, 1996; Moradi et al., 1998). There was also a recent study using entropy in the context of Web search (Mei and Church, 2008). In other settings, entropy has also been employed as a tool for studying the linguistic properties of ancient scripts (e.g., Indus Script) (Rao et al., 2009). While this seems like an interesting application of information theory for linguistic studies, it has also generated some controversies (Farmer et al., 2004).

In contrast, our work departs from traditional scenarios significantly. We perform entropy studies over texts generated in online settings which are more informal in nature. Additionally, we utilize the properties of language predictability within a novel application for automatically completing responses in conversational settings. Also, in our case we do not have to worry about issues like *"is this a language or not?"* because we work with real English news data which include articles written by professional editors and comments generated by users reading those articles.

3 Model

In this section, we first state our problem more formally, followed by descriptions of the basic N-gram language model we use, as well as two approaches that model both stimulus and preceding words in response as the context for the next-word generation. Given the intended application, we hope to achieve better prediction without incurring significant increase in model size.

3.1 Problem definition

Consider a stimulus-response pair, where the stimulus is a sequence of tokens $\mathbf{s} = (s_1, s_2, ..., s_m)$, and the response is a sequence of tokens $\mathbf{r} = (r_1, r_2, ..., r_n)$. Let $\mathbf{r_{1..i}} = (r_1, r_2, ..., r_i)$, our task is to generate and rank candidates for r_{i+1} given $\mathbf{s}$ and $\mathbf{r_{1..i}}$.

Note the models described in this section do not assume any knowledge of partial input for r_{i+1}. For the setting where the first c characters of r_{i+1} were also entered, we can restrict the candidate list to the subset with the matching prefix, and use the same ranking function.

3.2 Generic Response Language Model

First, we consider an N-gram language model trained on all responses in the training data as our generic response language model. Here we consider $N = 3$. Normally, trigram models use back-off to both bigrams and unigrams; in order to compare the effectiveness of trigram models vs. bigram models under comparable model size, we use back-off only to unigrams in both cases:

$$\textbf{trigram:} \quad P(r_{i+1} \mid \mathbf{r_{1..i}}) = \lambda_1 * P_3(r_{i+1} \mid r_i, r_{i-1}) + (1 - \lambda_1) * P_1(r_{i+1})$$

$$\textbf{bigram:} \quad P(r_{i+1} \mid \mathbf{r_{1..i}}) = \lambda_1 * P_2(r_{i+1} \mid r_i) + (1 - \lambda_1) * P_1(r_{i+1})$$

If we ignore the context provided by texts in the stimulus, we can simply generate and rank candidate words from the dictionary according to the generic response LM: $P(r_{i+1} \mid \mathbf{r_{1..i}})$.

As we will discuss in more detail in Section 6.2, modeling $\mathbf{s}$ and $\mathbf{r_{1..i}}$ jointly in the prediction of r_{i+1} would be rather expensive. In the following sections, we follow two main approaches to break this down into separate components: $P(r_{i+1} \mid \mathbf{r_{1..i}})$ and $P(r_{i+1} \mid \mathbf{s})$, and model each one separately.

3.3 Translating Stimuli to Responses

As mentioned in Section 1, Ritter et. al (2011) have considered a related task of generating a response in its entirety given only the text in the stimulus. They cast the problem as a translation task, where the stimulus is considered as the source language and the response is considered as the target language. We can adapt this approach for our response completion task.

Consider the noisy channel model used in statistical machine translation: $P(\mathbf{r}|\mathbf{s}) \propto P(\mathbf{r}) * P(\mathbf{s}|\mathbf{r})$. In

order to predict r_{i+1} given $\mathbf{r_{1..i}}$ and $\mathbf{s}$, for each candidate r_{i+1}, in principle one can marginalize over all possible completions of $\mathbf{r_{1..i+1}}$, and rank candidate r_{i+1} by that. That is, let P(n) be the distribution of response length, let $\mathbf{r'}$ be a possible completion of $\mathbf{r_{1..i+1}}$ (i.e., a response whose first $i+1$ tokens match $\mathbf{r_{1..i+1}}$). For each possible $n > i$, we need to marginalize over all possible $\mathbf{r'}$ of length n, and rank r_{i+1} according to

$$P(\mathbf{r_{1..i+1}} \mid \mathbf{s}) = \sum_{n>i} P(n) \cdot \sum_{|\mathbf{r'}|=n} P(\mathbf{r'} \mid \mathbf{s})$$

