UIC-CSC: The Content Selection Challenge entry from the University of Illinois at Chicago

Hareen Venigalla
Computer Science
University of Illinois at Chicago
Chicago, IL, USA
hareen058@gmail.com

Barbara Di Eugenio
Computer Science
University of Illinois at Chicago
Chicago, IL, USA
bdieugen@uic.edu

Abstract

This paper described UIC-CSC, the entry we submitted for the Content Selection Challenge 2013. Our model consists of heuristic rules based on co-occurrences of predicates in the training data.

1 Introduction

The core of the Content Selection Challenge task is formulated as *Build a system which, given a set of RDF triples containing facts about a celebrity and a target text (for instance, a Wikipedia - style article about that person), selects those triples that are reflected in the target text.* The organizers provided training data consisting of 62618 pairs of texts and triple sets. The text is the introductory text $tf_{\mathcal{C}}$ of the Wikipedia article corresponding to celebrity $\mathcal{C}$; the set of triples $tr_{\mathcal{C}}$ concerning $\mathcal{C}$ was grepped from the Freebase official weekly RDF dump. It is important to note that we do not know which specific triples from $tr_{\mathcal{C}}$ are rendered in $tf_{\mathcal{C}}$.

A sample triple in the file is as follows:

```
ns:m.04wqr
ns:award.award_winner.awards_won
ns:m.07ynmx5
```

In the above triple, `ns:m.04wqr` is the subject id, of Marilyn Monroe in this case (`ns` denotes `namespace`); `ns:award.award_winner.awards_won` is the predicate and `ns:m.07ynmx5` is the object id of the award. Since this format is not readable, the organizers provided a script to transform the turtle file into a human readable form, where the object id is replaced by its actual value:

```
/award/award_winner/awards_won ``Golden
Globe Awards for Actress - Musical or
Comedy Film - 17th Golden Globe Awards
- Some Like It Hot - 1960 - earliye -
Award Honor'' /m/07ynmx5
```

In the following, we will refer to the first element of these expressions as the *predicate*. Our approach relies on heuristics derived from clustering predicates directly, or clustering them based on the co-occurrence of the argument of predicate p_i in a text tf and in turtle files tr that contain both p_i and another predicate p_j.

2 Deriving heuristic rules

We observed that in total there are 613 distinct predicates. Out of these 613 predicate, only 11 are present in over 40 percent of the files and only 19 predicates are present in over 10 percent of the files. This means that a large number of predicates are present only in a few files. This makes it harder to decide whether we have to include these predicates or not. Conversely, nearly 40 percent of text files only contain one or two sentences, which compounds the sparsity problem.

Predicate Clustering. In the first method, we generate predicate clusters by simply removing the leaf from each predicate expression. For example, `/people/person/place_of_birth`, and `/people/person/education` belong to the same cluster, labelled `/people/person` as they have the same parent `/people/person`. We found 35 such clusters. We then analyzed the frequency of each predicate p_i on its own, and conditional on other predicates in the same cluster: for example, how frequent `/people/person/education` is, and how often it occurs in those triple files, where `/people/person/place_of_birth` is also present.

Intersection on Arguments. For each predicate p_i, we compute the set of its intersection sets $IS_{i,j}$. Each set $is_{i,j}$ comprises all the turtle files $tr_{i,j}$ where p_i co-occurs with a second predicate p_j. For each $tr_{i,j}$, we retrieve the corresponding text file tf (recall that each turtle file is associated with one text file) and check whether the argument of

Proceedings of the 14th European Workshop on Natural Language Generation, pages 210–211,
Sofia, Bulgaria, August 8-9 2013.

p_i occurs in tf – this is indirect evidence that the text does include the information provided by p_i (of course this inference may be wrong, if this argument occurs in a context different from what p_i conveys). If the argument of p_i does occur in tf, we keep $tr_{i,j}$, otherwise we discard it. As above, we then proceed to compute the frequencies of the occurrences of p_i on its own, and of p_i when p_j is also present, over all the turtle files $tr_{i,j} \in is_{i,j}$ that have not been filtered out as just discussed.

Given these two methods, we derive rules such as the following:

```
IF /baseball/baseball_player/position ∈ tr_k
   AND
   /baseball/baseball_player/batting_stats
   ∈ tr_k
THEN
   select
   /baseball/baseball_player/position
```

The set of rules is then filtered as follows. On a small development set, we manually annotated which triples are included in the corresponding text files. We keep a rule if the F-measure concerning predicate p_i (i.e., concerning the triples whose predicate is p_i) improves when using the rule, as opposed to including p_i if it belongs to a set of frequent predicates.

We also need to deal with multiple occurrences of p_i in one single turtle file. Predicates such as `/music/artist/track` can have multiple instances, up to 30, in a certain tr_k, with different arguments; however, those predicates may occur far fewer times in the corresponding text files – because say tr_{MM} on Marilyn Monroe includes one triple for each of her movies, but the corresponding tf_{MM} only mentions a few of those movies. Hence, we impose an upper limit of 5 on the number of occurrences in the same turtle file, for a certain predicate to be included, for example:

```
IF      /music/artist/track
        AND its count ≤ 5
THEN    select /music/artist/track
```

3 Evaluation

Apart from our participation in the Challenge, we evaluated our system on a small test set composed of 96 pairs of text and turtle files, randomly selected from the data released by the organizers. This resulted in a total of 153 unique predicates (hence, about $\frac{1}{4}$ of the total number of distinct predicates). We manually annotated the predicates in the turtle files as present/absent in the corresponding text file.

We consider four domains:

1. *Basic facts*: general, very frequent information, such as `people/person/profession`, `people/person/nationality`.
2. *Books*: predicates whose root is `book`, like `book/author/works_written`, `book/book_subject/works`.
3. *Sports*: predicates whose root is a sport, like `baseball/baseball_player/position_s`, `ice_hockey/hockey_player/former_team`.
4. *Film and Music*: predicates whose root is `film` or `music`, like `/film/director/film`, `/music/artist/track`.
5. Television: predicates whose root is `tv`, like `/tv/tv_director/episodes_directed`.

As apparent from Table 1, the performance of our system varies considerably according to the domain of the predicates. Specifically, we believe that the exceedingly low precision for predicates of type `book, film & music, tv` is due to the sparseness of the data. As we noted above, 40% of the text files only include one or two sentences. Hence, our system selects many more predicates than are actually present in the corresponding text file.

Table 1: Performance on in-house test set

Domain	**P**	**R**	**F-score**
Basic Facts	79.83	51.25	62.40
Sports	79.84	49.22	60.90
Books	12.80	66.30	21.47
Film & Music	5.77	55.19	10.45
TV	5.46	43.36	9.70

4 Future Enhancements

UIC-CSC could be improved by more closely analyzing the features of the text files, especially the shortest ones: when they include only few sentences, which kinds of predicates (and arguments) do they include? For example, if only two movies are mentioned as far as Monroe is concerned, what else can we infer from the Monroe turtle file tr_{MM} about those two movies?