**Algorithm Concepts**

Clustering Project Report

1. Introduction

In computer science, an algorithm is a step-by-step procedure for calculations. Algorithms are used for calculation, data processing, and automated reasoning. Clustering is a kind of algorithm about data processing; it’s a way of grouping together data samples that are similar in some way-according to some criteria that we pick. So it’s a method of data exploration, a way of looking for patterns of structure in the data that are of interest. My project is developing a system that can cluster. After analyzed and designed our project. We divided our project into four steps. Firstly we crawl the news that we interested from the internet; secondly we preprocess the data we got in the first steps, clear and organize it; and then in the third step, we convert data to math model-Vector Space Model (VSM), in VSM model the tf and idf are main parameter we should calculate; finally based on the VSM model, we use K-means Algorithm to process our data and then cluster it into 20 clusters.

**My contribution to the team is the fourth step – K-means Algorithm. I organized the K-means Algorithm and implemented it with my partner.**

1. Theory and Algorithm

Since we had divided our project into four steps, we should analyze and design it for each of step. In detail, at this stage, we should deeply study the theory and algorithms to analyze the pros and cons of various algorithms, and to choose a good fit algorithm for our project.

* 1. Crawl the news

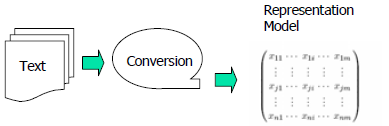
This is the first step in our project, it mainly function is to build a crawler to crawl the news that we interested from internet. First of all, we get two feasibility of establishing web crawler program, one is using open source crawler such as Heritrix, WebSPHINX and so on, and the other is to build a crawler by ourselves. They both have their advantages and disadvantages, open source crawler powerful, and easy to use, but it has a huge amount of code, and there are many features that we might don’t use; writing our own crawler will take more time, but it would be streamlined, better controllability, and it don’t difficult to implement. Considering the scale of our projects, we determine to write our own crawler.

* 1. Preprocessing

After we designed our crawler, we got the data of news that we need. And then we should preprocess the data, thus we enter the next stage-preprocessing. In preprocessing stage, it has three steps: Splitting, deleting and Stemming. First we split the news into single word; and second we delete stop words such as “of”, “a”, “by”, “and”, “the”; and then we stem the data, remove inflections that convey parts of speech, tense. After preprocessing, we get the keywords of each of news, and then we can build our math model.

* 1. Convert to math model (VSM)

In preprocessing, we got the keywords of each of news, thus we can convert our data to math model-Vector Space Model (VSM), as shown in the picture below.



In order to build our VSM model, we should calculate the weighting terms for each keyword to weight the frequency and importance of keyword in our data. The tf, idf parameter is needed in that step, it calculated as follows:



Where tf (dj, ti) is the frequency of term ti in document di, |D| is the total number of documents, and df (ti) is the number of documents in which ti occurs.

* 1. K-means Algorithm—**What I have done**

My work mainly focuses on this module, what I have done is the K-means Algorithm analysis, design and implementation. K-means is a method of [cluster analysis](http://en.wikipedia.org/wiki/Cluster_analysis) which aims to [partition](http://en.wikipedia.org/wiki/Partition_of_a_set) n observations into k clusters in which each observation belongs to the cluster with the nearest [mean](http://en.wikipedia.org/wiki/Mean). This results in a partitioning of the data space into [Voronoi cells](http://en.wikipedia.org/wiki/Voronoi_cell" \o "Voronoi cell). K-means Algorithm is carried out in four steps:

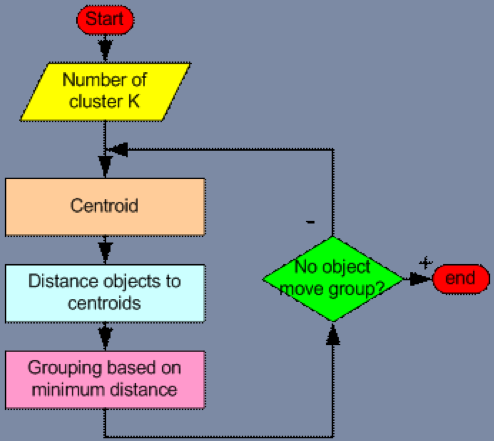
Firstly, given the cluster number k and set seed points, this is the initialization of algorithm;

Secondly, assign each object to the cluster with the nearest seed point;

Thirdly, compute seed points as the centroids of the clusters of current partition (the centroid is center, i.e. mean point of the cluster);

Finally, go back to step 2, stop when no more new assignment.

