Migrating a Privacy-Safe Information Extraction System to a Software 2.0 Design

Ying Sheng
Google
Mountain View, CA, USA
yingsheng@google.com

Nguyen Vo
Google
Mountain View, CA, USA
nguyenvo@google.com

James B. Wendt
Google
Mountain View, CA, USA
jwendt@google.com

Sandeep Tata
Google
Mountain View, CA, USA
tata@google.com

Marc Najork
Google
Mountain View, CA, USA
najork@google.com

ABSTRACT

This paper presents a case study of migrating a privacy-safe information extraction system in production for Gmail from a traditional rule-based architecture to a machine-learned Software 2.0 architecture. The key idea is to use the extractions from the existing rule-based system as training data to learn models that in turn replace all the machinery for the rule-based system. The resulting system a) delivers better precision and recall, b) is significantly smaller in terms of lines of code, c) is easier to maintain and improve, and d) allowed us to leverage machine learning advances to build a cross-language extraction system even though our original training data was only in English. We describe challenges encountered during this migration around generation and management of training data, evaluation of models, and report on many traditional "Software 1.0" components we built to address them.

1. INTRODUCTION

Advances in machine learning (ML) over the last decade have opened up a radically new approach to building software systems. Dubbed "Software 2.0" [14], this approach focuses on training models to learn from data instead of explicitly writing code for the required behavior. Applications like language translation, speech recognition, and image recognition have always made heavy use of ML techniques. However, the current state of the art *replaces* these complex systems that use several traditional ML algorithms with a trained neural network [7]. More interestingly, systems that have *not* made heavy use of ML in the past like database indexes [15] and data cleaning systems [25] are now being re-designed as learned Software 2.0 systems.

In this paper, we present a case study of migrating a privacy-safe information extraction system over email serving over a billion users worldwide from a traditional rule-based architecture to a machine-learned Software 2.0 architecture. This extraction system [27] is responsible for extracting structured objects from emails in different *verticals* – e.g. bill reminders, shipping confirmations for online purchases, hotel reservations, etc. Each vertical defines a set of fields to extract. In the case of hotel confirmations, this includes the check-in and check-out dates as well as the hotel name and address. These extractions enable experiences like allowing a smart assistant to answer personal queries such as "when is my electric bill due?" and "where is my Macy's package?".

A legacy rule-based extraction system using hand-crafted rules has been in production since 2013. We leveraged extractions from the existing rule-based system as well as previously hand-written parsers as the source for training data. The biggest challenges we encountered during this migration were in generating and managing training data. Training data for *field-classifiers* (described in Section 2.2) requires writing a high-recall candidate generator and a high-quality labeler. Detecting low quality training data from low-precision rules and improving data quality over time is critical. Developing these in the context of a privacy-safe system where no one can visually inspect the underlying data is particularly challenging.

We tackled these problems by building several traditional "Software 1.0" components. We designed two abstractions called *CandidateGenerators* and *CandidateLabelers* that allowed us to generate training data for all fields across different verticals. We also built tools to evaluate CandidateGenerators and CandidateLabelers on synthetic emails that are generated using the boilerplate from real templatic emails. The same tool is used to acquire additional ground truth data, improve data quality over time, and evaluate the performance of new models

The new architecture has delivered benefits along four dimensions. First, the precision and recall of the extractions have already surpassed the metrics from the heuristics-based system. Second, the new system has a smaller code footprint and is easier to maintain – we deleted over 45K lines of code within a few weeks of the migration. Section 6 describes the kinds of components from the old system that are now redundant. Third, the new system is easier to maintain – the rule-based system was brittle and made it difficult to debug errors and improve precision and recall. It is well known

10th Annual Conference on Innovative Data Systems Research (CIDR '20) January 12-15, 2020 , Amsterdam, Netherlands.

that the hand-crafted rules in an extraction system get too complicated to maintain and improve beyond a point [22]. The ML system on the other hand has provided steady improvements in precision and recall as we gather more training data. Finally, the new system can take advantage of recent advances in machine translation to produce extractions in non-English languages, a task that would not have been possible with the traditional system.

