A Probabilistic Approach for Automatically Filling Form-Based Web Interfaces

Guilherme A. Toda[1,2] Eli Cortez[1] Altigran S. da Silva[1] Edleno de Moura[1]

[1]Federal University of Amazonas
{gat,eccv,alti,edleno}@dcc.ufam.edu.br

[2]Nhemu Technologies
guilhermetoda@nhemu.com

ABSTRACT

In this paper we present a proposal for the implementation and evaluation of a novel method for automatically using data-rich text for filling form-based input interfaces. Our solution takes a text as input, extracts implicit data values from it and fills appropriate fields. For this task, we rely on knowledge obtained from values of previous submissions for each field, which are freely obtained from the usage of the interfaces. Our approach, called *iForm*, exploits features related to the content and the style of these values, which are combined through a Bayesian framework. Through extensive experimentation, we show that our approach is feasible and effective, and that it works well even when only a few previous submissions to the input interface are available.

1. INTRODUCTION

The web is abundant with applications where casual users are required to enter data to be stored in databases for further processing. The most common solution deployed in these cases is to design form-based interfaces which contain *multiple data input fields*, such as text boxes, radio buttons, pull-down lists, check boxes and other input mechanisms. Unlike typical search forms, these web input forms usually have a larger number of fields.

Although these interfaces are popular and effective, in many cases interfaces that accept *data-rich free text* as input , i.e., documents or text portions that contain implicit data values, would be preferable. Indeed, in many cases the data required to fill the form fields could be taken from text files in which they are already available. For instance, a job applicant may use data taken from a resume text file to fill several fields of forms in many different job search sites. ma The alternative approach we propose in this paper consists of deploying a system that receives a data-rich free text input (e.g., an offer or ad), such as the one illustrated in Figure 1, and recognizes implicit data values occurring in it that can be used to appropriately fill out the fields in a form based interface. For practical purposes, the user could check if the fields were correctly filled by the system and make any necessary corrections before inserting the data into the underlying web database.

2005 Honda new **Accord** Ex, Clean, very **low Mileage**, Maintained By Dealer! Vehicle Located in Stockton, Ca. Ad Id # 28147
This is a brand new car with **automatic transmission**!

Car with Air Conditioning, clock, **Cruise Control**, Digital Info Center, Dual Zone Climate Control, Heated Seats, Leather Steering Wheel, Memory Seat Position, Power Driver's Seat, **Power Steering**, **Power Breaks**, Power Passenger Seat, **Power Windows**, **Cup Holder**, **Rear Air Conditioning**, **Sunroof**, Tilt Steering Wheel, Original Owner, **Alloy Wheels**, Am/Fm, **Cd Changer**, Mp3, Satellite.

Contact Us At XXX-XXXX-XXXX more information Visiti xxx xxx motors

Figure 1: An example of car ad in free text.

The problem we address here is common on popular auction sites, such as *eBay* and *Amazon.com*, which extensively use form-based interfaces to allow users to register offers. In fact, there may be distinct form-based interfaces with specific fields depending on the product being offered. For instance, in the experiments presented in this paper, we consider distinct interfaces corresponding to the categories "cars", "mobile phones" and "books" from a popular Brazilian auction site *TodaOferta.com*, which receives thousands of new offers each day and usually contains about half a million active product offers. Interestingly, as some of these sites (e.g., *eBay* and *TodaOferta*) also allow offers to be entered using generic free text descriptions, users often avoid using form-based interfaces. However, the lack of structured information obtained through form-based interfaces may prevent the proper use of services based on searching, mining, recommendation and integration over offers.

More recently, on-line services such as *Craiglists*, *Googlebase* and *Freebase* were built to allow users to share and exchange data on a variety of domains (e.g., cooking recipes, sports statistics, etc.). These services maintain databases that receive data directly from web users, mainly by means of multi-field input forms[1]. Such services can also benefit from our proposed approach to help users in populating their databases in a simple and intuitive way.

We observe that the problem of filling out web forms presents particular restrictions not usually present in a traditional information extraction (IE), mainly because, in this

[1]Other ways are also available, but they also require structured data to be provided.

setting: (1) Examples of text segments of interest often need to be manually labeled for training. In the form filling problem this training process may not be feasible, since a large diversity of segments is expected. Thus labeling a representative set of text inputs would require a huge effort; (2) It is usually necessary to label many text segment from the input text, even spurious ones, e.g., those that do not contain any information, but that play the role of delimiters around the data of interest. In the form filling problem, such delimiters are not available from previous user inputs of the form-based interface, which contain only field values themselves.

Our approach to solve the form filling problem, which we call *iForm*, consists of automatically selecting segments from the input data-rich text and associating them with the appropriate fields in the form. With this approach, users provide a free text document or portions as input, and iForm automatically extracts values for filling the form relying on information about the previous values assigned to each field. For simplicity, we refer to the user free text document or portions as *input text* from now on. Users may want to verify the form filled by our method, make corrections and then proceed with the request submission. After that, the new assigned values are stored and used as extra evidence when new input texts are provided by users.

As our experiments show, *iForm* works well even when only a few previous submissions are available. This is accomplished by using a model that estimates the probability of each field in the form given the input text based on the values previously used or filling the form, the tokens (words) composing them and also their wording style. An interesting property regarding our strategy for estimating such probabilities is that it allows us to correctly identify segments in the input text that may not correspond to values previously entered in the field, as long as these segments include terms typically found in the values of this field or have a format usually associated to the values previously used in that field.

Consequently, iForm is flexible enough to deal with any text style (e.g. punctuation, font case) or structure (e.g. order of implicit values). In fact, our approach does not rely on features of the text, but rather in features of fields along with their values. We also show in this work how to deal with constraints imposed by a particular interface, such as selecting an item in pull-down lists or selecting options in check boxes. For developers using our approach, no extra effort is required beyond designing form-based interfaces.

Through extensive experimentation on real datasets, we show that our approach is feasible and effective, outperforming the best baseline we found in the literature. Experiments show that *iForm* has a good performance even when only a few previous submissions to the input interface are available.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents an overview of the problem addressed here. Section 4 describes the details of our proposed method. Section 5 reports and analyzes experimental results. Finally, Section 6 presents our final remarks, conclusions and suggestions for future work.

2. RELATED WORK

Numerous approaches have addressed the problem of providing more intuitive alternatives for users accessing web databases than form-based interfaces. Solutions range from keyword-based queries to natural language query specifications. One prominent approach is translating keyword-based queries submitted by users into form-based queries [5] or SQL-like commands [1, 16]. Another well-studied problem is how to provide natural-language interfaces for databases [14] and service requests [2] (e.g. requesting an appointment with a dermatologist). While in these systems the user intentionally types a query composed with a handful of meaningful keywords or small phrases, iForm accepts as input larger portions of text, such as the one in Figure 1. Such portions of text potentially have meaningful data for filling a form, but they were not necessarily composed to this end. Thus, several useless strings are expected to occur. In addition to this, while these systems only support query specifications, our approach is especially suitable for the problem of *populating* web databases which are only accessible through form-based interfaces.

