DRAFT DO NOT DISTRIBUTE

Improved sentence completions in email using context

Kenneth C. Arnold
Harvard University

Kai-Wei Chang
Microsoft Research

Adam Tauman Kalai
Microsoft Research

Abstract

Many people find text autocompletion helpful, but current systems are limited to predicting only one or two words at a time. We consider the "Megacomplete" text prediction problem of suggesting substantial chunks of text (e.g., sentences or longer), in situations where the context makes a certain completion highly likely. We design a system that learns to identify good completions based on an individual's corpus of thousands of sent email messages, based on features of the context (e.g., subject) and the text typed so far. We evaluate the system based both on exact text matches and on human evaluation of completion acceptability. A companion paper studies usability factors of the system.

1 Introduction

While composing a message, many smartphone keyboards offer to *autocomplete* the next word.However, their length of one (or occasionally two) words limits their utility in composing long messages. To what extent can one similarly offer long completions of natural language text? Email is a suitable domain to study this question since sent mail folders often contain years worth of messages from the author [Hangal *et al.*, 2011]. We are developing an autocomplete system called "Megacomplete" that offers long completions of natural language as a user types an email message, based on the context and the user's own prior sent messages. There are numerous factors and features (ranging from personalized salutations and signatures to complete messages) to consider in the design of such an interface; a companion paper studies the usability of our system [reference withheld].

Consider the following motivating illustration taken from our experiment. The email being revisited had subject, "Fwd: We have a delivery for you!", and the author was forwarding a package delivery notification to an employee. Before a single character was typed, Megacomplete would have suggested: [Can you please pick up the package?]. The message the author had actually sent was simply, "When you get a chance...". Our system's suggestion was "plagiarized" from a prior message he had composed, one of sixteen similar messages with identical subjects over the course of three years. This example illustrates several features of autocomplete in the context of authoring natural language. First, it can offer an entire sentence sometimes before a single letter is typed (and, in this case, the whole email consisted of a single sentence). Second, clues from context, such as the message subject, can be crucial for offering good suggestions based on a short (or zero-length) prefix. And finally, autocomplete can sometimes offer suggestions that are preferable to what was actually intended. In this case, the suggestion conveyed the same request more politely.

For simplicity, this paper focuses on completing sentences (though Megacomplete can complete other blocks of text including greetings and entire emails in certain cases). Departing from the prior work that introduced this problem [Grabski and Scheffer, 2004], we focus on *long* completions and employ *context* such as previous sentences in the message and other email-specific features including subject and date. The algorithm produces at most one suggested completion, depending on its confidence.

Based on prior text and other context, our algorithm efficiently searches for the "best matched" sentence ending from among all sentences and all positions within those sentences in earlier messages sent by the author. If its confidence in is sufficiently high, it suggests that suffix as a completion. Our notion of best-matched is a based on a number of features including a novel feature which we call the *contextual local density*. This feature favors completions that use frequent words *in that context* such as "Thank [you so much for your interest in the position.]" over "Thank [you for the application you sent Tuesday morning.]" (suggested text displayed in brackets). The suggestion is made by first pruning to a small set of candidates and then maximizing a matching score where the weights on the scoring function are trained by maximizing a heuristic performance criterion on the author's prior messages.

Like prior work, evaluation is performed on completions of sentence prefixes from a test set of messages from the corpus. To evaluate acceptability of suggested completions, we consider two metrics: (a) whether or not the suggested completion exactly matched the actual sentence ending in the corpus, and (b) the author's own assessment of whether or not they would have accepted such a completion *as is*, without editing. The motivation for such a stringent criterion, pointed out in prior work [Grabski and Scheffer, 2004], is that accepting a suggestion and editing it may often require more work than simply entering the sentence in its original form.