The rest of the paper is organized as follows: Section 2 first describes Juicer's template induction system. It then provides an overview of the rule-based extraction system followed by the learned extraction system. Section 3 describes the challenges encountered during the migration effort along with anecdotal examples. Section 4 describes the systems we built to manage training data, evaluate models, and manage label quality in the presence of privacy constraints. Section 5 presents related work. Finally, Section 6 summarizes our results and concludes the paper.

2. BACKGROUND

Juicer [27] is an information extraction system that runs over business-to-consumer (B2C) emails in Gmail. We have previously described the design of the system, focusing on the modeling choices and how we implemented new extraction tasks for verticals such as bill reminders, offers, and hotel reservations. We provide a brief summary here before describing the task of migrating legacy verticals and the challenges they pose.

The vast majority of B2C emails are machine-generated, and thus, instantiations of predefined templates. Juicer induces these templates—or at least a best guess—by clustering emails together that share similar structure. This is done by clustering incoming emails [28] based on a locality sensitive hash—we use Minhash [5]—of their set of XPaths that make up each email's HTML DOM tree. A cluster that has emails to at least k unique recipients, adhering to k-anonymity constraints, is considered viable for template induction.

Once a cluster is formed, Juicer labels each email in the cluster using a set of pre-trained *vertical* classifiers. Once all emails are classified, the template is assigned a single vertical label based on a majority vote of all the emails in the cluster. This label then determines the set of vertical-specific fields to extract. Below, we describe two approaches to extract field information from these emails.

2.1 Rule-Based Extraction

The legacy rule-based system used a traditional rule-based architecture [6, 16]. Several rules were developed by engineers over many years for each field in each vertical using emails donated by internal users for this purpose.

The rules were scoped to trigger on specific subsets of emails and within certain contexts in an email using regular expressions. For example, consider extracting the *order number* field from a purchase confirmation. Extraction rules were written using regular expressions to identify key context phrases (such as "order", "number", "order no.", etc.) that need to be present, and regular expressions to match the actual order number.

These rules were managed by limiting their scopes. For example, a scope could be defined so that a particular set of rules was only applicable for a subset of senders. Scopes could also be defined for portions of the email where the rule could be applied. For example, certain rules may only be applied to the content in the subject line. Other effective scoping strategies included requiring that the target field be present in a variable region in the email,[1] or disallowing extractions from certain kinds of HTML elements, such as URLs. Finally, each rule could also require additional validation conditions. For example, the check-out date should always be after the check-in date in a hotel reservation.

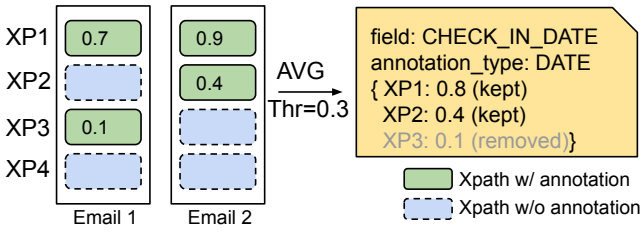

Figure 1: Field extraction rule generation. For each email in a template cluster, XPaths with candidate annotations (green boxes) are scored by the field classifier. XPaths with an average score above the threshold (Thr) are converted to extraction rules.

2.2 Machine-Learned Extraction

In the Software 2.0 approach, the extraction rules are machine-learned. Each field has a corresponding *field classifier* which is applied to a set of candidates in the email to identify the target field value.

Candidates are defined as all the spans of text with annotations of a certain type specific to that field, e.g. the candidates for hotel check-in date consist of all the date annotations in an email while the candidates for hotel address consist of all the address annotations. We rely on a library of annotators—employing a variety of techniques, such as regular expressions, heuristics, and knowledge graph lookup[2]—to annotate dates, email addresses, numbers, prices, addresses, cities, and so on in the emails. This library has been developed over years for various search and extraction tasks at Google.

A field classifier is trained to predict if a candidate corresponds to a field in the vertical. Then, during template induction, after a template has been assigned its vertical, the appropriate field classifiers are applied to all candidates in all the emails of the cluster. The field classifier scores are averaged across the emails per XPath in which the annotation is observed. If the average score is above the prescribed threshold, an extraction rule is added to the template for that XPath. Figure 1 depicts this process in detail.

These templates, along with their labels and field extraction rules, are stored in a key-value store. When a new email arrives, Juicer computes the Minhash of the email and performs a lookup for the template. If found, any extraction rules present are executed by traversing to the XPath in the email and extracting the text corresponding to the annotation in that XPath. If there are multiple rules for a field,

[1]Once we induce a template, we can identify which portions of a template are boilerplate text (fixed) vs. portions that vary across emails (variable).

