

인공지능

Clustering & Word Embedding

2013011800 구장희

구성

- (1) 코드설명 : **data structure** , **flow** , 주요 **method** 설명 , 추가적인 **method**
- (2) **Implementaion detail** : 유사도 계산 **metric** , **clustering** 평가 **metric**
- (3) **Cluster** 평가결과 + 분석
- (4) 컴파일방법 + 사용버전

코드설명

[data structure]

(1) HAC class

```
public class HAC {  
    final String simtype = "euclidean"; // type of similarity measure  
    final String wordembedding = "WordEmbedding.txt"; // input-file name  
    final String wordtopic = "WordTopic.txt"; // class-file name  
    final String wordclustering = "WordClustering.txt"; // output-file name  
    Map<String, List<Double>> vectormap; // Map(String, WordEmbedding vector(dimension=300))  
    Map<Integer, Set<String>> clustormap; // Map(Cluster Index, Set of Strings in each cluster)  
    Map<String, Integer> wtoi; // Map(String, String Index)  
    Map<Integer, String> itow; // Map(String Index, String)  
    Map<Integer, PriorityQueue<Unit>> pqmap; // Map(String Index, PriorityQueue of Unit(String Index,similarity))  
    List<Pair> resultset; // List of Pair(String Index,String Index) as complete-link clustering goes  
    Double[][] smatrix; // similarity matrix(about all String Index-String Index pair)  
    int[] onelist; // check whether a String is XXXXXX in complete-link clustering  
    int[] stringIdxToClassNum, stringIdxToClusterNum; // String Index to ClassNum/ClusterNum  
    int[] clusterToClass; // # of Class in each Cluster  
    int word, classnum, clusternum; // # of word, # of class, # of cluster  
    double[] threshold = { 0.2, 0.4, 0.6, 0.8 }; // threshold for partitioning clusters  
    String[] type = { "euclidean", "cosine" }; // types of similarity measure  
    double smax = 0, smin = 2 * Math.sqrt(300); // max,min for euclidean-distance  
    double smax = 0, smin = 1.0; // max,min for cosine-similarity  
}
```

(2) Unit class

설명 : all word-word pair에서 유사도를 우선순위큐로 관리하기 위한 단위.

```
public class Unit implements Comparable<Unit> {  
    int index;  
    double sim;  
  
    public Unit(int _index, double _sim) {  
        index = _index;  
        sim = _sim;  
    }  
  
    @Override  
    public int compareTo(Unit o) {  
        // TODO Auto-generated method stub  
        if (this.sim < o.sim) {  
            return 1;  
        } else if (this.sim > o.sim) {  
            return -1;  
        } else {  
            return 0;  
        }  
    }  
}
```

(3) Pair class

설명 : all word-word pair에서 유사도를 관리하기 위한 단위.

```
public class Pair {  
    int kidx, lidx;  
    double sim;  
  
    public Pair(int _kidx, int _lidx, double _sim) {  
        this.kidx = _kidx;  
        this.lidx = _lidx;  
        this.sim = _sim;  
    }  
}
```

[flow]

main method에서 HAC class의 인스턴스를 생성하고 work() method를 호출한다.

HAC class의 인스턴스가 work() method내에서

- init() : 변수들을 초기화
- readInputfile() : input-file을 읽고 (word,wordembedding vector)를 저장한다.
- computeMatrix() : all word-word pair의 유사도에 대해서 행렬과 우선순위큐를 만든다.
- initialization() : 변수들을 초기화
- computeClustering() : 실제 clustering을 하는 부분
- readClassfile() : class-file을 읽고 (class,word)를 저장한다.
- partitioningAndAnalysis() : 정해진 threshold에 따라 cluster를 분할하고, 3가지 metric에 대해 평가.
- writeOutFile() : clustering결과를 output-file로 쓴다.

```
430 void work() {  
431     init();  
432     readInputfile();  
433     computeMatrix();  
434     initialization();  
435     computeClustering();  
436     readClassfile();  
437     partioningAndAnalysis();  
438     writeOutputFile();  
439 }  
440  
441 public static void main(String[] args) {  
442     new HAC().work();  
443 }  
444
```

[주요 method 설명]

(1) computeMatrix() : all word-word pair의 유사도에 대해서 행렬과 우선순위큐를 만든다.

