Social Computing
제 5 강
TF-IDF

2014/4/4

Homework – etl로 제출

- 오늘 배운 것을 바탕으로 다음을 수행하세요.

1. Hierarchical Clustering
   - blogdata.txt를 이용 Hierarchical Clustering을 수행 dendrogram.json을 생성
   - dendrogram.html radial.html에서 로드한 결과를 각각 이미지를 저장/캡처해서 hcdendrogram.jpg, hcradial.jpg로 제출
   - 해당 코드를 blogdata.py로 저장해서 제출

2. RSS feed 수집 및 Hierarchical Clustering
   - 교재 p30-33을 보고 자신이 관심있는 영문 블로그의 rss를 50개 이상 수집 blogdata.txt와 같은 형식으로 rss.txt로 저장 제출
   - 해당 수집 코드를 rss.py 로 저장해서 제출
   - 생성한 rss.txt를 이용해서 Hierarchical Clustering을 수행 dendrogram.json을 생성
   - dendrogram.html radial.html에서 로드한 결과를 각각 이미지를 저장/캡처해서 rssdendrogram.jpg, rssradial.jpg로 제출

3. k-Means Clustering
   - blogdata.txt를 rotate해서 얻은 자료를 바탕으로 k=10으로 단어간의 k-Means Clustering을 수행
   - 'google' 과 'apple'이 속해있는 group의 단어들을 정리해서 제출

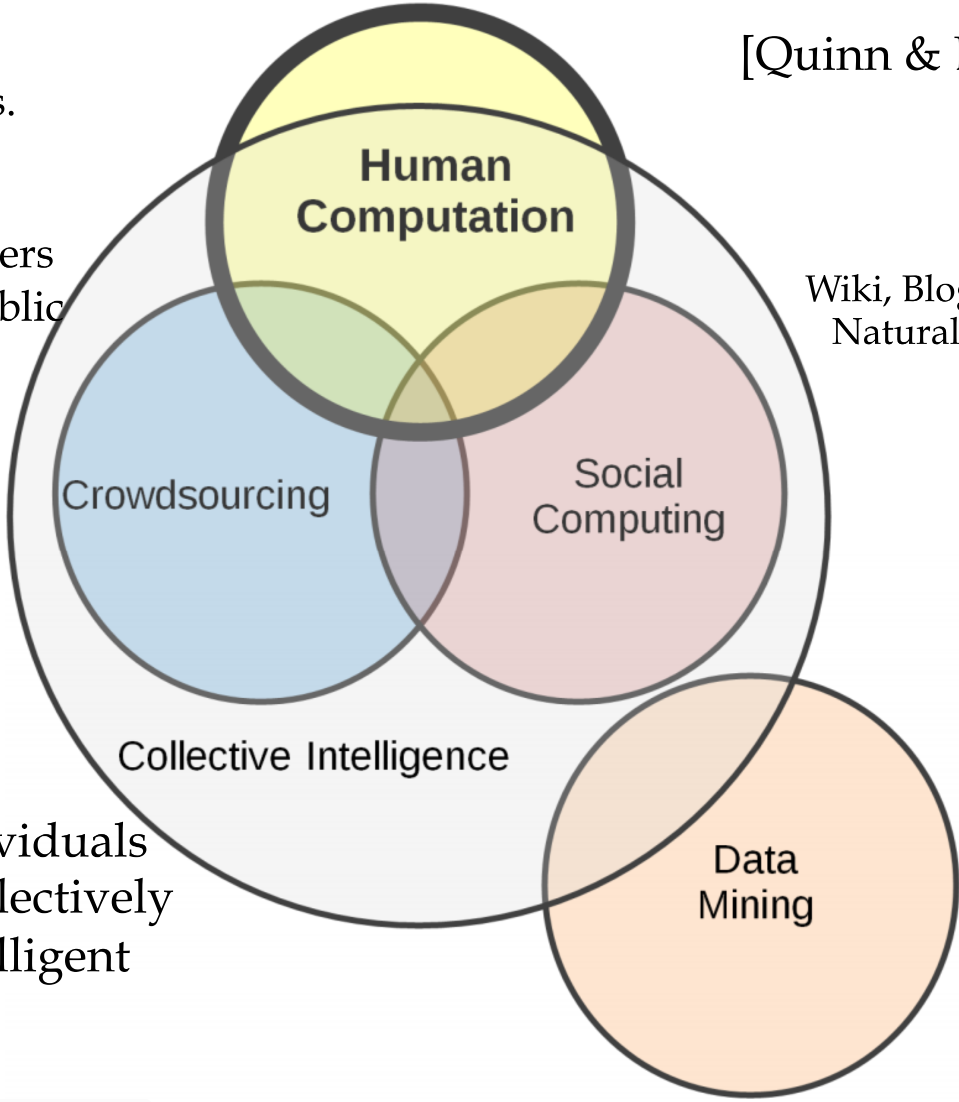
- 4월3일 자정까지 모든 파일을 zip으로 압축해서 etl 을 통해 제출

Human Computation Taxonomy

HC replaces
computers with humans.