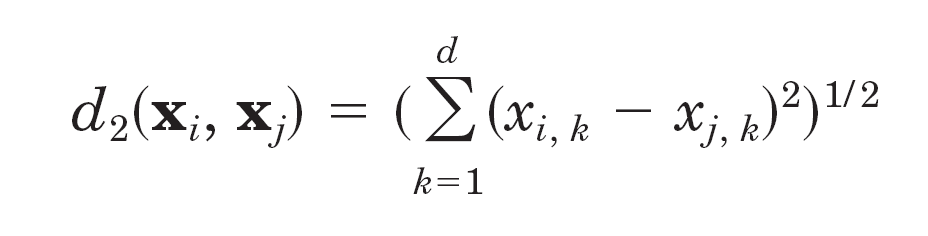
K-means Algorithm flow chart is as follows：



Now let’s think in detail about K-means Algorithm.

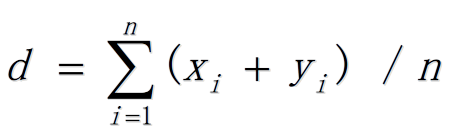
In the first step of K-means Algorithm, there are two points we should consider: 1, how to determine the K, the number of clusters, in K-means Algorithm, the number of k is not easy to choose unless we have preview of our data, but actually we do not know the detail of data, fortunately the teacher gave us advance the value of k; 2, how to select the initial seed points. In most cases, we get the initial seed points by randomly, but I did a little improvement, I random select the points that are already exist.

In the second step, in order to assign each object to the cluster with the nearest seed point, we need calculate the distance of two points, algorithm is as follows:

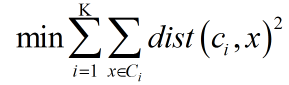


For each of object point, we calculate the distance between it and each of centroid point, and then find the minimum distance, thus the object point is assign to the minimum distance centroid point.

In the third step, we already had clustered our data in to K clusters, at this step we should recomputed seed points as the centroids of the clusters of current partition. In order to calculate the new centroids, we use the following formula to calculate the central value of each dimension.



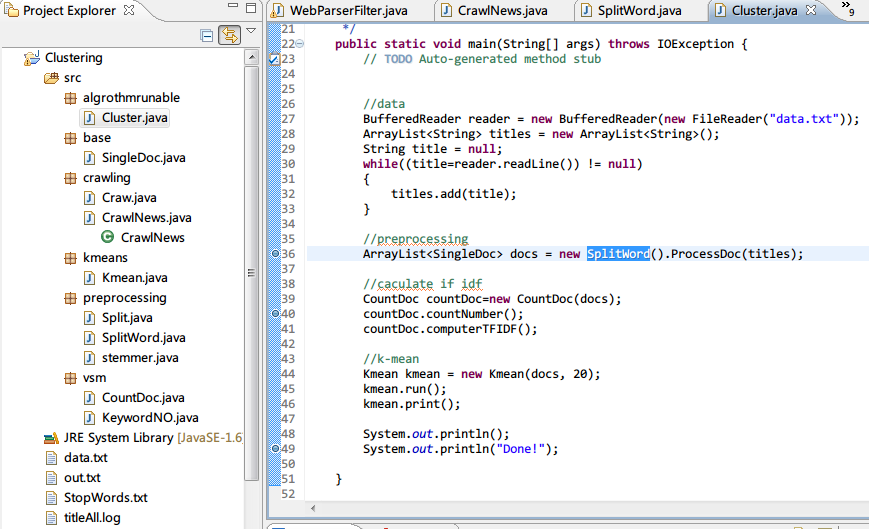
In the last step, when no more new assignment we should stop the algorithm, so we need to find a way to determine when we should stop. So the following formula is used to determine when to stop. We store an old dist. of all cluster, and for each iteration, we recalculate the new dist. When the difference between the old and the new is less than 1, we stop algorithm.

