University of Massachusetts Amherst

From the SelectedWorks of Andrew McCallum

2011

Selecting Actions for Resource-bounded Information Extraction using Reinforcement Learning

Andrew McCallum, *University of Massachusetts - Amherst*

Available at: https://works.bepress.com/andrew_mccallum/68/

Selecting Actions for Resource-bounded Information Extraction using Reinforcement Learning

Pallika Kanani
University of Massachusetts, Amherst
Amherst, MA, USA
pallika@cs.umass.edu

Andrew McCallum
University of Massachusetts, Amherst
Amherst, MA, USA
mccallum@cs.umass.edu

ABSTRACT

Given a database with missing or uncertain information, our goal is to extract specific information from a large corpus such as the Web under limited resources. We formulate the information gathering task as a series of alternative, resource-consuming actions to choose from and use Reinforcement Learning to select the best action to perform at each time step. We use temporal difference Q-learning method to train the function that selects these actions, and compare it to an online, error-driven algorithm called SampleRank. We present a system that finds information such as email, job title and department affiliation for the faculty at our university, and show that the learning-based approach accomplishes this task efficiently under a limited action budget. Applying our method to the task of filling missing values in a large scale database with millions of rows and a large number of columns can help obtain the required information from the Web efficiently, and lead to reduced resource consumption.

General Terms

Web Mining

Keywords

Resource-bounded Information Extraction, Active Information Acquisition, Reinforcement Learning, Missing Data

1. INTRODUCTION

Resource-bounded Information Extraction (RBIE) is the process of searching for and extracting specific pieces of information from an external information source under a limited budget of resources, such as computational time, network bandwidth, and storage space [4]. The problem of missing information in databases is ubiquitous. However, in many cases, this information is available on some external source, such as the Web. In order to fill in the missing slots, we need a mechanism to automatically extract the required information. In such settings, it is undesirable, infeasible, or even computationally intractable to use traditional information extraction pipelines on the entirety of a vast, external, unstructured or semistructured corpus for obtaining relatively small amount of information. Under the RBIE framework, we obtain the required information by issuing appropriate queries to the external source, such as a web search API, downloading only a small fraction of documents from the search results, and processing even fewer of them to extract the specified field.

Consider a real world example. We are building a database of all faculty at a university as shown in Table 1. We have the names, but some of the other information such as contact details, job titles and department affiliations are missing. Surprisingly, in some cases, even the administration may not have such a comprehensive, university-wide database. This may be due to the lack of data exchange, joint appointments across departments, changing contact details, etc. Building such a database would be incredibly useful, since it can help maintain up-to-date records of the faculty, foster collaboration across departments, and so on. A large portion of this information exists on the Web, but it may not always be found on faculty home pages. Lecturers and faculty in some of the departments do not have home pages, and their information could be scattered around the Web. Finding this information can be challenging, since it is hardly available in a uniform, structured manner. There are other problems like name ambiguities and incorrect or incomplete data.

One way to obtain this information is by crawling all the websites under the university domain. This, by itself is a resource-intensive task, since most university websites are large and complex, and we would need to use a lot of computational power to crawl and download the pages, along with the corresponding network bandwidth, and disk space for storing them. We would also lose out on all the information that is scattered on the Web, outside the university domain. Can we accomplish the same task using a much smaller fraction of these resources?

We know that the information missing in the database is available on some relatively small number of pages on the Web. We need to run some extraction algorithms on those pages in order to obtain the required information. But before we can run extraction, we need to download them to our computing infrastructure, and before we can download them, we need to know where they are located on the Web. A search engine API, such as Google can help us retrieve

Faculty Name	Phone	Email	Job Title	Department Name
Andrew McCallum	(413) 545-1323	mccallum@cs.umass.edu	Professor	Computer Science
Jerrold S. Levinsky	?	?	Lecturer	Legal Studies
Edward G. Voigtman	?	?	?	?
Robert W. Paynter	?	?	?	Anthropology

Table 1: Example Database of University Faculty

these web pages. We first formulate queries driven by information that is already available in the database, issue them to the search interface, obtain the location of the web pages, and download them. Then we can run the necessary algorithms to extract information, and use it to fill missing entries in the database. This process is much more efficient, and would use relatively smaller amounts of resources.

The problem of Resource-bounded Information Extraction (RBIE) was first introduced in our previous work [4]. Queries are formed by combining existing, relevant information in the database with user defined keywords. All such queries are issued to the search API, and all of the result documents are downloaded. The resource savings come from selecting a subset of the web documents to process by exploiting the network structure in the data. In general, we may need multiple queries to obtain information about a single entry in the database, and some queries work better than others. In our university faculty example, we may form different queries with keywords such as "curriculum vitae" or "home page", and it may be that one of them is always more successful than the other in finding the information we need. In some cases, the information in different fields may be interdependent, and finding one before another may be more efficient. In order to make the best use of available resources, we need to issue the most effective queries first.

In most scenarios, one only need process a subset of the documents returned by the queries. We need to know which of the search results are most likely to contain the information we are looking for. Information returned in the search result snippet can be exploited to decide if a web page is worth downloading. Similarly, some preliminary observation of the downloaded document can be useful to decide if it is worth passing through an expensive extraction pipeline. Hence, instead of viewing RBIE as selecting a subset of documents to process, [3] cast the information-gathering task as a series of resource-consuming actions, along with a mechanism to select the best action to perform at each time step.

In this paper, we formulate the RBIE problem formally as a Markov Decision Process (MDP), and propose the use of reinforcement learning techniques for solving it. The *State* of this MDP is the state of the database at each time step, and *action* is any act that leads to obtaining the required information, such that performing the action in one state leads to a different state. RBIE process is then finding the optimal policy in this MDP, so as to obtain most amount of information with the given budget of actions, since we assume that actions consume resources. In RBIE from the Web context, actions are - *query*, which is issuing a query to a search API, *download*, which is downloading a web document, and *extract*, which runs an actual extraction algorithm on a document. By formulating RBIE for the Web as an MDP, we can start exploring the rich methods of optimal action selection offered by Reinforcement Learning.

