Interactive System for Creation of Notes

Martin NEMČEK*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
xnemcekm@stuba.sk

Abstract. We are overwhelmed by information from various topics. The challenge in education is to create notes which covers important subset of information. There are known methods to extract information from text. In this article we propose a system to extract the notes from text which are important for educational purpose, so it should create personalized notes for students. We use mainly syntactic text analysis. Notes are created by help of part-of-speech tags and dependencies between words in sentences. The outcome will be an interactive system for creating notes based on learned rules from user.

1 Introduction

Nowadays, we are overwhelmed with huge amount of data and information.

Computers are not able to understand information in natural language. In our proposed system the notes are created from sentences by extracting relevant information from them. We use syntactic analysis of sentences and extract relations and dependencies between words from these sentences. The final result of our proposed method are personalized notes. The user will be able to modify the automatically created notes. The system will then learn new rules for sentence to note transformation from these changes and takes them into account for the next time.

2 Summarization in natural language processing

Note creation is a type of text summarization which attempts to extract the most important and the most relevant information from text. However, it is more personalized and user-specific task than ordinary text summarization. In case of note creation there are the text transformation rules created by user. These rules are used to create notes that best fit particular user's needs.

Author in [4] defines a result of summarization as "A text that is produced from one or more texts, that conveys important information in the original text(s), and that is no longer than half of the original text(s) and usually significantly less than that.". Automatic summarization has three important aspects [1]:

---

* Bachelor study programme in field: Informatics
Supervisor: Miroslav Blšták, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

IIT.SRC 2015, Bratislava, April 28, 2016, pp. 1–8.

– Summaries may be produced from a single document / text or multiple documents / texts,

– Summaries should preserve important information,

– Summaries should be short.

Summarization systems take one or more texts as input and produce concise summary of the most important information in input. Finding the most important information presupposes the ability to understand the semantics of text. Writing a concise summary [3] requires the capability to reorganize, modify and merge information expressed in different sentences from input.

3 Our proposed system

3.1 Usage of dependencies in note creation

At Stanford University a tool named *StanfordNLP*[1] is being developed. The tool is composed of several software such as *Stanford Parser*, *Stanford POS Tagger*, *Stanford EnglishTokenizer* and *Stanford Relation Extractor*. These software focus on tasks of natural language processing. They are implemented primarily in Java, but are also available in other programming languages such as C#, PHP or Python.

One of the most popular tools for natural language processing is StanfordNLP[2]. It allows to extract dependencies. *Identification of dependencies* gives us simple description of grammatical relations between words in sentence. The dependency consist primarily from two tokens (words) - a governor token and a dependent token and also include the type of relation between those tokens. By applying dependency identification on sentence "Bell, based in Los Angeles, makes and distributes electronic, computer and building products." a dependency tree (see Figure 1) is created [2].

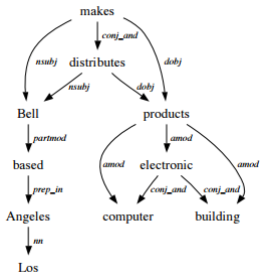

Figure 1. Dependency tree [2]

In oriented graph like the one in Figure 1 words represent nodes and edges are represented by relations between words.

In our example on Figure 2 there are relations between words in pairs. Between words *She* and *looks* is relation with name *nominal subject* (nsubj), between *looks* and *beautiful* is adjectival complement (acomp) and between words *very* and *beautiful* is relation called adverb modifier (advmod) [2].

[1] www.nlp.stanford.edu
[2] www.nlp.stanford.edu

Dependency beside other thing is composed of governor and dependent token and relation between them. In Figure 2 is shown that among others there is a dependency which governor token is *looks*, dependent token is *She* and relation is *nsubj*.

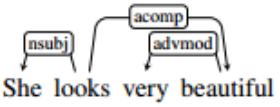

Figure 2. Dependencies in sentence [2]

3.2 Text processing

For our experiments we used educational articles from *Wikipedia*[3] written in simple English. We split these articles into our text database to three collections: texts, sentences and rules. *Text collection* holds whole texts which are processed. *Sentences collection* keeps sentences of texts from text collection. For each sentence in this collections there also is a note which was created by processing the sentence. Last collection named *Rules* holds rules for creating notes from sentences of texts.

