Template Induction over Unstructured Email Corpora

Julia Proskurnia[†] Marc-Allen Cartright[‡] Lluís Garcia-Pueyo[‡] Ivo Krka[‡]
James B. Wendt[‡] Tobias Kaufmann[‡] Balint Miklos[‡]
[†]EPFL [‡]Google, Inc.
[†]iuliia.proskurnia@epfl.ch
[‡]{mcartright, lgpueyo, krka, jwendt, snufkin, bal}@google.com

ABSTRACT

Unsupervised template induction over email data is a central component in applications such as information extraction, document classification, and auto-reply. The benefits of automatically generating such templates are known for structured data, e.g. machine generated HTML emails. However much less work has been done in performing the same task over unstructured email data.

We propose a technique for inducing high quality templates from plain text emails at scale based on the suffix array data structure. We evaluate this method against an industry-standard approach for finding similar content based on shingling, running both algorithms over two corpora: a synthetically created email corpus for a high level of experimental control, as well as user-generated emails from the well-known Enron email corpus. Our experimental results show that the proposed method is more robust to variations in cluster quality than the baseline and templates contain more text from the emails, which would benefit extraction tasks by identifying transient parts of the emails.

Our study indicates templates induced using suffix arrays contain approximately half as much noise (measured as entropy) as templates induced using shingling. Furthermore, the suffix array approach is substantially more scalable, proving to be an order of magnitude faster than shingling even for modestly-sized training clusters.

Public corpus analysis shows that email clusters contain on average 4 segments of common phrases, where each of the segments contains on average 9 words, thus showing that templatization could help users reduce the email writing effort by an average of 35 words per email in an assistance or auto-reply related task.

Keywords

Human-generated email; structural template; fixed phrase extraction; templatization; suffix array generalization; ENRON corpus.

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052631

1. INTRODUCTION

Template induction, the technique of generating a skeleton of repeated content based on previously seen examples, has seen substantial success for structured content such as web pages, where metadata such as the underlying DOM[1] provides multiple presentational and structural signals that can be exploited algorithmically. These structures can be useful for tasks such as automatic labeling, plagiarism detection, duplicate detection, and structured information extraction. Despite the success of template induction for structured data, we have found little prior research for the same task, but for data that is not explicitly structured (i.e. plain text). This duality has difficult implications for a domain such as email. Despite often having some amount of structure, emails almost always contain some significant portion of freeflowing plain text that have, to date, not yet been sufficiently modeled in induced structured templates.

In this paper, we develop a template induction algorithm that focuses on the plain text content. We use email as the target domain, as we envision two potentially high-impact applications of templates generated for plain text content: 1) structured information extraction, where particular important pieces of information in the email are extracted; 2) email autocompletion, where the system suggests content to add during the composition of an email, based on the already present content the user has added; and 3) facilitation of the spam detection and filtering. Although the potential benefits of templatizing users emails are considerable (email autocompletion etc), targeting email documents provides nontrivial challenges in addition to the lack of explicit structure. Unlike the public domain for web pages, email documents are virtually always considered private, making it difficult to obtain training data. This is a sharp contrast to the freely observable and highly-structured web domain. Consequently, there are vastly fewer suitable datasets available for this kind of investigation, and we were unable to find any direct prior research on this topic. Given this setting, the primary task becomes one of establishing that template induction can be effective for the unstructured content. We look to further reduce the email response cost to users by investigating the usefulness of a template suggestion mechanism that is initiated when composing emails from a set of automatically constructed templates.

A template creation method consists of two parts: First, clustering similar messages. Second, for each cluster, determining the parts which are considered "fixed" and storing

[1]http://wikipedia.org/wiki/Document_Object_Model

the information in a standard representation, which is the produced template. In this paper, we focus on the latter.

To determine the fixed regions, we use an implementation of the suffix array data structure [24], which is efficient in space and time complexity, and can be easily parallelized. We show that the quality of produced templates created with our approach is consistently better than the baseline regardless of the quality of the clusters and with better latency performance. The results of the public corpora analysis determine that text suggestion would affect a significant number of users and would save them a significant volume of writing in an autocompletion task. In addition, we show that the portion of emails detected as fixed-text is larger than for the baseline, which would allow an automated information extraction system to focus on fewer parts of the emails when extracting transient information.

To determine the effectiveness of using a suffix array to generate templates, we compare it against a standard baseline approach for template creation. We test both methods on a synthetically generated corpus as well as a publicly available corpus of emails from the Enron Corporation[2]. The results of these experiments indicate that the suffix array serves as a superior approach to correctly identifying an optimal number and span of fixed regions.

The main contributions of this work are:

- An extensive analysis of the feasibility of email templatization for emails sent by a given user or bulk sender;
- A scalable technique that results in high quality templates regardless of the clustering quality;
- A novel application of the generalized suffix array algorithm to detect common phrases over similar emails; and
- An evaluation of the efficiency and quality of both the suffix array and baseline approaches.

The rest of the paper is organized as follows. Section 2 discusses prior work in the area of email template creation. Section 3 describes the template creation task in detail as well as our suffix-array based approach to generating templates from pre-formed clusters. Section 4 describes the experimental setup for comparative analysis, and concludes with the results of those experiments. Finally, Section 5 presents further applications and potential future work based on the results of this study.

2. RELATED WORK

In this section we discuss state of the art techniques related to this work. Specifically, we cover methods for email content mining (e.g. spam classification, labeling, and threading), template induction (for web and for emails), and autocompletion systems.

2.1 Email Content Mining

Most email content mining techniques are originally derived from more established web information extraction methods, such as template or wrapper induction [5, 11, 15, 16, 22]. In addition to information extraction, email content mining techniques also span document classification tasks, such as spam filtering, automatic labeling, as well as document clustering for email threading.

Spam classification. The popularity and necessity of email as a communication medium has also made email a popular target for spam attacks. Simple manual spam filtering rules have given way in the last decade to more complex and effective machine learning systems that now serve as the defacto defenders tasked with detecting and removing spam messages from inboxes [4, 21, 26]. More recently, these binary classification have expanded beyond textual classification to include much richer feature sets, collaborative filtering techniques, and peer-to-peer and social networking ontology-based semantic spam detection techniques [7, 8].

