Streaming Fact Extraction for Wikipedia Entities at Web-Scale

Morteza Shahriari Nia, Christan Grant, Yang Peng, Daisy Zhe Wang
Data Science Research Lab
University of Florida
Gainesville, Florida 32611

Milenko Petrovic
Institute for Human and Machine Cognition
15 SE Osceola Ave
Ocala, Florida 34471

Abstract

Wikipedia.org is the largest online resource for free information and is maintained by a small number of volunteer editors. The site contains 4.3 million english articles; these pages can easily be neglected, becoming out of date. Any news-worthy event may require an update of several pages. To address this issue of stale articles we create a system that reads in a stream diverse web documents and recommends facts to be added to specified Wikipedia pages. We developed a three-stage streaming system that creates models of Wikipedia pages, filters out irrelevant documents and extracts facts that are relevant to Wikipedia pages. The systems is evaluated over a 500M page web corpus and 139 Wikipedia pages. Our results show a promising framework for fast fact extraction from arbitrary web pages for Wikipedia.

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 [**?**]. 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. However, the actual citable information is a small percentage of the total documents that appear on the web. To help WP editors, a system is needed to parse through terabytes of documents and select facts that can be recommended to particular 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 [**?**, **?**, **?**, **?**, **?**]. Entities of three categories *person*, *organization* and *facility* are considered. This work 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 webpage:

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

After parsing the sentence, we must first note that there are two entities named *Boris Berezovsky* 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 efficient framework is necessary to execute these operations at web scale.

In this paper, we introduce an efficient fact extraction system or given WP entities from a time-ordered document stream. Fact extraction is defined as follows: match each sentence to the generic sentence structure of {*subject — verb — adverbial/complement*} [**?**]. The first *subject* represents the entity (WP 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 associated slot. In our example sentence, the entity of the sentence is *Boris Berezovsky* and the slot we extract is *DateOfDeath* with a slot value of *March 2013*. The resulting extraction containing an entity, slot name and slot value is a *fact*.

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

Table 1: The set of possible slot name for each entity type.

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

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

System

In this section, we introduce the main components of the system. Our system is built with a pipeline style architecture 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 are divided into sections entitled *Model* for entity resolution purposes, *Wikipedia Citation* to annotate cite-worthy documents, and *Slot Filling* to generate the actual slot values.

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.org 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' $\rightarrow$ 'Berezovsky, Boris'). Next, we iterate over documents in the stream and filter out all documents that do not explicitly contain a string matching the list of entities. To extract relevant facts we perform pattern matching over each sentence that matches the entity based on a dictionary of patterns. If a sentence activates one of the patterns in the dictionary we emit this sentence as a candidate contribution for the WP entity. With the candidate set, we infer new facts from the set and clean up the set by removing the set of values that violate a list of constraints such as duplicates.

Entity Model

We use the Wikipedia.org API to 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* entry they are sent to the page for *Bill Gates* — we treat such redirects as aliases. To extract more aliases we parse the HTML source of a WP entity page. Using regular expressions we extract the bold phrases of the initial paragraph as aliases. This method provides several inline aliases from the wiki page. In WP page for the businesman 'Boris Berezovski' , there is a mention of 'Boris Abramovich Berezovsky' given in bold in the wiki page which obtained by regular expression extraction.

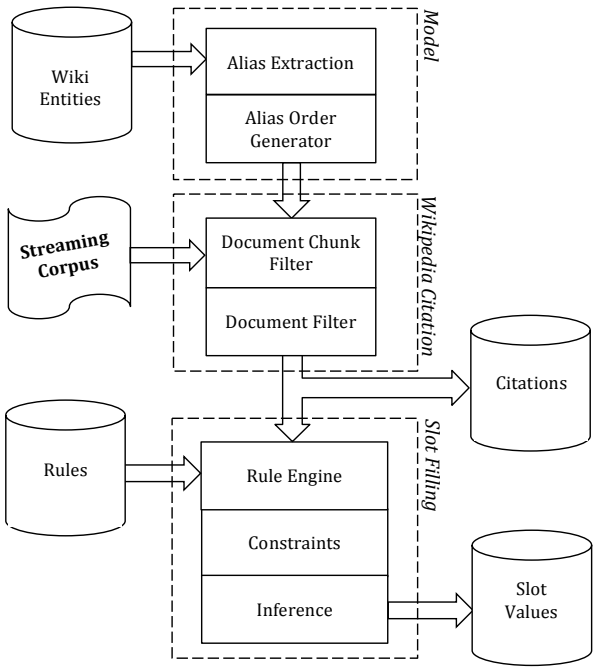

Figure 1: System Architecture. Components are logical groups noted with dotted boxes.

We pass the full set of *person* entities through rules for generating alternate name orders. This module produces various forms of expressing 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 states non-mentioning documents have a low chance of being citable in Wikipedia [**?**]. Given our large number of aliases we can be confident that if an alias does not appear in a document it does not need to be cited.

