Extracting Product Information from Email Receipts Using Markov Logic

Stanley Kok*
Department of Computer Science & Engineering
University of Washington
Seattle, WA 98195
koks@cs.washington.edu

Wen-tau Yih
Microsoft Research
One Microsoft Way
Redmond, WA 98052
scottyih@microsoft.com

ABSTRACT

Email receipts (e-receipts) frequently record e-commerce transactions between users and online retailers, and contain a wealth of product information. Such information could be used in a variety of applications if it could be reliably extracted. However, extracting product information from e-receipts poses several challenges. For example, the high labor cost of annotating e-receipts makes traditional supervised approaches infeasible. E-receipts may also be generated from a variety of templates, and are usually encoded in plain text rather than HTML, making it difficult to discover the regularity of how product information is presented. In this paper, we present an approach that addresses all these challenges. Our approach is based on Markov logic [22], a language that combines probability and logic. From a corpus of unlabeled e-receipts, we identify all possible templates by jointly clustering the e-receipts and the lines in them. From the non-template portions of e-receipts, we learn patterns describing how product information is laid out, and use them to extract the product information. Experiments on a corpus of real-world e-receipts demonstrate that our approach performs well. Furthermore, the extracted information can be reliably used as labeled data to bootstrap a supervised statistical model, and our experiments show that such a model is able to extract even more product information.

1. INTRODUCTION

Many e-commerce transactions (e.g., product purchase, order cancellation, etc.) are recorded in email receipts (e-receipts), which are a rich source of product information. Such information, when adequately extracted, can enable a huge variety of applications.

For example, a software agent can automatically keep track of the transactions a user has executed, and compile a database of purchased products. Such a database can be used to remind a user of the expiration of a warranty, or to periodically suggest the replenishment of products such as batteries, toners, or ink cartridges.

From the perspective of an email service provider, transactional information aggregated from e-receipts can be used for building a recommendation system. If a set of products are often observed as being purchased together in e-receipts, then these products can be treated as being related. When the system has identified what a user has bought by mining his e-receipt, it can automatically suggest related products to the user. For example, if the user has bought a cell phone, the system may suggest Bluetooth earphones; if the product is a movie DVD, the system may suggest other related movies.

Business intelligence can also be collected from e-receipts. From the product information extracted from a large collection of e-receipts, we can learn about the business volume of retailers, track the price range of products over time, identify trends in product popularity, build user profiles, etc.

All the above applications rely on the core technology of production information extraction. There are three challenges in mining product information from e-receipts. First, there is no training data, and hand-labeling the product information in e-receipts from every retailer is a costly and error-prone enterprise. Hence, an unsupervised approach is required. Second, a lot of e-receipts are not structured, i.e., encoded in standard markup languages like HTML. (All the e-receipts in our corpus are in plain text, and only about 40% are also encoded in HTML.) However, the e-receipts are also not completely unstructured. Most, if not all, of the e-receipts are generated from predefined templates. (See Figure 1 for examples of e-receipts that are generated from the same template.) E-receipts from the same template share features, which allow us to reverse engineer the template from the e-receipts. We can then identify the non-template elements in each e-receipt, and extract them as potential product information. Third, a corpus of e-receipts may be generated from many templates, and we must identify which e-receipts are generated from the same template. Even when the e-receipts are from the same retailer, they can still be generated from several different templates (e.g., templates for purchase confirmation, cancellation, etc.).

Previous product mining approaches require labeled training data (i.e., are supervised) [14, 18, 16, 25], can only work with marked-up input (e.g., HTML webpages) [4, 9, 1, 27, 3], or require their input to be generated from the same template [4, 9, 1].

In this paper, we present PIEML, a system that addresses all three challenges, i.e., it is unsupervised, works on plain-text e-receipts, and learns which e-receipts are from the same template. PIEML is defined using Markov logic [22], and

*This work was done while the author was an intern at Microsoft Research.

CEAS 2009 - Sixth Conference on Email and Anti-Spam July 16-17, 2009, Mountain View, California USA

is short for Product Information Extraction with Markov Logic. PIEML works by jointly clustering e-receipts and the lines in them. Each cluster of e-receipts corresponds to a type of transaction, and are generated from the same template. Each cluster of lines corresponds to (part of) a template when its lines appear in most of the e-receipts in a cluster. For each e-receipt cluster, PIEML marks the template lines in its e-receipts. On the remaining lines, PIEML learns the layout of the product information, and uses it to extract the product information.

We applied PIEML to a real-world corpus of e-receipts, and found that it extracted product information with high precision. We also used the learned layouts and extracted product information to label the e-receipts, and trained a standard conditional random field (CRF) [15] on the labeled data. We found that the CRF performed well in extracting new product information.

We begin by reviewing Markov logic in the next section. Next we describe our PIEML system (section 3), and report our experiments with it (section 4). Finally, we discuss related work (section 5), and future work (section 6).

2. MARKOV LOGIC

Markov logic combines first-order logic and Markov networks so as to model both relational dependencies and uncertainty. A key advantage of Markov logic is that it allows us to easily define a statistical model by simply writing intuitive (weighted) first-order rules.

In *first-order logic* [10], formulas are constructed using four types of symbols: constants, variables, functions, and predicates. (In this paper we use only function-free logic.) Constants represent objects in the domain of discourse (e.g., people: `Anna`, `Bob`, etc.). Variables (e.g., `x`, `y`) range over the objects in the domain. Predicates represent relations among objects (e.g., `Friends`), or attributes of objects (e.g., `Student`). Variables and constants may be typed. An *atom* is a predicate symbol applied to a list of arguments, which may be variables or constants (e.g., `Friends(Anna, x)`). A *ground atom* is an atom all of whose arguments are constants (e.g., `Friends(Anna, Bob)`). A *world* is an assignment of truth values to all possible ground atoms. A database is a partial specification of a world; each atom in it is true, false or (implicitly) unknown. A *database* is a partial specification of a world; each atom in it is true, false or (implicitly) unknown. In this paper, we make a closed-world assumption: a ground atom not in the database is assumed to be false. Formulas are recursively constructed from atoms using logical connectives and quantifiers. If F_1 and F_2 are formulas, the following are also formulas: $\neg F_1$ (negation), which is true iff F_1 is false; $F_1 \vee F_2$ (disjunction), which is true iff F_1 or F_2 is true; $F_1 \wedge F_2$ (conjunction), which is true iff both F_1 and F_2 are true; $F_1 \Rightarrow F_2$ (implication), which is true iff F_1 is false or F_2 is true; $F_1 \Leftrightarrow F_2$ (equivalence), which is true iff F_1 and F_2 have the same truth value; $\exists \mathtt{x}\ F_1$ (existential quantification), which is true iff F_1 is true for at least one object `x` in the domain; and $\forall \mathtt{x}\ F_1$ (universal quantification), which is true iff F_1 is true for every object `x` in the domain; Formulas allows us to represent complex dependencies among several atoms.

