Clustering

Dr. Johan Hagelbäck

johan.hagelback@lnu.se

http://aiguy.org

Discovering Groups

- Recommendation Systems is one way of, for example, finding users similar to yourself
- We will now look into a related method called *data clustering*
- It is used for discovering and visualizing groups of items, people, blog entries, …, that are closely related
- We will take a look at two different algorithms: *hierarchical* and *k-means* clustering

The dataset

- First, we need a dataset to work with
- The dataset consists of 120 blogs
- The data for each blog is the number of times a particular word appear in the feed
- There are in total 706 pre-selected words
- The data is in a tab-separated text file
- A small subset of the data looks like:

The dataset

Blog/Word	”china”	”kids”	”music”	”yahoo”
Gothamist	0	3	3	0
GigaOM	6	0	0	2
Quick Online Tips	0	2	2	22

The basic idea

- If we can cluster blogs based on word frequencies, we might find groups of similar blogs
- It can be useful when searching, cataloging and discovering online blogs

- The dataset can be downloaded at the course web page
- It is also possible to generate a new data set using a blog feed parser tool

The dataset

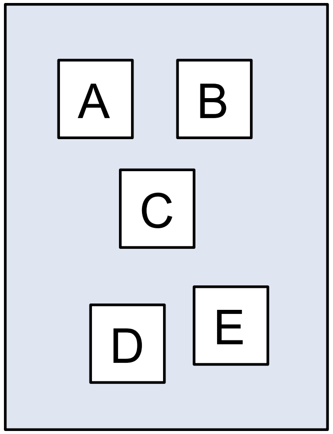
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
1	Blog	china	kids	music	yahoo	want	wrong	service	tech	saying	lots	had	address	working	following	years	d
2	The Superficial - Because You're Ugly	0	1	0	0	3	3	0	0	3	0	6	0	1	0	4	
3	Wonkette	0	2	1	0	6	2	1	0	4	5	25	0	0	0	6	
4	Publishing 2.0	0	0	7	4	0	1	3	6	0	0	1	0	0	0	1	
5	Eschaton	0	0	0	0	5	3	2	0	1	0	1	0	1	0	0	
6	Blog Maverick	2	14	17	2	45	11	8	0	4	7	24	2	4	3	12	
7	Mashable!	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	
8	we make money not art	0	1	1	0	6	0	0	1	0	1	21	3	20	4	6	
9	GigaOM	6	0	0	2	1	0	3	1	0	0	1	0	0	1	1	
10	Joho the Blog	0	0	1	4	0	0	1	0	0	0	4	1	0	0	1	
11	Neil Gaiman's Journal	0	0	0	0	2	1	0	0	3	8	28	0	1	0	5	
12	Signal vs. Noise	0	0	0	0	12	2	0	2	4	3	8	2	2	0	1	
13	lifehack.org	0	0	0	0	2	1	0	0	0	2	0	0	1	1	1	
14	Hot Air	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
15	Online Marketing Report	0	0	0	3	0	0	0	0	0	0	0	0	2	3	0	
16	Kotaku	0	5	2	0	10	1	3	0	4	1	13	0	0	1	0	
17	Talking Points Memo: by Joshua Micah Marshall	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	
18	John Battelle's Searchblog	0	0	0	3	1	0	1	0	0	0	1	1	0	0	0	
19	43 Folders	0	0	0	0	1	0	0	0	1	1	0	0	0	0	1	
20	Daily Kos	0	1	0	0	9	4	0	0	1	1	6	0	0	0	2	
21	Deadspin	1	2	3	0	0	5	1	0	4	1	25	0	0	0	4	
22	Go Fug Yourself	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	
23	O'Reilly Radar	7	0	3	4	6	0	7	1	1	0	1	1	4	1	2	
24	Andrew Sullivan | The Daily Dish	0	0	0	0	0	1	2	0	1	0	1	1	0	0	1	