Improving the way users deal with web forms is an important practical problem that has been an active research topic in the recent literature. For instance, while many web browsers provide a list of suggested values for a given field the user is typing in, research initiatives have been carried out to correctly predicting these values based on machine learning techniques [3]. Such techniques apply to fields that are common in many forms (e.g., name, zip code, etc.) used by a user or by a community of users. This problem departs from the problem we address in iForm in two main aspects: (1) in iForm, input comes from a preexisting text and not from a user typing values; (2) iForm aims at automatically filling several fields of forms from a given domain, rather than a few common fields occurring in several forms. Another example, is the USHER system [7], used to automatically adapt the form design according to user experience. As in iForm, this is accomplished by learning from field values previously submitted.

Information Extraction methods can be used to automatically "fill-in" input forms from unstructured data such as Web documents or email. Kristjansson et al [12] proposed an interactive method for filling forms based on CRF [15, 13], which we use here as a baseline and will be referred to as *iCRF*. Their interactive information extraction system assists the user in filling in form fields while giving the user confidence in the integrity of the data. The user is presented with an interactive interface that allows both the rapid verification of automatic field assignments and the correction of errors. In cases where there are multiple errors, their system takes into account user corrections, and immediately propagates these constraints so that other fields are often corrected automatically. They proposed two extensions to allow the interactive filling of forms when using CRF: a constrained Viterbi decoding which finds the optimal field assignments consistent with the fields explicitly specified or corrected by the user; and a mechanism for estimating the confidence of each extracted field, so that low-confidence extractions can be highlighted.

Like in *iForm*, recent work [15, 9] has proposed that features can be learned from previously existing data, coming from databases or reference tables. This reduces the manual training labor. However, differently from *iForm*, in such approaches local context-related features (value ordering and positioning) are considered, since they assume values to appear in contiguous positions according to ordering patterns. Such patterns can be considered as fixed, or can be learned both previously from training instances [15] or on-demand from test instances [9].

In the form filling problem, we consider a population of casual Web users using distinct text portions for filling forms, what makes finding ordering patterns almost impossible. For instance, in auction web sites (e.g., *eBay*), a large number of casual users write product offer descriptions in diverse formats with a few values of interest scattered through the text, with no particular order, and mixed with other non-related strings. Under such conditions relying on local context-related features is unfeasible and hardly effective, given the expected diversity of styles and formats. Thus, we only rely on information about input values entered for each field of the target form on previous submission made by the users. As we show through experiments, this strategy is very effective, given the complexity of the problem.

As detailed through the paper, *iForm* assigns segments to fields based on the similarity between these segments and the set of known values of the fields. Similar strategies have been successfully applied in a number of problems in web data management, and, in particular in instance-based schema matching [18]. In our case, however, we tackle the additional problem of determining which, among all possible segments in text, represent suitable values for the field of a given form.

3. THE FORM FILLING PROBLEM

The problem we face in this work is automatically filling out the fields of a given form-based interface with values extracted from a data-rich free text document, or portions of such documents. In particular, we identify two subproblems: the problems of (a) extracting values from the input text and (b) filling out the fields of the target form using them.

Free text documents are treated as sequences of tokens $t_1, \dots, t_N$, representing individual words or punctuation. The extraction task consists of identifying segments from the free text document, i.e., a sequence of contiguous tokens, which are suitable values for fields in the form. A segment s_{ij} is composed by tokens from $t_i, \dots, t_j$, such that $i \leq j$, $i \geq 1$ and $j \leq N$. A valid set of values extracted from the input text must follow two conditions: (1) only a single segment can be assigned to each field in the form and (2) every extracted segment must be non-overlapping, i.e., there are no extracted segments s_{ab} and s_{cd} for $a < c$ such that $b \geq c$.

Most of the challenge of the form filling problem is related to subproblem (a), since suitable values are scarcely embedded in the text with other non-related strings. Furthermore, no particular format or order can be assumed for these values.

4. THE IFORM APPROACH

The iForm approach for dealing with the form filling problem consists of taking candidate segments from the input text and then estimating the probability of a field given each segment.

Consider an input text I, which is composed of $N > 0$ tokens (words). Let S_{ab} be a segment, i.e., a sequence of tokens in I that includes tokens $t_a, t_{a+1}, \dots, t_{b-1}, t_b$ ($0 < a \leq b \leq N$). We consider S_{ab} as a suitable value for a field f if the probability of the field given this segment is above a threshold ϵ[2]. Considering L as the maximum segment length, there are $N * L - \sum_{i=1}^{L-1} i$ segments in a text with N tokens[3]. As latter detailed, *iForm* deploys a dynamic programing strategy to avoid recomputing the probabilities for all pairs of segments and fields.

[2]In all of our experiments, we performed a previous training and selected $\epsilon = 0.2$.

The main idea behind iForm is to rely on information about previous values used for each field of a form to fill this form when a new text is given as input. We consider two types of features from these values: the values themselves and the tokens composing these values, which we call content related features; and the style (e.g., capitalization, punctuation, etc.), which we call the style related feature. We stress that no features from the input tests are considered. The style feature requires a more detailed explanation.

Let SV_j be the set of previous values entered for a field F_j. We automatically learn a Naive Hidden Markov Model $SM(F_j)$, which we call *Value Style Model*, that captures the wording style of the values in SV_j. This model is similar to the inner HMM used in [4], also used to capture the wording style of sequences.

For this, we first tokenize each value of SV_j on whitespaces. Using a taxonomy proposed in [4], we encode this value as a sequence of masks that represent the styles of characters found, which we name as *symbol masks sequences.* For example, the value "Peanuts inc." of the "Company" field is encoded as "[A-Z][a-z]+ [a-z]+.", i.e., string that starts with an uppercase letter, represented by the symbol mask "[A-Z]", plus a sequence of one or more lowercase letters, symbol mask "[a-z]+", followed by a lowercase word, symbol mask "[a-z]+" that finishes with a dot. This process is repeated for all known values of a given field. Notice that the symbol mask sequences can be considered as a style representation of values entered for each field.

A graph representing a *Value Style Model* $SM(F_j)$ is generated using the encodings of all symbol mask sequences found in values previously entered for field F_j. In $SM(F_j)$, each node represents a symbol mask that occurred in the values of SV_j. An edge, ordered pair $\langle n_x, n_y \rangle$, between nodes n_x and n_y is built if n_x is followed by n_y in the symbols mask sequences that were encoded from SV_j. Thus, we can now refer to each symbol mask sequence as a *path* of $SM(F_j)$. The first node of the path that represents a value in $SM(F_j)$ is marked as an initial node in $SM(F_j)$ and the last node of such path is marked as a final node of $SM(F_j)$. Using the *Maximum Likelihood* approach [4] the *weight* of each edge in $SM(F_j)$ is computed as:

$$w(SM(F_j), n_x, n_y) = \frac{\#\text{ of pairs } \langle n_x, n_y \rangle \text{ in } SM(F_j)}{\#\text{ of pairs } \langle n_x, n_z \rangle, \forall n_z \in SM(F_j)}$$

We then use the paths found in the sequence models of fields as information to compute style related probabilities of a given segment, as explained in the next section.

