Anatomy of a Privacy-Safe Large-Scale Information Extraction System Over Email

Ying Sheng, Sandeep Tata, James B. Wendt
Jing Xie, Qi Zhao, Marc Najork
{yingsheng,tata,jwendt,lucyxie,zhaqi,najork}@google.com
Google
Mountain View, CA, USA

ABSTRACT

Extracting structured data from emails can enable several assistive experiences, such as reminding the user when a bill payment is due, answering queries about the departure time of a booked flight, or proactively surfacing an emailed discount coupon while the user is at that store.

This paper presents *Juicer*, a system for extracting information from email that is serving over a billion Gmail users daily. We describe how the design of the system was informed by three key principles: scaling to a planet-wide email service, isolating the complexity to provide a simple experience for the developer, and safeguarding the privacy of users (our team and the developers we support are not allowed to view any single email). We describe the design tradeoffs made in building this system, the challenges faced and the approaches used to tackle them. We present case studies of three extraction tasks implemented on this platform—bill reminders, commercial offers, and hotel reservations—to illustrate the effectiveness of the platform despite challenges unique to each task. Finally, we outline several areas of ongoing research in large-scale machine-learned information extraction from email.

CCS CONCEPTS

• **Information systems** → **Email**; *Wrappers (data mining)*;

KEYWORDS

Information extraction, wrapper induction, email, document classification

ACM Reference Format:
Ying Sheng, Sandeep Tata, James B. Wendt, Jing Xie, Qi Zhao, Marc Najork. 2018. Anatomy of a Privacy-Safe Large-Scale Information Extraction System Over Email. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3219901

1 INTRODUCTION

Email is one of the most pervasive forms of online communication today, and remains at the center of online identity. It is estimated that nearly 270 billion emails are received daily, projected to increase to 320 billion by 2021[1]. This enormous volume of email traffic has lead to the notion of *email overload*, in which users become overwhelmed when they can no longer process their inbox at the rate at which emails arrive [10]. Traditional methods to handle this influx include user-driven labeling, foldering, and filtering. These techniques require manual tuning by inbox owners. Automatic techniques have also been developed, including spam filters [8], automatic foldering [21], labeling [44], and prioritization [1].

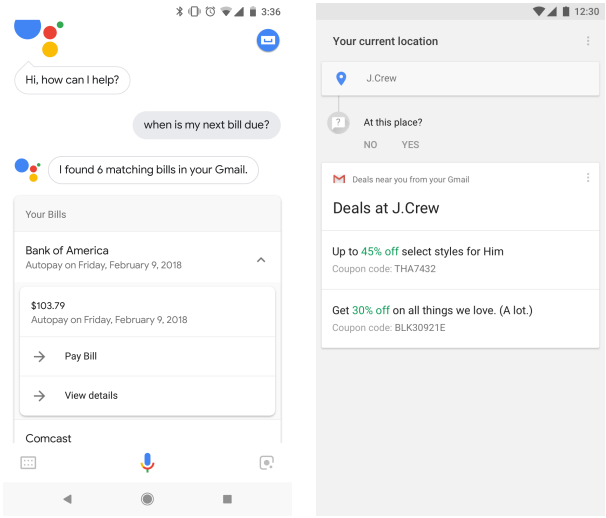

Figure 1: Google Assistant responding to a user query for their recent bills, and proactively displaying deals extracted from Gmail when the user enters the relevant store.

More recently, smart assistants surface the pertinent parts of emails proactively or in response to personal queries. For example, the Google Assistant may trigger a notification with a discount coupon extracted from an email when the user walks into a physical store corresponding to the email's sender. Smart assistants are also able to answer personal queries, such as "when is my electric bill due?". See Figure 1 for some examples of these experiences.

The key enabler for these capabilities is information extraction. Large-scale information extraction of web documents is a well researched topic that spans back to the earliest days of the World Wide Web [23]. Many techniques exist to build large-scale information extraction systems [17, 36]. Unlike the general web, which is predominantly public, emails are private. Thus, to date, relatively

[1] https://www.radicati.com/wp/wp-content/uploads/2017/01/Email-Statistics-Report-2017-2021-Executive-Summary.pdf

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08.
https://doi.org/10.1145/3219819.3219901

little research has been presented on information extraction over this domain [4, 46]. Learning and iterating is a much more difficult task when privacy must be preserved [13].

Upwards of 90% of messages are machine-generated [29]. Virtually all of these business-to-consumer (B2C) emails are created by filling in variable fields of a fixed *template*. For example, shipping confirmation emails from an e-commerce site will contain much of the same boilerplate text and images shared across all instances, while information such as name, address, tracking numbers, and products will be personalized. In this paper, we present Juicer—a machine-learned system for information extraction from B2C emails that is serving over a billion Gmail users daily. Juicer relies on three key design principles:

Scalability Gmail is a planet-scale email system. While web documents can be processed and extracted in batch offline, email is a near-real-time service. Our information extraction system must not only scale to the massive number of emails flowing through Gmail, but also do so at low latency and low cost. A key property of Juicer's scalability is that extraction rules are learned offline and aggregated into the template, thus removing the need for an expensive model inference call online.

Simplicity Juicer is designed so generalist software engineers with a basic understanding of machine learning can implement a new extraction task quickly and effectively. Developers focus on generating training data for two kinds of classifiers. The first is vertical classification, in which we determine which category the email belongs to (e.g. offers, bills, hotel reservations, etc.). The second task consists of learning extractors for the fields that are relevant to that vertical. Developers use off-the-shelf models to train these classifiers, and do not focus their energy on novel machine learning models. Details on the workflow are presented in Section 5.

Safety Most importantly, our system respects user privacy. None of the engineers may view any single email. Evaluation of our system relies on k-anonymity to protect user's privacy [13]. A detailed discussion is presented in Section 4.1.

In Juicer, we first cluster a representative sample of emails according to the templates from which they were instantiated. As part of this process, we also execute a set of classifiers and field extractors over the emails within each cluster, and aggregate the results into a set of static rules used online for extraction. For example, if a majority of the emails in a template cluster are classified as purchase receipts, then we label that template as a purchase template. Likewise, if a majority of emails contain a shared XPath containing the tracking number, then we write a *rule* in the template indicating that an email's matching XPath will contain the tracking number. Thus, online, when an email arrives, the information extraction task is reduced to a simple lookup for the email's matching template, followed by applying any labels and rules that are associated with the template.

