Streaming Slot-Value Extraction for Wikipedia Entities at Web-Scale

Morteza Shahriari Nia, Christan Grant, Yang Peng, Daisy Zhe Wang*, Milenko Petrovic†

{msnia, cgrant, ypeng, daisyw}@cise.ufl.edu, mpetrovic@ihmc.us

Abstract

In this paper we address extracting slot values from internet pertinent to entities in wikipedia, which is the most popular web-based, colaborative multilingual encyclopieda on the internet. This is comparable to Freebase in a query-driven manner. Our contributions are two fold, first we design a system to efficiently find central documents on the internet that contain directly citable content regarding a given wikipedia entity; second, we design a system to extract certain slot-values of the entity from those documents. Our results demonstrate that the system is very efficient in the web scale nature of the problem: being highly memory and I/O efficient and that we can achieve high accuracy and recall for given slot values.

Introduction

Wikipedia.org (WP) is the largest and most popular general reference work on the Internet. The website is estimated to have nearly 365 million readers worldwide. An important part of keeping WP usable it to include new and current content. Presently, there is considerable time lag between the publication of an event and its citation in WP. The median time lag for a sample of about 60K web pages cited by WP articles in the *living_people* category is over a year and the distribution has a long and heavy tail Frank et al. (2013). Such stale entries are the norm in any large reference work because the number of humans maintaining the reference is far fewer than the number of entities.

Reducing latency keeps WP relevant and helpful to its users. Given an entity page, such as *wiki/Boris_Berezovsky_(businessman)*[1], possible citations may come from a variety of sources. Notable news may be derived from newspapers, tweets, blogs and a variety of different sources include Twitter, Facebook, iBlogs, arxiv, etc. The actual citable information is a small percentage of the total documents that appear on the web. The goal of this system is to read web documents and recommend citable facts for WP pages.

[1] http://en.wikipedia.org/wiki/Boris_Berezovsky_(businessman)

Previous approaches are able to find relevant documents given a list of WP entities as query nodes Balog and Ramampiaro (2013); Bonnefoy, Bouvier, and Bellot (2013); Dalton and Dietz (2013); Ji and Grishman (2011); McNamee et al. (2012). Entities of three categories *person*, *organization* and *facility* are considered. This task involves processing large sets of information to determine which facts may contain references to a WP entity. This problem becomes increasingly more difficult when we look to extract relevant facts from each document. Each relevant document must now be parsed and processed to determine if a sentence or paragraph is worth being cited.

Discovering facts across the Internet that are relevant and citable to the WP entities is a non-trivial task. Here we produce an example sentence from a new website:

"`Boris Berezovsky, who made his fortune in Russia in the 1990s, passed away March 2013.`"

First, we must realize that there are two *Boris Berezovsky* entities in WP, one a businessman and the other a pianist. Any extraction needs to take this into account and employ a viable distinguishing policy (entity resolution). Then, we match the sentence to find a topic such as *DateOfDeath* valued at *March 2013*. Each of these operations is expensive so an intelligent efficient framework is necessary to execute these operations at web scale.

In this paper we develop an efficient query-driven slot extraction for given WP entities from a time oriented document stream. Fact or slot extraction is defined as follows: match each sentence to the generic sentence structure of [*subject - verb - adverbial/complement*] Stevens (2008), where *subject* represents the entity and *verb* is the relation type (slot) we are interested in (e.g. Table 1). The third component, *adverbial/complement*, represents the value of the enity slot. In our example sentence, the entity of the sentence is *Boris Berezovsky* and the slot we extract is *DateOfDeath* with a slot valye of March 2013.

Our system contains three main components. First, we pre-process the data and build models representing the WP query entities. Next, we use the models to filter a large stream of documents so they only contain can-

Table 1: Ontology of Slots

Person	Facility	Organization
Affiliate		
AssociateOf		
Contact_Meet_PlaceTime		
AwardsWon	Affiliate	Affiliate
DateOfDeath	Contact_Meet_Entity	TopMembers
CauseOfDeath		FoundedBy
Titles		
FounderOf		
EmployeeOf		

didate citations. Lastly, we processes sentences from candidate extractions and return slot values. Overall, we contribute the following:

- Introduce a method to build models of WP name variations (Section **??**);
- Built a system to filter a large amount of diverse documents using a natural language processing rule-based extraction system (Section **??**);
- Extract, infer and filter entity-slot-value triples of information to be added to KB (Section **??**).

