GatorDSR: University of Florida Knowledge Base Acceleration Notebook

Morteza Shahriari Nia, Christan Grant, Yang Peng, Daisy Zhe Wang*
{msnia, cgrant, ypeng, daisyw}@cise.ufl.edu

Milenko Petrovic†
mpetrovic@ihmc.us

October 10, 2013

Abstract

In this paper we will present the system design and algorithm adopted by the GatorDSR team, University of Florida to efficiently process TREC KBA 2013 — SSF track. Here we will describe the system as well as the details the algorithms used to extract slot values for the given slot name. Scalability, efficiency, precision and recall are the major goals of this work, given the overly limited time limitation and available computational resources.

1 Introduction

An important challenge in maintaining Wikipedia.org (WP), the most popular web-based, collaborative, multilingual encyclopedia in the internet, is to make sure its contents are up-to-date. Presently, there is considerable time lag between the publication date of cited news and the date of an edit to WP creating the citation. 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 [1]. Also, the majority of WP entities have updates on their associated article much less frequently than their mention frequency. Such stale entries are the norm in any large knowledge base (KB) because the number of humans maintaining the KB is far fewer than the number of entities. Reducing latency keeps KBs such as WP relevant and helpful it users.

Given an entity page, possible citations may come from a variety of sources. The actual citable information is a small percentage of the total documents that appear on the web. We develop a system to read streaming data and filter out articles that are candidates for citations. Given these documents, we extract the pairs of information in the article that is recommended for citations for each knowledge base page. add details

Our system contains three main components. First, we pre-process the data and build models representing the knowledge base entries. Next, we use the models to filter a stream of documents so they only contain candidate recommendations. Lastly, we processes sentences from candidate extractions and return specific information to be cited. add details

We built this system as part of a formal task described in Section 2. The data set from this task drop most our our design systems. Overall, our contributions are the following:

- we introduce a method to build models of name variations (Section 4.1);
- we built a system to filter a large amount of diverse documents (Section 4.2);
- we extract entity-slot-value triples of information to be added to KB (Section 4.3);
- we filter the final results using deduplication and inference (Section 4.4);
- we self-evaluate our results over a 4.5 TB data set (Section 5).

*University of Florida, Gainesville, Florida, USA
†Institute for Human and Machine Cognition, Ocala, Florida, USA

2 KBA Task Background

The National Institute of Standards (NIST) hosted the Text REtrieval Conference (TREC) — Knowledge Base Acceleration challenge in 2013. The task contains two main sections designed for this track, Cumulative Citation Recommendation and Streaming Slot Filling. Due to the importance of knowledge bases, both of these tracks aim to accelerating populating them, hence the title Knowledge Base Acceleration (KBA). Below we describe each of these tracks and their purposes.

2.1 Cumulative Citation Recommendation (CCR)

For this track, assessors were instructed to "use the Wikipedia article to identify (disambiguate) the entity, and then imagine forgetting all info in the WP article and asking whether the text provides any information about the entity" [1]. Documents are divided according if an entity is mentioned and a relevance level to the entity.

More specifically, a document may have a mention or be without a mention.

- **Mention:** Document explicitly mentions target entity, such as full name, partial name, nickname, pseudonym, title, stage name.
- **Zero-mention:** Document does not directly mention target. Could still be relevant, e.g. metonymic references like "this administration" → "Obama". See also synecdoche. A document could also be relevant to target entity through relation to entities mentioned in document – apply this test question: can I learn something from this document about target entity using whatever other information I have about entity?

The relevance of a document to a query is split into the four classifications.

- **Garbage:** not relevant, e.g. spam.
- **Neutral:** Not relevant, i.e. no info could be deduced about entity, e.g., entity name used in product name, or only pertains to community of target such that no information could be learned about entity, although you can see how an automatic algorithm might have thought it was relevant.
- **Relevant:** Relates indirectly, e.g., tangential with substantive implications, or topics or events of likely impact on entity.
- **Central:** Relates directly to target such that you would cite it in the WP article for this entity, e.g. entity is a central figure in topics/events.