Our key contributions are as follows:

- We provide and end-to-end description of a privacy-safe, planet-scale, machine-learned information extraction system over email.
- We describe how to leverage templates to both reduce runtime cost of extraction as well as increase the accuracy of these extractions.
- We discuss several design choices informed by our principles of *scale*, *simplicity*, and *safety* that can be applied to new information extraction systems on other corpora.
- We present three different case studies to highlight the effectiveness of the platform even when each task poses unique challenges.
- Finally, we close with a description of several ongoing areas of investigation in this space that are of interest to researchers and practitioners alike.

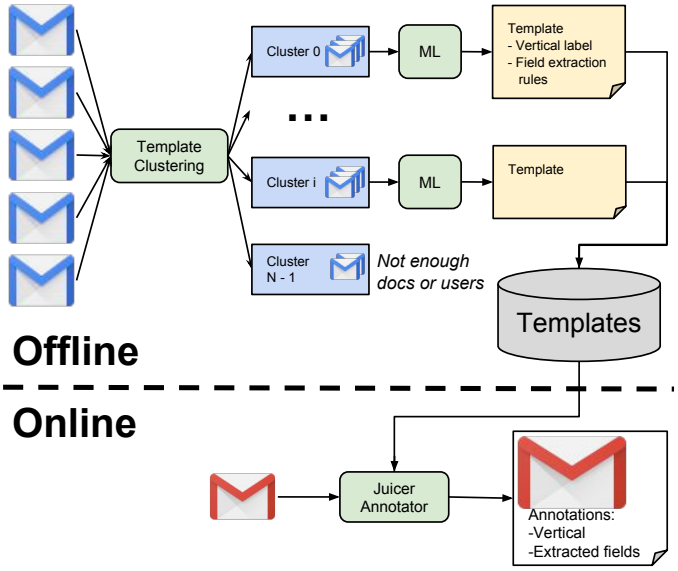

Figure 2: The Juicer architecture. Details of the ML component are described in the text and depicted in Figure 3.

The remainder of the paper is organized as follows. A description of the system architecture is presented in Section 2 followed by a detailed discussion in Section 3 of the models trained to perform the vertical classification and field extraction tasks and how the learned results are attached to the template. Section 4 recounts some of the challenges encountered during development. Section 5 describes the typical workflow for a developer to implement a new or improve an existing extraction task. And Section 6 presents three case studies and the challenges associated with each.

2 SYSTEM ARCHITECTURE

The Juicer system consists of three main components: (1) template induction, in which we cluster emails together to form templates that can be looked up quickly online, (2) machine-learned rule generation, in which we classify each template into a "vertical" (e.g. bills, hotel reservations, flights, etc.) and learn a set of extraction rules specific to each vertical, and (3) an online extraction system that looks up an incoming email's template and executes the rules. A high level depiction of this architecture is presented in Figure 2.

2.1 Template Induction

Juicer induces email templates by sampling recent emails from the Gmail corpus and clustering them. Intuitively, our objective is to cluster sampled emails in a way that groups together all emails instantiated from the same template, and no others. The two failure modes are grouping together emails from multiple templates (*template conflation*), and grouping emails from the same template into multiple clusters (*template fragmentation*). Conflation interferes with the vertical classification and rule generation steps, while fragmentation may cause us to ignore that template due to k-anonymity constraints.

There are several features we could use for mapping an email to a cluster: the email's *content* (that is, its words); its *structure* (that is, its DOM tree, assuming the email is HTML-formatted); or its RFC 5322 *header* information (including sender and subject line). Using shared content for clustering is prone to template fragmentation, particularly for templates that have substantial non-fixed portions. Therefore, we rely on header and structural information. We conducted experiments on a variety of clustering techniques (beyond the scope of this paper) and eventually deployed two techniques into production. The data source for both is a random 0.5% sample of emails from the last 90 days of the Gmail corpus.

The first clustering technique [2] takes advantage of the fact that most instantiations of a single B2C template are delivered from the same email address (e.g. "googlestore-noreply@google.com") with subject lines that can be represented by fixed terms and wildcards (e.g. "Your order of *<wildcard>* has shipped!"). After clustering emails by sender, subject regular expressions are derived by generating a dictionary of fixed terms, which are observed in at least k documents and n unique users within the sender cluster, where k and n are determined by policy. The subject lines are reduced to regular expressions by replacing non-fixed terms with regular expression wildcards, `(.+)`. Subject regular expressions that pass the aforementioned privacy constraints within the cluster are saved as templates for the given sender, and the remaining are discarded.

For online access, these templates are stored in a table keyed by sender, and valued by the list of privacy-passing subject regular expressions along with any data (such as extraction rules) that are associated with those templates. This list is loosely organized from most specific to least specific (from regular expressions that match the fewest emails to ones that match the most).

Juicer's second B2C template representation is derived by clustering emails on their HTML DOM structure. For each email, we emit a set of locality-sensitive hashes—we use Minhash by Broder et al. [7]—computed over the XPaths of all leaf nodes in the DOM tree. Clusters that pass the anonymity thresholds are saved as templates, while the rest are discarded. A similar approach presented by Di Castro et al. [13], called "Mail-Hash", clusters emails based on a single hash over the XPaths. The key difference between these two approaches is that "Mail-Hash" clusters identically structured emails together, while Juicer's Minhash-based implementation clusters together emails with similar but not necessarily identical structures. Note that Di Castro et al. also study Minhash, but apply it over both the markup and text of the email message.

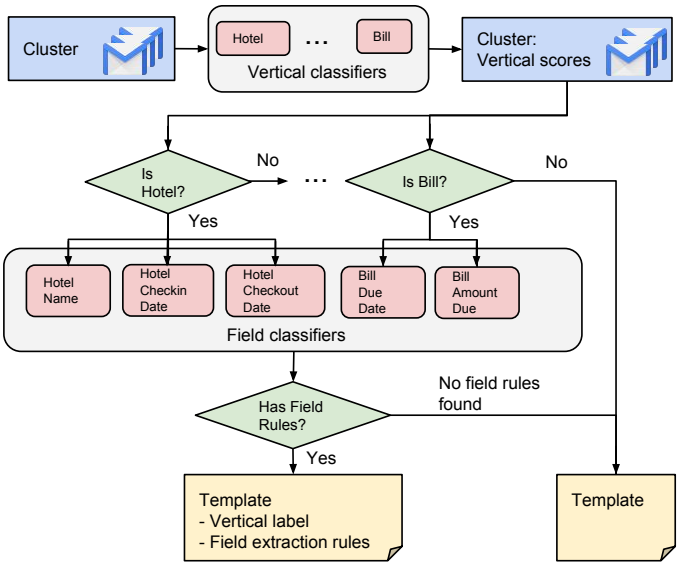

Figure 3: Procedure for vertical classification and rule generation.