A *Markov network* [20] is a model for the joint distribution of a set of variables $X = (X_1, \ldots, X_n) \in \mathcal{X}$. It is composed of an undirected graph G and a set of potential functions ϕ_k. The graph has a node for each variable, and the model has a potential function for each clique in the graph. A potential function is a non-negative real-valued function of the state of the corresponding clique. The joint distribution represented by a Markov network is given by $P(X=x) = \frac{1}{Z}\prod_k \phi_k(x_{\{k\}})$ where $x_{\{k\}}$ is the state of the kth clique (i.e., the state of the variables that appear in that clique). Z, known as the *partition function*, is given by $Z = \sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}})$. Markov networks are often conveniently represented as *log-linear models*, with each clique potential replaced by an exponentiated weighted sum of features of the state, leading to $P(X=x) = \frac{1}{Z}\exp\left(\sum_j w_j f_j(x)\right)$. A feature may be any real-valued function of the state. This paper will focus on binary features, $f_j(x) \in \{0, 1\}$. In the most direct translation from the potential-function form, there is one feature corresponding to each possible state $x_{\{k\}}$ of each clique, with its weight being $\log \phi_k(x_{\{k\}})$. This representation is exponential in the size of the cliques. However, we are free to specify a much smaller number of features (e.g., logical functions of the state of the clique), allowing for a more compact representation than the potential-function form, particularly when large cliques are present. Markov logic takes advantage of this.

A first-order knowledge base (KB) is a set of first-order formulas, which can be viewed as hard constraints on the set of possible worlds. If a world violates a formula, it has zero possibility. This causes first-order KBs to be brittle. To fix this, Markov logic softens the constraints by attaching weights to the formulas so that when a world violates a formula it becomes less probable, but not impossible. More formally, a *Markov logic network (MLN)* is a set of weighted first-order formulas. Together with a set of constants representing objects in the domain, it defines a Markov network with one node per ground atom and one feature per ground formula. The weight of a feature is the weight of the first-order formula that originated it. The probability distribution over possible worlds x specified by the ground Markov network is given by $P(X = x) = \frac{1}{Z}\exp\left(\sum_{i \in F}\sum_{j \in G_i} w_i g_j(x)\right)$, where Z is the partition function, F is the set of all first-order formulas in the MLN, G_i is the set of groundings of the ith first-order formula, and $g_j(x) = 1$ if the jth ground formula is true and $g_j(x) = 0$ otherwise. Markov logic enables us to compactly represent complex models in non-i.i.d. domains. General algorithms for inference and learning in Markov logic are discussed in [22].

3. PRODUCT INFO EXTRACTION

We call our algorithm PIEML, for Product Information Extraction with Markov Logic. PIEML takes as input a set of e-receipts sent from a common retailer email address, and outputs the product information in them. It makes two assumptions. First, it assumes that product information is laid out in two simple formats that can be expressed as regular expressions. These regular expressions are flexible enough to represent many different product layouts. Second, it assumes the existence of a database of product names, which it uses to match candidate product names that it extracts. The matched product names are then used to validate the product information layouts that PIEML creates.

PIEML is made up of two components: TemplateMarker and LayoutDiscover. TemplateMarker jointly clusters e-receipts and the lines in them. Each cluster of e-receipts

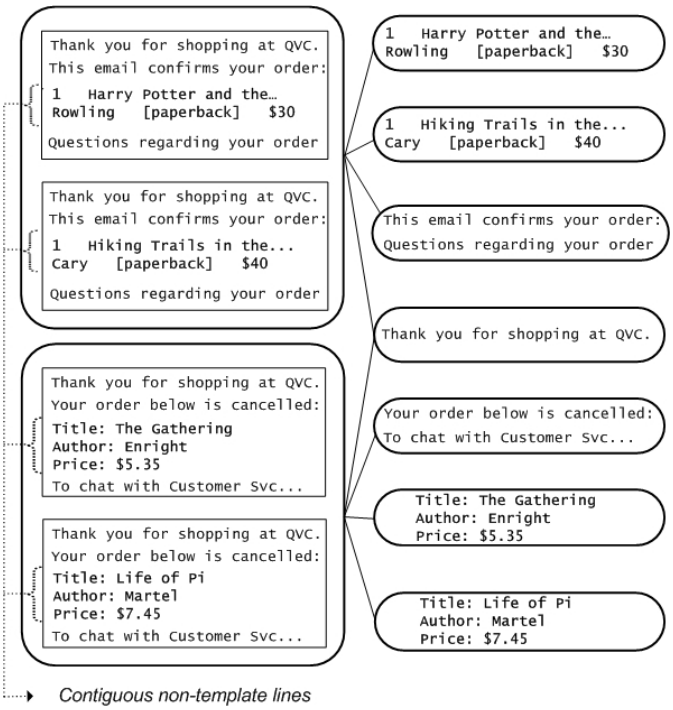

Figure 1: Clusters created by TemplateMarker. The e-receipt and line clusters are respectively on the left and right. The e-receipts in a cluster are generated from the same template. The bold lines contain product information, and the non-bold lines are part of a template. Each line cluster is linked to e-receipt clusters that contain at least one of its lines. E-receipt and line clusters are linked together as a cluster combination.

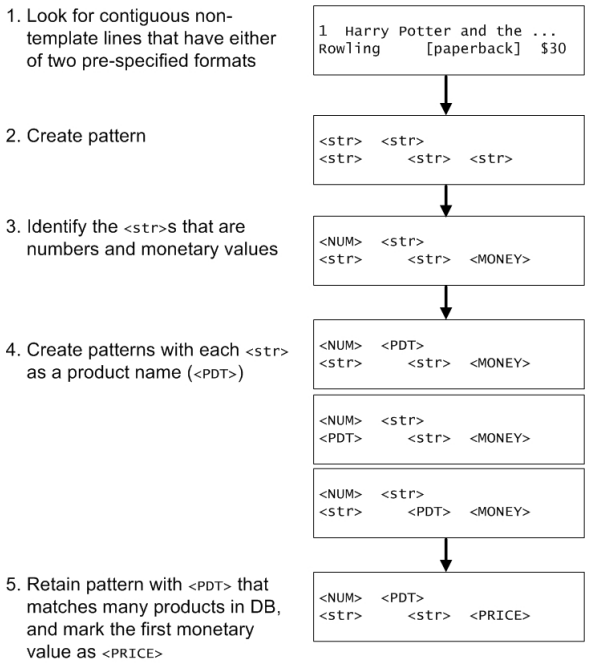

Figure 2: Illustration of LayoutDiscover algorithm. The pattern learned in step 5 is used to extract new product information.

corresponds to a type of transaction, and are generated from the same template. Each cluster of lines potentially corresponds to (part of) a template. TemplateMarker marks the lines in e-receipts that belong to a template, and hands the clusters of marked e-receipts to LayoutDiscover. For each cluster, LayoutDiscover learns how product information is laid out on the non-template lines in the e-receipts, and extracts the product information. In this paper, for simplicity, we focus on extracting product name and price.

