**Artificial Intelligence**

**assignment #2**

Hanyang University

Computer Science(컴퓨터전공)

2012004087

이기준

**코드설명**

**import** numpy **as** np  
**import** pandas **as** pd  
**import** math

자료분석을 쉽게 하기 위하여 numpy, pandas를 import하였다. 또한 제곱과 루트를 쉽게 사용하기 위해 math를 import하였다.

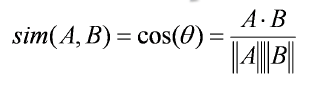
**def** loadWord () :  
  
*# fileName = sys.argv[1]* fileName = **"WordEmbedding.txt"  
 try**:  
 fp = open(fileName, **'r'**)  
 **except**:  
 print(**"파일을 열 수 없습니다. 다시 시도해주세요.\n"**)  
  
 list = []  
  
 **while True**:  
 templist = []  
 line = fp.readline()  
 **if not** line: *# 파일의 내용을 한줄씩 읽고 다음줄이 없으면 while문 break* **break** templist.append(line.split()[0])  
 line = fp.readline()  
 **if not** line: *# 파일의 내용을 한줄씩 읽고 다음줄이 없으면 while문 break* **break** templist += [line.split(**','**)]  
 **for** i **in** range(0, 300):  
 templist[1][i] = float(templist[1][i])  
 list.append(templist)  
 *# list.sort(key = lambda x: (x[1], x[2]))* **return** list

WordEmbedding.txt파일을 열어서 리스트 형태로 저장한다.

**def** loadWordTopic (DB) :  
*# fileName = sys.argv[1]* fileName = **"WordTopic.txt"  
 try**:  
 fp = open(fileName, **'r'**)  
 **except**:  
 print(**"파일을 열 수 없습니다. 다시 시도해주세요.\n"**)  
 i = 0  
 list = []  
 **while True**:  
 templist = []  
 wordClass = fp.readline()  
 wordClass = wordClass.split(**'['**)[1]  
 wordClass = wordClass.split(**']'**)[0]  
  
 **while True**:  
 line = fp.readline()  
 **if** line ==**'\n'** : *# 파일의 내용을 한줄씩 읽고 다음줄이 없으면 while문 break* **break  
 if not** line: *# 파일의 내용을 한줄씩 읽고 다음줄이 없으면 while문 break* **return** DB  
 DB[i].append(wordClass)  
 i += 1

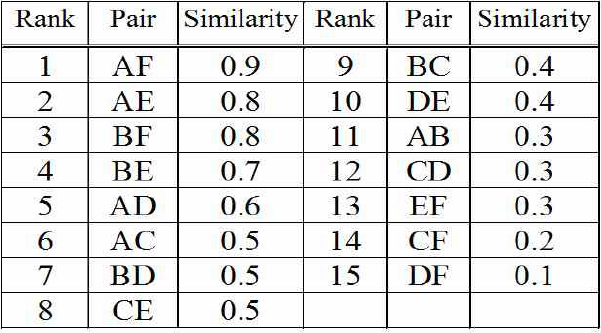
WordTopic.txt파일을 열어서 리스트 형태로 저장한다.

**def** Cos\_Sim (word1, word2) :  
 word1\_ProductSum = .0  
 word2\_ProductSum = .0  
 product1and2 = .0  
  
 **for** i **in** range(0, 300) :  
 word1\_ProductSum += math.pow(word1[1][i],2)  
 word2\_ProductSum += math.pow(word2[1][i],2)  
 product1and2 += word1[1][i] \* word2[1][i]  
  
 word1\_ProductSum = math.sqrt(word1\_ProductSum)  
 word2\_ProductSum = math.sqrt(word2\_ProductSum)  
  
 **return** product1and2 / (word1\_ProductSum \* word2\_ProductSum)

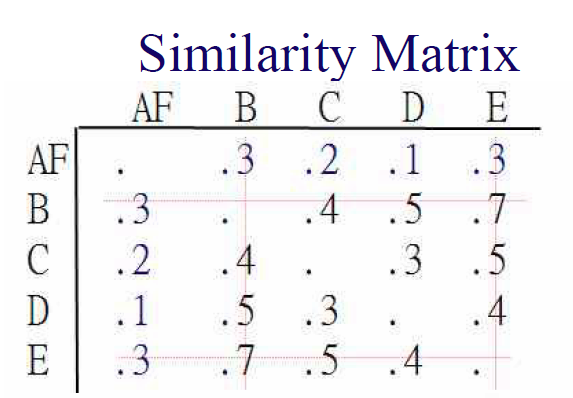


1에 가까울수록 연관이 있고, -1에 가까울수록 연관성이 떨어진다.