Online, templates are keyed by their hash. A set of hashes is produced for each incoming email and looked up; the first hit is declared the matching template.

2.2 Classification and Rule Generation

Having identified templates, we use a family of classifiers to attach vertical labels (e.g. Hotel) and field (e.g. Hotel check-in time) extraction rules to the templates. This procedure is outlined in Figure 3 and described in detail here. More details for Bill, Hotel and Offer verticals are discussed in Section 6.

A *vertical classifier* is a binary classifier which decides if an email belongs to a given vertical. Templates are labeled by aggregating the results of the classifier across the emails in the template. This also helps to de-noise the template-level label.

Each vertical is associated with a set of fields we want to extract. We train a *field classifier* for each of these fields. Once a template has been identified as belonging to a given vertical, we generate candidate spans in the emails which are potential instances of the field we want to extract. This is done for each email in the template and for each field in the vertical.

For instance, to extract the check-in date, we can use a date-annotator to identify all instances of dates in the email. The actual check-in date (obtained from a ground truth extraction) is used as the positive example. All other candidates identified by the date-annotator are treated as negative examples. Note that the candidate generating annotator needs to have high recall. Precision is provided by the classifier we train.

These scores are aggregated at the template level to generate rules in the following way. The field candidates are identified using their HTML XPath in the email. We average the field classification scores for candidates in the same XPath across different emails in the template cluster. We then construct a table with *(XPath, NumInstances, AvgScore)*. This table is used to determine the extraction rules using suitable values for minimum support and precision.

2.3 Online Extraction

Online, when a new email arrives that matches a template associated with a known vertical, the extraction component is triggered. The extraction rules corresponding to each of the fields for the given vertical are triggered in order. A single field may contain more than one extraction XPath—they are triggered starting from the highest scoring XPath to the lowest. Execution for a given field stops as soon as we identify that the contents of an XPath contains exactly one candidate span that can be extracted.

If any of the required fields for a vertical fail to extract, we discard the other field values and generate nothing. This can happen if the template changes over time, and the XPaths we learned no longer match anything in the email. The key advantage of the rule-based extraction system over directly using the classifiers online are improved quality from aggregating the predictions across the samples in a template and the significantly lower cost of applying XPath rules compared to executing a series of sophisticated ML models online.

Feature Name	Description
subject-text	Words in the subject line
sender-text	Tokens in the sender field
top-text	Top 150 words in the body
strong-text	Text marked header, title, bold etc.
alt-text	Alt-text supplied for image content
footer-text	Last 100 words in the body
html-tag-count	Number of HTML tags in the body
text-token-count	Number of text tokens in the body
link-tag-count	Number of link tags
image-tag-count	Number of image tags
script-tag-count	Number of script tags
table-tag-count	Number of table tags
datetime-count	Number of candidate date-time spans
salient-entities	Top entity IDs

Table 1: Features used in the vertical classifiers.

3 LEARNING CLASSIFIERS

This section describes how the vertical and field classifiers are learned, and the tradeoffs considered in making the design choices.

3.1 Training Data

We obtain training data for both vertical and field classifiers by leveraging three major pre-existing sources for extraction:

Microdata Some email-senders include structured microdata using a standard[2] for machine-readable annotations in HTML documents. As expected, this data tends to have very high precision, but extremely low recall.

Manual Parsers Sender-specific parsers are hand-crafted based on several instances of emails donated by users for this purpose. These parsers are typically only available for very popular templates (e.g. hotel reservation confirmations from major chains). These too tend to have very high precision. They are expensive to develop, brittle to changes in the email's format, and offer low recall since they only cover one particular sender.

Generic Parsers This is a class of rule-based extractors developed using more traditional information extraction techniques leveraging donated emails, dictionaries (e.g. a database of all hotel names crawled from the web), and heuristic regular-expression based rules (e.g. look for "check-in" up to n tokens before a date for the check-in date). This class of parsers have lower precision but have significantly higher recall than microdata and manual parsers.

3.2 Vertical Classifiers

We learn a binary classifier for each of the verticals of interest. An email is a positive example for a given class an extraction of that type is produced by any of the three sources described above. The rest of the emails are treated as negatives. Note that there are several techniques [16, 28] that are aimed at learning classifiers from positive and unlabeled data that we don't discuss at length here.

When generating training examples, instead of randomly sampling emails across all users, we sample a fixed number of emails from each template. This ensures that the training data is not dominated by examples from popular templates whose email count can exceed that of tail domains by several orders of magnitude. (A large online retailer may send out millions of times more purchase receipts than a small local retailer.) As a natural consequence of this approach, personal email that are not generated from an underlying template are automatically excluded from training data. For some verticals, only a very small fraction of the emails are positive examples. In order to effectively deal with class skew, we downsample the examples in the negative class while training, but compute our final precision and recall metrics on the full template-level data.

Each example is represented using simple bag-of-words, count, and bag-of- entities [39] features. Table 1 summarizes the key features that we currently use. Only those tokens that pass the necessary k-anonymity thresholds are used in constructing these features. The rest are replaced by a special "REDACTED" marker.

The embeddings for each of these text features are concatenated together with the count features and fed to a deep network with a stack of ReLU layers. We minimize the log-loss using Proximal Adagrad [15]. The hyperparameters are picked using the Gaussian Process Bandits algorithm in Vizier [20]. Figure 4 shows the objective values (precision-recall AUC) for trials of different hyperparameters. Clearly some hyperparameters lead to poor quality classifiers. We picked the best sets of hyperparameters for our model training. We currently do not use any convolutional layers or an attention mechanism. We are currently investigating if these mechanisms improve our classifiers.

3.3 Field Classifiers

For each field of a given vertical, a library of existing annotators (mentioned above) is used to identify candidate text spans that might be potentially valid values for this field. Then, a binary classifier is trained to identify the most likely candidate.