- all word-word pair에 대해서 유사도를 계산해서 유사도 행렬에 저장한다.
 - 같은 word에 대해서는 유사도를 0
 - 다른 word에 대해서는 유사도를 euclidean-distance or cosine-similarity으로 계산.
- all word-word pair에 대해서 유사도 normalization해서 유사도 행렬에 다시 저장한다.
 - 같은 word에 대해서는 하지 않는다.
- all word-word pair의 유사도를 유사도행렬의 row마다 우선순위큐에 저장한다.(유사도의 max heap)

```
227 void computeMatrix() {
228     smatrix = new double[mcnt + 1][wcnt + 1];
229     for (String k : vectormap.keySet()) {
230         int kidx = wtoi.get(k);
231         for (String l : vectormap.keySet()) {
232             int lidx = wtoi.get(l);
233             if (kidx == lidx) {
234                 smatrix[kidx][lidx] = (double) 0;
235             } else {
236                 double sim = 0;
237                 if (simtype.equalsIgnoreCase(type[0])) {
238                     sim = euclideanDistance(vectormap.get(k), vectormap.get(l));
239                 } else {
240                     sim = cosinesimilarity(vectormap.get(k), vectormap.get(l));
241                 }
242                 smatrix[kidx][lidx] = sim;
243             }
244         }
245     }
246     for (String k : vectormap.keySet()) {
247         PriorityQueue<Unit> pq = new PriorityQueue<>();
248         int kidx = wtoi.get(k);
249         for (String l : vectormap.keySet()) {
250             int lidx = wtoi.get(l);
251             if (kidx != lidx) {
252                 double sim = smatrix[kidx][lidx];
253                 double temp = 0;
254                 if (simtype.equalsIgnoreCase(type[0])) {
255                     temp = (1 / (sim * sim) - 1 / (cmax * cmax)) / (1 / (cmin * cmin) - 1 / (cmax * cmax));
256                 } else {
257                     temp = (sim - cmin) / (cmax - cmin);
258                 }
259                 smatrix[kidx][lidx] = temp;
260                 pq.add(new Unit(lidx, smatrix[kidx][lidx]));
261             }
262         }
263         pqmap.put(kidx, pq);
264     }
265 }
```

(2) computeClustering() : 실제 clustering을 하는 부분

- (word의 갯수-1)번의 반복문을 돌면서
 - clustering 되지 않은(checklist로 체크) all word에 대해서, 그 word의 우선순위큐에서 peek()해서 나온 word 중 유사도행렬에서 최댓값을 가지는 pair와 유사도를 찾는다
 - 그 pair를 clustering한다.
 - pair의 첫번째 word의 우선순위큐를 초기화
 - pair의 두번째 word의 checklist를 초기화
 - clustering 되지 않고, pair의 첫번째 word가 아닌 all word에 대해서
 - complete-link clustering의 과정으로 유사도를 재 계산한다.
 - pair의 첫번째 word의 우선순위큐에 새로 계산된 유사도를 저장한다.
 - 나머지 word들의, pair의 첫번째 word에 대한 유사도 또한 우선순위큐에 update한다.

```
326 void computeClustering() {
327     for (int i = 1; i < wcnt; i++) {
328         int k1 = 0, k2 = 0;
329         double max = 0;
330         for (int j = 1; j <= wcnt; j++) {
331             if (chklist[j] == 1) {
332                 Unit u = pqmap.get(j).peek();
333                 if (Math.max(u.sim, max) == u.sim) {
334                     max = u.sim;
335                     k1 = j;
336                     k2 = u.index;
337                 }
338             }
339         }
340
341         resultset.add(new Pair(k1, k2, smatrix[k1][k2]));
342         chklist[k2] = 0;
343         pqmap.get(k1).clear();
344         for (String l : vectormap.keySet()) {
345             int lidx = wtoi.get(l);
346             if (lidx != k1 && chklist[lidx] == 1) {
347                 smatrix[k1][lidx] = Math.min(smatrix[lidx][k1], smatrix[lidx][k2]);
348                 smatrix[lidx][k1] = Math.min(smatrix[lidx][k1], smatrix[lidx][k2]);
349                 PriorityQueue<Unit> temppqmap = new PriorityQueue<Unit>();
350                 for (Unit u : pqmap.get(lidx)) {
351                     if (u.index != k1 && u.index != k2)
352                         temppqmap.add(u);
353                 }
354                 pqmap.get(lidx).clear();
355                 for (Unit u : temppqmap) {
356                     pqmap.get(lidx).add(u);
357                 }
358                 pqmap.get(k1).add(new Unit(lidx, smatrix[k1][lidx]));
359                 pqmap.get(lidx).add(new Unit(k1, smatrix[lidx][k1]));
360             }
361         }
362     }
363 }
364 }
```

(3) partitioningAndAnalysis() : 정해진 threshold에 따라 cluster를 분할하고, 3가지 metric에 대해 평가.