This work builds significantly on [3], which uses an online, error driven algorithm, called SampleRank to learn a value function. In contrast, we use Temporal Difference Q-learning, which takes into account delayed reward. In the RBIE for the web setting, query actions might not lead to immediate reward, but they are necessary to perform before download and extract actions, which may lead to positive rewards. As opposed to the URL classification in the previous work, our example task requires actual text extraction, making it more important to model delayed reward. Since both SampleRank and Q-learning are novel approaches for the RBIE framework, in this paper, we compare their relative performance on the task of finding emails, job titles and department affiliation for faculty in our university.

In general, we can use any model of choice for information extraction in our framework that can extract the required information from a web page, and provide a confidence score for the extracted value. This score can be used to choose the best among the potential candidate values, and to determine whether or not an existing entry in the database should be updated by the newly extracted value. We present a simple, but novel information extraction method that can easily scale to large problem domains. The basic idea is to generate a list of potential candidate values from the web page, and using a binary classifier, such as Maximum Entropy, to classify them as being correct values or not, by observing features of the context in which they are found. The candidate with the maximum probability of being correct is used to fill the entry in the database.

Our experiments show that the Q-learning strategy performs better than a random action selection strategy, as well as SampleRank based approach to learning a value function. Given the large number of actions to choose from at each time step, and the size of the corresponding state space, the policy learned is impressive, even though it is not an 'optimal' one. The Q-learning based approach is able to obtain 90.2% of the final F1, by only using 28.5% of the total actions, demonstrating the effectiveness of our method.

2. RESOURCE-BOUNDED INFORMATION EXTRACTION

2.1 General Problem Definition

The general problem of Resource-bounded Information Extraction (RBIE) was first defined in our previous work [3]. We are given a database with missing values in some of the entries, and a set of possible actions to help acquire that information from an external source, such as the Web. Select the best action from the set of alternative actions available

at each time point, so as to acquire most information with least number of actions.

2.2 Resource-bounded Information Extraction From the Web

For RBIE from the Web, we consider three different types of actions - *query*, *download*, and *extract*. A *query* action consists of issuing a single query to a web search API and obtaining a set of search results. In order to form the query, we need to use some existing information from an input record in the database and a set of keywords. A *download* action consists of downloading the web page corresponding to a single search result. Finally, an *extract* action consists of performing extraction on the downloaded webpage to obtain the required piece of information and using it to fill the slot in the original database.

Note that in the case of RBIE from the Web, the *query* actions can be initialized at the beginning of the task because they are fixed, but *download* actions and *extract* actions are generated dynamically and added to the list of available actions. That is, after a *query* action is performed, the *download* action corresponding to each of the search results is generated. Similarly, after a web page is downloaded, the corresponding *extract* action is generated. At each time point, only the actions that are instantiated can be considered as alternative actions to be performed. The RBIE task is to select the "best" action at each time point from a set of all valid actions.

Before selecting an action to perform at each time step, we need to consider several factors. We need to take into account the current state of the database, such as the number of slots filled and the uncertainty about them. We need to take into account the context provided by the results of all the actions so far, such as the results of the queries, pages that are not yet downloaded and processed. Even if this context is not yet in the database, it can provide valuable information for deciding which action to select. Finally, we also need to consider the properties of the candidate action itself, before selecting it.

We assume that we are given an existing model, M_e for extracting the required pieces of information from a single web page. We also assume that this model provides a confidence score for each value predicted. This score can be used to choose the best among the potential candidate values, and to determine whether or not an existing entry in the database should be updated by the newly extracted value.

2.3 Markov Decision Process Formulation

In this paper, we cast the Resource-bounded Information Extraction problem as solving a Markov Decision Process (MDP), M, where the states represent the state of the database at a given time, along with any intermediate results obtained from the Web, and actions represent the *query*, *download*, and *extract* actions as described in the previous section. We represent state as a tuple $S_t\langle DB_t, R_t, R'_t\rangle$, where DB_t is the state of the database at time t, R_t is the list of intermediate URL results and R'_t is the list of intermediate page results. The MDP for RBIE is described as a tuple, $M\langle S_0, \gamma, T(S,a,S'), R(S)\rangle$, where S_0 is the initial state of the database, γ is the discount factor, $T(S,a,S')$ is the state transition probability, or the probability that action a in state S at time t will lead to state S' at time $t+1$, and $R(S)$ is the reward function for being in state S.

Algorithm 1 Resource-bounded Information Extraction for the Web Using Q-function

Input:
  Database DB with missing entries, E_i
  Learned Q-function $Q(a,S)$
  Learned extraction model, M_e
  Time budget, b
Initialize all queries using keywords
$t = 0$
while $t <= b$ **do**
  $a_{t+1} = \arg\max_a Q(a,S)$
  if a_{t+1} is a *query action* **then**
    Issue query to a web search API
    Enqueue corresponding *download actions*
  else if a_{t+1} is a *download action* **then**
    Download the web page
    Enqueue corresponding *extract action*
  else if a_{t+1} is an *extract action* **then**
    Extract all candidate values from the web page
    Score each candidate using the model, M_e
    Fill the value of the best candidate in E_i
  end if
  $t = t + 1$
end while

One of the standard ways of solving an MDP is Q-learning[15], which provides a way to learn to select the best action at each time step. The Q-function, $Q(a,S)$ is the expected utility of taking action a in state, S. Hence, the best action to select at each step is:

$$a_{t+1} = \arg\max_a Q(a,S) \tag{1}$$

Algorithm 1 summarizes the RBIE for Web framework for filling missing information in a database using Q-function.

2.4 Learning Q-function

At the heart of solving an MDP for RBIE is the Q-function, $Q(a,S)$. We now discuss a method to learn it from real data. Much of the material in this section follows from [12, 14].