4.1 Probability of a Field given a Segment

Our approach assigns text segments to fields using three distinct features, each one providing independent evidence about how suitable is each possible assignment. In order to select the final assignments between text segments and fields, we need to combine the results of these three distinct features. We have considered several alternatives for such combination, including the use of machine learning approaches, such as SVM [11], Genetic Programming [10], linear combination of values and the use of a Bayesian Network

[3]In our experiments L is no greater than 10

approach. The use of machine learning is certainly an attractive alternative, but has the disadvantage of requiring a training collection, which would hamper the use of iForm in practical applications. The linear combination approach has provided fairly good results, but the quality of the assignments was a bit worse than the one obtained by the Bayesian Network approach. Further, the Bayesian Network approach has the advantage of providing a quite simple and flexible formalism to model the combination problem. This formalism was also successful when applied in a scenario quite similar to our problem [17]. Thus in this article, we focus only on this latter alternative.

We model the computation of the probability of field f_j given a segment S_{ab} through a Bayesian belief network model similar to the one proposed by Ribeiro-Neto et al [17, 6] for ranking documents in search tasks.

Our Bayesian network provides a graphical formalism for representing the probability model we develop allowing for a better visualization of how the features we consider lead to the final probability of a segment given a field. For the interested reader, an illustration of our network is presented in Figure 4, Appendix A.

Segments and fields of a form are represented in this network by tokens, complete values and styles associated to them. Node S_{ab} on the top represents a segment from the input text. On the bottom, each node F_j models a field of the form being filled.

Nodes V_1 to V_q represent the distinct values previously submitted to the fields, nodes from T_1 to T_w model the tokens that appear in these values, and nodes P_1 to P_r represent each of the distinct symbol mask sequences found when processing previously entered values of all fields. Vectors $\mathbf{v}$, $\mathbf{t}$ and $\mathbf{p}$ are used to refer to any of the possible states of the *root* nodes from $V_1, \ldots, V_q$, $T_1, \ldots, T_w$ and $P_1, \ldots, P_r$, respectively.

Nodes from FV_1 to FV_N model the probability of each field from F_1 to F_N, respectively, given the occurrence of a value in the segment S_{ab} (i.e., the entire string that represents the segment). Analogously, FT_1 to FT_N model the probability of fields given the occurrence of a set of distinct tokens found in the segment, and nodes from FP_1 to FP_N model the probability of fields, given the occurrence of a path from the *Value Style Model* matching the symbol mask sequence of the segment. Information available in these nodes is combined through an *or* operator to compute the final probability of each field given the segment S_{ab}, which are modeled by nodes from F_1 to F_N.

Each node X of the network is associated to a *binary* random variable also denoted by X. For instance, node S_{ab} is associated to the binary random variable S_{ab}. In this notation, it should always be clear whether we are referring to the segment, to the node in the network, or to its associated binary variable. For any variable X in the model, we say that it is 1, denoted by x, to indicate that node X is *on* (active).

4.1.1 Probabilities using Content Related Features

Following the our Bayesian network described above, we take the information provided by the field description to compute the probability $P(ft_j|s_{ab})$, i.e., the probability that the node FT_j is *on* given that the node S_{ab} is *on.* By using the model proposed in [17], it can be stated that:

$$P(ft_j|s_{ab}) = P(ft_j|\mathbf{t})\ P(s_{ab}|\mathbf{t})\ P(\mathbf{t}) \qquad (1)$$

where $\mathbf{t}$ is the state of token root nodes for which the active tokens are exactly those in the segment S_{ab} and $P(s_{ab}|\mathbf{t})$ and $P(\mathbf{t})$ are set to 1. We need now to explain how to compute the value $P(ft_j|\mathbf{t})$, which should be related to the likelihood of each token present in the segment S_{ab} occurring in the field F_j represented by node FT_j. We model such probability as:

$$P(ft_j|\mathbf{t}) = \eta \sum_{\tau \in tokens(\mathbf{t})} \frac{\mathtt{freq}(\tau, F_j)}{\sum\limits_{f \in F} \mathtt{freq}(\tau, f)} \qquad (2)$$

where the function *tokens*$(\mathbf{t})$ returns the set of tokens related to the active nodes in $\mathbf{t}$, i.e., the tokens that occur in the segment S_{ab}. F_j is the field associated to the node FT_j. F is the set of all fields present in the form. Function $\mathtt{freq}(\tau, f)$ gives the frequency of token τ for field f. Thus, we estimate the probability of each token given a field F_j dividing the frequency of τ in the previous values entered by users for F_j by the frequency of τ in all fields of the form. The intuition behind this formulation is that the more concentrated the previous occurrences of a term are in a field, the higher the likelihood of this field being related to the term.

Finally, η is a normalizing constant [17], whose value is set to $1/\max(|S_{ab}|, avg(F_j))$, where $|S_{ab}|$ is the number of words in S_{ab} and $avg(F_j)$ is the average number of words in the values entered as input for the field F_j in previous user interactions with the form. Thus, segments with a length smaller than the average number of tokens of the field are penalized. This ensures that the extracted values will present a length compatible with the typical values in their respective fields.

As it can be seen from Equation 2, the computation of the values of $P(ft_k|s_{ij})$ for every possible segment leads to a redundant computation of several probabilities which can be avoided by using dynamic programming.

Considering Equation 2 without the normalization constant (i.e., only the sum of token probabilities), we can define mp_{ij}, the matrix containing the probability of a field ft_k given segment s_{ij} as follows:

Let $mp_{ij} = P(ft_k|s_{ij})$, the following recurrence can be used to compute this probability:

$$mp_{ij} = \begin{cases} P(ft_k|s_{ij}) & i = j \\ mp_{i(j-1)} + mp_{jj} & i < j \end{cases} \qquad (3)$$

Our dynamic programing algorithm for solving this equation first computes the simplest case, that is, elements mp_{ij} such that $i = j$. The algorithm then computes elements in the first row from left to right, and proceeds to the following rows until all elements in the matrix are defined. This process is repeated for the matrices of each field. Finally, we apply the normalization component of Eq. 2 to each element in every matrix in order to calculate the final probability values.

Following our bayesian network , we also compute the value of $P(fv_j|s_{ab})$ as:

$$P(fv_j|s_{ab}) = P(fv_j|\mathbf{v})\ P(S_{ab}|\mathbf{v})\ P(\mathbf{v}) \qquad (4)$$

where $\mathbf{v}$ is a state where the only active node is the one associated to the value found in S_{ab}, $P(s_{ab}|\mathbf{v})$ and $P(\mathbf{v})$ are set to 1, and $P(fv_j|\mathbf{v})$ is computed analogously as $P(ft_j|\mathbf{t})$, but now comparing the frequencies of values in the fields, instead of comparing frequency of tokens.

