Internship Report

May - July 2016

Cognitive Dictionary Building and Sentence Template Matching for Unstructured Data

Author:
Ishu Dharmendra Garg
CS13B060
IIT Madras

Mentors:
Salil R. Joshi
Vikas V. Joshi
IBM Research Lab

IBM Research Labs
Bangalore India

1 Introduction

This document is a project report by Ishu Dharmendra Garg (CS13B060), a student at the department of computer science and engineering at IIT Madras enrolled in B.Tech, M.Tech Dual degree program (2013 Batch). He worked as a research intern at IBM Research Lab Bangalore India during May - July 2016. This report gives a summary of the work done by him as a part of the research project assigned to him during the internship. This report only gives the summary of the work and does not include details and extensions of the work done during the internship.

2 Abstract

Usually for the task of information retrieval and sentiment analysis we write rule annotators for mining information (structured data) mostly from unstructred data. Such annotators are usually written by a person hence writing such annotators usually requires a lot of human time and resources. Mostly the task requires a lot of language and domain expertise because of which a whole new set of annotators have to written when the task of information retrieval has to be performed on a different language and domain.

For the task a person always have to design some kind of templates (patterns), regular expressions to mine sentences, entities/relations or some useful information from the text. Also one needs complete dictionaries of various entities which are used in template matching.

Our aim is to reduce to manual labour in building dictionaries, designing different regular expressions and templates.

3 Definitions

- Dictionary: A set of tokens (words/phrases).
  For example: Organization {IBM, Box, Microsoft, ...}
- Template: A sequence of dictionaries.
  For example: ⟨Organization⟩ ⟨Hire⟩ ⟨Person⟩
- Concept: A structured representation of an entity type or relation that can be collectively captured using one or more templates.

4 Problem Statement

- Template Matching: Given a template and without having the complete dictionaries, mine all the sentences from the corpus that follow the template.
- Dictionary Building: Enrich the component seed dictionaries corresponding to the template.

For example consider the following input:

Template: ⟨ Company1, AquireVerb, Company2 ⟩

Seed Dictionaries

- Company1 = ⟨Boeing, Atlantic⟩
- AquireVerb = ⟨acquire⟩
- Company2 = ⟨Argon, Buckeye⟩

The output must be like:

Mined Sentence: Oracle which makes database and other acquired the java computer programming language and related when it bought Sun.

Mined Tokens: ⟨Oracle, bought, Sun⟩

Now we see that given a text corpus one should find all the sentences where some company has acquired or took over other company. Also it must extract the words that match (semantically similar) to the corresponding dictionaries in order to enrich them. Also one peculiar thing to notice is that none of the companies or the verb in the output tokens may be in the initial seed dictionaries.

5 Flow of work

Hundred of ideas were though, many of them were implemented but here we list the summary of the necessary steps we performed in order to achieve the task for the best method we found out.

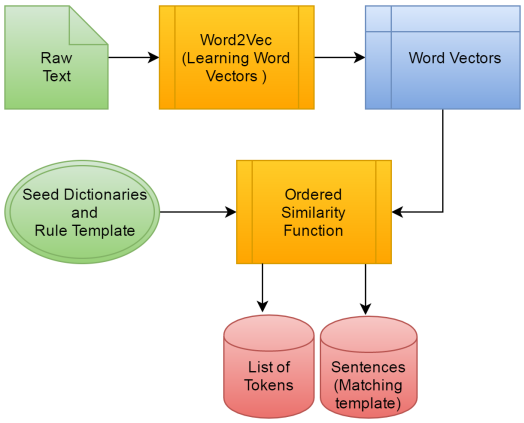

Figure 1: Model Architecture.

1. Decided what kind of pre-processing was required. Designed and coded the python programs in order to achieve the task.

2. Creating Data set: Text from millions of News articles in the xml format was first extracted and and then preprocessed accordingly.

3. Extensive research done to find out a process by which the schematic similarity of the word can found out without referring to the any dictionary. We found out a state of art machine learning model called word2vec (which uses shalllow neural networks). It uses context of the words in the sentences and assigns a fixed size vector to each word with the property that semantically or conceptually similar words are placed nearby in the vector space.

4. Trained the machine learning model and found out the optimal parameters for our purpose. After this step for each word we had a corresponding vector associated with it.

5. Designed various matching functions. The function takes a sentence and a template and gives a score that indicates how much the given sentence matches with the template. Different kind of matching functions were designed for different purposes.

6. Since the input data was huge, an optimal way of implementation was figured out so as to make the model fast enough for our purpose. Vari-

ous programming optimization techniques were used including dynamic programming.

7. Bug fixing and refining of the model was done. Some of new features were introduced in order to control the precision recall trade-off.

8. For benchmarking, a test set was prepared and then the model was tested against the common methods for comparison purposes.

6 Results and Novelty in work

Following are points that highlights our work

- Precision and Recall: Given the constraints on input (which can be unstructured text of any language) and seed dictionary size (5-10 words each), model outperformed many current techniques achieving a precision and recall of 0.76 and 0.65 respectively. The model was trained on AP news courpus and tested on the

- Domain Independence: The method works for any unstructured text in any language and any domain.

- Minimal inputs and human guidance: No need of labelled data or any kind of support from the language dictionary as the similarity between words is determined only from the text.

- Input dictionary size: The size of input dictionaries can be very small. Even $5 - 10$ words per dictionary is enough for the model to work.

- Fast: Shallow neural networks are used along with optimized matching functions makes the process really fast.

- Controlling hyper parameters: Matching functions designed such that precision-recall trade-off can be controlled using hyper parameters.

7 Tools Used

The list of the tools used are as follows:

- Python xml etree parsers for pre-processing.

- Stanford Natural Language Toolkit for pre-processing.
- Gensim Library for its word2vec implementation.
- Rest of the codes were self written using python language.