CS replaces
traditional human workers
with members of the public

[Quinn & Bederson 2011]



Wiki, Blogs, Online communities
Natural vs. Directed Working

the application of
specific algorithms
for extracting
patterns from data

Groups of individuals
doing things collectively
that seem intelligent

Chapter 3 Discovering Groups

- Two types of machine learning
  - Use example inputs and outputs → **Supervised Learning**
    - Learn by examples
    - Classification
  - Not trained by examples → **Unsupervised Learning**
    - Find the structure within a data set
    - Clustering

- Clustering
  - 비슷한 것 끼리 모아서 그룹으로 만든다
  - 비슷한 것이란?
  - Jaccard Distance
  - Euclidean Distance
  - Pearson's Distance

Hierarchical Clustering

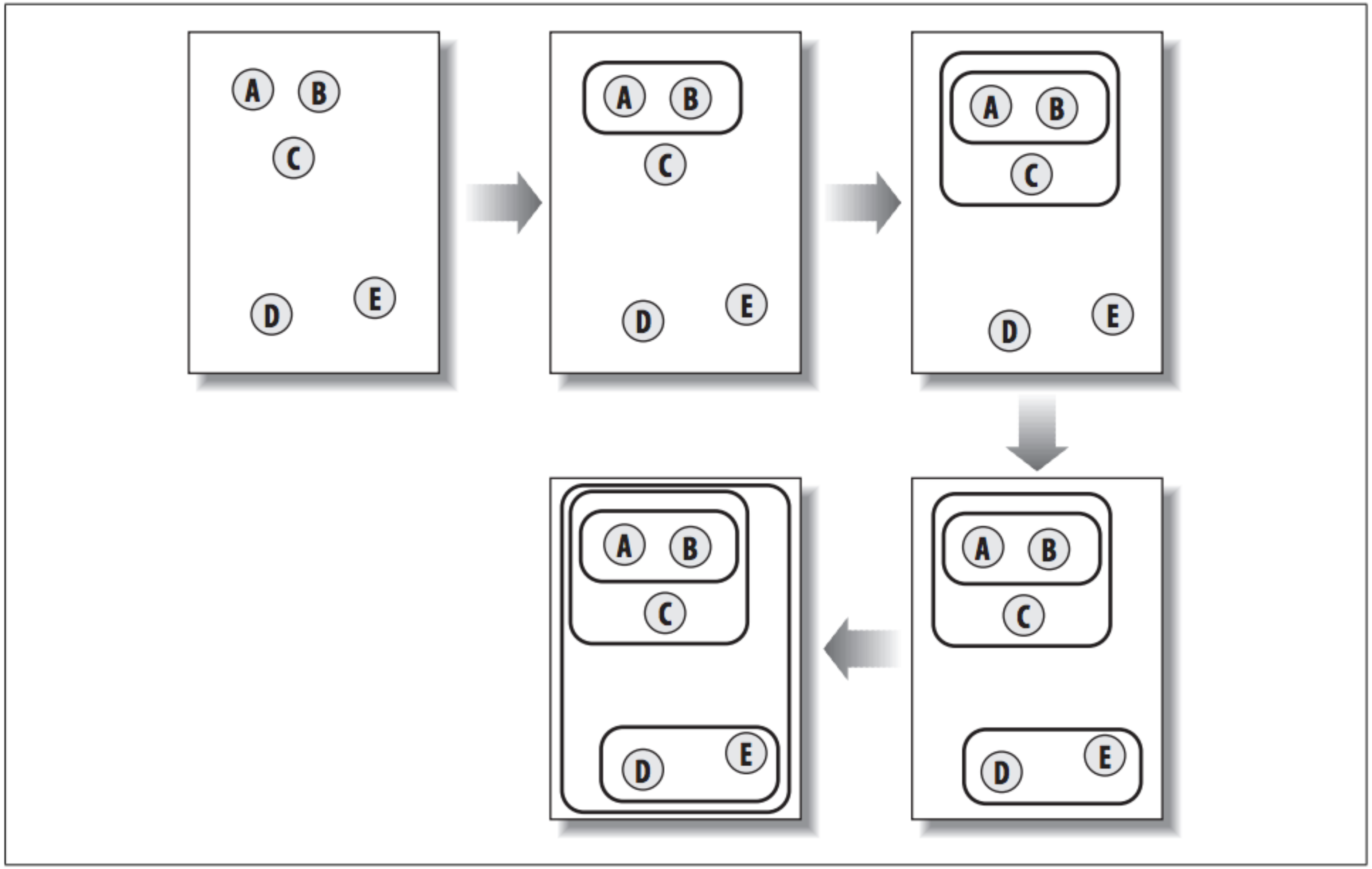
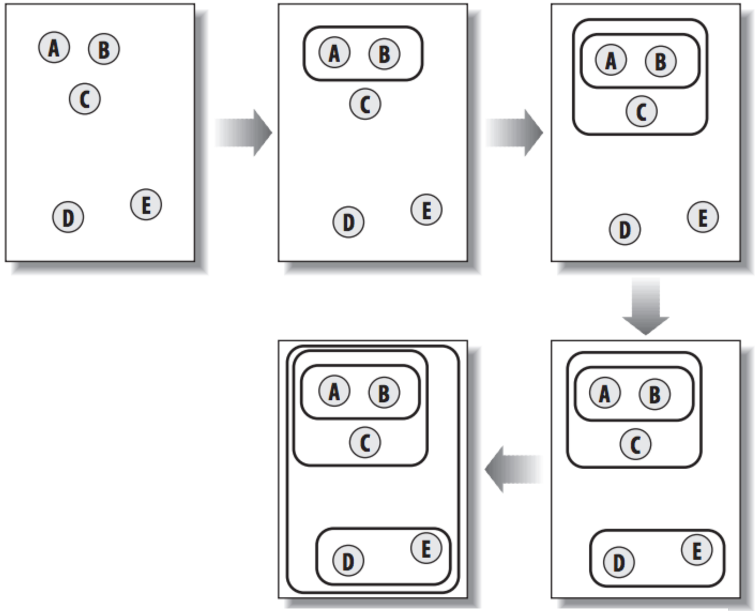