Clearly this will be computationally expensive. Instead, we take a greedy approach, and choose r_{i+1} which yields the optimal partial response (without looking ahead):

$$P(\mathbf{r_{1..i+1}} \mid \mathbf{s}) \propto P(\mathbf{r_{1..i+1}}) * P(\mathbf{s} \mid \mathbf{r_{1..i+1}})$$

which is equivalent to ranking candidate r_{i+1} by

$$P(r_{i+1} \mid \mathbf{r_{1..i}}) * P(\mathbf{s} \mid \mathbf{r_{1..i+1}}) \quad (1)$$

Since the first component is our LM model, and the second component is a translation model, we denote this as the *LM+TM model*. We use IBM Model-1 to learn the translation table on the training data. At test time, equal number of candidates are generated by each component, and combined to be ranked by Eq. 1.

3.4 Mixture Model

One potential concern over applying the translation model is that the response can often contain novel information not implied by the stimulus. While technically this could be generated from the so-called *null* token used in machine translation (added to the source text to account for target text with no clear alignment in the source text), significant amount of text corresponding to new information not in the source text is not what null tokens are meant to be capturing. In general, our problem here is a lack of clear word-to-word or phrase-to-phrase mapping in a stimulus-response pair, at least not what one would expect in clean parallel data.

Alternatively, one can model the response generation process with a mixture model: with probability λ_s, we generate a word according to a distribution over $\mathbf{s}$ ($P(w \mid \mathbf{s})$), and with probability $1 - \lambda_s$, we generate a word using the response language model:

$$\begin{aligned} P(r_{i+1} \mid \mathbf{s}, \mathbf{r_{1..i}}) &= \lambda_s \cdot P(r_{i+1} \mid \mathbf{s}) \\ &+ (1 - \lambda_s) \cdot P(r_{i+1} \mid \mathbf{r_{1..i}}) \end{aligned} \quad (2)$$

We examine two concrete ways of exploiting the context provided by $\mathbf{s}$.

Model 1 — LM + Selection model First, we examine a very simple instantiation of $P(w \mid \mathbf{s})$ where we select a token in $\mathbf{s}$ uniformly at random. This is based on the intuition that to be semantically coherent, a reply often needs to repeat certain content words in the stimulus. (Similar intuition has been explored in the context of text coherency (Barzilay and Lapata, 2005).) This is particularly useful for words that are less frequently used: they may not be able to receive enough statistics to be promoted otherwise. More specifically,

$$P(r_{i+1} \mid \mathbf{s}) = \frac{\mathbf{1}_{r_{i+1} \in \mathbf{s}}}{|\mathbf{s}|}$$

We can take λ_s to be a constant λ_{select}, which can be estimated in the training data as the probability of a response token being a repetition of a token in the corresponding stimulus.

Model 2 — LM + Topic model Another way to incorporate information provided in $\mathbf{s}$ is to use it to constrict the topic in $\mathbf{r}$. We can learn a topic model over conversations in the training data using Latent Dirchlet Allocation (LDA) (Blei et al., 2003). At test time, we identify the most likely topic of the conversation based on $\mathbf{s}$, and expect r_{i+1} to be generated from this topic. That is,

$$P(r_{i+1} \mid \mathbf{s}) = P(r_{i+1} \mid t^*)$$

$$\text{where } t^* = \text{argmax}_t P(\text{topic} = t \mid \mathbf{s})$$

More specifically, we first train a topic model on ($\mathbf{s}$, $\mathbf{r}$) pairs from the training data. Given a new stimulus $\mathbf{s}$, we then select the highest ranked topic as being representative of $\mathbf{s}$. Note that alternatively we could consider all possible topic assignments; in that case we would have had to sum probabilities over all topics, and that could also introduce noise. A similar strategy has been previously employed for

other topic modeling applications in information retrieval, where documents are smoothed with their highest ranked topic (Yi and Allan, 2009). We lower the weight λ_s if $P(t^* \mid \mathbf{s})$ is low. That is, we use $\lambda_s = \lambda_{topic} * P(t^* \mid \mathbf{s})$ in Eq. 2.

4 Data

In order to investigate text completion in a conversational setting, we need to construct a large-scale dataset with textual exchanges among users. An ideal dataset would have been a collection of instant messages, but these type of datasets are difficult to obtain given privacy concerns. To the best of our knowledge, existing SMS (short message service) datasets only contain isolated text spans and do not provide enough information to reconstruct the conversations. There are, however, a high volume of textual exchanges taking places in public forums. Many sites with a user comment environment allow other users to reply to existing comments, where the original comment and its reply can form a (stimulus, response) pair for our purposes.

