Learning to Select Actions for Resource-bounded Information Extraction

Pallika Kanani and Andrew McCallum
{pallika, mccallum}@cs.umass.edu
University of Massachusetts Amherst

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 cast the information gathering task as a series of alternative, resource-consuming actions to choose from and propose a new algorithm for learning to select the best action to perform at each time step. The function that selects these actions is trained using an online, error-driven algorithm called SampleRank. We present a system that finds the faculty directory pages of top Computer Science departments in the U.S. and show that the learning-based approach accomplishes this task very efficiently under a limited action budget, obtaining approximately 90% of the overall F1 using less than 2% of actions. If we apply our method to the task of filling missing values in a large scale database with millions of rows and a large number of columns, the system can obtain just the required information from the Web very efficiently.**

Keywords-**Resource-bounded Information Extraction, Active Information Acquisition, Learning Value Function, Missing Data, SampleRank**

I. 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[2]. In many scenarios, we have a partial database and we need to fill in its missing values. Under such scenarios, it is undesirable, even computationally intractable to use traditional information extraction methods on the entirety of a vast, external, unstructured or semistructured corpus. For example, it is clearly not feasible to run an information extraction pipeline on the entire Web, in order to obtain the values of a single column in a database. The approach of RBIE framework for obtaining the required information is to automatically issue appropriate queries to the external source, such as a search API, select a subset of retrieved documents and extract the specified field.

Consider the following example of a real world RBIE task. Given a database of top Computer Science departments in the United States (Table I). Such a database may compile a lot of relevant information about the departments, such as location, admission and course information, statistics about the faculty and student body, etc. The faculty directory on the department websites are often a useful resource to obtain more information about the faculty, and it is desirable to be able to point the users of such a database directly to this page. It would also be a very useful starting point for automatic extraction of more detailed information about the faculty (such as the number of faculty, research interests, etc).

One way to obtain this information is to find the home pages of the departments and crawl the entire site to find the faculty directory pages. However, most department websites are large and complex, requiring us to process thousands of documents. Even though this particular case might not be computationally intractable, it does use significant resources in terms of computational time for crawling and processing documents, storage for all the crawled documents and the corresponding network bandwidth. Can we accomplish the same task using a much smaller fraction of these resources?

The information missing in the database is available on a finite, and relatively small number of pages on the web. We need to examine those web pages in order to obtain the required information. Before we can examine and extract information from these web pages, we need to download them to our computing devices. But before we can do that, we need to know where these relevant documents are located on the Web. The search interface of a large scale search engine, such as Google can help. We can issue queries that are driven by relevant information already available in the database to the search interface, obtain the location of web documents that potentially contain the information we need, download them, extract the required piece of information, and fill in the missing values in the database. This approach would use significantly limited resources.

Kanani et al.[2] introduced the Resource-bounded Information Extraction framework, which takes the focused information gathering approach of issuing queries to a search API, instead of crawling. They use existing, relevant information in the database and combine it with user defined keywords to generate queries. In most cases, there are several different queries to try for finding one piece of information, and some queries work better than the others. In the faculty directory example, a query could be formed by combining name of the university with keywords like "faculty directory", or a more sophisticated query could be "faculty inurl:department homepage url". The latter query looks for the key word "faculty" in the URL of the corresponding department homepage. Also, in many cases, it might be better to attempt finding one piece of information before another. For example, if the department home page URLs

University Name	Grad. Student Count	Fall App. Deadline	Dept Homepage	Faculty Dir Page	Faculty Count
Stanford University	550	December 13, 2011	http://www.cs.stanford.edu/	?	?
Massachusetts Institute of Technology	890	December 15, 2011	http://www.eecs.mit.edu/	?	?
Princeton University	100	December 15, 2010	http://www.cs.princeton.edu/	?	?
University of California-Berkeley	222	December 16, 2010	http://www.cs.berkeley.edu/	?	?
Carnegie Mellon University	?	December 15	http://www.cs.cmu.edu/	?	?

Table I
EXAMPLE DATABASE OF TOP COMPUTER SCIENCE DEPARTMENTS

are also missing, it might be important to fill that column before attempting to fill the faculty directory column. 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.

Unlike the previous work, which viewed RBIE as selecting a subset of documents to be processed, we 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. This leads to a reformulation of the Resource-bounded Information Extraction framework.

We consider the information available in the database at each time step as a *state*, and any act that helps obtain information as an *action*, such that performing an action on one state leads to a new state. The RBIE process then, is to select the best action from all available ones at each time step, so as to obtain most information under the given budget of actions. In the RBIE from the Web context, the actions are - issuing query to an external source, downloading a web document, or extracting a specific piece of information from a document. However, this method can be extended to other actions. In the Computer Science department database example, another action could be deciding if a piece of information is stale and needs to be refreshed, like the application deadline. In Table I, these dates are obtained from the department webpages on the same day, but only some of them need to be updated.