Hierarchical Clustering

Hierarchical Clustering

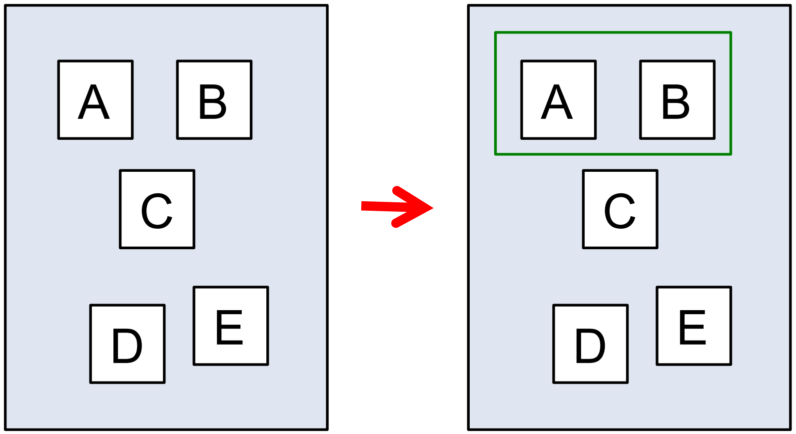
- In HC, a tree hierarchy of groups are constructed by continuously merging the two most similar groups
- Each group starts as a single blog
- In each iteration, the distances to all other groups are calculated and the two closest ones are merged to form a new group
- This is repeated until we only have one large group