Label classification and ranking. The expansion of email as one of the primary communication media in both personal and commercial use has led to the notion of *email overload*, in which users become overwhelmed when the rate of incoming messages in their inbox outpaces the rate at which they can process those messages [9]. Automatic email foldering has been proposed to alleviate this problem [6, 14, 19]. However, these works present methods for email organization based on the underlying context of each message rather than occurrences of specific strings.

In many of these scenarios, sparsity of labeled data remains a hindrance to accurate and robust model development. Kiritchenko and Matwin deal with the sparsity issue by using a co-training algorithm to build weak classifiers, then label unlabeled examples, and add the most confident predictions to the labeled set [13]. Somewhat similarly, Wendt et al. utilize a graph-based label propagation algorithm to label unlabeled emails from a small set of labeled emails, but do so at the template level to improve scalability in very large mail provider systems [25].

Possibly one of the most popular and widely known automatic email organization systems is Google's Gmail priority inbox, which distinguishes between important and non-important emails by predicting the probability that the user will interact with the email (e.g. open, respond) within some time window from delivery [1].

Threading. Email threading is another solution to the *email overload* problem that assists in inbox organization and can furthermore reduce the user's perceived inbox load by clustering emails from the same conversation together [18]. Current threading techniques cluster messages together by header information, such as sender, subject, and subject prefixes (e.g. 'Re:', 'Fwd:'). Personal correspondence emails are generally threaded very accurately using this technique, however, many commercial emails, such as purchase receipts, tracking numbers, and shipment confirmations are often split into multiple threads due to their different subject lines despite arriving from the same sender domain and belonging to the same semantic thread. Ailon et al. presented techniques to thread such commercial emails through leveraging email templates and learning temporal causal relationships between emails from similar senders [2]. Similarly, Wang et al. attempt to recover implicit threading structures by sorting messages by time and construct a graph of conversations of the same topic, however their analysis is limited to newsgroup style conversations [23].

2.2 Template Induction

Web data is generally formatted in a human-readable format which a machine renders might not necessarily understand. Web extraction techniques have been proposed to solve this issue of extracting information structured for human consumption from data.

[2]https://www.cs.cmu.edu/~./enron/

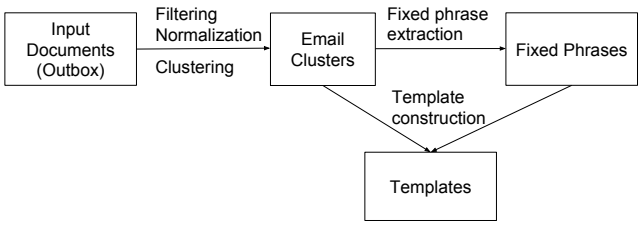

Figure 1: An overview of the clustering and template induction.

The web is comprised of over a trillion documents in the public domain [3], many of which are dynamically created and generated using templates. Hence, web information extraction is a well explored topic and is often closely coupled with template induction techniques. Since many emails utilize HTML markup and it is estimated that nearly 60% of emails are created from templates (e.g. B2C emails) [2], it is natural to also apply similar web-based techniques for template induction and information extraction on emails. However, emails are among the most sensitive data on the Internet. Hence, little research has been presented on email template induction due to privacy constraints, although recent work has proposed methods for enforcing anonymity in web mail auditing [10].

While there is very little published work on using structural templates for processing commercial email data, templates have been used for annotating semantic types within the DOM trees of emails [27] and used in hierarchical classification of emails [25]. To our knowledge, no techniques have yet been published that propose template induction for plain text email content.

[17] summarizes the short messages about same events into their shorter representaton. However, it has two major drawbacks that are not suited for the email use case. First, clustering of the messages relies on the edit distance which does not scale well over average size emails and might produce clusters where all words are different. Second, the method requires strong preprocessing.

2.3 Assisted Email Composition

SmartReply is the first email-specific machine-learned tool designed to assist the user in composing email. The tool is built on recurrent neural networks (one to encode the incoming email and the other to predict possible responses) that automatically suggests replies to email messages [12]. The work presented here aims to provide assistance that is learned specifically from an individual email sender, whereas SmartReply provides canned responses meant to satisfy as many users as possible. In a assisted email composition context SmartReply would infer intent, while our work would detect content that was written before by the same sender.

3. METHODOLOGY

The design of the template induction system is shown in Figure 1. We first describe the preprocessing and clustering steps which generate the input for the template induction algorithms. We then present two approaches for the fixed phrase extraction, as well as template induction algorithm.

3.1 Preprocessing

During the preprocessing phase, we perform necessary enrichment and filtering of the raw input emails to make them suitable for use as experimental input for comparison.

For each email in the raw corpus, we determine if the message is a reply (its subject begins with the pattern `re:`) or a forward (its subject begins with the pattern `fwd:`) of a prior email. All other emails are treated as original messages. Any non-original message is analyzed, and if all of the content of the message is simply a quote of the original message, the email is filtered out. Of the remaining emails, all messages determined to be non-English (or consisting primarily of non-English content) are additionally filtered out. Each of the remaining emails is then tokenized and each produced token is a structure that carries the original form of the token as well as its lemma, using the WordNet database for lemmatization [20].

3.2 Clustering

After normalization and filtering, we then cluster similar emails as follows. Given the task at hand, a natural basis for similarity is to first group all emails sent from the same email address together. Each of these initial groups is then further processed to produce the final set of email clusters, which are then passed as input to the template generation algorithms.

Formally, let D_i and D_j be two emails in our corpus, represented as term count vectors. We define the distance between two emails:

$$\delta(D_i, D_j) = \left\| \frac{D_i}{\|D_i\|_1} - \frac{D_j}{\|D_j\|_1} \right\|_2 \quad (1)$$

which we use as a measure when constructing the final set of clusters. Note that based on this definition, $0 \leq \delta \leq \sqrt{2}$ ($\sqrt{2}$ indicates orthogonal document vectors). A maximal distance indicates that two documents have disjoint vocabularies, while minimal distance indicates that, at the very least, the token frequency histograms of the two documents are identical. This measure does not account for token ordering differences; we assume that statistical similarity is sufficient when considering documents from a single sender.