**def** SetAllCos\_Sim (old\_list) :  
 print(**"2차원 행렬의 형태로 Cosine Similarity를 생성중입니다."**)  
 new\_list = []  
 row\_names = []  
 rank = []  
  
 **for** i **in** range(0, len(old\_list)-1) :  
 **for** j **in** range(i+1, len(old\_list)):  
 temp\_list = []  
 temp\_list.append(old\_list[i][0])  
 temp\_list.append(old\_list[j][0])  
 temp\_list.append(Cos\_Sim(old\_list[i], old\_list[j]))  
 rank.append(temp\_list)  
  
 rank.sort(key=**lambda** x : x[2])  
  
 **for** i **in** range(0, len(old\_list)) :  
 row\_names.append(old\_list[i][0])  
 **for** j **in** range(0, len(old\_list)):  
 **if** i == j :  
 new\_list.append(**None**)  
 **else** :  
 new\_list.append(Cos\_Sim(old\_list[i], old\_list[j]))  
  
 matrix = np.reshape(new\_list, (len(old\_list), len(old\_list)))  
 df = pd.DataFrame(matrix, columns=row\_names, index=row\_names)  
 **return** [rank, df]

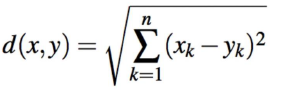
****

Rank는 위와 같은 리스트를 생성한 것이다.



Df는 위와 같은 Cosine Similarity Matrix를 생성한 것이다.

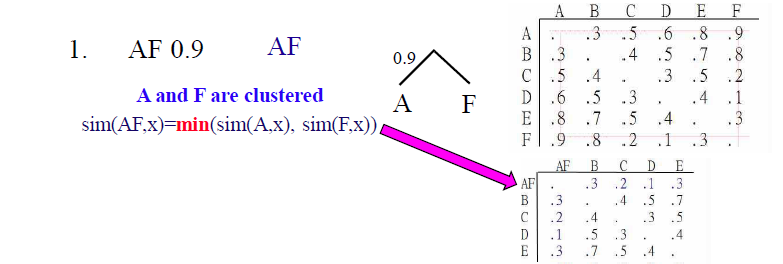
**def** Eu\_Sim (word1, word2) :  
 word\_ProductSum = .0  
  
 **for** i **in** range(0, 300) :  
 word\_ProductSum += math.pow(word1[1][i] - word2[1][i],2)  
  
 **return** math.sqrt(word\_ProductSum)



**def** SetAllEu\_Sim (old\_list) :  
 print(**"2차원 행렬의 형태로 Euclidian Similarity를 생성중입니다."**)  
 new\_list = []  
 row\_names = []  
 rank = []  
  
 **for** i **in** range(0, len(old\_list)-1) :  
 **for** j **in** range(i+1, len(old\_list)):  
 temp\_list = []  
 temp\_list.append(old\_list[i][0])  
 temp\_list.append(old\_list[j][0])  
 temp\_list.append(1 - Eu\_Sim(old\_list[i], old\_list[j]) / (2\*math.sqrt(300)))  
  
 rank.append(temp\_list)  
  
 rank.sort(key=**lambda** x : x[2])  
  
 **for** i **in** range(0, len(old\_list)) :  
 row\_names.append(old\_list[i][0])  
 **for** j **in** range(0, len(old\_list)):  
 **if** i == j :  
 new\_list.append(**None**)  
 **else** :  
 new\_list.append(1 - Eu\_Sim(old\_list[i], old\_list[j]) / (2\*math.sqrt(300)))  
  
  
 matrix = np.reshape(new\_list, (len(old\_list), len(old\_list)))  
 df = pd.DataFrame(matrix, columns=row\_names, index=row\_names)  
  
 **return** [rank, df]

Euclidian distance는 클수록 유사하지 않기 때문에 임의로 Euclidian Similarity는 1 - Euclidian distance 로 계산한뒤 2\*sqrt(300)으로 나눠서 normalizing을 한다. Cosine과 마찬가지로 rank와 df를 생성한다.