2.2 Streaming Slot Filling (SSF)

The task is that given certain WP or Twitter entities (wiki/twitter URLs) and certain relations of interest (given in Table 1), extract as many triple relations as possible (hence, slot filling). This can be used to automatically populate knowledgebases such as free-base or DBPedia or even fill-in the information boxes at Wikipedia. Below, you can view some examples of what it means to fill in a slot value; in each example there is a sentence of interest that we wish to extract slot values from, an entity that the slot value is related to, and a slot name which can be thought of as the topic of the slot value:

Example 1: "Matthew DeLorenzo and Josiah Vega, both 14 years old and students at Elysian Charter School, were honored Friday morning by C-SPAN and received $1,500 as well as an iPod Touch after winning a nationwide video contest."

Entity: http://en.wikipedia.org/wiki/Elysian_Charter_School
Slot name: Affiliate
Possible slot values: "Matthew DeLorenzo", "Josiah Vega"
Incorrect slot values: "C-SPAN", "iPod Touch"

Example 2: "Veteran songwriters and performers Ben Mason, Jeff Severson and Jeff Smith will perform on Saturday, April 14 at 7:30 pm at Creative Cauldron at ArtSpace, 410 S. Maple Avenue."

Entity: http://en.wikipedia.org/wiki/Jeff_Severson
Slot name: Affiliate
Possible slot values: "Ben Mason", "Jeff Severson", "Jeff Smith"
Incorrect slot values: "Creative Caldron", "Art Space"

Example 3: "Lt. Gov. Drew Wrigley and Robert Wefald, a retired North Dakota district judge and former state attorney general, unveiled the crest Friday during a ceremony at the North Dakota Capitol."

Entity: http://en.wikipedia.org/wiki/Hjemkomst_Center

Slot name: Contact_Meet_PlaceTime

Slot value: "Friday during a ceremony at the North Dakota Capitol"

In *streaming* slot filling, we are only interested in new slot values that were not substantiated earlier in the stream corpus. In Table 1 you can view some slot values and their types, the comprehensive description of which can be found at [2] and [3]. The details of the metric for SSF will favor systems that most closely match the changes in the ground truth time line of slot values. This is done by searching for other documents that mentioned an entity and exactly matched the slot fill strings selected by the assessors.

In the rest of the paper, we address the following material. In Section 2, we sketch the main components of the system and their purposes. In Section 3, describe the details of how we designed and implemented certain components critical to the behavior of the system. Section 4 addresses the performance and the results we achieved. Section 5, goes through a discussion of the challenges we had to produce reliable results over the massive amount of information available, and how we overcome these issues. For similar approaches regarding last year's track you can refer to [4].

3 System Overview

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 for performing preprocessing to prepare the required data, Cumulative Citation Recommendation to annotate cite-worthy documents, Streaming Slot Filling to generate the actual slot values and PostProcessing to increase precision/recall. TREC provides a streaming data set and entities from Twitter or WP to query.

Preprocessing. The contest prohibits Wikipedia entities to have any manual aliases being added and only allows automatic ways. We use Wikipedia API backlink references (redirects to a wiki entity) as aliases of the entity. We also extract aliases from within the text of a wiki entity, which will be described in Section 4.1. This whole process is referred to as *Alias Extraction*. The manual extraction rule is lifted and participants are allowed to manually add aliases for Twitter entities. This is allowed because the Twitter website does not provide an example page from the beginning of the document stream. This process of extracting aliases for Twitter entities is referred to as *Manual Alias Extraction*.

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. This will allow the system to capture various notation forms of aliases. We refer to this part as *Name Order Generator*.

Cumulative Citation Recommendation. The main goal of CCR 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 in former TREC KBA reports [1] there were observations of how non-mentioning documents have a low chance of being citable in Wikipedia. So we take on that and ignore non-citing documents.