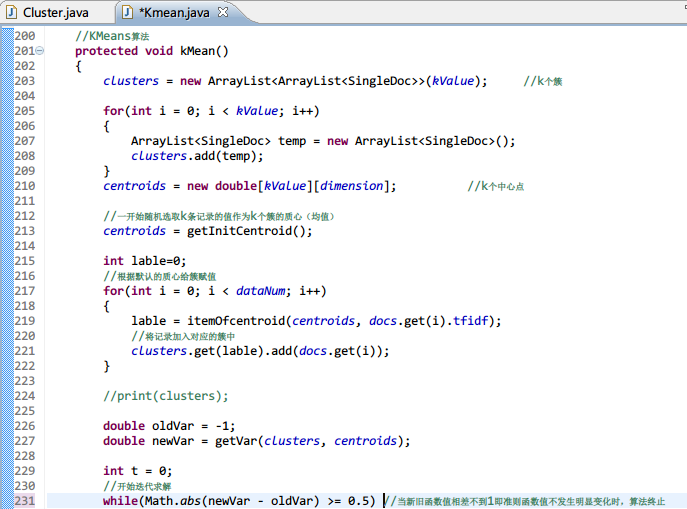


1. Result

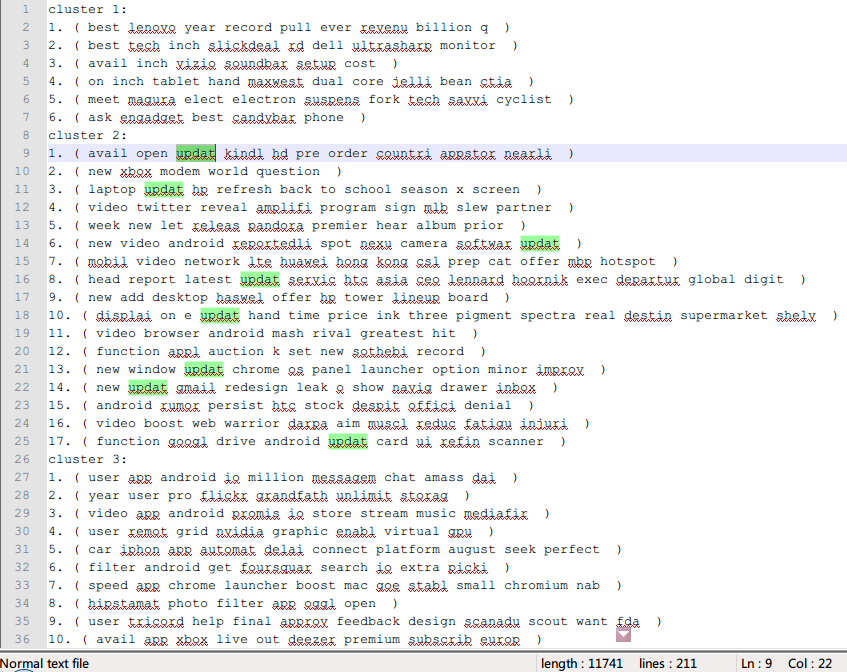
Since we had analyzed and design our project algorithm, we divided our team into five parts, and assigned to each part of their work. When all part of our team finished their work, our project finished.

Here is our project structure and the code of my work:





After we execute our project, we get the clusters of our data:



We can see from the result that the cluster 1 has 6 records, the cluster 2 has 17 records, the cluster 3 has 10 records, and there are other 17 clusters that don’t show in this picture. We can also see that the cluster 2 mainly talk about update about the news, and the cluster 3 mainly talk about app.