3.3 Rules for processing

A rule consists mainly from two parts - *list of data of original sentence* and *list of data of note*. Each entry in *list of data of original sentence* and *list of data of note* contains these parts: relation name and list of grouped dependencies with the same relation name. Each dependency contains a governor token, a dependent token and its position considering all dependencies. The governor and dependent token consists of Part-Of-Speech (POS) tag and index of word in sentence to which is the token connected. Index of the token is bounded with a position of its word in sentence.

Dependencies from the second list are applied to sentence to create a new note. The rule may order to create a compound note from a sentence. The compound note is composited of some simple sentences. The positions in sentence on which the note should be split into smaller sentences are kept within the rule.

When processing a sentence an applicable rule has to be looked up in database before creation of the note. Dependencies of rule and dependencies of the sentence being processed has to correspond to each other. Evaluation is based upon two conditions. The sentence that is being processed has to have the exact amount of entries in list of data of original sentence while these entries contain exactly the same relation names as the rule's relation names.

The applicable rule is found if these two conditions are met. However, the conditions can cause a situation that more than one rule is found. In this case we have to calculate the match probability of this sentence and the original sentence obtained from the rule. The rule with the highest probability of the match is applied.

Calculating the match consists of several steps. First, the POS tags match of governor and dependent tokens is calculated separately. Indices of governor and dependent tokens are calculated also separately. These first steps determine if the sentence contains arbitrary dependency with same value of POS tag or index. In followed step is determined a half-match of dependencies. Half-match

[3] www.wikipedia.org

of dependency is match of POS tag and index at the same time at governor or dependent token of dependency. We calculate matches of POS tags and index of governor or dependent token for every dependency. Finally, in the last step we calculate the number of absolute-matched dependencies. Absolute-match dependencies is the total match of POS tags and indices in governor and dependent tokens. Every step has assigned a rating. If a condition in the step is evaluated as true, the rating of the step is added to the final result. The final result is a percentage value of the match. The rating is based on importance of the step in calculating a precise match, while depending on the number of steps and dependencies, so the final result cannot exceed a limit of 100%. A pseudo code for an algorithm calculating the match is outlined in Algorithm 1 and specific example is shown on Figure 3.

Algorithm 1 Calculating match

```
1:  procedure CALCULATEMATCH(sentence, originalDependencies)
2:      oneCompareTypeRating ← calculate percentage rating of one comparison
3:      for all originalDependencies do
4:          if count(sentence, dependency) = count(originalDependencies, dependency) then
5:              match ← match + oneCompareTypeRating
6:          counter ← counter + count(originalDependencies, dependency)
7:      oneCompareTypeRating ← oneCompareType/counter
8:      for all originalDependency do
9:          for all dependency do
10:             for all comparison do
11:                 if applyComparison(sentence, comparison, dependency) then
12:                     match ← match + oneCompareTypeRating
        return match
```

If rule look up does not find any applicable rule, it means that the system have not processed the same or similar sentence yet. A manual rules of parser are used in this case. The output of the parser is a note. A new rule is created based on the note. Dependencies of original sentences are taken and used to create a *list of data of original sentence*. This list is then assigned to the rule. Dependencies of note are used to create a *list of data of note* which is then also assigned to the rule. The sentence ends are determined depending on how many sentences the note contains. POS tags and indices of tokens are stated by the corresponding words of the original sentence and the newly created note.

3.4 Rule application

By the principle of rule look up, the sentence being processed has to contain dependencies from the *list of data of original sentence* and also dependencies from the *list of data of note*.

The process of applying a rule has several steps. For each dependency in the list of data of original sentence, the respective dependency is looked up in sentence that is being processed. The word corresponding with dependent token from the looked up dependency is taken and added to the note on its index position. In case of dependency relation *nominal subject* the word corresponding with governor is also added. After processing all dependencies the last minor changes are done such as capitalization of the first letter of the note, splitting note into more sentences if rule defined so. Algorithm 2 shows pseudo code of the process of applying rule on sentence.