Table 1: Ontology of Slot Name Categories

PER	FAC	ORG
Affiliate		
AssociateOf		
Contact_Meet_PlaceTime		
AwardsWon	Affiliate	Affiliate
DateOfDeath	Contact_Meet_Entity	TopMembers
CauseOfDeath		FoundedBy
Titles		
FounderOf		
EmployeeOf		

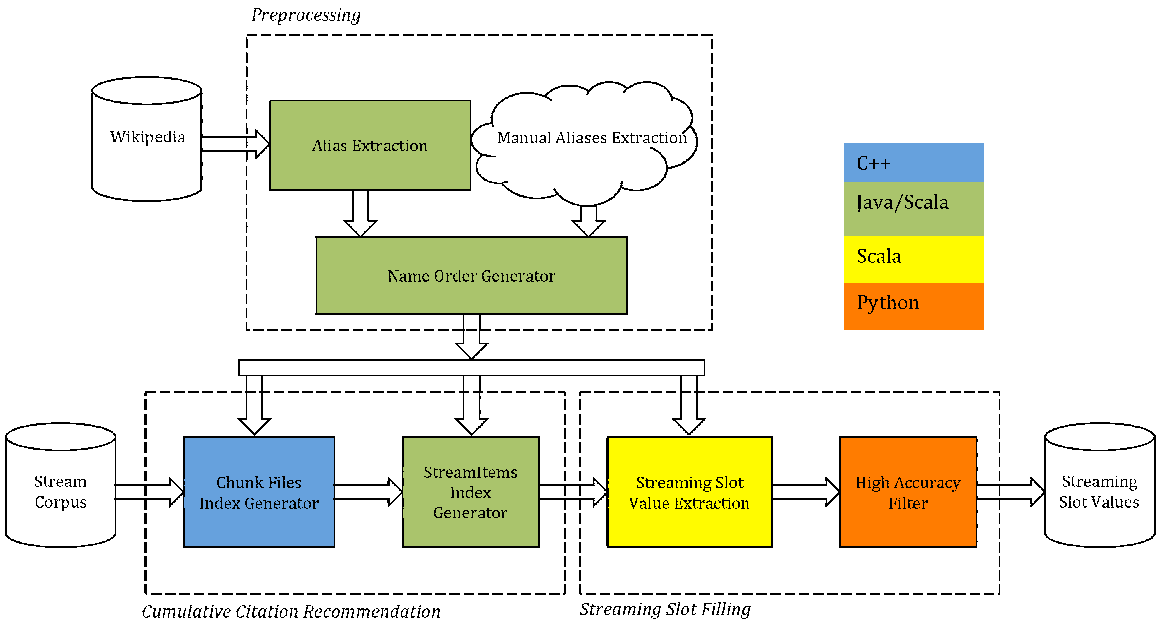

Figure 1: The GatorDSR team System Architecture. Components are logically groups with dotted boxes. Color details the programing language implementation of components.

Streaming Slot Filling. The purpose of SSF is to extract proper values for relations of interest, which can be found in Table 1. This is called Stream Slot Filling because data is being generated as time goes on and for each extraction we should only consider current or past data. In Figure 1 we refer to this as *Streaming Slot Value Extraction*. Stream 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.

Post Processing Algorithm. The SSF output of many extractions is noisy. The data contains duplicates and incorrect extractions. We can define rules to sanitize the output only using the information present in the SSF file. The 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 rules and inference rules. In our diagram we refer to this component as **High Accuracy Filter**.

4 Implementation

We extract aliases for entities from Wikipedia automatically both using API and using the actual page content, then apply pattern matching rules for slot value extraction. Our contribution is that we perform pattern matching that conforms to each slot value along with post-processings to eliminate noisy outputs.

4.1 Alias Generation

We use Wikipedia API to get some aliases automatically. This is done by retrieving backlink references (redirects of a wiki entity). 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. To consider other typical cases we consider some generic first name last name order swapping conventions such as Bill Gates $\rightarrow$ Gates, Bill. Meanwhile, William Henry Gates is an alias for Bill Gates in WP as a backlink reference. These kinds of aliases are also included in matching entities.