We use the distance defined in Equation 1 and a given distance threshold $\theta \in \text{range}(\delta)$ to partition each input sender-based group into smaller clusters based on email distances. Using these definitions, we present Algorithm 1, which receives an input cluster C_{in} (e.g. every email sent by a given user or bulk sender), and a threshold θ to produce a set of output clusters $\mathbf{C}_{\text{out}}$[3].

The InitializeCluster function adds a new cluster to the output set $\mathbf{C}_{\text{out}}$, and places D in the new cluster as the "representative" of that cluster. The Rep function returns the assigned representative email of a given cluster, e.g., the first email added to the cluster. While we could drop this assumption and attempt to find the best representative email for the entire cluster, performing this operation reduces to an instance of the set cover problem, which is

[3]The large volume of emails in a real-world scenario makes the usage of complex clustering techniques not scalable. Although cluster creation is not the focus of this work, we observed that Algorithm 1 and k-means clustering did not show any significant differences in the quality of the resulting clusters, however, the method in Algorithm 1 showed a significant gain in speed.

Algorithm 1 Creating distance-based clusters.

```
ProcessClusters(C_in, θ)
1  C_out = {}
2  for D ∈ C_in
3      if C_out == ∅
4          InitializeCluster(C_out, D)
5      else C_best = argmin_{C∈C_out} δ(D, Rep(C))
6          if SIM(D, Rep(C_best)) > θ
7              C_best = C_best ∪ {D}
8          else InitializeCluster(C_out, D)
9  return C_out
```

known to be NP-complete. Although we may be able to find the optimal representative for some smaller-scale sets, such an approach would not scale to real-world collections. Therefore we use the pre-selected representative to keep the problem tractable.

Line 1 initializes an empty result, where the main iteration over the input cluster emails occurs on lines 2–8. We initialize the first output cluster with the first email in line 4. For each subsequent email, we calculate the distance of the email from all the representatives. If the best score is greater than θ, the email is added to the cluster with the best score; otherwise it creates a new cluster, with the email set as the representative (lines 5–8). When all emails are processed, the resulting set of clusters is returned in line 9. Each of the final clusters represents a "template" of an email - each email in the cluster is a minor variant of a mostly static template.

3.3 Baseline Phrase Extraction

We now describe the two methods we compare when conducting experiments. In both cases, we assume the input documents have already been clustered based on our selected distance metric. The task is to determine the parts of content that are frequent enough in the cluster to consider them "fixed", which we can subsequently use to construct a template representing the clustered emails.

We adopt a greedy version of a longest common subsequence algorithm as a baseline approach. To identify fixed phrases for the templatization task, our baseline algorithm operates sequentially on emails of a given cluster. Pseudocode of the algorithm is presented in Algorithm 2. We define the input $0 \leq \gamma \leq 1.0$ as a threshold for determining if a token is fixed. Additionally, let t_i be the i^{th} token in an email vector i.e., $D = t_0 \dots t_{|D|-1}$. A token will be classified as fixed if it is present in $\gamma * |C|$ documents. Given a cluster $C \in \mathbf{C}_{\text{out}}$ from the clustering step, and a value for γ, the algorithm will return a set of terms considered as fixed for the given cluster.

On line 1, we iterate over C to calculate the document frequencies of the unique tokens in C, and store the tabulated data in *df*. Line 2 initializes the *acc* variable, which acts as an accumulator for the frequency counts of observed phrases. Lines 3–11 loop over every email in C, filling *acc* with candidate phrases. The function InsertInc inserts the entry if it does not exist, or increments the count of the existing entry. Lines 5–11 construct the candidate phrases by iterating over the token sequences from the current D. The *phrase* (line 4) variable tracks the "current" candidate

Algorithm 2 Baseline method for finding fixed text.

```
BaselineExtractPhrases(C, γ)
 1  df = DocumentFrequencies(C)
 2  acc = {}
 3  for D ∈ C
 4      phrase = []
 5      for i = 0 to |D| − 1
 6          if df[t_i] ≥ γ * |C|
 7              Append(phrase, t_i)
 8          else
 9              if |phrase| > 0
10                  InsertInc(acc, phrase, 1)
11              phrase = []
12      if |phrase| > 0
13          InsertInc(acc, phrase, 1)
14  RemoveInfrequent(acc, γ * |C|)
15  return Keys(acc)
```

phrase. If a token has a high enough document frequency, it is appended to the current phrase (lines 6–7). Otherwise if the current phrase is non-empty, its count is incremented in *acc*, and the phrase list is reset to empty (lines 9–11). The final step on line 14 involves iterating over the entries in the *acc* variable, and removing any candidate phrases that are below the $\gamma * |C|$ threshold.

One of the major advantages of this approach is that the space and time complexity constraints of the algorithm are both linear with respect to the input size. Both aspects of the algorithm are $\mathcal{O}(|C||D|_{\max})$, where $|D|_{\max}$ is the length of the longest email in C. Constructing *df* requires a scan over each email in C, while the execution of Algorithm 2 is a linear scan over the emails in C. In terms of space complexity, the greedy construction of the candidate phrase table makes the space requirements be linear as well.

3.3.1 Limitations

While Algorithm 2 is straightforward in its execution, it suffers from the assumption that frequent tokens tend to co-occur with each other. Only the longest identified phrases will be computed for the given emails; any subphrases that may also pass the frequency threshold, but are not in the same order or consecutive, will not be included in the output set of fixed phrases unless they occur as unique phrases elsewhere in the emails. For example, if the current tracked phrase is "Hi your order is here", the potentially higher-frequency phrase "your order is here" will not be added as a resulting fixed phrase simply because it occurs as a subphrase. Let us consider the following two clusters with 5 emails each, with 5 words in each document:
C1: (A B C D E),(A B C D F),(A B C D G),(A B Q C D)
C2: (A B C D E),(E A B C D),(A X C Y E),(A K C L E)

Let $\gamma * |C| = 2$ for this example. For the first cluster *C1*, the resulting fixed phrase would be A B C D, since it is greedily constructed and passes the frequency checks. However, because A B C D was greedily constructed with the first three emails, the more useful fixed phrase pair A B and C D are not considered, resulting in a template with suboptimal coverage over its constituent emails. For the

second cluster *C2*, the resulting fixed phrases would be A, C and E, while more optimal A B C D and E are not emitted.