Figure 3-1. Hierarchical clustering in action

Dendrogram



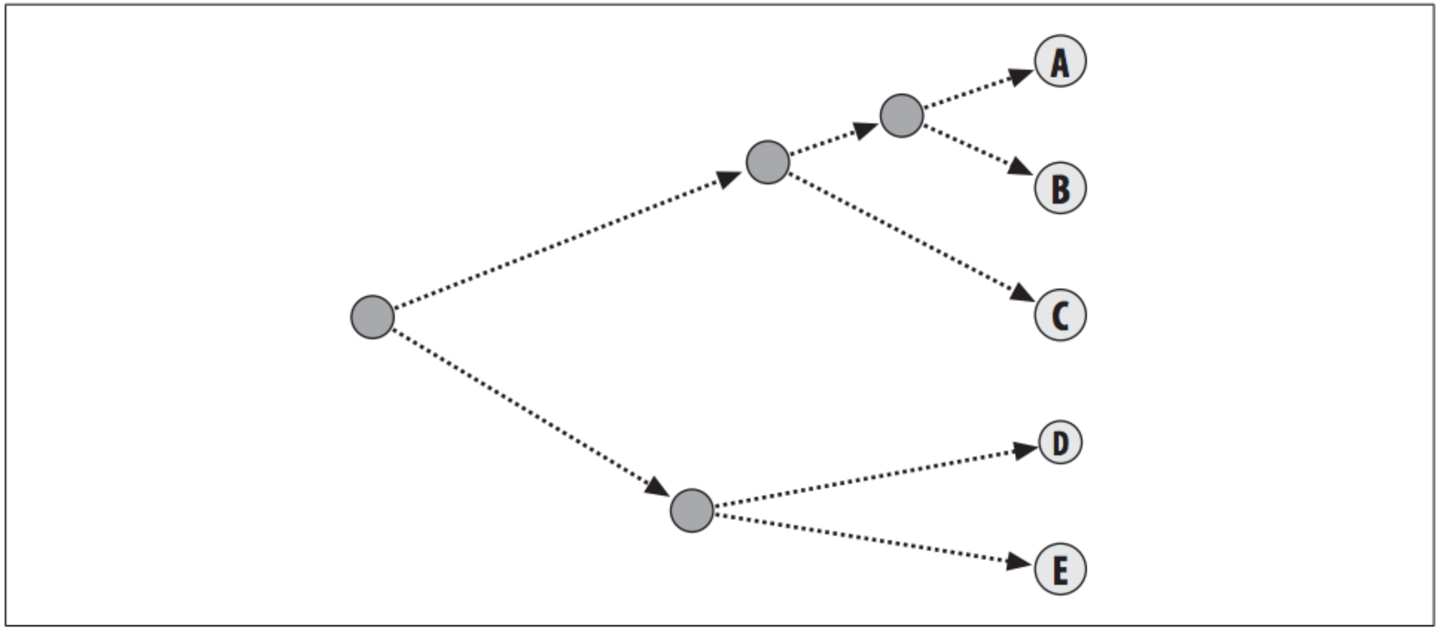


Figure 3-2. A dendrogram is a visualization of hierarchical clustering

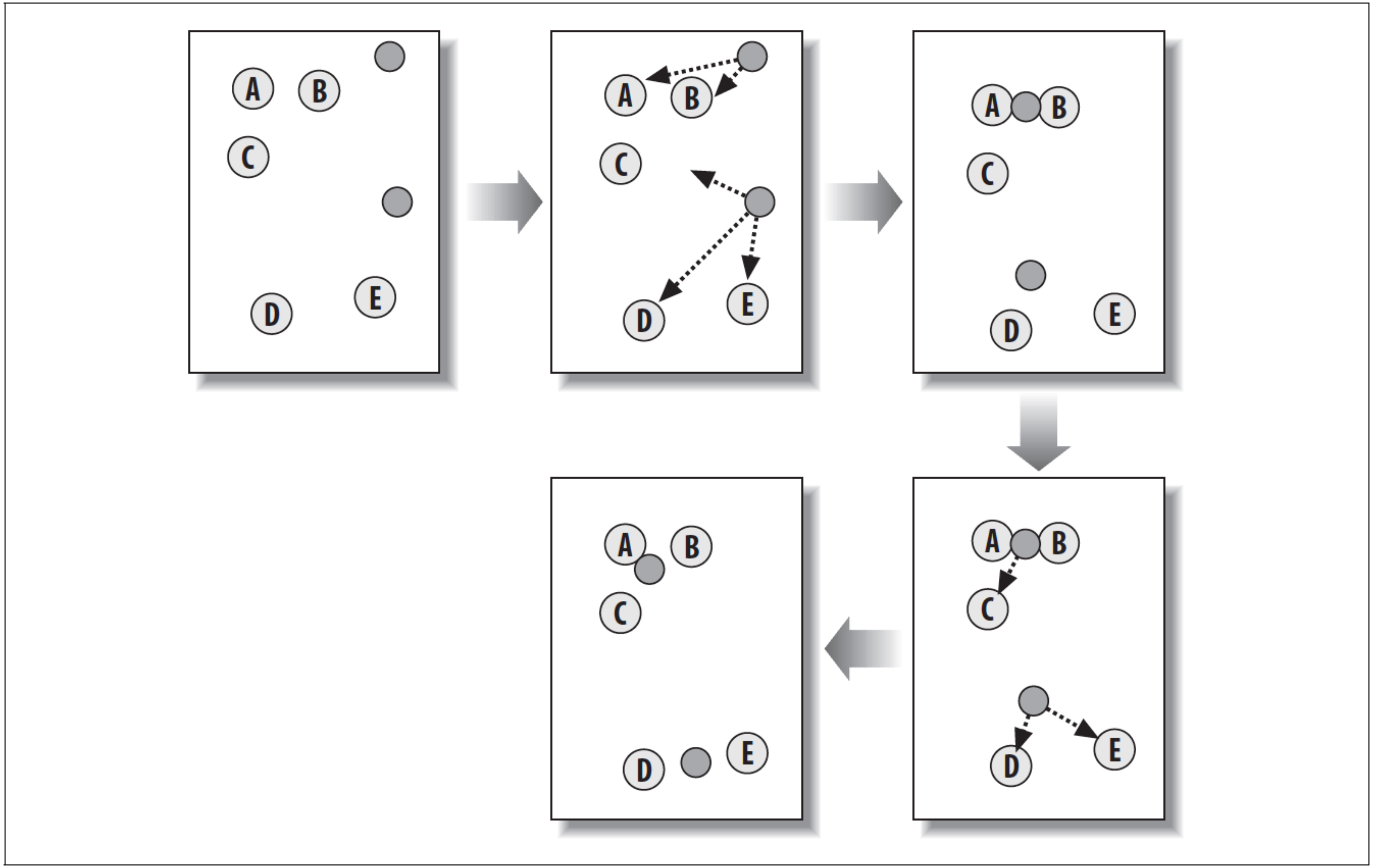