[2] https://www.w3.org/TR/microdata/

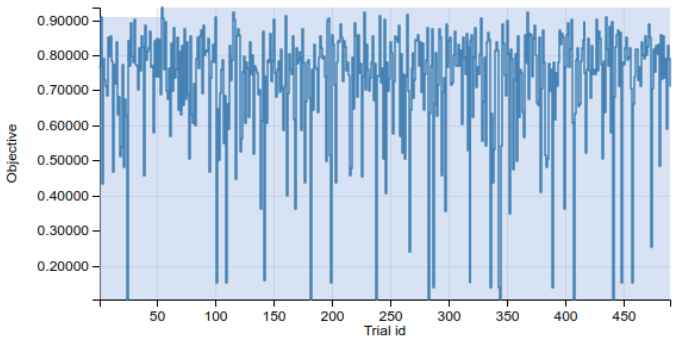

Figure 4: Hyperparameter tuning with Vizier.

Feature Name	Description
{5/10/20}-w-before	5/10/20 words before the candidate span
{5/10/20}-w-after	5/10/20 words after the candidate span
field-text	Contents of the candidate span
doc-index	Position of the field in the document (0-1)
candidate-index	Positional rank relative to all candidates

Table 2: Features used in the field classifiers.

Note that only the emails that actually contain an extraction of the class under consideration contribute (positive and negative) training examples to the field classifier. The top features are described in Table 2. Similar to the features used for vertical classifiers, only the tokens that pass privacy thresholds are included.

3.4 Design Choices

An obvious first question is why we train a binary classifier for each vertical instead of a multi-class (or multi-label) classifier? The latter approach has two major advantages. First, the multi-class approach may offer better precision by leveraging labeled data from multiple verticals. Second, it may be easier to maintain and improve a single multi-class classifier compared to several binary classifiers. However, choosing multiple binary classifiers allows us clear separation of concerns between verticals and makes it possible to develop and improve verticals in parallel. This also allows us to choose different tradeoffs in precision and recall for different verticals. Taking on the cost of maintaining separate binary classifiers actually results in better experiment velocity without having to debug problems like a new multi-label classifier improving metrics for one vertical, but doing worse for a different vertical.

Second, we chose to build an email-level classifier, followed by template-level aggregation of the scores instead of directly building a template classifier. Building template-level labels requires us to deal with the case where an existing extraction approach (say generic parsers) only extracts from a fraction of the emails (say 30%) in the template. This could either be because the template is conflated (there are two different underlying templates that are getting clustered together, and only one of them belongs to the vertical we're trying to extract), or because the generic parser was brittle, and only matched some of the emails in the template. By building email-level classifiers, we are able to better distinguish between these cases. This approach also lets us leverage existing Google infrastructure for email classification used for applications like spam detection and smart folders [21].

Third, we use deep networks instead of simpler model classes like logistic regression or decision trees that are sometimes considered more "debuggable" to learn these classifiers. This is in contrast to the choice made in systems like DeepDive [36] which use carefully engineered human-understandable features instead of the opaque features yielded by deep learning. An *explicit goal* in building an extraction system over private data is that an engineer must be able to improve the system without having access to user data and developing the intuition for complex human-understandable features. As a result, we rely on simple features like bag-of-words representations of the document and leverage large training data sets where deep learning has shown great results. Using simple features like embedding bag-of-words features allows us to reuse them for multiple vertical and field classifiers. This also opens the door to transfer learning techniques by using unsupervised embeddings such as Word2Vec [31].

4 CHALLENGES

The key challenges encountered in training classifiers as well as incrementally improving the extraction system fall into two broad categories: protecting user privacy, and dealing with data quality.

4.1 Protecting User Privacy

Unlike public web pages, email documents are private. Google cares deeply about protecting users' privacy. Our team and the engineers we support are not allowed to view any single email. Yet, evaluating performance of our classifiers on unlabeled data requires some form of human assessment.

For sanity-checking, we use a dataset consisting of emails explicitly donated by users and compute precision and recall against manually labeled instances in this dataset. As one would expect, the donated emails are a very small collection, and do not represent the distribution over which the extraction system will be run.

A second approach is to rely on a larger pool of users who have opted into a rating program for occasionally answering questions about their own emails. In order to evaluate a new set of models in the extraction system, we check to see if there are any extractions corresponding to the users in the rating program, and ask them to rate the quality of our extractions. The aggregated results are used to calculate the precision. This approach, while allowing us access to more recent emails than the donated corpus, suffers from the same drawbacks of providing a non-representative distribution and long delays. The rating program includes only a few thousand people, and requires that the participants go through training. It is also not available in all the countries and languages in which Gmail is available.

In order to conduct more informative evaluations, we built an anonymized review system where we leverage the templates to generate a synthetic email with any potential personal information redacted for each email that we want to evaluate. Since template induction follows k-anonymity [13], the synthetic email does not reveal any personal information. During the template induction process, we identify the boilerplate portions of the email common across most samples, and mark them as *fixed text* (e.g. "Thank you

for your order."). The rest of the email is marked as *transient text*. When generating a synthetic email for human review, only the *fixed text* are kept and the *transient text* are replaced by suitable fixed values for dates, numbers, names, addresses, or just obfuscated text ("XXXXX"). Given the aggregated nature of the *fixed text*, the reviewer will not be exposed to any personal information, but the fixed text usually allows them to evaluate the extraction performance. We ask two classes of questions—one to determine if we identified the category of the email correctly ("Is this email a hotel reservation confirmation?"), and another to determine if we correctly extracted each field of interest ("Is this highlight the check-in date").

A major problem is that the anonymization may hide several key parts of the email, making it too difficult to answer a question. This results in the samples being marked 'N/A' and we discard them from our evaluation. Even for the samples where the human evaluator determines that the classifier was wrong, anonymization often makes it difficult to understand what new features might be useful to improve the classifier.

The human-assessment results reported in the next section are based on this approach, which we have found to be the most effective so far. Good evaluation results here often correlate with application level metrics such as error reports where the extractions are used.

4.2 Data Quality

A key observation about the training data is that the existing extractions are often *not* representative of a random subset of the overall dataset (templates) we want to classify. Microdata is biased towards sophisticated senders, and manual and generic parsers are biased towards popular senders for which our users have donated emails. As a result, our models are trained on data from popular and/or sophisticated senders, but need to be applied to the long tail of smaller senders that in aggregate affect a large volume of email. This manifests itself as a much lower validation precision on new templates classified as belonging to the vertical compared to the precision on a holdout set. For example, a classifier we trained to detect commercial offers (Section 6.2) had a holdout precision of 0.993, while assessments on new templates put the precision at 0.85.

