The lilian Cold Start System for TAC 2016

Yuncong Wu
Lilianinfo.com / Floor 3, Building E, ShangDaGuoJi,
3100 HuTai Road, Baoshan, Shanghai
chinsnlia@gmail.com

Abstract

This paper describes the full pipeline KBP system implemented by lilianinfo.com for the TAC KBP 2016. In TAC KBP 2016, lilianinfo.com participated in one track: Cold Start Track and focused on English Cold Start KB Construction tasks. Our pipeline is based on RelationFactory, which include 4 parts : preprocessing, candidate generation, candidate validation and post-processing. In candidate validation part, besides the pattern and SVM model provided by RelationFactory, we propose a simple fuzzy match to improve the validation performance and show a considerable improvement in the final result.

1 Introduction

Knowledge Base (KB) plays an important role in many aspects (e.g., AI, Intelligent Q&A System). But it's difficult and inefficient to extract information from open-source data to construct or populate a Knowledge Base system. Hence, automatically Knowledge Base Population (KBP) was proposed. The goal of TAC KBP 2016 is to develop and evaluate technologies.

The Cold Start KBP track in TAC KBP 2016, which we participated, requires us to build up a Knowledge Base with given scheme and texts come from newswire and multi-post discussion forums. It's multi-lingual, but the system we perform only supports English.

The system we perform is a pipelined system, which contains four parts: preprocessing; candidate generation; candidate validation; postprocessing.

Our system is mostly based on RelationFactory[1] (Roth et al. ,2014), which contains distantly supervised classifiers (SVM) and patterns.

But traditional pattern-based relation extractors suffer from low recall (Angeli et al. ,2015). To improve the recall, New York University implied a fuzzy match in KBP 2015's system (He and Grishman ,2015), which can be treated as an impressive and effective tech.

It's also a tough work to summarize patterns from texts, which may require a large number of linguistics knowledge and many human resources. The RelationFactory shows us a new thought : they use distant supervision to extract some "raw" patterns and calculate a score for each "raw" pattern based on the frequency of the pattern. Finally, they manually summarize patterns from high score "raw" patterns and use the manually crafted pattern to extract relation. However, the manually crafted patterns get a low recall (about 6.8%) in high score "raw" patterns (i.e., "raw" patterns with a score higher than 0.95, with a maximum of 1). It shows that a manually crafted pattern can't make full use of the "raw" patterns.

But the "raw" patterns suffer from a terrible low recall. It's inefficient to extract information from them directly. So, we combining the thoughts from NYU and RelationFactory and come up with a strategy to make full use of the "raw" patterns. We use a fuzzy match to match the patterns. The NYU use an edit-distance-based algorithm to fuzzy the dependency path, which may only consider syntactic information in the sentence, but lost the seman-

[1] https://github.com/beroth/relationFactory

tics information (e.g., since “Chinese” and “German” are both nationality, hence, “PER, a Chinese per:title” may match the sentence like “PER, a German per:title”). It has been shown that word2vec (Mikolov et al. ,2013) done well in word embedding. So we perform a simple algorithm to do the fuzzy match based on word2vec.

The rest of paper is organized as follows. We first present our system in section 2 with detail descriptions of each module in system pipeline. Then section 3 shows the performance of the system on TAC 2016 KBP datasets in different aspects. Finally, in section 4, we draws some conclusions.

2 System Description

Figure 1 shows the overall architecture of our TAC KBP 2016 Cold Start system. In the first part, we process on the corpus and collect all needed information about entities and slot values. To extract slot values which type are string from corpus, we define 7 extra kind of entities, which is showed in tabular 2 and extract them with patterns.

Then the candidate generation part performed. This part will extract all possible (entity, slot, relation) tuple considering their type and distance. Only the closest entities with the corresponding type will be retrieved.

Then the third part comes. we use the model provided by RelationFactory to valid the candidates, which include an SVM model trained by distant supervision and distantly supervised pattern. Then we perform a fuzzy match to do further extraction.

The preprocessing part can be mainly divided into two parts: cross-document coreference and prediction combining. Then we can use the validated tuple (entity, slot, relation) to build up the Knowledge Base.

