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 **??**. 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 . 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 specific relevant task 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 from the Internet that are relevant and citable to the WP entities is a non-trivial task. For example, take a sentence from the internet `Boris Berezovsky made his fortune in Russia in the 1990s when the country went through privatisation of state property and 'robber capitalism', and 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 (slot) 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 timely stream of documents on internet as they are being created. Slot or fact extraction is formally 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 we are interested in (e.g. Table 1). If a sentence matches these two components of the sentence pattern, *Adverbial/Complement* would be returned as the other side of the relation (which we refer to as slot value). Slot value extraction is a challenging task in current state of the art Knowledge Bases (KB). Popular graphical KB such as Freebase or DBPedia keep data in structured format where entities are connected via relationships (*Verb*) and the associated attributes (*Adverbial/Compelement*). Our system can be used to auto-

Table 1: Ontology of Slots

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

matically populate such KBs or even fill-in the information boxes at entity WP page itself.

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 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:

- Introduce a method to build models of name variations
- Built a system to filter a large amount of diverse documents
- 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 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 cite-worthy documents, *Slot Filling* to generate the actual slot values.

To walk you through the steps we take, assume we only care about on single WP entity, the first step is to extract aliases of the entity. We use several approaches to get as many viable aliases as possible. Then we look into the stream of content that is being generated on the internet, apply two levels of filtering to finally end up with the documents are central to that entity. To extract the relevnat slot values we perform pattern matching in each sentence or coreferent sentence to see if we can find a match. As a match is found from the content of the sentence to the patterns that we have generated regarding slot name, the associated slot value is extracted as a final result.

Model

We use Wikipedia API to get these aliases automatically. This is done by retrieving backlink references (redirects of a wiki entity), e.g. William Henry Gates is an alias for Bill Gates in WP as a backlink reference.

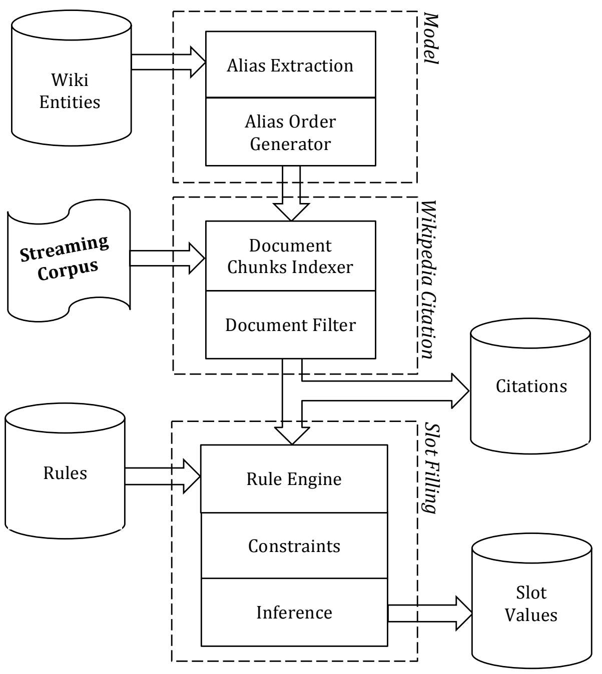

Figure 1: System Architecture. Components are logical groups noted with dotted boxes.

Unfortunately this is not good enough and to enhance recall we need more aliases. To have better use of a wiki page we parse HTML DOM of the page, then use regular expressions to extract the bold phrases of the first paragraph as alias of the actual entity. Based on our observation this is a very accurate heuristic and provides us with lots of famous aliases of the entities. As an example of when this might not wirk, is that there might be occasions that some other topic is written in bold typesetting in the first paraph apart from the entity aliases itself but these are very rare.

Once aliases are available we pass them through rules of generating proper name orders which will produce various forms of writing a name. As a basic example Bill Gates can be written as Gates, Bill also. This will allow the system to capture various notation forms of aliases. We refer to this part as *Alias Order Generator*.

Wikipedia Citation

The main goal of this section is to have an aggregate list of documents that are worthy of being cited in a Wikipedia page. We perform exact string matching and treat all the documents that mention an entity equally likely to be citable. One of the reasons for this is that there have been observations of how non-mentioning documents have a low chance of being citable in Wikipedia Frank et al. (2013). So we take on that and ignore non-citing documents.