We know that Q-function obeys the following constraints:

$$Q(a,S) = R(S) + \gamma \sum_{S'} T(S,a,S') \max_{a'} Q(a',S') \tag{2}$$

To use this update equation, we need to learn the transition probability model, $T(S,a,S')$, which is difficult in our setup. Hence, we use the temporal-difference, or TD Q-learning approach, which is also called model-free, because it lets us learn the Q-function without using the transition probability model. The update equation for TD Q-learning is [1]:

$$Q(a,S) \leftarrow Q(a,S) + \alpha(R(S') + \gamma \max_{a'} Q(a',S') - Q(a,S)) \tag{3}$$

[1]There is some disagreement amongst Q-learning experts about using $R(S)$ vs. $R(S')$. We choose to use $R(S')$.

Algorithm 2 Temporal Difference Q-learning for RBIE, with ϵ-greedy exploration

Input:
Training database, DB
Initial parameters, θ
Q Function, $Q_\theta(a, S) = \sum_i \theta_i f_i(a, S)$
Reward Function, $R(S)$
Learning Rate, α
Discount factor, γ
$S_0 \leftarrow$ Initial State of DB
for $t \leftarrow 0$ to number of iterations T **do**
  $\epsilon = 1 - \frac{1}{T}$
  With probability ϵ, pick a random action, a_t
  With probability $1 - \epsilon$, pick $a_t = \arg\max_a Q_{\theta_t}(a, S_t)$
  $S_{t+1} = a_t(S_t)$ //perform a_t
  Let a' be all the valid actions from state, S_{t+1}
  for $i = 0$ to number of features **do**
    $\theta^i_{t+1} = \theta^i_t + \alpha[R(S_{t+1}) + \gamma \max_{a'} Q_{\theta_t}(a', S_{t+1}) - Q_{\theta_t}(a_t, S_t)]f_i(a_t, S_t)$
  end for
end for

Where, α is the learning rate. For any real-world RBIE task, the state space for the corresponding MDP is large enough to make it very difficult to learn this function accurately. Hence, we use function approximation. We represent the Q-function as a weighted combination of a set of features as follows:

$$Q_\theta(a, S) = \sum_i \theta_i f_i(a, S) \tag{4}$$

Where $f_i(a, S)$ are the features of the state S and action a, and θ_i are the weights on those features that we wish to learn. We now use the following equation for updating the values of θ_i to try to reduce the temporal difference between successive states.

$$\theta_i \leftarrow \theta_i + \alpha[R(S') + \gamma \max_{a'} \hat{Q}_\theta(a', S') - \hat{Q}_\theta(a, S)] \frac{\partial \hat{Q}_\theta(a, S)}{\partial \theta_i} \tag{5}$$

We can now use this update equation to learn the parameters of our Q-function from training data. The TD-Q-learning algorithm for RBIE is described in Algorithm 2. Note that we use ϵ-greedy approach for exploring the state space, where ϵ decreases in proportion to the number of training iterations.

We also need to design a custom reward function, $R(S)$ for using this algorithm. Under the RBIE from the Web setting, we can compute value of the reward function after performing action a_{t+1} on $S_t = \langle DB_t, R_t, R'_t \rangle$ as a weighted sum of correct, incorrect and total number of filled values and some properties of the intermediate results.

2.5 Building Extraction Model, M_e

In general, we can use any model of choice for information extraction in our framework that can extract the required information from a web page, and provide a confidence score for the extracted value. In this section, we present a simple, but novel information extraction method that can easily scale to large problem domains. The basic idea is to generate a list of potential candidate values from the web page, and using a binary classifier, such as Maximum Entropy, to classify them as being correct values or not, by observing features of the context in which they are found. Algorithm 3 describes how we train the model.

Let E be the set of entries with missing values in the database. We use patterns and lexicons to generate a list of candidates, C_{E_i} for each entry, $E_i \in E$. A *candidate* is a unique string that is a potentially correct value for an entry in the database. Each candidate, $c_j \in C_{E_i}$, consists of a list of *mentions*, M_j, which represent the actual occurrence of the candidate string in the web documents. Each candidate may have multiple mentions, across different web pages. Corresponding to each mention, $m_k \in M_j$, we have a list of properties, or features, $f(m_k)$ which describe the context in which it was found. Since we are interested in classifying the single, canonical value of these mentions, i.e, the candidate, we collapse the properties of different mentions for a candidate c_j into a single feature function, $f(c_j)$.

Let y_{ij} be a binary variable that represents whether c_j is the correct value for entry E_i. We can then represent the probability of c_j being the correct candidate as:

$$P(y_{ij}|c_j) = \frac{1}{Z} \exp(\sum_l \lambda_l f_l(c_j, y_{ij})) \tag{6}$$

Where, λ_l are the weights on the features, and Z, the normalization factor is given by:

$$Z = \sum_y \exp(\sum_l \lambda_l f_l(c_j, y_{ij})) \tag{7}$$

Since this is a supervised approach, our training data consists of the true values of E, which can be used to train the classifier. At test time, during an extract action, we classify each $c_j \in C_{E_i}$ at that time point, and select the one with the maximum posterior probability, $P(y_{ij}|c_j)$, as the "best" candidate to fill the slot.

3. SAMPLERANK FOR RBIE

In our previous work, we presented a different approach to learning a value function for selecting actions for a different problem domain[3]. We proposed the use of an online, error driven learning algorithm, called SampleRank [1, 16]. Since, we are also interested in investigating the effectiveness of the SampleRank approach in our current problem domain, we give its brief introduction here. For further details on training SampleRank for RBIE, please refer to [3].

Remember that we represent a state as, $S_t\langle DB_t, R_t, R'_t \rangle$. Our goal is to learn a value function similar to the Q-function, called $V(DB_t, R_t, R'_t, a)$, which is used to select the best action in a given state. In order to learn this function from training data, we first assume that its functional form is as follows:

$$V(DB_t, R_t, R'_t, a) = \exp(\sum_k \lambda_k f_k(DB_t, R_t, R'_t, a)) \tag{8}$$

Where, λ_k are model parameters and f_k are feature functions, defined over the database context, the current action, and the results of all previous actions.

