Lecture 12
Seriation/Clustering

Breitzman 4/24/2017

Introduction

- We haven’t done much with clustering in Data Mining 2.
- In Data Mining 1 we talked about k-means clustering and hierarchical clustering
- Tonight we’ll talk about seriation.
- Seriation can be used with clustering to clean up the results or if we’re lucky…
  - It can be used by itself for clustering
- Go to http://www.atgc-montpellier.fr/permutmatrix/manual/SeriationCorps.htm

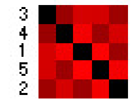
Simple Example Stolen from Previous Website

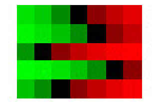
Simple example n°2

Dissimilaritiy matrix *D* | Graphical representation of *D* | Graphical data matrix

Step 1 — The smallest dissimilarity in *D* is 2,58. This value is already near the diagonal. No movement is necessary at this step.

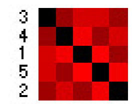
D	3	4	1	5	2
3	0,00				
4	2,58	0,00			
1	5,33	7,90	0,00		
5	5,22	2,65	10,55	0,00	
2	2,73	5,29	2,61	7,94	0,00

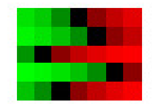




Step 2 — The dissimilarity between items 1 and 2 is now the smallest dissimilarity available.
Dmin = 2,61.
To put this value against the diagonal, we swap items 2 and 5.

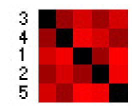
D	3	4	1	5	2
3	0,00				
4	2,58	0,00			
1	5,33	7,90	0,00		
5	5,22	2,65	10,55	0,00	
2	2,73	5,29	2,61	7,94	0,00

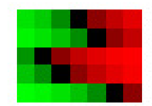




Step 3 — At this step, the dissimilarity between items 4 and 5 is the currently smallest dissimilarity. Items 4 and 3 were pooled at the first step and we are not allowed to modify this pair. We must put item 5 between items 4 and 1. At the end of this step, items 3, 4 and 5 form a "*fragment*" of three elements.

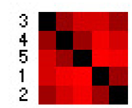
D	3	4	1	2	5
3	0,00				
4	2,58	0,00			
1	5,33	7,90	0,00		
2	2,73	5,29	2,61	0,00	
5	5,22	2,65	10,55	7,94	0,00

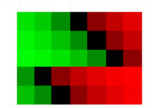




Step 4 — For the final step, the dissimilarity between items 2 and 3 is the smallest dissimilarity available. To put this value against the diagonal, we swap fragments{1,2} and {3,4,5}.

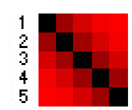
D	3	4	5	1	2
3	0,00				
4	2,58	0,00			
5	5,22	2,65	0,00		
1	5,33	7,90	10,55	0,00	
2	2,73	5,29	7,94	2,61	0,00

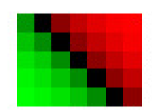




End — At the end of this algorithm, small dissimilarities are spread along the main diagonal of *D*.
Close elements are close in the seriation.

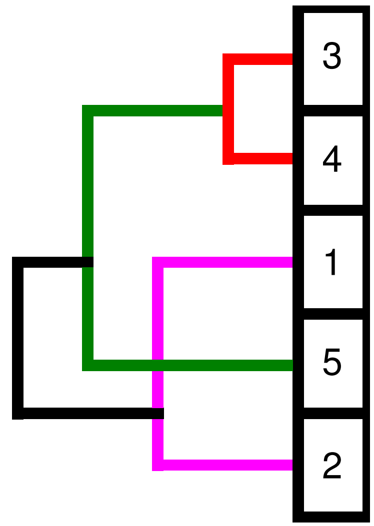
D	1	2	3	4	5
1	0,00				
2	2,61	0,00			
3	5,33	2,73	0,00		
4	7,90	5,29	2,58	0,00	
5	10,55	7,94	5,22	2,65	0,00