3.1 Template Marking

TemplateMarker takes as input a set of e-receipts from the same retailer email address, and outputs e-receipt clusters. It marks the lines in the e-receipts that are part of a template. TemplateMarker works by jointly clustering the e-receipts and the lines in them in a bottom-up agglomerative manner, allowing information to propagate from one cluster to another as they are formed. This process groups together e-receipts with many lines in common (i.e., e-receipts generated from the same template), and groups together lines that appear together in many e-receipts (i.e. lines that are part of a template). The number of clusters need not be specified in advance.

TemplateMarker's statistical model is defined using Markov logic by specifying a few simple rules. We define a binary predicate $HasLine(e, l)$, which is true if and only if the e-receipt in its first argument contains the line in its second argument. (A line is defined as sequence of non-newline characters that are terminated by a newline.) We use e and l to respectively denote the first and second arguments of $HasLine$; E and L to respectively denote a set of e-receipts and a set of lines; γ_e and Γ_E to respectively denote a cluster and clustering (i.e., a partitioning) of e-receipts; and γ_l and Γ_L to respectively denote a cluster and clustering of lines. If e and l are respectively in cluster γ_e and γ_l, we say that (e, l) is in the *cluster combination* (γ_e, γ_l), and that (γ_e, γ_l) *contains* the ground atom $HasLine(e, l)$. We say (γ_e, γ_l) contains a true ground atom if there exist $e \in \gamma_e$ and $l \in \gamma_l$ such that the ground atom $HasLine(e, l)$ is true.

The learning problem in TemplateMarker consists of finding the cluster assignment $\Gamma = (\Gamma_E, \Gamma_L)$ that maximizes the posterior probability $P(\Gamma|D) \propto P(\Gamma, D) = P(\Gamma)P(D|\Gamma)$, where D is a vector of truth assignments to the observable $HasLine(e, l)$ ground atoms. The prior $P(\Gamma)$ is simply an MLN containing two rules. The first rule states that each e-receipt or line belongs to exactly one cluster:

$$\forall x \, \exists^1 \gamma \;\; x \in \gamma$$

(x could be an e-receipt or line, and γ could be a cluster of e-receipts or lines.) This rule is hard, i.e., it has infinite weight and cannot be violated. If this rule violated, $P(\Gamma)$ and consequently $P(\Gamma|D)$ equal zero.

The second rule is

$$\forall \gamma_e, \gamma_l \, \exists \, e, l \;\; e \in \gamma_e \wedge l \in \gamma_l$$

with negative weight $-\infty < -\lambda < 0$, which imposes an exponential prior on the number of cluster combinations to prevent the proliferation of cluster combinations (and thus

Algorithm 1: TemplateMarker Algorithm

function $TemplateMarker(E, k, f)$
  input: E, a set of e-receipts from the same retailer email address. Each e-receipt is an ordered multiset of lines.
    k, minimum e-receipt cluster size.
    f, minimum fraction of e-receipts in a cluster that a line must appear in before it is considered part of a template.
  output: Γ_E, a set of e-receipt clusters. Each e-receipt has its template lines marked.
  calls: $CreateTrueGroundAtoms(E)$, returns a set of true $HasLine$ ground atoms. The set contains a $HasLine(e, l)$ ground atom for every email $e \in E$ that contains line l.
    $GetAllLines(E)$, returns a set of the lines in emails in E.
    $ClusterPairWithBestGain(\Gamma_T)$, returns the pair of clusters in Γ_T that gives the highest improvement in log-posterior when merged.
    $MarkLine(l, \gamma_e)$, marks line l in γ_e's e-receipts as being part of a template.

$D \leftarrow CreateTrueGroundAtoms(E)$
$L \leftarrow GetAllLines(E)$
for each $T \in \{E, L\}$
  $\Gamma_T \leftarrow \emptyset$
  for each $x \in T$
    $\Gamma_T \leftarrow \Gamma_T \cup \{\gamma_x\}$ ($\{\gamma_x\}$ is a unit cluster containing x)
$C \leftarrow \emptyset$ (C contains cluster combinations)
for each true ground atom $HasLine(e, l) \in D$
  $C \leftarrow C \cup \{(\gamma_e, \gamma_l)\}$
repeat
  for each $T \in \{E, L\}$
    $(\gamma_{best}, \gamma'_{best}) \leftarrow ClusterPairWithBestGain(\Gamma_T)$
    if $\{(\gamma_{best}, \gamma'_{best})\} \neq \emptyset$
      $\gamma_{new} \leftarrow \gamma_{best} \cup \gamma'_{best}$
      $\Gamma_T \leftarrow (\Gamma_T \setminus \{\gamma_{best}, \gamma'_{best}\}) \cup \{\gamma_{new}\}$
      for each $\gamma \in \{\gamma_{best}, \gamma'_{best}\}$
        if $T = E$
          for each $(\gamma, \gamma_l) \in C$
            $C \leftarrow (C \setminus \{(\gamma, \gamma_l)\}) \cup \{(\gamma_{new}, \gamma_l)\}$
        else
          for each $(\gamma_e, \gamma) \in C$
            $C \leftarrow (C \setminus \{(\gamma_e, \gamma)\}) \cup \{(\gamma_e, \gamma_{new})\}$
until no clusters are merged
for each $\gamma_e \in \Gamma_E$
  if $|\gamma_e| \geq k$
    for each $(\gamma_e, \gamma_l) \in C$
      for each $l \in \gamma_l$
      if l appears in more than $f \times |\gamma_e|$ e-receipts in γ_e
        $MarkLine(l, \gamma_e)$
  else $\Gamma_E \leftarrow \Gamma_E \setminus \gamma_e$
return Γ_E

Algorithm 2: LayoutDiscover Algorithm

function $LayoutDiscover(\Gamma_E, DB, n, m)$
  input: Γ_E, a set of e-receipt clusters from the same retailer email address. The e-receipts have their template lines marked.
    DB, database of product names.
    n, minimum length of candidate product name
    m, minimum number of product names in DB that a pattern must match
  output: DB', database of (product name, price) pairs.
  calls: $GetContiguousNonTemplateLines(e)$, returns a set of regions of contiguous non-template lines in e-receipt e.
    $GetAllContiguousSubRegions(\Lambda)$, returns all sub-regions of contiguous non-template lines from the region Λ. Each sub-region is an ordered multiset of contiguous non-template lines.
    $GetPattern(L)$, returns one of the two pre-specified formats that the lines in L conform to; returns $\emptyset$ if L does not match any format.
    $ReplaceNumberAndMoney(p)$, marks the `<str>`s in pattern p that are numbers or monetary values.
    $Match(\texttt{<str>}), DB)$, returns true if `<str>` matches a product name in database DB; otherwise returns false.

