Assignment #2

Clustering & Word Embedding

소프트웨어전공

2015004120 박선하

1. 코드 설명

**\* complete\_link\_clustering**

|  |
| --- |
|  |

인자로 받은 sim\_name에 해당하는 유사도 혹은 거리를 기준으로 complete link clustering을 수행하는 함수입니다. sim\_name을 통해 어느 similarity 함수를 사용할 지 정하고, 각각의 기준에 따라 값이 클수록 유사한지, 작을수록 유사한지를 판단하는 most를 설정합니다.

클러스터 레벨을 저장하는 levels 리스트와 각각의 단어 벡터 하나씩으로 이루어진 클러스터의 리스트 clusters를 만든 뒤, 각 단어 벡터간의 유사도를 저장하는 행렬 prox\_mat를 만듭니다. prox\_mat는 (num\_of\_words-1) \* (num\_of\_words-1) 크기의 하삼각행렬입니다.

하나의 클러스터로 합쳐질 때까지 prox\_mat에서 가장 유사한 클러스터 쌍을 찾습니다. 12번째 줄에서 이 작업을 수행하는데 most\_sim은 [클러스터2 idx, [클러스터1 idx, similarity]] 형태로 저장됩니다. 그리고 하삼각행렬의 대각선 부분이 없기 때문에 실제 가장 유사한 두 클러스터의 인덱스는 most\_sim[0] + 1, most\_sim[1][0]이 됩니다. 클러스터 쌍의 두 클러스터를 합한 뒤 levels 리스트의 가장 끝에 저장합니다. clusters 리스트에서 두 클러스터를 지우고 두 클러스터를 합한 새 클러스터를 리스트의 가장 마지막에 저장합니다.

이후 prox\_mat를 수정해야 하는데, 두 클러스터의 거리 row를 지운 뒤, 모든 row의 column을 확인하며 지워진 두 클러스터와의 거리를 지웁니다. 이후 prox\_mat의 가장 마지막 줄에 새로 생긴 클러스터와 기존의 클러스터들 간의 거리를 계산한 row를 추가합니다.

모든 클러스터가 합쳐져 클러스터가 하나 남게 된다면 [[similarity1, cluster1], [similarity2, cluster2], [similarity3, cluster3], …] 로 이루어진 배열 levels를 반환합니다.

**\* cosine\_similarity**

|  |
| --- |
|  |

공식에 따라 두 300차원 단어 벡터 a, b 간의 cosine similarity를 구하는 함수입니다.

**\* euclidean\_distance**

|  |
| --- |
|  |

공식에 따라 두 300차원 단어 벡터 x, y 간의 euclidean distance를 구하는 함수입니다.

**\* find\_least\_sim**

|  |
| --- |
|  |

두 클러스터간에 가장 덜 유사한 원소간의 similarity(혹은 distance)를 반환하는 함수입니다. euclidean distance는 거리가 멀 수록 (max) 덜 유사하고, cosine similarity는 거리가 가까울 수록 덜 유사하기 때문에 이 부분을 선택하여 두 클러스터의 각 원소간 거리 중 가장 덜 유사한 값을 반환합니다.

**\* normalize**

|  |
| --- |
|  |

euclidean distance를 기준으로 클러스터링 했을 경우에만 사용합니다. 전체 level 거리 중 가장 큰 값과 가장 작은 값을 이용해 가장 가까운 거리가 가장 유사하기 때문에 1, 가장 먼 거리를 가장 덜 유사하다는 의미에서 0으로 정규화 합니다. 이 함수에서 euclidean distance가 euclidean similarity로 바뀝니다.

**\* get\_words\_vectors**

|  |
| --- |
|  |

WordEmbedding.txt에서 word만으로 이루어진 리스트, 각 word의 300차원 vector만으로 이루어진 리스트를 만들어 총 개수와 함께 반환합니다. 정상적인 WordEmbedding.txt라면 word\_list의 길이와 vector\_list의 길이는 같습니다.

**\* divide\_cluster**

|  |
| --- |
|  |

ascending order로 저장된 levels에서 인자로 받은 threshold에 따라 분할하는 함수입니다. 결과를 저장할 0으로 초기화된 행렬 cluster\_num과 클러스터 id를 붙일 때 사용하고 전체 클러스터 개수를 나타내는 cluster\_idx 변수를 만듭니다.