Techniques used to deal with this challenge include a mix of stratified sampling by templates (to avoid over-representation for the large senders), regularization to avoid memorizing features tightly linked to existing senders, and active learning (to obtain more representative training/test data over time). Space constraints prevent us from describing the details and impact of these techniques in greater detail.

A second challenge is that the manual parsers and generic parsers are used to varying degrees in different verticals, and have different levels of accuracy. An error analysis on the output of these parsers suggests that they do suffer from false positive errors and, of course, false negatives. For instance, they may erroneously parse and extract a bill reminder from an email that doesn't actually contain one. This may be because a new template from the same sender is based on a previous template, and accidentally matches the rules in a manual or generic parser.

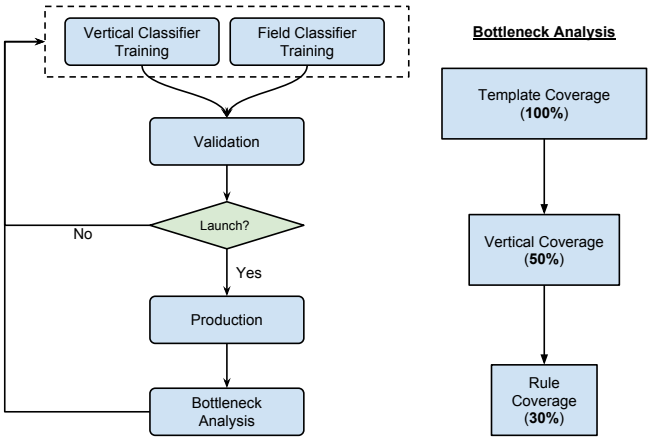

Figure 5: Workflow to develop an extraction vertical in Juicer. The left outlines steps to train and improve classifiers. The right depicts the relationship between template, vertical, and rule coverage to determine where to concentrate the developer's efforts.

5 DEVELOPER WORKFLOW

This section outlines the typical developer's workflow to implement and improve an extraction task in Juicer. As illustrated in Figure 5, this involves four major steps: vertical classifier training, field classifier training, model assessment, and extraction bottleneck analysis.

Vertical Classifier Training. The first step is to train a binary classifier which determines if an email belongs to the vertical of interest. The developer's key task is to write the code that determines the label (positive or negative) for each email that is sampled from the template. This is usually done by simply examining if there is an extraction from one of the existing approaches. The model described in Section 3 is used for training. No feature-generation or machine-learning code is required.

Field Classifier Training. Field classifiers are trained similar to the vertical classifiers. The key task here is to implement a candidate generator that can produce positive and negative examples for a given email. Writing a good candidate generator is vital since it informs the maximum recall for the final extraction system. Over the years, we have developed a library of candidate generators for standard types like dates, currency amounts, addresses, etc. Some fields, however, can pose challenges. Examples like *hotel name*, *product name* or *event description* don't lend themselves to convenient regular expressions, and require a non-trivial amount of developer time.

Note that the vertical classifier and field classifier(s) can be updated separately. In other words, developers can update the vertical classifier while keeping field classifiers unchanged, and vice versa.

Validation. In this step, the developer generates sample extractions on synthetic anonymized emails (as described in Section 4.1) on a hold-out set and sends them for assessment. If the resulting precision of these manually assessed extractions meets the vertical-specific thresholds, the models for the vertical and field classifiers are pushed to production.

Statistic	Bills	Offers	Hotels
AUC ROC	0.9727	0.9995	0.9999
AUC Precision Recall	0.8010	0.9998	0.8837

Table 3: AUC-ROC and AUC-PR on a 10% holdout when training the vertical classifier.

Bottleneck Analysis. Most extraction tasks in Juicer require high precision, so the improvement steps usually entail increasing coverage of extractions while maintaining high precision. In order to increase extraction coverage, the developer first identifies the current bottleneck to determine where to focus her efforts. She runs a report that computes several key statistics on a sample of emails. *Template coverage* is the fraction of emails with existing extractions which matched a known Juicer template. In the example in Figure 5, this is 100%. If this value is low, the developer in charge of implementing the extraction task needs to bring this to our attention (the team that owns the Juicer platform) to see if additional templates can be induced to cover the emails that are not matching known templates. For instance, this may happen for relatively new templates for which the batch template induction job has not yet seen enough examples.

Vertical coverage is the fraction of emails with existing extractions that match a template that was classified as belonging to this vertical (50% in the example in Figure 5). If this number is low, the developer should focus her efforts on improving the recall of the vertical classifier. *Rule coverage* imposes the additional constraint that we also have identified extraction rules for the template. (30% in the example in Figure 5.) Rule coverage can be further refined to understand which of the fields in the vertical are difficult to learn a rule for, so the developer's attention can be focused there. Finally, *extraction coverage* is the fraction of extractions recovered by Juicer (not shown in Figure 5). It may be lower than rule coverage when online extraction fails. Section 2.3 describes a few scenarios under which online extraction may fail. This tends to be difficult to address, but a detailed discussion is beyond the scope of this paper.

To improve any of the classifiers, the developer has three choices: write a better candidate generator, gather additional training data, or write additional features to feed into the model. None of these steps require deep machine learning expertise. Additional training data can be gathered by writing more manual parsers or simply using the extractions from the templates that Juicer previously identified.

6 CASE STUDIES

This section describes the details of three extraction tasks that have been implemented using the Juicer infrastructure: bill payment reminders, commercial offers, and hotel reservations. The first two have been in production for over a year, serving over a billion users of Gmail. The third task is under active development. These tasks have been chosen to illustrate that while each one poses unique and interesting challenges, the underlying infrastructure is sufficiently general that it can be used to solve all of them.

Model	Structural Templates		Sender-Subject Templates	
	Existing	New	Existing	New
Bills	84.6	66.8	94.1	76.2
Offers	100.0	87.8	100.0	87.4
Hotels	97.7	98.7	100.0	78.0

Table 4: Precision on a sample of templates classified positive both for those templates that correspond to existing parsers and newly identified templates that do not correspond to existing parsers.

Description	Bills	Offers	Hotels
Emails matching a positive template	73.3	70.0	75.9
Emails matching templates with rules	58.7	40.0	38.2
New templates discovered as a percentage of pre-existing templates	79.0	8.0	57.6

Table 5: Bottleneck analysis for each of the verticals identifying lost email-level recall at each stage. The baseline is the set of emails with extractions from one of the existing approaches.