This setup is similar to Markov Decision Processes, in that we start with an initial state of the database with missing values, perform an action, which can result in a different state of the database, and we can define a reward function that depends on the action as well as its outcome.

In this paper, we propose a new method for learning to select the best action from a set of alternative actions, given a certain state of the database. The function takes into account the properties of the action itself, the properties of the context in the database, as well as the context of results of all the previous actions and assigns a score to each candidate action. The action with the highest score assigned by this *value function* is then performed.

We use SampleRank[4], [5], which is an online, error-driven learning approach proposed by Culotta et al to train this function. It has shown to perform very well on various different learning tasks, leading to faster training and more accurate results[4]. As we shall see later, the online nature of SampleRank also helps learning by sampling and exploring the state space.

We build a system that applies our method to the task of finding faculty directories. Using SampleRank, we learn a value function the evaluates each state action pair, which can then be used to select the best action at each time step. We show that the learning-based approach accomplishes this task efficiently under a limited action budget, and we obtain around 90% of the overall F1 using a budget of less than just 2% of the actions. Our method is much more broadly applicable than previous work, since the training framework can be implemented for any database whose missing values can be filled from the web.

II. RESOURCE-BOUNDED INFORMATION EXTRACTION

A. General Problem Definition

The general problem of Resource-bounded Information Extraction (RBIE) is as follows. We are given a database with some missing information and a set of possible actions that can help acquire that information from an external source, such as the web. At each time point, we need to select the best action from the set of alternative actions available, so as to acquire most information with least number of actions.

Let S_t be the state of database DB at time t and R_t be the result of all actions up to time t. Let $F(S_t)$ be the function that selects action a_t to be performed at time t and $\theta(S_t)$ be the objective function defined over the state of the database. Our goal is to optimize F for θ with a budget on t.

B. Resource-bounded Information Extraction From the Web

For RBIE from the Web, we consider three different types of actions - query actions, download actions, and extract actions. Queries are formed using information from an input entry in the database and a set of keywords. A query action is issuing a single query to a web search API and obtaining a set of search results. A download action is downloading the web page corresponding to a single search result. Finally, an extract action is performing an extraction task on the downloaded webpage to obtain the required piece of information and using it to fill the slot in the original database.

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 determine whether or not an existing entry in the database should be updated by the newly extracted value.

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. For e.g., after a query action is performed, the download action corresponding to each of the search result is generated. Similarly, after a web page is downloaded, the corresponding extract action is generated. At each time point, only the actions that have been created can be considered as alternative actions to perform.

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 can summarize all this information as a value function, V, that takes as an input the context of the database, the context of intermediate results, and properties of the action, and assigns a score to each candidate action. Using this value function, the best action to select at each time step is:

$$a_{t+1} = \arg\max_{a} V(DB_t, R_t, R'_t, a) \qquad (1)$$

Where, DB_t is the state of the database that contains filled and missing entries, R_t is the intermediate URL results, which contains a list of search results obtained from queries issued up to time t, and R'_t is the intermediate page results, which contains a list of web pages downloaded up to time t. In Section III, we describe how we can learn the value function V from data.

Algorithm 1 summarizes the RBIE for Web framework for filling missing information in a database.

Algorithm 1 Resource-bounded Information Extraction for the Web

```
Input:
Database $DB$ with missing entries, $E_i$
Learned value function $V(DB_t, R_t, R'_t, a)$
Learned extraction model, $M_e$
Time budget, $b$
Initialize all queries using keywords
Set all $c_i$ = 0, confidence in the value of $E_i$
$t = 0$
while $t <= b$ do
  $a_{t+1} = \arg\max_a V(DB_t, R_t, R'_t, a)$
  if $a_{t+1}$ is a query action then
    Issue query to a web search API
    Update $R_t$ with the new URLs
    Enqueue corresponding download actions
  else if $a_{t+1}$ is a download action then
    Download the web page
    Update $R'_t$ with new web page
    Enqueue corresponding extract action
  else if $a_{t+1}$ is an extract action then
    $E'_i$ = information extracted by $M_e$
    $c'_i$ = prediction confidence of $M_e$
    if $c'_i > c_i$ then
      $E_i = E'_i$
      $c_i = c'_i$
    end if
  end if
  $t = t + 1$
end while
```