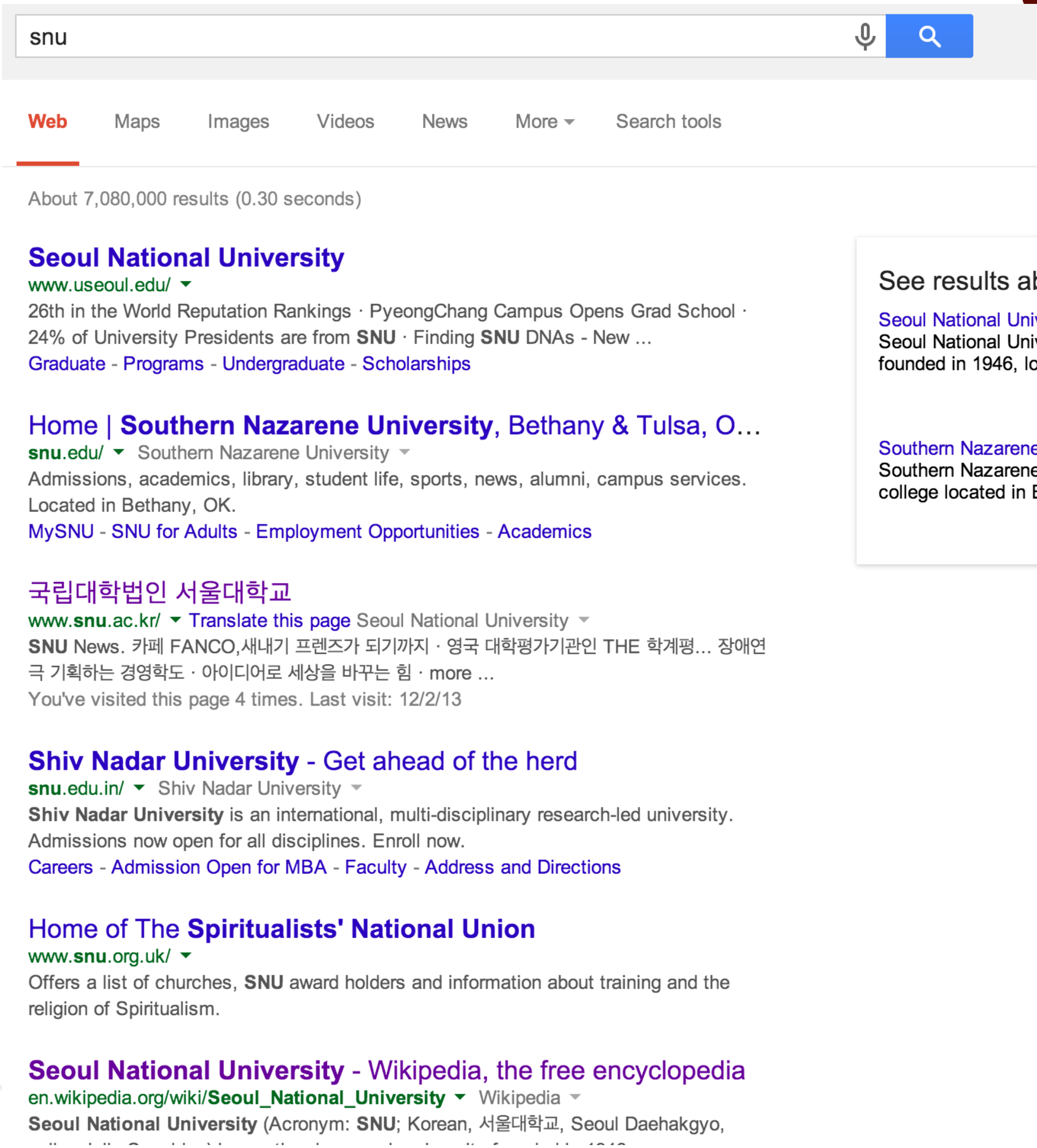


Figure 3-5. K-means clustering with two clusters

Information Retrieval: Searching

- Given a keyword, provide a list of items in the order of relevance
- 키워드를 주면 관계있는 자료를 연관된 순서대로 제공해준다

- 관계있는 자료란?



