Web Science:
Clustering Algorithms

(Part 1 - Intro to Clustering and Preparing the Data)

CS 432/532

Old Dominion University

Permission has been granted to use these slides from Frank McCown, Michael L. Nelson, Alexander Nwala, Michele C. Weigle

Main reference:

Ch 3 from *Programming Collective Intelligence* by Toby Segaran

(*abbreviated as PCI*)

Ch 3 GitHub repo

From Similarity to Clusters...

- Ch 2 - compute similarity between pairs of items
- Ch 3 - discover and group *(cluster)* things that are similar
- We want to cluster our information because we want to:
  - find unknown groups or patterns
  - visualize our results

Supervised vs. Unsupervised Learning

- Clustering is an example of *unsupervised* learning
  - we don't know what the correct answer is before we start
- Supervised learning is based on first being provided a set of inputs and known outputs
  - supervised learning examples in later chapters

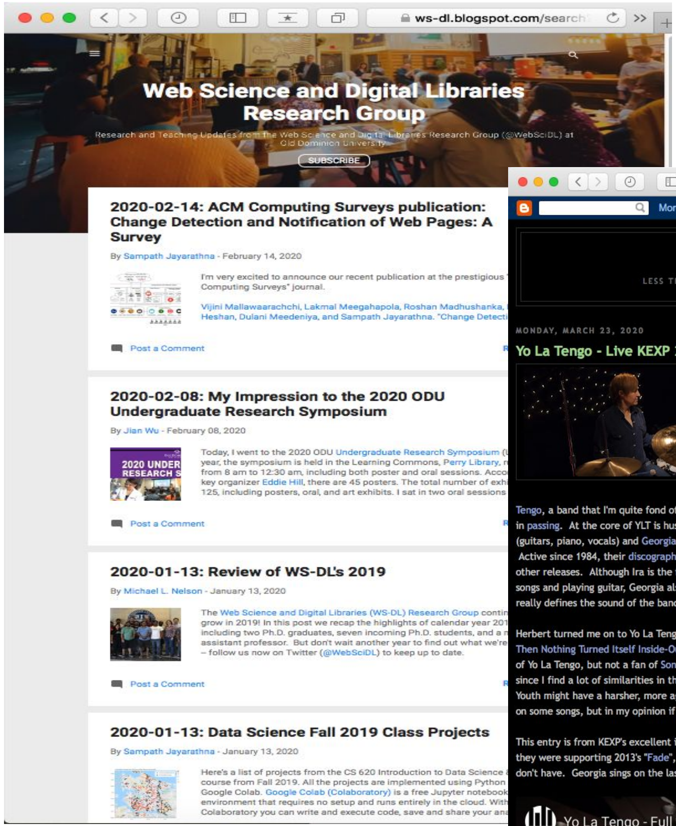
First Things First…

- Items to be clustered need numerical scores that "describe" the items
- Some examples:
  - customers can be described by the amount of purchases they make each month
  - movies can be described by the ratings given to them by critics
  - *documents (and webpages) can be described by the number of times they use certain words*