To this end, we extracted (comment, reply) pairs from Yahoo! News[2], where under each news article, a user can post a new comment or reply to an existing comment. In fact, a popular comment can have a long thread of replies where multi-party discussions take place. To ensure the reply is a direct response to the original comment, we took only the first reply to a comment, and consider the resulting pair as a textual exchange in the form of a (stimulus, response) pair. We gathered data from a period of 14 weeks between March and May, 2011. A random sample yielded a total of 1,487,995 exchanges, representing 237,040 unique users posting responses to stimuli comments authored by 357,811 users. In the raw dataset (i.e., before tokenization), stimuli average at 59 tokens (332 characters), and responses average at 26 tokens (144 characters).

We took the first 12 weeks of data as training data (1,269,732 exchanges) and the rest 2 weeks of data as test data (218,263 exchanges).

[2]Note that previous work has used a dataset with 1 million Twitter conversations extracted from a scraping of the site (Ritter et al., 2011), where a status update and its replies in Twitter form "conversations". This dataset is no longer publicly available. At the time of this writing, we were not able to identify a data source to re-construct a dataset like that.

5 Experiments

5.1 Evaluation measures

Recall@k : Here, we follow a standard evaluation strategy used to assess ranking quality in information retrieval applications. For each word, we check if the correct answer is one of the top-k tokens being suggested. We then compute the recall at different values of k. While this is a straight-forward measure to assess the overall quality of different top-k lists, it is not tailored to suit our specific task of response completion. In particular, this measure (a) does not distinguish between (typing) savings for a short word versus a long one, and (b) does not distinguish between the correct answer being higher up in the list versus lower as long as the word is present in the top-k list.

TypRed : Our main evaluation measure is based on "reduction in typing effort for a user of the system", which is a more informative measure for our task. We estimate the typing reduction via a hypothetical typing model[3] in the following manner:

Suppose we show top k predictions for a given setting. Now, there are two possible scenarios:

1. if the user does not find the correct answer in the top-k list, he/she gives up on this word and will have to type the entire word. The typing cost is then estimated to be the number of characters in the word l_w;
2. if the user spots the correct answer in the list, the cost for choosing the word is proportional to the rank of the word $rank_w$, with a fixed cost ratio c_0. Suppose the user scrolls down the list using the down-arrow ($\downarrow$) to reach the intended word (instead of typing), then $rank_w \cdot c_0$ reflects the scrolling effort required, where c_0 is the relative cost of scrolling down versus typing a character.

In general, pressing a fixed key can have a lower cost than typing a new one, in addition, we can imagine a virtual keyboard where navigational keys occupy bigger real-estate, and thus incur less cost to press. As a result, it's reasonable to assume c_0 value that is smaller than 1. In all our experiments, $c_0 = 0.5$ unless otherwise noted. Note that if the

[3]More accurate measures can be be developed by observing user behavior in a lab setting. We leave this as future work.

System	TypRed ($c = 0$)	TypRed ($c = 1$)	TypRed ($c = 2$)
1. Generic Response LM (trigram)	15.10	22.57	14.29
2. Generic Response LM (trigram) + TM	9.03	17.53	11.56
3. Mixture Model 1: Generic Response LM (trigram) + Selection	**15.18***	**23.43***	**15.13***
4. Mixture Model 2: Generic Response LM (trigram) + Topic Model	15.10	22.57	14.33

Table 1: Comparison of various prediction models (in terms of TypRed score @ rank 5) on all (stimulus,response) pairs from a large test collection (218,263 exchanges) when the first c characters of each word are typed in by the user. A higher score indicates better performance. * indicates statistical significance ($p < 0.05$) over the baseline score.

typing model assumes a user selects the intended word using an interface that is similar to a mousing device, the cost may increase with $rank_w$ at a sub-linear rate; in that case, our measure will be over-estimating the cost.

In order to have a consistent measure that always improves as the ranking improves, we assume a clever user who will choose to finish the word by typing or by selecting, depending on which cost is lower. Combining these two cases under the clever-user model, we estimate the reduction in typing cost for every word as follows:

$$TypRed(w, rank_w) = 100 \cdot [1 - \frac{min(l_w, rank_w \cdot c_0)}{l_w}]$$

where w is the correct word, l_w is the length of w, and $rank_w$ is the rank of w in the top-k list. A higher value of TypRed implies higher savings achieved in typing cost and thereby better prediction performance.

