AskWorld: Budget-Sensitive Query Evaluation for Knowledge-on-Demand

Mehdi Samadi
Carnegie Mellon University
msamadi@cs.cmu.edu

Partha Talukdar
Indian Institute of Science
ppt@serc.iisc.in

Manuela Veloso
Carnegie Mellon University
veloso@cs.cmu.edu

Tom Mitchell
Carnegie Mellon University
tom.mitchell@cs.cmu.edu

Abstract

Recently, several Web-scale knowledge harvesting systems have been built, each of which is competent at extracting information from certain types of data (e.g., unstructured text, structured tables on the web, etc.). In order to determine the response to a new query posed to such systems (e.g., *is sugar a healthy food?*), it is useful to integrate opinions from multiple systems. If a response is desired within a specific time budget (e.g., in less than 2 seconds), then maybe only a subset of these resources can be queried. In this paper, we address the problem of knowledge integration for on-demand time-budgeted query answering. We propose a new method, AskWorld, which learns a policy that chooses which queries to send to which resources, by accommodating varying budget constraints that are available *only* at query (test) time. Through extensive experiments on real world datasets, we demonstrate AskWorld's capability in selecting most informative resources to query within test-time constraints, resulting in improved performance compared to competitive baselines.

1 Introduction

Over the last few years, several large knowledge bases (KBs), such as NELL [Mitchell *et al.*, 2015], Yago [Suchanek *et al.*, 2007], Freebase [Bollacker *et al.*, 2008], have been developed. These KBs include thousands of predicates (e.g., *City*, *Country*, *cityLocatedInCountry(City, Country)*, etc.) and millions of instances (facts) of these predicates (e.g., *cityLocatedInCountry(London, UK)*). While some of these KBs are user contributed (e.g., Freebase), some others are constructed from semi-structured data (e.g., Yago), or from unstructured Web data (e.g., NELL). While many of these KBs may be considered as repositories of facts which are updated periodically, techniques such as OpenEval [Samadi *et al.*, 2013] which can extract such facts on an on-demand basis have also been developed. Together, we shall refer to these KBs and on-demand extractors as *Knowledge Resources (KRs)* in the rest of the paper.

Given the heterogeneity of source data and extraction algorithms involved, these KRs can contain complimentary or often conflicting facts. Moreover, the degree of these differences and expertise of each KR may vary from one predicate to another. Hence, to satisfy the knowledge need of an end user or application [Samadi *et al.*, 2012], a single KR is not sufficient and it is necessary to integrate evidence from all these different KRs and return a consolidated response. In other words, we need a *Knowledge-On-Demand (KoD)* service which is able to aggregate opinions from all these diverse KRs taking their respective complementarity, conflicts, and expertise into account.

Ideally, given an input query such as *"Is City(Buenos Aires) true?"*, we would like the KoD service to aggregate opinions from all available KRs. Response time from a KR may vary depending on the predicate, especially for on-demand extractors such as OpenEval. In many applications of practical significance, the final response is desired within a specified time budget, and so polling all available KRs is infeasible, unfortunately. Thus, the KoD service has to devise a *policy* to decide on which subset of KRs to poll as a function of the query predicate and the specified time budget. We emphasize that the time budget may vary depending on the tolerance limits of the agent using the KoD service, and we would like the policy to be able to incorporate this gracefully, returning a more accurate response when more time budget is available. Also, the polling policy needs to be prepared a-priori (as there may not be time to learn the policy on the fly).

Additionally, we would like to point out that predicates in the KRs often have coupling relationships among them. For example, *hasMayor(Buenos Aires)* might help us infer that *City(Buenos Aires)* is indeed true. Thus, a *non-query predicate*, *hasMayor*, may help us derive a more accurate response for the *query predicate*, *City*. As we shall see in Figure 2 (Section 5.1), significantly more accurate responses are obtained when KR opinions on non-query predicates are also aggregated. While this is promising, this also makes the problem of identifying the right polling policy more challenging as it dramatically increases the number of polling possibilities.

While some aspects of the problems mentioned above have been studied in previous research which we shall review in Section 2, to the best of our knowledge, no previous research has tackled all the issues simultaneously, and definitely not in the context of a KoD service. We bridge that gap in this paper and propose AskWorld, whose architecture is shown in Figure 1. Given a user query with associated time budget,

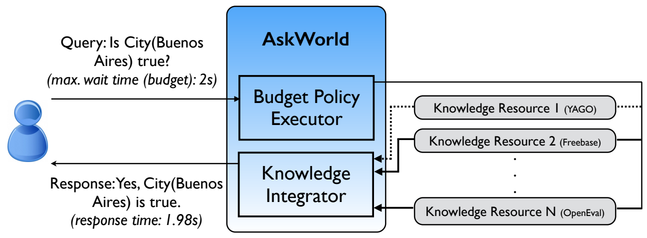
Query: Is City(Buenos Aires) true?
(max. wait time (budget): 2s)
AskWorld
Budget Policy Executor
Knowledge Integrator
Knowledge Resource 1 (YAGO)
Knowledge Resource 2 (Freebase)
Knowledge Resource N (OpenEval)
Response: Yes, City(Buenos Aires) is true.
(response time: 1.98s)

Figure 1: Architecture of the AskWorld system. Given a user query (e.g., *UnHealthyFood(sugar)?*) and a time-budget within which this query must be answered (e.g., 2sec above), the Budget Policy Executor selects a subset of the available knowledge resources (KRs) to poll (shown by solid lines in the figure; dotted lines connect knowledge resources that are not used to answer this particular query at the specified time budget). Responses from the KRs are integrated by the Knowledge Integrator and the final response is returned to the user within the specified time budget.