통채로 있는 levels 리스트에서 bisect\_left를 이용하여 threshold보다 더 similarity가 높은 클러스터만 남겨둡니다. 남아있는 levels는 크기가 큰 (더 많은 부분 클러스터를 가진) 순서대로 리스트에 저장되어 있기 때문에 앞에서부터 순회하면서 클러스터에 id를 붙여줍니다. 만약 이미 id가 붙어있는 (어떤 클러스터의 부분 클러스터)라면 새로 id를 붙이지 않고 넘어갑니다.

levels의 모든 원소를 순회하며 id를 붙인 뒤에도 0으로 남아있는 원소는 각각이 하나의 클러스터라는 의미이기 때문에 cluster\_idx를 하나씩 증가시켜가며 id를 붙여줍니다. 전체 클러스터의 개수와 클러스터 분할 결과 리스트를 반환합니다.

**\* write\_on\_file**

|  |
| --- |
|  |

클러스터링 결과를 결과 파일 WordClustering.txt에 저장하는 함수입니다. WordEmbedding.txt를 열어 str 형태로 읽어와 해당 단어가 어느 클러스터에 속했는지를 결과 파일에 함께 저장합니다.

**\* get\_word\_class**

|  |
| --- |
|  |

WordTopic.txt를 읽어와 각 단어가 어느 class에 속하는지를 가져오는 함수입니다. 대괄호로 묶인 클래스 이름 등을 제외하고 모두 소문자로 치환한 단어만 클래스 단위로 묶어 리스트에 저장한 뒤 이 결과를 word\_topic 리스트에 저장합니다.

전체 단어를 순서대로 모두 순회하면서 이 단어가 몇 번째 class인가를 구해 리스트로 반환합니다.

**\* entropy\_measure**

|  |
| --- |
|  |

엔트로피 계산을 이용해 클러스터링을 평가하는 함수입니다. 클러스터 인덱스를 가지고 있던 c\_list를 이용해 각각의 단어가 어떤 클래스에 속하는지를 word\_cls를 이용해 확인한 후 각 클러스터에 클래스별로 몇 개의 단어가 있는지를 리스트에 저장합니다. 이때 내장 collections의 Counter 클래스를 써서 쉽게 원소의 수를 구할 수 있습니다.

리스트에서 엔트로피를 계산한 뒤, 전체 크기에서 각 클러스터의 크기를 구한 뒤 앞에서 구한 엔트로피에 weighted sum을 적용하고 그 결과를 반환합니다.

**\* silhouette\_measure**

|  |
| --- |
|  |

실루엣 지표로 클러스터링을 평가하는 함수입니다. 클러스터의 개수와 클러스터 리스트를 인자로 받습니다. 모든 단어 벡터의 실루엣 계수의 평균을 지표로 삼을 것이기 때문에, 실루엣 지표를 저장할 리스트를 만듭니다. 계산량을 줄이기 위해 먼저 각 벡터간의 거리를 미리 계산해놓은 거리 행렬을 만들어 dist\_mat라 이름 붙입니다.

클러스터 인덱스를 가지고 있던 c\_list를 각 클러스터 리스트를 가진 리스트 clustered\_idx 로 바꾸고 이 리스트를 이용해 각각의 실루엣 계수를 계산합니다.

어느 클러스터에 속했는지를 확인한 후, 자신 혼자만 속한 클러스터(길이가 1인 클러스터)이면 0, 그렇지 않은 경우에는 , a(i)(코드의 a\_i)와 b(i)(코드의 b\_i)를 구해 실루엣 계수를 계산한 뒤 silhouette\_list에 추가합니다. 전부 계산한 후 silhouette\_list의 평균을 반환합니다.

**\* main routine**

|  |
| --- |
|  |

위에서 정의한 함수들을 이용해 프로그램을 실행하는 부분입니다. 클러스터링 조건을 저장하는 argument 리스트가 있고, get\_words\_vectors 함수를 이용해 단어 리스트, 300차원 벡터 리스트, 전체 단어의 개수를 받아온 뒤, get\_word\_class함수를 이용해서 이후 엔트로피 계산에 사용할 단어 클래스를 입력받습니다. argument의 첫번째 원소가 ‘c’이면 cosine similarity를 기준으로, ‘c’ 이외의 다른 것이면 (‘e’를 권장) euclidean distance를 기준으로 complete\_link\_clustering 함수를 이용해 클러스터링을 진행합니다. 가장 가까운 원소부터 descending order로 나열되어 있기 때문에 이후 divide하는 과정에서 bisect를 사용하기 위해 리스트를 ascending order으로 만듭니다. euclidean distance였다면 euclidean similarity로 바꾸고 similarity threshold를 적용하기 위해 클러스터간 거리중 최대 거리를 0, 최소 거리를 1으로 정규화 시켜줍니다.

argument의 두번째 원소인 similarity threshold에 따라 divide\_cluster 함수를 이용해 생성된 클러스터 수와 클러스터링 결과 리스트를 반환합니다. write\_on\_file 함수를 불러와 결과를 파일에 저장합니다.