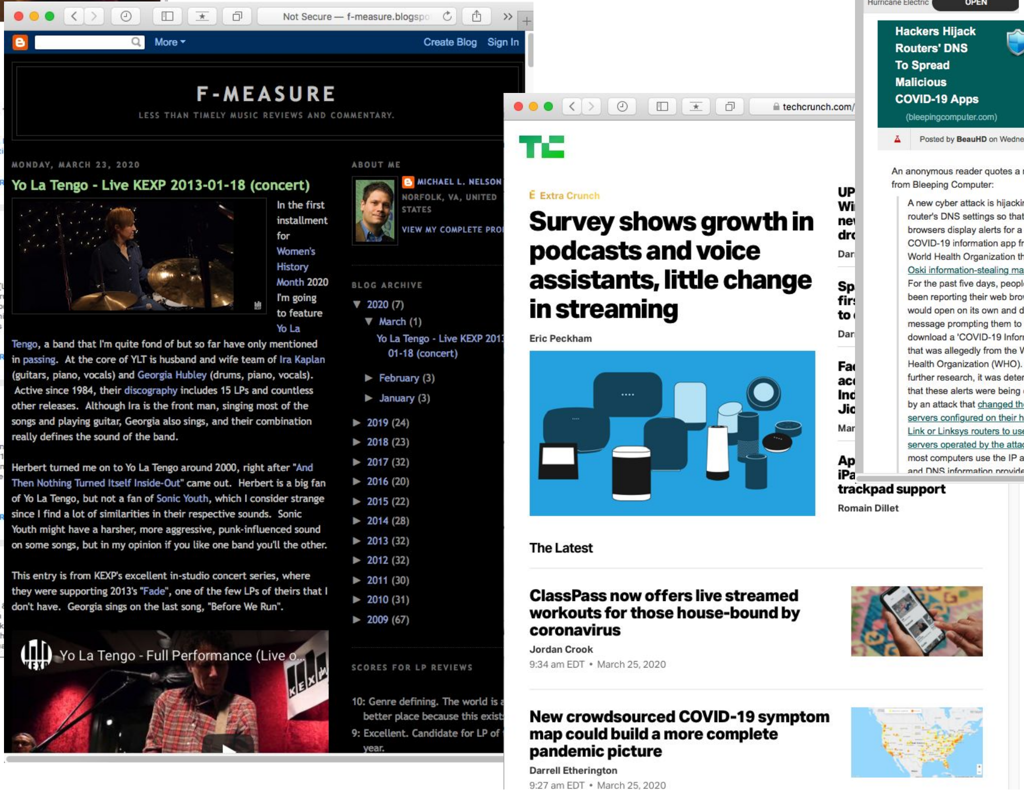
Finding Similar Webpages

- Given *N* webpages, how would we cluster them?
  - *documents (and webpages) can be described by the number of times they use certain words*
- Extract terms
  - break each string by whitespace
  - convert to lowercase
  - remove HTML tags (boilerplate removal)
- Find frequency of each term in each document
  - remove common terms (i.e., stop words, high TF) and very unique terms (i.e., high IDF)

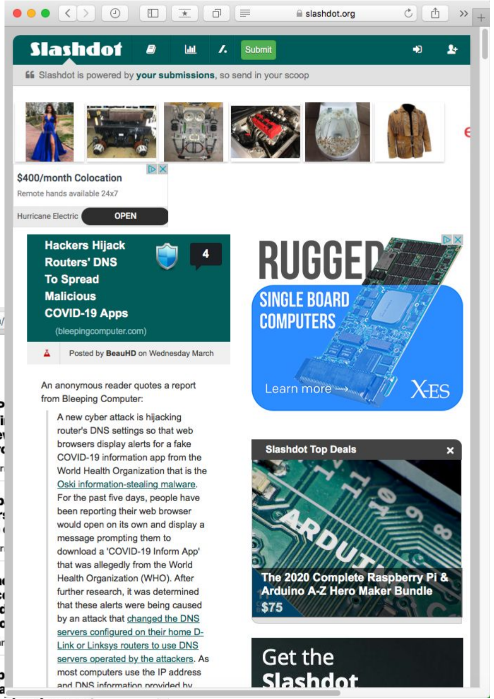
Blogs



ws-dl.blogspot.com/search
Web Science and Digital Libraries Research Group
SUBSCRIBE
2020-02-14: ACM Computing Surveys publication: Change Detection and Notification of Web Pages: A Survey
By Sampath Jayarathna - February 14, 2020
Post a Comment
2020-02-08: My Impression to the 2020 ODU Undergraduate Research Symposium
By Jian Wu - February 08, 2020
Post a Comment
2020-01-13: Review of WS-DL's 2019
By Michael L. Nelson - January 13, 2020
Post a Comment
2020-01-13: Data Science Fall 2019 Class Projects
By Sampath Jayarathna - January 13, 2020



Not Secure — f-measure.blogspo
More
Create Blog
Sign In
F-MEASURE
LESS THAN TIMELY MUSIC REVIEWS AND COMMENTARY.
MONDAY, MARCH 23, 2020
Yo La Tengo - Live KEXP 2013-01-18 (concert)
ABOUT ME
MICHAEL L. NELSON
NORFOLK, VA, UNITED STATES
BLOG ARCHIVE
2020 (7)
March (1)
February (3)
January (3)
2019 (24)
2018 (23)
2017 (32)
2016 (20)
2015 (22)
2014 (28)
2013 (32)
2012 (32)
2011 (30)
2010 (31)
2009 (67)
SCORES FOR LP REVIEWS
techcrunch.com/
Extra Crunch
Survey shows growth in podcasts and voice assistants, little change in streaming
Eric Peckham
The Latest
ClassPass now offers live streamed workouts for those house-bound by coronavirus
Jordan Crook
9:34 am EDT • March 25, 2020
New crowdsourced COVID-19 symptom map could build a more complete pandemic picture
Darrell Etherington
9:27 am EDT • March 25, 2020



slashdot.org
Slashdot
Submit
Slashdot is powered by your submissions, so send in your scoop
$400/month Colocation
Hackers Hijack Routers' DNS To Spread Malicious COVID-19 Apps
(bleepingcomputer.com)
RUGGED
SINGLE BOARD COMPUTERS
Learn more
Slashdot Top Deals
The 2020 Complete Raspberry Pi & Arduino A-Z Hero Maker Bundle
$75
Get the Slashdot

Grabbing Blog Feeds

- Humans read blogs, newsfeeds, etc. in HTML, but machines read blogs in the XML-based syndication formats RSS or Atom
  - RSS (Wikipedia)
  - Atom (Web standard) (Wikipedia)
- These feeds contain a number of blog posts and include the blog post title and often the full text of each blog post

Creating a Blog-Term Matrix

	"china"	"kids"	"music"	"yahoo"
Gothamist	0	3	3	0
GigaOM	6	0	0	2
Quick Online Tips	0	2	2	22

Table 3-1 from *PCI* - Subset of blog word frequencies

Code for Creating a Blog-Term Matrix

- Code on pp. 31-32 for grabbing feeds, getting terms from title and summary (RSS) or description (Atom)

- Sample feed URIs in chapter3/feedlist.txt
- Sample blog-term matrix in chapter3/blogdata.txt

[Ch 3 GitHub repo]