Budget Policy Executor (BPE) module of AskWorld selects the subset of available KRs to poll, such that the ultimate response can be provided within the specified time budget. Responses from the KRs are integrated by the Knowledge Integrator module and returned to the user. In this paper, we make the following contributions:

- We present AskWorld, a novel Knowledge-on-Demand (KoD) service that aggregates opinions from multiple knowledge resources to return the most accurate response to a query.
- AskWorld is designed to provide a best-effort response within the time-budget specified at query time by the user or the application. To the best of our knowledge, AskWorld is the first KoD system of its kind that is flexible enough to handle varying test-time budget constraints without model retraining. AskWorld achieves this by posing the problem as learning a policy in a Markov Decision Process (MDP).
- Through extensive experiments on real-world datasets, we observe that AskWorld significantly outperforms the state-of-the-art baselines, demonstrating its effectiveness.

2 Related Work

To explain the relation between AskWorld and the previous work, we can think of KoD as a classifier, with each predicate-specific confidence from a KR becoming a feature. In this classification setting, the cost incurred during feature computation, and the cumulative cost is upper bounded by a user specified limit. In this section, we briefly describe related research on cost-sensitive feature acquisition and draw their connection to AskWorld.

There is much previous work in the literature focusing on cascaded classifiers. The main idea behind these works is to train a cascade of classifiers which reject a set of test examples using inexpensive features at a very early stage. Early work on the cascaded classifiers focused on real-time object detection systems and assumed that all the features have the same feature cost [Viola and Jones, 2001; Bourdev and Brandt, 2005; Zhang and Viola, 2007]. Raykar et al. [Raykar *et al.*, 2010] extended the cascaded algorithms by jointly training different stages (i.e., classifiers) and reflecting the tradeoff between cost and accuracy during training. Unlike Raykar's work that preassigns features to cascade stages, Xu et al. [Xu *et al.*, 2014] proposed *Cronus* technique that makes the order of the feature extraction part of the training process. In contrast to *Cronus* which optimizes the cascade stages globally, Xu et al. [Xu *et al.*, 2012] proposed a stage-wise regression technique, called *Greedy Miser*, which outperforms *Cronus* approach by strictly incorporating the feature cost into the weak learners. Similar to these work but using reinforcement learning, Karayev et al. [Karayev *et al.*, 2013] proposed an approach that dynamically selects features which optimizes the cost-accuracy tradeoff. These techniques *indirectly* handle the input budget-constraint during the training by considering a hyper parameter that defines the trade-off between cost and accuracy. Hence they do not guarantee to provide a response within the input budget during the test time. AskWorld aims to overcome this shortcoming by being flexible enough to handle varying test-time budget without model retraining.

The idea of using multiple classifiers with different expected costs has been also studied by other researchers. Xu et al. [Xu *et al.*, 2013] proposed a cost-sensitive tree of classifiers [Tan, 1993] to reduce the average test time of classification (including feature acquisition cost) while maximizing the accuracy. Gao and Koller [Gao and Koller, 2013] presented an active classification process which builds a rich family of base classifiers during training, and iteratively selects base classifiers whose opinions should be integrated to classify the input test instance. Azari et al. [Azari *et al.*, 2004; 2012] proposed an optimization algorithm to optimize the net expected value of features, computed as the difference between the expected value and cost. These learning algorithms optimize either the *expected cost* without considering the exact input budget constraint, or only consider the trade-off between cost and accuracy during training, which make them inapplicable for the setting in which AskWorld operates.

The prior research most relevant to ours is the method presented in [Weiss and Taskar, 2013], where reinforcement learning techniques are used to learn a policy for feature selection under a specified test-time budget. Even though a relaxation to handle varying test-time budgets is presented in the same paper, it is not clear if *any* test-time budget can be handled within the relaxed version of their model. Kanani and McCallum [Kanani and McCallum, 2012] also presented a similar reinforcement learning approach, but the budget is used only during the policy execution. It is worth mentioning that our work is also loosely related to some other work in task allocation in crowdsourcing systems [Karger *et al.*, 2011; Chen *et al.*, 2013] and in information retrieval systems [Arnt *et al.*, 2004]. Due to space constraints, we don't delve into the details of their techniques.

3 Problem Statement

Let $(p, x, \mathcal{B})$ be a user query, where p is a predicate from an ontology $\mathbf{O}$, x is a candidate instance, and $\mathcal{B}$ is a response time budget (e.g., *(isCity, Buenos Aires, 2sec)*). We want AskWorld