5.2 Experimental setup

We run experiments using the models described in Section 3 under two different settings: (1) *previous words* from the response are provided, and (2) *previous words from response + first c characters of the current word* are provided.

During the candidate generation phase, for every position in the response message we present the top 1,000 candidates (as scored by the generic response language model or mixture models). We reserve a small subset of (∼1,000) exchanges as development data for tuning parameters from our models.

For the generic response language models, we set the interpolation weight λ_1 = 0.9. For the selection-based mixture model, we estimate the mixture weights on the training data and set λ_{select} (0.09). For the topic-based mixture model, we ran a grid search with different parameter settings for λ_{topic} on the held-out development set and chose the value (0.01) that gave the best performance (in terms of TypRed).

5.3 Results

Previous words from response observed: We first present results for the setting where only previous words from the response are provided as context. We use TypRed scores as our evaluation measure here (higher TypRed implies more savings in typing effort). Even with a unigram LM we achieve a small but non-negligible reduction (TypRed=2.15) in the typing cost. But a bigram LM significantly improves performance (TypRed=11.91), and with trigram LM we observe even better performance (TypRed=15.10). Since the trigram LM yields a high performance, we set this as our default LM for all other models.

Recall that in all experiments, we set c_0, the cost ratio of selecting a candidate from the ranked top-k list (via scrolling) versus typing a character to a value of 0.5. But we also experimented with a hypothetical setting where $c_0 = 1$ and noticed that the trigram LM achieves a slightly lower but still significant typing reduction (TypRed score of 9.58 versus 15.10 for the earlier case).

The first column of Table 1 ($c = 0$) compares the performance of other models for this setting. We find that adding a translation model (LM+TM) does not help for this task; in fact, it results in lower scores than using the LM alone. This suggests that a translation-based generative approach may not be suitable, if the goal is to predict text as intended by the user. This is consistent with previous observations on a related task (Ritter et al., 2011), as we

discussed in Section 1.

In contrast, the mixture models do much better. In fact, LM+Selection model produces better results than trigram LM alone. We also note that estimating the mixture parameter on the training data rather than using a fixed value increases TypRed scores: 14.02 with a fixed λ_{select} = 0.5 versus 15.18 with $\lambda^*_{select} = 0.09$. This comparison also holds for $c > 0$ — that is, a naive version of LM+Selection that selects a word from the stimulus whenever the prefix allows would not have worked well.

In principle the LM+Topic model is potentially more powerful in that $P(w \mid \mathbf{s})$ is not limited to the words in $\mathbf{s}$. However, in our experiments it does not yield any considerable improvement over the original LM. We postulate that this could be due to the following reason: once the context provided by $\mathbf{s}$ is reduced to the topic level, it is not specific enough to provide additional information over preceding words in the response.

Previous words from response + first c characters of current word observed: Table 1 also compares the TypRed performance of all the models under settings where $c > 0$. We notice striking improvements in performance for $c = 1$ which is consistent across all models. Our best model is able to save the user approximately 23% in terms of typing effort (according to TypRed scores). Interestingly a lot less reduction was observed for $c = 2$: the second character, on average, does not improve the ranking enough to justify the cost of typing this extra character.

Next, we pick our best model (Mixture Model 1) and perform some further analysis. We examine the effect of providing longer list (shown in Table 2) and notice little further improvement beyond $k = 10$. We also note that the TypRed improvement achieved over the baseline (LM) model at rank 10 is more than twice the gain achieved at rank 5.

We also evaluated its performance using a standard measure (Recall@k). Figure 2 plots the recall achieved by the system at different ranks k. An increasing recall at even high ranks ($k = 100$) suggests that the quality of the candidate list retrieved by this model is good. This also suggests that there is still room for improvements, and we leave that as interesting future work.

Rank (k)	**TypRed score**
1	9.02
5	15.18
10	16.14
15	16.28
20	16.29
25	16.29

Table 2: Comparison of typing reductions achieved over the entire test data when top k list is provided to the user.

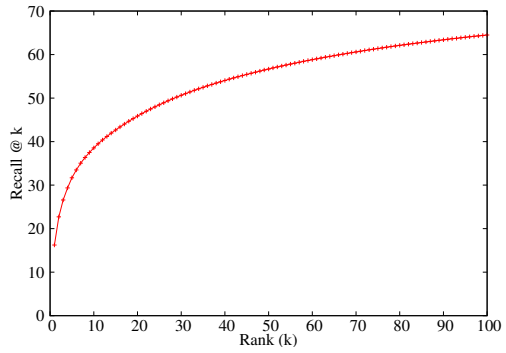


Figure 2: Recall @ rank k for Mixture Model 1 on the entire test data.