We first had an individual evaluate completions of his own

sent messages based on a corpus of ten years of sent mail to determine which ones were acceptable. Second, we performed a fully automatic evaluation based on the largest corpus of publicly available sent messages from a single account, those of Florida governor Jeb Bush[1]. We find that, at least for these two corpora, context does indeed improve suggestions, both in the automatic and human evaluations. However, we see differences in qualitative ways in which context is useful for these two accounts. Quantifying this improvement across a wider range of users is left for future work.

The primary contributions of this paper are: formulating the prediction problem of offering long, accurate text completions based on context; designing and evaluating a learning-based algorithm for this problem; and demonstrating that, for at least some authors, context is useful for improving predictions.

1.1 Related work

Compared to autocompleting words or search queries, the sentence completion problem has received less attention. Grabski and Scheffer [2004] initially formulated the problem and applied information retrieval techniques to find the most suitable sentence completion. However, their formulation and that of follow-up work [Bickel *et al.*, 2005] focused on the more challenging scenario of completing sentences in isolation of context, where sentence prefixes must typically be longer and sentence completions shorter. Other work has explored phrase prediction, where phrases need not necessarily end sentences [Nandi and Jagadish, 2007]. However, these previous formulations combine both short (single word) and long completions, whereas we focus solely on longer completions.

It is worth noting that N-gram models often rely on a large collection of training data, which is often inaccessible in our personal email setting. Also, such language models are have difficulty generating grammatical and semantically proper sentences [Bickel *et al.*, 2005], especially on collections of text with high entropy. Along the line of research of language models, neural language models [Bengio *et al.*, 2003; Botha and Blunsom, 2014; Kim *et al.*, 2015] have recently proven very effective. Even more than N-gram approaches, however, they typically require large amounts of training data. Furthermore, even when they achieve low perplexity in modeling language, they can be difficult to interpret.

In addition, several studies in natural language processing and information retrieval have been conducted for problems similar to sentence completion. They often focus on focus on completing a sentence with a short phrase [Gubbins and Vlachos, 2013] or completing queries to a search engine [Bast and Weber, 2006; Shokouhi and Radinsky, 2012].

As of November 2015, Google's Inbox app suggests up to three short replies to entire messages message[2]. No formal evaluation is discussed.

2 Problem formulation

Consider a sequence of documents $D^1, D^2, \ldots$, where each document $D^i = (M^i, s^i_1, s^i_2, \ldots, s^i_{n^i})$ consists of metadata $M^i \in \mathcal{M}$ and n^i sentences. Sentence $s^i_j \in \mathcal{W}^*$ is a sequence $w^i_{j1}, w^i_{j2}, \ldots, w^i_{jn^i_j}$ of n^i_j words $w \in \mathcal{W}$. We omit i and j when clear from context. For sequence $x_1, x_2, \ldots, x_n$, we write $x[:i]$ to denote the *prefix* $x_1, x_2, \ldots, x_{i-1}$ and $x[i:] = x_i, x_{i+1}, \ldots x_n$ to denote the *suffix*. Before writing the kth word of the jth sentence of the ith document[3], the completion $c^i_{jk} = A(w^i_j[:k], \chi^i_{jk})$ is the result of the algorithm A run with input the prefix $w^i_j[:k] = w^i_{j1}, \ldots, w^i_{j(k-1)}$ and the *context* χ^i_{jk}, defined as:

$$\chi^i_{jk} = \big(D[:i], M^i, s^i[:j]\big)$$

consisting of the previous documents, current metadata, and preceding sentences. The user may choose to accept or reject the suggested completion c^i_{jk}. The algorithm may also choose to abstain from offering a completion by outputting the *null* completion ϵ.

2.1 Replay scenario

Following prior work, we evaluate performance in a "replay" model in which an algorithm is run over a corpus of sent messages, word by word, and completions are generated from chronologically earlier messages. They are evaluated for acceptability, either manually by the original author or automatically. We focus on what we call "big wins": completions of length at least $M = 6$ words that are acceptable to the author. The threshold of six was chosen to roughly capture the observed usefulness of such a system and also to distinguish it from the more familiar problem of completing a single word or phrase.