[2]https://cloud.google.com/natural-language/docs/reference/rest/v1/Entity

they are executed in descending order of the field classifier score and the first successful extraction is used.

2.3 Evaluation

Extractions are evaluated by inspecting synthetic emails generated from the underlying templates and highlighting the results from either the legacy system or the machine-learned rules. Since email data is private no one has access to visually inspect any real data. Instead, emails are synthetically generated from the fixed text of the templates that pass the k-anonymity constraints [27]. This text is shared across all the emails in the template and contain no personal information. The variable portions of the email are identified by masking strings like 'XXXX'.

To ensure high-precision extraction, we maintain a whitelist of high-quality templates such that only emails matching templates in the whitelist trigger extractions. Any template which is evaluated to have accurate extractions on the synthetically generated emails are whitelisted for extraction in production. Since there may be thousands of new templates for each vertical every week, if a random subset of a large weekly batch of templates is evaluated with precision above 90%, the entire batch is whitelisted.

3. MIGRATION

The legacy system based on a rule-based architecture with hand-crafted rules has been in production since 2013. We focus on two verticals for migration in this paper: event confirmations (e.g. appointments, ticketed events, etc.), and purchases (e.g. order confirmations, shipment notifications, etc.).

Improving heuristic rules is a long development cycle. An engineer begins by writing extraction logic after observing donated emails, evaluates the logic on a larger set of donated emails, then repeats the process in an attempt to fix error cases. Oftentimes, fixing error cases results in breaking correct cases. Engineers end up spending significant time iterating through many develop-evaluate cycles to produce high precision heuristics. Over time, we developed sophisticated infrastructure to support more expressive rules, which become an increasing burden to maintain. This is a common pattern with rule-based extractions systems that has been observed by several practitioners [22, 29].

Our primary goal in migrating these verticals was to replace the existing rule-based extraction system with the much simpler machine-learned system. In doing so, we wanted to be able to guarantee continuing incremental improvements to verticals by gathering additional training data instead of writing increasingly complicated and difficult-to-maintain rules.

3.1 Challenges

The very first models that we trained were only able to replace 6% of the extractions from the heuristic rules. In order to close this gap and replace the rule-based system entirely with the new ML-based system, our challenges centered around managing our training data: building high coverage candidate generators, maintaining high precision labelers, and obtaining high quality ground-truth. These tasks become particularly challenging when dealing with email, since they must be approached with the additional constraint of maintaining complete privacy.

3.1.1 Building high coverage candidate generators

There is a huge variation in the types and formats of various fields. When building a candidate generator for the *purchase order number* field, we began by using all of the text spans annotated as numeric in an email. We discovered through analysis and tools described in Section 4 that the coverage of this annotator was insufficient to replace the rule-based extractions, as we were missing order numbers that contained alphabetical characters. We were able to increase coverage by incorporating alphanumeric annotations.

Other fields, such as *event location*, required much more work. We began by using only address annotations as candidates (e.g. "123 E. Monument Pkwy, Mountain View, CA"), however, many emails may not contain the full address of their events, but rather just shortened versions, such as venue names (e.g. "Cinemark 16"). Thus we had to consider many types of annotations to generate event location candidates, including address patterns or annotations of buildings or sites from an internal Knowledge Graph [4].

3.1.2 Maintaining high precision labels

Our next challenge was matching candidates with ground truth for generating labeled training data. Consider, for example, that the ground truth for an event location field is "123 East Monument Parkway, Mountain View, California 94043-1234". But the candidate that is most similar to the ground truth might only be in a slightly different format (e.g. "123 E. Monument Pkwy, Mountain View, CA"), only contain a portion of the full address (e.g. "Yoga Studio"), have different capitalization, or may even have each address component annotated separately (number, street, city, etc.).

In such cases, we may need to use multiple methods to match candidates to the available ground truth. Some simple field types, like dates, can be converted to a canonical format prior to comparison. But if information is missing, such as the year, we may still need to guess it from the context.