$DB' \leftarrow \emptyset$ (DB' contains new product names and prices)
for each $\gamma_e \in \Gamma_E$
  $P \leftarrow \emptyset$ (P is a set of patterns)
  for each $e \in \gamma_e$
    $\Lambda \leftarrow GetContiguousNonTemplateLines(e)$
    $\Lambda' \leftarrow GetAllContiguousSubRegions(\Lambda)$
    for each $L \in \Lambda'$ (L is an ordered set of contig. non-template lines)
      $p \leftarrow GetPattern(L)$
      if $p \neq \emptyset$
        $p \leftarrow ReplaceNumberAndMoney(p)$
        for each `<str>` $\in p$
          if $NumTokens(\texttt{<str>}) \geq n$ **and** $MatchInDB(\texttt{<str>}, DB)$
            $p' \leftarrow MarkAsProductName(\texttt{<str>}, p)$
            if $p' \in P$
              $NumMatches[p'] \leftarrow NumMatches[p'] + 1$
            else
              $P \leftarrow P \cup p'$
              $NumMatches[p'] \leftarrow 1$
  $p \leftarrow GetPatternWithMostMatches(P, NumMatches)$
  if $NumMatches[p] \geq m$
    for each $\gamma_e \in \Gamma_E$ (use pattern p to extract product info)
      $\Lambda \leftarrow GetContiguousNonTemplateLines(e)$
      $\Lambda' \leftarrow GetAllContiguousSubRegions(\Lambda)$
      for each $L \in \Lambda'$
        $(name, price) \leftarrow ExtractProductInfo(p, L)$
        if $(name, price) \notin DB$
          $DB' \leftarrow (name, price)$
return DB'

clusters), and consequent overfitting. This rule helps to 'compress' e-receipts and lines together into clusters. The parameter λ is fixed during learning, and is the penalty in log-posterior incurred by adding a cluster combination.

The main MLN for the likelihood $P(D|\Gamma)$ contains the following rules. For each cluster combination (γ_e, γ_l) that contains a true ground atom, the MLN contains the rule:

$$\forall e, l \;\; e \in \gamma_e \wedge l \in \gamma_l \Rightarrow HasLine(e, l)$$

We call these *atom prediction* rules because they state that the truth value of an atom is determined by the cluster combination it belongs to. These rules are soft. At most there can be one such rule for each true ground atom (i.e., when each e-receipt and line is in its own cluster). These rules 'encourage' e-receipts that are generated from the same template to be clustered together. They also 'encourage' lines that belong to the same template to be clustered together. This causes the combination of e-receipt cluster and line cluster to be highly predictive of the template lines in each e-receipt.

We also have the rule

$$\forall e, l \;\; \left(\bigwedge_{i=1}^{m} \neg(e \in \gamma_e^i \wedge l \in \gamma_l^i)\right) \Rightarrow HasLine(e, l)$$

where $(\gamma_e^1, \gamma_l^1), \ldots, (\gamma_e^m, \gamma_l^m)$ are cluster combinations containing true ground atoms. This rule accounts for all $HasLine$ ground atoms (all false) that are not in any cluster combination with true ground atoms. We call such a rule a *default* atom prediction rule because its antecedents are analogous to a default cluster combination that contains all atoms that are not in the cluster combinations of any atom prediction rule.

TemplateMarker simplifies the learning problem by performing hard assignments of e-receipts and lines to clusters (i.e., instead of computing probabilities of cluster membership, an e-receipt/line is simply assigned to its most likely cluster). The weights and the log-posterior can now be computed in closed form (see appendix).

Algorithm 1 shows the pseudocode of TemplateMarker. TemplateMarker simply searches over cluster assignments, evaluating each one by its posterior probability. It begins

by assigning each e-receipt e and line l to its own cluster $\{e\}$ and $\{l\}$, and creating a cluster combination $(\{e\}, \{l\})$ for each true ground atom $HasLine(e, l)$. Next it creates candidate pairs of clusters of each type (i.e., e-receipt or line), and for each pair, it evaluates the gain in posterior probability if its clusters are merged. It then chooses the pair that gives the largest gain to be merged. When e-receipt clusters γ_e and γ'_e are merged to form γ_e^{new}, each (γ_e, γ_l) is replaced with $(\gamma_e^{new}, \gamma_l)$ (and similarly for cluster combinations containing γ'_e). To avoid trying all possible candidate pairs of clusters, TemplateMarker only tries to merge γ_e and γ'_e if they appear in cluster combinations (γ_e, γ_l) and (γ'_e, γ_l) (i.e., they have a line cluster in common). Line clusters are merged in the same way. In this manner, TemplateMarker incrementally merges clusters until no merges can be performed to improve posterior probability. After that, for each e-receipt cluster γ_e containing at least k e-receipts, it finds all cluster combinations (γ_e, γ_l) that contains true ground $HasLine$ atoms. For each line in γ_l, it marks that line as being part of a template in all of γ_e's e-receipts if the line appears in more than f fraction of the e-receipts.

TemplateMarker is similar to the MRC and SNE algorithms [12, 13]. It differs from them in the following ways. TemplateMarker finds a single clustering of e-receipts and lines, whereas MRC can find multiple clusterings. TemplateMarker does not cluster predicates since there is only one predicate (i.e., $HasLine$), whereas both MRC and SNE cluster predicates. TemplateMarker does not contain a rule of SNE that states that symbols tend to be in different clusters. SNE uses this rule to combat the extreme noise and sparsity of its Web data, characteristics which are not present in our e-receipt corpus. Empirically, we found that we perform better on our e-receipt corpus when we set the weight of this rule to zero (which is equivalent to excluding the rule).

3.2 Product Layout Discovery

LayoutDiscover takes as input the e-receipt clusters output by TemplateMarker, and outputs a database of product names and prices. Each e-receipt in the clusters has its template lines marked. LayoutDiscover assumes that product names and prices are laid out on contiguous non-template lines in two simple user-specified formats, which are specific to e-receipts. (A set of non-template lines are contiguous if they are not separated by any template lines. See Figure 1 for examples of non-template lines.) The two formats can be represented in regular expression as: `<str>(<sep><str>)+(<newline>(<str>(<sep><str>)*))*` and `(<str><sep>*:<sep>*<str>(<newline>|<sep>)*)+`. The symbol `<sep>` represents a separator that begins with a tab or two contiguous whitespaces, followed by any combination of tabs and whitespaces (possibly none). `<str>` represents a string that is a potential product name or price. It can contain any characters except tabs and two or more contiguous whitespaces. (The bold lines in the e-receipts in Figure 1 show examples of the two formats.)

Algorithm 2 shows the pseudocode of LayoutDiscover. LayoutDiscover iterates over the clusters of e-receipts learned by TemplateMarker. For each e-receipt in a cluster, LayoutDiscover extracts the contiguous regions of non-template lines from them. For each region, it finds all sub-regions of contiguous lines that satisfy any of the two formats. (For example, in each of the lower two e-receipts in Figure 1, there are 6 sub-regions formed from three contiguous non-template lines l_0, l_1, l_2: $\{l_0, l_1, l_2\}$, $\{l_0, l_1\}$, $\{l_1, l_2\}$, $\{l_0\}$, $\{l_1\}$, and $\{l_2\}$.) For each sub-region, it instantiates a pattern representing its layout. (See Figure 2 for an illustration of LayoutDiscover from this step onwards.) LayoutDiscover identifies which `<str>`s in the pattern are the product name and price. (We assume there are only one product name and price in each sub-region). It first identifies `<str>`s that are numbers and monetary values. For each remaining `<str>` with at least n tokens, it queries an existing database of product names to check whether there is any existing product that matches the first n tokens of `<str>`. If there is a match, the `<str>` is marked as a potential product name. A separate pattern is created for each matched `<str>`. We keep a count of the number of product names that a pattern matches, and only retain patterns with at least m matches. The first monetary value (if present) that appears in the pattern is then heuristically considered the price. For each e-receipt cluster, LayoutDiscover may learn several patterns, but it only retains the one with the largest number of matches. It then uses this pattern to extract new product names (i.e., those that do not appear in the database) from the non-template regions of the e-receipts that conform to the pattern.