III. Learning Action Selection Function

We can represent a state, S_t in the RBIE for Web framework as a triplet $< DB_t, R_t, R'_t >$, which consists of state of the database at a time t, intermediate URL results, R and intermediate page results, R'. We also denote new states as results of an action, i.e., $S_{t+1} = a_{t+1}(S_t)$. Learning to select actions in this framework is equivalent to learning a value function $V(S, a) \Rightarrow \Re$. In order to learn this function from training data, we first assume that its functional form is as follows:

$$V(DB, R, R', a) = \exp(\sum_k \lambda_k \Phi_k(DB, R, R', a)) \qquad (2)$$

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

A. Using SampleRank to Learn Value Function, V

We propose a new method to learn this function, using an online, error driven learning algorithm, called SampleRank[4], [5]. The online nature of SampleRank lets us update the parameters for each new sampled state during the training process without the need to perform inference between each step. SampleRank also allows us to define a custom objective function, $\theta(S)$, which enforces *ranking constraints* between pairs of samples.

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. In the most general version of these algorithm, we can use multiple samples at each time step to update the parameters. In our version, we choose the state S^*, which is the result of the best action $a*$, predicted by V, and the state S', which is the best state predicted by θ. Table II shows the notation for a quick reference.

B. Parameter Update

SampleRank is an error driven learning algorithm, which lets us update parameters when the function learned up to this 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, i.e.:

$$[(V_\Lambda(S^*) > V_\Lambda(S')) \wedge (\theta(S^*) < \theta(S'))] \vee [(V_\Lambda(S^*) < V_\Lambda(S')) \wedge (\theta(S^*) > \theta(S'))]$$

When this condition is true, we update the parameters, Λ using perceptron update. Note that there are also other options available for the functional form of parameter update, which are not explored here. The perceptron update we use is shown at Line 8 in Algorithm 2, where η is the learning rate used to temper the parameter updates.

We now choose the next best action according to function with the new parameters and perform it to get to the next state. Note that we can use different exploration techniques in the state space to choose the next state. We continue this process for the specified number of training iterations to obtain the final parameters of the learned value function. Algorithm 2 describes how we learn the parameters λ_k, given training data.

C. Objective Function

Under the RBIE from the Web setting, we can compute a custom objective function after performing action a_{t+1} on $S_t =< DB_t, R_t, R'_t >$ as a weighted sum of correct, incorrect and total number of filled values and intermediate results.

$$\theta_t(S_{t+1}) = C_n{*}n + C_d{*}d + C_r{*}r + C_{r'}{*}r' - C_{\bar{d}}{*}\bar{d} - C_{\bar{r}}{*}\bar{r} - C_{\bar{r'}}{*}\bar{r'} \quad (3)$$

Where, 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 correct URLs in the intermediate URL results, $\bar{r}$ is the number of incorrect URLs in the intermediate URL results, r' is the number of correct web pages downloaded in the intermediate page results and $\bar{r'}$ is the number of incorrect web pages downloaded in the intermediate page results.

V	Value Function
Λ, λ_k	Parameters
Φ	Feature Functions
θ	Objective Function
η	Learning Rate

Table II
NOTATION

Algorithm 2 SampleRank Estimation

```
1:  Input: Training database $DB$
    Initial parameters $\Lambda$
    Value Function $V_\Lambda(S, a)$
    Objective Function $\theta(S)$
2:  $S_0 \leftarrow$ Initial State of $DB$
3:  for $t \leftarrow 1$ to number of iterations $T$ do
4:      $a^*_t = \arg\max_a V_{\Lambda^{t-1}}(S_{t-1}, a)$
5:      $S^*_t = a^*_t(S_{t-1})$
6:      select sample from all states $S$ reachable from $S_{t-1}$:
        $S'_t = \arg\max_S(\theta(S))$
        $a'_t$ = Action that led to $S'_t$
7:      if Ranks of $S'_t$ and $S^*_t$ assigned by $V_{\Lambda^{t-1}}$ and $\theta$ are
        inconsistent then
8:          Update $\Lambda^t \Leftarrow \Lambda^{t-1} + \eta(\Phi(S'_t, a'_t) - \Phi(S^*_t, a^*_t))$
9:      end if
10:     $a_t = \arg\max_a V_{\Lambda^t}(S_{t-1}, a)$ //perform best action
11:     $S_t = a_t(S_{t-1})$
12: end for
```

IV. Related Work

A. Resource-bounded Reasoning

Our work reformulates the Resource-bounded Information Extraction framework proposed by Kanani et al. [2]. Their main approach is selecting a subset of documents to process, whereas, we propose the state-action model and present a general method to train an action selection function. Knoblock et al. [11] did some of the early work in planning for information gathering, followed by more Resource-bounded Reasoning work by Zilberstein et al. [10].

B. 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 [6].

There are some large scale systems that extract information from the web. Among these are KnowItAll [6], InfoSleuth [8] and Kylin [12]. 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 feature values.