System

In this section, we introduce the main components of the system. Our system is built with a pipeline architecture in mind giving it the advantage to run each section separately to allow stream processing without blocking the data flow of components (Figure 1). The three logical components include sections on *Model* for entity resolution purposes, *Wiki Citation* to annotate citeworthy documents, *Slot Filling* to generate the actual slot values.

If we are looking to discover facts for a single WP entity, the first step is to extract aliases of the entity. We extract several name variations from the Wikipedia api and from the WP entity page. Also, if the entity type is *person* we can change the order of user names to increase coverage (e.g. 'Boris Berezovsky' → 'Berezovsky, Boris'). Next, we iterator over documents in the stream and filter out all document that are not central to the entity. To extract relevant facts we perform pattern matching in each sentence that matches the entity based on a dictionary of patterns. If the sentence can activate on of the patterns in the dictionary we emit this sentence as a candidate contribution to the WP entity. With the candidate set, we infer new facts from the set and clean up the set by removing the set that violate a list of constraints such as duplicates. Lastly, we try to infer a new set of facts based on a list of rules.

Entity Model

We use Wikipedia API to automatically retrieve aliases. The API allows us to requests pages that redirect users to an entity page. For example, if a WP user tries to access the *William Henry Gates* they are sent to the page for *Bill Gates* therefore we treat these two names as

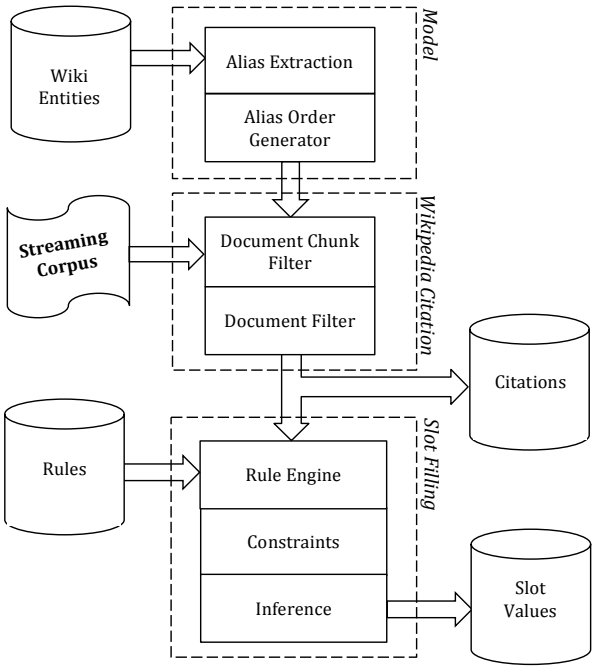

Figure 1: System Architecture. Components are logical groups noted with dotted boxes.

aliases. To extract more aliases is we parse the HTML source of a WP entity page. Then using regular expressions we extract the bold phrases of the initial paragraph as aliases. Based on our observation this is a very accurate method and it provides several entities of WP with more inline aliases from the wiki page itself. As an example, for 'Boris Berezovski' which was mentioned early-on, there is an alias of 'Boris Abramovich Berezovsky' given in bold in the wiki page which will be obtained via thi method.

We pass the full set of *person* entities through rules for generating proper name orders. This module produces various forms of writing entity names and titles. For example, *Bill Gates* can be written as *Gates, Bill.* This allows the system to capture various notation forms of aliases that appear in text documents.

Wikipedia Citation

The goal of this section is to use the models created to discover a set of documents that are relevant to the WP entity. As a stream of documents come in we first perform a string match between the model aliases and document text. We use this technique as a first filter with confidence because previous work has stated non-mentioning documents have a low chance of being citable in Wikipedia Frank et al. (2013). Given our large number of aliases we can be confident if an alias does not appear in a document it does not need to be cited.