3.4 Suffix Array Based Approach

In order to overcome the limitations discussed in the previous section, we introduce an algorithm to perform template induction based on a suffix array. Since the performance and precision of the extracted phrases play an important role in template extraction and user profiling, we examine both the quality and scalability characteristics of this approach.

Our approach uses two main data structures to compute fixed phrases. The first is a suffix array (SA), which is the lexicographically sorted array of all suffixes of the input documents, i.e., pointers to the original positions. SA typically operate over the character space of the input, however in order to ensure that we produce valid fixed phrases, we need the SA to operate on the token space. Therefore, when constructing the SA, we only permit suffixes to be added at standard token separators such as whitespace and punctuation. The second data structure is an array of the longest common prefix (LCP), which is produced while computing the SA. Entries in the LCP correspond to the number of common characters in the prefixes between two consecutive suffixes in the SA.

Algorithm 3 provides a sketch of the fixed phrase selection process using SA and LCP. We define $\mu \in \mathbb{N}$ to be a threshold for the minimum number of shared characters allowed between two suffixes, and provide it as an input argument. In line 1, we initialize an accumulator *acc*, and we construct the SA and LCP structures. In this instance, *acc* has the added functionality of maintaining the insertion order of the entries, which will be needed in Algorithm 3. Practically speaking, this can be achieved by internally maintaining both a table and list to accomodate both access patterns.

The main loop of the algorithm (lines 3–15) iterates over the contents of the SA, using each iteration to examine a particular suffix and then decide whether to track it as a candidate fixed phrase. On line 4 we only admit suffixes that have a high enough overlap with the previous value; all admitted suffixes then have their counts incremented on line 5[4]. The PHRASE function extracts the actual phrase from the input by its position (SA) and length (LCP).

The next action depends on the delta of the "current" LCP; when there is a decrease in the LCP, we know that the currently tracked phrase has ended, and we need to check its frequency (lines 6–11). The completed phrase's count is updated (lines 7–11). The PREVLARGER function emits all previously inserted *acc* entries as long as the corresponding LCP values are higher than the current value. Similarly, the PREVSMALLER function emits contiguous prior entries with lower corresponding LCP values. If the frequency of the recovered phrases is below the $\gamma * |C|$, then the phrase is dropped from *acc*. Alternatively, if the LCP increases, this indicates that a new phrase has started, and we need to increment (or insert) it and all contained subphrases (lines 12–15). In the case where the LCP delta is zero, no additional action is taken. After the loop completes, the remaining keys in *acc* are returned as the fixed phrases (line 16).

[4] Since all the documents are treaded as a single string during the SA construction, we maintain an additional data structure D that contains indexes of beginnings of the documents. Thus, ids of the documents are obtained with SA.

Algorithm 3 Fixed phrase extraction based on SA, LCP.

```
EXTRACTPHRASES(C, γ, μ)
1  acc = {}; SA, LCP = BUILDSUFFIXARRAY(C)
2  INSERTINC(acc, PHRASE(SA[0], LCP[0]), 1)
3  for i = 1 to |SA| − 1
4      if LCP[i] > μ
5          INSERTINC(acc, PHRASE(SA[i], LCP[i − 1]), 1)
6          if LCP[i] < LCP[i − 1]
7              c = acc[PHRASE(SA[i], LCP[i − 1])]
8              INSERTINC(acc, PHRASE(SA[i], LCP[i]), c)
9              for phrase ∈ PREVLARGER(acc, LCP[i])
10                 if acc[phrase] < γ ∗ |C|
11                     REMOVEKEY(acc, phrase)
12         elseif LCP[i] > LCP[i − 1]
13             INSERTINC(acc, PHRASE(SA[i], LCP[i]))
14             for phrase ∈ PREVSMALLER(acc, LCP[i])
15                 INSERTINC(acc, phrase, 1)
16 return KEYS(acc)
```

ID	LCP	SA	
1	9	1	A B C D E...
2	9	11	A B C D F...
3	5	21	A B C D G...
4	0	31	A B Q C D...
5	7	3	B C D E...
6	7	13	B C D F...
7	3	23	B C D G...
8	0	33	B Q C D
9	...	47	C D
...	...	...	...

" A B " : 4

" A B C D " : 3

" B C D " : 3

Table 1: LCP, SA and actual suffixes ordered lexicographically. Sentence starts with space.

Let us again consider the simple example from the end of Section 3.3.1. Table 1 provides a view of the contents of the SA and LCP after constructing them over the example emails. Recall that the SA is an array with pointers to the suffixes' positions in the original input, (e.g., 1, 11, 21, and so on), and the LCP stores the number of shared characters between two consecutive suffixes. For the course of the example, we make the following assumptions: the cluster contains only the suffixes present in Table 1, each suffix comes from a different document [5] , and $\gamma * |C| = 3$.

We start with the first suffix "A B C D ", which belongs to the first document. The suffix has an overlap of 9 characters with the next suffix (i.e., there are at least two documents that have a repeated phrase of length 9 including spaces) and thus we insert the phrase into *acc* with count 1 (line 5). As we progress in the example, we see the decrease of the LCP between IDs 3 and 4 which is valid, i.e., number of remaining overlapping characters passes μ. A negative LCP delta indicates that the suffix has shortened, and the current phrase has ended. In this case, its accumulated frequency is checked, the phrase passes, and it is emitted. We observe that the LCP value drops to 5 characters, thus the count of the corresponding suffix should include the frequency of the previous larger phrases, and *acc* is updated accordingly (line 7-8). As a result, "A B " is added to *acc* with count

[5]In the algrithm 3 we proceed ensure that phrase increment only happen for phrases in different document.

3. When the LCP increases, the frequency of the current tracked phrase should continue to accumulate and longer phrases should be added to *acc* with count 1 (lines 12–15). Following this example to its termination, we would produce A B C D, A B, C D for the first cluster and A B C D, E for the second one, which prove to be more optimal selections than those produced by Algorithm 2.

The computation complexity consists of two factors: suffix array construction and fixed phrase extraction loop. The former is $\mathcal{O}(|C||D|_{\max})$ complexity. The latter is proportional to the input size, or more preciesly, proportional to the number of words in the input. Overall, the complexity is $\mathcal{O}(|C||D|_{\max})$.

The algorithm also has modest space demands, since it only requires the following input: the suffix array and the longest common prefix array, both of which are proportional to the input size and store only pointers to the original input. During operation, the only data structure maintained is the output argument, the final set of fixed phrases.