Our pipeline of processing the corpus consists of a two layer indexing system referred to as *Document Chunks Indexer* and *Document Filter*. The purpose of these indexes is to reduce I/O cost while generating slot values for various entities. Document chunks Indexer

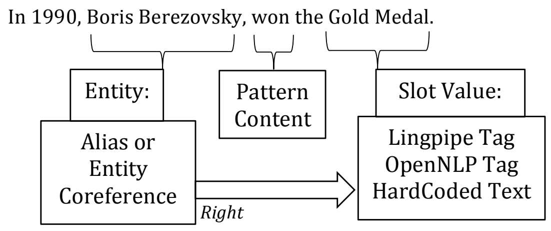

Figure 2: Pattern Matching with Slot Value on the Right Side of Entity.

will generate indexes of the chunk files that contain a mention of any of the desired entities. Document Filter on the other hand will index documents that contain a mention of a given entity respectively. This two level indexing will eliminate the need to process each and every chunk file/document for every entity. The reason for splitting this task into two steps is that not all chunk files contain any mention of the entities and we want to get rid of them as soon as possible. Document Chunks Indexer which discards non-mentioning chunk files and will stop further processing a chunk file as soon as it finds a mention there. Each chunk file can contain up to thousands of documents which can be so time consuming if we were to process them in our Java base code. Processing StreamItems on the other hand is done in Java with ideas in mind for later on extensibility by adding other Java libraries.

Slot Filling

The purpose of SSF is to extract proper values for relations of interest, which can be found in Table 1. In Figure 1 we refer to this as *Slot Filling*.

Slot filling is done by pattern matching documents with manually produced patterns for slots of interest. The way we do this is by observing a sentence that has a mention of the entity or one of its coreferences. An anchor word in the sentence related to the slot name is located and we match either left or right of the anchor word for potential slot values.

In the data set, we are given a date range of documents as training data. Instead of building a classifier we use pattern matching methods to find corresponding slot values for entities. Pattern matching is simple to manipulate results and implement. Additionally, a classifier approach is more difficult to evaluate and explain results due to the lack of proper training data.

With the documents indexes generated by the Wikipedia Citation, we first fetch the sentences containing entites by using alias names and coreference information provided by Lingpipe tags[2]. Then use these senteces to match patterns and when patterns matched, generate SSF results.

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

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 streamitems containing entities $\mathcal{S} = \{s_0, \ldots, s_{|\mathcal{S}|}\}$