We start training with state S_0, that represents the original state of the database. We consider all available actions at this point, and sample from states that result from these actions. We choose the state S^*, which is the result of the best

Algorithm 3 Building Extraction Model, M_e

Input:
Training Database DB with entries, E
Pattern or Lexicon Matcher, $L(w)$ that returns a set of matches, M_w from a Web Page, w
Feature functions, $f(.)$ describing context of M_w
A Supervised Learning algorithm, such as Max Ent
Initialize all queries using keywords
Initialize set of potential candidates per entry, $C_{E_i} = \{\}$
Initialize set of candidates for training, $C_t = \{\}$
while Any more actions remain **do**
  Pick a random action, a to perform
  if a is a *query action*, or a *download action* **then**
    Perform a and enqueue corresponding *download* or *extract actions*
  else if a is an *extract action* for Web Page, w **then**
    $M_w \leftarrow L(w)$
    for Each match, $m_k \in M_w$ **do**
      if String value (m_k) matches $c_j \in C_{E_i}$ **then**
        Add m_k as a mention of c_j
        Merge the features, $f(m_k)$ with $f(c_j)$
      else
        Create a new candidate, c_j, and add to C_{E_i}
        $label(c_j) \leftarrow$ string value (c_j) = true value (E_i)?
        Add m_k as a mention of c_j
        Set $f(c_j) \leftarrow f(m_k)$
      end if
    end for
  end if
end while
for all C_{E_i} **do**
  $C_t \leftarrow C_t \cup C_{E_i}$
end for
$M_e \leftarrow$ Train a Max Ent classifier with $f(c_t)$, for $c_t \in C_t$

action $a*$, predicted by V, and the state S', which is the best state predicted by the objective, or reward function, $R(S)$. SampleRank is an error driven learning algorithm, which lets us update parameters when the function learned up to a given point makes a mistake. We say the ranking is *in error* if the function learned so far assigns a higher score to the sample with the lower objective, or reward value, $R(S)$, i.e.:
$[(V_\Lambda(S^*) > V_\Lambda(S')) \wedge (R(S^*) < R(S'))] \vee [(V_\Lambda(S^*) < V_\Lambda(S')) \wedge (R(S^*) > R(S'))]$

When this condition is true, we update the parameters, Λ using perceptron update as follows:

$$\Lambda^t \Leftarrow \Lambda^{t-1} + \alpha(f(S'_t, a'_t) - f(S^*_t, a^*_t)) \quad (9)$$

where α is the learning rate used to temper the parameter updates. Note that SampleRank is a special case of reinforcement learning, which makes the comparison between the two approaches very interesting.

4. RELATED WORK

4.1 Resource-bounded Reasoning

Knoblock et al. [6] did some of the early work in planning for information gathering, followed by more Resource-bounded Reasoning work by Zilberstein et al. [18]. The problem of Resource-bounded Information Extraction (RBIE) was first introduced in our previous work [4], in which the main idea was to select a subset of the web documents to process by exploiting the network structure in the data. The example task in [4] is to find a missing year of publication in citation data. All available queries are issued, and all the search results are downloaded, which are then filtered using a simple heuristic. The number of documents that need to be processed for extraction is reduced by propagating information obtained from the Web through the underlying citation graph structure.

The state-action framework for RBIE was formulated in our more recent work [3], in which we also proposed the use of SampleRank [1, 16] algorithm to learn a value function for selecting actions. The task in [3] is to find URL of the faculty directory pages of top Computer Science departments in the U.S. The state-action framework is similar to the one described in this paper, except that the extract actions consist of simply classifying the web page as a faculty directory or not. Note that SampleRank does not have a direct notion of delayed reward. Hence, the delayed reward is 'baked into' the reward function, which includes intermediate results. Since URL of the web page plays a big role in predicting whether or not the web page is a faculty directory or not, examining the intermediate results provide a useful signal to the value function learner.

We introduce a novel extraction method using candidates and mentions, which can easily scale to large scale RBIE from the web applications. This paper also builds significantly over [3] by reformulating the RBIE problem as an MDP, introducing the use of Reinforcement Learning to solve it, and demonstrating the effectiveness of temporal difference Q-learning on a task that requires extracting values from web pages, as opposed to classifying URLs. Note that, in our case, the intermediate results are not as useful as in [3], which means that the notion of delayed reward is much more significant in learning to select actions. However, due to the novelty of both SampleRank and Q-learning as applicable in this domain, we choose to test the effectiveness of both the algorithms empirically.

4.2 Information Extraction From the Web

In the traditional information extraction settings, we are usually given a database schema, and a set of unstructured or semi-structured documents. The goal of the system is to automatically extract records from these documents, and fill in the values in the given database. These databases are then used for search, decision support and data mining. In recent years, there has been much work in developing sophisticated methods for performing information extraction over a closed collection of documents. Several different approaches have been proposed for different phases of information extraction task, such as segmentation, classification, association and coreference. Most of these proposed approaches make extensive use of statistical machine learning algorithms, which have improved significantly over the years. However, only some of these methods remain computationally tractable as the size of the document corpus grows. In fact, very few systems are designed to scale over a corpus as large as, say, the Web [2].

There are some large scale systems that extract information from the Web. Among these are KnowItAll [2], InfoSleuth [11] and Kylin [17]. The goal of the KnowItAll system is a related, but different task called, "Open Information Extraction". In Open IE, the relations of interest are not known in advance, and the emphasis is on discovering new relations and new records through extensive web access. In contrast, in our task, what we are looking for is very specific and the corresponding schema is known. The emphasis is mostly on filling the missing fields in known records, using resource-bounded web querying. Hence, KnowItAll and RBIE frameworks have very different application domains. InfoSleuth focuses on gathering information from given sources, and Kylin focuses only on Wikipedia articles.