The algorithm affords several opportunities for parallelization. For example, each part of the suffix array starts with a different letter, and can be processed independently. Additionally, instead of partially rescanning the suffix array when there is a change in the current LCP value, a new process/thread can be spawned that will be responsible for updating the occurrences of the bigger phrase.

3.5 Constructing a Template from Phrases

We now describe the process of building a template based on the induced fixed phrases. We design the template itself to be an ordered list of fixed and non-fixed phrases. We define the *coverage* of a given template/email pair to be the fraction of characters of the email that can be aligned with the fixed text present in the template. It is an important measure since it describes an email comression, i.e., number of characters that might be saved while typing or while representing an email through the fixed and non-fixed regions etc.

The main task is to align the fixed phrases with the following optimization criterion: choose a set of non-overlapping phrases that maximize the coverage over the email. Fundamentally the process of template building involves dynamic programming to align the fixed phrases of the template onto the email.[6]

When the alignments of the phrases to the email are found, the email is transformed into the sequence of the matched fixed phrases separated by the parts of the template which were not mapped to a fixed phrase, and are presumed to contain variable content. For example, if we have 2 fixed phrases: "hi", "how are you" and the email is "hi John, how are you", we would like to generate the template "hi _____ how are you". Example templates generated using our techniques on the Enron corpus are presented in Table 2.

Overall, the quality of the templates is hard to evaluate and can be either performed by humans or automatically. Human evaluation is rather expensive and error prone. Therefore, we rely on the coverage metric as a reflectino of the cluster edit distance.

[6]A more detailed description is omitted due to complexity and limits on space.

Template 1: The report name : _____ p / l _____ , published as of _____ / 2001 is now available for view on the website .
Template 2: We have received the executed Letter Agreement date _____ / 2 _____ 00 amending the ... . Copy will be distributed .
Template 3: Please , put it on my _____ . Vince

Table 2: Examples of the templates created based on SA. _____ correspond to the non-fixed regions of the templates.

4. EXPERIMENTS

In this section we describe the experimental setup, data sources and insights we have obtained from both the baseline and suffix array based approaches. We perform two main experiments to analyze the quality of the templates produced by the two presented methods. We conduct the same experiment with two separate corpora: a synthetically generated corpus and a real email corpus from the Enron Corporation. We continue to use the *coverage* over emails for quality assessment, and we introduce the *template entropy* measure, which is the proportion of fixed phrases that a template contains. More fixed phrases in the template indicate more variable content separating it, thus introducing more uncertainty which we aim to minimize. Using these measures and given an induced template, our goal is to maximize coverage and minimize template entropy. To test the robustness of the proposed approach we have performed multiple runs varying clustering threshold θ and fixed phrase document frequency γ.

4.1 Synthetic Corpus

Since the algorithm could be run over all user's emails, it is important to know how sensitive the template creation algorithms are to cluster quality. We test both methods against clusters of varying quality, so in order to perform these tests while minimizing confounding factors from the input, we use a synthetically generated corpus of emails, which is constructed as follows. We automatically generated 3 independent sets of 100 emails each. For each set we create 10 clusters with 10 emails each. For a given cluster, we selected a set of predefined "fixed" phrases that we then separated in each email by randomly generated text that acted as the "non-fixed" portion of the email. For example, in a cluster with two predefined fixed phrases, we randomly chose phrases to put before, between and after the fixed phrases for each email in the cluster.

To ensure that uniform email size did not confound our results, we create another synthetic set where the cluster sizes are distributed normally instead of uniformly. Table 3 provides details on the generated corpora. The average coverage per template of cluster sizes is about 80%, and the average template entropy is 3. The randomly generated emails had an average size of 260B, which is in the 92nd percentile of email sizes in the public corpus.

We characterize the templatization performance by varying two parameters that are specified prior to processing as described in Section 3: the clustering threshold θ and the fixed phrase document frequency γ. We found that with $\theta \leq 0.6$ all synthetic emails were grouped into one cluster, while for $\theta = 0.7, 0.8, 0.9$ average cluster sizes were 20, 11, 6 respectively. Since generation of the fixed and variable

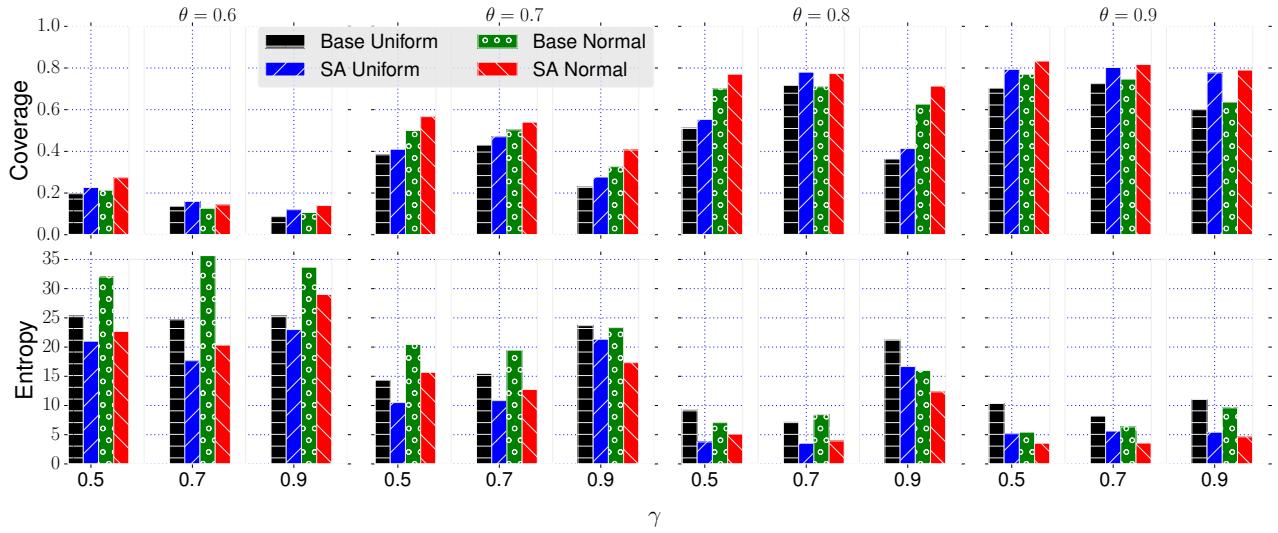

Figure 2: Average template coverage and entropy for two synthetically generated corpuses, i.e., equally and normally distributed cluster sizes. We present the coverage by varying clustering threshold θ and fixed phrase frequency γ. Suffix Array based approaches consistently show higher coverage (portion of the email marked as fixed) with lower Template Entropy (number of fixed-phrases per template). Template coverage of a cluster is computed for each email and averaged for the whole cluster.