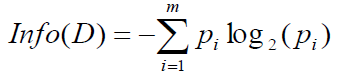
**def** Clustering (rankAndDf, threshold) :  
 print(**"clustering"**)  
 return\_list = []  
  
 cur\_idx = 0  
 cur\_threshold = threshold[cur\_idx]  
  
 rank = rankAndDf[0]  
 df = rankAndDf[1]  
  
 **while** len(rank) != 0 :  
 temp = rank.pop()  
 word1 = temp[0]  
 word2 = temp[1]  
 **for** i **in** df.index :  
 **if** word1 **in** i :  
 word1 = i  
 **break  
 for** i **in** df.columns:  
 **if** word2 **in** i:  
 word2 = i  
 **break  
 if** temp[2]< cur\_threshold :  
 print(len(df))  
 return\_list.append(df.index)  
 cur\_idx += 1  
 **if** cur\_idx == len(threshold):  
 **return** return\_list  
  
 cur\_threshold = threshold[cur\_idx]  
  
 **if** df[word1][word2] **is None** :  
 **continue  
 if** df[word1][word2] == temp[2]:  
  
 temp\_df1 = df.pop(word1)  
 temp\_df2 = df.pop(word2)  
 temp\_df1.name += **"+"** + temp\_df2.name  
 **for** i **in** temp\_df1.index:  
 **if** temp\_df1[i] **is None**:  
 **continue  
 elif** temp\_df2[i] **is None**:  
 temp\_df1[i] = **None  
 elif** temp\_df2[i] < temp\_df1[i]:  
 temp\_df1[i] = temp\_df2[i]  
  
 df = pd.concat([df, temp\_df1], axis=1, join\_axes=[df.index])  
 temp\_df1 = df.loc[word1]  
 temp\_df2 = df.loc[word2]  
  
 temp\_df1.name += **"+"** + temp\_df2.name  
 **for** i **in** temp\_df1.index:  
  
 **if** temp\_df1[i] **is None**:  
 **continue  
 elif** temp\_df2[i] **is None**:  
 temp\_df1[i] = **None  
 elif** temp\_df2[i] < temp\_df1[i]:  
 temp\_df1[i] = temp\_df2[i]  
  
 df = df.drop([word1, word2])  
 df = df.append(temp\_df1)  
  
 **return** df.index

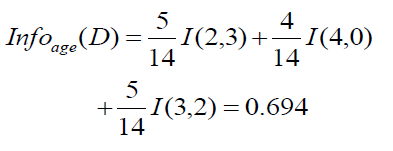


Rank 값이 큰 조합부터 확인을 한다. Rank 리스트에 있는 Similarity값과 Matrix의 Similarity값이 동일하다면 클러스터링을 진횅한다. 그렇게 되면 Df 사이즈는 가로 세로가 하나씩 감소한다. 이 Matrix가 완전히 1x1 행렬이 될때까지 진행하거나, 초기에 입력받은 변수 threshold 으로 내려가게 되면 함수를 종료한다.

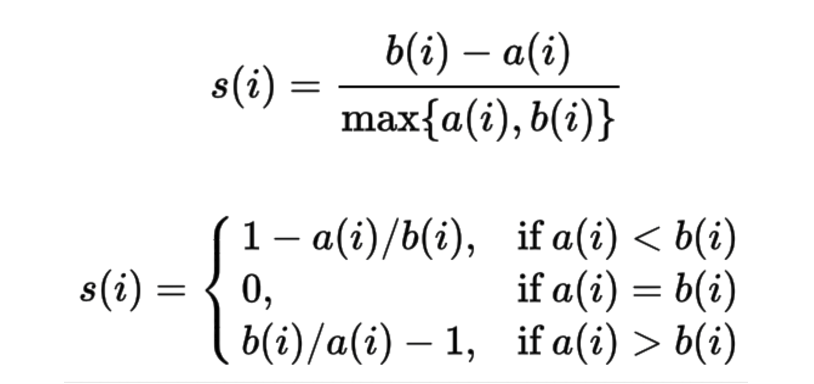
**def** SetClusterNumber (DB,Cluster):  
 **for** i **in** range(0, len(DB)) :  
 **for** j **in** range(0, len(Cluster)):  
 **if** DB[i][0] **in** Cluster[j] :  
 DB[i].append(j)  
 **break  
 return** DB

기존의 DB에서 WordTopic에서 알게된 정보를 새로운 column으로 추가한다.