Our system streams in documents in the form 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 filters 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 on

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 patterns. Table 1 lists 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 our 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 a template of a fact to be extracted and added to a WP entity. Patterns are used to find and extract facts from text. 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. 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 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 contains the example sentence from the introduction labeled by the rule engine. The matching pattern is ⟨`PER, DateOfDeath, passed away, right, NP`⟩. In the figure, 'Boris Berezovsky' matches as an alias

[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/

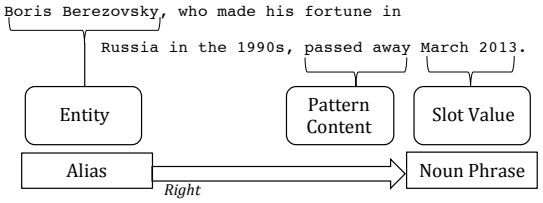

Figure 2: Pattern matching for rule evaluation. The slot value is on the right side of the entity. The pattern context discovered is 'passed away' with a value of 'March 2013'. This conforms to type II of our pattern categories.

for WP entity *wiki/Boris_Berezovsky_(businessman)*, DateOfDeath is the slot name, 'passed away' is the content of the pattern, direction is 'Right' and the actual value of the slot is 'March 2013' which is a Noun Phrase.

There are three types of patterns, each distinguished by different types of slot values (p_5) 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 is a PER entity; `FounderOf` means this is a pattern for FounderOf slot. *founder* is the word we match in a sentence - the content of pattern occuring in the 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 types 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*. The 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 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

Our data set contains some duplicate webpages, webpage texts with similar content, and some of the entity tags are incomplete. This causes some duplicates or highly similar content in the extracted list. We implement a filter to remove duplicates or the fact extractions that match patterns that are general and highly susceptible to be 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.

We develop 172 extraction patterns covering each slotname/entity-type combinations. Out of the 500M 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.

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

Table 2: Benchmark Server Specifications

Spec	**Details**
Processor	32 core AMD Opteron^{TM} 6272
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**
Arxiv	10988
Classified	34887
Forum	77674
Linking	12947
Mainstream News	141936
Memetracker	4137
News	280629
Review	6347
Social	688848
Weblog	740987

In Table 4 we performed two samples of a baseline and estimate the correctness of the extractions. The first was addressing the overall performance measures of the system, e.g. precision and recall. The latter experiment was performed over an enhanced version of the system; we included the aliases from WP API, the alias generation process, and some additional patterns. We produced accuracies in range 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.

After enahncing the system via better and more extraction patterns we provide more detailed statistics, as displayed in Table 5 and Table 6. Table 5 shows the recall for each slot name. Entities can have different coverages across the entire web, some were more popular (William H. Gates) or less well known such as (Stevens Cooperative School). Similarly, slot names have various coverages for example *Affiliate* is more probable across the entities when compared to *AwardsWon*. The slot name *Affiliate* has was extracted the most number of times; *AwardsWon* contained the next fewest with 38 instances found. , affiliations of an organization the following cases have been enumerated [**?**]:

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%

An affiliate relationship can be defined in three general ways [**?**]:

- A relationship consisting solely of the two groups interacting in a specific event context is not enough evidence to constitute a religious/political affiliation;
- Former political or religious affiliations are correct responses for this slot;
- Any relation that is not of parent-child form; a sub-organization is not an affiliate its parent organization but rather a Memberof.

Affiliate is a generic slot name; extracting affiliate relationships is difficult because the actual relationship must be determined. Our patterns for this relationship led to noisy results.

However, less ambiguous slot names (AssociateOf) obtained higher accuracy but we have recall. We developed patterns that explicitly expressed these relationships but we did not create enough patterns to express all forms of those slot names.

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.

Discussion

Table 5 shows the distribution of extracted slot names. The number of extraction between slot names vary greatly. Some slots naturally have more results than other slots. For example, DateOfDeath and CauseOfDeath have some of the fewest entities because only a few entities are deceased. Some patterns use common words as as part of their patterns causing more extractions. For example, Affiliate looks for common words like *and* and *with* as part of the pattern content. These words are more common than *dead*, *died* or *founded* in other patterns.

Some of the entities are popular and appear at a greater frequency in the data set. For example, a 'Corn Belt Power Cooperative' Google search results in 86,800 documents, where as 'William H. Gates' returns 3,880,000 documents. We observed that more than half of the entities appear in less than 10 documents in the data set; a large portion have appeared only once. This magnificient change in coverage support the viability of our seach and filter schemes.

The system pipeline architecture is an efficient method of processing the stream of data. Each hour of in the corpus contains and average of 380 MB of compressed data. It takes and hour for the system to extract facts from 140 hours worth of data fom the KBA corpus.

Conclusion

In this paper we described an approach to perform fact extraction over one of the largest data sets to date. We generate aliases for Wikipedia entities using an 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 tagged entities in sentences. Finally post processing will filter, cleanup and infers some new slot values to enhance recall and accuracy.

We sampled documents from the training data period to generate an initial set of patterns and then use these patterns to generate results. After manually examining the results, we prune patterns with poor performance and add patterns that may add to extraction coverage. We use several iterations to find the best patterns.

We look to continue exploration of streaming extraction models over this large data set. Our models are simple and provide a great baseline framework to develop and compare innovative techniques.

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
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%
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%

Acknowledgements

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

References