Our stream system receives documents in the from of chunk files. Each chunk file contains thousands of documents. This corpus of documents is processed by a two-layer filter system referred to as *Document Chunk Filter* and *Document Filter*. The purpose of these fil-

ters is to reduce I/O cost while generating slot values for various entities. Document Chunk Filter removes the chunk files that do not contain a mention of any of the desired entities. Each chunk file may contain thousands of documents — each document is expensive to process. The Document Filter removes documents that do not contain a mention of an entity. This two-level filter allows us to perform detailed slower processing over a smaller set of documents. Not all chunk files contain mention of the entities so filtering out large chunk files early saves no I/O and processing. Document Chunk Filter discards non-mentioning chunk files and promotes chunk files as soon as an entity mention is found. The document filter additionally notes the sentences that contain entity mentions. This data is passed to the Slot Filling system.

Slot Filling

Streaming Slot Filling (SSF) extracts fact values from sentences according to a list of pattern. Table 1 list the slot relationships that we looks to extract. In Figure 1 we refer to this task as *Slot Filling*.

SSF reads documents filtered by the Wikipedia Citation step and fetches and tags sentences containing WP entities. All entities are extracted from the document using a natural language processing tool[2]. In the next section, we describe how WP entities are matched against the set of patterns. Following, we discuss out approach to inference over the extracted facts.

Algorithm 1 Slot Value Extraction Pseudocode

List of entities $\mathcal{E} = \{e_0, \ldots, e_{170}\}$
List of patterns $P = \{p_0, \ldots, p_{|P|}\}$
List of documents containing entities $\mathcal{S} = \{s_0, \ldots, s_{|\mathcal{S}|}\}$

```
for si ∈ S do
  for sentence ∈ si do
    for entity ∈ E do
      if Contains(sentence, entity) then
        for pattern ∈ P suitable for entity do
          if Satisfies(sentence, pattern) then
            Emit(sentence, pattern)
```

Rule Engine A pattern is as a record representing knowledge going to be added to a WP entity. A pattern $\mathcal{P}$ is represented as a five-tuple $\mathcal{P} = \langle p_1, p_2, p_3, p_4, p_5 \rangle$.

The first value, p_1 represents the type of entity. These entity types are in the set $\{\texttt{FAC}, \texttt{ORG}, \texttt{PER}\}$ where `FAC` represents a type of facility, `ORG` represents an organization and `PER` represents a person. The p_2 represents a slot name. A list of slot names is present in Table 1. The third element p_3 is the pattern content — a string found in the sentence that identifies a slot name. The extractor looks specifically for a pattern content. The pattern evaluator uses a direction (`left` or `right`) found in p_4 to explore sentence. The final element p_5 represents the slot value of a pattern. The type of slot value may be the entity type labeled by the named entity extractor, a noun phrase (`NP`) tagged by a part of speech tagger[3] or a phrase described in the pattern list.

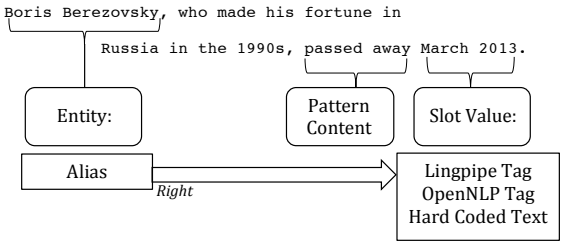

Figure 2: Pattern matching for rule evaluation. The slot value is on the right side of the entity. The pattern context discovered is 'passed away' with a value of 'March 2013'.

Figure 2 contains the example sentence from the introduction labeled by the rule engine.

There are three types of patterns distinguished by different types of slot values in the patterns. The matching methods using these three types of patterns are implemented according to the different structures of slot values.

Type I. This pattern type is driven by the entity type. For example, in pattern ⟨`PER, FounderOf, founder, right, ORG`⟩ the `PER` tag means the entity we are finding slot values for a PER entity; `FounderOf` means this is a pattern for FounderOf slot. *founder* is the anchor word we are match in a sentence; `right` means that we are going to the right part of the sentence to match the pattern and find the slot value; ORG means the slot value should be a ORG entity to the right of the entity.