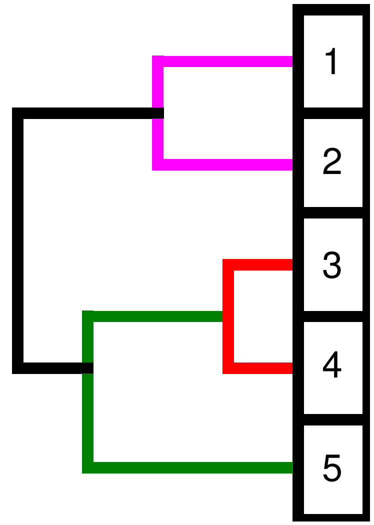




Hierarchical Clusters from Before and After Seriation

- Note seriation (from previous page) cleans up the hierarchy
- In the non-seriated hierarchy we have the green cluster crossing the pink cluster
- Note also that things that belong together tend to be next to each other even without the clustering
- That is, 1 is next to 2, 3 is next to 4, (3,4) is next to 5 etc
- If we plotted this as a heatmap, after seriation we could see where we should make our clusters without running a clustering algorithm
- This will become more obvious when we do our real example





The Setup…

- Last semester in Text Mining, Chris, John, Matt, Mihir, Parvati, Yousuf and others created a corpus of Linked-in docs, web pages, press releases, google scholar articles etc. on a topic of their choice
- They then removed stopwords, lemmatized, etc. their corpus
- After the semester I put all 1,292 documents into one giant corpus to test whether k-means clustering would be smart enough to separate each set into it's proper topic
- Results are on the next page

Results

	Cluster #	True Positive (Belongs)	False Positive (Doesn't Belong)	False Negative (Should be in)	F Score
Chris	9	191	40	0	0.905
Eric	3	153	12	0	0.962
John	0	111	8	0	0.965
Matt	4	78	11	19	0.839
Mihir	7	97	31	2	0.855
Parvati	1	43	2	190	0.309
Stephen	5	98	34	1	0.848
Tim	6	100	12	0	0.943
Tony	2	146	16	0	0.948
Yousuf	8	63	46	0	0.733
all	all	1080	212	212	0.836

	word1	word2	word3	word4	word5	word6	word7	word8	word9	word10
Cluster 0	limb	prosthetic	advanced	arm	artificial	prosthesis	hand	amputee	control	darpa
Cluster 1	biotechnology	university	verified	game	email	edu	video	information	engineering	science
Cluster 2	text	mining	information	document	analysis	literature	gene	tool	biomedical	drug
Cluster 3	trump	donald	clinton	woman	debate	campaign	republican	presidential	president	people
Cluster 4	biophysics	chemistry	journal	biophysical	physical	molecular	et	cell	al	patent
Cluster 5	quantum	compute	computer	physic	qubits	university	institute	science	state	waterloo
Cluster 6	airplane	wing	plane	flight	fly	mode	design	pilot	aircraft	jan
Cluster 7	machine	learning	learn	proceeding	international	conference	cloud	google	api	workshop
Cluster 8	nanotechnology	nature	nanoscience	ibm	ieee	nanomaterials	tj	nano	center	watson
Cluster 9	game	video	play	player	violent	study	time	world	child	gaming

Results (II)

- Note that k-means does the best on cluster 0 (prosthetics) because the topic has nothing to do with the others
- It also does well on the politics, airplane, and gaming clusters
- It inexplicably puts gaming in the biotechnology cluster though
- Overall it does a decent job, and we could make it better by making things like email, et al, jan, edu, etc. into stopwords

Seriation of Documents

- The question is whether we could do a decent job of clustering the documents with seriation?
- How?
- Take entire corpus; do TF/IDF to get a list of 1000 or 2000 most important words
- (A human edited list would work better, but we don't have time)
- Each document is then a 2000 dimension sparse vector with term frequencies
- Seriation puts closest vectors together which ought to work as a clustering method