Algorithm 1 AskWorld: Query Evaluation for Knowledge-on-Demand

```
Input: $\langle p, x, \mathcal{B}, Q, h\rangle$ /* $p$: predicate, $x$: instance to be evaluated, $\mathcal{B}$: input budget,
    $Q$: Budget policy learned during training $h$: knowledge integrator function. */
 1: /* Step 1: BudgetPolicyExecutor */
 2:   $t \leftarrow 0; \mathcal{B}_0 \leftarrow \mathcal{B}; \mathcal{K}_0 \leftarrow \{\}$
 3:   $S_0 \leftarrow \langle \mathcal{K}_0, \mathcal{B}_0 \rangle$
 4:   while $\mathcal{B}_t > 0$
 5:     // use budget policy $Q$ to select next resource-predicate pair, $\langle r, p'\rangle$, to poll
 6:     $\boldsymbol{a}_{\langle \boldsymbol{r},\boldsymbol{p'}\rangle} \leftarrow argmax_{a_{\langle r,p'\rangle}} Q(S_t, a_{\langle r,p'\rangle})$
 7:     Poll resource $\boldsymbol{r} \in \mathbf{R}$ to validate if $x$ is an instance of predicate $\boldsymbol{p'}$
 8:     // note that $\boldsymbol{p'}$ is not necessarily equal to $p$
 9:     $\mathcal{K}_{t+1} \leftarrow \mathcal{K}_t \cup \{\langle \boldsymbol{r}, \boldsymbol{p'}(x)\rangle\}$ // update poll response set
10:     $\mathcal{B}_{t+1} \leftarrow \mathcal{B}_t - c(\boldsymbol{r}, \boldsymbol{p'})$ // update residual budget
11:     $S_{t+1} \leftarrow \langle \mathcal{K}_{t+1}, \mathcal{B}_{t+1}\rangle$ // update state
12:     $t \leftarrow t + 1$
13:   end while
14:
15: /* Step 2: KnowledgeIntegrator */
16:   Build feature vector $x'$ and indicator vector $z$ from the result of polls in $\mathcal{K}_t$
17: return $\mathbf{y}^* = \arg\max_{y\in\{true,false\}} h(x', y, z)$
```

to validate whether x is a true instance of p by classifying it within the time budget $\mathcal{B}$ to one of the labels from $\mathcal{Y} = \{\text{false}, \text{true}\}$. AskWorld may poll a set of knowledge resources (KRs), $\mathbf{R}$, to determine whether x is an instance of p or of any other predicate from the ontology $\mathbf{O}$, and aggregate all the responses. The poll (p', x, r) checks the opinion of resource $r \in \mathbf{R}$, on whether x is an instance of predicate $p' \in \mathbf{O}$ and costs $c(r, p')$ time (note that p' is not necessarily equal to p). We assume that $c(r, p')$ remains constant during train and query time. To simplify our explanation, we assume that we have a set of polling queries $\mathbf{K} = \{\langle r, p'\rangle \mid r \in \mathbf{R}, p' \in \mathbf{O}\}$, i.e., all possible combinations of knowledge resources in $\mathbf{R}$ and predicates in $\mathbf{O}$.

Given a user query, the main challenge here is to learn a policy that identifies a subset of resource-predicate polling queries $\mathcal{K} \subseteq \mathbf{K}$ so that the most accurate response is provided within the response time budget. Also, the policy should be able to handle varying query-time budget (subject to a maximum upper bound) without the need of retraining.

4 Our Approach: AskWorld

In this section, we describe the AskWorld system. AskWorld (Algorithm 1) consists of two steps. In step 1, given a query and a time budget, AskWorld identifies the subset of knowledge resources to poll using BUDGET POLICY EXECUTOR (BPE) (Section 4.1); and in step 2 it aggregates the responses obtained from step 1 using KNOWLEDGE INTEGRATOR (Section 4.3) and returns the final result to the user.

4.1 Markov Decision Process (MDP) Formulation of BUDGET POLICY EXECUTOR (BPE)

We cast the budget-sensitive query evaluation problem as solving a Markov Decision Process (MDP), $\mathcal{M}$, represented as a tuple $\mathcal{M} = \langle \gamma, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}\rangle$ where γ is a discounting factor. The remaining components of the MDP are described below:

States ($\mathcal{S}$): Each state, $\mathcal{S}_t = \langle \mathcal{K}_t, \mathcal{B}_t\rangle \in \mathcal{S}$, of the MDP represents intermediate knowledge acquired, where $\mathcal{K}_t \subset \mathbf{K}$ is the set of knowledge resource-predicate polling queries issued by AskWorld and responses received, and $\mathcal{B}_t$ is the residual budget allowed to be used starting from state $\mathcal{S}_t$. During query time, the agent starts executing the MDP policy starting from $\mathcal{S}_0 = \langle \{\}, \mathcal{B}\rangle$, where $\{\}$ indicates that no polling response (feature) has been acquired and $\mathcal{B}$ is the input budget.

Actions ($\mathcal{A}$): Each action $a_{\langle r,p'\rangle} \in \mathcal{A}$ corresponds to a resource-predicate tuple $\langle r, p'\rangle$ which indicates which knowledge resource r to poll to validate if x is an instance of predicate p'. By taking the action $a_{\langle r,p'\rangle}$, we acquire the response from resource r at the cost of $c(r, p')$.

Transition function ($\mathcal{T}$): The transition function, $\mathcal{T}(\mathcal{S}_t, a_{\langle r,p'\rangle}, \mathcal{S}_{t+1})$, is defined as following:

$$\mathcal{T}(\mathcal{S}_t, a_{\langle r,p'\rangle}, \mathcal{S}_{t+1}) = \begin{cases} 1 & \text{if } \mathcal{B}_{t+1} = \mathcal{B}_t - c(r, p') \geq 0 \\ & \& \ \langle r, p'\rangle \in \mathcal{K}_{t+1}, \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\mathcal{B}_t$ is the remaining budget that we are allowed to use starting from state S_t. Intuitively, Equation 1 says that the probability of moving from state $\mathcal{S}_t$ to $\mathcal{S}_{t+1}$ by taking action $a_{\langle r,p'\rangle}$ is equal to 1 if the cost of $a_{\langle r,p'\rangle}$ is less than the remaining budget $\mathcal{B}_t$ and if the query $\langle r, p'\rangle$ is part of $\mathcal{K}_{t+1}$.