In our replay model, we consider that the interface offers a suggestion if the score of that suggestion is above a confidence threshold. The author then decides whether to accept that suggestion. Since we require suggestions to be at least 6 words, each accepted completion corresponds to a big win.

Our definitions of precision and recall are similar to those of earlier work [Bickel *et al.*, 2005] except that we focus only on big wins. Since any sentence in any message to retype could potentially be completed, *recall* is relative to the total number of sentences. We define *precision* = (# accepted completions) /# suggested completions), and *recall* = (# accepted completions) / (# sentences considered). Once a suggestion is accepted in a sentence, we do not count any other suggestions in that same sentence, to avoid giving multiple credit.

3 Our completion algorithm

Our algorithm generates completions by finding verbatim text from the author's own prior email, essentially plagiarizing the author's own emails. Prior work has called this approach *instance-based* [Bickel *et al.*, 2005]. We use verbatim text from the author's prior email to increase the chance that the completion is grammatical, sensible, and in the author's own voice. The outline of the process is: candidate search, vectorization, and scoring.

[1] `http://jebemails.com/email`

[2] `http://gmailblog.blogspot.co.uk/2015/11/computer-respond-to-this-email.html`

[3] For simplicity of presentation, we omit punctuation and consider predictions at word boundaries, but our actual system accounts for punctuation and offers completions after each character.

3.1 Searching for candidates

A *candidate location* is a message in the corpus together with a position in its body. We consider offering the completion of the sentence found at that position, which we refer to as the *candidate suffix*. We also may refer to *candidates* when it is clear from context whether we are referring to the suffix or location. However, our algorithm maintains locations at all points in time since they are more efficient and contain additional information about the message.

We first find candidate locations that are preceded by text which perfectly agrees back with as many words as possible with what the user has just typed, i.e., the end of the prefix to complete. We use a suffix array [Manber and Myers, 1993] to identify these locations efficiently. We look for matches of the last k words of the prefix, finding the maximal k such that at least 5 suffixes remain. If more than 1,000 locations remain after this process, we take the 1,000 locations from the most recent messages. Since we are looking to suggest at least $M = 6$ words to finish a sentence, we prune any candidate suffixes with fewer than M words remaining in the sentence. Importantly, we allow the prefix match to span backwards over sentence and paragraph boundaries, which helps make suggestions even before the user has typed a single character.

3.2 Vectorization

Once a set of candidates has been identified, the challenge is then to determine which suffix is most likely to complete the current sentence. To do this we use several features:

Contextual local density score Among a set of candidates, some may be more generally useful than others and thus more likely to be appropriate for the message currently being written. A clue that a suffix is generally useful is if many suffixes like it occur within its context. We compute the Jaccard similarity [] between each candidate and all other candidates in the set. We then define the *contextual local density score* of a candidate as the mean similarity with the top 10 most similar other candidates.

Unlike the remaining features, contextual local density is a score that depends on the other candidates and is in some sense functions similarly to the clustering performed by Grabski and Scheffer [2004]. However, one property of contextual local density is that no hard decisions have to be made – it is a real value that can be computed for each candidate.

Prefix match For each candidate c, we compute a *prefix match score*, given by $\log(1 + k_c)$, where k_c is number of trailing words in common between the prefixes of c and the current message. k_c may be larger than the initial candidate match k because there may be fewer than 5 suffixes with the larger k_c. Like in the candidate search process, we allow the prefix match to span sentence and paragraph boundaries.

N-gram likelihood When few words from the prefix match, some candidates may be ungrammatical or simply nonsensical. This feature was added when at one point the system suggested "come over for dinner tomorrow [morning...]", which a a trigram model might identify as unlikely. We define the *n-gram likelihood score* of a candidate as the mean log likelihood of each word in the resulting sentence. We use the probability estimates and smoothing weights computed by the n-gram baseline model, to be described later.