The Information Retrieval community is rich with work in document relevance (TREC). However, traditional information retrieval solutions can not directly be used, since we first need to automate the query formulation for our task. Also, most search engine APIs return full documents or text snippets, rather than specific values. A closely related family of methods is question answering [7]. These systems do retrieve a subset of relevant documents from the Web, along with extracting a specific piece of information. However, they target a single piece of information requested by the user, whereas we target multiple, interdependent fields of a relational database. They need to interpret natural language question, whereas we need a keywords based mechanism for formulating queries. Most QA systems do not focus on prioritizing information acquisition actions, and the ideas in this paper could prove useful in building them. The semantic web community has been working on similar problems, but their focus is not targeted information extraction.

4.3 Active Information Acquisition

Learning and acquiring information under resource constraints has been studied in various forms. Consider these different scenarios at training time: *active learning* selects the best instances to label from a set of unlabeled instances; *active feature acquisition* [10] explores the problem of learning models from incomplete instances by acquiring additional features; *budgeted learning* [8] identifies the best set of acquisitions, given a fixed cost for acquisitions. At test time, the two common scenarios are selecting a subset of features to acquire, e.g. [13], and selecting the subset of instances for which to acquire features [5]. In our work, the resource-constraints are only applied at test time, and availability of unlimited resources is assumed at training time.

5. EXPERIMENTS

5.1 Extracting Faculty Information

Given a list of names of university faculty, our goal is to extract their email address, job title and department affiliation from the Web. In this section, we describe how we apply the RBIE framework to build a system that can efficiently acquire the required information. We also describe our experiments to test the effectiveness of SampleRank and Q-learning algorithms at selecting the most effective actions at each time step.

This is a challenging task due to several factors. In some cases, this information is readily available on faculty home pages, which are semi-structured. However, lecturers and faculty in many departments do not have home pages. Their information is scattered around the Web, without a uniform structure. Web pages are extremely noisy, and may lead to unexpected errors while performing extraction. Name ambiguity is another challenge, since many of the faculty have common names they share with other famous personalities. Some information on the Web is stale, or contradicting. For example, a faculty can be listed on one page as "assistant professor", while on another as "associate professor", reflecting a recent change of title. Finally, some information is not available on the Web at all.

Dataset	# Faculty	# Queries	# Docs	Total Actions
Training	70	1400	13686	28772
Testing	30	600	6065	12730
Total	100	2000	19751	41502

Table 2: Datasets

5.2 Dataset Description

We start with a list of faculty from University of Massachusetts at Amherst. We randomly choose 100 of these records as our dataset. The fields contain the first, middle and last name of the faculty, their email address, a list of job titles, and a list of department affiliations. The reason for multiple job titles and department affiliations is joint appointments. Unfortunately, the dataset we received contained several inaccuracies and was cleaned for better evaluation of our methods. For example, in some cases, a single column contained names of different departments. These are split into multiple columns. Punctuation and abbreviations, such as "Assoc. Prof." are cleaned and expanded. Despite the cleaning effort, the dataset we use is incomplete or contains errors. For example, the most current job titles are not reflected, and only one email address is included in the dataset, which may not be the one used by the person, or published on the Web. These imperfections in the data make both training and evaluation of our system challenging. Another problem in evaluating the accuracy of our system is the "generic-specific" problem in department names. For e.g., our system might predict the department affiliation for a faculty as "finance", while it might be listed as "management" in the ground truth dataset, or vice-versa. Since we use exact string match, we may even miss a match such as "school of management". Despite the difficulties, it is an interesting real world task for RBIE.

We use the Google search API for our experiments. In our task, the three fields that we extract are related to each other and often found in the proximity of each other on the same web pages. Hence, our query actions correspond to the entire record in the database, as opposed to a single 'entry', or cell. We formulate 20 different types of queries per faculty, as shown in Table 3, and consider top 20 hits returned by the search API. Assuming that we are not operating under resource-constraints, i.e., we perform all possible actions available, we get the dataset as described by Table 2.

5.3 Training The Extraction Models

Before we move to the action selection experiments, we need to build a model for extracting the required information from

Name
Name + Univ
Name + Univ + Curriculum Vitae
Name + Univ + Resume
Name + Univ + Profile
Name + Univ + Bio
Name + Curriculum Vitae
Name + Resume
Name + Profile
Name + Bio
Name in quotes + Univ
Name with middle name + Univ
Name + HomePage
Name + Contact
Name In Univ
Name in quotes In Univ
Name with middle name In Univ
Name + HomePage In Univ
Name + Contact In Univ

Table 3: Types of Queries. "Name" refers to first and last name, 'Univ' refers to keywords "university of massachusetts at amherst" and "In Univ" refers to "site:umass.edu"

individual web pages. Section 2.5 describes the algorithm we use for training the model. We use mallet [9] toolkit's implementation of the Maximum Entropy classifier. The available data is first split by 70%-30% for training and testing. The training phase for the extraction model is not resource-constrained, i.e., we use all possible query, download and extract actions.

The algorithm described in section 2.5 uses a pattern or lexicon matcher that returns a set of matches from a web page. These matches are added as a list of candidates to be filled in the database entry. For emails, we use a regular expression to match all the emails found in the web document. For job titles and department affiliations, we first build N-grams from body of the web page, where $N = 1, 2, 3, 4$. These N-grams are matched against lexicons to find candidate mentions in the web page. The features used to describe the context of these matches are shown in Table 4. The features across a mention are collapsed by using an 'OR' operator, since they are mostly binary. That is, if any feature is turned on in one of the mentions, it would be turned on for the candidate. In our early experiments, we found this method perform better than other merging operations, however, in future, we can build a more sophisticated method.

Let us first study the performance of the candidate classifier, in isolation of the Resource-bounded Information Extraction task. Any inaccuracy in this model, will not only result in poor accuracy during the RBIE process, but also mis-guide it due to inaccurate confidence prediction. Table 5 shows the classification performance of M_e. Note that F1 is the geometric mean of Precision and Recall. The main reasons of relatively lower F1 values on this model are inaccuracy of training data as described above, as well as the noisy nature of the web data. The advantage of using this model is that it is easy to build, and is scaleable for very large scale problems. In the future, we would like to experiment with a more sophisticated extraction model, in order to facilitate better accuracy of the classifier, as well as the RBIE process.