Metric	**Syn1**		**Syn2**		**Syn3**		**Mean**	
Cluster size	U	N	U	N	U	N	U	N
Coverage (%)	75	85	82	79	79	82	79	82
Entropy	3	2.7	4	4	2.7	3.2	3.2	3.3
Characters	178	246	148	162	249	177	192	195

Table 3: Average characteristics of the generated synthetic corpus. U and N correspond to the uniform and normal distribution of the cluster sizes.

parts in the synthetic emails is performed randomly, multiple clusters may share similar fixed and variable parts, which might result in over- or under-clustering. $\theta = 0.8$ showed the closest cluster distribution to the expected results. The results are shown on Figure 2 using our two selected measures. The higher the clustering threshold is set, the more template coverage converges towards a steady value for both methods. Similar behavior is shown for the template entropy; as cluster quality increases, templates are built over more homogeneous sets of emails and therefore have a more compact structure.

We can see the consistent improvement over the baseline approach both in terms of email coverage and template entropy. As shown on the synthetic corpus results Table 3, the expected coverage and entropy are 79% and 3.2 respectively for the first batch of generated emails. We can see that by varying both θ and γ, the suffix array based approach maintains dominance across all metrics. This consistently improved performance stems from the fact that the suffix array provides a higher quality set of candidate fixed phrases to use in template construction. Overall, having more phrases is beneficial when it comes to user profiling and better template construction even if the higher number of phrases requires an increase in extraction time.

4.2 Enron Corpus

For the second series of experiments we used the publicly released email corpus from the Enron Corporation. The corpus contains a large database of over 600,000 emails generated by over 150 users, mostly from senior management of Enron. For the experiment we extracted all of the sent emails of the corpus users. This resulted in over 125,000 emails with more than 300 distinct email addresses[7].

We preprocessed the corpus as described in Section 3. Multiple statistics are collected during clustering, such as user outbox (sent mail) counts, cluster quality, and so on. We do this in order to provide a sense as to whether the clusters formed would be suitable for template induction. By exposing an explicit notion of the quality of the clusters, we avoid the possibility of an unknown feature acting as a confounding factor when evaluating the induced templates.

To measure quality of the clusters themselves, we use a variant of the "edit distance" (ED) measure. For an output cluster $C \in \mathbf{C}_{\text{out}}$, we calculate the ED of the cluster as:

$$\text{ED}(C) = \frac{1}{\text{chars}(C)} \sum_{D \in C} \text{CharED}(\text{Rep}(C), D) \qquad (2)$$

where $\text{chars}(C)$ is the sum of the character lengths of all member emails in C. We rely on this metric to emphasize the compression of the outbox and thus typing reduction of the repetitive parts. When a cluster has a body edit distance of less than 20% of its average content length and contains at least 5 emails, we call that cluster a "high-quality" (HQ) cluster. Based on this analysis and definition we obtain the following corpus statistics shown in Table 4.

We break down the user sent mail volume into deciles to get a sense of how many users could be characterized as "high-volume" senders, as shown in Table 4. Only active users that fall into the 40th percentile and higher by their

[7]We treat multiple corporate accounts of a user as separate entities since each account is used for a distinct purpose.

User deciles	Number of users	Total outbox size	Average outbox size	Average email length	Average cluster size for $\theta = 0.8$	High quality clusters for $\theta = 0.8$
0	79	79	1	284.81	1	0
20	19	50	2.63	372.21	1.97	0
30	28	310	11.07	1,352.31	3.22	0
40	31	1,372	44.26	589.07	5.15	2
50	32	2,885	90.16	641.81	5.63	2
60	32	5,886	183.94	483.96	4.8	7
70	31	10,809	348.68	618.29	6.16	11
80	33	23,377	708.39	409.21	4.56	53
90	31	80,336	2,591.48	570.87	7.84	102
Aggregates	**316**	**125104**	**442**	**591**	**4.48**	**177**

Table 4: Description of the Enron sent mail corpus.

sent mail volume are capable of producing templates that are of suitable quality, and as the decile increases, the number of HQ clusters grow superlinearly. The data in the table suggests that a user would need at least ∼40 emails to produce any useful templates. Given the increasing prevalence of email for communication, this suggests that templatization could be useful for virtually every current user of email.

The first two deciles did not have more than one email sent which is shown in the 4th column. Interestingly, the average size of a sent email is significantly greater in the third decile compared to others. This observation is driven by the data, i.e., users with a relatively low outbox size (∼11 emails on average) tend to send mainly long annual reports. Moreover, the average size of the user outbox correlates with the tendency to write similar emails.

We also investigated whether the average edit distance in a cluster behaved as a function of the cluster size, as shown in Figure 3. Indeed, the greater the size of the produced cluster, the lower the chance to observe a small edit distance within the cluster. The two exceptions to this observation occur at cluster sizes 10 and 100, which consist of annual reports (within a cluster) that only vary in the month mentioned in the email.

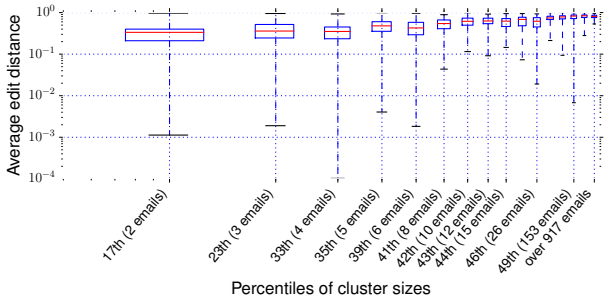

Figure 3: The average ED as cluster size increases for $\theta = 0.8$

Using the edit distance as defined, we find that the edit distance for the bodies and subjects of the emails in a given cluster have a Pearson correlation coefficient of 0.31. As the edit distance increases for the subjects, the body edit distance tends to increase as well. The trends for subject vs. body are presented in Figure 4. Such information could be used in email clustering and cluster filtering, or for efficiency gains when performing clustering at large scale.