Subject similarity Often the subject of a message is very informative for predicting the resulting text, as in the motivating example in the introduction. We construct tf-idf-weighted word count vectors of the subjects of all messages. We then define the *subject similarity* as the cosine similarity of the subject vector of the current message with the subject vector of the message from which the candidate originated.

Quoted-message similarity If the current message is a reply or a forward, the content of the quoted message could be informative. As with subject similarity, we define the *quoted-message similarity* as the cosine similarity of the tf-idf vectors of the quoted message of the current message with that of the candidate, or 0 if either message lacks a quoted message.

3.3 Scoring candidates

We form a feature vector $\vec{x}$ for each candidate location (with respect to the current sentence prefix and context) from the features listed above. We score each candidate using a linear combination of the feature vectors $\vec{w} \cdot \vec{x}$. The system outputs the highest scoring candidate if its score exceeds a threshold α. Otherwise it outputs the null suggestion ϵ.

Since we do not know which candidates would be accepted by a human, and to smooth our acceptance criterion so that partial progress can be noted during training, we introduce a real-valued heuristic score h for a completion c with respect to true sentence completion c^*, described in Section 3.5. A large positive value for h indicates a good completion while a large negative value reflects a poor completion.

The null completion ϵ is always a candidate, with heuristic score 0. The feature vector of ϵ is identically 0 except in a designated coordinate where it is 1 and all other candidates are 0. This enables the system to learn when not to offer a suggestion, and it also serves the purpose of normalizing scores to rank confidences across examples.

3.4 Learning feature weights

To find a weight vector $\vec{w}$ that accurately scores candidates relative to each other, we identify a set of *opportunities* in the training set in which an exact match is possible, find the set of candidates and their vectors that our algorithm would consider at that point, and then use a margin-based approach to find a weight vector that best identifies the best candidate among the set.

We define an *opportunity* as a location within the training set where exactly M words remain in the sentence and those words also complete at least two other sentences in the training set. For the public email corpus discussed later, this definition yields 15,172 opportunities; in the author email corpus, this definition yields 2,233 opportunities. We randomly sample 500 of those opportunities for training.

Our learning algorithm follows a large-margin approach similar to that used in structured prediction [Taskar *et al.*,

2005]. For the ith opportunity, let the vectors corresponding to the candidates be $\vec{x}^i_1, \vec{x}^i_2, \ldots, \vec{x}^i_{m_i}$. For the moment, suppose that the b_ith candidate was a unique "best" candidate in the sense of maximizing its heuristic, for each opportunity. Then a noiseless max-margin optimization problem would be:

$$\min \|\vec{w}\|^2 \text{ such that } \forall i \forall a \neq b_i \; \vec{w} \cdot (x^i_{b_i} - x^i_a) \geq 1.$$

Here $1/\|\vec{w}\|$ represents the *margin* between each best candidate and the rest.

To tolerate model imperfection and noise, one replaces this hard constraint with the hinge loss and regularizes, yielding:

$$\min_{\vec{w}} \lambda \|\vec{w}\|^2 + \sum_i \max_{a \neq b_i} \left(1 - \vec{w} \cdot (x^i_{b_i} - x^i_a)\right)_+,$$

where $(z)_+ = \max\{z, 0\}$ denotes the positive part of a real z. Continuing with the approach of [Taskar *et al.*, 2005], we require a larger margin depending on the heuristic h. In particular, if the heuristic score of completion j on example i is h^i_j then we penalize its margin based on the difference $h^i_{b_i} - h^i_j$ instead of 1 for each example j, leading to:

$$\min_{\vec{w}} \lambda \|\vec{w}\|^2 + \sum_i \max_{a \neq b_i} \left(h^i_{b_i} - h^i_a - \vec{w} \cdot (x^i_{b_i} - x^i_a)\right)_+,$$

The above is the optimization problem we solve. Since $\max$ is convex, this is a convex optimization problem which is readily minimized using gradient descent on our bounded-size training sets. Since we are in low-dimensional space, the effect of regularization is minor and λ can be taken to be quite small. Finally, in the case where there are multiple candidates with maximal heuristic values, we choose the most recent one prior to the actual message.