Features for Email Extractor
Type of query used
Email domain is from UMass
Web page domain name from UMass
Email host and web page URL host match
Relative position of faculty name and email
Match between faculty name and email username
Similarity between faculty sname and email username
Features for Job Title Extractor
Too many matches found on page
Web page domain name from UMass
Web page URL contains faculty name
Position of match on the document
The words "Assistant" or "Associate" preceds match
Relative position of faculty name and job title
Features for Department Extractor
Too many matches found on page
Web page domain name from UMass
Web page URL contains faculty name
Position of match on the document
The word "Department" precedes match
Relative position of faculty name and job title

Table 4: Features of the Extraction Models

Measure	Email	JobTitle	Department
Accuracy	97.97	92.76	95.83
Yes Precision	100.0	42.85	38.70
Yes Recall	43.75	23.07	44.44
Yes F1	60.86	30.00	41.37
No Precision	96.89	94.63	98.09
No Recall	100.0	97.78	97.59
No F1	98.42	96.18	97.84

Table 5: Performance of the Extraction Models

5.4 Evaluation

At test time, we start with the database that contains names of faculty. All other columns are empty. We consider this as time, $t = 0$. We assume that each action takes one time unit. The action selection scheme that we are testing selects one of the available actions, which is performed as described in Algorithm 1. The action is then marked as completed and removed from all available actions. If an extraction action is selected, it may affect the database by filling a slot and altering the confidence value associated with that slot. We evaluate the results on the database at the end of a given budget, b, or if we run out of actions.

We are interested in finding the email address, job title and department affiliation, all of which can have multiple true values. Note that this also includes minor variations. Throughout our evaluations, we compare against the multiple values of a column, and declare a match if the predicted value matches at least one of them. We use the following evaluation metrics to measure our system's performance. Since our task is slightly different from a traditional information extraction task, we use the following definitions of

evaluation metrics. Note that extraction recall measures the proportion of entries for which a true candidate value has been extracted from the web page. It may or may not get ranked as the "best" candidate. However, for the purpose of evaluating the order of selecting the query, download and extract actions, this is a very important metric. Even though at test time, it is independent of the performance of the underlying extraction model, M_e, it is still influenced indirectly by M_e through training.

$$\text{Precision} = \frac{\text{No. of Correctly Filled Entries in the Database}}{\text{No. of Filled Entries in the Database}}$$

$$\text{Recall} = \frac{\text{No. of Correctly Filled Entries in the Database}}{\text{No. of Test Entries in the Database}}$$

$$\text{Extraction Recall} = \frac{\text{No. of Correct Candidates Extracted}}{\text{No. of Test Entries in the Database}}$$

$$\text{F1} = \frac{2*\text{Precision}*\text{Recall}}{(\text{Precision}+\text{Recall})}$$

5.5 Baselines

We use two baselines for our experiments : random, and straw-man. At each time step, the random approach selects an action randomly from all available actions. The straw-man approach works as follows. The first query in the list is issued for each test instance. Next, the first hit from the search result for each test instance is downloaded and processed for extraction. Then, subsequent hits from the search result for each test instance are downloaded and processed. Finally, subsequent queries are issued in the descending order, followed by their corresponding download and extract actions. Note that this approach would quickly fill up the slots with the top hits of the queries, making it a very tough baseline to beat.

5.6 Learning Q-function from Data

We now describe how parameters θ for Q-function $Q_\theta(a, S)$ are learned using training data. Table 6 describes the features used. Note that at train time, we do not impose resource constraints. That is, training is performed till more actions are available. However, we only run Q-function parameter updates for a given number of iterations, which acts as a type of budget. We determine the number of iterations and learning rate empirically.

Similar to the test time, we start with a database with the email, job title and department name columns empty. The true values of these columns are only used to calculate the reward function. We initialize the parameters to zero. At each time step, we explore all possible actions, and update the parameters as described in Algorithm 2. We then choose the next action to perform as per the updated parameters and proceed similarly for the specified number of iterations.

We use the following objective function for training. Here, n is the number of slots filled in the database, d is the number of slots filled correctly, $\bar{d}$ is the number of slots filled incorrectly, r is the number of web pages downloaded so far that contain the correct slot value, and $\bar{r}$ is the number of web pages downloaded so far that do not contain the correct slot value. We choose these particular coefficients because of their emphasis on pracision, along with balancing recall.

$$\theta_t(S_{t+1}) = n * 1 + d * 100 + r * 10 - \bar{d} * 200 - \bar{r} * 0.5 \quad (10)$$

Features related to query action
Type of query
Features related to download action
Type of the corresponding query
Hit value in the search result
URL is from UMass
Webpage is HTML
Title contains keywords
Title contains faculty name
Features related to extract action
Type of the corresponding query
Hit value in the search result
URL is from UMass
Webpage is HTML
Title contains keywords
Title contains faculty name
Appropriate Size
Bad request code found

Table 6: Features for learning using SampleRank and Q-function

Since we are interested in comparing the SampleRank approach, we use the same features and objective function, or reward function as Q-learning.

5.7 Results and Discussion

We now compare the test-time performance of the two baselines, the value function learned using SampleRank, and the Q-function on their ability to select good actions at each time step. Note that we have already evaluated performance of the extraction method, and we are now focussing on quality of the action selection strategy. We evaluate performance after each 2000 actions from 0 to 14000 (since the total number of actions at test time is 12730). The most effective action selection scheme is the one that is fastest in achieving high values of evaluation metrics.

