Categorizing distributed dialogues by interests

Natural Language Processing with Python

학번 | 20000000
이름 | sbcode

학번 | 20000000
이름 | todokaist

1. Input

이 program의 input은 드라마 House랑 Friends의 대본을 통해 구성된다. 먼저 program에서 불필요한 부분을 sliceparent(op)로 삭제하고, 우리의 목적에 맞는 mingled text를 만들기 위해 대사 순서를 유지하며 무작위로 섞는다. 이 때 말하는 speaker는 첫 문자를 대문자로 끝에 :를 달아 구분될 수 있도록 했다.

2. First Process

가. Purpose and I/O

대화를 topic에 따라 분류하기 위해서 원본 text를 Input으로 두 group으로 분류된 리스트를 Output으로 얻는 것이 목표다. 알고리즘의 기본 아이디어는 비슷한 주제로 대화를 하고 있는 사람들은 같은(비슷한) 단어를 쓴 다는 점에 착안하다.

나. Details in Algorithm

1) 파일을 받아 텍스트로 바꾸고, 단어들의 list로 변환을 한다.
2) wordlist를 받아서 대사를 말 한 사람에 따라 분류한다. Speake들의 Dict로 아웃풋을 만든다.
3) 특정 카테고리가 아니더라도 자주 쓰는 단어를 배제하기 위해 removeStopwords()을 이용해 stopwords를 제거해준다. 또 rid_extra()을 이용해서 TV와 같은 의미 없는 등장인물을 제거해준다.
4) 각 캐릭터 Set끼리의 비교를 통해 함께 말한 단어 수를 체크한다.
5) 한 캐릭터 A1에 대해서 A2,A3 .. An 과의 겹치는 단어 수의 평균을 기준으로 두 그룹으로 나눈다. 같은 작업을 A2,A3.. An에 대해서도 실행한다.
6) 그리고 이후에 모든 x에 대한 Ax를 바탕으로 group A, group B, ambiguous set를 반환한다.

다. Description of functions to solve problem

1) fileToWordList(file):
   Consume : file
   Produce : text(word list)

2) classifySpeaker(wordList):
   Consume : wordlist of text
   Produce : Dict of speaker

3) Stopwords(word):
   Consume : word
   Produce : return true if word is one of stopwords else false

4) removeStopwords(speakerDict):
   Consume : speaker Dict
   Produce : modified Dict with removing stopwords

5) rid_extar(Dict) :
   Consume : Dict processed with removeStopwords()
   Produce : Remove dict of speker who rarely talk with

6) num_same_words (human1_words, human2_words) :
   Consume : two speaker in Dict
   Produce : number of same words between speaker A and speaker B

7) seperate_into_two (dict) :
   Consume : Dict processed with rd_extar() and removeStopwords()
   Produce : list that separate all of speaker into two

3. Second Process

가. Input

첫 번째 프로그램으로 group A와 group B를 나눈 후, ambiguous로 판별된 speaker들에 대해 다음 프로그램을 적용한다. 즉 이 program의 input은 raw text, group AB, ambiguous speaker list다.