A family of methods closely related to RBIE, is question answering systems [13]. 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 formulate queries by interpreting a natural language question, whereas we formulate and rank them based on the utility of information within the database. They do not address the problem of selecting and prioritizing instances or a subset of fields to query. This is why, even though some of the components in our system may appear similar to that of QA systems, their functionalities differ. The semantic web community has been working on similar problems, but their focus is not targeted information extraction.

C. *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* [14] explores the problem of learning models from incomplete instances by acquiring additional features; *budgeted learning* [7] 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. [15], and selecting the subset of instances for which to acquire features [3].

V. Experiments

We start with a list of top 125 Computer Science departments in the United States, as per the 2006 ranking[1] by Computer Research Association. We split the existing data by 70%-30% into training set and testing test. We use Google Search API to issue queries.

Our goal is to find URL of the faculty directory home page for each of these departments. This is a non-trivial task to perform in an automated fashion. Faculty directory pages of different departments have drastically different formats. They may or may not contain images and contact informations of faculties. It is also easy for an automated system to confuse the faculty directory page with other related pages like the faculty hiring (both types of pages almost always contain the word "faculty" in the URL) or even the home page of a particular faculty. A faculty home page may contain many names of co-authors of papers listed, contact information, as well as the word "faculty" somewhere in the URL, all of which could contribute to the mix up.

Furthermore, results of web queries are very noisy. Some of them may contain faculty directory of another university with similar name. They also tend to return the home pages of popular faculties in the department, along with some commercial websites that rank universities, and so on. Hence, we need a sophisticated model to identify faculty directory pages among all the web pages that the search interface returns.

As a precursor to our task of finding faculty directories, we find the URLs of the department home pages. This is a fairly easy task. We combine the name of the university, with keywords "computer science" to form a query and examine the top hit. In almost all cases, this returns the correct value of the department home page. We fill the department homepage column with the returned URL.

For the faculty directory finding task, we formulate four different types of queries per university, as shown in Table III, 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 IV.

A. *Building Extraction Model, M_e*

Since we are looking for only the URL of the faculty directory page, rather than some other information contained within the webpage, we cast the extraction problem as a classification problem. Hence, we build a Maximum Entropy based classification model, M_e to classify each web page as a faculty directory or not. Furthermore, we use the posterior of this classifier as the confidence value for each filled entry in the database. We use mallet [16] toolkit for building this

[1]http://www.cs.iit.edu/~iraicu/rankings/CRA-CS-Rankings-1993-2006.htm

model, and Stanford NER model for the NER features[1]. Table V describes all the features used for this model.

One of the difficulties in building this model is that there are multiple correct values of faculty directory pages. This is because pages are redirected, or web sites have multiple host names. Since it is difficult to manually label all web pages in the search results (> 8000 documents), we label at most one URL from the results as the true value. Availability of more labeling resources would help improve accuracy of the model. This is because some actually correct URL could get labeled as false during the training and testing phase of the classifier and might adversely affect its performance. Note that in some cases, none of the URLs returned by the search API are correct. In such cases, we manually find one correct URL for the faculty directory page from the web and use that as the true label. There were 17 departments for which the faculty directory page URL was not returned at all.

Let us first study the performance of the 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 VI shows the classification performance of M_e. We also show results on the training data to show the degree of fitting of the model. Note that F1 is the geometric mean of Precision and Recall. The main reasons of relatively lower F1 values on this model are the missing true URLs as well as the potentially inaccurate labeling as described above.

B. The Resource-bounded Information Extraction Process

At test time, we start with a database that contains the university names and the home page URLs of their computer science departments. All the faculty directory URL entries 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 use the following definitions of evaluation metrics for our task (Note the distinction between accuracy and precision):

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

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

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

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

Features related to queries
The type of query used
Hit value in the search result
Features related to URL
The document is HTML like
Words like “faculty”, “directory” and “people” found
Words like “job”, “hire”, “recruit”, or “employ” found
A tilda sign found (might indicate a user homepage)
First or last name found in non-host part of URL
URL host is dot com (not a university)
Same as department website
Same host as department website host
Features related to Web page title
Words related to bad request found
Words like “faculty”, “directory” and “people” found
Words like “job”, “hire”, “recruit”, or “employ” found
Features related to Web page body
Reasonable size
Phrase “bad request” or “error” found
Words like “faculty”, “directory”, or “people” found
Words like “phone”, “email”, “office”, or “professor” found
Word “publications” found
Count of Named Entities found
Count of email pattern matches found
Count of “PhD” pattern matches found
Features related to Web page layout
Count of images found
Count of tables and cells found