Beginning of `generatefeedvector.py`

```
# Returns title and dictionary of
# word counts for an RSS feed

def getwordcounts(url):
  # Parse the feed
  d=feedparser.parse(url)
  wc={}

  # Loop over all the entries
  # description==RSS; summary==Atom
  for e in d.entries:
    if 'summary' in e: summary=e.summary
    else: summary=e.description

    # Extract a list of words
    words=getwords(e.title+' '+summary)
    for word in words:
      wc.setdefault(word,0)
      wc[word]+=1

  return d.feed.title,wc
```

PCI Code "Fakes" Stopwords, TF-IDF

[Stop word](#) (Wikipedia)

- *Stopwords* - most common words in a language
  - "the", "and", "in", …
- Can be approximated with words that have high TF

$$\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

- Very unique words - may only appear in a single blog
- Can be approximated with words that have high IDF
  - low IDF means that it appears in many docs

$$\mathrm{idf_i} = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

- Intuition
  - term that appears in many documents will get a low IDF; that term is not what the document is "about"
  - term that appears in just one or a few documents is a good discriminator and will get a high IDF; if appears in that document a lot, it will also get a high TF and the resulting high TF-IDF means that term captures the "aboutness" of the document

PCI Code "Fakes" Stopwords, TF-IDF

```
wordlist=[]
for w,bc in apcount.items():
  frac=float(bc)/len(feedlist)
  if frac>0.1 and frac<0.5:
    wordlist.append(w)
```

- `apcount` is an array that indicates how many blogs each word appears in
- `w` is the word
- `bc` is the blog count
- `frac` is the percentage of blogs that the word appears in
- `wordlist` is the final list of words

Creating a Blog-Term Matrix

	"china"	"kids"	"music"	"yahoo"
Gothamist	0	3	3	0
GigaOM	6	0	0	2
Quick Online Tips	0	2	2	22

Table 3-1 from *PCI* - Subset of blog word frequencies

blogdata.txt

Web Science: Clustering Algorithms

(Part 2 - Hierarchical Clustering and Dendrogram)

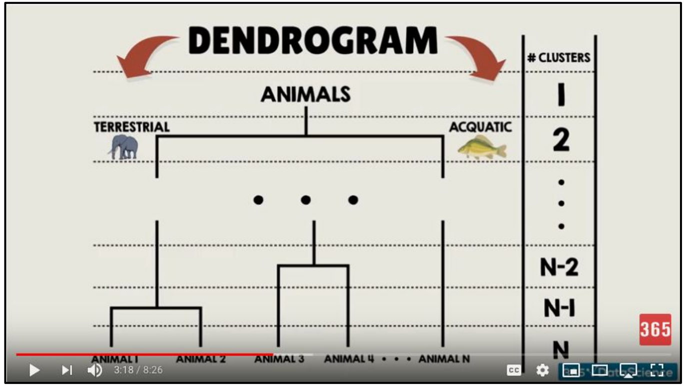
CS 432/532

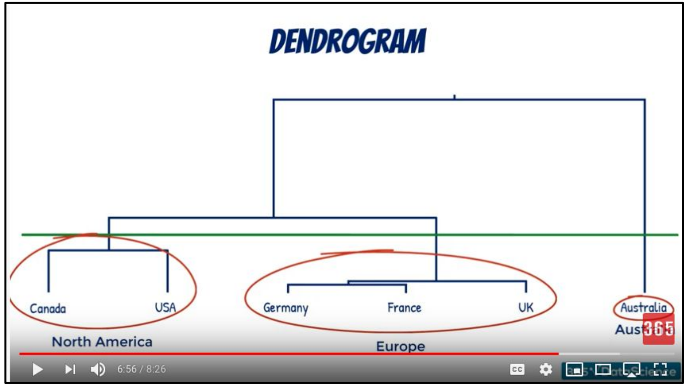
Old Dominion University

Permission has been granted to use these slides from Michele C. Weigle

Hierarchical Clustering

These slides assume you have already watched the assigned 365 Data Science video on ["Flat and Hierarchical Clustering | The Dendrogram Explained"]()





Main reference:

Ch 3 from *Programming Collective Intelligence* by Toby Segaran

(*abbreviated as PCI*)