TF-IDF

- 자료에 포함되어 있는 단어의 중요성에 따라서 연관성을 계산하는 방법
- Term frequency – inverse document frequency
- A numerical ***statistic*** that is intended to reflect how important a word is to a document in a collection or ***corpus***.
- Can be used to ***query*** a ***corpus*** by calculating ***normalized scores*** that express the ***relative importance*** of ***terms*** in documents
- 각 어휘가 무슨 의미인지 잘 알아야 합니다.

Corpus and Terms

```
corpus = {
 'a' : "Mr. Green killed Colonel Mustard in the study with the candlestick. \
Mr. Green is not a very nice fellow.",
 'b' : "Professor Plum has a green plant in his study.",
 'c' : "Miss Scarlett watered Professor Plum's green plant while he was away \
from his office last week."
}

terms = {
 'a' : [ i.lower() for i in corpus['a'].split() ],
 'b' : [ i.lower() for i in corpus['b'].split() ],
 'c' : [ i.lower() for i in corpus['c'].split() ]
}
```

Term Frequency for "Mr. Green"

- Mr. Green killed Colonel Mustard in the study with the candlestick. Mr. Green is not a very nice fellow.
- Professor Plum has a green plant in his study.
- Miss Scarlett watered Professor Plum's green plant while he was away from his office last week.

Document	tf(mr.)	tF(green)	Sum
corpus['a']	2/19	2/19	4/19 (0.21)
corpus['b']	0/9	1/9	1/9 (0.11)
corpus['c']	0/16	1/16	1/16 (0.06)

Term Frequency for "the green plant"

- Mr. Green killed Colonel Mustard in the study with the candlestick. Mr. Green is not a very nice fellow.
- Professor Plum has a green plant in his study.
- Miss Scarlett watered Professor Plum's green plant while he was away from his office last week.

Document	tf(the)	tf(green)	tf(plant)	Sum
corpus['a']	2/19	2/19	0/19	4/19 (0.21)
corpus['b']	0/9	1/9	1/9	2/9 (0.22)
corpus['c']	0/16	1/16	1/16	2/16 (0.13)

뭐가 문제인가?

Inverse Document Frequency

- 모든 문서에 공통적으로 나오는 단어는 변별력이 없다.
- 많은 문서에 나오는 단어를 penalize해 줘야 한다.

- idf (term, D) = 1 + log (전체 document의 수 / term을 포함하고 있는 document의 수)

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

$$1 + |\{d \in D : t \in d\}|$$

Inverse Document Frequency

- Mr. Green killed Colonel Mustard in the study with the candlestick. Mr. Green is not a very nice fellow.
- Professor Plum has a green plant in his study.
- Miss Scarlett watered Professor Plum's green plant while he was away from his office last week.

idf(mr.)	idf(green)	idf (the)	idf(plant)
1+log(3/1)	1 + log(3/3)	1+log(3/1)	1+log(3/2)
2.0986	1.0	2.0986	1.4055

“green” vs. “mr. green” vs. “the green plant”

Document	tf(mr.)	tf(green)	tf(the)	tf(plant)
corpus[‘a’]	0.1053	0.1053	0.1053	0
corpus[‘b’]	0	0.1111	0	0.1111
corpus[‘c’]	0	0.0625	0	0.0625

idf(mr.)	idf(green)	idf (the)	idf(plant)
1+log(3/1)	1 + log(3/3)	1+log(3/1)	1+log(3/2)
2.0986	1.0	2.0986	1.4055

Document	tf-idf(mr.)	tf-idf(green)	tf-idf(the)	tf-idf(plant)
corpus[‘a’]	0.1053*2.0986 0.2209	0.1053*1.0 0.1053	0.1053*2.099 0.2209	0*1.4055 0
corpus[‘b’]	0*2.0986 0	0.1111*1.0 0.1111	0*2.099 0	0.1111*1.4055 0.1562
corpus[‘c’]	0*2.0986 0	0.0625*1.0 0.0625	0*2.099 0	0.0625*1.4055 0.0878

Summed TF-IDF for Sample Queries

Document	tf-idf(mr.)	tf-idf(green)	tf-idf(the)	tf-idf(plant)
corpus['a']	0.1053*2.0986 0.2209	0.1053*1.0 0.1053	0.1053*2.099 0.2209	0*1.4055 0
corpus['b']	0*2.0986 0	0.1111*1.0 0.1111	0*2.099 0	0.1111*1.4055 0.1562
corpus['c']	0*2.0986 0	0.0625*1.0 0.0625	0*2.099 0	0.0625*1.4055 0.0878

Query	corpus['a']	corpus['b']	corpus['c']
Green	0.1053	**0.1111**	0.0625
Mr. green	0.2209 + 0.1053 = **0.3262**	0 + 0.1111 = 0.1111	0 + 0.0625 = 0.0625
The green plant	0.2209 + 0.1053 + 0 = **0.3262**	0 + 0.1111 + 0.1562 = 0.2673	0 + 0.0625 + 0.0878 = 0.1503

Python Code for TF-IDF

```
from math import log

def tf(term, doc):
    doc = doc.lower().split()
    return doc.count(term.lower()) / float(len(doc))

def idf(term, corpus):
    num_texts_with_term = len([True for text in corpus if term.lower()
                         in text.lower().split()])
    try:
        return 1.0 + log(float(len(corpus)) / num_texts_with_term)
    except ZeroDivisionError:
        return 1.0

def tf_idf(term, doc, corpus):
    return tf(term, doc) * idf(term, corpus)
```