Table V
FEATURES OF THE WEB PAGE CLASSIFICATION MODEL

Measure	Training Set	Testing Set
Accuracy	98.38	98.07
Yes Precision	82.14	77.27
Yes Recall	72.52	61.44
Yes F1	77.03	68.45
No Precision	98.93	98.65
No Recall	99.38	99.36
No F1	99.16	99.00

Table VI
PERFORMANCE OF THE WEB PAGE CLASSIFICATION MODEL

C. Baselines

We use two baselines for our experiments : random and strawman. At each time step, the random approach selects an action randomly from all available actions. The strawman approach works as follows. From our initial analysis of the results of the queries, we found that queries can be sorted by their *coverage* values as $Q03$, $Q02$, $Q04$, and $Q01$. *Coverage* of a query is the proportion of all faculty directory URLs that are contained in that query's results. This means that the first query in this order is most likely to return the correct URL. Note that this pre-processing analysis provides a huge advantage to the strawman method. The action selection order is as follows :

- The query with the highest *coverage* value is issued for each test instance
- The first hit from the search result for each test instance

Query Type	Query String
Q01	“*University Name* + computer science + faculty directory”
Q02	“*University Name* + computer science + inurl:faculty”
Q03	“*University Name* + computer science + faculty site:*departmentSite*”
Q04	“*University Name* + computer science + site:*departmentSite* + inurl:faculty”

Table III
TYPES OF QUERIES

Dataset	Number of Universities	Number of Queries	Number of Documents	Total Actions
Training Set	88	352	5941	12234
Testing Set	37	148	2437	5022
Total	125	500	8378	17256

Table IV
DATASETS

is downloaded

- The first hit from the search result for each test instance is processed for extraction (classification)
- Subsequent hits from the search result for each test instance are downloaded and processed
- Subsequent queries are issued in the descending order of their *coverage* value, followed by their corresponding download and extract actions.

Note that this approach would quickly fill up the slots with the top hits of potentially effective queries, making it a very strong baseline to test our SampleRank method against.

D. Learning Value Function From Data

We now describe how parameters Λ for value function $V_\Lambda(S, a)$ are learned using training data. Table VII 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 SampleRank 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 faculty directory URL column empty. We initialize the parameters to zero. At each time step, we explore all possible actions, sample the states 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. In our early experiments, we tried the technique of parameter averaging, which is recommended in SampleRank literature, but in our case it did not prove to be very useful, since different types of actions lead to the update of different parameters.

We use the following objective function for training. Please refer to Equation 3 for explanation of terms. We choose these particular coefficients because of their emphasis on recall, along with balancing precision.

$$\theta_t(S_{t+1}) = n*300+d*100+r*10+r'*10-\bar{d}*200-\bar{r}*0.5-\bar{r'}*0.5 \quad (4)$$

Features related to counts
Counts of Filled Entried Counts of Intermediate Results Word ’Faculty’ inside intermediate results
Features related to corresponding entry
Corresponding entry is empty Confidence value of the entry (binned)
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 and Title contains keywords URL and Title contains job related keywords The host is “.com” Same host as department website Same as department website
Features related to extract action
Type of the corresponding query Hit of the corresponding result URL and Title contains keywords URL and Title contains job related keywords Appropriate Size Bad request code found

Table VII
FEATURES FOR LEARNING VALUE FUNCTION

E. Results And Discussion

We now compare the test-time performance of the two baselines and the learned value function on selecting actions at each time steps. We evaluate performance after each 1000 actions from 0 to 6000. Since the the initial performance of RBIE systems is the most crucial, we zoom into the first 1000 actions, and look at the performance at each 100 action interval. The most effective action selection scheme is the one that is fastest in achieving high values of evaluation metrics.

1) RBIE Using an Oracle: We first evaluate performance of the three action selection schemes in the presence of an oracle that perfectly classifies each webpage as a faculty

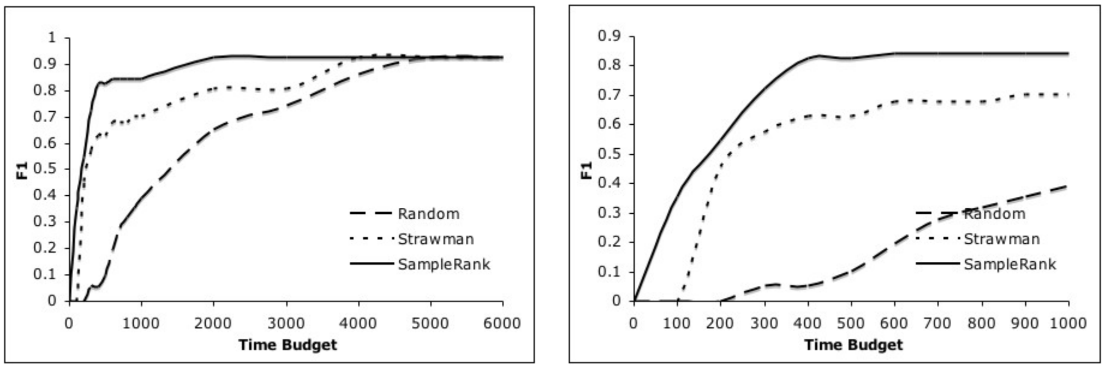


Figure 1. RBIE Using an Oracle : F1 (Figure on the right zooms to the first 1000 actions)