Ch 3 GitHub repo

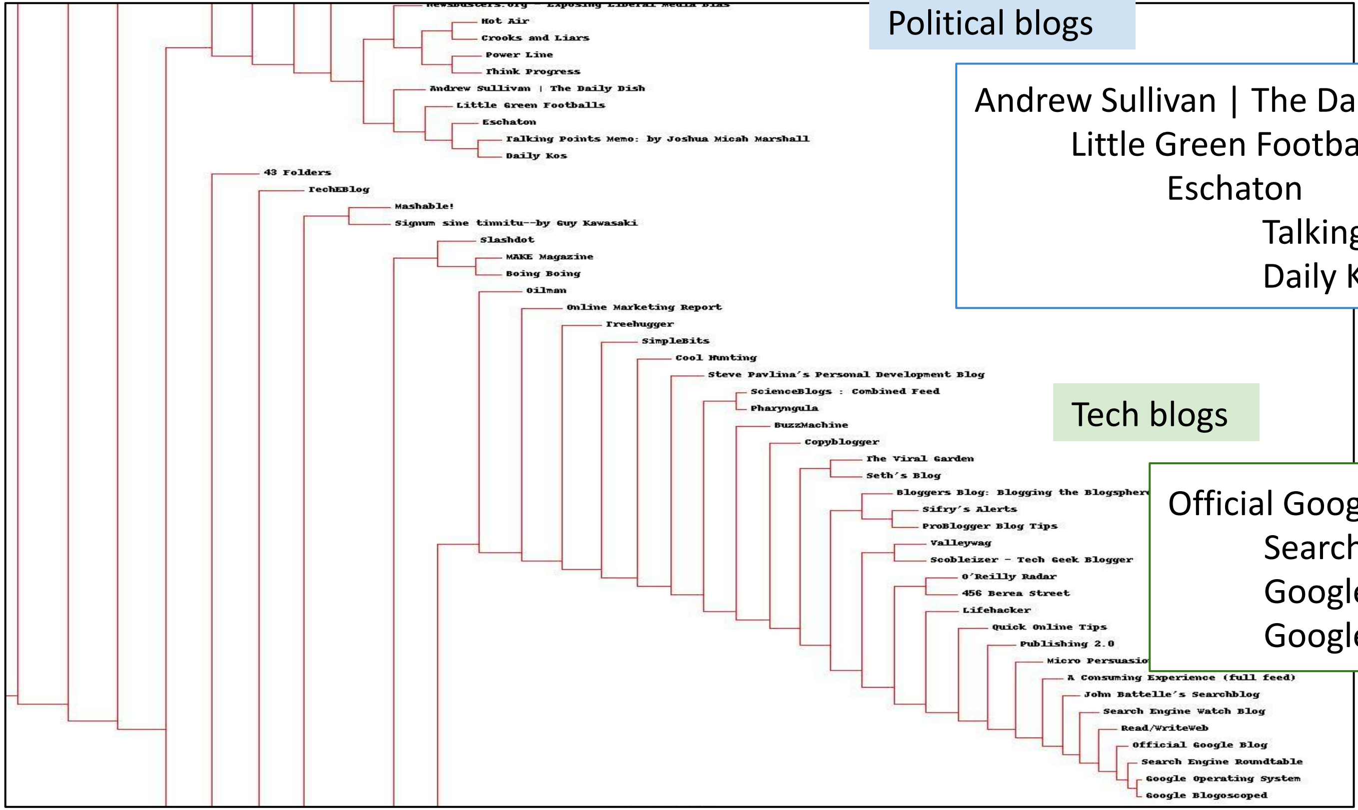
Remember the Blog-Term Matrix

	"china"	"kids"	"music"	"yahoo"
Gothamist	0	3	3	0
GigaOM	6	0	0	2
Quick Online Tips	0	2	2	22

Table 3-1 from *PCI* - Subset of blog word frequencies

blogdata.txt

Working Towards a Dendrogram



Political blogs

Andrew Sullivan | The Daily Dish
Little Green Footballs
Eschaton
Talking Points Memo
Daily Kos

Tech blogs

Official Google Blog
Search Engine Roundtable
Google Operating System
Google Blogoscoped

General Functions

- required imports
- `readfile(filename)` - returns arrays of rownames, colnames, data
- `pearson(v1, v2)` - returns Pearson correlation between two vectors of numbers
- `rotatematrix(data)` - returns rotated matrix (rows become columns and columns become rows)

Original code: [Ch 3 GitHub repo]

Hierarchical Clustering

- class `bicluster` - data structure to hold the clustering information
- `hcluster(rows, distance=pearson)` - does the hierarchical clustering, default distance function is `pearson()`
- `printclust(clust, labels=None, n=0)` - traverses the cluster and prints an ASCII text representation

Original code: [Ch 3 GitHub repo]

Example 1 (pg. 37)

- Generate hierarchical cluster, print ASCII clusters

```
blognames, words, data = readfile("blogdata.txt")

clust = hcluster(data)

printclust(clust, labels=blognames)
```

```
    -
     Lifehacker
      -
       Quick Online Tips
        -
         Publishing 2.0
          -
           Micro Persuasion
            -
             A Consuming Experience (full feed)
              -
               John Battelle's Searchblog
                -
                 Search Engine Watch Blog
                  -
                   Read/WriteWeb
                    -
                     Official Google Blog
                      -
                       Search Engine Roundtable
                        -
                         Google Operating System
                         Google Blogoscoped
```

Original code: [Ch 3 GitHub repo]