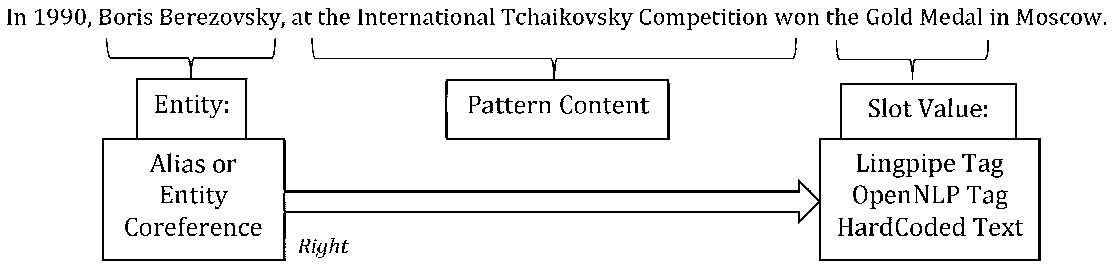

Figure 2: Pattern Matching with Slot Value on the Right Side of Entity.

4.2 Cumulative Citation Recommendation

Our pipeline of processing the corpus consists of a two layer indexing system referred to as *Chunk Files Index Generator* and *StreamItems Index Generator*. Chunk Files Index Generator will generate indexes of the chunk files that contain a mention of any of the desired entities. StreamItems Index Generator on the other hand will index StreamItems that contain a mention of a given entity respectively. This two level indexing will eliminate the need to process each and every ChunkFile/StreamItem 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. Chunk Files Index Generator which is written in C++ will discard 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 SIs 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.

4.3 Streaming Slot Value Extraction

In the data set, we are given 4 months of data (October 2011 - February 2012) 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, we can simply to explain and evaluate results.

The procedure to extract slots with the patterns is to first find Streamitems that contain the entities by using alias names of the entities. Next, inside these Streamitems, fetch the sentences which contain entities by using alias names and the coreference information provided by Lingpipe tags. Use these sentences to match existing patterns, and when patterns matched, generate SSF results.

Given a set of Streamitems $\mathcal{S} = \{s_0, \ldots, s_{|\mathcal{S}|}\}$
Set of patterns $P = \{p_0, \ldots, p_{|P|}\}$

1: **for** $si \in \mathcal{S}$ **do**
2: **for** $s \in si$ **do**
3: **if** *check_one_of_the_patterns* **then**
4: Emit(sentence, pattern)
5: **end if**
6: **end for**
7: **end for**

A pattern is defined as a record representing knowledge going to be added to a knowledge base. A pattern $\mathcal{P}$ is represented as $\mathcal{P} = \langle p_1, p_2, p_3, p_4, p_5 \rangle$.

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. 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. Thois The type of slot value may be the entity type tagged by Lingpipe, a noun

phrase (NP) tagged by OpenNLP or a hard-coded phrase. For these three kinds of patterns, we implement them in different ways accordingly. Next we will explain the patterns with more details, a brief example can be found in Figure 2.

Slot value type, tagged by Lingpipe: For slot FounderOf, we have this pattern: <PER, FounderOf, ''founder'', right, ORG>. PER means that the entity we are finding slot values for is a PER entity; FounderOf means this is a pattern for FounderOf slot. Founder is the anchor word we are trying to match in the sentence text; 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.

Slot value tagged as noun phrase by OpenNLP: <PER, AwardsWon, ''awarded'', right, NP>, NP meaning noun phrase. This pattern can be interpreted as that we are looking for a noun phrase after the "awarded" since that noun phrase could possibly represent an award. Since titles and awards are usually not the Lingpipe entities, we use the OpenNLP noun phrase chunker to fetch the noun phrases.

Slot value of time phrases, hard coded: <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.

We have found that there are three major classes of accuracy errors: wrong entities found, wrong tags by the Lingpipe, and finally wrong results matched by the patterns. We solve the first problem by using the Wikipedia or twitter information of the entities to get better alias names. And we use post-processing to reduce the second and third types of errors (Section 4.4).

4.4 High Accuracy Filter

The SSF output of many extractions is noisy. The data contains duplicates and incorrect extractions. We can define rules to sanitize the output only using the information present in the SSF file. The 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 rules 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.

5 Results

Our system was composed of a single node cluster, which we refer to as the *server*; the configuration of which is as 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 [5]. 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. 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 first portion of the data which ranges from October 2011 to February 2012 is considered as training data. The source code of our system is stored as an open source project where enthusiasts can also contribute to [6], also the relevant discussion mailing list is accessible here [7].