**def** Info (lst):  
 ret = .0  
 w\_count = {}  
 start\_idx = 0  
 end\_idx = 0  
 lst\_T = [list(i) **for** i **in** zip(\*lst)]  
 **for** item **in** lst\_T[0]:  
 **try**:  
 w\_count[item] += 1  
 **except**:  
 w\_count[item] = 1  
  
 **for** val **in** w\_count.values():  
 ret -= val/len(lst) \* math.log(val/len(lst) ,2)  
 **return** ret  
  
**def** Info\_A (lst):  
  
 ret = .0  
 w\_count = {}  
 start\_idx = 0  
 end\_idx = 0  
 lst.sort(key = **lambda** x : (x[1], x[0]))  
 lst\_T = [list(i) **for** i **in** zip(\*lst)]  
 **for** item **in** lst\_T[1]:  
 **try**:  
 w\_count[item] += 1  
 **except**:  
 w\_count[item] = 1  
 **for** val **in** w\_count.values() :  
 end\_idx += val  
 ret += val / len(lst) \* Info(lst[start\_idx : end\_idx])  
 start\_idx = end\_idx  
  
 print(**"엔트로피 : "** +str(ret))



**def** Silhouette (DB):  
 ret = .0  
 inner\_dis = 0  
 w\_count = {}  
  
 count\_a = 0  
 count\_b = 0  
 newDB = DB[:]  
 newDB.sort(key=**lambda** x: x[3])  
 DB\_T = [list(i) **for** i **in** zip(\*newDB)]  
  
 silhouette = .0  
 print(DB\_T[3])  
 **for** item **in** DB\_T[3]:  
 **try**:  
 w\_count[item] += 1  
 **except**:  
 w\_count[item] = 1  
  
 **for** i **in** range(0, len(DB)):  
 start\_idx = 0  
 end\_idx = 0  
 a\_val = .0  
 b\_val = 10000.0  
  
  
 **for** val **in** w\_count.values():  
  
 sum = .0  
 idx = 0  
 word\_ProductSum = .0  
 end\_idx += val  
 **while** idx != val:  
  
 **for** j **in** range(0, 300):  
 word\_ProductSum += math.pow(newDB[start\_idx + idx][1][j] - newDB[i][1][j], 2)  
 sum += math.sqrt(word\_ProductSum)  
 idx += 1  
  
 **if** start\_idx <= i **and** i < end\_idx :  
 a\_val = sum / val  
 **elif** b\_val > (sum / val):  
 b\_val = sum / val  
 start\_idx += val  
  
 **if** b\_val < a\_val :  
 count\_a += 1  
 silhouette += (b\_val / a\_val - 1.0)  
 **elif** b\_val == a\_val :  
 silhouette += .0  
 **else** :  
 count\_b +=1  
 silhouette += (1.0 - a\_val / b\_val)  
  
 print(**"silhouette : "** + str(silhouette / len(DB)))  
 print(**"the number of a > b"** + str(count\_a)+ **" The number of a < b "** + str(count\_b))



실루엣 값은 한 클러스터 안의 데이터들이 다른 클러스터와 비교해서 얼마나 비슷한가를 나타낸다. 1. 클러스터 안의 거리가 짧을 수록 좋고(cohesion), 다른 클러스터와의 거리는 멀수록 좋다(separation) 2. 실루엣은 -1 부터 1사이의 값을 가진다. (실루엣이 1일수록 잘 부합하는 거고, -1일수록 필요없는 데이터) 3. 높을 수록 좋다.

모든 좌표에서 실루엣 값을 구한뒤 평균을 출력한다.