- all threshold에 대해서 partitioning을 진행하고, 3가지 metric(entropy,silhouett,dunnindex)로 clustering 결과를 평가한다.

```
405 void partitioningAndAnalysis() {  
407     double entropy = 0, silhouett = 0, dunnindex = 0;  
408     for (int i = 0; i < 4; i++) {  
409         partitioning(i);  
410         entropy = entropyanalysis();  
411         silhouett = silhouettanalysis();  
412         dunnindex = dunnindex();  
413         System.out.println("-----");  
414         System.out.println("threshold : " + threshold[i]);  
415         System.out.println("# of cluster : " + clusternum);  
416         System.out.println("entropy : " + entropy);  
417         System.out.println("silhouett : " + silhouett);  
418         System.out.println("dunnindex : " + dunnindex);  
419         System.out.println("-----");  
420     }  
421 }
```

(4) partitioning(double threshold) : clustering의 결과를 인자로 받은 threshold로 분할한다.

- word가 속한 class/cluster의 index를 각각 배열에 저장하여 관리해주면서
- 전역변수로 0으로 초기화된 clusternum 변수를 증가시키면서 cluster index를 할당해준다.
- clustering된 all word-word pair에 대해서
 - 유사도 > threshold이면
 - 두 word의 cluster index가 비어있다면, clusternum을 하나 더해주고 cluster index로 저장한다.
 - 두 word 중 하나의 word의 cluster index가 있다면, 나머지 word를 그 cluster index로 저장한다.
 - 두 word의 cluster index가 모두 있다면, do nothing.
 - 유사도 < threshold이면
 - 두 word의 cluster index가 비어있다면, clusternum을 각각 더하면서 cluster index로 저장한다.
 - 두 word 중 하나의 word의 cluster index가 있다면, 나머지 word를 clusternum을 하나 더해주고 그 cluster index로 저장한다.
 - 두 word의 cluster index가 모두 있다면, do nothing.
- cluster analysis의 편의를 위해서
 - cluster 별 class에 속한 word의 갯수를 세는 행렬을 만든다.
 - cluster 별 word를 저장해준다.

```
160 void partitioning(int idx) {
161     clusternum = 0;
162     stringIdxToClusterNum = new int[word + 1];
163     for (int i = 0; i < resultset.size(); i++) {
164         Pair pair = resultset.get(i);
165         if (Math.max(pair.sim, threshold[idx]) == pair.sim) {
166             if (stringIdxToClusterNum[pair.kidx] == 0 && stringIdxToClusterNum[pair.lidx] == 0) {
167                 clusternum++;
168                 stringIdxToClusterNum[pair.kidx] = clusternum;
169                 stringIdxToClusterNum[pair.lidx] = clusternum;
170             } else if (stringIdxToClusterNum[pair.kidx] == 0 && stringIdxToClusterNum[pair.lidx] != 0) {
171                 stringIdxToClusterNum[pair.kidx] = stringIdxToClusterNum[pair.lidx];
172             } else if (stringIdxToClusterNum[pair.kidx] != 0 && stringIdxToClusterNum[pair.lidx] == 0) {
173                 stringIdxToClusterNum[pair.lidx] = stringIdxToClusterNum[pair.kidx];
174             } else {
175             }
176         } else {
177             if (stringIdxToClusterNum[pair.kidx] == 0 && stringIdxToClusterNum[pair.lidx] == 0) {
178                 clusternum++;
179                 stringIdxToClusterNum[pair.kidx] = clusternum;
180                 clusternum++;
181                 stringIdxToClusterNum[pair.lidx] = clusternum;
182             } else if (stringIdxToClusterNum[pair.kidx] == 0 && stringIdxToClusterNum[pair.lidx] != 0) {
183                 clusternum++;
184                 stringIdxToClusterNum[pair.kidx] = clusternum;
185             } else if (stringIdxToClusterNum[pair.kidx] != 0 && stringIdxToClusterNum[pair.lidx] == 0) {
186                 clusternum++;
187                 stringIdxToClusterNum[pair.lidx] = clusternum;
188             } else {
189             }
190         }
191     }
192     clusterToClass = new int[clusternum + 1][classnum + 1];
193     for (int i = 1; i <= word; i++) {
194         int clusterIdx = stringIdxToClusterNum[i];
195         int classIdx = stringIdxToClassNum[i];
196         clusterToClass[clusterIdx][classIdx]++;
197     }
198     clustermap = new HashMap<>();
199     for (int i = 1; i <= clusternum; i++) {
200         clustermap.put(i, new HashSet<>());
201     }
202     for (int i = 1; i <= word; i++) {
203         clustermap.get(stringIdxToClusterNum[i]).add(iword.get(i));
204     }
205 }
```