For more complicated fields, we used more advanced matching mechanisms. For instance, for the event location field, we match candidates to ground truth by their relative offset positions in the email (when both have such information), or use string matching and knowledge-based methods [20]. Approximate matching of complex fields is a well-studied problem with many sophisticated solutions [10].

In our experience, no one-size-fits-all rule applies to every field. Instead, we must try to strike a balance between high precision and high recall. If the equality function is too strict, we may miss many useful positive labels, but if it is too relaxed, we may create a noisy labeled dataset which the model cannot learn from.

3.1.3 Obtaining high quality ground-truth

Initial training data comes from sender-provided markup in the email in addition to extractions from the legacy rule-based system. As we discovered and whitelisted new templates for extractions through the ML system, those too contributed to additional training data. For the event and purchase verticals, a majority of extractions used as sources of ground truth were derived from the legacy rule-based system. While these extractions were incredibly useful, there were many cases in which the data was faulty, most likely due to bad rules that may have been put into production at some point in the past, and went undetected by human

```
template <typename CandidateType>
class CandidateGenerator {
  // Called once per email by the FieldExtractor.
  virtual std::vector<CandidateType> GetCandidates(
    const AnnotatedEmail& e) = 0;
};

template <typename CandidateType>
class CandidateLabeler{
  // Called once after initialization and before
  // calling GetLabelForCandidate().
  virtual void Start(const AnnotatedEmail& e) {}

  // Called once for each candidate produced by the
  // CandidateGenerator linked via the
  // FieldExtractor.
  virtual Label GetLabelForCandidate(
    const CandidateType& candidate) = 0;
};
```

Figure 2: Interfaces for a CandidateGenerator and CandidateLabeler.

evaluators at the time. Small volume templates are particularly susceptible to these issues. Identifying templates that were getting incorrect extractions from the legacy system and preventing them from contributing noise to the training data was key to improving the classifiers.

4. MANAGING TRAINING DATA

The underlying emails from which we generate training data is stored encrypted. Read-access is only available to specific role accounts running binaries whose code was reviewed by an engineer and checked into our source-control system. All access is audited to guarantee privacy and ensure that the only use of the data is to train models without allowing any engineer to inspect the data.

4.1 Training Data for Field Classifiers

We built two core abstractions to generate training data for each field: *CandidateGenerators* and *CandidateLabelers*, whose interfaces are defined in Figure 2. The CandidateGenerator implements the necessary logic to convert an annotated email to a collection of candidates to be used in model training or inference for field rule generation. The CandidateLabeler implements the logic to label each candidate as positive, negative, or unknown for use in model training. Recall from the discussion in Section 2.2 that candidates are derived from existing annotators for basic types likes dates, numbers, emails, and entity annotators that rely on the Knowledge Graph [4].

Consider the hotel confirmation check-in date field. To generate training data for this field we implement a generic *DateCandidateGenerator* that overrides the `GetCandidates` method to emit all of the dates present in the email. To label these candidates, we implement a *HotelCheckinDateLabeler*. Recall that training data for an individual field is derived from emails that have extractions from hand-written parsers and heuristics. When an extraction is successful, the email is annotated with a *hotel confirmation* object. Our implementation of the `Start` method stores the extracted check-in value in a ground truth member variable. The override of `GetLabelForCandidate` simply returns *positive* if the candidate matches the extracted ground truth, *negative* if it does not match, or *unknown* when a ground truth value could not be parsed to begin with.

These two class types are then bound together using a *FieldExtractor* which generates training data for that field:

```
class HotelCheckinDateGenerator
    : FieldExtractor<DateCandidateGenerator,
                     HotelCheckinDateLabeler> {}
```

4.1.1 Standard Counters

Given the private nature of emails, it is particularly difficult to gain insights into the quality of candidate generation or labeling logic if one cannot actually view the results over a real email sample. However, by linking a CandidateGenerator and a CandidateLabeler into the StandardFieldExtractor, we can produce a standard set of counters that can help provide statistical insights for debugging. These counters help us first understand what the coverage of a CandidateGenerator is, and furthermore provide ways to understand how to improve it by examining sample templates, which we discuss further below in Section 4.2. We highlight some of the particularly useful counters in Table 1 which track the number of emails which were ignored for the purposes of generating training data. For example, if the *emails-ignored-due-to-no-candidates-produced* count is a significant proportion of the number of emails with an existing extraction from the rule-based system, we know that we should first focus our efforts on increasing candidate coverage. Section 4.2 describes tools we used to improve this coverage.