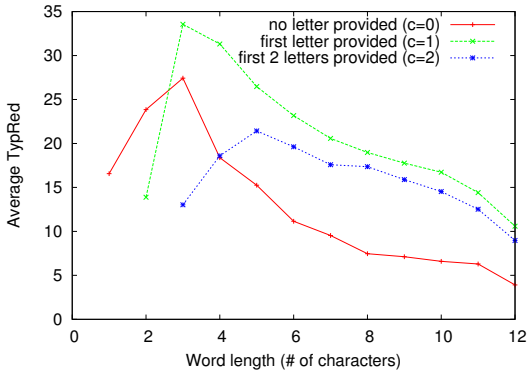


Figure 3: Average TypRed score versus Word length (# of characters) for Mixture Model 1 when the first c characters of the word is typed in by the user.

Finally, in Figure 3, we plot the average TypRed scores against individual token (word) length. Figure 3 indicates that the model is able to achieve a higher reduction for shorter words compared to very long ones. This demonstrates the utility of such a response completion system, especially since shorter words are predominant in conversational settings. We also compared the average reduction achieved on messages of different lengths (number of words). Overall we observe consistent reduction for different message lengths. This suggests our system can

Source	Top translations				
:)	:)	!	you	?	:D
lmao	lol	lmao	u	...	i
feeling	feeling	feel	better	!	you
question	.	question	the	,	to
Are	I	.	,	are	yes

Table 3: Examples of a few stimulus/response translations learned using IBM Model-1.

OBAMA, USA, Fact, Meghan, GIVE, PRESIDENT, Canadian, Mitch, Jon, Kerry, TODAY, Justice, Liberalism, ...
President, Notice, Tax, LMAO, Hmmm, Trump, people, OBAMA, common, Aren, WAIT, Bachman, mon, McCain, ...
Great, Cut, Release, Ummm, Rest, Mark, isnt, YAHOO, Sad, END, RON, jesus, Ugh, TRUMP, ...
Nice, Navy, Make, Interesting, Remember, Excuse, WAKE, Hooray, Birth, mon, Yeah, Dumb, Michael, geronimo, ...

Table 4: Examples of top representative words for a few topics generated by the LDA topic model trained on news comment data.

be useful for both Tweet-like short messages as well as more lengthy exchanges in detailed discussions.

5.4 Discussion

Table 3 displays some sample translations learned using the TM model described in Section 3.3. Interestingly, emoticons and informal expressions like `:)` or `lmao` in the stimulus tend to evoke similar type of expressions in the response (as seen in Table 3). Some translations (e.g., *feeling* → *better*) are indicative of question/answer type of exchanges in our data. But most of the other translations are noisy or uninformative (e.g., *Are* → .). This provides further evidence as to why a translation-based approach is not well suited for this particular task and hence does not perform as well as other methods.

Finally, in Table 4, we provide a few sample topics generated by the LDA topic model (which is used by Mixture Model 2 described in Section 3.4). We find that while a few topics display some semantic coherence (e.g., political figures), many of them are noisy (or too generic) which further supports our earlier observation that they are not useful enough to help in the prediction task.

6 Entropy of user comments

We adapt the notion of predictability of English as examined by Shannon (1951) from letter-prediction to token-prediction, and define the predictability of English as how well can the next token be predicted when the preceding N tokens are known. How much does the immediate context in the response help reduce the uncertainty? How does user-generated content compare with more formal English in this respect? And how about the corresponding stimuli — given the preceding N tokens, does the knowledge of stimulus further reduce the uncertainty? These questions motivated a series of studies over entropy in different datasets.[4]

6.1 Comparison of N-gram entropy

Following Shannon (1951), we consider the following function F_N, which can be called the N-gram entropy, as the measure of predictability:

$$F_N = -\sum_{i,j} p(b_i, j)\ \log_2\ p(j \mid b_i)$$

where b_i is a block of $N-1$ tokens, j is an arbitrary token following b_i, and $p(j \mid b_i)$ is the conditional probability of j given b_i. This conditional entropy reflects how much is the uncertainty of the next token reduced by knowing the preceding $N-1$ tokens.

Under this measure, is user-generated content more predictable or less predictable than the more formal "printed" English examined by Shannon? Maybe it is more predictable, since most users in informal settings use simpler English, which may contain fewer variations than the complex structures observed in more formal English. Or perhaps it is less predictable — variations among different users (who may not follow proper grammar) may lead to more uncertainty in the prediction of "the next word". Which would be the case?