3.5 Heuristic acceptability score

We employ a heuristic score to indicate the quality of a suggested completion with respect to the true completion. This heuristic provides a more refined notion of quality than exact match, and is also meant to correlate with the likelihood that a human would accept a completion. As mentioned, the heuristic score of a null completion (no suggestion) is defined to be 0, so a negative heuristic score indicates a case where one would prefer no suggestion to making that suggestion. The heuristic we use is defined as follows.

Say the true suffix is $w_1, w_2, \ldots, w_n$ and the candidate suffix was $u_1, u_2, \ldots, u_m$. We first determine how many common words they share in common from the beginning, i.e., $k = \max\{i \mid u_i = w_i\}$. Then the resulting heuristic value is defined to be:

$$h = -10 + \sum_{i \leq k} |w_i|,$$

where $|w|$ denotes the number of characters in word w. The constant 10 represents a cost of roughly 10 characters for showing a suggestion in the first place. For example, if the true completion was "Round the corner and take the first right" and the candidate was "Round the corner and turn right" the heuristic would be $|\text{Round}| + |\text{the}| + |\text{corner}| + |\text{and}| - 10 = 7$. Note that heuristic does not reward the completion for the fact that the words 'take' and 'turn' both start with the same letter.

In future work, one might train a classifier to better approximate the human notion of acceptability, and then test to see if that better approximation led to better results.

4 Experimental evaluation

We evaluate our suggestions in two ways, one automatic and one involving human annotations. The automatic evaluation is performed on a corpus of sent messages released by Florida Governor Jeb Bush, which we henceforth refer to as the *public corpus*. This corpus contains over two-hundred thousand sent messages from a single account, which is orders of magnitude greater than other publicly available corpora.[4] The second evaluation was performed interactively by one of the authors, henceforth referred to as *the author* on a corpus of thirty-thousand of their own sent messages (personal and professional). The former has the advantage of being complete on a large data set, but only exact string matches are counted, whereas the latter offers the possibility of human judgment as to what completions are acceptable.

4.1 Data Preparation

We preprocessed both corpora similarly, removing duplicate messages, greetings and signatures, parsing sentences, and tokenizing. We use the NLTK sentence tokenizer [Bird *et al.*, 2009] to split message bodies into sentences. Our tokenizer then outputs start-of-message, start-of-paragraph, and start-of-sentence tokens, splits words on spaces, slashes, and hyphens, and separates trailing punctuation such as periods and commas into separate tokens. We keep track of the actual string after each token in order to accurately reconstruct the original text.

Note that for counting the number of *words* in a suggestion and determining if it is a "big win," we use a simpler definition than our tokenization: words are delimited by whitespace. That is, we do not give our algorithm credit for completing a period at the end of a sentence.

We divided each of our corpora into a training set for learning feature weights and a chronologically later test set for testing the prediction algorithm using those learned weights. For the public corpus, we took the most recent 30,000 messages as a test set and the preceding 27,000 messages as a training set. For the author corpus, the ten years of email was divided into a training set of 26,838 messages (roughly nine years) and a test set consisting of 2,982 messages (roughly one year). Learning was performed on the training set and the learned weights were frozen on the test set.

4.2 No-Context Baseline

We compare our full algorithm that finishes sentences in the context of an email-in-progress with a variant that attempts to finish a sentence without knowledge about its context. In the "no-context" condition, the subject and quoted-message features are not included, and only tokens within the current sentence so far are provided to the algorithm. In particular, for finding candidates that agree as far back as possible, we do not check agreement across sentence boundaries.