Drawing the Dendrogram

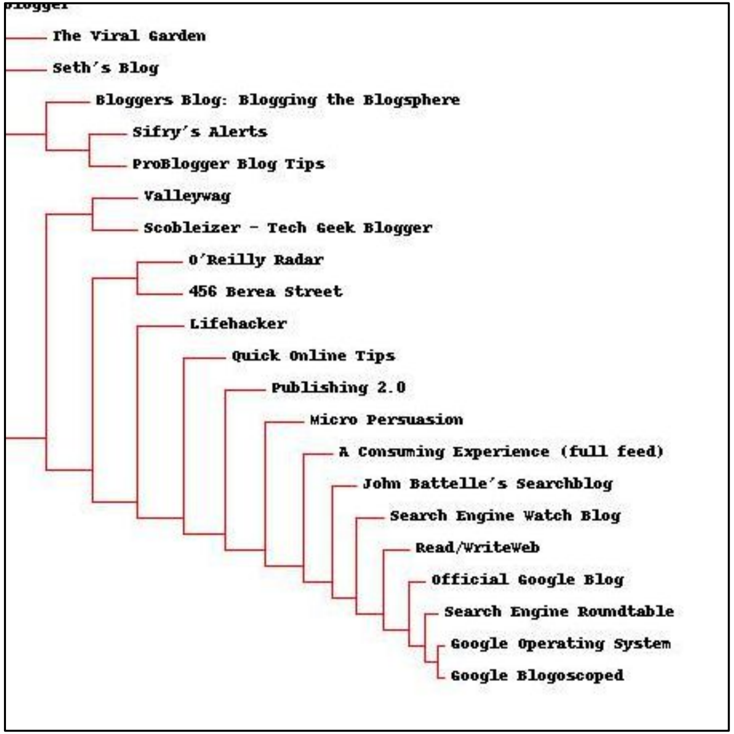
- `getheight(clust)` - returns the total height of a given cluster
- `getdepth(clust)` - returns the error depth of the cluster, the maximum possible error from each branch
- `drawdendrogram(clust, labels, jpeg='clusters.jpg')` - creates a dendrogram image with the default name of clusters.jpg
- `drawnode(draw, clust, x, y, scaling, labels)` - takes and cluster and its location, calculates where the child nodes should be, and draws lines to them

Original code: [Ch 3 GitHub repo]

Example 2 (pg. 40)

- Draw dendrogram

```
drawdendrogram(clust, blognames,
               jpeg='blogclust.jpg')
```



Original code: [Ch 3 GitHub repo]

Generate a Term-Blog Matrix

Rotate the matrix

	"china"	"kids"	"music"	"yahoo"
Gothamist	0	3	3	0
GigaOM	6	0	0	2
Quick Online Tips	0	2	2	22

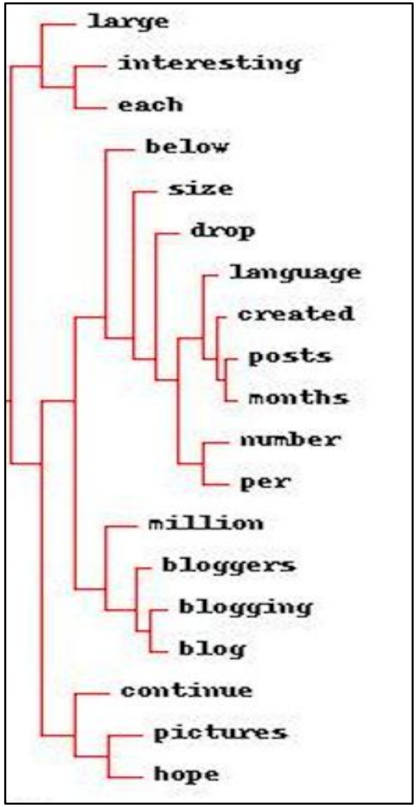
	Gothamist	GigaOM	Quick Online Tips
"china"	0	6	0
"kids"	3	0	2
"music"	3	0	2
"yahoo"	0	2	22

Original code: [Ch 3 GitHub repo]

Example 3 (pg. 40)

- Rotate the matrix, cluster, draw dendrogram of words

```
rdata = rotatematrix(data)
wordclust = hcluster(rdata)
drawdendrogram(wordclust, labels=words,
               jpeg='wordclust.jpg')
```



Original code: [Ch 3 GitHub repo]

Python Libraries for Dendrograms

- Plotly - Dendrograms | Python
- SciPy - scipy.cluster.hierarchy.dendrogram
- SciPy dendrogram tutorial/examples
  - #400 Basic Dendrogram
  - #401 Customised dendrogram
- SciPy and sklearn - Plot Hierarchical Clustering Dendrogram

Web Science: Clustering Algorithms

(Part 3 - k-Means Clustering)

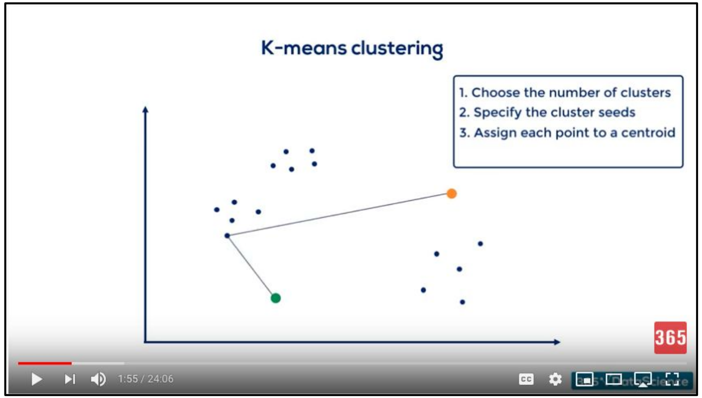
CS 432/532

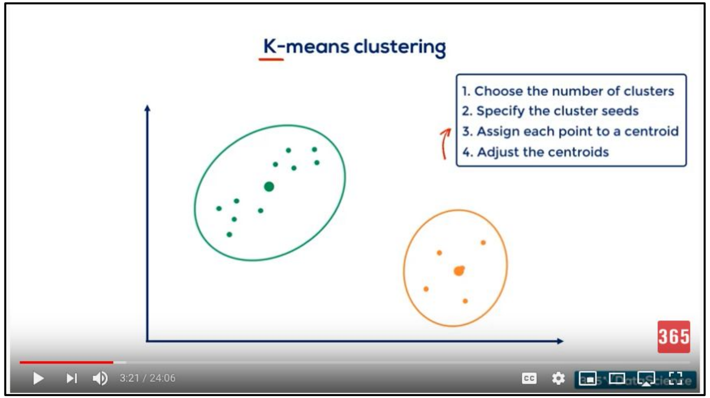
Old Dominion University

Permission has been granted to use these slides from Michele C. Weigle

k-Means Clustering

These slides assume you have already watched up to time 4:44 of the assigned 365 Data Science video on ["K Means Clustering: Pros and Cons of K Means Clustering"]()