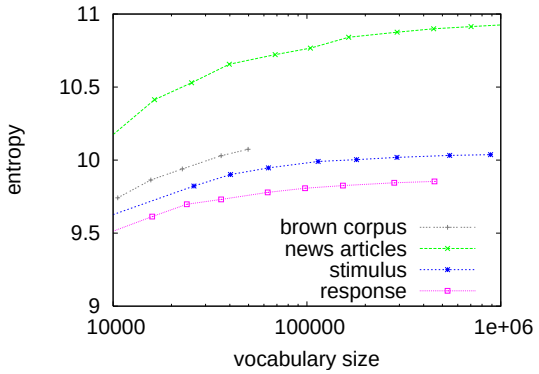
To answer this question empirically, we construct a reference dataset written in more formal English (D_f) to be compared against the user comments dataset described in Section 4 (D_u). If D_f covers very different topics from D_u, then even if we do observe differences in entropy, it could be due to topical differences. A standard mixed-topic dataset like

[4]Note that our findings are not to be interpreted as prediction performance over unseen texts. For that, one needs to compute cross-entropy between training and test corpora. Since Section 5 is already addressing this question with proper training / test split, in this section, we focus on the variability of language usage in a corpus. This also avoids having to control for "comparable" training/test splits in different types of datasets.

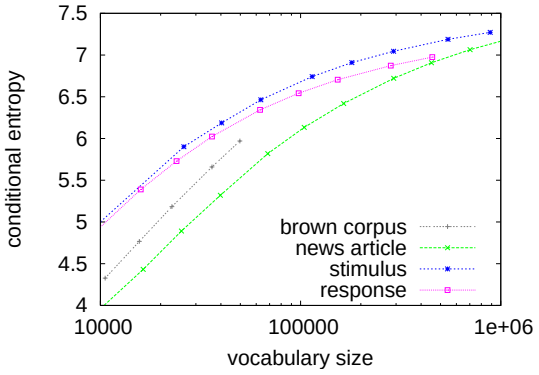
the Brown Corpus (Kucera and Francis, 1967) may not be ideal in this sense (e.g., it contains fiction categories such as "Romance and Love Story", which may not be represented in our D_u). Instead, we obtained a sample of news articles on Yahoo! News during March - May, 2011, and extracted unique sentences from these articles. This yields a D_f with more comparable subject matters to D_u.[5]

Next, we compare both the entropy over unigrams and N-gram entropy in three datasets: the news article dataset described above, and comments data (Section 4) separated into stimuli and responses. We also report corresponding numbers computed on the Brown Corpus as references. Note that datasets with different vocabulary size can lead to different entropy: the entropy of picking a word from the vocabulary uniformly at random would have been different. Thus, we sample each dataset at different rates, and plot the (conditional) entropy in the sample against the corresponding vocabulary size.

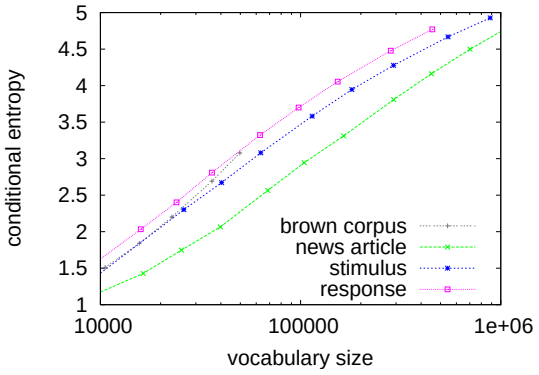
As shown in Figure 4(a), the entropy of unigrams in D_u (both stimuli and responses) is consistently lower than in D_f.[6] On the other hand, both stimuli and responses exhibit higher uncertainty in bigram entropy (Figure 4(b)) and trigram entropy (Figure 4(c)). That is, when no contexts are provided, word choices (from similarly-sized vocabularies) in D_f is more evenly distributed than in D_u; but once the proceeding words are given, the next word is more predictable in D_f than in D_u. We postulate that the difference in unigram entropy could be due to (a) more balanced topic coverage in D_f vs. more skewed topic coverage in D_u, or (b) professional reporters mastering a more balanced use of the vocabulary. If (b) is the main reason, however, the lower trigram entropy in D_f would seem unexpected — shouldn't professional journalists also have a more balanced use of different phrases? Upon further contemplation, what we hypothesized earlier could be true: professional writers use the "proper" English expected in news coverage, which could limit



(a) Entropy of unigrams



(b) Bigram entropy (F_2)



(c) Trigram entropy (F_3)

Figure 4: Entropy of unigrams and N-gram entropy.

their trigram uncertainties; on the other hand, users are not bound by conventions (or even grammars), which could lead to higher variations.