for $si \in \mathcal{S}$ do
    for $sentence \in si$ do
        for $entity \in \mathcal{E}$ do
            if Contains($sentence$, $entity$) then
                for $pattern \in P$ suitable for $entity$ do
                    if Satisfies($sentence$, $pattern$) then
                        Emit($sentence$, $pattern$)
```

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. FAC, ORG and PER are Lingpipe entity types. 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. This is a string found in the sentence. The extractor looks for this exact string or pattern in a sentence. The pattern evaluator uses a direction (left or right) found in p_4 to explore sentence. The final element p_5 represent the slot value of a pattern. This The type of slot value may be the entity type tagged by Lingpipe, a noun phrase (NP) tagged by OpenNLP[3] or a hard-coded phrase. For these three kinds of patterns, we implement them in different ways accordingly. Next, we explain the patterns with more details, an example of which can be found in Figure 2.

Types of patterns 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 implemenented according to the different information and structures of slot values.

Type I. This pattern type is driven by the slot value type, a pattern tagged by Lingpipe. For example, pattern $\langle$PER, FounderOf, *founder*, right, ORG$\rangle$. PER means that 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.

Type II. This pattern type is unique because it only looks for a slot value tagged as noun phrase (NP) by OpenNLP. For example, pattern $\langle$PER, AwardsWon, *awarded*, right, NP$\rangle$. This pattern can be interpreted as that we are looking for a noun phrase after

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

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

the *awarded* since that noun phrase may represent an award. Titles and awards are usually not the Lingpipe entities, hence the use of the OpenNLP noun phrase chunker to fetch the noun phrases.

Type III. Some relations are best discovered by hard coding the slot values. Examples of these include time phrases: ⟨`PER, DateOfDeath,` *`died`*`, right,` *`last night`*⟩. In this pattern, *last night* means we are looking for exactly the phrase *last night* to the right of *died*. This pattern is inspired by the intuition that in news articles, people often mention that somebody died last night instead of mentioning the accurate date information and Lingpipe tends not to tag phrases like *last night* as a DATE entity.

Constraints - Inference The output of streaming slot value extraction is noisy. The data contains duplicates and incorrect extractions. We can define rules to sanitize the output only using the information present at this stage. The input file is processed in time order, in a tuple-at-a-time fashion to minimize the impact on accuracy. 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" equals "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 erronious slots that have extractions that are several pages 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

The purpose of evaluation is to measure the effectiveness of extracting slot values for various entities, metrics such as precision and recall would be considered. We perform random samplings to extract these. Our system was developed on a 32-core server described in Table 3. The corpus is a snapshot of the web in English. Each document is annotated using lingpipe and is called StreamItem, a bundle of StreamItems are put together and serialized as Apache Thrift objects, then compressed using xz compression with LempelZivMarkov chain algorithm (LZMA2) and finally encrypted using GNU Privacy Guard (GPG) with RSA asymmetric keys. The total size of the data after XZ compression and GPG encryption is 4.5TB and just over 500M StreamItems s3. Data is stored in directories the naming of which is date-hour combination: from 2011-10-05-00 (5th of October 2011, 12am) until 2013-02-13-23 (13th of Feburary 2013, 11pm), which consists of 11952 date-hour combinations. The first four months of data (October 2011 - February 2012) is for training purposes, and we use this portion to lookup sample lines and for system parameter settings purposes. This corpora consists of various media types the distribution of which can be found in Table 2. To have a sense of the scale of objects and compression as an example a 6mb gpg.xz files would become 45 mb thrift objects which can contain a couple of thousand StreamItems depending on their size. Some of the documents have null values for their annotation fields. The source code of our system is stored as an open source project where enthusiasts can also contribute to git, also the relevant discussion mailing list is accessible here goo.

We have 172 extraction patterns covering each slot-name/entity-type combinations. Our final submission was named *submission_infer*. Our results are as follows: Document extraction using query entity matching with aliases, sentence extraction using alias matching and co-reference. Slot extraction using patterns, NER tags and NP tags. 158,052 documents with query entities, 17885 unique extracted slot values for 8 slots and 139 entities, 4 slots and 31 entities missing.

On the performance of our initial submission run we performed random sampling via two processes, the results of which are according to Table 4. You can view that we have had an accuracy of around 55%, and about 15% wrong entity identified and 30% incorrect value extracted across all entities and slot types. Most of our issues for this submission were regarding poor slot value extraction patterns and incomplete aliases whih were tried to be mtigated later on. For our final submis-

Table 2: Document Chunks Distribution

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

Table 3: 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 4: Initial Performance Measure

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

sion, we provide a more detailed statistics, which has been elaborated in Table 5 and Table 6. Table 5 shows the extent of search outreach for each slot name. You can see that *Affiliate* has been the slot name with highest hits and *CauseOfDeath* our lowest hit with 0 instances found matching our patterns, after that *AwardsWon* has been the next with 38 instances found. Affiliate is a very generic term and extracting real affiliates can be quite challenging using the extraction patterns provided. This can lead to noisy results. On the other hand for more precise terms our accuracy increases but we have less recall. Table 6 addresses the relative accuracy measure per slot value. There you can view that we have had the highest accuracy of 63.6% for *AssociateOf* and the lowest of 1% - 5% for *Affiliate*, *Contact_Meet_PlaceTime* and *EmployeeOf*.

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.

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%

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 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 inhouse system for processing the corpus and generating the index. 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

Christan Grant is funded by a National Science Foundation Graduate Research Fellowship under Grant No. DGE-0802270.

References

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.

Gatordsr opensource project. https://github.com/cegme/gatordsr.

Gatordsr mailing list. https://groups.google.com/forum/#!forum/gatordsr.

Trec kba stream corpora. http://aws-publicdatasets.s3.amazonaws.com/trec/kba/index.html.

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