4.1.2 *Probability using Style Related Feature*

Finally, we also compute the value of $P(fp_j|s_{ab})$ as:

$$P(fp_j|s_{ab}) = P(fp_j|\mathbf{p})\ P(s_{ab}|\mathbf{p})\ P(\mathbf{p}) \quad (5)$$

where $\mathbf{p}$ is a state where the only active node is the one associated to the mask sequence derived from the segment represented by S_{ab}, which is generated as it was done for deriving symbol mask sequences from each value field. We define the prior probabilities $P(\mathbf{p})$ and $P(s_{ab}|\mathbf{p})$ analogously as $P(\mathbf{t})$ and $P(s_{ab}|\mathbf{t})$, setting them to 1.

We can now define and compute $P(fp_j|\mathbf{p})$ as the result of *Value Style Model* of F_j when processing the path associated to segment S_{ab}:

$$P(fp_j|\mathbf{p}) = \begin{cases} 0 \text{ if } SM(F_j) \text{ does not recognize } path(\mathbf{p}) \\ \dfrac{\sum_{\langle n_x,n_y\rangle \in path(\mathbf{p})} w(SM(F_J), n_x, n_y)}{|path(\mathbf{p})|} \quad \text{otherwise} \end{cases} \quad (6)$$

where $path(\mathbf{p})$ returns the path from the *Value Style Model* $SM(F_j)$ associated to state $\mathbf{p}$, which corresponds to the symbol mask sequence of S_{ab}. The *Value Style Model* of F_j then processes $path(\mathbf{p})$. The result will be zero if $path(\mathbf{p})$ is not recognized by $SM(F_j)$, i.e., if it is not present in $SM(F_j)$ or it does not start with a starting node or does not finish with a finishing node of $SM(F_j)$. Otherwise, $SM(F_j)$ intuitively verifies the likelihood of the sequence following the same wording style of the known values for this field by computing the average weight of edges followed by $path(\mathbf{p})$.

4.1.3 *Putting all Probabilities Together*

The final conditional probability $P(f_j|\mathbf{s_{ab}})$ can now be computed using a disjunctive operator *or* over probabilities derived from each feature. As in [6], a disjunctive operator *or* was used because the two content-related features provide a high confidence individually. This means that when one of these probability values is close to 1, the likelihood of segment S_{ab} being a correct value for F_j is extremely high.

On the other hand, we verified through experiments that style is less precise than the content-related features. Indeed, the style information is helpful when token and value features fail to match some segments to a given field. Because of this, we decided to use the style information as part of a refining process.

Thus, the mapping process, described in the next section, uses these probabilities in two phases, and the style feature is not taken into account in the first phase. Thus, the final probability for the first phase is accomplished by taking only tokens and values into account, as follows:

$$P(f_j|s_{ab}) = 1 - (1 - P(ft_j|s_{ab})) \times (1 - P(fv_j|s_{ab})) \quad (7)$$

If style is taken into account, information from the set of sequence models has to be added to the computation. In this case, the resulting formula is:

$$\begin{aligned} P(f_j|s_{ab}) &= 1 - (1 - P(ft_j|s_{ab})) \times \\ &\quad (1 - P(fv_j|s_{ab})) \times (1 - P(fp_j|s_{ab})) \end{aligned} \quad (8)$$

4.2 Mapping Segments to Fields

Let C_j be the set of segments S_{ab} such that $P(f_j|\mathbf{s_{ab}})$ is above threshold ϵ. We say that C_j is a set of *candidate values* for field F_j.

We aim at finding a *mapping* $\mathcal{M}$ between candidates values and fields in the form-based interface with a maximum aggregate probability, such that (1) only a single segment is assigned to each field and (2) the selected segments are non-overlapping, i.e., there are no segments S_{ab} and S_{cd} for $a < c$ in the mapping such that $b \geq c$. This is accomplished by means of a two-phase procedure as follows.

In the first phase, we begin by computing the candidate values for each field F_j, based only on content-based features, i.e., using Eq. 7. Let $\mathcal{I}$ be a set composed by the union of the sets of candidate values C_j for all fields F_j. We refer to $\mathcal{I}$ as the *initial mapping*, which contains segment-field pairs $\langle S_{ab}, F_j\rangle$. We say that two pairs in $\mathcal{I}$ are in *conflict* if they violate any of the conditions above. Hence, the problem is finding a subset of value-field pairs in $\mathcal{I}$ without conflicts whose aggregate probabilities are maximum.

Finding the optimal solution for this problem requires assessing all possible subsets – an exponential number. In practice, we use a simple greedy heuristic to find an approximate solution. First, we extract the pair with the highest probability from $\mathcal{I}$ and verify whether it presents conflict to any pair in $\mathcal{M}$ or not. If such pair is non-conflicting, we add it into the final mapping. We repeat this process until every pair in $\mathcal{I}$ is extracted. This ends the first phase.

In the second phase, if any field remains not mapped to a segment, we use the probabilities derived from the style-related features to try to find further assignments, using equation Eq. 8 to compute the probability of each field given each segment. We then repeat the mapping process, but now considering only pairs of segments and fields that were not mapped in the first phase.

We adopted the two phase mapping after verifying through experiments that style is less precise than the other two features adopted. On the other hand, it is still interesting to use style information when token and value features fail in matching some segment to a given field. Thus, we decided to use the style information as part of a refining process, which is performed in the second phase of the mapping.

4.3 Filling Form-based interfaces

The last step in our approach consists of using the final mapping $\mathcal{M}$ to fill out the fields of the form-based interface.

In the case of text boxes, we simply enter each mapped text segment as a value into its corresponding field. For check boxes, we set true for fields that were mapped in $\mathcal{M}$ and false for other check boxes. Since extracted values are rarely equal to items in pull down lists, this type of field requires more work as we discuss in the following.

In the case of pull-down lists, we aim at finding an item such that its similarity with the extracted value is maximum.

We measure this similarity by using a "soft" version of the well-known cosine measure, named softTF-IDF [8]. Unlike the traditional cosine measure, softTF-IDF relaxes the requirement that terms must exactly match and yields better results in our problem. The softTF-IDF model also assesses the similarity between terms by using a similarity measure for strings s. In this way, given a value A and a pull-down list item B, we define $close(\theta, A, B)$ as the set of term pairs (a, b), where $a \in A$ and $b \in B$, and such that $s(a, b) > \theta$ and $b = \arg\min_{b' \in B} s(a, b')$; i.e., b is a term in B with the highest similarity to a.

The similarity between a value A and an item B in a pull-down list is defined as follows.

$$soft(A, B) = \frac{\sum_{(a,b) \in close(\theta,A,B)} w(a, A) \cdot w(b, B) \cdot s(a, b)}{\sqrt{\sum_{a \in A} w(a, A)^2} \cdot \sqrt{\sum_{b \in B} w(b, B)^2}}$$

where $w(a, A)$ and $w(b, B)$ are the weights of terms a and b related to the value A and item B, respectively. $w(a, A)$ returns 1 if a occurs in A or 0, otherwise. For computing $w(b, B)$ we consider the inverse frequency of term b in the pull-down list, i.e, $N_L/\texttt{freq}(b, L)$, where N_L is the number of items in the pull-down list L and $\texttt{freq}(b, L)$ is the number of values in L containing term b.