4.3 N-gram Baseline

We also compare the proposed method with an N-gram language model baseline [Bickel *et al.*, 2005]. The language model estimates the probability of a word w_i given previous

[4] For instance, [Bickel *et al.*, 2005] studied an Enron employee's account with only 3,189 sent messages.

word sequence by a mixture of N-gram models, $1 \leq n \leq N$, $w_{i-1}, w_{i-2}, \ldots, w_1$ by

$$P(w_i \mid w_{i-1}, w_{i-2}, \ldots, w_1)$$
$$= \lambda_1 P(w_i) + \sum_{n=2}^{N} \lambda_n P(w_i \mid w_{i-1}, \ldots, w_{i-n+1}),$$

where the conditional probability is estimated on the training corpus. This is a commonly used technique for overcoming the sparsity of high-order N-grams. Following [Bickel *et al.*, 2005], we consider a 4-gram model and set the values of λ_1, λ_2, λ_3, and λ_4 to be 0.5, 0.3, 0.1, and 0.1, respectively, following [Bickel *et al.*, 2005], who tuned the parameters on the Enron dataset.

In the decoding phase, given a sentence segment $w_1, w_2, \ldots w_t$, the language model applies a beam search Viterbi algorithm to find the most likely completion with length $T - t$

$$\max_{w_{t+1}, \ldots, w_T} P(w_{t+1} \ldots w_T \mid w_t, w_{t-1}, \ldots, w_1)$$
$$= \max_{w_{t+1}, \ldots, w_T} \prod_{i=t+1}^{T} P(w_i \mid w_{i-1}, \ldots, w_1).$$

In order to compare completions with different lengths, we normalized the log-likelihood of each predicted completion by its length and use it as the score for the comparison. Like our instance-based approach, we restrict the length of the predicted string to be at least $M = 6$ words.

4.4 Automatic evaluation on public corpus

We run the "replay" scenario on a random sample of messages from the test set, requesting completions before each word (for words up to 500 characters into the message) from each of the three algorithms: our "full" algorithm and the "no-context" and "n-gram" baselines. For the automatic evaluation, we consider a suggestion to be accepted if it matches the original text exactly, ignoring whitespace, case, and punctuation.

Figure 1 shows the precision-recall curve for the public corpus computed by simulating the system at different levels of the confidence threshold. Including context benefits performance in most scenarios. For example, one message begins "Florida Governor Jeb Bush received the email you sent him and asked me to reply. The Governor is sorry to learn about the immigration concerns you describe. As you may know ..." The true completion was ", the federal government oversees visas, deportations, asylums, and citizenship and ..." The completions offered by each algorithm were:

full , the federal government oversees immigration, visas and deportations and governor bush ...

no-context , insurance company regulatory matters are under the administrative authority of the ...

ngram , insurance company regulatory matters are under the administrative authority

In this example, the context before the sentence (in fact the end of the previous paragraph), "immigration concerns you describe," enables the system to make a more relevant completion for "as you may know." Figure 2 shows that context especially helps early in the sentence.

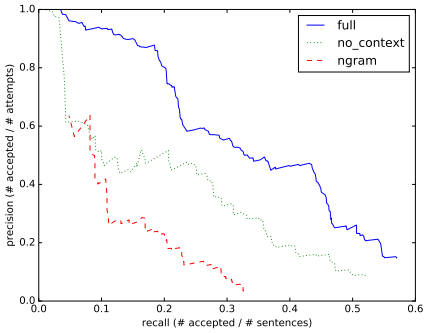

Figure 1: Precision vs recall for "big wins" on the public corpus. The "full" system performs better than the "no-context" and "ngram" systems, showing that context helps in many situations.