PIEML is general enough to be adapted to work in other domains. All we need to do is to modify the formats (i.e., the regular expressions) representing the layout of the product information, and provide a database of product information that is specific to the domain of interest.

4. EXPERIMENTS

In this section, we describe how we construct a corpus of real-world e-receipts, and present the results of applying PIEML to it. Because PIEML is fairly robust in accurately discovering the templates and product information layouts, product items extracted by PIEML can be reliably used to train a statistical information extractor, which helps discover more product information. We explore this direction, and report the result at the end of this section.

4.1 Data

Despite the fact that there are publicly available email corpora, such as the TREC dataset [8, 7] and the Enron corpus [11], those corpora typically do not contain a sizeable number of e-receipts. Therefore, we only conducted our experiments on the proprietary corpus created from the Hotmail Feedback Loop email collection.

Originally designed as a mechanism for collecting email messages labeled as spam or non-spam, the Feedback Loop data consists of roughly 50,000 messages per day from Hotmail volunteers. A message sent to a user is randomly selected, regardless of whether it is headed to the inbox, junk folder, or is deleted. The user is then asked to label this message as either *spam* or *non-spam*. Notice that although this sampling method is not truly uniform, internal studies suggest that this dataset is pretty close to the mail distribution received by Hotmail users. Interested readers can find more information about the Hotmail Feedback Loop dataset in [26, 5].

From the non-spam emails in the Feedback Loop dataset sampled during the period from 2006 to mid-2008, we extracted a corpus of 76,875 e-receipts sent from 16,398 unique retailer email addresses by selecting email satisfying the fol-

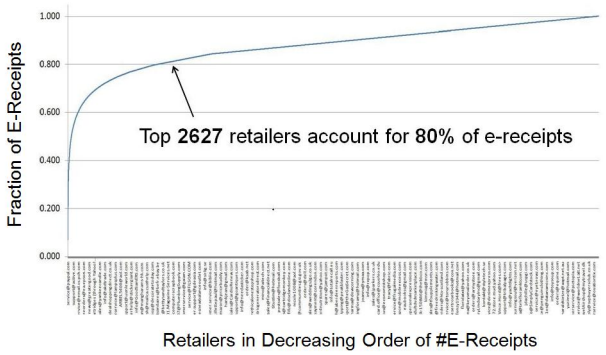

Figure 3: Cumulative distribution of the number of e-receipts from different retailers. We ranked the retailers (i.e., sender addresses) in decreasing order of the numbers of their e-receipts in our corpus. Most messages come from a few top retailers. For example, the top 111 retailers account for 50% of e-receipts, and the top 2,627 retailers account for 80% of e-receipts.

lowing criteria:

- Email subject contains the words 'receipt', 'purchase', or 'order', but not the words 'free', 'coupon', or 'off' (words indicative of promotional messages).
- Email body has the words 'qty', 'quantity', 'tax', 'item', 'price', 'total', 'subtotal', or 'amount'.

In spite of its simplicity, this heuristic method achieved high precision (i.e., a large fraction of selected emails were actual e-receipts). To evaluate the fraction of emails in this corpus that were actual e-receipts, we randomly sampled 200 emails and found that all were e-receipts. Although training an e-receipt classifier in a principled manner will no doubt increase the recall (i.e., fraction of e-receipts selected from among all e-receipts in the corpus), we focus in this paper on extracting product information *given* a corpus of e-receipts, and leave the e-receipt classifier as an item of future work.

There are several points of interest about this corpus. Most e-receipts come from relatively few retailers. As shown in Figure 3, 80% of the e-receipts were sent by the top 2,627 retailers, and each of them has at least 3 e-receipts in our corpus. All the e-receipts in the corpus are encoded in plain text, and about 40% repeat their contents in HTML. In order to have a general system that can work on all e-receipts, our method works directly on plain text without using the structural information encoded in HTML.

4.2 Results

We applied PIEML to the corpus of e-receipts, and show the results of template marking and product layout discovery respectively.

4.2.1 Template Marking

As described in Section 3, the TemplateMarker algorithm (Algorithm 1) builds on top of Markov logic, and concurrently constructs e-receipt clusters and template line clusters. We divided the corpus into disjoint subsets by their sender (retailer) email addresses, and provided each subset in turn as the input to TemplateMarker. When running TemplateMarker on our e-receipt corpus, we require each e-receipt cluster to have at least 3 e-receipts, and a line to appear in at least half of the messages in an e-receipt cluster before it can be considered a template line (i.e., the parameter values used in Algorithm 1 were $k = 3$ and $f = 0.5$). We also set the weight λ of the second rule in the MLN defining the prior $P(\Gamma)$ to 10. In order to find a good trade-off between the coverage of e-receipts and the risk of generating unreliable templates, we selected those values based on some preliminary experiments. Notice that this setting effectively ignored roughly 20% of the email in our corpus. The total number of e-receipts that are processed by TemplateMarker is 61,035. Among them, 60,914 (99.8%) e-receipts were clustered into 3,621 templates. The average number of e-receipts in each template cluster is 16.8, and the average number of template clusters for each retailer is 1.39. Recall that a big portion of e-receipts is sent from a smaller number of top retailers. These retailers generally have more templates and cover more e-receipts.

4.2.2 Layout Discovery

Given the 3,621 clusters of e-receipts identified by TemplateMarker, we applied our LayoutDiscover algorithm (Algorithm 2) to the non-template lines of these e-receipts to generate patterns for extracting product information. One of the important step for LayoutDiscover to conclude that a sequence of words is a product name is to find a match in our product database, which consists of 22.7 million entries. To allow high-confidence pattern generation, the candidate product name needs to contain at least 4 words. In addition, the discovered pattern needs to match at least 10 product names in the database before it can be considered as a valid pattern (i.e., the parameter values used in Algorithm 2 were $n = 4$ and $m = 10$).

Overall, when applied to our corpus of e-receipts, PIEML extracted 40,538 pairs of product names and prices. To evaluate the correctness of the extracted pairs, we sampled 100 of them, and verified that they were valid product names and prices. Although we have a fairly large product database, only 11,454 (28.3%) product names could be found there; 29,084 (71.7%) discovered items were actually new product names.

We found that the extracted product information is from 25,865 e-receipts (34% of total) from 76 retailers. On average, we extracted about 1.1 new (product name, price) pairs per e-receipt from those retailers. PIEML did not extract any product information from the remaining 35,170 e-receipts (58%) from 2,551 retailers. Analyzing the e-receipts from those retailers, we found that PIEML did not work on them because their product information either did not have the two assumed formats of LayoutDiscover; or when they did have the formats, the patterns that LayoutDiscover learned were discarded because they extracted too few product names that matched those in the existing database.