5. EXPERIMENTS

In this section, we report the results of experiments we have conducted with *iForm* on tasks of automatically filling form-based web interfaces. In all experiments performed here we simulate a real form-based web interface where each data-rich free text document is submitted at a time. Users manually verify its results and, if needed, correct minor errors. After that interaction, the submission will be completed and new added values will be considered when processing new submissions from this point on. Notice that we evaluate the system according to the errors produced on each iteration. In all cases there is no intersection between the sets of test submissions and the set of previously submitted documents. The details on all datasets used and their sources are presented in Appendix C.

To evaluate the results of our experiments we have used the well-known metrics precision, recall and f-measure. The detailed definition of these metrics are presented in the Appendix B.

Prior to these experiments, we performed an evaluation of the sensibility of *iForm* with respect to the threshold ϵ, which is presented in Appendix D. Based on this evaluation, we use $\epsilon = 0.2$ in all the experiments here presented.

In the first experiment, we tested our method with multi-typed web forms for submissions of *Short Movie Reviews*, *Car offers*, *Cellphone offers* and *Book offers*. Next, we evaluated, in turn, how the number and the coverage of the previously submissions impacts on the performance of *iForm*.

Finally, experiments using a *Jobs postings* dataset were conducted for comparing *iForm* with a solution previously proposed to interactively fill forms [12], which is referred to as *iCRF* in our experiments. As it was already mentioned, *iCRF* is a method for interactive form filling based on CRF [15, 13], the current state of the art in information extraction.

5.1 Experiments with multi-typed web forms

To evaluate *iForm* within typical different form-based interfaces from distinct websites, we tested our approach with submissions from *Movies Reviews*, *Car offers*, *Cellphone offers* and *Book Offers*. A detailed description of the data sources we have used in this experiment is presented in Appendix C.

To save space, we grouped the results by the type of each field, i.e., text box, check box or drop-down list, according to their occurrence in each web form. The results are presented in Table 1 by means of field-level and submission-level precision, recall and F-measure.

As we can notice, *iForm* achieved high quality results in all datasets. In the case of car offers, as shown in Table 1 (Cars) the quality of the form filling task was almost the same for the text box fields and the check box fields.

Much better results were obtained for the case of cellphone offers, in which the F-measure average reached above 0.90, as shown in Table 1. As a consequence, submission-level f-measure result for this dataset was 0.95, which means that on average, more than 90% of each submission was correctly entered in the web form interface.

Domain	Type of Field	# Fields	P	R	F
Movies	Text Box	4	0.74	0.69	0.71
	Submission		**0.73**	**0.67**	**0.69**
Cars	Text Box	5	0.78	0.73	0.76
	Check Box	30	0.79	0.79	0.79
	Average		*0.79*	*0.78*	*0.79*
	Submission		**0.77**	**0.73**	**0.75**
Cellphones	Text Box	2	0.89	0.69	0.78
	Check Box	35	0.94	0.94	0.94
	Average		*0.94*	*0.93*	*0.93*
	Submission		**0.96**	**0.94**	**0.95**
Books 1	Text Box	4	0.88	0.67	0.76
	Drop Down	1	0.96	0.96	0.96
	Average		*0.90*	*0.73*	*0.80*
	Submission		**0.89**	**0.67**	**0.76**
Books 2	Text Box	4	0.72	0.54	0.62
	Submission		**0.74**	**0.55**	**0.63**
Books 3	Text Box	2	0.73	0.55	0.63
	Submission		**0.70**	**0.56**	**0.62**
Books 4	Text Box	3	0.85	0.56	0.68
	Submission		**0.75**	**0.55**	**0.63**

Table 1: Results for multi-typed web forms.

A detailed inspection on the offers entered by users in this interface, revealed that the values available on these offers are usually more uniform than the values of car offers and movie reviews. This explains the excellent results obtained by *iForm* and corroborates our claims regarding the frequent reuse of data-rich texts for providing data to fill form-based interfaces on the web.

In the case of the movie dataset the inspection of the text inputs entered (see Appendix C) revealed a large degree of ambiguity, since it is very common, for instance, to have actors that are directors and directors that are also actors. As well as this, movie titles contain ordinary words that appear within reviews not necessarily composing the title (e.g., "Bad Boys") and each review itself sometimes presents more than one movie title. In addition, names and titles that are entirely composed by terms not known from previous submissions frequently appear. In such cases, the style features play an important role. All these shortcomings make this dataset a real challenge. Similar difficulties were found in *Books* datasets. Despite this, *iForm* presented good results. As shown in Table 1, precision levels are above 0.7 in all cases, and submission-level f-measure results for these datasets are above 0.6.

5.2 Number of previous submissions

In this experiment we verify how the performance of our method behaves when the number of previous submissions varies. The result of this experiment is presented in Figure 2, in which for each dataset, we used an increasing number of submissions, from 500 to 10000, and calculated the average submission-level f-measure resulting from running the form filling process over each collection.

As it is shown in Figure 2, for the Movies and Books 1 datasets, the quality achieved by *iForm* increases proportionally with the number of previous submissions[4].

In the cases of the Cars and Cellphones, it is important to notice that F-measure values stabilize at around 3000 previous submissions and remain the same until 10000 submissions. This shows that our method does not require a

[4]The same behavior was observed for the other Books datasets. For saving space, we present their results in Appendix F.

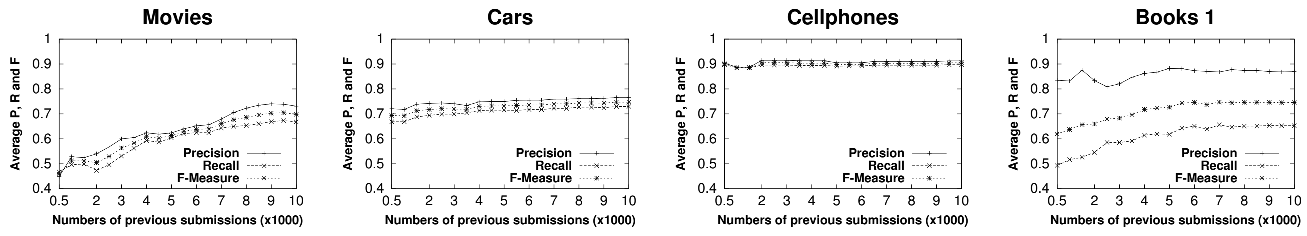


Figure 2: Behavior of the form filling quality with the increasing of the previous submissions.

large number of previous submissions to reach a good quality of results. Besides, even starting with a small number of submissions, *iForm* is able to help decrease the human effort in the form filling task. Notice that the expected volume of previous submissions in the application scenarios which motivated our work, i.e, sites such as *eBay* and *TodaOferta*, is far higher than the number of previous submissions we simulated in this experiment.

5.3 Content Overlap

In this experiment we aim at studying how much *iForm* depends on the overlap between the contents of the text inputs and the contents of the previous submissions, i. e., the known values submitted to each field.