These counters also help us debug problematic labeling logic. For example, if the *emails-ignored-due-to-only-negative-candidates* makes up a significant fraction of the total emails we generate candidates from, we know that our candidate labeling logic is failing to match the ground truth available in these emails and indicates we need to focus on improving that labeling logic.

In the initial stages of the migration, each engineer independently wrote a job that generated training data for a given field. Engineers made independent choices about when to include the data from a particular email for training. Consider the case where none of the candidates in an email were labeled positive. For some fields, engineers chose to use the data as negative examples, and in other fields engineers chose to treat that as a sign that the labeler was being too strict with approximate matching, and discard those training examples rather than pollute the training data.

This meant that every field extractor, and the methods by which they generated, labeled, counted, and returned the candidates had to be understood individually. With only a few engineers working on dozens of fields, the overhead to ramp up on an individual field extractor became too costly.

The FieldExtractor removes this overhead by providing a standard abstraction for all of the above: candidate generation, candidate labeling, counting, and consistent logic. Consequently, engineers are able to come up to speed on any field very quickly. Standardizing the counters across fields enables an engineer to quickly and easily deduce if the problems for a particular field lie in the coverage of candidate generation, or the labeling logic, or in the quality of the ground truth.

Furthermore, the FieldExtractor makes consistent choices across fields about whether to include data from an email if none of the candidates get labeled positive. More on this design choice below.

emails-ignored-due-to-	
no-candidates-produced	The number of emails for which no candidate annotations are available. This is a good indication to spend more effort to increase the coverage of the CandidateGenerator.
only-negative-candidates	The number of emails ignored due to having only negative labels. If it is expected that a positive example should exist whenever a ground-truth extraction is present, then this indicates that the CandiateLabeler's matching logic needs more attention.
only-unknown-candidates	The number of emails ignored due to having only unknown labels. This usually indicates that the labeler was unable to parse the ground truth, which may indicate that the ground truth itself is unavailable for this example.

Table 1: A sample of standard counters provided by the FieldExtractor that are particularly helpful when debugging training data generation for private data.

4.1.2 Consistent Training Data Generation

A high quality field classifier requires sufficient training data to distinguish between the positive and negative candidates for a field. Recall that we use existing extractions from the legacy rule-based system and other parsers hand-crafted using donated emails as our training data. Since the extracted fields are normalized slightly differently in different verticals we have to rely on an approximate match to assign labels. The FieldExtractor is designed to discard potentially noisy examples and thereby produce a higher quality training data set than might be implied by our confidence in the approximate labeler.

Consider the case where we have an existing extraction and all the candidates are assigned a negative label. This might happen for one of two reasons. The approximate labeler may be too strict (for example, not equating "Amphitheatre Pkwy" and "Amphitheatre Parkway"), or the correct candidate may not have been identified by the CandidateGenerator and the labeler, as expected, labeled all the other candidates as negative. It is difficult to know the underlying reason without actually inspecting a sample email. The FieldExtractor is designed to discard the training data from such emails for any field, and make this choice consistently across all fields. This allows engineers to focus on field-specific matching logic and rely on the infrastructure for providing high-quality training data. Empirically, we have observed that the classifiers trained on data where we excluded low-quality examples perform better in terms of precision on a holdout set.

Separating the candidate generation logic from the labeling logic allows fields that share the same underlying types to reuse CandidateGenerators. For example, the hotel check-in, hotel check-out, and bill due-date fields can all share the same underlying DateCandidateGenerator. This reduces the complexity of the codebase by removing duplicate implementations.

4.2 Tools for Evaluation

The counters described above can identify templates that heuristics successfully extracted from, but the ML system is unable to tackle either because the CandidateGenerator's coverage is insufficient (no-candidate-produced in Table 1) or the CandidateLabeler is too rigid (only-negative-candidates in Table 1). We extended the visual inspection tool described in Section 2.3 to highlight all the candidates as well as the extractions from the legacy rule-based system. This tool allowed us to discover during the migration of the purchase vertical that the CandidateGenerator for the order number field needed to use annotations from an alphanumeric annotator in addition to the number annotator. We were also able to identify cases where the extraction from the legacy rule-based system was incorrect and remove them.