5.7.1 RBIE Using an Oracle

We first evaluate performance of the four action selection schemes in the presence of an oracle that perfectly classifies each candidate as the correct value for an entry or not with infinite confidence. We do this to isolate the effect of inaccuracies in the extraction model, M_e, which can severely misguide the RBIE system with wrong confidence values. For e.g., even if the action selection scheme selects a good web page for extraction, M_e can choose the wrong candidate for updating the value in the corresponding slot. While training the Q-function, this translates to incorrect reward values, which can severely impede learning. Table 5 shows that the F1 value for 'yes' label for each of the extractors are not high enough to avoid these problems.

Figure 1 shows the extraction recall values during the RBIE process for different fields, and the total number of entries. Note that in the presence of an Oracle, other evaluation metrics are not useful, since the precision is always 1, and the recall is the same as the extraction recall. We ran 1000 iterations of both SampleRank and Q-learning training with a learning rate, $\alpha = 1$, and discount factor, $\gamma = 0.9$ for this experiment. As we can see, the straw-man method is ex-

tremely effective, because it knows to process the top hits for a good query for each entry first. Given the complexity of the action domain, and the size of the state space, this policy is very difficult for a Q-learner to learn, especially after around 2000 iterations. It does, however, learn to beat the random action selection, as well as the value function learned by SampleRank. As expected, Q-learning performs better than SampleRank due to its modeling of delayed rewards, despite the use of exactly the same features and reward functions. For example, in the case of department name, it is able to obtain 82.8% of the best extraction recall, using only 42.8% of all actions.

5.7.2 RBIE Using Extraction Model, M_e

We now study the performance of our proposed method using an actual extraction model, M_e. In this case, each action selection strategy needs to balance both precision and recall. We ran 1000 iterations of both SampleRank and Q-learning training with a learning rate, $\alpha = 1$, and discount factor, $\gamma = 0.9$ for this experiment. Figure 2 shows the extraction recall, precision, recall and F1 values for total number of entries in the database. In these methods, precision and recall curves go down towards the end of information gathering process due to noise in the web data, and the extraction process. As before, we see that the straw-man method performs better in terms of extraction recall. However, its precision and recall drops mid-way through the information acquisition process, and Q-learning method performs better. Q-learning also comfortably out-performs random and SampleRank approaches. It achieves 90.2% of the final F1, by only using 28.5% of the total actions. This demonstrates the effectiveness of the policy learned by the Q-learner for selecting good actions for information gathering task. We believe that with more accurate labeling, a better extractor, and longer training, Q-learning method can be shown to be even more efficient.

6. CONCLUSION AND FUTURE WORK

In this paper, we formulated the problem of RBIE for the Web as a Markov Decision Process, and proposed the use of temporal difference Q-learning to solve it. We learn a policy for effectively selecting information-gathering actions, leading to significant reduction in resource-usage. On our example task of extracting faculty email, job titles and department names, the Q-learning based approach is able to achieve 90.2% of the final F1, by only using 28.5% of the total actions. We also compare it to a recently proposed, online, error-driven algorithm called SampleRank, and found that Q-learning performs better due to it's ability to model delayed reward. We also presented a novel extraction technique that can scale well for large scale, information gathering tasks.

The basic formulation of RBIE as an MDP opens up many interesting avenues of research. Use of TD Q-learning is one of the first attempts to learn general information gathering policies. Reinforcement Learning literature is rich with variations of Q-learning, which can be explored further. This framework is extendable in many ways. We can easily replace the candidate-mention scheme described in the paper with a more sophisticated extraction algorithm. We can also extend the set of information gathering actions defined here to suit the specific needs of a problem, and still use

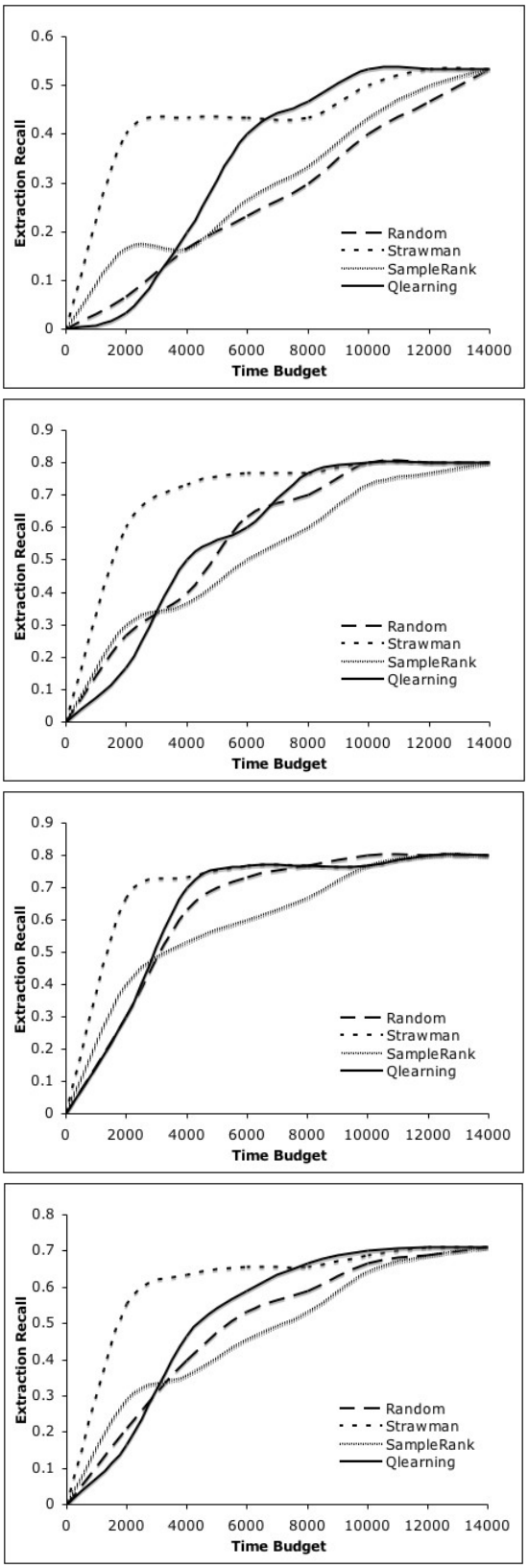

Figure 1: RBIE Using the Oracle. The graphs from top to bottom are : Email, Job Title, Department Name and Total Entries