Type II. This pattern type, after finding a matching pattern content, looks for a noun phrase (NP) that is representative of the slot value. For example, pattern ⟨`PER, AwardsWon, awarded, right, NP`⟩ is looking for a noun phrase after the *awarded* that may represent an award. Titles and awards are not named entity tagged hence the use of the part of speech tagger to fetch the noun phrases.

Type III. These type of facts are best discovered by hard coding the slot values. Examples of these include time phrases: ⟨`PER, DateOfDeath, died, right, last night`⟩. In this pattern, the phrase *last night* is exactly searched in the text last night to the right of the term *died*. This intuition behind this pattern is in news articles that report events in the past relative to the article. For example, an article will mention that a person died 'last night' instead of mentioning precise

[2]Lingpipe (available in the dataset) provides entity tags and in-document coreference. `http://alias-i.com/lingpipe/`

[3]OpenNLP was used for part of speech tagging. `http://opennlp.apache.org/`

date-time information. Additionally, part of speech taggers and name entity extractors did not label these terms such as *last night* as a DATE entity.

Constraints and Inference what specifically about the algorithm causes duplicates and errors? — I asked Yang to elaborate on this as he is more familiar but still not sure what's causing those issues

The fact extractions match patterns that are general and highly susceptible to noisy. The data contains duplicates and incorrect extractions. We define rules to read ordered sets of facts to sanitize the output. The input is processed in time order, in a tuple-at-a-time fashion to allow rules to discover noisy slots that appears in close proximity. We define two classes of rules: *deduplication* and *inference* rules.

The output contains many duplicate entries. As we read the list of extracted slots we create rules to define "duplicate". Duplicates can be present in a window of rows; we use a window size of 2 meaning we only be adjacent rows. Two rows are duplicates if they have the same exact extraction or if the rows have the same slot name and a similar slot value or if the extracted sentence for a particular slot types come from the same sentence.

New slots can be deduced from existing slots by defining inference rules. For example, two slots for the task are "FounderOf" and "FoundedBy". A safe assumption is these slot names are biconditional logical connectives with the entities and slot values. Therefore, we can express a rule "X FounderOf Y" $\leftrightarrow$ "Y FoundedBy X" where X and Y are single unique entities. Additionally, we found that the slot names "Contact_Meet_PlaceTime" could be inferred as "Contact_Meet_Entity" if the Entity was a FAC and the extracted sentence contained an additional ORG/FAC tag. We also remove erroneous slots that have extractions that are thousands of characters in length or tool small. Errors of extracting long sentences can typically be attributed to poor sentence parsing of web documents. We have some valid "small" extractions. For example, a comma may separate a name and a title (e.g. "John, Professor at MIT"). But such extraction rules can be particularly noisy, so we check to see if the extracted values have good entity values.

Evaluation

We evaluate the effectiveness of extracting slot values for 139 entities. We look at the baseline coverage for entities and slot names we present in a 500M page snapshot of the English web. We estimate the precision and recall of our extractions over several extracted fact.

Our system was developed on a 32-core server described in Table 2. Each document is annotated using a name entity extraction and in document coreference. A bundle of documents are serialized into chunks and encrypted. The total size of the data after compression and encryption is 4.5TB [4]. Data is ordered into 11952 date-hour buckets ranged from 2011-10-05-00 (5th of October 2011, 12am) until 2013-02-13-23 (13th of Feburary 2013, 11pm). The first four months of data (October 2011 - February 2012) is for training purposes, and we use this portion for rule and pattern creation and tuning. The data set contains text from several web page types as listed in Table 3.

Table 2: Benchmark Server Specifications

Spec	Details
Model	Dell xxx 32 cores
OS	CentOS release 6.4 Final
Software Stack	GCC version 4.4.7, Java 1.7.0_25, Scala 2.9.2, SBT 0.12.3
RAM	64GB
Drives	2x2.7TB disks, 6Gbps, 7200RPM

Table 3: Document Chunks Distribution

Document Type	# of Documents	Slots Found
Arxiv	10988	
Classified	34887	
Forum	77674	
Linking	12947	
Mainstream News	141936	
Memetracker	4137	
News	280629	
Review	6347	
Social	688848	
Weblog	740987	

We develop 172 extraction patterns covering each slot-name/entity-type combinations. Out of the 500 M documents and 139 entities we found 158,052 documents containing query entities, 17,885 unique extracted slot values for 8 different slots. We did not get any results from 31 entities missing and 4 slots.