Python Code for TF-IDF (Cont.)

```
query_scores = {'a': 0, 'b': 0, 'c': 0}
QUERY_TERMS = ['mr.', 'green']

for term in [t.lower() for t in QUERY_TERMS]:
    for doc in sorted(corpus):
        print 'TF(%s): %s' % (doc, term), tf(term, corpus[doc])
    print 'IDF: %s' % (term, ), idf(term, corpus.values())
    print
    for doc in sorted(corpus):
        score = tf_idf(term, corpus[doc], corpus.values())
        print 'TF-IDF(%s): %s' % (doc, term), score
        query_scores[doc] += score
    print

print "Overall TF-IDF scores for query '%s'" % (' '.join(QUERY_TERMS), )
for (doc, score) in sorted(query_scores.items()):
    print doc, score
```

```
...         query_scores[doc] += score
...     print
...
TF(a): mr. 0.105263157895
TF(b): mr. 0.0
TF(c): mr. 0.0
IDF: mr. 2.09861228867

TF-IDF(a): mr. 0.220906556702
TF-IDF(b): mr. 0.0
TF-IDF(c): mr. 0.0

TF(a): green 0.105263157895
TF(b): green 0.111111111111
TF(c): green 0.0625
IDF: green 1.0

TF-IDF(a): green 0.105263157895
TF-IDF(b): green 0.111111111111
TF-IDF(c): green 0.0625

>>>
>>> print "Overall TF-IDF scores for query '%s'" % (' '.join(QUERY_TERMS), )
Overall TF-IDF scores for query 'mr. green'
>>> for (doc, score) in sorted(query_scores.items()):
...     print doc, score
...
a 0.326169714597
b 0.111111111111
c 0.0625
>>> 
```

NLTK Again

```
import nltk

# Download ancillary nltk packages if not already installed
nltk.download('stopwords')

all_content = " ".join( [corpus[index] for index in corpus] )

# Approximate bytes of text
print len(all_content)

tokens = all_content.split()
text = nltk.Text(tokens)

# Examples of the appearance of the word "open"
text.concordance("open")

# Frequent collocations in the text (usually meaningful phrases)
text.collocations()

# Frequency analysis for words of interest
fdist = text.vocab()
fdist["green"]
fdist["mr."]
fdist["the"]
```

NLTK Utilities

```
# Number of words in the text
len(tokens)

# Number of unique words in the text

len(fdist.keys())

# Common words that aren't stopwords
[w for w in fdist.keys()[:100] \
  if w.lower() not in nltk.corpus.stopwords.words('english')]

# Long words that aren't URLs
[w for w in fdist.keys() if len(w) > 15 and not w.startswith("http")]

# Number of URLs
len([w for w in fdist.keys() if w.startswith("http")])

# Enumerate the frequency distribution
for rank, word in enumerate(fdist):
   print rank, word, fdist[word]
```

Test Data Set

- UC Irvine Machine Learning Repository
- https://archive.ics.uci.edu/ml/datasets.html

- Reuter 50 50 Data Set
  - Top 50 authors & their top 50 articles
  - Test set and train set
  - Uploaded in etl

Load Reuter 50 50 Data Set

- Download C50.zip file from etl
- Unzip somewhere in your computer

```
import os
DATA_DIR = '/Users/bongwon/Downloads/C50/C50train'
#DATA_DIR = 'C:\\temp\\C50\\C50train'

reuter = []
for authorname in os.listdir(DATA_DIR):
    if authorname.startswith('.'):
        continue
    author_dir = os.path.join(DATA_DIR, authorname)
    for filename in os.listdir(author_dir):
        if not filename.endswith('.txt'):
            continue
        filepath = os.path.join(author_dir, filename)
        file = open(filepath, 'r')
        text = file.read()
        reuter += [{'author':authorname, 'filepath':filepath, 'filename':filename, 'text':text}]
```

Inspecting Reuter Data Set

```
import nltk

all_content = " ".join([doc['text'] for doc in reuter])

tokens = all_content.split()
text = nltk.Text(tokens)

# Frequent collocations in the text (usually meaningful phrases)
text.collocations()

# Frequency analysis for words of interest
fdist = text.vocab()
fdist["U.S."]
```

Using TextCollections to Compute TF-IDF

```
QUERY_TERMS = ['china']

activities = [article['text'].lower().split() for article in reuter]

# TextCollection provides tf, idf, and tf_idf abstractions so
# that we don't have to maintain/compute them ourselves
tc = nltk.TextCollection(activities)

relevant_activities = []

for idx in range(len(activities)):
    score = 0
    for term in [t.lower() for t in QUERY_TERMS]:
        score += tc.tf_idf(term, activities[idx])
    if score > 0:
        relevant_activities.append({'score': score, 'author': reuter[idx]['author'],
                    'filepath': reuter[idx]['filepath']})
```

Using TextCollections to Compute TF-IDF (Cont.)

```
# Sort by score and display results

relevant_activities = sorted(relevant_activities, key=lambda p:
p['score'], reverse=True)
for activity in relevant_activities[:10]:
    print activity['author']
    print '\tFile: %s' % (activity['filepath'], )
    print '\tScore: %s' % (activity['score'], )
    print
```