Main reference:

Ch 3 from *Programming Collective Intelligence* by Toby Segaran

(*abbreviated as PCI*)

Ch 3 GitHub repo

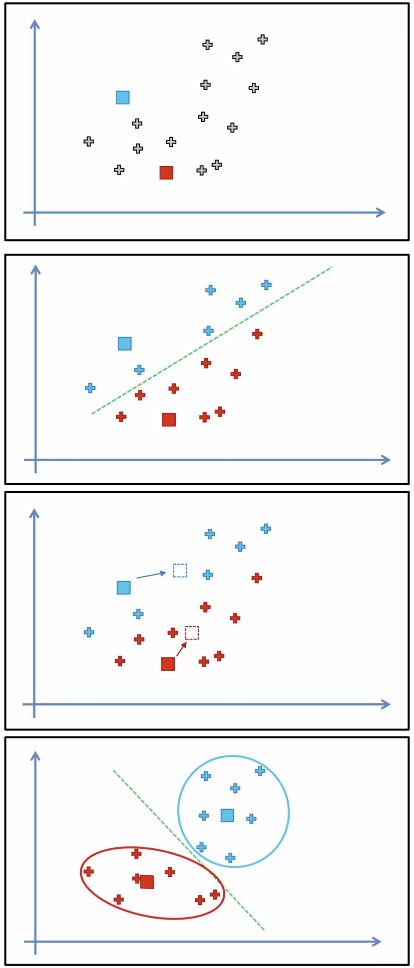
k-Means Review

1. Select *k* (e.g. 2) random points as cluster centers called *centroids*

2. Assign each data point to the closest cluster by calculating its distance with respect to each centroid

3. Determine the new cluster center by computing the average (*mean*) of the assigned points

4. Repeat steps 2 and 3 until none of the cluster assignments change



1

2

3

4

[K-means Clustering Python Example]

Remember the Blog-Term Matrix

	"china"	"kids"	"music"	"yahoo"
Gothamist	0	3	3	0
GigaOM	6	0	0	2
Quick Online Tips	0	2	2	22

Table 3-1 from *PCI* - Subset of blog word frequencies

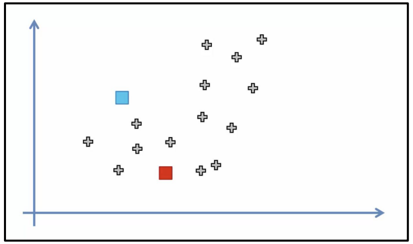
[blogdata.txt]

k-Means Clustering

- `kcluster(rows, distance=pearson, k=4)` - does the k-means clustering algorithm, default distance function is `pearson()`, default *k* is 4

Original code: [Ch 3 GitHub repo]

kcluster() - Step 1



1

```
# Create k randomly placed centroids
  clusters = [[random.random() * (ranges[i][1] - ranges[i][0]) + ranges[i][0]
              for i in range(len(rows[0]))] for j in range(k)]
```

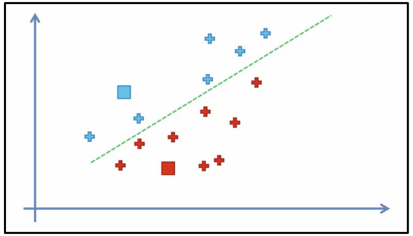
`i` - 0-700 (number of columns in the first row of data, or number of words)

`j` - 0-3 (assume k=4)

`ranges[]` - minimum and maximum values for each row

Original code: Ch 3 GitHub repo

kcluster() - Step 2



2

```python
# Find which centroid is the closest for each row
    for j in range(len(rows)):
        row = rows[j]
        bestmatch = 0
        for i in range(k):
            d = distance(clusters[i], row)
            if d < distance(clusters[bestmatch], row):
                bestmatch = i
        bestmatches[bestmatch].append(j)
```

`j` - current row

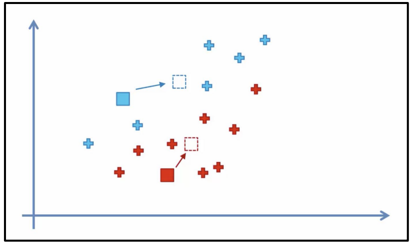
`i` - current cluster

`distance(clusters[bestmatch], row)` - distance between centroid and row

`bestmatches[i]` - list of row ids in cluster `i`

Original code: [Ch 3 GitHub repo]

kcluster() - Step 3



```
# Move the centroids to the average of their members
    for i in range(k):
        avgs = [0.0] * len(rows[0])
        if len(bestmatches[i]) > 0:
            for rowid in bestmatches[i]:
                for m in range(len(rows[rowid])):
                    avgs[m] += rows[rowid][m]
            for j in range(len(avgs)):
                avgs[j] /= len(bestmatches[i])
            clusters[i] = avgs
```