In our solution, we can characterize three different forms of content overlap: (1) *Value Overlap*: the overlap between the set of complete values found in a given input text and the set of previously known values; (2) *Term Overlap*: the overlap between **all** terms on the input text and all terms composing the previously know values; (3) *Term-Value Overlap*: the overlap between the terms in the input text that compose values to be extracted and all terms composing the previously known values.

To exemplify, consider the input text $I = \{$"Brand New Honda Accord Hybrid"$\}$, from which the values "Honda" and "Accord Hybrid" are to be extracted for fields *Make* and *Model*, respectively. Suppose that the following values are known for fields *Make* and *Model*: *Make*= {"Honda", "Mercedes"} and *Model*= {"City","Civic Hybrid", "A310"}.

In this example, for input text I: (a) the value overlap is 1/2, since from the two values to be extracted only one is known; (b) the term overlap is 2/5, since from the five terms in the input text, only 2 are available in the known values; (c) the term-value overlap is 2/3, since from the three terms composing values to be extracted from I, only two are previously known.

In Figure 3 we present the quality results of experiment described in Section 5.1 for the *Books1* dataset presented for different ranges of overlap, considering the three forms of overlap described above.

Figure 3(a) shows that for most of the inputs (36 out of 50) the value overlap is not greater than 50%, and, despite that, the quality of the results in terms of precision, recall and f-measure is close to the quality obtained with a larger value overlap, 76% to 100%, observed on 3 inputs. This is accordance with the results presented in Figure 3(b), since most of the inputs have most of terms present in the previous submissions. This characteristic is exploit by our model through Eq. 1, resulting in the good results obtained for this dataset. Moreover, in this dataset, as shown in Figure 3(c), the useful terms are well distributed among the values to be extracted from the input text.

These results show an important property in our approach: *iForm* does not rely on a high coverage of values in the previous submissions, as long as these submissions are representative from the domain.

This same analysis is present in Appendix E for datasets *Movies*, *Cars* and *Cellphones*.

5.4 Comparison with *iCRF*

Finally, we compare *iForm* and the interactive method proposed by Kristjansson et al [12], which we name here as *iCRF*, for the task of extracting segments from text inputs and filling a form. We took from the RISE Jobs collection a subset of 100 job postings already containing labels manually assigned for the segments to be extracted. These job postings form an adequate training set for *iCRF*, since this method requires examples of values to be extracted to appear within the context they occur. Thus, we could not use the remaining 450 job postings from the collection, for which extracted values are provided separately from the postings in which they occur. From the same set of 100 documents, we took the labeled segments to simulate submissions to the form-based interfaces for iForm. Notice that, unlike iCRF, *iForm* does not require the annotated input for training.

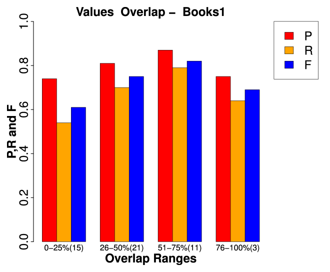
Next, we tested both approaches using a distinct set of 50 documents, whose extraction outcome was available from RISE, allowing us to verify the results. These results are reported in Table 2 in terms of field-level F-measure.

For the experiment with iCRF, we used a publicly available implementation of CRF by Sunita Sarawagi and deployed the same features described in [13]. Overall, these are standard features available on the public CRF implementation, e.g., dictionary features, word score functions and transition features. Further, we consider that the forms are filled in an interactive process, with the previously filled forms being corrected by a human and then being incorporated to the training set.

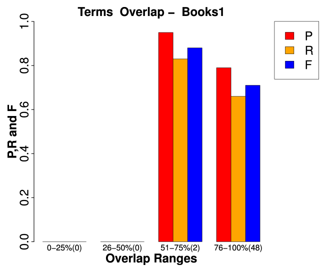
Field	iForm	iCRF
Application	**0.82**	0.37
Area	0.18	**0.23**
City	**0.70**	0.65
Company	**0.41**	0.17
Country	0.77	**0.87**
Desired Degree	**0.57**	0.37
Language	**0.84**	0.69
Platform	**0.47**	0.38
Recruiter	**0.44**	0.22
Req. Degree	0.31	**0.59**
Salary	0.22	**0.25**
State	**0.85**	0.81
Title	**0.72**	0.49

Table 2: Field-level f-measure

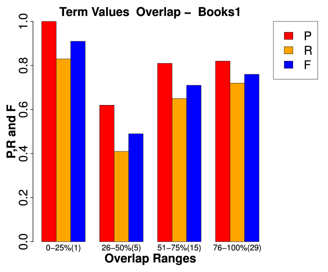
According to the results presented in Table 2, *iForm* had superior F-measure levels in nine fields, while iCRF had significant superior F-measure levels in four fields only, as indicated by boldface numbers. The lower quality obtained by iCRF is explained by the fact that segments to be extracted from typical free text inputs, such as jobs postings, may not appear in a regular context, which is an important requirement for CRF-based methods. For the case of iForm, this



(a)



(b)



(c)

Figure 3: Form filling quality on *Books1* dataset with different overlap ranges.

context is less important, since it relies on features related to the *fields* instead of relying on features from the *input texts*. In addition, iForm was designed to conveniently exploit these field-related features from previous submissions. If we consider each submission as whole, i.e., the submission-level quality (see Appendix B), iCRF and iForm achieved, respectively, 0.46 and 0.59. Recall that, as we have seen, for one to apply iCRF to this problem, labor-intensive preparation of training data from a representative sample of text inputs is required.

6. CONCLUSION AND FUTURE WORK

Our experiments demonstrate that our approach is able to properly deal with different types of input fields, such as text boxes, pull-down lists and check boxes and has proved to be useful as an alternative to automatically filling forms. There are two main reasons for this conclusion: (1) any form-based interface can be boosted with iForm; (2) users can easily verify the correctness of iForm's outcome (and fix possible mistakes) through a form-based interface filled by our approach.

The iForm approach can be deployed in a number of interesting applications that require repetitively entering the same data available in a text document into several distinct form-based interfaces of the same domain. For instance, the same car ad presented in Figure 1 could be used for entering data on form interfaces of several car advertising sites.

For future work, we will investigate how to help users to find an appropriate form-based interface given an input data-rich free text document. This includes finding out, among a possibly large set of forms, which of them can properly answer a given user request. This problem requires an efficient yet scalable and flexible solution for the web scenario. In this case, it would be interesting to extend our framework to deal with free text queries as well.

iForm assumes that references to values in a data-rich text are positive indications for using it to fill a form field. This was corroborated by an inspection we carried out looking for values with negative references (e.g. "Model is not Honda") on our datasets. We discovered that less than 3.0% of value references in the *Cellphones* dataset were negative. In the *Cars* dataset this percentage was lower than 0.4%. Similar percentages were found in the other datasets. Notice that a user can easily correct the system in these rare cases. Nonetheless we plan to better investigate this in future work.