4.3 Managing Ground Truth Data

The same synthetic email generation system is also used for obtaining and managing new high quality ground truth data. Once a template is classified as belonging to a particular vertical, and the required field extraction rules are generated for that template, we synthetically generate emails for that template and request human assessments for the vertical label and field extractions. These assessments are rather cheap, since they only require a yes/no answer to questions such as, "Is this a hotel confirmation?", "Is the correct check-in date extracted?", etc.

The answers to the vertical label question are stored per template, and used to label training data during periodic re-training of the vertical models. These labels are particularly important in two cases. First, when the human labels differ from the ground truth annotations, thereby correcting faulty heuristic-based or parser-based extractions and improving the quality of our training data overall. And second, when newly discovered templates are assessed, they provide additional training data. Note that negative assessments are equally, if not more, important since these are cases which the current classifier got wrong, and are thus valuable examples that can be used to adjust the decision boundary in the next retraining.

By assessing template vertical classification and periodically retraining, we can continuously improve our models over time without significant engineering effort.

5. RELATED WORK

This work draws on decades of research on information extraction from the data management, data mining, and machine learning communities. We focus on literature that tackles extracting to a given schema rather than *Open IE* [3] which tackles extracting relations in free text into triples for a knowledge base.

There is a rich body of work dealing with the idea of wrapper induction [2, 8, 11, 17]. HTML-formatted data is assumed to be produced by populating a template with values from a database, and given several examples, the challenge is to recover the underlying objects in the database. Most of the algorithms were designed for the web (as opposed to email) and required annotations for each site, and were brittle to changes in the layout of the page.

Rule-based extraction systems [16] have worked well in the context of writing high-precision extractors for narrow

types, in particular, for generating the kinds of candidate annotations for known entity types that systems like DIADEM [9] and Juicer rely on as part of a larger scalable solution. However, it is well known that explicit rules get very complicated rather quickly [29].

Sequence tagging and slot-filling tasks in the NLP community are closely related to the kind of information extraction problem we tackle. Recurrent neural architectures [12] produce close to state-of-the-art performance on these tasks. High quality extraction systems have recently been built leveraging such advances from machine learning. Snorkel [22] argued that acquiring training data is the key technical challenge. The authors have advocated for data programming [23] as a solution strategy along with leveraging techniques like multi-task models and weak supervision [24] to maximize value from training data. Ceres [18] shows that distant supervision can be effectively applied for relation extraction tasks on the web.

There is a large body of recent work studying privacy-preserving ML training using differential privacy [13]. Studies have shown shortcomings of using k-anonymity for publishing datasets [19, 21]. This paper focuses on the challenges of managing training data in the presence of privacy constraints. The training data is not made available for any purpose other than to train the models described. In fact, the data is stored encrypted, and read-access is granted only to role accounts running the training binary built from reviewed and checked-in code. The training procedure, while not the focus of this paper, can be made differentially private using existing techniques [1].

Our work reinforces the argument that a critical ingredient of a Software 2.0 approach for the real-world is *managing* training data. In contrast to solving a de-novo extraction problem, we focus on replacing a complex heuristics-based production extraction system with a completely machine-learned system that is easy to understand and improve. We argue that a key component for any such effort is a system to manage training data – including acquiring, debugging, versioning, and transforming it.

6. SUMMARY AND CONCLUSIONS

The migration of the Event and Purchase verticals from the heuristic rule-based extraction system to a Software 2.0 system has resulted in several benefits. Discovering and fixing incorrect legacy extractions resulted in increasing precision by 8 to 9 points. Second, the migration has allowed us to delete the heuristic rules and the code that executed them. This included various scoping rules, whitelists, blacklists, dictionaries of key phrases, etc. This shrank the extraction system codebase by over 45KLoC representing a very big long-term maintenance win. Third, the system makes it possible to deliver steady improvements in coverage and recall by gathering more training data and improving the models. Within a few weeks, the ML models discovered additional templates and rules resulting in an increase in the volume of extractions — 3.3% for purchases and 32.6% for events. The improvements for both verticals are summarized in Table 2. In contrast, the coverage of the heuristic-based extraction system had been flat for several months since it was too brittle to improve without introducing erroneous extractions. Fourth, and perhaps most importantly, this migration has allowed us to tackle cross-language extractions. Even though our original training data was only in English, we are able to leverage state-of-the-art cross-language word embeddings [26] to learn field and vertical models that work across multiple languages. To our knowledge, this would have been impossible with traditional rule-based techniques.