Even though PIEML's recall is moderate at 34% of total e-receipts, its precision is high at an estimated 100%. From all the e-receipts in a large email corpus like Hotmail, it will be able to extract a large quantity of useful product names and prices.

Finally, the computational cost of PIEML is low – on a 2GHz, 3GB RAM machine, it took about 3 hours to finish both TemplateMarker and LayoutDiscoverer. Notice that

this offline learning process only needs to be done once. After the templates and layouts for a retailer are identified, extracting product information from the e-receipts sent by the same retailer can be done in real-time.

4.3 Bootstrapping a Supervised Learner

The high-precision predictions of PIEML allow us to use them to learn a classifier in order to improve upon PIEML's recall. We investigated this direction using conditional random fields (CRFs) [15], which is a state-of-the-art sequence prediction model that has been shown to be effective for information extraction [17, 21, 24]. Given a text document represented as a sequence of words, CRFs aim to predict the label of each token, such as whether it is a part of the product name or price, using *features* of each token and of each pair of neighboring tokens.

In order to estimate the recall and precision of this approach, we took the 25,865 e-receipts from which PIEML managed to extract product information as our dataset. PIEML has effectively labeled the tokens in the e-receipts that correspond to the extracted product name and price. All other tokens in the e-receipt are labeled as not belonging to any product name or price.

We used 119 simple features defined on individual tokens, such as whether a token starts with an uppercase letter, whether a token is terminated with a period, etc. We also conjoined the features to create an additional 7021 features. For each pair of neighboring tokens, we had features stating whether they tend to have the same label (e.g., both are product names, or both are not). Note that we only used features capturing characteristics in the neighborhood of each token, and those features did not capture the regularities of LayoutDiscover's assumed formats. We did so because we wished to investigate how well the CRFs would perform on e-receipts without those formats.

We divided the dataset into 5 folds based on the sender address of the e-receipts (each sender address appears in exactly one fold). We performed 5-fold cross validation. In each fold, we sampled 2000 e-receipts from the training set, trained a CRF model on them, and evaluated it on the test set. This setting is close to the real scenario – the statistical model will be applied to e-receipts that are potentially generated by a different, unseen template.

The result of bootstrapping a supervised learner using the output of PIEML is promising. We found that the CRF model had a recall, precision and F_1 score of 0.67, 0.85 and 0.75 respectively (averaged over the 5 folds). This suggests that the product information extracted by PIEML could be effectively used to bootstrap a CRF to extract more product information. Note that the features used by the CRF are rather simple. With more feature engineering, the CRF could conceivably achieve better results.

5. RELATED WORK

As a system that discovers templates and patterns to extract information from machine-generated e-receipts, PIEML can be viewed as a novel *wrapper* induction approach. A wrapper is generally defined as a piece of code that transfer text into some database tuples that represent the target information [14]. Due to the high labor cost of crafting domain-specific wrappers for different targets, researchers have been investigating different ways to automate the process of creating wrappers. For example, in the pioneering work of wrapper induction, Kushmerick et al. defines a special wrapper language and proposes an algorithm to learn the wrappers from human-labeled data [14]. Another example is the STALKER system, which is a hierarchical approach developed by Muslea et al. and can learn wrappers with fewer labeled data [18].

Although the learning approach significantly reduces the load of coding wrappers, preparing labeled data for training becomes the new bottleneck, which inspires a different branch of research – automatically identifying templates and patterns without any labeled data. Similar to our TemplateMarking algorithm, systems of this sort usually operate on the assumption that text generated from the same template appears repeatedly in the same document or in the documents from the same source. For example, in IEPAD, a system developed by Chang and Lui [4] that aims to retrieve information from a Web page, consecutive text tokens separated by HTML tags are first substituted by a special tag. The whole page is then encoded as a binary string to construct a PAT tree to efficiently discover repeated patterns. In contrast, Arasu and Garcia-Molina proposed a more sophisticated method with a formal language that models the page creation process [1]. In this framework, HTML tags do not have to be treated as parts of the templates, but can be included in the target text to be extracted.

Compared to the aforementioned work, PIEML differs in several ways. First, we aim to discover templates and patterns from plain-text e-receipts instead of Web documents, where the semi-structured information provided by HTML tags usually provide more hints of the templates and make the problem easier. Second, we do not require all documents in the input corpus to be generated from the same template. Third, we decouple template marking and pattern discovery explicitly into two stages. The former focuses on finding repeated text across different e-receipts using a novel and principled co-clustering algorithm based on Markov logic; the latter produces regular expressions for extracting target information from non-template lines. This design can be analogous to the approach taken by Patwardhan and Riloff [19]. Because their goal is to extract information from human generated, natural language text, they first train a sentence classifier to filter out potentially uninteresting sentences, and then apply a separate extractor on the remaining ones.

Finally, our pattern discovery method effectively exploits an additional database to provide evidence that a sequence of tokens are indeed a product name. Although the idea of using a database of target instances has been explored before, previous approaches label the corpus by matching the text with the database entries directly for a supervised learner to train an information extraction classifier statistically [2], which is quite different from our approach.

When there is less regularity on how the information is embedded in the text document, statistical learning approaches that train classification models to predict whether a sequence of words is the target information (e.g, the product name) are usually more effective. Although there exist some methods that consider all possible word sequences in the given text as classification candidates (e.g., [23]), most approaches treat this problem as a sequence labeling problem and aim to predict the label of each word or token (e.g., [6, 15, 17]). Example token labels in this framework could be "part of a product name" or "not belonging to any target".

In addition, such sequence prediction approaches typically have an inference procedure (e.g., the Viterbi algorithm) to find the most probable legitimate label prediction during run-time. Compared to the wrapper induction methods, statistical models do not rely on the assumption that text is generated according to some templates and patterns, but consider various features describing the target to make the most probable prediction.

While PIEML works quite effectively when there are enough e-receipts sampled from the same retailer, experimental results in Section 4 have shown that statistical learning methods can be complementary and increase the overall coverage to handle e-receipts from tail retailers better.

6. CONCLUSION AND FUTURE WORK

We proposed PIEML, a novel unsupervised algorithm that clusters together plain-text e-receipts that are generated from the same template, and extracts product information from them. PIEML simultaneously clusters e-receipts and the lines in them, marks the lines that are part of a template in each e-receipt cluster, and extracts product information from the remaining non-template lines. Empirical results on a real-world corpus of e-receipts show the promise of PIEML. The extracted product information has high enough precision to be used as labeled data to train a supervised statistical model, allowing more product information to be extracted.

In the future, we plan to classify emails as e-receipts in a principled manner, learn the templates of e-receipts without assuming their general formats, etc.

Acknowledgements. We thank Ye-Yi Wang for answering our questions on his CRF implementation.

7. REFERENCES

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

[2] S. Canisius and C. Sporleder. Bootstrapping information extraction from field books. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 827–836, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[3] A. Carlson and C. Shafer. Bootstrapping information extraction from semi-structured web pages. In *Proceedings of the Nineteenth European Conference on Machine Learning*, pages 195–210, Antwerp, Belgium. Springer.