6.1 Bills

The first task we describe is one of extracting bill reminders with two key fields: the amount due and the due date. These extractions are key to answering questions like "When is my electric bill due?" and even offering proactive reminders.

For this task, we built two vertical classifiers: one to determine that the template is a bill email, and a second to determine the subtype of the bill (payment confirmation, autopay notification, reminder, and overdue). We focus on reminders and overdue notifications in this task. The training data for the bill classifier is highly skewed. It consists of ~1 million positive and ~200 million negative training examples. The neural network selected by hyperparameter optimization is a 6 layer network with 20 ReLU neurons per layer. The embedding dimension is set to 50. We use the Adagrad algorithm[14] for optimizing with an initial learning rate of 0.05.

Table 3 presents the area under the ROC and PR curves for each of the vertical classifiers. While the AUC-ROC value exceeding 0.97 for the bill classifier might suggest that this is an easy problem; the extreme imbalance between the positive and negative classes combined with the high precision requirement makes this a challenging task.

Table 4 reports the precision of the vertical classifiers on a sample of the templates predicted to be positive (using thresholds corresponding to 90% precision on the holdout set). We stratify the results for templates known to correspond to existing parsers versus newly identified parsers and for the two clustering strategies (sender-subject and structural). These were assessed using the techniques described in Section 4.1. As the table shows, we are only able to obtain a precision of 66.8% over unseen templates for the bill classifier. Training a high-quality vertical classifier for Bills is still a fairly challenging task, and our results suggest that either the current model does not generalize well or that tail templates are significantly different from the big senders that comprise our

training data. An example of a false positive error is where the classifier identified a statement of an account balance as a bill reminder. The types of the fields (a date and a currency amount) as well as the high-level language in such an email is fairly similar to what one expects in a bill reminder, and therefore this presents a difficult challenge. Improving this model is ongoing work.

Table 5 presents a high-level breakdown of where extraction recall is lost. Recall that no extraction is performed for emails matching a template if we are unable to induce a rule for one of the fields. This snapshot informs the developer where to invest her time to improve the coverage and quality of extractions. For Bills, this table shows that the vertical and subtype classifiers are only able to identify 73.3% of the existing emails as correctly belonging to the bill reminder class. Of these templates, we are only able to induce extraction rules for a subset which corresponds to about 58.7% of the existing emails. Further analysis (not presented in Table 5) showed that the due date field classifier adds extraction rules for 93.5% of the bill templates while the amount due field classifier only adds rules to 81.8% of the templates. Thus, the latter field is another bottleneck for bill extractions.

Table 5 shows that Juicer is able to identify 79% more templates compared to the existing parsers. Even though the machine-learned approach does not yet recover 100% of the extractions from the existing parsers, we are able to expand the recall to new templates and increase the overall coverage of extractions. Accurately estimating Juicer's true recall over the emails not covered by existing parsers is very challenging — we do not tackle that in this paper. Table 5 indicates that improving the vertical classifier is likely to yield the broadest benefits.

6.2 Commercial Offers

Offer emails are those that provide an explicit discount or other savings for the recipient. We currently extract two fields: the expiration date and a coupon code for these emails. These extractions can be used to power experiences such as proactively letting the user know about a discount if she is at a store for which she has received an email offer even if she hasn't yet read the email.

Constructing the training data set for the offer class poses a unique problem. Since a large fraction of email consists of commercial promotions, many of the emails that don't have an offer extraction from an existing parser may actually be offer emails. Treating all the unlabeled examples as negatives does not work well in practice, resulting in much worse precision scores from human assessments. As a result, we exclude emails tagged as promotional (a pre-existing classifier in Gmail) from the negative examples. The resulting training dataset comprises about 2.4 million positives and 21.6 million negatives.

Table 4 shows that the precision of the offer classifier on templates with existing parsers is 100%, and is substantially lower on new templates without existing parsers. This suggests that the offer classification task is particularly challenging for generalizing well to unseen templates.

The bottleneck analysis in Table 5 shows that coverage decreases significantly at the *rule induction* stage. While 70% of emails with extractions match an offer template, only 40% match an offer template with extraction rules for the fields of interest. We discovered that commercial offers tend to have more variability within a cluster, and learning a fixed extraction XPath may be too restrictive for these types of templates. We are exploring ways to generalize the XPath to increase the coverage of the extraction rules learned.

There are two challenges unique to the offer vertical. First, templates churn frequently. By the time we discover a template, learn extraction rules, and are ready to deploy the learned rules to production, marketers have moved on to using new or modified templates. Addressing this challenge is ongoing work. Second, offer information is often encoded in images instead of text. In a previous paper [34] we showed that the offer classifier can be improved by leveraging the text extracted from the images.

6.3 Hotel Reservations

The third task we describe here is to extract hotel reservation confirmations. Extracted reservations are used in a variety of applications like Google Now, automatic updates to the calendar, and personal queries to the assistant (e.g. "What is the address of my hotel in Beijing?"). This is the newest extraction task, and is not yet in production. We extract the hotel name, address, check-in, and check-out dates.

The training data for vertical classification is, again, highly skewed. The fraction of positive to negative examples is fewer than 1 in 1000. The vertical classifier was trained with 190M examples using 90% of the templates. The negative examples were downsampled to 1% of all negatives in the training set. The test set was constructed using 10% of the templates.

Human assessments of the classifier's predictions show high precision in Table 4. Table 5 shows that this model only recovers templates corresponding to 75.9% of the existing extractions. However, the model also recovers new templates corresponding to 57.6% more templates than the existing set, helping expand extraction coverage. The templates where rules are induced for all four fields correspond to only 38.2% of existing extractions. Additional analysis revealed that over 40% of the templates are missing the rule for the hotel's name and address fields. We are currently improving the candidate generators and classifiers for these fields.

Examining the false positives from the vertical model at lower thresholds shows that promotional emails from hotels pose one of the biggest challenges. For instance, consider an email stating "Enjoy this weekend in Cancun! Special promotional rates at the Grand Hotel only for this weekend. Check in on Friday the 12th and check out on Monday the 15th." This contains a hotel name, usually a corresponding hotel address, and finally, dates that look like check-in and check-out dates. The features that distinguish this email from an actual confirmation require semantic understanding. Improving this classifier's performance is ongoing work.