directory page or not with infinite confidence. We do this to isolate the effect of inaccuracies in the classification 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 assign a very low confidence value to it and hence discourage updating the value in the corresponding slot. Similarly, a wrong URL with a high confidence could replace a correct one in the database slot. Table VI shows that the F1 value for 'yes' label in the classifier is only 68.45, which may not be high enough to avoid some of these problems. The experiments with an oracle allow us to evaluate how well does the value function learn to select potentially useful actions early on.

Figure 1. shows the F1 values during the RBIE process for the three action selection schemes. We ran 2000 iterations of SampleRank training with a learning rate of 0.5 for this experiment. Figure 5. shows plots for the corresponding precision, recall and accuracy values. We see that SampleRank outperforms both the baselines at each budget interval in the first 1000 actions. The precision curve shows a value of one, because, the oracle always returns the correct answer with infinite confidence. Hence, we know that each entry filled in the table is correct, and the scheme that obtains higher recall first has been successful in identifying the best webpages to process early on.

2) RBIE Using Classification Model M_e: We now study the performance of our proposed method using an actual classification model, M_e. In this case, each action selection strategy needs to balance both precision and recall. We ran 2000 iterations of SampleRank training with a learning rate of 1.0. Figure 2. shows that the strawman approach is better at achieving high recall early on, but SampleRank is better at selecting more useful web pages to be able to fill the slots more accurately. This lets SampleRank to obtain most of the F1 value within the first 100 actions, and outperform the baselines in the first, crucial 400 actions. The drop in precision later on is due to inaccuracies in the confidence values predicted by M_e, which leads to a correct entry being replaced by an incorrect one.

SampleRank is able to obtain approximately 90% of the best F1 value it can achieve using all available actions, within the first 100 actions (which is less than 2% of all actions). In contrast, the strawman method takes 400 actions (approx. 7% of all actions) and the random method takes 4000 actions (approx. 67% of all actions) to reach their best F1 values. This demonstrates the effectiveness of the learning-based approach in selecting good actions for information gathering task. We believe that with more accurate labeling and a better classifier, SampleRank method can be shown to be even more efficient.

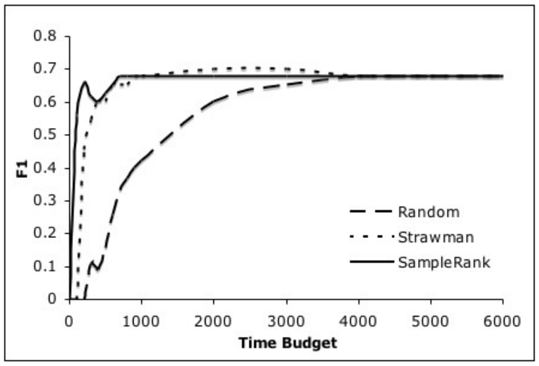
VI. Conclusion and Future Work

We propose a method for learning to select the most efficient actions for Resource-bounded Information Extraction. We formulate the RBIE problem as a state-action model and train a value function using an online, error-driven training method called SampleRank. This is the most important contribution of our paper. We also use a challenging, real world task of finding faculty directory URLs as a test bed for our algorithm. We demonstrate that the learning-based approach for selecting information-gathering actions consistently outperforms both, a random and a strong strawman baselines. By only using less than 2% of all available actions, the proposed method can achieve approximately 90% of the best F1 value it can achieve using all available actions.

On the experimental front, more variations of SampleRank can be tested, such as different sampling strategies, parameter update strategies and state exploration strategies. However, we can also explore a completely different method for learning the value function proposed in this paper. The general idea of state-action formulation and learning to select actions can be used for any RBIE for Web task, and can be applied to filling missing values in a large scale database efficiently.