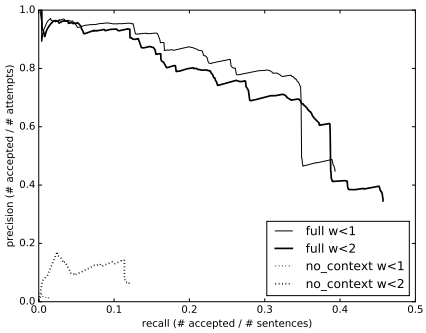

Figure 2: Precision vs recall on the public corpus for suggestions before the first ($w = 0$) or second ($w = 1$) word of a sentence. The superior performance of the full system indicates that context from prior sentences can aid prediction of the current sentence.

4.5 Human evaluation on author corpus

Inexact completions are often still useful (or entertaining) but judging that requires knowing context. For instance, the system suggested the completion: "My name is [xxx xxx and I work at xxx xxx, just down the road from you.]" when the author had in fact typed "My name is xxx xxx and I work at xxx xxx." In this case, the author did in fact work down the road from the recipient, so the suggestion was acceptable, but this relies on knowledge of the recipient's location (and also whether or not the author would choose to point that out). Several further illustrative examples are given in Table 1 So we asked the author of the emails to rate to completions on their own email. Since we automatically detect perfect match completions, we only need to ask for human judgments on the inexact matches.

For evaluating the suggestions, we automatically detected perfect matches. For the messages where one or more of the algorithms did not provide a perfect match, the author was shown the entire message and its context (i.e., metadata and prior messages in the thread). The author was shown

1. Do you guys want to (a) [come over to our place for ice cream later tonight?] (b) come over for dinner/lawn this evening?
2. Thanks so much for the invitation, but I won't be able to make (a) [it – it sounds like a lovely event but it's just too much travel this summer.] (b) it to the meeting.
3. My name is (a) [xxx xxx and I'm a xxx at xxx.] (b) xxx xxx and I work at xxx.
4. Cool, I think (a) [this is the beginning of a beautiful friendship.] (b) that was xxx's question.
5. Welcome home! (a) [I hope you had a great trip!] (b) How was your trip?
6. We are home! (a) [I hope you had a great trip!] (b) Heads up that we are thinking...

Table 1: Illustrative examples of completions (a) that differ from what was typed (b). In 1, the author liked the suggestion of ice cream rather than dinner. In 2, the proposed response was more polite than what the author typed. In 3, the proposed text was a simple variation of what was typed. In 4, the author was amused but would not have accepted the suggestion. In 5 and 6, the pattern that the system is using is evident. Note that there were numerous such failures as in 6.

suggestions incrementally, one word at a time, randomizing the order in which the different algorithm's suggestions were presented. The author was blind to which algorithm generated the suggestion. The author selected the completions that he would have accepted without modification if writing that email with an autocomplete tool. For efficiency, if it was identified that all algorithms provided big wins on a message, further suggestions for that message were suppressed.

Figure 3 shows the precision and recall for the author email corpus, quantitatively confirming that context helps make better predictions.

5 Discussion

In the public corpus, many of the completions appeared to be *canned text* that may have been inserted via copy/paste or some other semi-automatic mechanism. We considered attempting to filter out this sort of text, but opted not to, not just because it is difficult to identify canned text automatically. In formative studies, we observed that many heavy email users (e.g., university professors, administrative assistants, and receptionists) have canned text and appropriate a wide variety of techniques to re-find and use it in their email. we conjecture that an autocomplete interaction would be easier to use than finding and copy/pasting template text; our companion HCI paper explores this and other questions.