[부가적인 method]

- double entropy(int[] arr) {}
- double cosinesimilarity(List<Double> l1, List<Double> l2) {}
- double euclideanistance(List<Double> l1, List<Double> l2) {}
- double entropyanalysis() {}
- double silhouetteanalysis() {}
- double dunnindex() {}

Implementation detail

[euclidean-distance]

실제 word-word간 euclidean-distance가 [3:5]의 분포를 보이므로
유사도가 거리에 반비례 or 거리의 제곱에 반비례하도록 [0:1]에 normalization했다.
(threshold를 변화시키지 않고 사용하기 위함)
(동일 word-word pair에 대해서는 최솟값 0으로)
결과는 유사도가 거리에 반비례하도록 하는것이 더 좋았다.

[cosine-similarity]

cosine-similarity또한 threshold를 변화시키지 않고 사용하기 위해서 [0:1]로 normalization했다.
(동일 word-word pair에 대해서는 최솟값 0으로)

[표 : 유사도 metric에 따른 word-word pair의 유사도 분포([0:1] 로 정규화)]

	0~0.2	0.2~0.4	0.4~0.6	0.6~0.8	0.8~1.0
sim $\propto 1/ed$	71	29	3	0.3	0.07
sim $\propto 1/ed^2$	98	11	0.5	0.1	0.03
sim $\propto 1/cs$	20	67	11	10	0.07

[결과파일 출력시]

word embedding vector의 각 value를 double형 변수로 바꾼 값을 적었다.
(실제로 근사한 차이가 있기 때문)

[clustering 평가 metric]

- (1) entropy : 명세에 의해 무조건 분석.
- (2) silhouett 지표 : 간단한 방법으로 데이터들이 얼마나 잘 클러스터링 되었는지를 나타낸다. 실루엣 지표가 1에 가까울 수록 데이터 i 는 올바른 클러스터에 분류된 것이며, -1에 가까울 수록 잘못된 클러스터에 분류되었음을 나타낸다.
- (3) dunn index : 밀도가 높고 잘 나뉜 클러스터링 결과를 목표로 하는 metric. Dunn index값이 높은 클러스터링 알고리즘은 클러스터링 성능이 좋은 것으로 판단할 수 있다.
- (4) 외부 평가 지표 : 본 과제에 적용 불가.