[4] C. Chang and S. Lui. IEPAD: Information extraction based on pattern discovery. In *Proceedings of the Tenth International World Wide Web Conference*, pages 681–688, Hong Kong, China, 2001. ACM Press.

[5] M. Chang, W. Yih, and R. McCann. Personalized spam filtering for gray mail. In *CEAS-2008*, 2008.

[6] H. L. Chieu and H. T. Ng. A maximum entropy approach to information extraction from semi-structured and free text. In *AAAI/IAAI*, pages 786–791, 2002.

[7] G. Cormack. TREC 2006 spam track overview. In *Proceedings of TREC-2006*, 2006.

[8] G. Cormack and T. Lynam. TREC 2005 spam track overview. In *Proceedings of TREC-2005*, 2005.

[9] V. Crescenzi, G. Mecca, and P. Merialdo. ROADRUNNER: Towards automatic data extraction from large web sites. In *Proceedings of the Twenty Seventh International Conference on Very Large Data Bases*, pages 109–118, Rome, Italy, 2001. Morgan Kaufmann.

[10] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1987.

[11] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *In ECML*, pages 217–226, 2004.

[12] S. Kok and P. Domingos. Statistical predicate invention. In *Proceedings of the Twenty Fourth International Conference on Machine Learning*, pages 443–440, Corvallis, Oregon, 2007. ACM Press.

[13] S. Kok and P. Domingos. Extracting semantic networks from text via relational clustering. In *Proceedings of the Nineteenth European Conference on Machine Learning*, pages 624–639, Antwerp, Belgium, 2008. Springer.

[14] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 729–737, Providence, RI, 1997. AAAI Press.

[15] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, Williamstown, MA, 2001. Morgan Kaufmann.

[16] L. Liu, C. Pu, and W. Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *Proceedings of the Sixteenth International Conference on Data Engineering*, pages 611–621, San Diego, CA, 2000.

[17] A. McCallum. Efficiently inducing features of conditional random fields. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 403–410, Acapulco, Mexico, 2003. Morgan Kaufmann.

[18] I. Muslea, S. Minton, and C. A. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third International Conference on Autonomous Agents*, pages 190–197, Seattle, WA, 1999. ACM Press.

[19] S. Patwardhan and E. Riloff. Effective information extraction with semantic affinity patterns and relevant regions. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 717–727, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[20] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, CA, 1988.

[21] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *Proceedings of the Twenty Sixth International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 235–242, Toronto,Canada, 2003. ACM Press.

[22] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

[23] D. Roth and W. Yih. Relational learning via propositional algorithms: An information extraction case study. In *IJCAI*, pages 1257–1263, 2001.

[24] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 134–141, Edmonton, Canada, 2003. ACM Press.

[25] P. Viola and M. Narasimhan. Learning to extract information from semi-structured text using a discriminative context free grammar. In *Proceedings of the Twenty Eighth International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 330–337, Salvador, Brazil, 2005. ACM Press.

[26] W. Yih, J. Goodman, and G. Hulten. Learning at low false positive rates. In *CEAS-2006*, 2006.

[27] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proceedings of the Fourteenth International World Wide Web Conference*, pages 76–85, Chiba, Japan, 2005. ACM Press.

APPENDIX

A. DERIVATION OF LOG-POSTERIOR

In the MLN defining the prior component of the posterior probability, there are two rules. The first rule has infinite weight, and it states that each e-receipt or line belongs to exactly one cluster. The second rule has negative weight $-\infty < -\lambda < 0$), and it penalizes the number of cluster combinations. From that MLN, we get

$$\begin{aligned} P(\Gamma) &= \frac{\exp\left(\infty \cdot n_\Gamma - \lambda m_\Gamma\right)}{Z} \\ &= \frac{\exp\left(\infty \cdot n_\Gamma - \lambda m_\Gamma\right)}{\sum_{\Gamma'} \exp\left(\infty \cdot n_{\Gamma'} - \lambda m_{\Gamma'}\right)} \end{aligned} \quad (1)$$

where Z is the partition function; n_Γ and m_Γ are respectively the number of true groundings of the first and second rules for cluster assignment Γ.

We first consider the case where the first rule is violated in Γ, i.e., there is an e-receipt or line that does not belong to exactly one cluster. Note that there is a cluster assignment in which the first rule is not violated, specifically, the one where each e-receipt or line is in its own cluster. Let this cluster assignment be Γ^u. Rewriting Equation 1, we get

$$\begin{aligned} P(\Gamma) = \exp\left(-\lambda m_\Gamma\right) / [\, &\exp\left(\infty \cdot (n_{\Gamma^u} - n_\Gamma) - \lambda m_{\Gamma^u}\right) \\ &+ \sum_{\Gamma' \setminus \Gamma^u} \exp\left(\infty \cdot (n_{\Gamma'} - n_\Gamma) - \lambda m_{\Gamma'}\right)]. \end{aligned} \quad (2)$$

Since $n_\Gamma < n_{\Gamma^u}$, $0 < \lambda < \infty$, and $0 \leq m_{\Gamma^u} < \infty$, $\exp\left(\infty \cdot (n_{\Gamma^u} - n_\Gamma) - \lambda m_{\Gamma^u}\right) = \infty$. Consequently, the denominator of Equation 2 is ∞, and $P(\Gamma) = 0$. Thus when the first rule is violated, the posterior $P(\Gamma|D) = 0$ and $\log P(\Gamma|D) = -\infty$.

Henceforth we consider the case where the first rule is not violated. We divide the numerator and denominator of Equation 1 by $\exp\left(\infty \cdot n_\Gamma\right)$. Let Γ'' be a cluster assignment in the summation of Z. When Γ'' violates the first rule, its contribution to the summation is zero. This is because $n_{\Gamma''} < n_\Gamma$ and $\exp(\infty \cdot (n_{\Gamma''} - n_\Gamma) - \lambda m_{\Gamma''}) = 0$. When Γ'' does not violate the first rule, $n_{\Gamma''} = n_\Gamma$, and $\exp(\infty \cdot (n_{\Gamma''} - n_\Gamma) - \lambda m_{\Gamma''}) = \exp(-\lambda m_{\Gamma''})$. Consequently, we can write Equation 1 as

$$P(\Gamma) = \frac{\exp\left(-\lambda m_\Gamma\right)}{\sum_{\Gamma''} \exp\left(-\lambda m_{\Gamma''}\right)} = \frac{\exp\left(-\lambda m_\Gamma\right)}{Z'} \quad (3)$$

where the summation in the denominator is over cluster assignments that do not violate the first rule.

Taking logs, we get

$$\log P(\Gamma) = -\lambda m_\Gamma + K \quad (4)$$

where $K = -\log(Z')$ is a constant.