Interestingly, distributions in the stimulus dataset are closer to news articles: they have a higher unigram entropy than responses, but a lower trigram entropy at comparable vocabulary sizes. In particular, recall from Section 4 that our comments dataset contains roughly 237K repliers and 357K original com-

[5]We note that this does not guarantee the exact same topic distribution as in the comment data.

[6]For reference, Shannon (1951) estimated the entropy of English to be 11.82 bits per word, due to an incorrect calculation of a 8727-word vocabulary given Zipf distribution. The correct number should be 9.27 bits per word for a vocabulary size of 12,366 (Yavuz, 1978).

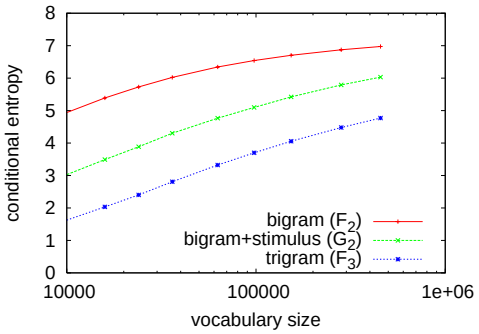


Figure 5: Predicting the next word in responses: bigram entropy vs. bigram+stimulus entropy vs. trigram entropy.

menters. If higher trigram entropy is due to variance among different users, the stimulus dataset should have had a higher trigram entropy. We leave an explanation of this interesting behavior as future work.

6.2 Information in stimuli

We now examine the next question: does knowing words in the stimulus further reduce the uncertainty of the next word in the response? For simplicity, we model the stimulus as a collection of unigrams. Consider the following conditional entropy:

$$G_N \quad = \quad -\sum_{i,k,j} p(b_i, j, s_k) \log_2 p(j \mid b_i, s_k)$$

where b_i is a block of $N-1$ tokens in a *response* $\mathbf{r}$, j is an arbitrary token following b_i, and s_k is an arbitrary token in the corresponding stimulus $\mathbf{s}$ for $\mathbf{r}$. Note that for each b_i, we consider every token in the corresponding $\mathbf{s}$. That is, a (stimulus, response) pair with m and n tokens respectively generates $m*(n-N+1)$ observations of (b_i, j, s_k) tuples. We refer to this as the N-gram+stimulus entropy. If knowing s_k in addition to b_i does not provide extra information, then $p(j \mid b_i, s_k) = p(j \mid b_i)$, and $G_N = F_N$.

Figure 5 plots G_N for $N = 2$. Interestingly, we observe $F_2 > G_2 > F_3$ (this trend holds for larger values of N, omitted here for clarity). That is, knowing both the preceding $N-1$ tokens and tokens in the stimulus results lowers the uncertainty over the next token in response (bigram+stimulus entropy $<$ bigram entropy); on the other hand, this is not as effective as knowing one more token in the preceding block (trigram entropy $<$ bigram+stimulus entropy).

Note that from the model size perspective, modeling $p(j \mid b_i, s_k)$ as in G_N would have been much more expensive than $p(j \mid b_i)$ in F_{N+1}. Take the case of G_2 vs. F_3. Let V be the vocabulary of user comments (ignore for now differences in responses and stimuli). While both seem to require computations over $V \times V \times V$, the number of unique observed (b_i, j, s_k) tuples for G_2 (i.e., number of unique bigrams in responses paired up with unigrams in corresponding stimuli) is 725,458,892, whereas the number of unique observed (b_i, j) pairs for F_3 (i.e., number of unique trigrams) is only 14,692,952. This means modeling trigrams would result in a model 2% the size of bigram+stimulus, yet it could achieve better reduction in uncertainty.

Note that in order to reduce model complexity, the models proposed in Section 3 all broke down $P(r_{i+1} \mid \mathbf{s}, \mathbf{r_{1..i}})$ into independent components $P(r_{i+1} \mid \mathbf{s})$ and $P(r_{i+1} \mid \mathbf{r_{1..i}})$, rather than modeling the effect of $\mathbf{s}$ and $\mathbf{r_{1..i}}$ jointly as the underlying model corresponding to G_N. Indeed, it would have been impractical to model $p(j \mid b_i, s_k)$ directly. Our studies confirmed the validity of this choice: even if we look at the performance on the training data itself (i.e., igoring data sparseness issues), the smaller trigram model would have yielded better results than the significantly more expensive bigram+stimulus model. Still, since G_N shows a consistent improvement over F_N, there could be more information in the stimulus that we are not yet fully utilizing, which can be interesting future work.