나. Details in Algorithm

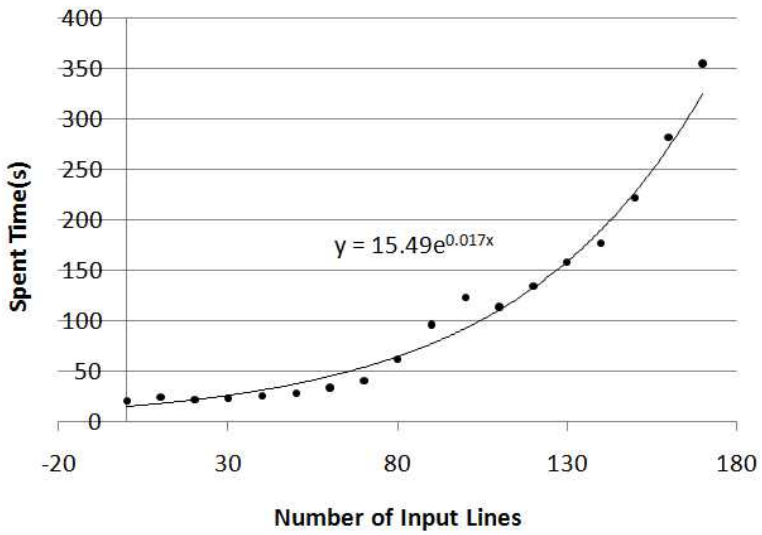
1) input text를 speaker를 key로, 한 말의 list를 value로 하는 dictionary를 만든다.
2) brown corpus를 기반으로 만든 combing tagger로 input text를 tag한다. 이 때 tagger는 bigram tagger, unigram tagger, default tagger을 결합한 것이다. 그리고 정확도를 위해 언급된 speaker의 이름은 NNP(proper noun)를 tagging한다.
3) tagged word 중에서 noun으로 tag된 word만을 추출한다. 그리고 이 list에서 wordnet에 저장된 synset이 없는 단어를 제외한다.
4) 임의의 speaker1의 word 집합과 speaker2의 word 집합이 주어졌을 때 이 word pair의 wordnet path similarity를 구한다. 이 word pair의 평균을 return한다.
5) ambiguous speaker X가 주어졌을 때 group A와 X의 wordnet path similarity score의 평균을 구한다. group B와 X의 wordnet path similarity score의 평균을 구한다. 이 중 큰 값을 가진 group에 X을 더한다.

다. Wordnet path similarity score function

어떤 명사가 주어졌을 때 wordnet에 저장된 synset은 1개가 아닐 수도 있다. 그런데 해당 input text에서 이 명사가 wordnet에 저장된 synset 중 어떤 것에 해당하는 지 알 수 있는 방법을 찾지 못했다. 따라서 주어진 synset 중 적절한 synset을 선택하는 기준을 임의로 설정했다.

1) average of wn.synsets(word)

두 번째 기준은 각 synset의 평균 similarity를 주어진 word pair의 path similarity로 반환하는 것이다. 임의의 word1과 word2가 있을 때, word1의 synsets 개수를 M, word2의 synsets 개수를 L이라고 하면 구해야 하는 path similarity는 ML개다. 즉 input text의 크기가 선형적으로 커지면 path similarity는 지수적으로 증가한다. 따라서 이 방법을 쓰는 것은 적절하지 않는 것으로 판단했다.



2) wn.synsets(word)[0]

첫 번째 기준은 synsets의 첫 번째 element가 가장 많이 사용되는 단어로서 input text에서 사용된 해당 단어도 이처럼 사용되었을 것으로 가정한다. 이는 2번째 방법 보다 빠르다. 따라서 우리는 이 방법을 선택했다.

4. Output and Performance

가. Output

400 line의 input text로 두 개의 process를 적용했을 때 output은 아래와 같다.

Input of First Process	mingled text of Houst MD. and Friends(size 400)	
Output of First Process Input of Second Process	Group A, B	[['Wilson', 'Foreman', 'Cameron', 'Rebecca'], ['MONICA', 'RACHEL', 'PAUL', 'CHANDLER', 'JOEY']]
	Ambiguous Speaker	['Cuddy', 'House', 'PHOEBE', 'Chase', 'ROSS']
Output of Second Process	[['Wilson', 'Foreman', 'Cameron', 'Rebecca', 'Cuddy', 'House', 'Chase'], ['MONICA', 'RACHEL', 'PAUL', 'CHANDLER', 'JOEY', 'PHOEBE', 'ROSS']]	
golden list	[['Wilson', 'Foreman', 'Cameron', 'Rebecca', 'Cuddy', 'House', 'Chase'], ['MONICA', 'RACHEL', 'PAUL', 'CHANDLER', 'JOEY', 'PHOEBE', 'ROSS']]	
Performance	100%	

나. Performance

다음은 Input text의 크기에 따른 performance를 보인 것이다. Input text가 너무 작으면 performance가 떨어진다. 오히려 Input text의 크기가 클수록 오히려 performance가 낮아지는 경향을 보였는데, 이는 전체 드라마에서 항상 같은 주제에 대해 이야기 하는 것이 아니기 때문으로 추측된다.