Similarly to the synthetic corpus experiments we performed template induction for the baseline and SA based approaches. As a result of clustering with $\theta = 0.8$ we obtained over 25,000 clusters. As expected, the higher the cluster threshold, the more clusters are produced. In particular, we observe strong

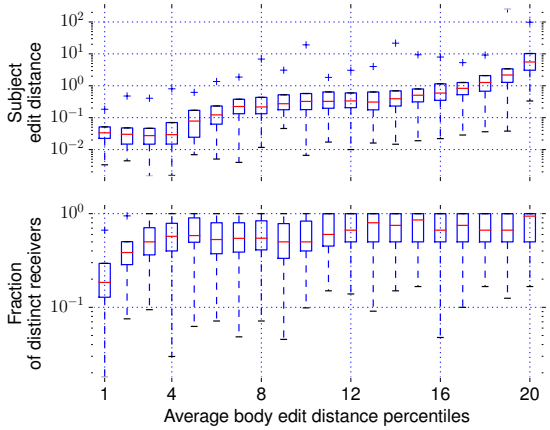

Figure 4: Correlation between average body edit distance and average subject edit distance within cluster and fraction of distinct receivers (1 - all receivers are distinct) respectively.

positive correlation of 0.91 between θ and the number of clusters produced, while the synthetic data has a moderate positive correlation of 0.62. More prolific users (top deciles) tend to write more similar emails as described in Table 4. We tested the variations of the SA approach when limiting the size of the accepted fixed phrase (2, 3 words).

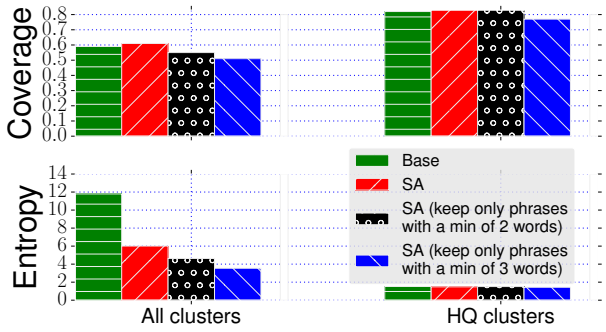

Figure 5: Average template coverage and entropy for $\gamma = 0.8$ and $\theta = 0.8$ over Enron corpus all and high quality clusters.

As can be seen from Figure 5, the suffix array based approach performs better than the baseline alternative in terms of coverage and entropy. We show only one pair of values for θ and γ here, but we observed similar behaviour for other values of these parameters. By varying the constraints on the phrase quality, we show that it is possible to balance between the coverage and the entropy of the template. For

example, the most restrictive results (SA with phrases with a min of 3 words) have the least template coverage. However, this significantly reduces the number of fixed regions in the template. Overall, average entropy for various setups of our approach is ∼4, while maintaining the coverage within 60%. Considering an average email length of 600 characters and average word size to be ∼10, we obtain a ∼35 word reduction in typing in an autocompletion setting for users with induced templates.

Our findings indicate that using SA for template induction offers better performance than the baseline both in terms of email coverage and template entropy. As can be seen in Figure 2, where expected performance is ∼80% and ∼3 for coverage and entropy respectively, the SA based fixed phrase extraction shows better results for both metrics no matter the quality of the cluster. Similar behavior is observed for the Enron corpus and shown in Figure 5.

4.3 Scalability Analysis

To test the performance characteristics of both approaches we performed fixed phrase extraction over various cluster sizes. Even though this work focuses on the template extraction for a single user account the size of the outbox should not be neglected, since it could reach millions of emails. We kept each email size to be almost identical in size (approx. 1 KB each[8]). We varied the cluster size from 2 KB to 33 MB with the corresponding number of emails from 2 to 33,000. Figure 6 shows the execution time taken to create templates as the size of the input cluster increases. When the clusters are relatively small, the methods are equivalent in efficiency. However, the growth trends depicted in Figure 6 clearly show that the baseline approach takes longer to complete than the SA approach as the cluster size scales up. The baseline approach proves to be sensitive to both the number of emails in the cluster and email size variations within the cluster. The SA is agnostic to these variations due to the fact that the input is treated as a blob of information for which the SA is built and valid phrases are added to the result.

Additionally, one can easily see that the growth of the baseline is also superlinear - the baseline requires less than 200 seconds for 5K emails and 400 seconds for 15K emails, but requires over 1600 seconds for 30K emails. While the earlier segment has a slope of approximately 2/3 (2x time for 3x input), the next segment is closer to a slope of 2 (4x time for 2x input). This suggests that the slope will continue to grow as the input size increases. The SA approach shows a slight slope increase as well, but it is multiplicalely less than the baseline, making the SA approach more scalable.

5. CONCLUSIONS AND FUTURE WORK

We have demonstrated the feasibility of performing high-quality plain text template induction using proposed highly scalable solution. The experiments, perfomed on both a synthethic and organic email corpora, illustrate the efficacy of using suffix arrays to induce templates, even in the face of input clusters of varying quality.

We continued our investigation by then comparing two template induction algorithms. We have shown both in theory and in practice that using a suffix array is more effective

[8]1 KB is a upper bound on the 95% of the emails send in our corpus.