Why include budget in the transition function?
The alternative option, similar to [Weiss and Taskar, 2013], is to incorporate the budget constraint as part of the reward function. For example, when a transition is invalid (i.e., we are out of budget), the value of the reward function would be either a large negative value or zero. This approach has two main disadvantages: (i) defining the reward to be negative for the invalid transitions may affect the learning process since the expected reward value of a state can be potentially penalized as the agent receives negative value for the states that it is not able to move to, and (ii) defining the reward of the invalid transitions to be equal to zero also makes it impossible for the agent to differentiate between states that are not reachable and those that have the expected reward of zero. By defining the budget constraint as a part of the transition function, we avoid both these problems since the agent is not able to traverse to states that require more budget than the remaining budget $\mathcal{B}_t$. Thus, unlike [Weiss and Taskar, 2013]'s approach which may find a policy that violates the input budget, our approach always finds a policy that satisfies the input budget constraint.

Reward function ($\mathcal{R}$): The reward function $\mathcal{R}$ is defined as the value of the information added by each action $a \in \mathcal{A}$ given current state $\mathcal{S}_t$. More precisely,

$$R(\mathcal{S}_t, a, \mathcal{S}_{t+1}) = \begin{cases} \frac{1}{|\Psi'|}\sum_{(x_i,y_i)\in\Psi'} \bar{h}(x_i, y_i, z_{t+1}) - \bar{h}(x_i, y_i, z_t) & t \neq 0 \\ 0 & t = 0 \end{cases} \tag{2}$$

where Ψ' is the set of evaluation data (subset of the input training data Ψ) in the form of $\Psi_i = (x_i, y_i)$. z_t is defined as a binary vector where $|z_t| = |\mathbf{K}|$. The jth element of z_t is equal to 1 if the value of query $\langle r, p\rangle$ appears in the state $\mathcal{S}_t$ (as part of $\mathcal{K}_t$), and otherwise it is equal to 0. Vector z can be seen as an indicator vector used as input of a predictor (knowledge integrator, Section 4.3) function. Function $\bar{h}(x, y, z)$ is defined in terms of $h(x, y, z)$. Function $h(x, y, z)$ is a predictor function which returns the confidence value of predicting label $y \in \mathcal{Y}$ for the input instance x using queries indicated by vector z. The function $\bar{h}(x, y, z)$ is defined as follows:

$$\bar{h}(x, y, z) = h(x, y, z) - \max_{y' \neq y} h(x, y', z) \tag{3}$$

Function $\bar{h}$ returns the maximum difference in the confidence value of classifier in predicting the true label compared to all the other labels in $\mathcal{Y}$.

Intuitively, Equation 2 says that each time that we poll $\langle r, p\rangle$, the reward that we receive is equal to the change in the margin of our predictor h, averaged over all the training data points. In other words, the reward function measures the value of the knowledge that we acquire from each of the resources. If the value of $R(\mathcal{S}_t, a, \mathcal{S}_{t+1}) > 0$, then it means that the selected action increases the confidence value of our predictor in predicting the true label compared to the other labels, and if $R(\mathcal{S}_t, a, \mathcal{S}_{t+1}) < 0$, it means that it is decreasing the confidence value. Ideally, we would like to choose a polling query that increases the confidence of our prediction when moving from state $\mathcal{S}_t$ to $\mathcal{S}_{t+1}$.

Given a deterministic policy Π and a sequence of states $\mathcal{S}_0, ..., \mathcal{S}_n$ computed from Π, we define $\mathcal{R}_\Pi$ as the reward that we receive by the following policy determined by Π. Then,

$$\mathcal{R}_\Pi = \sum_{(x_i, y_i) \in \Psi'} \bar{h}(x_i, y_i, z_n) - \bar{h}(x_i, y_i, z_0) \tag{4}$$

Equation 4 says that the reward that we receive from following policy Π is equal to the difference between the confidence value of the predictor h when using the acquired queries in state $\mathcal{S}_n$ and the queries in state $\mathcal{S}_0$.

Predictor Function: To build the predictor function $h(x, y, z)$, we use Support Vector Machines (SVM). The SVM is trained by assuming that all the polling query responses (feature values) are acquired for all the training instances. For any new instance where some of the feature values are missing (i.e., zero elements in the z vector), we use the prior value for the missing feature as the average feature value over all the training data (assuming the size of positive and negative data are equal). Thus, each feature of the SVM corresponds to a response to a polling query in $\mathbf{K}$, thereby making its feature space $|\mathbf{K}|$-dimensional.

Theorem 1 *(***Property of MDP***) In the MDP $\mathcal{M}$, for every two policies Π and Π' which respectively map states $\langle\{\}, \mathcal{B}\rangle$ and $\langle\{\}, \mathcal{B}'\rangle$ ($\mathcal{B} \neq \mathcal{B}'$) to some actions, $\mathbf{S}(\Pi) \cap \mathbf{S}(\Pi') = \varnothing$, where $\mathbf{S}(\Pi)$ and $\mathbf{S}(\Pi')$ are the sets of all the states that can be generated by policies Π and Π', respectively.*

The proof of Theorem 1 is omitted for space reasons, but is available in [Samadi *et al.*, 2015]. Intuitively, Theorem 1 says that the policies we learn for the different initial budgets are independent of each other. In other words, there is no advantage of simultaneously learning policies for the different initial budget values. In the next section, we explain how to address this problem by abstracting states in the MDP.