The detail description of all components in the pipeline is followed.

2.1 Preprocessing Corpus

2.1.1 extract posters/authors name & remove message in tag

We found that the posters and authors’ name should be taken account as PER type mentions. But it’s difficult for NER toolkit to detect them. So we perform a regex to extract their names. Besides, we find that the information in tag is mostly uselessly, but may do harm to the NER toolkit, so we remove all message in the text and simply replace them with equivalent space.

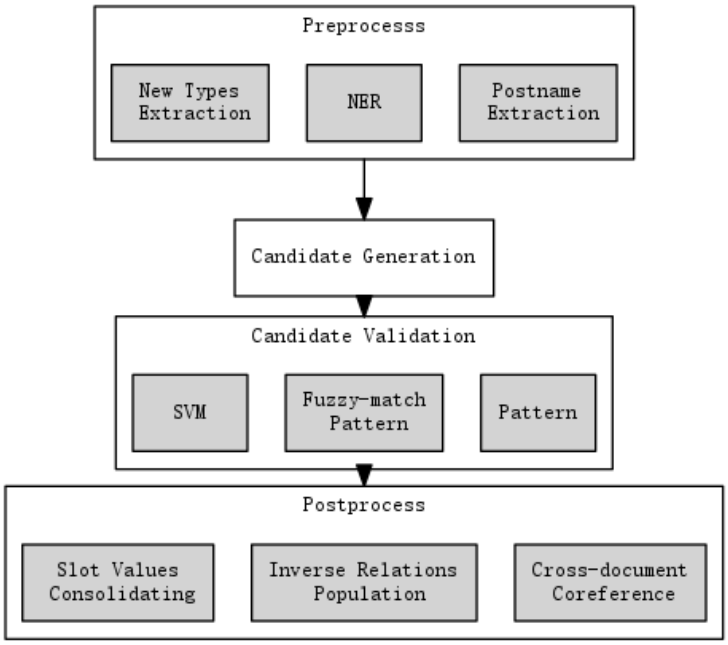

Figure 1: Architecture of the system

2.1.2 Named Entities Recognition

In KBP 2016, participators are required to perform a system which finds all names or nominal mentions of specific, individual PER, ORG, GPE, LOC, and FAC entities in the corpus. Our system use MITIE[2] to do the NER, only score higher than 0.3 will be considered as an entity. It can only support PER, LOC, and ORG 3 types, and fails to find NOM.

The MITIE may mistake some number or punctuation as part of the entities, so we remove all punctuation at the begins and ends of each entity, remove all one-character entities, and remove entity without any alphabet.

2.1.3 Dates Extraction

There are may kinds of dates information in the corpus, some are absolute (e.g., “August 17, 1926”, “1926-”, “08/17/1926”), some are relative (e.g., “a

[2]https://github.com/mit-nlp/MITIE

Type	Coressponding slots
Date	org:date_dissolved org:date_founded per:date_of_birth per:date_of_death
Number	org:number_of_employees_members per:age
Job	per:title
Religion	org:political_religious_affiliation
Url	org:website
CauseDeath	per:cause_of_death
Charges	per:charges

Table 1: The new-defined type and their coressponding slot. Note that, there are some string slots can be filled with the surface of a entity metion, like per{org}:alternate_names and per:origin, so it's unnecessary to define a new type.

week from today", "last year", "two weeks before last Monday"). We extract both of them by patterns. We use the document date as the current day to calculate the relative dates, and calculate partly (e.g., for those like "90 years ago", we will only take years into account, and produce XX/XX/1926).

2.1.4 GPE Extraction

The GPE can be divided into 4 levels: nationality, country, province and city.

We use GeoText[3], whose raw data extracted from geoname[4] to extract cities from text. The amount of all cities a large number, so only whose population more than 15,000 are taken into account. Some cities with strange name (e.g., "Same", "University") are removed manually.

The number of nationalities, countries and provinces is much smaller and has more alternate names, so, use GeoText may suffer from low recall. Hence, we grab all nationalities/countries name and their alternate name from wikidata[5] and use a regex to extract them.