7 RELATED WORK

Template induction is the technique of generating a skeleton of repeated content based on previously seen examples. It has been widely used in information extraction over structured web pages [3, 23]. For emails, multiple algorithms for template induction have been described [2, 4] along with applications like email threading [2] and hierarchical classification [44]. A technique has also

been suggested for plain text emails [35] where data is not explicitly structured.

Traditional information extraction techniques focused on supervised methods such as hidden Markov Models [19, 38], conditional random fields [37], or rule learning [40], etc. While these methods work well on small homogeneous corpora, they are not scalable. Later systems like KnowItAll [17], TextRunner [6], and OpenIE [18] allowed information extraction approaches to scale to the diversity and size of the web. These systems mostly focus on extracting large collections of facts from the web, and are not specifically designed for email. Extraction from email presents unique challenges of privacy and scale and has not been addressed much in the literature.

Recent systems like DeepDive [11] focus on leveraging statistical inference techniques paired with an iterative developer workflow for incrementally constructing knowledge bases. Juicer differs from systems like DeepDive in that our design is specifically targeted at email, and leverages template induction along with recent breakthroughs in deep learning in contrast to a custom engine for statistical inference.

A system for machine-generated email extraction for the Yahoo mail backend was recently described in [12]. This system uses the "Mail-Hash" clustering technique [13], and a similar approach to learning extraction rules. The experiments in the paper [12] focus on the travel vertical, but, much like Juicer, their techniques are applicable to other verticals as well. While there are many similarities, a key difference is that in Juicer, we rely on a deep-learning model with simple bag-of-words features for all our verticals in contrast to the hand-crafted "light-annotations" that can be tailored to each vertical as in [12]. An explicit design goal in Juicer is to *not* require per-vertical feature-engineering. Another interesting difference is that [12] creates tens of thousands clusters for each sender domain, while Juicer only creates tens of clusters for each domain. This suggests that the Yahoo system may suffer from template fragmentation which when combined with k-anonymity leads to loss in recall.

We build on several techniques from the machine learning literature such as training from only positive and unlabeled data without explicit negatives [24, 26–28], using pre-trained vector embeddings of words [31], and combining multiple noisy sources of training data [9, 22].

A recent technique for generalizing XPaths using decision trees to "forgiving XPaths" has been shown to improve extraction accuracy [32]. We expect this technique to be broadly applicable to email in addition to web pages where it has been previously used. Recent results in document classification [45] show that using a hierarchical attention mechanism can improve accuracy on a variety of text classification tasks. While email classification differs from classifying short text documents, we expect that the techniques may be adapted to email content with HTML.

Gathering training data is often a major bottleneck in machine learning. Weak supervision methods are showing success in overcoming this issue. Weak supervision refers to a broad class of methods that can be used to programmatically create training sets, such as using heuristics rules or knowledge-bases [22]. While these weak supervision methods provide larger scale data set, they are also noisier and with lower quality. Recent techniques leveraging generative models [5, 36, 43] to combine multiple sources of weak supervision have shown promise on several information-extraction tasks. Much like all practical data sets, our training data also contains noisy supervision, but with a small number of sources. Evaluating if additional weak supervision is likely to improve the quality of our models is an interesting avenue for future work.

Privacy is another major challenge when dealing with emails. Juicer's template induction follows k-anonymity [41] for privacy protection. Other methods for privacy protection like l-diversity [30] and t-closeness [25], overcome the limitations of k-anonymity and provide stronger privacy guarantees. Juicer mainly focuses on anonymizing emails to generate synthetic redacted emails for internal human assessment. Since redacted emails are not published, we can be sure that they will not be joined with other datasets which might weaken the privacy expectations from k-anonymity [33, 42]. Consequently, we limit our approach to k-anonymity.

8 SUMMARY AND FUTURE WORK

In this paper, we presented a broad overview of a scalable, privacy-safe email extraction system. We described how templates are used to improve extraction accuracy. Our design streamlines the iterative workflow for improving extraction coverage to enable developers without deep expertise in machine learning to be productive. The results from three extraction tasks show the machine-learned approach is able to extract from many new templates which were not detected by previous approaches. While the machine-learned extractions do not yet recover all the extractions from existing systems, they provide a strong complement and a simpler, more modular approach for continuing improvements.

For researchers in this area, this opens up several interesting questions. What's the right trade-off between model complexity and robustness so that non-experts can continually improve the quality of the classifiers? In particular, what model and document representation requires the least amount of feature-engineering, which can be particularly challenging in the context of private email. We are also investigating general approaches to incorporating additional signals in email that are typically not available in plain text such as markup tags and knowledge of template-structure (fixed vs. transient text).

We continue to investigate approaches to better generalize to the long tail of email-templates given that most of the training data is for the head templates. Finally, we are exploring the use of transfer learning as well as recent advances in machine translation to broaden Juicer's reach to emails in languages with limited training data.

ACKNOWLEDGMENTS

We would like to acknowledge Andrei Broder, Vanja Josifovski, and Lluis Garcia Pueyo for contributions to earlier versions of the system described in this paper. Several engineers including Mike Bendersky, Marc Cartright, Amitabh Saikia, and Jie Yang contributed to the infrastructure that made this work possible. We are also grateful to engineers on products including Gmail, Google Now, Google Assistant, and multiple infrastructure teams for inputs that influenced the design of the system.

REFERENCES

[1] Douglas Aberdeen, Ondrej Pacovsky, and Andrew Slater. 2010. The learning behind Gmail Priority Inbox. In *NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds (LCCC)*.

[2] Nir Ailon, Zohar S Karnin, Edo Liberty, and Yoelle Maarek. 2013. Threading machine generated email. In *WSDM*. 405–414.

[3] Arvind Arasu and Hector Garcia-Molina. 2003. Extracting structured data from web pages. In *SIGMOD*. 337–348.

[4] Noa Avigdor-Elgrabli, Mark Cwalinski, Dotan Di Castro, Iftah Gamzu, Irena Grabovitch-Zuyev, Liane Lewin-Eytan, and Yoelle Maarek. 2016. Structural Clustering of Machine-Generated Mail. In *CIKM*. 217–226.

[5] Stephen H. Bach, Bryan He, Alexander Ratner, and Christopher Ré. 2017. Learning the structure of generative models without labeled data. (2017). arXiv:1703.00854

[6] Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. 2007. Open Information Extraction from the Web. In *IJCAI*. 2670–2676.

[7] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. 1997. Syntactic clustering of the web. *Computer Networks and ISDN Systems* 29, 8-13 (1997), 1157–1166.