In the author corpus, our algorithm made many suggestions in situations that were clearly not copy-paste, though less frequently than for the public corpus since personal email tends to be more varied. Notable accurate suggestions included *information* such as addresses, directions, responses to common

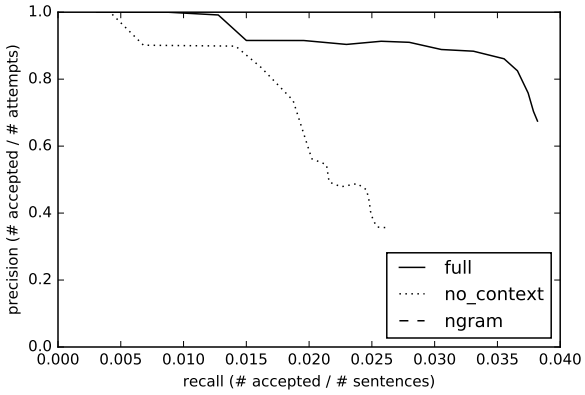

Figure 3: Precision vs recall for the author corpus. Far fewer suggestions were made overall than in the public corpus, reflecting the fact that personal email is more varied. Nevertheless, the author accepted many suggestions of 6 words or more, especially when the system was able to use context to help make those suggestions. The n-gram model, in contrast, did not make any successful suggestions of 6 words or longer.

questions, phone numbers, common requests, and *courtesies*.

The n-gram model performed much better on the public data set, probably because it contained significantly more exact copies of canned text. This canned text, however, only comprised part of the messages and personal information was interspersed. On the author corpus the n-gram model achieved near-zero performance–much worse than results reported in the prior work [Bickel *et al.*, 2005]. This is likely due to the lack of canned text combined with the fact that we focused heavily on long completions, and n-gram models often generated ungrammatical long text.

While a significant fraction of the public corpus could be autocompleted, only a small percentage of the author's sentences could be autocompleted with six words. However, such a system may still be valuable if it is high precision. In general, the percentage of sentences completed will vary by position and type of message, and should increase over time as systems become more intelligent and are able to integrate more and more information (e.g., calendars, social media, etc.).

6 Conclusion

In this paper, we demonstrated by existence proof that there are opportunities for "big wins" in sentence completion, even constrained to verbatim texts from prior writing. Moreover, simple ways of using the context of what someone is writing can help in finding and scoring these verbatim texts.

We propose intelligent interactive suggestion of long autocomplete as a practical challenge for further AI systems research. Future work in this area could explore generating completions, automatically identifying aspects of the completion that may need editing (such as dates or places), richer use of context such as recipients and specific details from prior messages, and extending beyond email to other authoring scenarios such as text messaging or writing articles.

References

[Bast and Weber, 2006] H. Bast and Ingmar Weber. Type less, find more: fast autocompletion search with a succinct index. In *SIGIR*, pages 364–371. ACM, 2006.

[Bengio *et al.*, 2003] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[Bickel *et al.*, 2005] Steffen Bickel, Peter Haider, and Tobias Scheffer. Learning to complete sentences. In *Machine Learning: ECML 2005*, pages 497–504. Springer, 2005.

[Bird *et al.*, 2009] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[Botha and Blunsom, 2014] Jan A Botha and Phil Blunsom. Compositional morphology for word representations and language modelling. In *ICML*, 2014.

[Grabski and Scheffer, 2004] Korinna Grabski and Tobias Scheffer. Sentence completion. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 433–439. ACM, 2004.

[Gubbins and Vlachos, 2013] Joseph Gubbins and Andreas Vlachos. Dependency language models for sentence completion. In *EMNLP*, pages 1405–1410, 2013.

[Hangal *et al.*, 2011] Sudheendra Hangal, Monica S. Lam, and Jeffrey Heer. Muse: Reviving memories using email archives. In *ACM User Interface Software & Technology (UIST)*, 2011.

[Kim *et al.*, 2015] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, 2015.

[Manber and Myers, 1993] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993.

[Nandi and Jagadish, 2007] Arnab Nandi and H. V. Jagadish. Effective phrase prediction. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 219–230. VLDB Endowment, 2007.

[Shokouhi and Radinsky, 2012] Milad Shokouhi and Kira Radinsky. Time-sensitive query auto-completion. In *SIGIR*, 2012.

[Taskar *et al.*, 2005] Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: A large margin approach. In *Proceedings of the 22nd International Conference on Machine learning*, pages 896–903. ACM, 2005.