Acknowledgements

This work was partially supported by the following projects funded by CNPq: InfoWeb, INCTWeb, SIRIAA and MinGroup. The authors would like to acknowledge CNPq for fellowship grants to Edleno S. de Moura and Altigran S. da Silva; and CAPES for scholarships to Guilherme A. Toda and Eli Cortez. This research was also sponsored by the UOL Bolsa Pesquisa Program (www.uol.com.br) (num. 20090213165000).

7. REFERENCES

[1] S. Agrawal et. al. DBXplorer: A system for keyword-based search over relational databases. In *Proc. of ICDE*, pages 5–16, 2002.

[2] M. Al-Muhammed and D. Embley. Ontology-Based Constraint Recognition for Free-Form Service Requests. In *Proc. of ICDE*, pages 366–375, 2007.

[3] A. Ali and C. Meek. Predictive models of form filling. Microsoft Research, Tech. Rep. MSR-TR-2009-1, 2009.

[4] V. R. Borkar et. al.. Automatic segmentation of text into structured records. In *Proc. of ACM SIGMOD*, pages 175–186, 2001.

[5] P. Calado et. al. Searching web databases by structuring keyword-based queries. In *Proc. of ACM CIKM*, pages 26–33, 2002.

[6] P. Calado et. al. Local versus global link information in the web. *ACM TOIS*, 21(1):42–63, 2003.

[7] K. Chen et. al. Usher: Improving data quality with dynamic forms. In *Proc. of ICDE*, page 321–332, 2010.

[8] W. Cohen et. al. A comparison of string distance metrics for name-matching tasks. In *Proc. of IIWeb*, pages 73–78, 2003.

[9] E. Cortez et. al. ONDUX: On-Demand Unsupervised Learning for Information Extraction. In *Proc. ACM SIGMOD*, pages 807–818, 2010.

[10] W. Fan et. al. Discovery of context-specific ranking functions for effective information retrieval using genetic programming. *IEEE Transactions on Knowledge and Data Engineering*, 16(4):523–527, 2004.

[11] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proc. of ECML*, pages 137–142, 1998.

[12] T. Kristjansson et. al. Interactive information extraction with constrained conditional random fields. In *Proc. of AAAI Intl. Conf.*, pages 412–418, 2004.

[13] J. D. Lafferty et. al. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*, pages 282–289, 2001.

[14] Y. Li et. al. Nalix: an interactive natural language interface for querying xml. In *Proc. of ACM SIGMOD*, pages 900–902, 2005.

[15] I. R. Mansuri and S. Sarawagi. Integrating unstructured data into relational databases. In *Proc. of ICDE*, page 29, 2006.

[16] F. Mesquita et. al. Labrador: Efficiently publishing relational databases on the web by using keyword-based query interfaces. *Inf. Process. Manage.*, 43(4):983–1004, 2007.

[17] B. Ribeiro-Neto and R. Muntz. A belief network model for IR. In *Proc. of ACM SIGIR*, pages 253–260, 1996.

[18] J. Wang et. al. Instance-based schema matching for web databases by domain-specific query probing. In *Proc. of VLDB Conference*, pages 408–419, 2004.

APPENDIX

A. BAYESIAN NETWORK ILLUSTRATION

In Figure 4 we illustrate the Bayesian Network that was deployed in the *iForm* method.

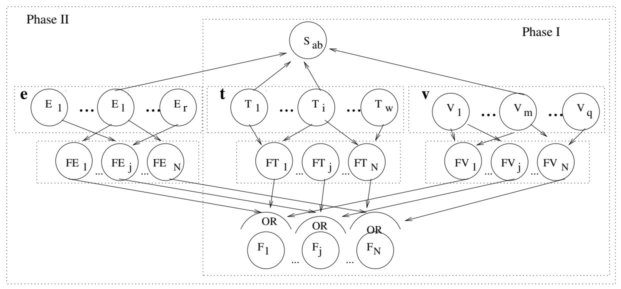

Phase II
Phase I
S_{ab}
e
t
v
E_1 ... E_l ... E_r
T_1 ... T_i ... T_w
V_1 ... V_m ... V_q
FE_1 FE_j FE_N
FT_1 FT_j FT_N
FV_1 FV_j FV_N
OR
OR
OR
F_1 ... F_j ... F_N

Figure 4: Belief network for a segment of text S_{ab}, related to tokens T_1 and T_i, complete attribute value V_m and path P_l derived from its style.

B. METRICS FOR THE EXPERIMENTAL EVALUATION

To evaluate the results of our experiments we have used the well-known metrics precision, recall and f-measure. We apply these metrics to evaluate the quality of filling a single field and a whole form submission.

In the case of text boxes, we calculate precision, recall and f-measure at field level as follows. Let A_i be the set of all tokens (words) from the input text that *should* be used for filling a given field i in the form. Let S_i be the set of all tokens from the input text that were used for filling in this field i by the automatic filling method. We define precision (P_i), recall (R_i) and F-measure (F_i) as: $P_i = \frac{|A_i \cap S_i|}{|S_i|}$, $R_i = \frac{|A_i \cap S_i|}{|A_i|}$ and $F_i = \frac{2(R_i.P_i)}{(R_i+P_i)}$, respectively.

For pull-down lists, set A_i contains the item in the list of field i that *should* be chosen and set S_i contains the items that were chosen for field i. For check boxes, A_i contains the *correct* boolean value for field i and S_i contains the boolean value that was set for field i.

Submission-level precision (recall and f-measure), i.e., the quality of a whole submission, is calculated as the average of the values of each field used in this submission, observing that there are submissions in which not all fields are used.

C. DATA SOURCES USED FOR THE EXPERIMENTS

Table 3 presents in detail each dataset used in our experimental evaluation. The column "Test Data" shows the number of input texts submitted to the form-based interface. The column "Previous Data" refers to the number of previous submissions that were performed prior to the test.

The *Jobs* dataset was obtained from RISE (Repository of on-line Information Sources used in information Extraction tasks) . The test set consists of 50 postings and the previous data consists of 100 postings previously annotated, as it is required for the experimental comparison with iCRF (Section 5.4). For the datasets *Cars* and *Cellphones* multi-field web form interfaces and input data-rich text documents were both taken from *TodaOferta.com* auction website. Similarly to *Cars* and *Cellphones*, for the *Books* datasets, we took input data-rich text documents from *TodaOferta.com*, but, in order to evaluate how good *iForm* adapts to form variations, we have taken the multi-field web form interfaces from distinct websites, *TodaOferta.com*, *IngentaConnect.com*, *Oupress.com* and *Netlibrary.com*, composing datasets *Books* 1 to 4, respectively. For the test in our experiments, we use real offers submitted to *TodaOferta.com* and automatically fill the corresponding form. In the case of *Movie Reviews*, we have built a web form and have taken real short movie reviews collected from IMDb[5] (Previous Submissions), and from Wikipedia and Freebase (Test Submissions).