3.5 Example

Lets consider example from Figure 1. We have rules for two sentences and we are processing the first one. In this situation, there are at least two rules which are applicable for the sentence a. Assume that we are calculating match with the sentence b. We iterate over all dependencies of processed

Algorithm 2 Applying rule

```
1: procedure ApplyRule(sentence, rule)
2:     note ← new Note
3:     for all ruleDependencies do
4:         dependency ← findDependency(sentence, ruleDependency)
5:         if isFound(dependency) then
6:             add(note, getDependent(dependency))
7:             if isNominalSubject(relation(dependency)) then
8:                 add(note, getGovernor(dependency))
9:     splitToSentences(note, sentencesEnds(rule))
       return note
```

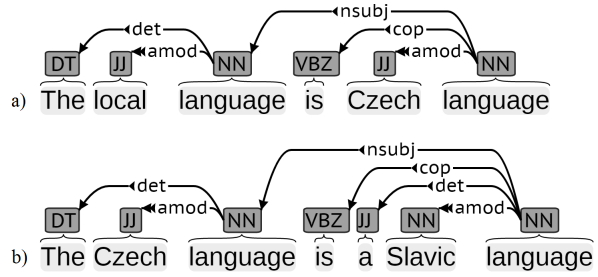

Figure 3. Example sentences

sentence *a*. The first dependency is *det* with the governor token NN (noun) and index 3 and the dependent token DT (determiner) and index 1. First, we find out, if the sentence *b* contains any dependency with tokens NN or DT and index equals to 1 or 3. This is the separate calculation of POS tags and indices. Then, we try to find in the sentence *b* any dependency, which has dependent or governor token tag of type NN and index equal to 3 or tag of type DT and index equal to 1. This is only the half-match step. As the last step, we check if sentence b contains dependency, where the governor token is NN and index is equal to 3 and the dependent token is DT and its index is 1. If any of these step were matched, the rating of that particular step is added to the final result and iteration continues with following dependency until all dependencies were iterated over.

4 First experiments

We used our system to create notes from three *wikipedia* texts. We compared results with Autosummarizer [4], a project focusing on text summarization. Autosummarizer is still in development and we used a beta version. This system uses an extractive summarization.

Texts contained 27 sentences and 294 words in total. On Figures **??** and **??** we show number of sentences and words written to output respectively. Figure **??** shows how many sentences from original text were shown in output without being processed. Figure **??** shows average number of irrelevant information (words) that were eliminated from final outputted sentences.

Autosummarization system is better in outputting less amount of data, but it lose lots of relevant information. Our system outputted more data and processed every important information. Our goal is to keep important text information. When finding a pattern for creating notes from text

[4] www.autosummarizer.com

a lot of possible combinations exists. By combining existing approaches with dependencies and relations the amount of possible combinations is decreased. There is limited number of relations and dependencies and combined with relevant data of texts the number of combinations is lower than if only one approach, for example the part-of-speech tagging is used.

Students are searching for information every day and it is difficult to choose the most relevant parts from text for them. We proposed a system which helps with this. Our system is capable of creating notes from texts. Our system uses *rules* for note creation. A rule is a collection of data such as dependencies, tokens, relations, indices and some other data. The rule is used to extract relevant information from sentence and create a note. Before applying a rule to a sentence, the system looks up applicable rule and calculates match value. We focus to design the system with intent to personalization. User is able to modify the outputted notes and it allows system to learn new patterns and create more personalized notes next time.

The future work can focus on automatization of creation of the notes. By learning rules from interaction with user a general patterns for note creation may be found.

Acknowledgement: This work was partially supported by the Scientific Grant Agency of Slovak Republic, grant No. VG 1/0646/15.

References

[1] Das, D., Martins, A.F.T.: A Survey on Automatic Text Summarization, 2007.

[2] Marie-catherine De Marneffe and Christopher D. Manning: Stanford typed dependencies manual, 2008.

[3] Nenkova, A., McKeown, K.: Automatic Summarization. *Foundations and Trends® in Information Retrieval*, 2011, vol. 5, no. 2–3, pp. 103–233.

[4] Radev, D.R., Hovy, E., McKeown, K.: Introduction to the special issue on summarization. *Comput. Linguist.*, 2002, vol. 28, pp. 399–408.