[8] Godwin Caruana and Maozhen Li. 2012. A survey of emerging approaches to spam filtering. *Comput. Surveys* 44, 2 (2012).

[9] Koby Crammer, Michael Kearns, and Jennifer Wortman. 2008. Learning from multiple sources. *Journal of Machine Learning Research* 9 (2008), 1757–1774.

[10] Laura A Dabbish and Robert E Kraut. 2006. Email overload at work: an analysis of factors associated with email strain. In *CSCW*. 431–440.

[11] Christopher De Sa, Alex Ratner, Christopher Ré, Jaeho Shin, Feiran Wang, Sen Wu, and Ce Zhang. 2017. Incremental knowledge base construction using DeepDive. *The VLDB Journal* (2017), 1310–1321.

[12] Dotan Di Castro, Iftah Gamzu, Irena Grabovitch-Zuyev, Liane Lewin-Eytan, Abhinav Pundir, Nil Ratan Sahoo, and Michael Viderman. 2018. Automated Extractions for Machine Generated Mail. In *Companion Proceedings of The Web Conference*. 655–662.

[13] Dotan Di Castro, Liane Lewin-Eytan, Yoelle Maarek, Ran Wolff, and Eyal Zohar. 2016. Enforcing k-anonymity in web mail auditing. In *WSDM*. 327–336.

[14] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12 (2011), 2121–2159.

[15] John Duchi and Yoram Singer. 2009. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research* 10 (2009), 2899–2934.

[16] Charles Elkan and Keith Noto. 2008. Learning classifiers from only positive and unlabeled data. In *KDD*. 213–220.

[17] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. 2004. Web-scale information extraction in KnowItAll (preliminary results). In *WWW*. 100–110.

[18] Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam. 2011. Open Information Extraction: The Second Generation. In *IJCAI*. 3–10.

[19] Dayne Freitag and Andrew McCallum. 1999. Information extraction with HMMs and shrinkage. In *Proc. of the AAAI-99 Workshop on Machine Learning for Information Extraction*. 31–36.

[20] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. 2017. Google Vizier: A service for black-box optimization. In *KDD*. 1487–1495.

[21] Mihajlo Grbovic, Guy Halawi, Zohar Shay Karnin, and Yoelle Maarek. 2014. How Many Folders Do You Really Need?: Classifying Email into a Handful of Categories. In *CIKM*. 869–878.

[22] Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S Weld. 2011. Knowledge-based weak supervision for information extraction of overlapping relations. In *ACL*. 541–550.

[23] Nicholas Kushmerick, Daniel S Weld, and Robert Doorenbos. 1997. Wrapper induction for information extraction. In *IJCAI*. 729–737.

[24] Wee Sun Lee and Bing Liu. 2003. Learning with positive and unlabeled examples using weighted logistic regression. In *ICML*. 448–455.

[25] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. 2007. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*. 106–115.

[26] Xiao-Li Li and Bing Liu. 2005. Learning from positive and unlabeled examples with different data distributions. *ECML* (2005), 218–229.

[27] Xiao-Li Li, Bing Liu, and See-Kiong Ng. 2010. Negative training data can be harmful to text classification. In *EMNLP*. 218–228.

[28] Bing Liu, Yang Dai, Xiaoli Li, Wee Sun Lee, and Philip S. Yu. 2003. Building text classifiers using positive and unlabeled examples. In *ICDM*. 179–186.

[29] Yoelle Maarek. 2016. Is Mail The Next Frontier In Search And Data Mining?. In *WSDM*. 203–203.

[30] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. 2006. l-diversity: Privacy beyond k-anonymity. In *ICDE*. 24–24.

[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. (2013). arXiv:1301.3781

[32] Adi Omari, Sharon Shoham, and Eran Yahav. 2017. Synthesis of Forgiving Data Extractors. In *WSDM*. 385–394.

[33] Roberto Pellungrini, Luca Pappalardo, Francesca Pratesi, and Anna Monreale. 2017. Fast estimation of privacy risk in human mobility data. In *International Conference on Computer Safety, Reliability, and Security*. Springer, 415–426.

[34] Navneet Potti, James B. Wendt, Qi Zhao, Sandeep Tata, and Marc Najork. 2018. Hidden in Plain Sight: Classifying Emails Using Embedded Image Contents. In *WWW*. 1865–1874.

[35] Julia Proskurnia, Marc-Allen Cartright, Lluís Garcia-Pueyo, Ivo Krka, James B Wendt, Tobias Kaufmann, and Balint Miklos. 2017. Template Induction over Unstructured Email Corpora. In *WWW*. 1521–1530.

[36] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. In *NIPS*. 3567–3575.

[37] Sunita Sarawagi and William W Cohen. 2005. Semi-markov conditional random fields for information extraction. In *NIPS*. 1185–1192.

[38] Kristie Seymore, Andrew McCallum, and Roni Rosenfeld. 1999. Learning hidden Markov model structure for information extraction. In *AAAI-99 Workshop on Machine Learning for Information Extraction*. 37–42.

[39] Amit Singhal. 2012. Introducing the knowledge graph: things, not strings. *Official Boogle Blog* (2012).

[40] Stephen Soderland. 1999. Learning information extraction rules for semi-structured and free text. *Machine Learning* 34, 1 (1999), 233–272.

[41] Latanya Sweeney. 2002. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 05 (2002), 557–570.

[42] Vicenç Torra. 2017. Privacy Models and Disclosure Risk Measures. In *Data Privacy: Foundations, New Developments and the Big Data Challenge*. Springer, 111–189.

[43] Paroma Varma, Bryan D He, Payal Bajaj, Nishith Khandwala, Imon Banerjee, Daniel Rubin, and Christopher Ré. 2017. Inferring Generative Model Structure with Static Analysis. In *NIPS*. 239–249.

[44] James B Wendt, Michael Bendersky, Lluis Garcia-Pueyo, Vanja Josifovski, Balint Miklos, Ivo Krka, Amitabh Saikia, Jie Yang, Marc-Allen Cartright, and Sujith Ravi. 2016. Hierarchical label propagation and discovery for machine generated email. In *WSDM*. 317–326.

[45] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *ACL*. 1480–1489.

[46] Weinan Zhang, Amr Ahmed, Jie Yang, Vanja Josifovski, and Alex J. Smola. 2015. Annotating Needles in the Haystack Without Looking: Product Information Extraction from Emails. In *KDD*. 2257–2266.