Next we derive the likelihood component of the posterior probability. Since each e-receipt or line belongs to exactly one cluster γ_e or γ_l, each ground atom $HasLine(e, l)$ is in exactly one cluster combination (γ_e, γ_l). Let $G_{HasLine(e,l)}$ be a set of all groundings of the atom prediction rules, and the (single) grounding of the default atom prediction rule containing ground atom $HasLine(e, l)$ as their consequents. (A *consequent* and *antecedent* respectively appear on the right and left of the implication symbol $\Rightarrow$.) Suppose the cluster combination (γ_e, γ_l) to which $HasLine(e, l)$ belongs contains at least one true ground atom. Then there is exactly one grounded atom prediction rule in $G_{HasLine(e,l)}$ whose antecedent is true. The antecedents of all other rules in $G_{HasLine(e,l)}$ are false, and the rules are trivially true. Similarly, when cluster combination (γ_e, γ_l) does not contain any true ground atom, there is exactly one grounded *default* atom prediction rule in $G_{HasLine(e,l)}$ whose antecedent is true, and all other rules have false antecedents and are trivially true.

From the MLN defining the likelihood component, we get

$$P(D|\Gamma) = \frac{\exp\left(\sum_{i \in F} \sum_{j \in G_i} w_i g_j(D)\right)}{Z} \quad (5)$$

where Z is the partition function (different from that of Equation 1); F is a set containing all atom prediction rules and the default atom prediction rule; G_i and w_i are respectively the set of groundings and weight of the ith rule in F; and $g_j(D) = 1$ if the jth ground rule in G_i is true and $g_j(D) = 0$ otherwise.

In the numerator of Equation 5, we sum over all grounded rules. We can rewrite the equation by iterating over ground atoms $HasLine(e, l)$, and summing over grounded rules that have $HasLine(e, l)$ as their consequents.

$$P(D|\Gamma) = \frac{\exp\left(\sum_{HasLine(e,l) \in D} \sum_{j \in G_{HasLine(e,l)}} w_j g_j(D)\right)}{Z} \quad (6)$$

where $G_{HasLine(e,l)}$ is a set of all groundings of the atom prediction rules, and the single grounding of the default atom prediction rule containing ground atom $HasLine(e,l)$ as their consequents; and w_j is the weight of the jth rule in $G_{HasLine(e,l)}$,

In $G_{HasLine(e,l)}$, there is exactly one grounded rule whose antecedent is true. All other grounded rules have false antecedents, and are trivially true in all worlds. Such rules cancel themselves out in the numerator and denominator of Equation 6. Hence we only need to sum over grounded rules whose antecedents are true. We can write Equation 6 as

$$P(D|\Gamma) = \frac{\exp\left(\sum_{c\in C}\sum_{j\in F_c} w_c g_j(HasLine_j(e,l))\right)}{Z'} \quad (7)$$

where C is a union of cluster combinations containing at least one true grounding of $HasLine$, and a default cluster combination containing only false groundings of $HasLine$; F_c is a set of grounded rules with cluster combination c in their true antecedents and a grounding of $HasLine$ as their consequents; w_c is the weight of the atom predication rule or default atom predication rule that has c in its antecedent; $HasLine_j(e,l)$ is the ground atom appearing as the consequent of rule j; $g_j(HasLine_j(e,l)) = 1$ if $HasLine_j(e,l)$ is true; $g_j(HasLine_j(e,l)) = 0$ otherwise; and Z' is the partition function.

Because a ground atom $HasLine(e,l)$ is in exactly one cluster combination c, and appears in exactly one grounded rule with c in its the antecedent, we can factorize Z', and write Equation 7 as

$$\begin{aligned}
&P(D|\Gamma) \\
&= \frac{\prod_{c\in C}\prod_{j\in F_c}\exp\left(w_c g_j(HasLine_j(e,l))\right)}{\prod_{c\in C}\prod_{j\in F_c}\sum_{HasLine_j(e,l)\in\{0,1\}}\exp\left(w_c g_j(HasLine_j(e,l))\right)} \\
&= \prod_{c\in C}\prod_{j\in F_c}\frac{\exp\left(w_c g_j(HasLine_j(e,l))\right)}{\sum_{HasLine_j(e,l)\in\{0,1\}}\exp\left(w_c g_j(HasLine_j(e,l))\right)} \\
&= \prod_{c\in C}\prod_{j\in F_c}\frac{\exp\left(w_c g_j(HasLine_j(e,l))\right)}{1+\exp(w_c)} \\
&= \prod_{c\in C}\left(\frac{\exp(w_c)}{1+\exp(w_c)}\right)^{t_c}\left(\frac{1}{1+\exp(w_c)}\right)^{f_c} \quad (8)
\end{aligned}$$

where t_c and f_c are respectively the number of true and false ground $HasLine(e,l)$ atoms in cluster combination c.

By differentiating Equation 8 with respect to w_c, setting the derivative to 0, and solving for w_c, we find that the resulting equation is maximized when $w_c = \log(t_c/f_c)$. Substituting $w_c = \log(t_c/f_c)$ in Equations 8, and taking logs, we get

$$\log P(D|\Gamma) = \sum_{c\in C} t_c \log\left(\frac{t_c}{t_c+f_c}\right) + f_c\log\left(\frac{f_c}{t_c+f_c}\right). \quad (9)$$

Adding smoothing parameters α and β, we get

$$\log P(D|\Gamma) = \sum_{c\in C} t_c \log\left(\frac{t_c+\alpha}{t_c+f_c+\alpha+\beta}\right) + f_c\log\left(\frac{f_c+\beta}{t_c+f_c+\alpha+\beta}\right). \quad (10)$$

(In our experiments, we set $\alpha+\beta = 10$ and $\frac{\alpha}{\alpha+\beta}$ to the fraction of true $HasLine$ groundings.) Separating the default cluster combination c' containing only false groundings of $HasLine$ from the set of cluster combinations C^+ containing at least one true grounding of $HasLine$, we obtain

$$\begin{aligned}
&\log P(D|\Gamma) = \\
&\sum_{c\in C^+} t_c \log\left(\frac{t_c+\alpha}{t_c+f_c+\alpha+\beta}\right) + f_c\log\left(\frac{f_c+\beta}{t_c+f_c+\alpha+\beta}\right) \\
&+ f_{c'}\log\left(\frac{f_{c'}+\beta}{f_{c'}+\alpha+\beta}\right). \quad (11)
\end{aligned}$$

$\log P(\Gamma|D) = \log P(\Gamma) + \log P(D|\Gamma) + K'$ where K' is a constant. Using the values of the prior and likelihood, we get

$$\log(P(\Gamma|D) = \begin{cases} -\infty \text{ if an e-receipt or line is not in exactly one cluster} \\ \sum_{c\in C^+} t_c \log\left(\frac{t_c+\alpha}{t_c+f_c+\alpha+\beta}\right) + f_c\log\left(\frac{f_c+\beta}{t_c+f_c+\alpha+\beta}\right) \\ \quad + f_{c'}\log\left(\frac{f_{c'}+\beta}{f_{c'}+\alpha+\beta}\right) - \lambda m_\Gamma + K'' \text{ otherwise} \end{cases}$$

where $K'' = K + K'$ is a constant. (When comparing candidate cluster assignments to find the one with the best log-posterior, we can ignore K'' because it is a constant.)