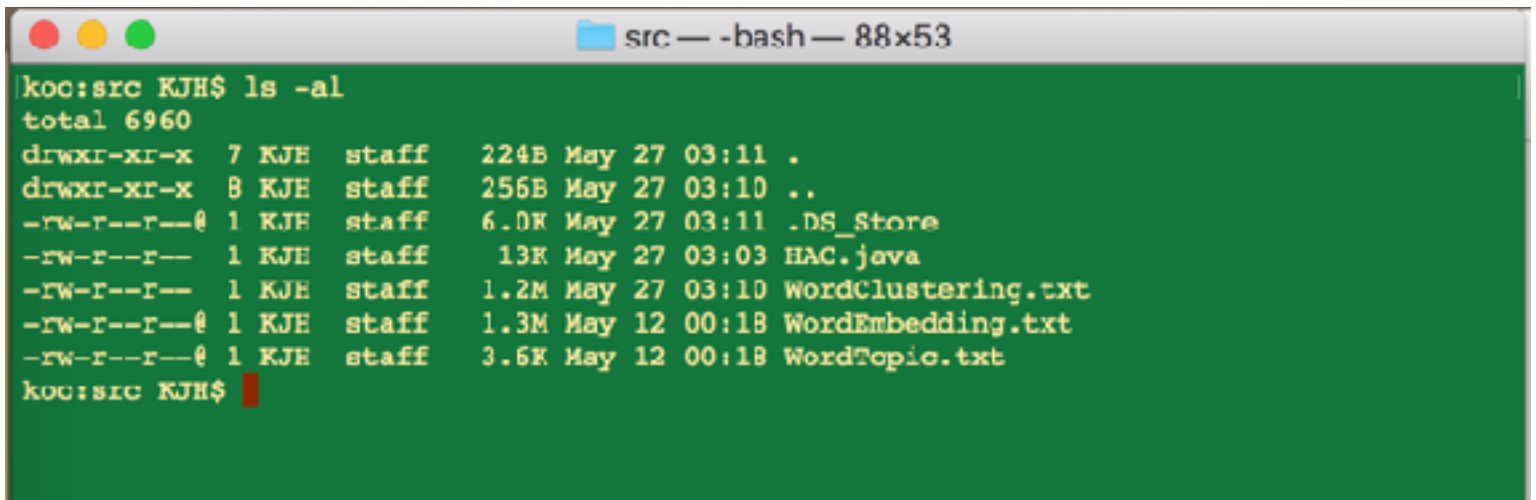
Clustering 평가 결과

	euclidean-distance	cosine-similarity
threshold=0.2	threshold : 0.2 # of cluster : 146 entropy : 0.67773 silhouett : -0.63039 dunnindex : 7.30051	threshold : 0.2 # of cluster : 129 entropy : 0.70417 silhouett : -0.67313 dunnindex : 4.88468
threshold=0.4	threshold : 0.4 # of cluster : 270 entropy : 0.19712 silhouett : 0.33573 dunnindex : 13.83416	threshold : 0.4 # of cluster : 144 entropy : 0.61326 silhouett : -0.5472 dunnindex : 6.09727
threshold=0.6	threshold : 0.6 # of cluster : 320 entropy : 0.04365 silhouett : 0.81377 dunnindex : 17.22967	threshold : 0.6 # of cluster : 225 entropy : 0.32807 silhouett : 0.02197 dunnindex : 12.61803
threshold=0.8	threshold : 0.8 # of cluster : 334 entropy : 0.00592 silhouett : 0.94274 dunnindex : 18.42827	threshold : 0.8 # of cluster : 310 entropy : 0.06141 silhouett : 0.74343 dunnindex : 21.52157

- 해석
 - 최고 성능 : similarity-measure(cosine-similarity) , threshold(0.4)
 - 유사도 metric을 euclidean-distance로 하는 경우가 3가지 clustering 평가 지표에서 모두 앞섰다. 하지만 euclidean-distance의 경우 cluster의 갯수가 상대적으로 많았기 때문에 절대적으로 앞선다고 하기 어렵다.
 - cosine-similarity의 경우 클러스터의 갯수가 그리 크지 않으면서 평가지표에서 euclidean-distance보다 많이 지지 않았다.
- 결론
 - 클러스터링이 가장 잘 된 경우 : similarity-measure(cosine-similarity) , threshold(0.4)
 - 이유 : 클러스터의 갯수가 너무 많지 않으면서 평가 지표가 그리 나쁘지 않았기 때문

컴파일방법

- 환경
 - OS : Mac OS
 - Language : Java
- 실행 이전

A screenshot of a macOS terminal window titled 'src — -bash — 88x53'. The terminal has a green background and displays the output of the command 'ls -al'. The output shows a directory listing with permissions, owner, group, size, date, and filename. The files listed are '.', '..', '.DS_Store', 'HAC.java', 'WordClustering.txt', 'WordEmbedding.txt', and 'WordTopic.txt'.

```
koc:src KJH$ ls -al
total 6960
drwxr-xr-x  7 KJE  staff   224B May 27 03:11 .
drwxr-xr-x  8 KJE  staff   256B May 27 03:10 ..
-rw-r--r--@ 1 KJE  staff   6.0K May 27 03:11 .DS_Store
-rw-r--r--  1 KJE  staff   13K May 27 03:03 HAC.java
-rw-r--r--  1 KJE  staff  1.2M May 27 03:10 WordClustering.txt
-rw-r--r--@ 1 KJE  staff  1.3M May 12 00:18 WordEmbedding.txt
-rw-r--r--@ 1 KJE  staff  3.6K May 12 00:18 WordTopic.txt
koc:src KJH$
```

- 컴파일 방법
 - 컴파일: \$javac HAC.java;
 - 실행: \$java HAC;

```
src — -bash — 88x53
koo:src KJH$ javac HAC.java ; java HAC
=====
threshold : 0.2
# of cluster : 129
entropy : 0.70417
silhouett : -0.67313
dunnindex : 4.88468
=====
threshold : 0.4
# of cluster : 148
entropy : 0.60813
silhouett : -0.53087
dunnindex : 6.40181
=====
threshold : 0.6
# of cluster : 228
entropy : 0.31371
silhouett : 0.03633
dunnindex : 12.84975
=====
threshold : 0.8
# of cluster : 311
entropy : 0.05325
silhouett : 0.7483
dunnindex : 21.60638
=====
koo:src KJH$
```

- 컴파일 및 실행시 주의사항

제출시 코드파일명 수정했음(assignment2_2013011800.java)
default package임