7 Conclusions

In this paper, we examined a novel application: automatic response completion in conversational settings. We investigated the effectiveness of several models that incorporate contextual information provided by the partially typed response as well as the stimulus. We found that the partially typed response provides strong signals. In addition, using a mixture model which also incorporates stimulus content yielded the best overall result. We also performed empirical studies to examine the predictability of user-generated content. Our analysis (entropy estimates along with upper-bound numbers observed from experiments) suggest that there can be interesting future work to explore the contextual information provided by the stimulus more effectively and further improve the response completion task.

References

Regina Barzilay and Mirella Lapata. 2005. Modeling local coherence: An entity-based approach. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, ACL'05*.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Peter F. Brown, Vincent J. Della Pietra, Robert L. Mercer, Stephen A. Della Pietra, and Jennifer C. Lai. 1992. An estimate of an upper bound for the entropy of English. *Comput. Linguist.*, 18:31–40.

Ivan Bulyko, Mari Ostendorf, and Andreas Stolcke. 2003. Getting more mileage from web text sources for conversational speech language modeling using class-dependent mixtures. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: companion volume of the Proceedings of HLT-NAACL 2003–short papers - Volume 2*, NAACL-Short '03, pages 7–9.

Thomas M. Cover and Roger C. King. 1978. A convergent gambling estimate of the entropy of English. *IEEE Transactions on Information Theory*, 24:413–421.

Steve Farmer, Richard Sproat, and Michael Witzel. 2004. The collapse of the Indus-script thesis: The myth of a literate Harappan civilization. *Electronic Journal of Vedic Studies*, 11:379–423 and 623–656.

Yijue How and Min-Yen Kan. 2005. Optimizing predictive text entry for short message service on mobile phones. In *Proceedings of the Human Computer Interfaces International (HCII)*.

Christina L. James and Kelly M. Reischel. 2001. Text input for mobile devices: comparing model prediction to actual performance. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '01, pages 365–371, New York, NY, USA. ACM.

Henry Kucera and W. Nelson Francis. 1967. *Computational analysis of present-day American English*. Brown University Press.

I. Scott MacKenzie and R. William Soukoreff. 2002. Text entry for mobile computing: Models and methods,theory and practice. *Human-Computer Interaction*, 17(2-3):147–198.

Qiaozhu Mei and Kenneth Church. 2008. Entropy of search logs: how hard is search? with personalization? with backoff? In *Proceedings of the international conference on Web search and web data mining*, WSDM '08, pages 45–54.

Hamid Moradi, Jerzy W. Grzymala-busse, and James A. Roberts. 1998. Entropy of english text: experiments with humans and a machine learning system based on rough sets. *Information Sciences*, 104:31–47.

Petteri Nurmi, Andreas Forsblom, Patrik Floréen, Peter Peltonen, and Petri Saarikko. 2009. Predictive text input in a mobile shopping assistant: methods and interface design. In *Proceedings of the 14th international conference on Intelligent user interfaces*, IUI '09, pages 435–438, New York, NY, USA. ACM.

Rajesh P. N. Rao, Nisha Yadav, Mayank N. Vahia, Hrishikesh Joglekar, R. Adhikari, and Iravatham Mahadevan. 2009. Entropic evidence for linguistic structure in the Indus script. *Science*.

Alan Ritter, Colin Cherry, and William B. Dolan. 2011. Data-driven response generation in social media. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 583–593.

Ronald Rosenfeld. 2000. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 88.

Claude E. Shannon. 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656.

Claude E. Shannon. 1951. Prediction and entropy of printed English. *Bell System Technical Journal*, 30:50–64.

W. J. Teahan and John G. Cleary. 1996. The entropy of English using PPM-based models. In *In Data Compression Conference*, pages 53–62. IEEE Computer Society Press.

Joseph Weizenbaum. 1966. Eliza: a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9:36–45.

D. Yavuz. 1978. Zipf's law and entropy (Corresp.). *IEEE Transactions on Information Theory*, 20:650.

Xing Yi and James Allan. 2009. A comparative study of utilizing topic models for information retrieval. In *Proceedings of the European Conference on IR Research on Advances in Information Retrieval*, pages 29–41.