2.1.5 Religions/Party/Jobs/Charges/CauseDeath Extraction

We use wikidata to collect all jobs/parties' name and use a regex to extract them.

The religions/charge/causedeath was extract by regex, using a proper word list from RelationFactory.

2.1.6 URL Extraction

Use urlextractor[6] to extract all url in the corpus.

2.1.7 Number Extraction

We first use regex to extract all word with number, i.e., words constructed by numeric word, and use semantic[7] to valid the number. All valided number will be retrieved.

2.2 Candidate Generation

This candidate generation should generation all possible (entity, slot, relation) tuple and retrieve their sentence for validation and provenance provided. It's obviously that neighbor entities are more likely to be related, so we assume only the closest entities with the corresponding type will be a candidate of (entity, slot, relation) tuple. Hence, we retrieve candidates for each sentence base on their types and positions.

2.3 Candidate Validation

2.3.1 SVM Model

The model is provided by RelationFactory, which uses Freebase[8] for distant supervision. The model takes only two features into account : the n-gram feature and sparse (or skip) n-grams feature.

2.3.2 Hand-Crafted Pattern

We used the set of prespecified rules from the RelationFactory relation extraction system to extract relation.

2.3.3 Fuzzy Match Pattern

We use spaCy[9] for word embedding. Two sentence's relative rate are defined like this:

$$dis_s(s1, s2) = \min_{for\ i\ in\ w1_n, w2_n} dis_w(w1_i, w2_i)$$

[3]https://github.com/elyase/geotext
[4]http://www.geonames.org/
[5]https://www.wikidata.org/
[6]https://github.com/lipoja/URLExtract
[7]https://github.com/crm416/semantic
[8]https://www.freebase.org
[9]https://spacy.io/

$s1$, $s2$ represent the sentence to be compared, $w1_n, w2_n$ are represented the word sequences for the two sentence. dis_w means the semantic distance between the words, which is calculated by spaCy.

But, there is still a problem in the definition of $w1_n$ and $w2_n$. A strictly matching may compare the sentence word by word, which may fail in some embarrassing situations. For instance, "Unite States" may be close to word "China", but word "Unite" will fail to match "China". To avoid these mismatching, we use a dynamic programming algorithm to do the fuzzy match.

The algorithm is shown in algorithm 1. $t1, t2$ is the tokens sequence of $s1, s2$, tokenized by split the sentence with space. n, m is the length of $t1, t2$, tr is a threshold to limit the length of a word (set as 2 in our system). $F_{i,j}$ stands for the maximum relative rate between the first i tokens in s1 and the first j tokens in s2. So $F_{n,m}$ is the relative rate between the sentence. Each time, we choose continuous tokens to form a word, which is $t1_{k,i}, t2_{l,j}$, calculate the relative rate if we chose the word in this way, then relax $F_{i,j}$.

The complexity of algorithm 1 is $O(nmtr^2)$, which may cause a hard run by comparing millions of candidates and ten thousand of patterns . We find that most of the candidates fail to match the patterns in the first word. But the algorithm continues calculating the relative rate even though the match has failed. To cut off the calculation in failed match, we implied an algorithm, using a double queue to consider the matched word set only.

The new algorithm is shown in algorithm 2. Q is a double queue, storing the available matches for updating other matches. In each iteration, we pick up an available match i, j and try to expand it by matching a new word-pair ($t1_{i,k}, t2_{j,l}$ in algorithm 2), and relax $F_{k,l}$ by this match. If the relaxed matching is an available matching, i.e. the new relative rate is higher than threshold T (set as 0.75 in our system), and $k,\ l$ is not in the updating queue Q, we push the new matching into the queue. This algorithm can be further optimized by using SLF (Smart Label First) and LLL (Large Label Last). But in this case, the update frequency is very small, so using an original version could get better performance.