1. My code
2. **package** kmeans;
3. **import** java.io.\*;
4. **import** java.util.\*;
5. **import** vsm.KeywordNO;
6. **import** base.SingleDoc;
7. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
8. \*
9. \* K-Mean算法
10. \* 外界只需要使用到Kmean类中的 构造函数：public Kmean(ArrayList<SingleDoc> d, int k)
11. \* 执行函数：public void run()
12. \* 打印函数：public void print()
13. \* 设置K值：public void setKValue(int k)
14. \* 查看K值：public int getKValue()
15. \* 取出聚类：public ArrayList<ArrayList<SingleDoc>> getClusters()
16. \* 函数使用方法
17. \* 构造函数：public Kmean(ArrayList<SingleDoc> d, int k)
18. \* 输入1：ArrayList<SingleDoc>类型的SingleDoc的ArrayList 即文档总数
19. \* 输入2：算法的聚类数k
20. \* 执行函数：public void run()
21. \* 运行K-Mean算法
22. \* 打印函数：public void print()
23. \* 打印运行后的聚类结果
24. \* 附：可以根据需要修改print()函数，输出自己需要的结果
25. \* 设置K值：public void setKValue(int k)
26. \* 重新设置k值
27. \* 查看K值：public int getKValue()
28. \* 取出k值查看
29. \* 取出聚类：public ArrayList<ArrayList<SingleDoc>> getClusters()
30. \* 取出聚类
31. \*
32. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
33. **public** **class** Kmean
34. {
35. // k 值
36. **protected** **static** **int** *kValue*;
37. //数据记录的数目
38. **protected** **static** **int** *dataNum*;
39. //每个记录的维度
40. **protected** **static** **int** *dimension*;
41. //总的数据记录
42. **protected** **static** ArrayList<SingleDoc> *docs*;
43. //输入需要聚类的数据
44. //double [][]clusterData;
45. // k 个中心
46. **protected** **static** **double**[][] *centroids*;
47. // k 个簇
48. **protected** **static** ArrayList<ArrayList<SingleDoc>> *clusters*;
49. //构造函数
50. **public** Kmean(ArrayList<SingleDoc> d, **int** k)
51. {
52. *docs* = d;
53. *kValue* = k;
54. *dataNum* = d.size();
56. *dimension* = *docs*.get(0).tfidf.length;
57. }
58. //执行函数
59. **public** **void** run()
60. {
61. kMean();
63. }
64. //计算两个元组间的距离
65. **protected** **double** getDistXY(**double**[] x, **double**[] y)
66. {
67. **if**(x.length != *dimension* || y.length != *dimension*)
68. **return** -1; //输入数据维数不一致
70. **double** sum = 0;
71. **for**(**int** i = 0; i < *dimension*; i++)
72. {
73. sum += (x[i]-y[i]) \* (x[i]-y[i]);
74. }
75. **return** Math.*sqrt*(sum);
76. }
77. //随机选择 K 个原有记录作为初始 质心
78. **protected** **double**[][] getInitCentroid()
79. {
80. **double**[][] centroids = **new** **double**[*kValue*][*dimension*];
82. //生成 k 个不同的随机数
83. **int**[] intRan = **new** **int**[*kValue*];
84. **int** intRd = 0; //存放随机数
85. **int** count = 0; //记录生成的随机数个数
86. **int** flag = 0; //是否已经生成过标志
87. **while**(count < *kValue*){
88. Random rdm = **new** Random();
89. intRd = Math.*abs*(rdm.nextInt()) % *dataNum*;
90. **for**(**int** i = 0; i < count; i++){
91. **if**(intRan[i] == intRd){
92. flag = 1;
93. **break**;
94. }**else**{
95. flag = 0;
96. }
97. }
98. **if**(flag==0){
99. intRan[count] = intRd;
100. count++;
101. }
102. }
103. // k 个初始质心
104. **for**(**int** i = 0; i < *kValue*; i++)
105. {
106. **for**(**int** j = 0; j < *dimension*; j++)
107. {
108. centroids[i][j] = *docs*.get(intRan[i]).tfidf[j];
109. }
110. }
111. **return** centroids;
112. }
113. //根据质心，决定当前元组属于哪个 质心 为中心的簇
114. **protected** **int** itemOfcentroid(**double**[][] centroids, **double**[] item){
115. **double** dist=getDistXY(centroids[0],item);
116. **double** tmp;
117. **int** label=0; //标示属于哪一个簇
118. **for**(**int** i = 1; i < *kValue*; i++){
119. tmp = getDistXY(centroids[i],item);
120. **if**(tmp < dist)
121. {
122. dist = tmp;
123. label = i;
124. }
125. }
126. **return** label;
127. }
128. //获得当前簇的均值（质心）
129. **protected** **double**[] getCentroid(ArrayList<SingleDoc> cluster)
130. {
131. //簇中记录数
132. **int** num = cluster.size();
133. //新的质心
134. **double**[] newCentroid = **new** **double**[*dimension*];
135. **for**(**int** i = 0; i < *dimension*; i++)
136. {
137. newCentroid[i] = 0;
138. }
140. **for** (**int** i = 0; i < *dimension*; i++)
141. {
142. **for**(**int** j = 0; j < num; j++)
143. {
144. newCentroid[i] += cluster.get(j).tfidf[i];
145. }
146. }
147. **for**(**int** i = 0; i < *dimension*; i++)
148. {