`j` - current row

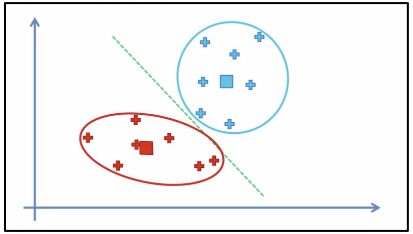
`i` - current cluster

avgs - average of each column over all rows in the cluster

`bestmatches[i]` - list of row ids in cluster `i`

Original code: [Ch 3 GitHub repo]()

kcluster() - Step 4




```
for t in range(100):
        print ('Iteration %d' % t)
        bestmatches = [[] for i in range(k)]

     [...]

# If the results are the same as last time, this is complete
        if bestmatches == lastmatches:
            break
        lastmatches = bestmatches
```

Original code: [Ch 3 GitHub repo]()

Example (pg. 44)

- Run k-means with 10 clusters on blogdata

```
kclust = kcluster(data, k=10)
```

```
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
```

Note that because we start with randomly-placed centroids, we may get different clusters each time we run the algorithm.

Original code: [Ch 3 GitHub repo]

sklearn Python Library for k-Means

- Reference
  - sklearn.cluster.KMeans
- Examples
  - In Depth: k-Means Clustering | Python Data Science Handbook
  - K-Means Clustering

Web Science: Clustering Algorithms

(Part 4 - Multidimensional Scaling)

CS 432/532

Old Dominion University

Main reference:

Ch 3 from *Programming Collective Intelligence* by Toby Segaran

(abbreviated as PCI)

Ch 3 GitHub repo

Remember the Blog-Term Matrix

	"china"	"kids"	"music"	"yahoo"
Gothamist	0	3	3	0
GigaOM	6	0	0	2
Quick Online Tips	0	2	2	22

Table 3-1 from *PCI* - Subset of blog word frequencies

blogdata.txt

Multidimensional Scaling (MDS)

- Allows us to visualize *N*-dimensional data in 2 or 3 dimensions
- Not a perfect representation of data, but allows us to visualize it without our heads exploding

- Start with item-item distance matrix (note: 1-similarity)

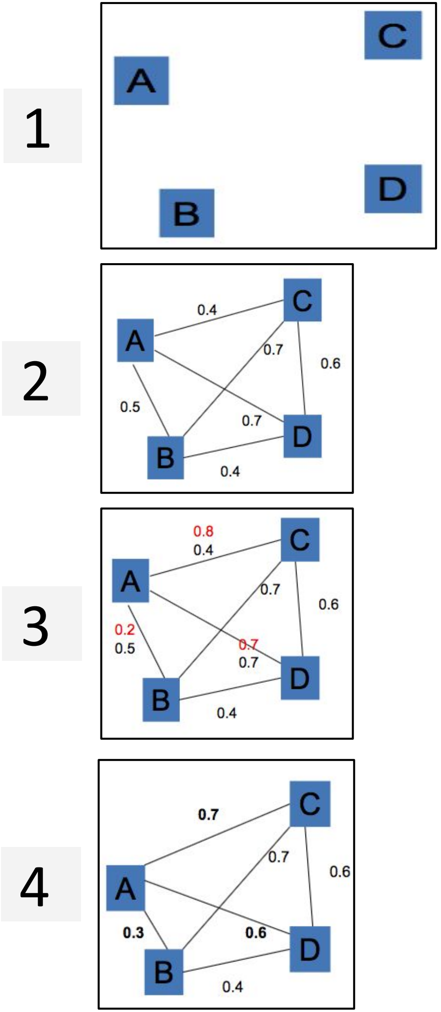
Table 3-2. Sample distance matrix

	A	B	C	D
A	0.0	0.2	0.8	0.7
B	0.2	0.0	0.9	0.8
C	0.8	0.9	0.0	0.1
D	0.7	0.8	0.1	0.0

PCI

MDS Overview

1. Randomly drop items in 2d graph
2. Measure all inter-item distances
3. Compare with actual item-item distance for 1 item
4. Move item in 2D space (ex. A moves closer to B (good), further from C (good), closer to D (bad))
5. Repeat steps 2-4 for all items until no more changes can be made without increasing the error



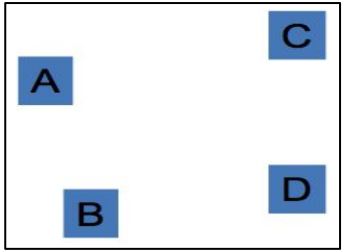
Multidimensional Scaling - Python

- `scaledown(data, distance=pearson, rate=0.01)` - performs the MDS algorithm, returns a vector with the x,y coordinates of each item, learning rate (how much to move items) is default 0.01

- `draw2d(data, labels, jpeg="mds2d.jpg")` - function to display the labels in 2d space