Algorithm 1

$F_{0,0} = \inf$
for $i \leftarrow 1, n$ **do**
  for $j \leftarrow 1, m$ **do**
    for $k \leftarrow i - tr + 1, i$ **do**
      for $l \leftarrow j - tr + 1, j$ **do**
        $F_{i,j} = max(F_{i,j},$
$min(F_{k-1,l-1},\ dis_w(t1_{k,i}, t2_{l,j}))$
      end for
    end for
  end for
end for

2.4 Postprocessing

2.4.1 Cross-document Coreference

All the components performed before are in mention-levels, i.e., they only consider the surface form and textual context. To construct a Knowledge Base, we need to cluster the mention into entity node.

Clustering by wiki's "also known as". For type like GPE, which is completely extracted from a gazetteer. We cluster them by wikidata's "also known as" label.

Two-stage string similarity clustering. For PER, LOC, ORG. Intuitively, similar names are likely to refer to the same named entity. Hence, we perform a two-stage clustering algorithm, which mainly consider the mention's surface.

We use the method like Lodie's system in TAC2015 (Gao et al. ,2015) to compute string similarity: we use different measures for different named entity types: the Levenshtein distance (LD) (normalized by maximum length) for ORG and GPE; and the Jaro-Winkler (JW) distance (normalized by the longest common prefix) for PER.

To speed up the clustering, we use the algorithm as Lodie performed in TAC2015 to do the clustering.

The algorithm contains many rounds. First, all mention are in the "remaining pool". In each round, a mention in the remaining pool is chosen randomly as the central of a new cluster. Then for every other mention in the "remaining pool", we compute a string similarity score between them. All mention whose score passes a threshold T (set as 0.9 in our system), are added to this new cluster and

Algorithm 2

$F_{0,0} = \inf$
$Q \leftarrow (1, 1)$
while $Q \neq \emptyset$ **do**
  $i, j \leftarrow pophead(Q)$
  for $k \leftarrow i, i + tr - 1$ **do**
    for $l \leftarrow j, j + tr - 1$ **do**
      $tmp \leftarrow min(dis_w(t1_{i,k}, t2_{j,l}),$
$F_{i-1,j-1})$
      if $F_{k,l} < tmp$ **then**
        $F_{k,l} = tmp$
        if $F_{k,l} > T$ **then**
          if k, l **not in** Q **then**
            $Q.pushtail(k, l)$
          end if
        end if
      end if
    end for
  end for
end while

removed from remaining pool. The complexity to generate the first cluster in this algorithm is $O(n)$, which n stands for the total number of name mentions of a given type. But the following clustering will speed up, because the size of remaining pool can be reduced gradually during the processing, and finally lead to a complexity about $O((n/k)^2)$, where k refers to the threshold T, means the average size of a cluster, the higher the threshold set, the smaller the k we have (about 8 when T = 0.9).

However, the first stage of clustering often produces an over-cluster result, so we perform a second stage clustering.

For each cluster, we want to divide them into many smaller parts, which size cannot be larger than 10 percent of the origin clusters. We use the same algorithm as the first stage to do this.

2.4.2 Inverses Relations

For each validated candidate, we add their inverse relations as a new candidate. Finally, all validated candidates were sent to a consolidating process.

2.4.3 Consolidating Slot Values

Slot value consolidation involves selecting the best value in the case of a single-valued slot (e.g., per:date_of_birth), and the best set of values for slots that can have more than one value (e.g., per:children). All unreasonable candidates are first removed (e.g., a PER whose per:age slot is 1000). For the single-valued slot, we simply pick the slot value with the highest score (calculated from SVM and relative rate). For the multi-valued slot, we use a strategy like HLTCOE (Finin et al. ,2013). We associate two thresholds for the number of values that are selected the first $T1$ represents a number that is suspiciously large and the second $T2$ is an absolute limit on the number of values reported. Table 2.4.3 shows the thresholds we used for selecting. We first rank all candidates by their score, the first $T1$ candidates are selected directly. The rest $T2 - T1$ candidates are picked iff their score are higher than a threshold $T3$ (0.2 in our system).