Step 1: Read in Corpus

```
In [174]: from sklearn.datasets import load_files
     ...:
     ...: ## folder containing 1292 lemmatized text files (stopwords removed)
     ...: ## about 100 files each under names Chris, Parvati, etc.
     ...: p="E:\Rowan\Classes\TextMining\hw3CaseStudy\LemmatizeWordContainer"
     ...:
     ...:
     ...:
     ...: ## Load all files with one command
     ...: dataset=load_files(p, description=None, categories=None,
     ...: load_content=True, shuffle=True, encoding=None,
     ...: decode_error='strict', random_state=0)
     ...:
     ...:
     ...:
     ...: print("%d documents" % len(dataset.data))
     ...: print("%d categories" % len(dataset.target_names))
     ...: print(dataset.target_names[0:9])

1292 documents
10 categories
['Chris', 'Eric', 'John', 'Matt', 'Mihir', 'Parvati', 'Stephen', 'Tim', 'Tony']
```

- A great thing about sklearn is that if you keep your corpus as a bunch of text files in a hierarchical directory, load_files will load all of the files and keep the directory structure; all in one line

Step 2: Make List of all words; sort by freq.

```
In [175]: wordDict = {}
     ...: for i in range(len(dataset.data)):
     ...:  words = dataset.data[i].split()
     ...:  for word in words:
     ...:       if wordDict.has_key(word):
     ...:            wordDict[word]+=1
     ...:       else:
     ...:            wordDict.setdefault(word,1)
     ...:
     ...: import operator
     ...: wordDist = [sorted(wordDict.iteritems(), key=operator.itemgetter(1),reverse=True)]
     ...: for s in wordDist:
     ...:  for (v,k) in s:
     ...:       if (k > 1000):
     ...:            print(v+" "+str(k))
game 4623
trump 3727
video 1904
penn 1540
anonymous 1268
time 1190
quantum 1052
airplane 1004
```

- A Python dictionary is useful for keeping track of the occurrences of each word
- Don’t know why ‘penn’ and ‘anonymous’ appear so frequently.

Step 3: Make List of Top 2000 Words

```
    ...: ## kill additional stopwords; we could go crazy, but we’ll just kill these 2.
In [176]: wordDict['penn']=0
    ...: wordDict['anonymous']=0
    ...:
    ...:
    ...: ## put top 2000 words in a list
    ...: ## we won't bother doing IDF
    ...: count=0
    ...: list=[]
    ...: for s in wordDist:
    ...:  for (v,k) in s:
    ...:       if (count <2000):
    ...:            list.append(v)
    ...:            count+=1
```