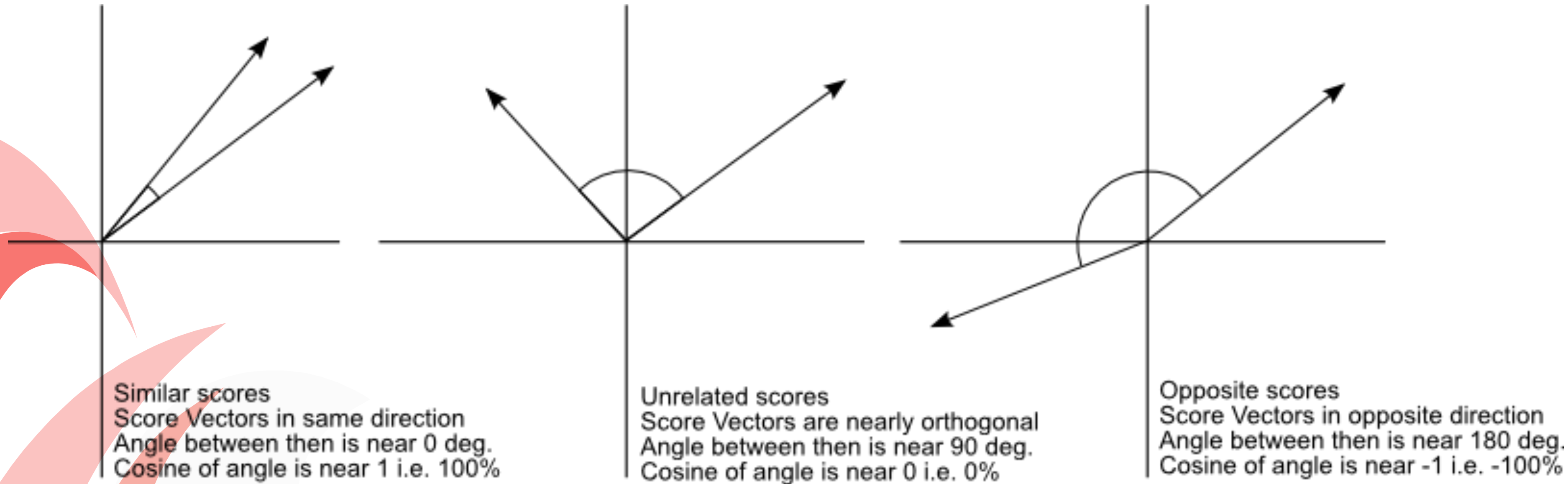
Cosine Similarity

- Jaccard distance
- Pearson's score (distance)
- Euclidean distance

$$\vec{a} = (0,3)$$
$$\vec{b} = (4,0)$$
$$\vec{a} \cdot \vec{b} = 0 * 4 + 3 * 0 = 0$$

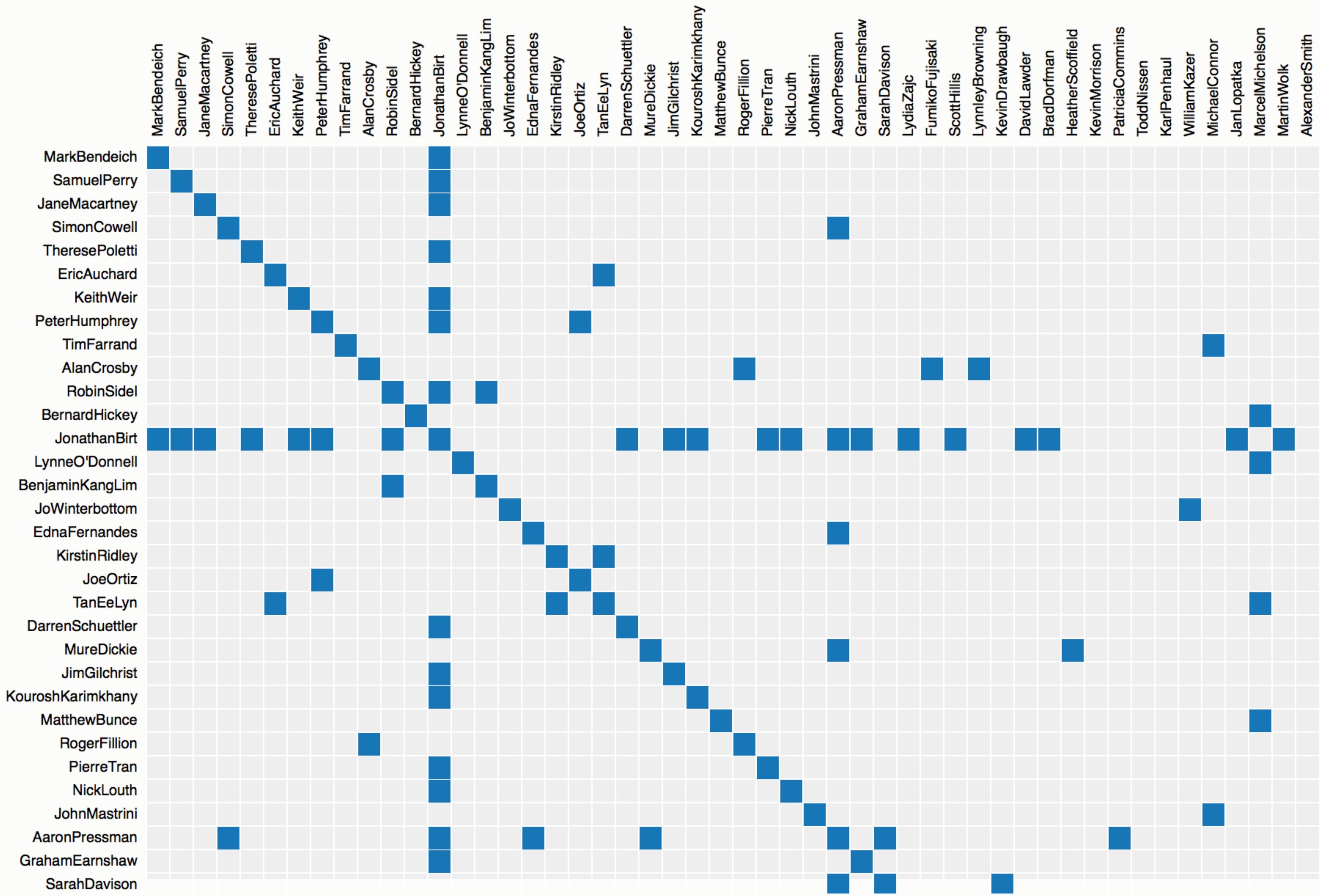
$$\vec{a} \cdot \vec{b} = \|\vec{a}\|\,\|\vec{b}\| \cos\theta$$
$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|\,\|\vec{b}\|}$$