4.2 Solving MDP

Depending on the number of predicates in the ontology $\mathbf{O}$ and the resources in $\mathbf{R}$, the state space can be significantly large. In this section, we present two approaches to solve the MDP whose performances are compared in Section 5.

1. AskWorld (V*): Abstracting MDP and Solving It Using Value Iteration: We abstract the state space of $\mathcal{M}$ to a smaller space. The resulting MDP is denoted by $\bar{\mathcal{M}}$. The abstraction is done by the following ways: (i) defining the remaining budget in each state to be within an interval, instead of being equal to an exact budget amount, and, (ii) reducing the size of polling queries in $\mathbf{K}$ to a smaller set $\bar{\mathbf{K}}$.

To define the budget interval for each state, we assume that $\mathbf{B}_U$ is the upper bound on the budget that we receive during query time. The size of each interval is defined to be equal to δ and we have $\frac{\mathbf{B}_U}{\delta}$ distinct intervals. In the abstract MDP, each state is defined as $\mathcal{S}_t = \langle \mathcal{K}_t, [\mathcal{B}_t^L, \mathcal{B}_t^U]\rangle$, where the second element is the budget interval, $\mathcal{B}_t^L = k \times \delta$, and $\mathcal{B}_t^U = (k+1) \times \delta$. The reward function in the abstract MDP is the same as the original MDP.

The transition function for the abstract MDP depends on the budget interval defined in each state, the cost of the actions, and the value of the budget that we receive during the query time. For example, assume that during the test time, we are in a state where the budget interval is $[40, 60]$, $\delta = 20$, and we are executing action $a_{\langle r,p\rangle}$. If the true remaining budget is 40 and $c(a_{\langle r,p\rangle}) = 5$, then we move to a state where the budget interval is $[20, 40]$, however if the remaining budget is 50, we then move to a state where the budget interval is $[40, 60]$ (in both these cases $\langle r, p\rangle$ is added as one of the queries in the state). The problem is that we don't know the exact budget that is given during the test time while learning a policy for the MDP. To address this issue, we assume that different budget values are equally likely to be given during test time, and then define different probabilities for transitioning to different states, e.g., in our example, we move to the state with budget interval $[20, 40]$ with probability $\frac{c(a_{\langle r,p\rangle})}{\delta} = 0.25$ and move to the state with budget interval $[40, 60]$ with probability $\frac{\delta - c(a_{\langle r,p\rangle})}{\delta} = 0.75$. This example helps us to formally define the transition function.

To define the transition function, we know that the cost of each action $a_{\langle r,p\rangle}$ can be written as $c(a_{\langle r,p\rangle}) = \delta \times k + m$, where $k = \lfloor \frac{c(a_{\langle r,p\rangle})}{\delta} \rfloor$. Therefore, $m = c(a_{\langle r,p\rangle}) - \delta \times k$. So, the transition function for the abstract MDP can be defined as,

$$\mathcal{T}(\mathcal{S}_t, a_{\langle r,p\rangle}, \mathcal{S}_{t+1}) = \begin{cases} \frac{m}{\delta} & \text{if } \mathcal{B}_{t+1}^U = \mathcal{B}_t^U - (k+1) \times \delta \\ & \ \& \ \mathcal{B}_{t+1}^L = \mathcal{B}_t^L - (k+1) \times \delta \\ & \ \& \ a_{\langle r,p\rangle} \in \mathcal{K}_{t+1} \ \& \ \mathcal{B}_{t+1}^L > 0 \\ \frac{\delta - m}{\delta} & \text{if } \mathcal{B}_{t+1}^U = \mathcal{B}_t^U - k \times \delta \\ & \ \& \ \mathcal{B}_{t+1}^L = \mathcal{B}_t^L - k \times \delta \\ & \ \& \ a_{\langle r,p\rangle} \in \mathcal{K}_{t+1} \ \& \ \mathcal{B}_{t+1}^L > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where the budget intervals of states $\mathcal{S}$ and $\mathcal{S}'$ are respectively defined by $[\mathcal{B}_t^L, \mathcal{B}_t^U]$ and $[\mathcal{B}_{t+1}^L, \mathcal{B}_{t+1}^U]$.

Abstracting the state space in MDP not only benefits us by reducing the number of states, but also creates shared states between the policies learned for the different budget values.

Theorem 2 *(***Property of Abstract MDP***)*
In the abstract MDP $\bar{\mathcal{M}}$, if $\delta \geq \frac{\mathbf{B}_U}{|\mathcal{A}|}$ and $c(a) > 0$ for all the actions a, then for any two initial states $\mathcal{S}_0$ and $\mathcal{S}_0'$ with different budget intervals, $\mathbf{S}(\mathcal{S}_0) \cap \mathbf{S}(\mathcal{S}_0') \neq \varnothing$, where $\mathbf{S}(\mathcal{S}_0)$ and $\mathbf{S}(\mathcal{S}_0')$ are the sets of all the states that are reachable from states $\mathcal{S}_0$ and $\mathcal{S}_0'$, respectively.

The proof of Theorem 2 is omitted for space reasons, but is available in [Samadi *et al.*, 2015].

