3 Development of a Review Summarizer

This part is developed by Jiang Haofeng.

Given a product, the summarizer should summarize all *reviewTexts* received for this particular product.

3.1 Ideal summary

An ideal summarizer will output a list of representative phrases having both pros and cons, regarding just this product, and much regardless of other products. The phrases are either “noun phrase with adjectives” or “verb phrases with adverbs”. Based on my understanding, for a certain product, the best way to give readers a sense of the picture is what adjective or adverb users utilize to describe the functionality, appearance, feelings, etc. I prefer phrases to just keywords because modifiers are more informative with the actual objects or actions following them. To achieve this, the algorithm I used is TF-IDF method learned in AI6122 lecture. As shown in the lecture, the largest TF-IDF value is, the more occurrences a term in the document and the more rarity a term in the collection. This is perfectly tailored to the ideal summary.

3.2 Technical challenges

I used NLTK library to help me deal with raw texts. However, when doing word tokenization and POS tagging, NLTK may not have the expected results. For example, some writers forgot to add a space after a period, which results in NLTK recognizing “the last word of a sentence”-“the period”-“the first word of the next sentence” as a single word. It will seriously affect the counting in TF-IDF weight calculation. This can be solved by manually adding strips to all punctuations. Another unexpected result is tagging wrong class to a word, since a word may have multiple classes depending on the context. This is a small issue since I used regular expressions to select grammar and hence my rule can be tolerant enough.

1. def punctuation\_strip(string):
2. puncts = ",.?:;!"
3. for p in puncts:
4. string = string.replace(p,' '+p+' ')
5. return string

In terms of regular expressions, I have to consider what is the best rule for the ideal output. Apart from lexical variants (like plural, comparative adjectives, or verb tenses), I add modal to the “adverb” category, since I think a modal verb can make the verb phrase more complete. Another tricky tag is the proper noun. I finally abandon proper nouns in “noun” category, since proper nouns are bound to win high TF-IDF weights and appear in the summary result. It will make the solution full of hard-understandable words. That is not an approachable summary supposed to be.

1. noun = "(<NN>|<NNS>)"
2. verb = "(<VB>|<VBD>|<VBG>|<VBN>|<VBP>|<VBZ>)"
3. adj = "(<JJ>|<JJS>|<JJR>)"
4. adv = "(<MD>|<RB>|<RBR>|<RBS>)"

Besides, when doing TF-IDF method, I realize that word normalization is important. Everything is fine except lemmatization. I used WordNetLemmatizer in NLTK and similarly, some lemmatization results are wrong. For example, the verb “has” will be lemmatized to “ha”. Although it will not cause huge problems as long as all words in the documents and collections have the same lemmatizing standard, the problem occurs when showing the final summary result. If showing the phrases with lemmatization, some of the words will be strange (like “ha”). Otherwise, very similar nouns phrases like “nice car” and “nice cars” may appear together. To solve it, I used a smart way to compromise both sides. That is, after lemmatizing all phrases, a lemmatized phrase will not be calculated TF-IDF value if the phrase contains a calculated phrase, or reversely, if it is contained by a calculated phrase.

1. for s in strlist:
2. keys = list(map(normalize, score\_dict.keys()))
3. if normalize(s) in ' '.join(keys) or \
4. sum(map(lambda k: k in normalize(s), keys))>0:
5. continue
6. ...

Finally, to maintain “length does not beat informativity”, I only calculate TF-IDF weights for words whose POS tag is not <DT> or <MD>.

3.3 Code walkthrough

Each component in my solution is as follows:

**[summarizer.py]**

* **get\_global\_values**, **set\_global\_values**

Enable using datasets among different python files.

* **adj\_NP**, **adv\_VP**

Generate the two regular expressions. Using functions to present them is easy to follow and debug.

* **punctuation\_strip**

Avoid unexpected tokenization result. (Shown in §3.2)

* **extract\_grammar**

The core of extract candidate noun phrases and verb phrases based on the two regular expressions just defined. Using nltk.RegexpParser to get a parse tree.

* **compress\_tree**, **make\_list**

Process the parse tree and get the list of both candidate strings and the corresponding tags. Using HOF “map” because it is superior to loop regarding on its efficiency and less function calls.

* **remove\_abb**, **normalize, normalize\_sentence**

Normalize the candidate phrases by case-folding, lemmatization, abbreviation processing. Using HOF for the same reason.

* **process\_product**

The first function that related to the actual dataset. Use a for loop to traverse through the dataset and find the target product id. Utilize a series helper functions defined above. To reduce repeated iteration, I record all useful information just in one loop. Therefore, the output is a tuple of raw\_text, strlist, tagdict.

* **tfidf**

The essence of our weight algorithm: TF-IDF. For TF, I use a “smoothing method” (make it 1 if 0) to deal with unexpected NLTK results. Just as lecture notes, use log to avoid large numbers, since log preserves monotonicity.

* **generate\_score**

Generate the TF-IDF weight of each candidate phrases. I normalize all words in the whole collection, in order to count the TF and DF of each token regardless of its variants. The two “abandoning situation” introduced in §3.2 also apply here. I believe this is the processing that best align to the algorithm I use.

* **generate\_summary**

Sort and output n highest scores among candidate phrases. Using heapq function because it is faster.

* **main body**

Since our group choose “Digital Music” and “Musical Instrument” as two datasets, the main body will first initialize the dataset with these two types. Then specifically choose a product which has 10 reviews from each dataset, and produce the review summary. (Will show in §3.6)

**[summarizerUI.py]**

* **main body**

This python program is an interactive UI for you to play with. You can view review summary of the sampled product **in any category** you want (i.e. not restrained to the datasets used in this assignment), just by following the instructions on the command line. An outstanding point is that this program can “memorize” the dataset you have been created. That is, it will only download and initialize dataset for each type once, unless you exit the program. Users will not have to wait for the dataset initialization if the type has been visited before.

If you want to view the products in the datasets chosen by our group, please choose index 21 or 22 in the first input.

3.4 Limitations

**Writer’s typos**

Since this part has implemented very few tolerant retrieval algorithms, some writer’s typos may appear in the final summary result. In TF-IDF algorithm, typos may be considered quite “unique” and results in high score, even though the phrase is meaningless.

**Weak informativity phrases**

The summary result may still contain phrases that are not so informative. The reason mainly because my regex rule is tolerant.

3.5 Evaluation

Some alternative solutions may have different intuition about an ideal summary. For example, text classification algorithms can be used to classify positive and negative reviews. I think the evaluation can be based on following criteria:

1. Running time / space complexity
2. Clearness and understandability (i.e. whether readers could understand the words in the summary)
3. Usefulness (i.e. whether a possible figure of the product can be imagined by reading the summary)
4. Alignment with the actual reviews

3.6 Output examples

As mentioned in §3.2, the main body of summarizer.py serves this section. It chooses product B000068O4H in “Musical Instrument” and product B000001DYS in “Digital Music”, each of which has exactly 10 reviews. Since every product has at least 5 reviews, I think 10 is a moderate number.

type = Musical\_Instruments, asin = B000068O4H

Review summary:

a basic female XLR

high impedance input

low impedance cable

an xlr signal

an old funky mixer

a nice little adapter

regular mic cables

handy many times

local guitar stores

particular situation

type = Digital\_Music, asin = B000001DYS

Review summary:

the funky musical pool

this solid single-disc collection

the inadequate number

roller skating

the smooth soulfulness

emotive vigor

innovative funk acts

great funk dance songs

unsatisfying example

more devout fans

Conference Name:ACM Woodstock conference

Conference Short Name:WOODSTOCK’18

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00