Interpreting Cosine Similarity



Near 1 Near 0 Near -1

Matrix Diagram



Cosine Similarity Function

```
import nltk, math, os, json

def cosine_similarity(v1,v2):
    "compute cosine similarity of v1 to v2: (v1 dot v1)/{||v1||*||v2||)"
    sumxx, sumxy, sumyy = 0, 0, 0
    for i in range(len(v1)):
        x = v1[i]; y = v2[i]
        sumxx += x*x
        sumyy += y*y
        sumxy += x*y
    return sumxy/math.sqrt(sumxx*sumyy)
```

Document Similarity Model

```
def compute_similarity(_terms1, _terms2):
    # Take care not to mutate the original data structures
    # since we're in a loop and need the originals multiple times
    terms1 = _terms1.copy()
    terms2 = _terms2.copy()
    # Fill in "gaps" in each map so vectors of the same length can be computed
    for term1 in terms1:
        if term1 not in terms2:
            terms2[term1] = 0
    for term2 in terms2:
        if term2 not in terms1:
            terms1[term2] = 0
    # Create vectors from term maps
    v1 = [score for (term, score) in sorted(terms1.items())]
    v2 = [score for (term, score) in sorted(terms2.items())]
    # Compute similarity amongst documents
    return cosine_similarity(v1, v2)
```

Read the first article for each author

```
DATA_DIR = '/Users/bongwon/Downloads/C50/C50train'

reuter = []

for authorname in os.listdir(DATA_DIR):
    if authorname.startswith('.'):
        continue
    author_dir = os.path.join(DATA_DIR, authorname)
    for filename in os.listdir(author_dir):
        if not filename.endswith('.txt'):
            continue
        filepath = os.path.join(author_dir, filename)
        file = open(filepath, 'r')
        text = file.read()
        reuter += [{'author':authorname, 'filepath':filepath, 'text':text}]
        break
```

Preparing td-idf score for each term

```
all_articles = [article['text'].lower().split() for article in reuter]

tc = nltk.TextCollection(all_articles)

# Compute a term-document matrix such that td_matrix[doc_title][term]
# returns a tf-idf score for the term in the document

td_matrix = {}
for idx in range(len(all_articles)):
    article = all_articles[idx]
    fdist = nltk.FreqDist(article)
    doc_title = reuter[idx]['author']
    td_matrix[doc_title] = {}
    for term in fdist.iterkeys():
        td_matrix[doc_title][term] = tc.tf_idf(term, article)
```

Preparing Node & Link Structure

- viz_links = []
- viz_nodes = [{'title' : title} for title in td_matrix.keys()]

- idnum = 0
- for vn in viz_nodes:
-     vn.update({'idx' : idnum})
-     idnum += 1

- idx = dict(zip([node['title'] for node in viz_nodes], range(len(viz_nodes))))

- distances = {}

Find the farthest document for each one

```
for title1 in td_matrix.keys():
    distances[title1] = {}
    min_dist = 1.0
    most_similar = None
    for title2 in td_matrix.keys():
        if title1 == title2:
            continue
        # Compute similarity amongst documents
        terms1 = td_matrix[title1]
        terms2 = td_matrix[title2]
        distances[title1][title2] = compute_similarity(terms1, terms2)
        if distances[title1][title2] < min_dist:
            min_dist = distances[title1][title2]
            most_similar = title2
    viz_links.append({'source' : idx[title1], 'target' : idx[most_similar], 'score' : 1 - min_dist})
```

Save JSON file and Open it with Firefox

```
f = open('matrix.json', 'w')
f.write(json.dumps({'nodes' : viz_nodes, 'links' : viz_links}, indent=1))
f.close()
```

- 생성된 matrix.json 화일을 matrix.html과 같은 디렉토리에 넣고 Firefox에서 matrix.html을 로드한다.

TF-IDF의 문제점은?

- Bag of Words
  - Context를 사용하지 못한다
  - 문장의 의미가 없어짐
- Homonym
  - 동음이의어
- Link Structure가 있다면?
  - PageRank

Homework – etl로 제출

- 오늘 배운 것을 바탕으로 다음을 수행하세요.

1. TF-IDF
   - Reuter data set중 C50train 디렉토리에 있는 2500개의 기사들 중에 다음 query에 대하여 tf-idf 값이 높은 상위 10개의 기사를 찾아서 각각 리스트를 제출하세요
   - "Hong Kong", "technology", "government", "human rights"

2. Matrix Diagram
   - 지난주 과제로 모은 블로그의 글들을 대상으로 Matrix diagram을 작성하세요
   - 해당 수집 코드를 blog.py 로 저장해서 제출
   - 생성된 JSON화일을 matrix.json로 저장해서 제출
   - matrix..html에서 로드한 결과를 이미지로 저장/캡처해서 matrix.jpg로 제출

3. Reuter 50 50 Data Set
   - Reuter data set중 C50test 디렉토리에 있는 50명의 저자의 첫번째 글을 이용하여 다음을 구하세요.
   - 가장 cosine similarity가 가까운 작가 pair와 가장 먼 작가 pair를 구하세요.

- 4월10일 자정까지 모든 파일을 zip으로 압축해서 etl 을 통해 제출