VII. Acknowledgements

This research draws on data provided by the University Research Program for Google Search, a service provided



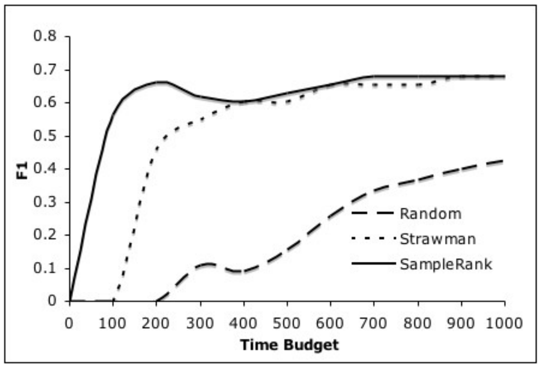


Figure 2. RBIE Using the classification model, M_e : F1 (Figure on the right zooms to the first 1000 actions)

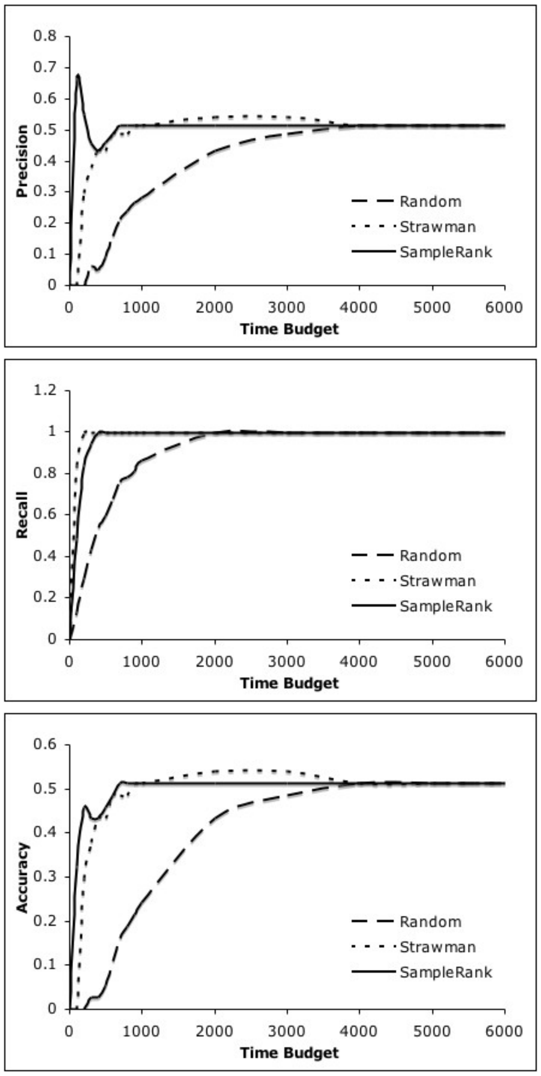


Figure 3. RBIE Using the classification model (Time reflects the number of actions performed)

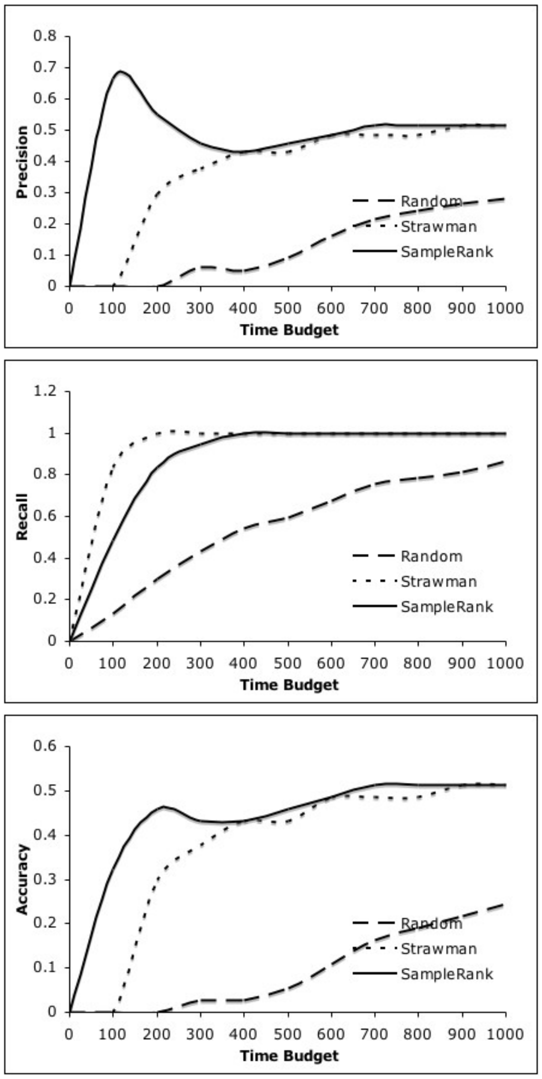


Figure 4. RBIE Using the classification model(Zoomed In to the first 1000 actions)