In Table 4 we performed two samples of a baseline submission and estimate the correctness of the extraction. Add here how the baseline is different from the final submission We our methods produced an accuracies of 54% and 55%. We classify the errors into two sets, an incorrect entities and incorrect extractions. We found 15% and 17% incorrect entity names and we identified 27% and 30% incorrect value extracted across all entities and slot types. The majority of errors were regarding poor slot value extraction patterns and incomplete aliases.

Probably should add again here a reason why our method is different from the baseline. Using our methods, we provide we provide more detailed statistics, as displayed in Table 5 and Table 6. Table 5 shows the recall for each slot name. (define 'entity coverage' here) The slot name *Affiliate* has was extracted the most number of times and *CauseOfDeath* was extracted the

[4]TREC Knowledge base acceleration stream corpus is available `http://aws-publicdatasets.s3.amazonaws.com/trec/kba/index.html`

Table 4: Sampled accuracy of the results of the extracted facts.

	Correct	**Incorrect Entity name**	**Incorrect Value**
Sampling #1	55%	17%	27%
Sampling #2	54%	15%	31%

fewest with 0 instances found matching the pattern list. *AwardsWon* contained the next fewest with 38 instances found. Affiliate is a generic slot name, using patterns to extract these relationships is difficult because almost any two items can be affiliated in some way. Define the affiliate slot name. Our patterns for this relationship led to noisy results. On the other hand for more precise slot names our accuracy increases but we have less recall. Table 6 addresses the relative accuracy measure per slot value. *AssociateOf* has the highest accuracy with 63.6% and *Affiliate*, *Contact_Meet_PlaceTime* and *EmployeeOf* have the lowest with lowest of 1%, 1% and 5% accuracy respectively.

Discussions

Table 5 show a varied distribution of extracted slot names. Some slots naturally have more results than other slots. For example, AssociateOf and Affiliate have more slot values than DateOfDeath and CauseOfDeath, since there are only so few entities that are deceased. Also, some patterns are more general causing more extractions. For example, for Affiliate, we use *and*, *with* as anchor words. These words are more common than *dead* or *died* or *founded* in other patterns.

As part of future work regarding enhancing precision, we can focus on fixing the wrong entities found, remove noisy tags by the Lingpipe or use more accurate NLP taggers such as Stanford NLP (Due to the fact that Stanford NLP is very slow we avoided using it, as due to speed its use was out of question) and finally use better patterns to enahnce results matched by the patterns. On the other hand to increase recall we need to find better aliases and go for more resources to discover other ways that an entity might be addressed (e.g. twitter id, website, etc), add powerful patterns that can capture more cases of slot values. We would also use entity resolution methods and other advanced methods to improve the accuracy and recall of entity extraction part.

For slot extraction, to improve the performance, we need: 1) Using multi-class classifiers instead of pattern matching method to extract slot values in order to increase both recall and accuracy for slots "Affiliate", "AssociateOf", "FounderOf", "EmployeeOf", "FoundedBy", "TopMembers", "Contact_Meet_Entity" and so on. 2) For special slots, like "Titles", "DateOfDeath", "CauseOfDeath", "AwardsWon", using different kind of advanced methods, e.g. classifiers, matching methods. 3) Using other NLP tools or using classifiers to overcome the drawbacks of the LingPipes inaccurate tags. The first and second tasks are the most important tasks

Table 5: Recall Measure: Generic slot names like affiliate had the most recall, compared to less popular slot names e.g. DateOfDeath

Slot Name	**Instances Found**	**Entity Coverage**
Affiliate	108598	80
AssociateOf	25278	106
AwardsWon	38	14
CauseOfDeath	0	0
Contact_Meet_Entity	191	8
Contact_Meet_PlaceTime	5974	109
DateOfDeath	87	14
EmployeeOf	75	16
FoundedBy	326	30
FounderOf	302	29
Titles	26823	118
TopMembers	314	26