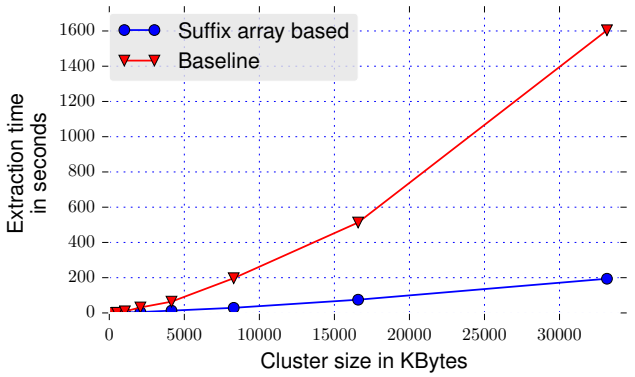

Figure 6: A comparison of the increase in extraction size as the average cluster size increases.

than an out-of-the-box shingling baseline for template induction. The results of our investigation show that plain text documents can be templatized more efficiently using suffix arrays: the baseline showed superlinear growth, while the suffix array's growth is multiplicatively slower. Additionally, the templates induced using the suffix array encode more useful information than the greedy approach: across our experiments the suffix array templates provided consistently better coverage than the greedily-built templates. Our ancillary experiments suggested that if used for email autocompletion, the generated templates could on average save 35 words of typing when composing emails. Overall the presented work is salient for numerous applications, including optimizing the production of content, extracting information from machine-generated content (imagine if the "user" sending the email is an algorithm written by an online seller), and profiling composition behaviors of users. We only considered forming templates for individual users or bulk senders, however we would like to further explore 1) accross user clustering to induce templates; 2) applications of the created templates. This could allow an even higher email compression and carry insights into the composition behaviors of users at an aggregate level.

Our initial implementation relies on a simple selection the representative document of a cluster, but we would like to further explore alternative methods to find a "near-best" representation of a given cluster. Although we have shown that suffix array templates are more robust to cluster quality than greedily built templates, the clustering method affects the number of clusters produced, potentially creating many more clusters than there should be.

Finally, although we discussed and analyzed the efficiency of using the suffix array for template induction, our collection sizes are not real-world scale, and did not fully push the limits of our implementation. To be truly web-scale, our approach would need to work on billions of input documents, which will most likely present several efficiency challenges worth investigating. Both the clustering and induction phases could be improved by developing parallelized versions of both algorithms.

6. REFERENCES

[1] D. Aberdeen, O. Pacovsky, and A. Slater. The learning behind gmail priority inbox. In *NIPS Workshop on Learning on Cores, Clusters and Clouds*, 2010.

[2] N. Ailon, Z. S. Karnin, E. Liberty, and Y. Maarek. Threading machine generated email. In *Proc. of the 6th ACM International Conference on Web Search and Data Mining*, pages 405–414, 2013.

[3] J. Alpert and N. Hajaj. We knew the web was big. *The Official Google Blog*, 21, July 25 2008.

[4] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, and C. D. Spyropoulos. An experimental comparison of naive bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *Proc. of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–167, 2000.

[5] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 337–348, 2003.

[6] R. Bekkerman. Automatic categorization of email into folders: Benchmark experiments on Enron and SRI corpora. *Computer Science Department Faculty Publication Series, University of Massachusetts, Amherst*, (218), 2004.

[7] E. Blanzieri and A. Bryl. A survey of learning-based techniques of email spam filtering. *Artificial Intelligence Review*, 29(1):63–92, 2008.

[8] G. Caruana and M. Li. A survey of emerging approaches to spam filtering. *ACM Computing Surveys*, 44(2):1–27, 2012.

[9] L. A. Dabbish and R. E. Kraut. Email overload at work: An analysis of factors associated with email strain. In *Proc. of the 20th Conference on Computer Supported Cooperative Work*, pages 431–440, 2006.

[10] D. Di Castro, L. Lewin-Eytan, Y. Maarek, R. Wolff, and E. Zohar. Enforcing k-anonymity in web mail auditing. In *Proc. of the 9th International Conference on Web Search and Data Mining*, to appear, 2016.

[11] C. Hachenberg and T. Gottron. Locality sensitive hashing for scalable structural classification and clustering of web documents. In *Proc. of the 22nd ACM International Conference on Information & Knowledge Management*, pages 359–368, 2013.

[12] A. Kannan, K. Kurach, S. Ravi, T. Kaufmann, A. Tomkins, B. Miklos, G. Corrado, L. Lukács, M. Ganea, P. Young, and V. Ramavajjala. Smart reply: Automated response suggestion for email. *CoRR*, abs/1606.04870, 2016.

[13] S. Kiritchenko and S. Matwin. Email classification with co-training. In *Proc. of the Conference of the Center for Advanced Studies on Collaborative Research*, pages 301–312, 2011.

[14] A. Kulkarni and T. Pedersen. Name discrimination and email clustering using unsupervised clustering and labeling of similar contexts. In *Proc. of the 2nd Indian International Conference on Artificial Intelligence*, pages 703–722, 2005.

[15] N. Kushmerick. *Wrapper induction for information extraction*. PhD thesis, University of Washington, 1997.

[16] A. H. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *ACM Sigmod*, 31(2):84–93, 2002.

[17] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 497–506. ACM, 2009.

[18] D. D. Lewis and K. A. Knowles. Threading electronic mail: A preliminary study. *Information Processing & Management*, 33(2):209–217, 1997.

[19] H. Li, D. Shen, B. Zhang, Z. Chen, and Q. Yang. Adding semantics to email clustering. In *Proc. of the 6th International Conference on Data Mining*, pages 938–942, 2006.

[20] G. A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[21] P. Pantel and D. Lin. Spamcop: A spam classification & organization program. In *Proc. of the AAAI Workshop on Learning for Text Categorization*, pages 95–98, 1998.

[22] S. Sarawagi. Automation in information extraction and integration. In *Tutorial of the 28th Internationl Conference on Very Large Databases*, 2002.

[23] Y.-C. Wang, M. Joshi, W. W. Cohen, and C. P. Rosé. Recovering implicit thread structure in newsgroup style conversations. In *Proc. of the 2nd International Conference on Weblogs and Social Media*, pages 152–160, 2008.

[24] P. Weiner. Linear pattern matching algorithms. In *Proc. of the 14th Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[25] J. B. Wendt, M. Bendersky, L. Garcia-Pueyo, V. Josifovski, B. Miklos, I. Krka, A. Saikia, J. Yang, M.-A. Cartright, and S. Ravi. Hierarchical label propagation and discovery for machine generated email. In *Proc. of the 9th International Conference on Web Search and Data Mining*, 2016.

[26] L. Zhang, J. Zhu, and T. Yao. An evaluation of statistical spam filtering techniques. *ACM Transactions on Asian Language Information Processing*, 3(4):243–269, 2004.

[27] W. Zhang, A. Ahmed, J. Yang, V. Josifovski, and A. J. Smola. Annotating needles in the haystack without looking: Product information extraction from emails. In *Proc. of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2257–2266, 2015.