149. **if**(num != 0)
150. {
151. newCentroid[i] /= num;
152. }
153. }
155. **return** newCentroid;
156. }
157. //获得给定簇集的平方误差
158. **protected** **double** getVar(ArrayList<ArrayList<SingleDoc>> clusters,**double**[][] centroids)
159. {
160. **double** var = 0;
161. **for** (**int** i = 0; i < *kValue*; i++)
162. {
163. ArrayList<SingleDoc> cluster = clusters.get(i);
164. **for** (**int** j = 0; j< cluster.size(); j++)
165. {
166. **double** dis = getDistXY(cluster.get(j).tfidf, centroids[i]);
167. var += dis \* dis;
168. }
169. }
170. **return** var;
171. }
172. //KMeans算法
173. **protected** **void** kMean()
174. {
175. *clusters* = **new** ArrayList<ArrayList<SingleDoc>>(*kValue*); //k个簇
177. **for**(**int** i = 0; i < *kValue*; i++)
178. {
179. ArrayList<SingleDoc> temp = **new** ArrayList<SingleDoc>();
180. *clusters*.add(temp);
181. }
182. *centroids* = **new** **double**[*kValue*][*dimension*]; //k个中心点
183. //一开始随机选取k条记录的值作为k个簇的质心（均值）
184. *centroids* = getInitCentroid();
185. **int** lable=0;
186. //根据默认的质心给簇赋值
187. **for**(**int** i = 0; i < *dataNum*; i++)
188. {
189. lable = itemOfcentroid(*centroids*, *docs*.get(i).tfidf);
190. //将记录加入对应的簇中
191. *clusters*.get(lable).add(*docs*.get(i));
192. }
193. **double** oldVar = -1;
194. **double** newVar = getVar(*clusters*, *centroids*);
196. **int** t = 0;
197. //开始迭代求解
198. **while**(Math.*abs*(newVar - oldVar) >= 0.5) //当新旧函数值相差不到1即准则函数值不发生明显变化时，算法终止
199. {
200. System.*out*.println("第 "+ ++t +" 次迭代开始：");
201. //更新每个簇的中心点
202. **for** (**int** i = 0; i < *kValue*; i++)
203. {
204. *centroids*[i] = getCentroid(*clusters*.get(i));
205. }
206. //计算新的准则函数值
207. oldVar = newVar;
208. newVar = getVar(*clusters*,*centroids*);
209. //清空每个簇
210. **for** (**int** i = 0; i < *kValue*; i++)
211. {
212. *clusters*.get(i).clear();
213. }
214. //根据新的质心获得新的簇
215. **for**(**int** i=0; i < *dataNum*; ++i)
216. {
217. lable = itemOfcentroid(*centroids*, *docs*.get(i).tfidf);
218. //将记录加入对应的簇中
219. *clusters*.get(lable).add(*docs*.get(i));
220. }
221. }
222. //给聚类中记录排序
223. relevancySort();
224. }
225. //给每个聚类中相关度排序
226. **protected** **void** relevancySort()
227. {
228. **for**(**int** i = 0; i < *clusters*.size(); i++)
229. {
230. ArrayList<SingleDoc> cluster = *clusters*.get(i);
231. //对于每一个簇 排序
232. **double**[] dis = **new** **double**[*dimension*];
233. **for** (**int** j = 0; j< cluster.size(); j++)
234. {
235. dis[j] = getDistXY(cluster.get(j).tfidf, *centroids*[i]);
236. }
237. **for** (**int** ii = 0; ii < cluster.size() - 1; ii++) //从第一个位置开始
238. {
239. **int** index = ii;
240. **for** (**int** jj = ii + 1; jj < cluster.size(); jj++) //寻找最小的数据索引
241. **if** (dis[jj] < dis[index])
242. index = jj;
243. **if** (index != ii) //如果最小数位置变化则交换
244. {
245. //DataSwap(&pDataArray[index], &pDataArray[ii]);
246. **double** temp = dis[index];
247. dis[index] = dis[ii];
248. dis[ii] = temp;
250. SingleDoc tempdoc = cluster.get(index);
251. cluster.set(index, cluster.get(ii));
252. cluster.set(ii, tempdoc);
253. }
254. }
255. }
256. }
257. //设置 k 值
258. **public** **void** setKValue(**int** k)
259. {
260. *kValue* = k;
261. }
262. //取出 k 值
263. **public** **int** getKValue()
264. {
265. **return** *kValue*;
266. }
267. //取出聚类
268. **public** ArrayList<ArrayList<SingleDoc>> getClusters()
269. {
271. **return** *clusters*;
272. }
273. //输出
274. **public** **void** print() **throws** IOException
275. {
277. PrintWriter out=**new** PrintWriter(**new** BufferedWriter(**new** FileWriter("out.txt")));
278. **for**(**int** i = 0; i < *clusters*.size(); i++)
279. {
280. ArrayList<SingleDoc> cluster = *clusters*.get(i);
281. **for**(**int** j = 0; j < cluster.size(); j++)
282. {
283. String info = (j+1) + ". ( ";
284. ArrayList<KeywordNO> keywords=cluster.get(j).afterKeyword;
285. **for**(**int** k = 0; k < *dimension*; k++)
286. {
287. **if**(keywords.get(k).num!=0)
288. info += keywords.get(k).keyword + " ";
289. }
290. info += " )";
291. out.println(info);
292. }
293. }
294. out.flush();
295. }
296. }