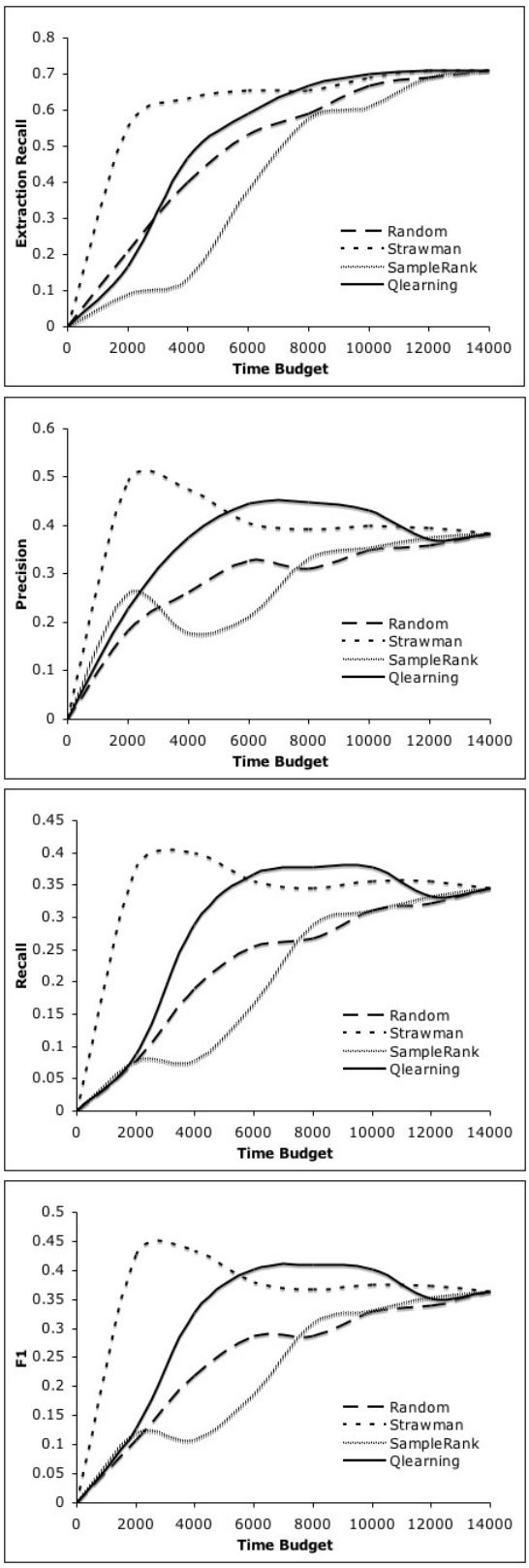

Figure 2: RBIE Using Extraction Model On Total Entries. The graphs from top to bottom are : Extraction Recall, Precision, Recall and F1

the general MDP framework. The success of such a learning based approach can lead to its application in many resource-conscious, real-world domains.

7. ACKNOWLEDGMENTS

This research draws on data provided by the University Research Program for Google Search, a service provided by Google to promote a greater common understanding of the web. We are thankful to Michael Wick for useful discussions. This work was supported in part by the Center for Intelligent Information Retrieval and in part by the Central Intelligence Agency, the National Security Agency and National Science Foundation under NSF grant number IIS-0326249. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect those of the sponsor.

8. REFERENCES

[1] A. Culotta. *Learning and inference in weighted logic with application to natural language processing*. PhD thesis, University of Massachusetts, May 2008.

[2] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-scale information extraction in knowitall. In *WWW'04*. ACM, May 2004.

[3] P. Kanani and A. McCallum. Learning to select actions for resource-bounded information extraction. *CIIR Technical Report 2011, IR-833 http://www.cs.umass.edu/∼pallika/publications/IR-833.pdf*.

[4] P. Kanani, A. McCallum, and S. Hu. Resource-bounded information extraction: Acquiring missing feature values on demand. In *Proceedings of the 14th PAKDD*, pages 415–427, 2010.

[5] P. Kanani and P. Melville. Prediction-time active feature-value acquisition for customer targeting. In *Workshop on Cost Sensitive Learning, NIPS*, 2008.

[6] G. A. Knoblock. Planning executing, sensing and replanning for information gathering,. In *Proceeding of the International Joint Conference onAI, IJCAI*, 1995.

[7] J. Lin, A. Fernandes, B. Katz, G. Marton, and S. Tellex. Extracting answers from the web using knowledge annotation and knowledge mining techniques, 2002.

[8] D. J. Lizotte and O. Madani. Budgeted learning of naive-bayes classifiers. In *UAI-2003*, pages 378–385. Morgan Kaufmann.

[9] A. K. McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[10] P. Melville, M. Saar-Tsechansky, F. Provost, and R. Mooney. An expected utility approach to active feature-value acquisition. In *ICDM'05*, pages 745–748, 2005.

[11] M. H. Nodine, J. Fowler, T. Ksiezyk, B. Perry, M. Taylor, and A. Unruh. Active information gathering in infosleuth. *IJCIS*, 9(1-2):3–28, 2000.

[12] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.

[13] V. S. Sheng and C. X. Ling. Feature value acquisition in testing: a sequential batch test algorithm. In *ICML '06*, pages 809–816, New York, NY, USA, 2006. ACM.

[14] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[15] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[16] M. Wick, K. Rohanimanesh, K. Bellare, A. Culotta, and A. McCallum. Samplerank: Training factor graphs with atomic gradients. In *ICML*, 2011.

[17] F. Wu, R. Hoffmann, and D. S. Weld. Information extraction from wikipedia: moving down the long tail. In *14th ACM SIGKDD*, pages 731–739, 2008.

[18] S. Zilberstein. Resource-bounded reasoning in intelligent systems. *ACM Comput. Surv*, 28, 1996.