Table 6: Accuracy Measure: Accuracy of AffiliateOf was the best and Affiliate applied poorly due to ambiguity of being an affiliate of somebody/something

Slot Name	**Correct**	**Wrong Entity**	**Incorrect Value**
Affiliate	1%	95%	5%
AssociateOf	63.6%	9.1%	27.3%
AwardsWon	10%	10%	80%
CauseOfDeath	0%	0%	0%
Contact_Meet_Entity	21%	42%	37%
Contact_Meet_PlaceTime	5%	20%	85%
DateOfDeath	29.6%	71%	25%
EmployeeOf	5%	30%	65%
FoundedBy	62%	17%	21%
FounderOf	50%	0%	50%
Titles	55%	0%	45%
TopMembers	33%	17%	50%

we need to do.

Some of the entities are not popular. For example, a 'Brenda Weiler' Google search result has 860,000 documents over the whole web. For our small portion of the web it might make sense. The histogram of the entities shows that more than half of the entities have appeared in less than 10 StreamItems. A good portion have appeared only once.

Alltogether, We experimented through different tools and approaches to best process the massive amounts of data on the platform that we had available to us. We generate aliases for wikipedia entities using Wiki API and extract some aliases from wikipedia pages text itself. We process documents that mention entities for slot value extraction. Slot values are determined using pattern matching over coreferences of entities in sentences. Finally post processing will filter, cleanup and infers some new slot values to enhance recall and accuracy.

We noticed that some tools that claim to be performant for using the hardware capabilities at hand sometimes don't really work as claimed and you should not always rely on one without a thorough A/B testing of performance which we ended up in generating our in-house system for processing the corpus and passing through the filter. Furthermore, on extracting slot values, pattern matching might not be the best options but definitely can produce some good results at hand. We have plans on generating classifiers for slot value extraction purposes. Entity resolution on the other hand was a topic we spent sometime on but could not get to stable grounds for it. Entity resolution will distinguish between entities of the same name but different contexts. Further improvements on this component of the system are required.

We sampled documents from the training data period to generate an initial set of patterns. We then use these patterns to generate results. By manully looking at these results, we prune some patterns with poor performance and add more patterns that we identified from these results. We use several iterations to find the best patterns. We found that it is very time consuming to identity quality patterns.

Acknowledgements

We would like to thank John Frank and TREC for providing the data set. Christan Grant is funded by a National Science Foundation Graduate Research Fellowship under Grant No. DGE-0802270.

References

Balog, K., and Ramampiaro, H. 2013. Cumulative citation recommendation: Classification vs. ranking. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, 941–944. New York, NY, USA: ACM.

Bonnefoy, L.; Bouvier, V.; and Bellot, P. 2013. A weakly-supervised detection of entity central documents in a stream. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, 769–772. New York, NY, USA: ACM.

Dalton, J., and Dietz, L. 2013. Bi-directional linkability from wikipedia to documents and back again: Umass at trec 2012 knowledge base acceleration track. *TREC'12*.

Frank, J. R.; Kleiman-Weiner, M.; Roberts, D. A.; Niu, F.; Zhang, C.; Ré, C.; and Soboroff, I. 2013. Building an entity-centric stream filtering test collection for trec 2012. In *21th Text REtrieval Conference (TREC'12)*. National Institute of Standards and Technology.

Ji, H., and Grishman, R. 2011. Knowledge base population: Successful approaches and challenges. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, 1148–1158. Stroudsburg, PA, USA: Association for Computational Linguistics.

McNamee, P.; Stoyanov, V.; Mayfield, J.; Finin, T.; Oates, T.; Xu, T.; Oard, D. W.; and Lawrie, D. 2012. Hltcoe participation at tac 2012: Entity linking and cold start knowledge base construction. *TAC KBP*.

Stevens, S. 2008. Introduction to sentence patterns. In *Tutoring and testing at UHV*. University of Houston - Victoria.