[4]`http://www.isi.edu/integration/RISE/`

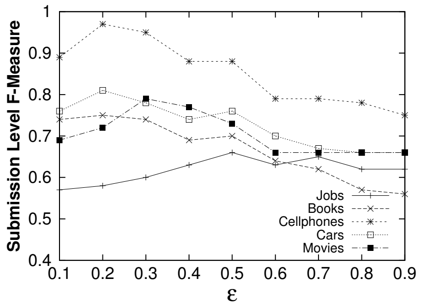


Figure 5: Results obtained when varying ϵ.

D. VARYING ϵ

One important question in our method is to determine the value of the threshold ϵ. Recall from Section 4 that we consider a segment S_{ab} as a suitable value for a field f if the probability of the field given this segment is above the value of this threshold. To study this parameter, we randomly selected 25 documents of each dataset, submitted them to *iForm* varying the parameter ϵ from 0.1 to 0.9. The results of the averaged submission-level f-measure achieved are shown in Figure 5, where in the case of *Books* datasets, the curve in the graph corresponds to the average results for *Books* 1 to 4. We can see that the results may vary according to the domain, which suggests a training adjusting step could be useful to produce optimized results. Notice however, that quite good results were achieved when using $\epsilon = 0.2$ in the samples submitted. For the *Jobs* dataset, the best form-filling result was obtained with $\epsilon = 0.5$. This can be explained by the small number of documents that compose the previous submissions, that requires a more restricted threshold. For all the experiments in this paper, we set the value 0.2 for ϵ, including experiments with the *Jobs* dataset. We suggest the possibility of introducing a training step for future work.

E. CONTENT OVERLAP FOR DATASETS MOVIES, CARS AND CELLPHONES

In this appendix we present, for the other datasets we have used, the same overlap analysis we discussed in Section 5.3. Figure 6, presents the results of this analysis. Here, we observed in the *Movies* datasets trends similar to the ones in *Books1*. For the case of datasets *Cars* and *Cellphones*, notice that the term overlap is quite low in all input texts. This is due to a large number of useless terms typically available on

[5]http://www.imdb.com

Datasets	Test Data	Previous Data	Source – Test Data	Source – Previous Data
Jobs	50	100	RISE	RISE
Movies	50	10000	IMDb	Freebase and Wikipedia
Cars	50	10000	TodaOferta.com	TodaOferta.com
Cellphones	50	10000	TodaOferta.com	TodaOferta.com
Books 1 to 4	50	10000	Submarino.com	TodaOferta.com, IngentaConnect.com, Oupress.com, Netlibrary.com

Table 3: Features of each collection used in the experimental evaluation

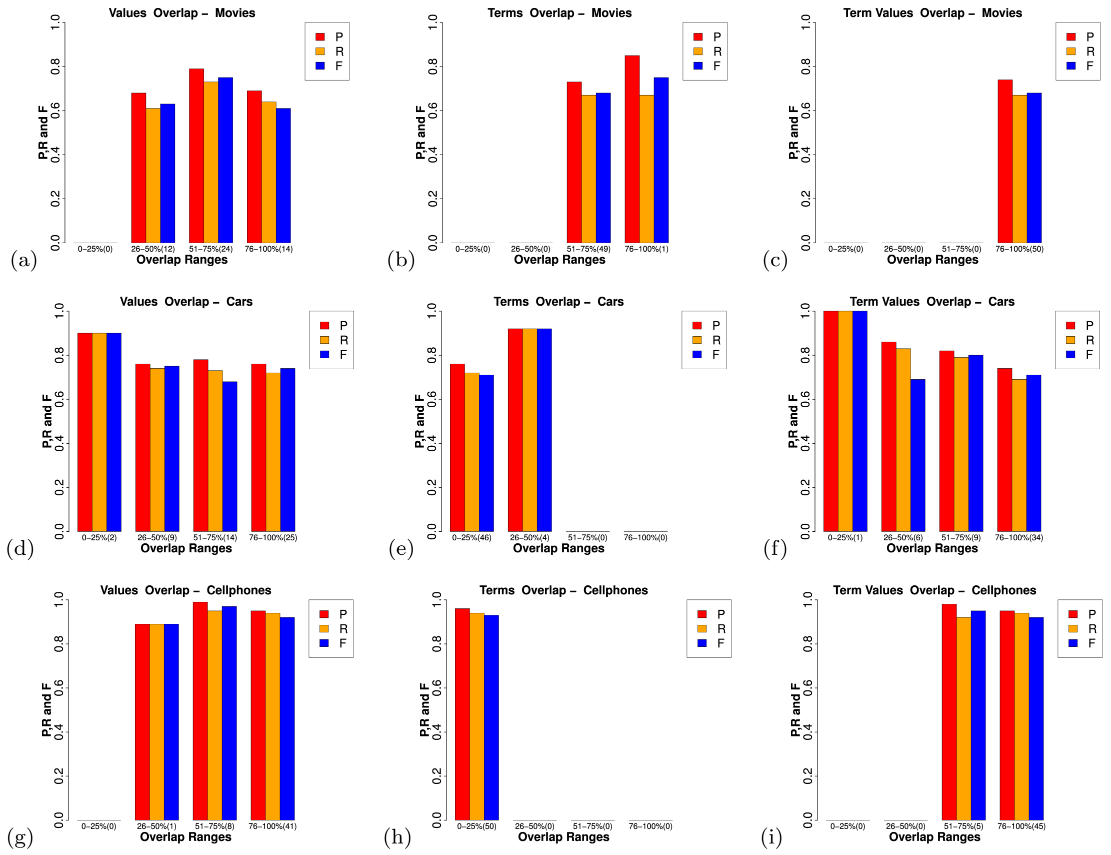


Figure 6: Form filling quality on *Cars*, *Cellphones* and *Movies* datasets with different overlap ranges.

such input text taken as whole. In these cases, however, useful terms appear within values to be extracted from these input texts, yielding to the good quality results achieved. This corroborates our claims regarding the independence of our approach from large coverage of possible values on previous submissions.

F. BOOKS DATASETS – ADDITIONAL RESULTS

In addition to the experiments presented in Section 5.2, where we evaluate the behavior of *iForm* when the number of previous submissions increases, Figure 7 presents more results obtained from different *Books* datasets.

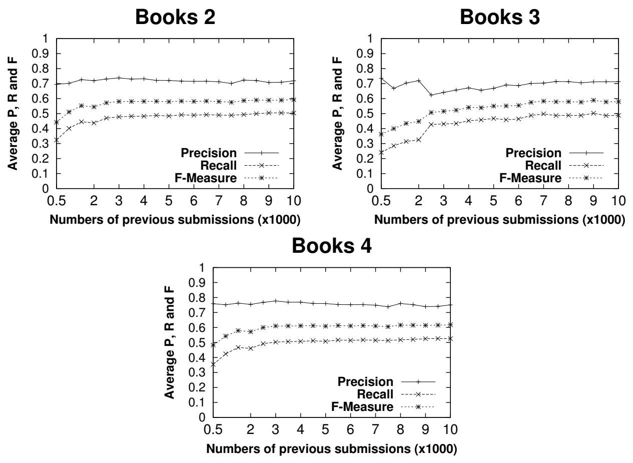


Figure 7: Behavior of the form filling quality with the increasing of the previous submissions with different Book forms.