relation	T1	T2
org:alternate_names	25	30
org:founded_by	25	30
org:member_of	18	22
org:members	25	30
org:parents	5	5
org:political_religious_affiliation	5	10
org:shareholders	25	30
org:subsidiaries	25	30
org:top_members_employees	8	10
per:alternate_names	25	30
per:charges	5	10
per:children*	8	10
per:religion*	2	3
per:cities_of_residence	7	12
per:countries_of_residence*	5	7
per:employee_or_member_of*	18	22
per:origin	2	3
per:other_family	17	20
per:parents*	5	5
per:schools_attended*	4	7
per:siblings*	9	12
per:spouse*	3	8
per:statesorprovinces_of_residence	6	8
per:title	5	8

Table 2: The number of values for some multi-valued slots are limited by two thresholds, relations with asterisks are refered to HLTCOE's TAC2015 system.

3 Result

Lilian submitted three runs to the CS KB task. These submissions are meant to estimate the perfor-

Table 3: Entity Discovery Result

strong_mention_match			strong_typed_metion_match			mention_ceaf			b_cubed		
P	R	F1	P	R	F1	P	R	F1	P	R	F1
0.854	0.558	0.675	0.766	0.500	0.605	0.718	0.469	0.567	0.803	0.347	0.484

Table 4: Slot Filling Result

RunID	hop0_P	hop0_R	hop0_F1	hop1_P	hop1_R	hop1_F1	All_P	All_R	All_F1
lilian1	0.1935	0.0225	0.0403	0	0	0	0.1765	0.0149	0.0275
lilian2	0.2647	0.0225	0.0414	0	0	0	0.2535	0.0149	0.0281
lilian3	0.2881	0.0212	0.0395	0	0	0	0.2742	0.0141	0.0268

mance of fuzzy pattern match.

A summary of three submissions is given below.

Lilian1 A system combines SVM model and pattern provided by RelationFactory and fuzzy pattern match.

Lilian2 A high precision system with a higher threshold (set T as 0.8 and T3 as 0.3). A system combines the SVM model and pattern provided by RelationFactory and fuzzy pattern match.

Lilian3 A system only includes SVM model and pattern provided by RelationFactory.

We use the same ED result for each run. The result of ED is showed in table 3, while the result of SF is showed in table 4.

It should be mentioned that we do not carry out the hop-1 typed cold start knowledge base construction because we do not keep the extraction of knowledge from the obtained facts.

From these results, we can conclude that a fuzzy pattern match algorithm can help to improve the performance of a CS KB system. A higher threshold should be set in our system since high threshold doesn't cause any decrease in recall while increasing the precision.

4 Conclusions

In this paper, we present the system for the Cold Start Track of the KBP 2016. The proposed system contains preprocessing, preprocessing, candidate generation, candidate validation and post-processing. We provide a fuzzy pattern match algorithm and it's proved to increase the performance of our system at this task.

References

Gabor Angeli, Victor Zhong, Danqi Chen, Arun Chaganty, Jason Bolton, Melvin Johnson Premkumar, Panupong Pasupat, Sonal Gupta, and Christopher D Manning. 2015. Bootstrapped self training for knowledge base population. In *Proc. Text Analysis Conference (TAC 2015)*.

Tim Finin, Dawn Lawrie, Paul McNamee, James Mayfield, Doug Oard, Nanyun Peng, Ning Gao, Yiu-Chang Lin, Joshi MacKin, and Tim Dowd. 2013. Hltcoe participation in tac kbp 2015: Cold start and tedl. In *Eighth Text Analysis Conference*, volume 22, page 2. NIST.

J Gao, Z Zhang, and AL Gentile. 2015. The lodie team (university of sheffield) participation at the tac2015 entity discovery task of the cold start kbp track. In *Proceedings of the 2015 Text Analysis Conference*. Sheffield.

Yifan He and Ralph Grishman. 2015. The nyu cold start system for tac 2015.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Benjamin Roth, Tassilo Barth, Grzegorz Chrupala, Martin Gropp, Dietrich Klakow, Shuly Wintner, Sharon Goldwater, and Stefan Rielzler. 2014. Relationfactory: A fast, modular and effective system for knowledge base population. In *EACL*, pages 89–92.