In addition to abstracting the budget, we also need to reduce the number of polling queries in $\mathbf{K}$ to a smaller set $\bar{\mathbf{K}}$. Different feature/variable extraction techniques that have been studied in the machine learning community [Blum and Langley, 1997] can be used to construct $\bar{\mathbf{K}}$. Among these techniques, we use feature ranking using weights from linear SVM classifier which has been shown to be an effective approach for feature selection [Guyon *et al.*, 2002]. The SVM is trained using the training data set and the top K ranked polling queries are chosen to be included in $\bar{\mathbf{K}}$. In our experiments, we drop all polling queries with zero weight and keep the rest as part of $\bar{\mathbf{K}}$.

Given the abstract MDP, we can directly learn a policy using the value iteration algorithm. Instead of calculating the policy as the optimal action to be taken from each state, we save the value of $Q(\mathcal{S}, a_{\langle r,p\rangle})$

for all the states $\mathcal{S}$ and actions $a_{\langle r,p\rangle}$, and calculate the optimal policy during the test time given at each state. This allows us to make sure that the action that we are choosing is always within our remaining budget. For example, during the test time, if we are in a state where the budget interval is $[0, 10]$ but the actual remaining budget is equal to 4, we should prune selecting actions that require budgets higher than 4.

Our approach to abstract the MDP and solve it using the value iteration algorithm needs tuning a few parameters such as the value of δ. In addition, even with the abstraction, the size of the state space could still be huge when the ontology is very large. Although in the experimental results we show that by solving the abstract MDP we could significantly outperform other baseline approaches, in the next section we explain how to approximate the value iteration algorithm using Q-learning with linear function approximation which might be more suitable for applications with a very large ontology.

2. AskWorld (PQL): Q-Learning with Linear Function Approximation: An alternative approach to solve the MDP is to approximate the policy using the temporal-difference, or TD Q-learning, with function approximation [Sutton and Barto, 1998; Lagoudakis *et al.*, 2003]. In the standard setting of TD Q-learning with linear function approximation, the Q-function is represented as a weighted combination of a set of features as follows,

$$Q_\theta(\mathcal{S}, a) = \sum_i \theta_i \phi_i(\mathcal{S}, a)$$

where $\phi_i(\mathcal{S}, a)$ are the features defined over state $\mathcal{S}$ and action a, and θ_i are the set of weights to learn. To learn the parameters θ_i, we follow the standard setting [Sutton and Barto, 1998; Lagoudakis *et al.*, 2003], where an online algorithm is used to update the values of θ_i and reduce their temporal differences between successive states.

4.3 Knowledge Integrator

Using the training data Ψ, we first train the predictor $h(x, y, z)$ (e.g., SVM classifier) and then calculate the Q values using either the state abstraction or function approximation techniques. Given a trained predicator h and the Q function, we follow the policy defined by Q starting from state $\langle\{\}, B\rangle$. On reaching the last state, denoted by $\mathcal{S}_n$, the label for x is calculated as:

$$\mathbf{y}^* = \arg\max_{y \in \mathcal{Y}} h(x, y, z_n)$$

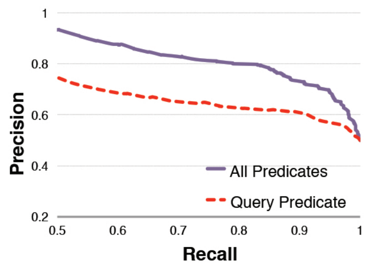

Figure 2: Precision-Recall curve comparing performance of AskWorld when using *Query Predicate* vs. *All Predicates*.

5 Experimental Results

For the experiments in this section, we use 25 categories randomly chosen from all the categories that are in common between Freebase [Bollacker *et al.*, 2008] and NELL [Mitchell *et al.*, 2015] knowledge bases. For each predicate, 200 random instances are provided as seed examples to train AskWorld, and these are partitioned into two sets: *classifier-training* and *policy-training*. The classifier-training set is used to train the SVM classifier and the policy-training set is used to solve the MDP (used as part of the reward function) or to find greedy ordering for the baseline approaches. 50 instances are also randomly chosen as the test data for each predicate: 25 positive and 25 negative examples. The negative examples are chosen from other predicates in the FreeBase ontology. We compare AskWorld to multiple baseline approaches using standard performance metrics of precision, recall, and F1 score.

For the knowledge resources in $\mathbf{R}$, we use three different knowledge acquisition techniques that are developed as part of the NELL project. The first one is CMC which looks for certain orthographic features in the query entity's name. The next resource is the NELL KB itself, which is built using several sub-components. We use CKB as the third resource. While the NELL KB has high precision and low recall, CKB is its noisy version (low precision), but covers many more facts (high recall). Queries against CKB tend to be slower as it has higher coverage. Please note that the performance of these resources significantly vary across predicates, and thus given an instance, it is not always clear which resource to query and how to combine their result. This is especially important in settings with runtime budget constraints that prohibit the exhaustive query of all the resources.

5.1 Does polling KRs for non-query predicates help?

One of the contributions of this paper is to show that the accuracy of a knowledge-on-demand system improves by aggregating opinions of different KRs for *all* the predicates in the ontology, compared to when we only poll KRs for the query predicate.

Figure 2 shows the precision-recall curve comparing the setting when only the query predicate (referred to as *Single Predicate*) is allowed compared to the setting when we poll KRs for all predicates in the ontology (*All Predicates*). All results are averaged over all 25 target predicates. The result of different queries are aggregated using a trained SVM classifier. The result shows that the precision of our system is improved by around 25% absolute when we aggregate opinions of KRs for all the predicates in the ontology compared to the single predicate approach, e.g., the precision is improved from 0.64 to 0.81 at the recall value of 0.75.