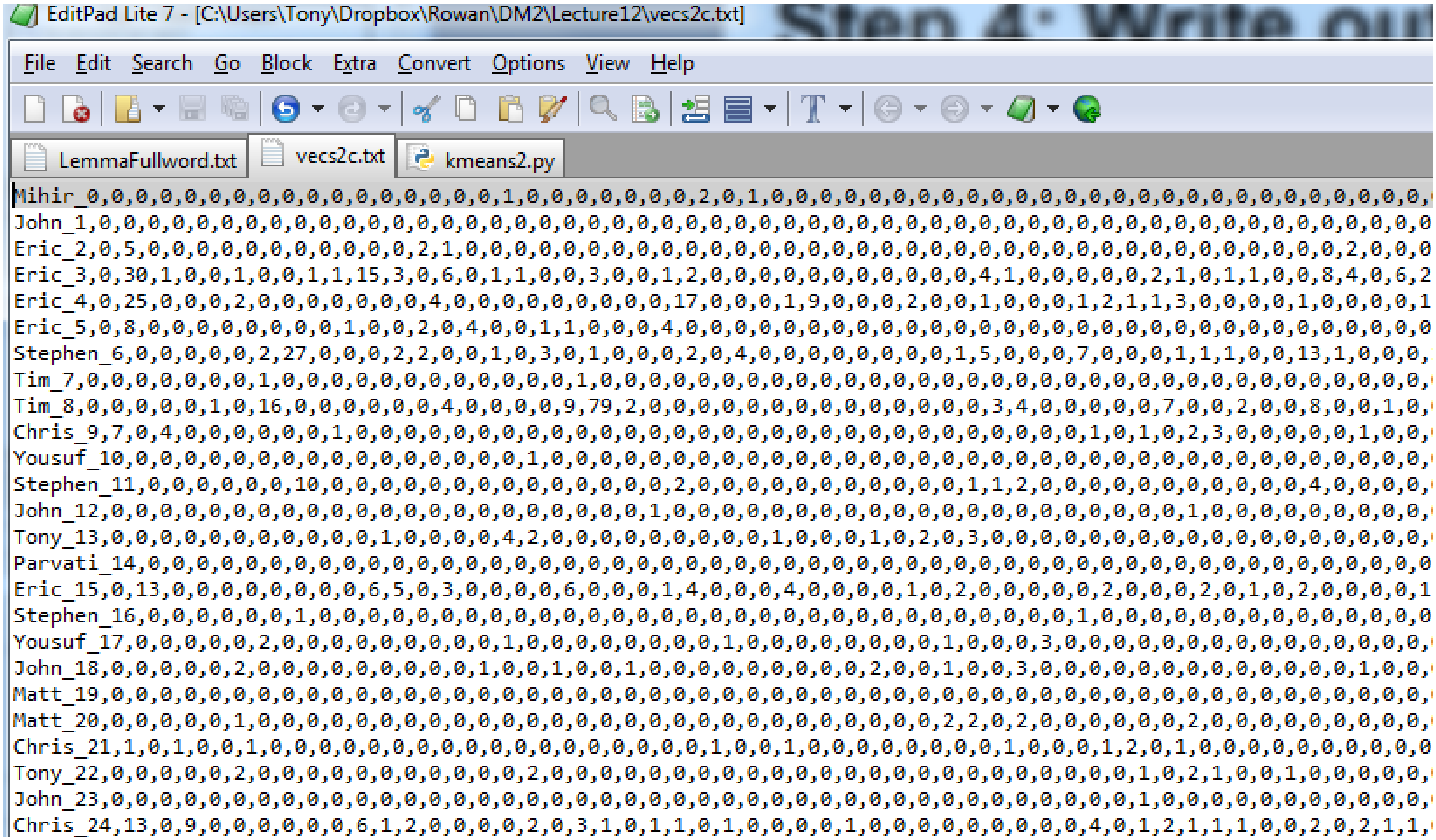
- 2000 is just arbitrary. I don’t know if it would work better with 100, 500, 1000, 10000.
- Note also, the first word and 2000th word have the same weight here. Don’t know if this will be a problem

Step 4: Write out each document as a sparse vector

```
In [177]: f5=open('vecs2c.txt','w')
     ...: papDict = {}
     ...: for i in range(len(dataset.data)):
     ...:   ## build a dictionary for each paper
     ...:   words = dataset.data[i].split()
     ...:   for word in words:
     ...:       if papDict.has_key(word):
     ...:            papDict[word]+=1
     ...:       else:
     ...:            papDict.setdefault(word,1)
     ...:   ## now build a vector containing frequencies
     ...:   vecc=[]
     ...:   for w in list:
     ...:       if papDict.has_key(w):
     ...:            vecc.append(papDict[w])
     ...:       else:
     ...:            vecc.append(0)
     ...:   ## print vector out
     ...:   f5.write(dataset.target_names[labels[i]]+"_"+str(i))
     ...:   for t in vecc:
     ...:       f5.write(","+str(t))
     ...:       f5.write("\n")
     ...:   papDict={}
     ...: f5.close()
```

- Use a dictionary for each document. Then check to see if any words from the document match the top 2000 words

Snapshot of output
(Total of 1292 vectors; each has 2000 elements)



Go to R

- Go to Seriation2000.pdf

Unrelated Seriation Demo in R

- This one has nothing to do with clustering.
- But it might be the coolest thing you’ve seen in a while
- Go to SeriationLouvre.pdf