Original code: [Ch 3 GitHub repo]

scaledown() - Step 1




```
  n = len(data)

# The real distances between every pair of items
  realdist = [[distance(data[i], data[j]) for j in range(n)] for i in
              range(0, n)]

# Randomly initialize the starting points of the locations in 2D
  loc = [[random.random(), random.random()] for i in range(n)]
  fakedist = [[0.0 for j in range(n)] for i in range(n)]
```

n - number of rows

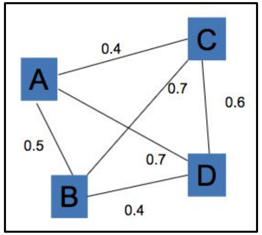
`realdist` - compute actual distance (with pearson) between every pair of items

`loc` - will hold the 2d x,y locations, initialize to random points

`fakedist` - will hold chart distance between every pair of items

Original code: [Ch 3 GitHub repo]

`scaledown()` - Step 2

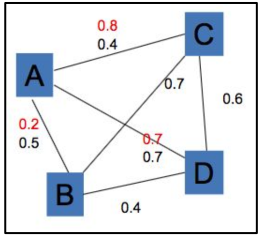



```
# Find projected distances
       for i in range(n):
           for j in range(n):
               fakedist[i][j] = sqrt(sum([pow(loc[i][x] - loc[j][x], 2)
                                     for x in range(len(loc[i]))]))
```

`fakedist[i][j]` - Euclidean distance between position of row i and position of row j on the graph

Original code: [Ch 3 GitHub repo]

scaledown() - Step 3

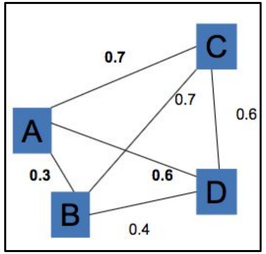



```
        for k in range(n):
            for j in range(n):
                if j == k:
                    continue
        # The error is percent difference between the distances
                errorterm = (fakedist[j][k] - realdist[j][k]) / realdist[j][k]
      [...]
      # Keep track of the total error
              totalerror += abs(errorterm)
```

`errorterm` - percent difference between the distances (real distance and graph distance)

Original code: [Ch 3 GitHub repo]

`scaledown()` - Step 4




```
    # Each point needs to be moved away from or towards the other
    # point in proportion to how much error it has
            grad[k][0] += (loc[k][0] - loc[j][0]) / fakedist[j][k] * errorterm
            grad[k][1] += (loc[k][1] - loc[j][1]) / fakedist[j][k] * errorterm

  [...]
# Move each of the points by the learning rate times the gradient
    for k in range(n):
        loc[k][0] -= rate * grad[k][0]
        loc[k][1] -= rate * grad[k][1]
```

Original code: [Ch 3 GitHub repo]

scaledown() - Step 5

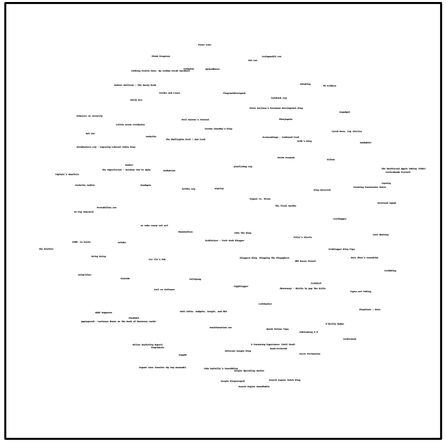
```
# If the answer got worse by moving the points, we are done
    if lasterror and lasterror < totalerror:
        break
    lasterror = totalerror
```

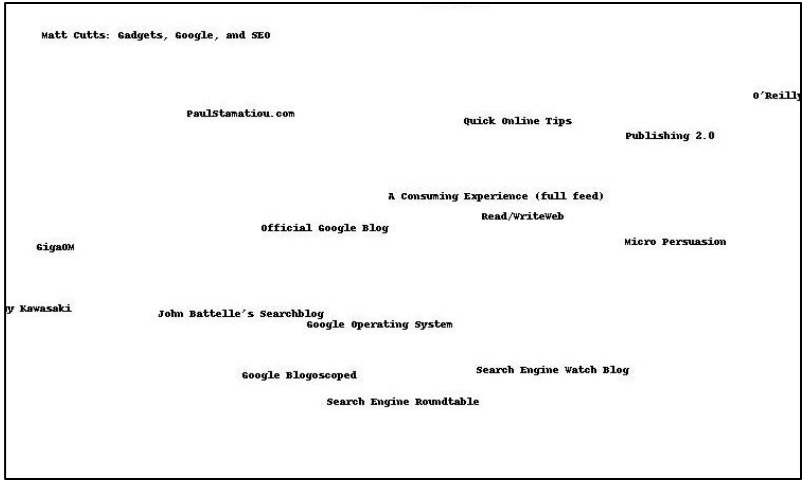
Original code: [Ch 3 GitHub repo]

Example (pg. 52)

- Run MDS on blogdata

```
coords = scaledown(data)
draw2d(coords, blognames, jpeg='blogs2d.jpg')
```





Original code: [Ch 3 GitHub repo]

Objectives

- Distinguish between unsupervised learning and supervised learning.
- Differentiate between agglomerative and divisive clustering.
- Explain how a dendrogram is constructed.
- Explain the main steps in k-means clustering.
- Describe the purpose of multidimensional scaling (MDS).
- Explain the steps of multidimensional scaling.