5.2 Are budget-sensitive policies able to select effective polling queries?

Note that the *All Predicates* setting in the previous section is slower than the *Query Predicate* approach since it requires polling KRs for all the predicates in the ontology. This leaves us with the question of how quickly we can achieve the same result as the *All Predicates* setting while using minimum budget. The first baseline that we consider is **Random**, where given a query-time budget, we randomly choose which resources and predicates to poll until running out of the budget. The values returned for different polling queries are given as an input to the trained SVM classifier, with missing values in the input of the classifier represented by zeros. **Random+** is similar to Random, except that missing feature values are represented by prior values calculated during training.

We also compare AskWorld to three different greedy-based algorithms. **Single Greedy+** ranks queries in the greedy order based on their information gain. For *each predicate* in the ontology, the algorithm finds one ordering for polling queries in **K**. SVM is used to measure the information gain for each query using the evaluation dataset. During query-time, given a new query, the algorithm polls resources by following the greedy ordering until running out of time budget. **Multiple Greedy** algorithm is similar to *Single Greedy+*, except that it iterates over different potential budget values and finds a *separate* greedy ordering for *every predicate* and for *every such budget value*. It also trains a separate classifier for each budget value. **Greedy Miser** is a learning-based approach presented by [Xu *et al.*, 2012], which we consider as a representative of the state-of-the-art. Greedy Miser uses step-wise regression [Friedman, 2000] which minimizes a loss function that explicitly trades off the feature cost and the accuracy. The output of the learning algorithm is an additive classifier which is a linear combination of a set of regression trees. GreedyMiser is trained with the same training data as other algorithms. Note that GreedyMiser handles the input budget-constraint *indirectly* using a λ parameter which defines the tradeoff between cost and accuracy. We tuned λ over the range of values suggested in [Xu *et al.*, 2012], and used the optimal value of $\lambda = 0.06$ for the experiments in this section. For other parameters, we use a learning rate of 0.1, depth of 2 for each decision tree (depth of higher than 2 makes GreedyMiser inapplicable for small budget values), squared loss function, and a total of 300 regression trees in the final additive classifier.

AskWorld (PQL) shows the result of Parametric Q-Learning (PQL) approach where the features for the linear approximation function are chosen as follows. First, each query in $q \in \mathbf{K}$ is associated with two boolean features: one which indicates if q is acquired as part of the knowledge of the MDP state, and the other indicates if q is selected as an action from the state. This allows us to represent both arguments of $Q(\mathcal{S}, a)$ function in the linear approximation function. The remaining budget in the state is also represented by one feature. We have tried other choices such as merging actions and states' features, representing the remaining budget by a

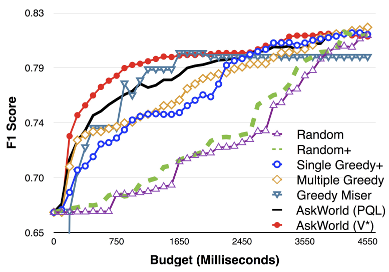

Figure 3: F1 scores comparing different systems against varying query-time budgets (in milliseconds). AskWorld(V*), the proposed system (top plot in the figure), outperform all other baselines.

set of boolean variables etc. However, the other alternative approaches either decreased or did not change performance. The result for **AskWorld (V*)** is obtained by abstracting MDP using $\delta = 5$, ordering queries using their information gain, and selecting top $k\%$ of features with non-zero information gain. In our experiments, we choose $k = 50\%$ which results in approximately 15M states in the MDP. Since AskWorld (V*) is trained only on a subset of queries, we may acquire all the queries using the learned policy and still have some leftover budget. In this case, we follow the greedy policy and acquire knowledge using queries that have not been already sent.

F1 curves comparing these systems are presented in Figure 3. Comparing the results for Random and Random+ in Figure 3, both algorithms perform equally, except that F1 score of Random algorithm is lower at the beginning. The main reason that Random algorithm is performing poorly at the beginning is that there are many zero elements in the feature vector of the classifier and therefore the constant value in the linear regression function of the SVM plays an important role in biasing the classification result.

Comparing the results of the two greedy algorithms, we can see that *Multiple Greedy* outperforms *Single Greedy+* since it learns different greedy orderings and learns separate SVM classifier for different budget values. Since *Multiple Greedy* approach requires iterating over the different budget values, its applicability is limited when the upper bound on the test time budget is very large (does not scale as the budget increases).

The figure also shows the result for the state-of-the-art approach: Greedy Miser [Xu *et al.*, 2012]. Comparing the result of the Greedy Miser with Single Greedy and Multiple Greedy algorithms, we can see that Greedy Miser performs poorly at the beginning (budget values less than 400), but then outperforms the other baselines for larger values of budget. GreedyMiser is using different classification technique (cascade of decision trees) and does not achieve the same F1 score as other approaches at higher budget levels.

From Figure 3, we observe that AskWorld(V*), our proposed approach, performs better than all the baselines. In contrast to *Multiple Greedy* approach, AskWorld is capable of handling varying query-time budgets (up to a specified upper bound) without retraining. Comparing the results of Greedy Miser and AskWorld(V*), we observe that AskWorld significantly outperforms Greedy Miser for smaller as well as higher values of the query budget, while achieving comparable performance at mid-level budgets.