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 the first 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. For our final submission, 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. 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*.

6 Discussion & Future Works

In this section we address some of the challenges that we faced during the course of this project. We started off by trying to use the off-the-shelf tools for this task. Mainly we started by using Scala/Spark [8] to benefit from the parallelization performance there. Unfortunately Spark was not performant and the distributed reliability overhead was much more than tolerable. We moved on to use Scala parallelization framework itself, and unfortunately it did not satisfy our needs as well; there was excessive memory overhead on map-reduce jobs. We migrated the core of the system to Java Parallelism APIs and that was not good enough either, we still demanded better. So we built our own parallel system which in actual performance had the least of overhead, the least memory consumption and the most robust memory model which avoided unpredictable CPU stalls on garbage collections in processing the corpus.

As mentioned before, when we evaluate the results of slot extraction, we find three kinds of problems for accuracy: 1) wrong entities found; 2) wrong tags by the Lingpipe; 3) wrong results matched by the patterns. We also have recall problems: 1) not enough good alias names to find all the entities. 2) not enough and powerful patterns to capture all the slot values.

We will use entity resolution methods and other advanced methods to improve the accuracy and recall of entity extraction part.

Table 2: Document Chunks Distribution

# of Documents	**Document Type**
10988	arxiv (full text, abstracts in StreamItem.other_content)
34887	CLASSIFIED (spinn3r)
77674	FORUM (spinn3r)
12947	linking (reprocessed from kba-stream-corpus-2012, same stream_id)
141936	MAINSTREAM_NEWS (spinn3r)
4137	MEMETRACKER (spinn3r)
280629	news (reprocessed from kba-stream-corpus-2012, same stream_id)
6347	REVIEW (spinn3r)
688848	social (reprocessed from kba-stream-corpus-2012 plus extension, same stream_id)
740987	WEBLOG (spinn3r)

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: SSF Performance Measure on Initial Submission

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

Table 5: Recall Measure on Submission Infer

Slot Name	**Total instances of slot value found**	**# of entities covered by slot value**
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: SSF Accuracy Measure on Submission Infer

Slot Name	**Correct**	**Incorrect Entity name**	**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.

About 50% of twitter entities are not found by the system. One reason could be that those entities are not very famous. For example Brenda Weiler google search result has 860,000 documents over the whole web. For our small portion of the web it might make sense. If you pay attention to the histogram of the entities found you'll note that more than half of the entities have appeared in less than 10 StreamItems. A good portion have appeared only once. Which the document does not necessarily have good information for us.

A theory is that we are falling behind because we are using the cleansed (from HTML tags) version of the corpus. We believe there has been a reason that TREC has included the actual HTML document as well as the cleansed version. This definitely will convey some information to us. If it was as easy as reading some clean text they wouldn't bother including so much data for teams to be useless. So we guess is that we are missing some information from not using the actual document. And, we are looking for tokens with entity value set which will depend us dramatically on the accuracy of lingpipe, which is a fast algorithm but is not as good as other NLP tools can be e.g. Stanford NLP.

7 Conclusions

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. On twitter entities we extract aliases manually as it is part of the rule of the KBA track. 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 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.

Acknowledgements

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

References

[1] John R Frank, Max Kleiman-Weiner, Daniel A Roberts, Feng Niu, Ce Zhang, Christopher Ré, and Ian Soboroff. Building an entity-centric stream filtering test collection for trec 2012. In *21th Text REtrieval Conference (TREC'12)*. National Institute of Standards and Technology, 2013.

[2] Tac kbp slots. http://www.nist.gov/tac/2012/KBP/task_guidelines/TAC_KBP_Slots_V2.4.pdf.

[3] Ace (automatic content extraction) english annotation guidelines for events. http://projects.ldc.upenn.edu/ace/docs/English-Events-Guidelines_v5.4.3.pdf.

[4] Heng Ji and Ralph Grishman. 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, pages 1148–1158, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

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

[6] Gatordsr opensource project. https://github.com/cegme/gatordsr.

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

[8] Apache spark. http://spark.incubator.apache.org/.