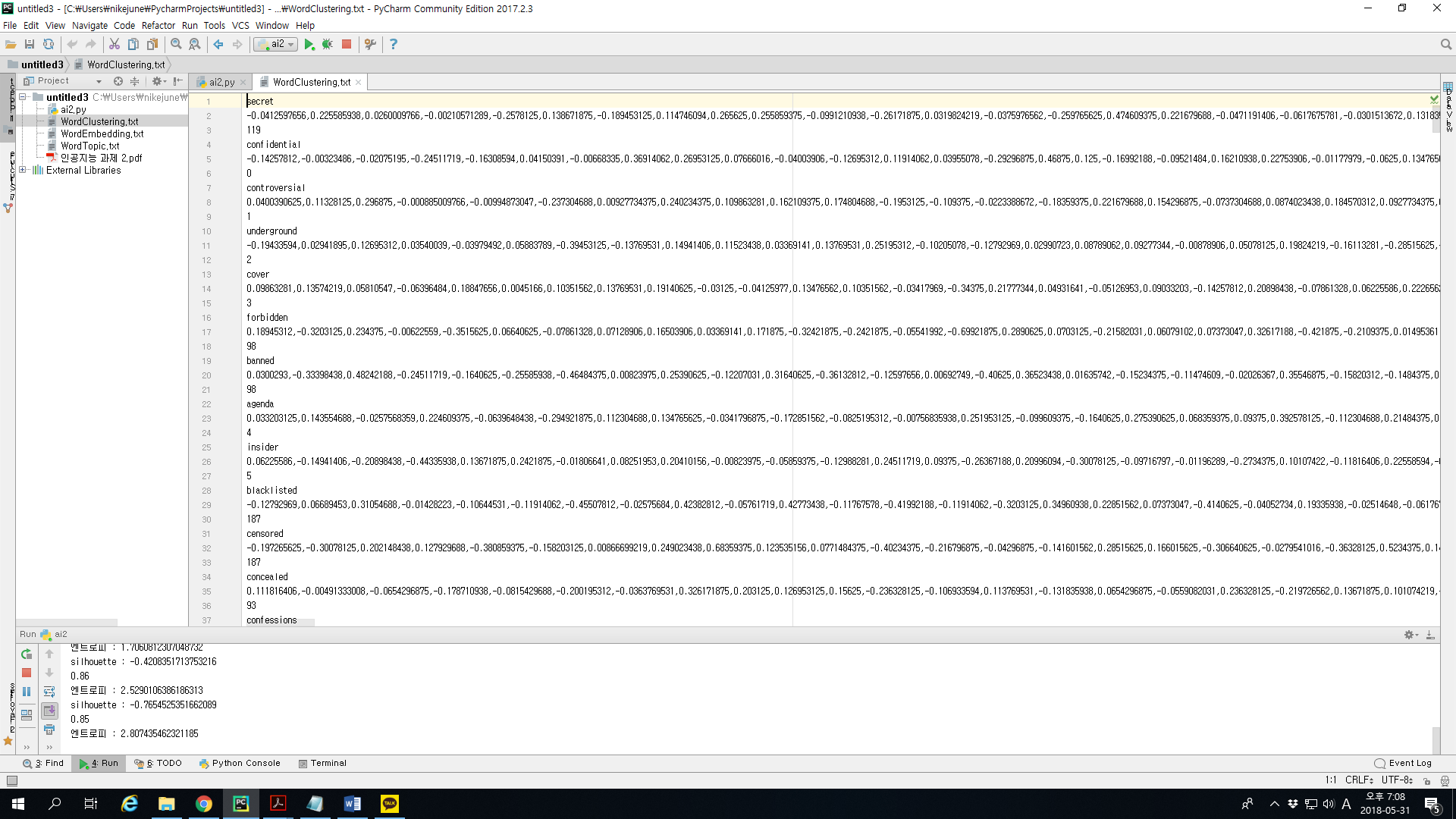
**실험결과**

사실 클러스터링 개수를 무수히 많이 설정할 경우 엔트로피 값은 좋게 나올 수 밖에 없다. 왜냐하면 하나의 그룹안에 하나의 원소만 있다면 엔트로피가 1로 나오게 된다. 하지만 338개의 단어를 331개로 클러스터링 하는 것은 아무 의미가 없다. 실루엣 값도 마찬가지다. 클러스터링 개수가 많으면 많을수록 좋게나온다는 단점이있다.

참고로 Euclidian 은 값이 골고루 퍼져있지 않아서 threshold를 0.89, 0.88, 0.86, 0.85 로 설정하였다.

개인적으로 Euclidian 방식은 백터적인 관점이 떨어진다고 판단하여 Cosine similarity로 결정하였고 실루엣값이 0보다 큰 값중 가장 작은 threshold 0.4로 설정하였다.





WordClustering.txt

**해당 코드에 대한 컴파일 방법 및 사용 버전**

OS : Windows 10 / IDE & compile : Pycharm Community Edition 2017.2.3

Language : Python

**추가적인 설치가 반드시 필요하다.**

pip3 install pandas

pip3 install numpy