클러스터의 수와, similarity threshold를 출력한 뒤, entropy와 silhouette 지표에 따라 클러스터링을 평가하고 그 결과를 출력합니다.

2. 실험 결과 및 분석

|  |
| --- |
|  |

euclidean distance를 이용해 계산하여 euclidean similarity로 정규화한 뒤, similarity threshold를 0.6으로 클러스터를 분할한 결과입니다.

| (e) ->entropy의 weighted sum  (s) -> silhouette coefficient 평균 | Cosine Similarity | Euclidean Similarity |
| --- | --- | --- |
| threshold = 0.2 | 82 clusters  (e) 1.025408407814967  (s) 0.030613150806732256 | 8 clusters  (e) 2.2026057890279946  (s) 0.02426327410199649 |
| threshold = 0.4 | 184 clusters  (e) 0.47491789898721876  (s) 0.05777897340508433 | 47 clusters  (e) 1.5030137409335789  (s) 0.024932774245022674 |
| threshold = 0.6 | 286 clusters  (e) 0.1228099852199022  (s) 0.04098978641469138 | 172 clusters  (e) 0.6232499153326599  (s) 0.06253858005405817 |
| threshold = 0.8 | 331 clusters  (e) 0.005917159763313609  (s) 0.010084158462648863 | 305 clusters  (e) 0.08284023668639054  (s) 0.03022169758125554 |

각각의 clustering 결과에 클러스터의 개수, entropy의 weighted sum과 silhouette 계수의 평균을 이용해 클러스터링을 평가했습니다. similarity 함수와 threshold에 따라 클러스터링 분할의 정도가 달라집니다. threshold가 커질수록 더 많은 클러스터가 생성됨을 볼 수 있습니다.

위의 표에서 보면 같은 threshold로 분할했을 때 cosine similarity를 이용한 클러스터링이 euclidean similarity를 이용한 클러스터링보다 더 낮은 엔트로피 합을 보여주는 것을 알 수 있습니다. 하지만 엔트로피 합의 위에 써놓은 분할된 클러스터의 개수를 본다면 cosine similarity와 euclidean similarity에 관계없이 더 많은 클러스터로 분할되어 각 클러스터의 purity가 높아졌기 때문에 값이 작아졌다 생각할 수 있습니다.

silhouette 평균을 보면 위 자료에서는 cosine similarity는 threshold 0.4 근처에서 euclidean similarity는 threshold 0.6 근처에서 최대인 것을 볼 수 있습니다. 다른 threshold의 값들과 비교했을때 클러스터의 응집도를 나타내는 silhouette 수치가 높을 수록 더 클러스터링이 잘 되었다고 하기 때문에 **Euclidean similarity + threshold 0.6** 조합을 가장 클러스터링이 잘 되었다고 평가했습니다.

3. 실행 환경

Python3 문법 아래 작성되었고 Python 3.6 환경에서 실행하였습니다. 기본 파이썬 환경이 python 2.x인 경우 코드를 실행할 때는 ‘**python3 assignment2\_2015004120.py**’와 같은 형식으로, 기본 파이썬 환경이 3.x인 경우 ‘**python assignment2\_2015004120.py**’와 같이 터미널에서 입력하면 같은 경로에 있는 WordEmbedding.txt와 WordTopic.txt를 입력받아 경로를 탐색한 뒤, 같은 경로의 WordClustering.txt에 결과를 저장합니다.

다른 similarity measure와 similarity threshold를 적용하고 싶다면 코드의 argument 리스트(line 147)의 원소를 수정하면 되는데, 첫번째 원소는 ‘c’(cosine similarity)와 ‘e’(euclidean distance)를 변경하는데 사용하고, 두번째 원소는 similarity threshold를 적용합니다.