	Event		Purchase	
	Heuristics	ML	Heuristics	ML
Precision	90.0	95.3	90.0	97.5
Extraction volume	100.0	132.6	100.0	107.1

Table 2: Precision and extraction volume improvements after migration.

Systems to continue to improve training data quality and acquiring more training data through techniques like active learning are areas of future investigation.

Acknowledgements

We would like to acknowledge the help and support of all the engineers who contributed to this project including Gang Feng, Jing Xie, Qi Zhao, and Zhengying Chen.

7. REFERENCES

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

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

[3] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *International Joint Conferences on Artificial Intelligence*, pages 2670–2676, 2007.

[4] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the SIGMOD International Conference on Management of Data*, pages 1247–1250, 2008.

[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166, 1997.

[6] L. Chiticariu, Y. Li, and F. R. Reiss. Rule-based information extraction is dead! Long live rule-based information extraction systems! In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 827–832, 2013.

[7] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, et al. State-of-the-art speech recognition with sequence-to-sequence models. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4774–4778, 2018.

[8] N. Dalvi, R. Kumar, and M. Soliman. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment*, 4(4):219–230, 2011.

[9] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang. Diadem: Thousands of

websites to a single database. *Proceedings of the VLDB Endowment*, 7(14):1845–1856, 2014.

[10] W. H. Gomaa and A. A. Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18, 2013.

[11] P. Gulhane, A. Madaan, R. Mehta, J. Ramamirtham, R. Rastogi, S. Satpal, S. H. Sengamedu, A. Tengli, and C. Tiwari. Web-scale information extraction with Vertex. In *IEEE International Conference on Data Engineering*, pages 1209–1220, 2011.

[12] Z. Huang, W. Xu, and K. Yu. Bidirectional LSTM-CRF models for sequence tagging. *arXiv:1508.01991*, 2015.

[13] Z. Ji, Z. C. Lipton, and C. Elkan. Differential privacy and machine learning: a survey and review. *arXiv:1412.7584*, 2014.

[14] A. Karpathy. Software 2.0. https://medium.com/@karpathy/software-2-0-a64152b37c35, 2017.

[15] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the SIGMOD International Conference on Management of Data*, pages 489–504, 2018.

[16] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. SystemT: A system for declarative information extraction. *ACM SIGMOD Record*, 37(4):7–13, 2009.

[17] N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence*, pages 729–737, 1997.

[18] C. Lockard, X. L. Dong, A. Einolghozati, and P. Shiralkar. Ceres: Distantly supervised relation extraction from the semi-structured web. *Proceedings of the VLDB Endowment*, 11(10):1084–1096, 2018.

[19] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 2007. Article 3.

[20] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *Proceedings of the National Conference on Artificial Intelligence - Volume 1*, pages 775–780, 2006.

[21] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pages 111–125, 2008.

[22] A. J. Ratner, S. H. Bach, H. R. Ehrenberg, and C. Ré. Snorkel: Fast training set generation for information extraction. In *Proceedings of the SIGMOD International Conference on Management of Data*, pages 1683–1686, 2017.

[23] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *Neural Information Processing Systems*, pages 3567–3575, 2016.

[24] A. J. Ratner, B. Hancock, and C. Ré. The role of massively multi-task and weak supervision in software 2.0. In *Proceedings of the 9th Biennial Conference on Innovative Data Systems Research*, 2019.

[25] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment*, 10(11):1190–1201, 2017.

[26] S. Ruder. A survey of cross-lingual embedding models. *arXiv:1706.04902*, 2017.

[27] Y. Sheng, S. Tata, J. B. Wendt, J. Xie, Q. Zhao, and M. Najork. Anatomy of a privacy-safe large-scale information extraction system over email. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 734–743, 2018.

[28] M. Whittaker, N. Edmonds, S. Tata, J. B. Wendt, and M. Najork. Online template induction for machine-generated emails. *Proceedings of the VLDB Endowment*, 12(11):1235–1248, 2019.

[29] S. Zhang, L. He, E. Dragut, and S. Vucetic. How to invest my time: Lessons from human-in-the-loop entity extraction. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2305–2313, 2019.