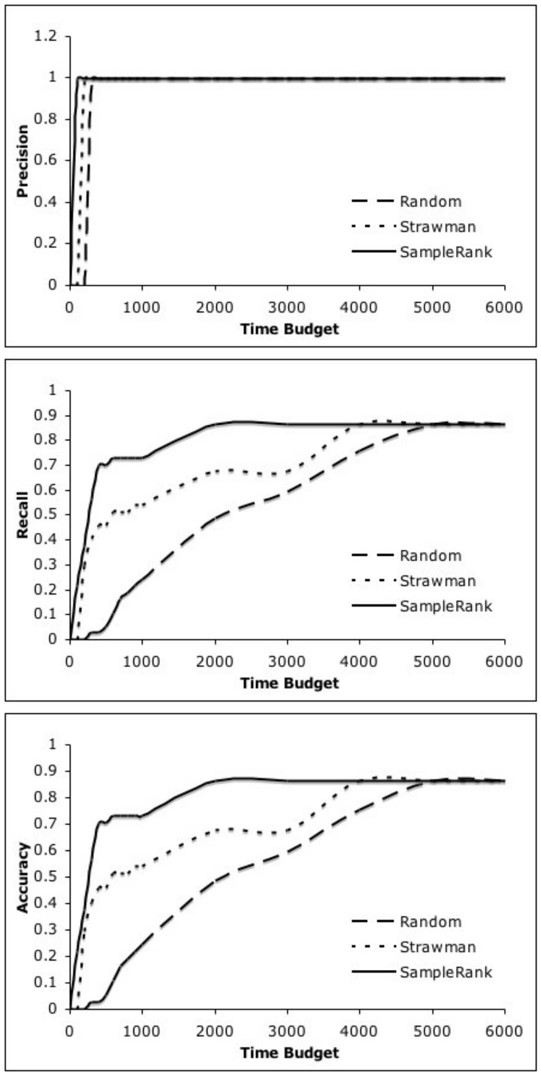


Figure 5. RBIE Using an Oracle (Time reflects the number of actions performed)

by Google to promote a greater common understanding of the web. We are thankful to Michael Wick for useful discussions, and Laura Dietz for reviewing the initial draft. 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 #IIS-0326249 . Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

REFERENCES

[1] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370.

[2] Kanani, P., McCallum, A. and Hu, S., ”Resource-bounded Information Extraction: Acquiring Missing Feature Values On Demand”, Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Hyderabad, India, June 21-24, 2010.

[3] Kanani, P. and Melville, P., ”Prediction-time Active Feature-value Acquisition for Customer Targeting”, NIPS 2008 Workshop on Cost Sensitive Learning.

[4] SampleRank: Training Factor Graphs with Atomic Gradients. Michael Wick, Khashayar Rohanimanesh, Kedar Bellare, Aron Culotta, Andrew McCallum. Proceedings of the International Conference on Machine Learning (ICML), 2011.

[5] Learning and inference in weighted logic with application to natural language processing. Aron Culotta, Ph.D. Thesis, University of Massachusetts, 2008

[6] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soder- land, D. Weld, and A. Yates. Web-scale information extraction in knowitall. In WWW04. ACM, May 2004.

[7] D.Lizotte, O.Madani, and R.Greiner. Budgeted learning of naive-Bayes classifiers. In UAI03, Acapulco, Mexico, 2003.

[8] M. H. Nodine, J. Fowler, T. Ksiezyk, B. Perry, M. Taylor, and A. Unruh. Active information gathering in infosleuth. IJCIS, 9(1-2):328, 2000.

[9] S. Zhao and J. Betz. Corroborate and learn facts from the web. In KDD, pages 9951003, 2007.

[10] Shlomo Zilberstein and Victor Lesser. Intelligent information gathering using decision models. Technical Report 96-35, Computer Science Department University of Massachusetts at Amherst, 1996.

[11] Planning, Executing, Sensing, and Replanning for Information Gathering. Knoblock, C. A. 1995. In Proceedings of IJCAI 1995.

[12] Wu, Fei, Hoffmann, Raphael, and Weld, Daniel S. Information extraction from wikipedia: moving down the long tail. In KDD 08:

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

[14] Melville, Prem, Saar-Tsechansky, Maytal, Provost, Foster, and Mooney, Raymond. An expected utility approach to active feature-value acquisition. In Pro- ceedings of the International Conference on Data Mining

[15] Sheng, Victor S., and Ling, Charles X. Feature value acquisition in testing: a sequential batch test algorithm. In ICML 06: Proceedings of the 23rd inter- national conference on Machine learning

[16] McCallum, Andrew Kachites. ”MALLET: A Machine Learning for Language Toolkit.” http://mallet.cs.umass.edu. 2002.