6 Conclusion

In this paper, we present AskWorld, a novel system which is capable of providing knowledge-on-demand. We show that the accuracy of knowledge acquisition improves when the system is allowed to issue polling queries corresponding to non-query predicates in the Ontology. Even though this relaxation results in an explosion of polling possibilities, AskWorld is able to select the most informative ones within runtime budget constraints. To the best of our knowledge, AskWorld is the first knowledge-on-demand system of its kind which is capable of handling varying test-time budgets without model retraining. Through extensive experiments on real world datasets, we demonstrate AskWorld's capability in selecting most-informative queries within query-time runtime constraints, resulting in improved performance while achieving reduced model footprint.

Acknowledgments

This research has been supported in part by a gift from Google.

References

[Arnt *et al.*, 2004] Andrew Arnt, Shlomo Zilberstein, James Allan, and Abdel-Illah Mouaddib. Dynamic composition of information retrieval techniques. *JIIS*, 23(1):67–97, 2004.

[Azari *et al.*, 2004] David Azari, Eric Horvitz, Susan Dumais, and Eric Brill. Actions, answers, and uncertainty: A decision-making perspective on web-based question answering. *Inf. Process. Manage.*, 40(5):849–868, September 2004.

[Azari *et al.*, 2012] David Azari, Eric Horvitz, Susan T. Dumais, and Eric Brill. Web-based question answering: A decision-making perspective. *CoRR*, abs/1212.2453, 2012.

[Blum and Langley, 1997] Avrim L. Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artif. Intell.*, 97(1-2):245–271, December 1997.

[Bollacker *et al.*, 2008] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of SIGMOD*, 2008.

[Bourdev and Brandt, 2005] Lubomir Bourdev and Jonathan Brandt. Robust object detection via soft cascade. In *Proceedings of CVPR*, 2005.

[Chen *et al.*, 2013] Xi Chen, Qihang Lin, and Dengyong Zhou. In Sanjoy Dasgupta and David Mcallester, editors, *ICML*, volume 28, pages 64–72. JMLR Workshop and Conference Proceedings, May 2013.

[Friedman, 2000] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[Gao and Koller, 2013] Tianshi Gao and Daphne Koller. Active classification based on value of classifier. In *Proceedings of NIPS*, pages 1062–1070, 2013.

[Guyon *et al.*, 2002] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

[Kanani and McCallum, 2012] Pallika H. Kanani and Andrew K. McCallum. Selecting actions for resource-bounded information extraction using reinforcement learning. In *Proceedings of WSDM*, New York, NY, USA, 2012. ACM.

[Karayev *et al.*, 2013] Sergey Karayev, Mario Fritz, and Trevor Darrell. Dynamic feature selection for classification on a budget. In *ICML Workshop on Prediction with Sequential Models*, 2013.

[Karger *et al.*, 2011] David R. Karger, Sewoong Oh, and Devavrat Shah. Budget-optimal task allocation for reliable crowdsourcing systems. *CoRR*, abs/1110.3564, 2011.

[Lagoudakis *et al.*, 2003] Michail G. Lagoudakis, Ronald Parr, and L. Bartlett. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:2003, 2003.

[Mitchell *et al.*, 2015] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Proceedings of AAAI*, 2015.

[Raykar *et al.*, 2010] Vikas C. Raykar, Balaji Krishnapuram, and Shipeng Yu. Designing efficient cascaded classifiers: tradeoff between accuracy and cost. In *Proceedings of KDD*. ACM, 2010.

[Samadi *et al.*, 2012] Mehdi Samadi, Thomas Kollar, and Manuela M. Veloso. Using the web to interactively learn to find objects. In *Proceedings of AAAI*, 2012.

[Samadi *et al.*, 2013] Mehdi Samadi, Manuela Veloso, and Manuel Blum. Openeval: Web information query evaluation. In *Proceedings of AAAI*, 2013.

[Samadi *et al.*, 2015] Mehdi Samadi, Partha Talukdar, Manuela Veloso, and Tom Mitchell. AskWorld: Budget-Sensitive Query Evaluation for Knowledge-on-Demand. Technical Report CMU-CS-15-107, Carnegie Mellon University, 2015.

[Suchanek *et al.*, 2007] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of WWW*, 2007.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[Tan, 1993] Ming Tan. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, 13(1):7–33, 1993.

[Viola and Jones, 2001] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of CVPR*, volume 1, 2001.

[Weiss and Taskar, 2013] David J Weiss and Ben Taskar. Learning adaptive value of information for structured prediction. In *Proceedings of NIPS*, 2013.

[Xu *et al.*, 2012] Zhixiang Xu, Kilian Weinberger, and Olivier Chapelle. The greedy miser: Learning under test-time budgets. In *Proceedings of ICML*, 2012.

[Xu *et al.*, 2013] Zhixiang Eddie Xu, Matt J. Kusner, Kilian Q. Weinberger, and Minmin Chen. Cost-sensitive tree of classifiers. In *Proceedings of ICML*, 2013.

[Xu *et al.*, 2014] Zhixiang Xu, Matt J. Kusner, Kilian Q. Weinberger, Minmin Chen, and Olivier Chapelle. Classifier cascades and trees for minimizing feature evaluation cost. *Journal of Machine Learning Research*, 15:2113–2144, 2014.

[Zhang and Viola, 2007] Cha Zhang and Paul A. Viola. Multiple-instance pruning for learning efficient cascade detectors. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Proceedings of NIPS*, 2007.