Hierarchical Clustering



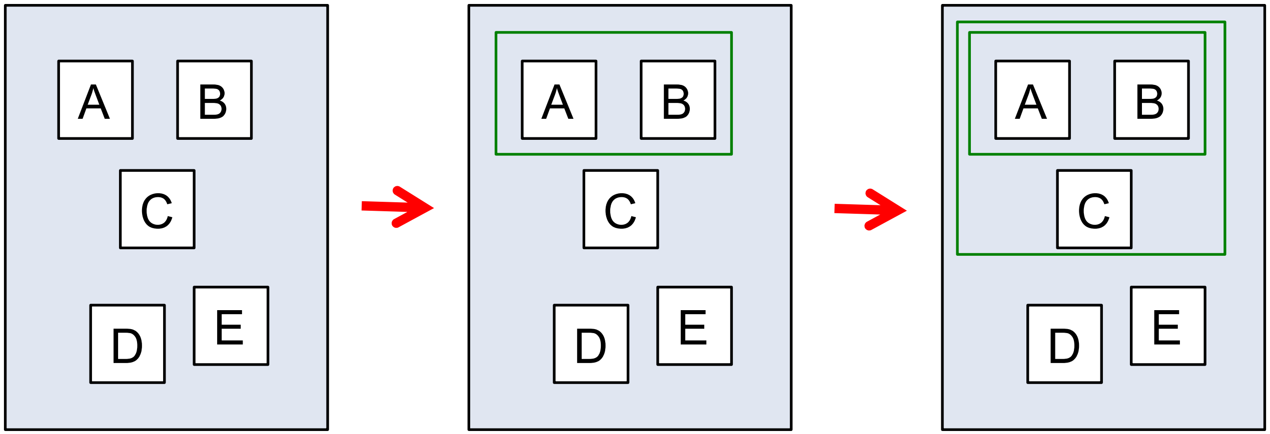
Groups
A
B
C
D
E

Hierarchical Clustering



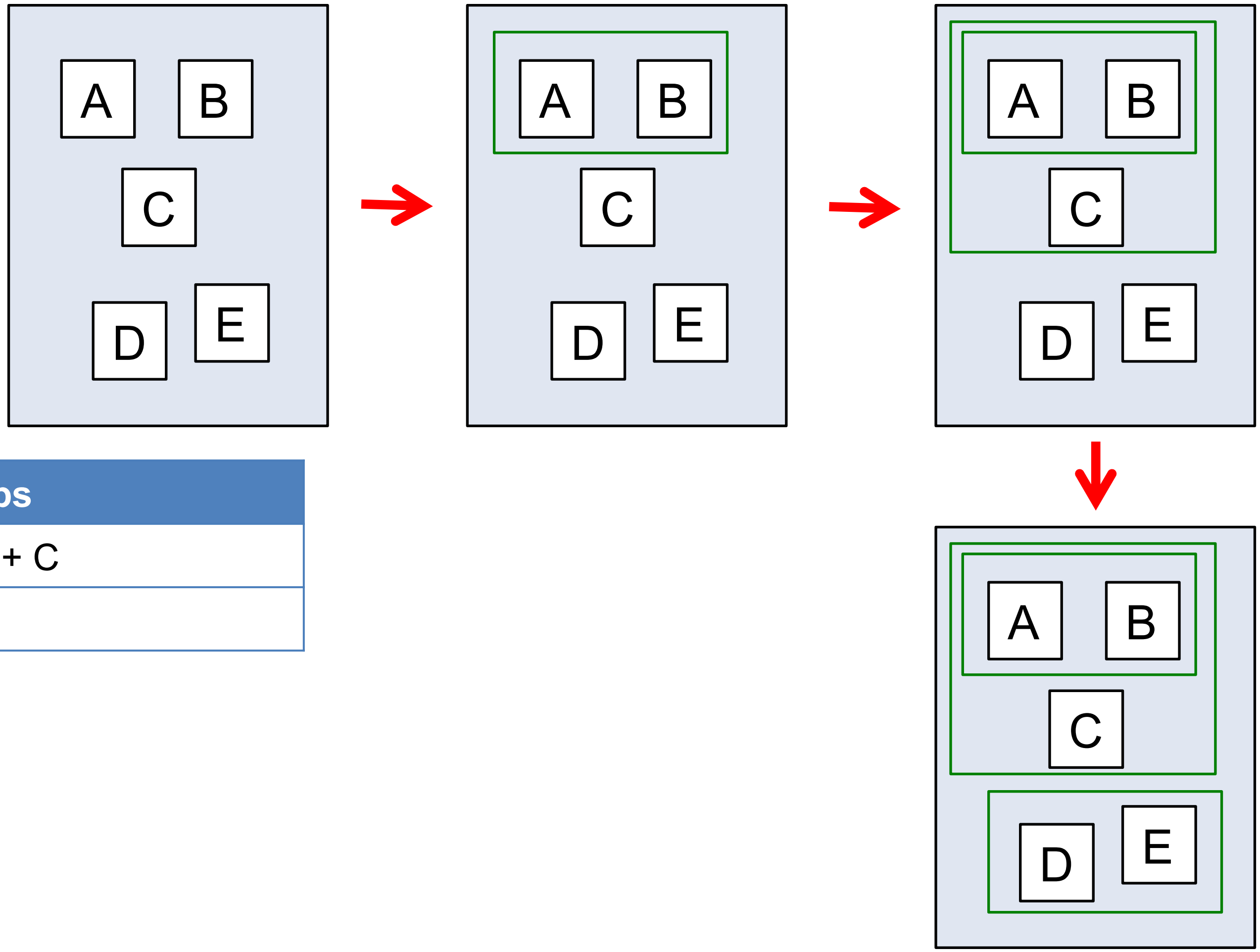
Groups
A + B
C
D
E

Hierarchical Clustering



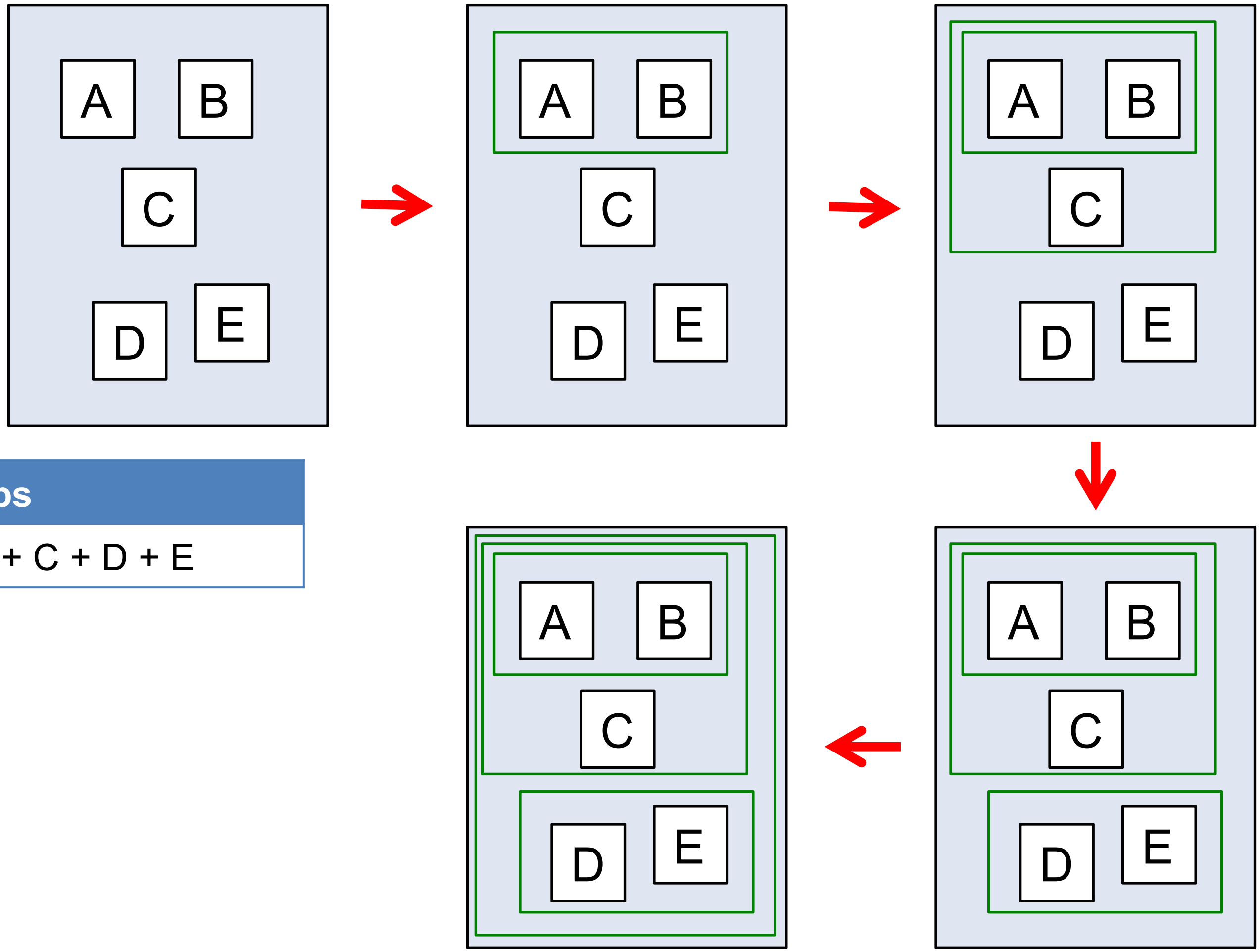
Groups
A + B + C
D
E

Hierarchical Clustering



Groups
A + B + C
D + E

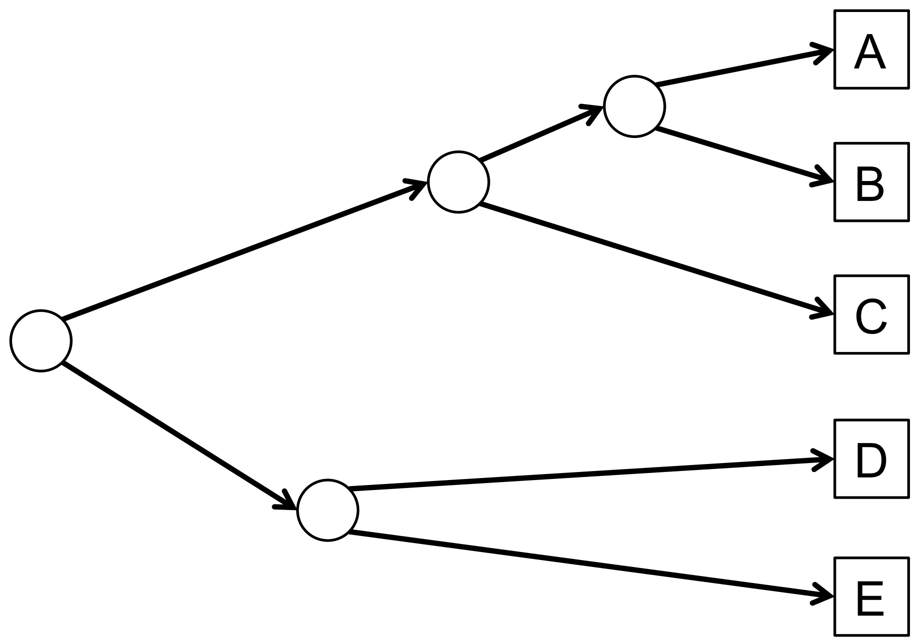
Hierarchical Clustering



Groups
A + B + C + D + E

Dendrogram

- We can display the result in a graph called *dendrogram*, which displays the nodes arranged into their hierarchy:



Closeness

- Central to the algorithm is a measure of how close two blogs are
- In the previous lecture we defined two similarity measures: *Euclidean* and *Pearson*
- In the dataset, some blogs contain more words than other blogs since they have more or longer blog entries
- Euclidean is not very good in this case
- We will therefore focus on Pearson

Pearson distance

- The more similar two blogs are, the smaller the distance value between the blogs
- If two blogs match perfectly, Pearson returns 1.0
- If they don’t match at all, Pearson returns -1.0
- The Pearson value must therefore be inverted

- The Pearson distance is calculated as:

Pearson distance

```
float pearson(Blog A, Blog B)
  //Init variables
  sumA=0, sumB=0, sumAsq=0, sumBsq=0, pSum=0
  //Number of words
  n = 706
  //Iterate over all words
  for (i = 0 to n)
    cntA = A.word_count(i) //word counts for each word in A
    cntB = B.word_count(i) //word counts for each word in B
    sumA += cntA //sum of word counts for A
    sumB += cntB //sum of word counts for B
    sumAsq += cntA**2 //sum of squared word counts for A
    sumBsq += cntB**2 //sum of squared word counts for B
    pSum += cntA * cntB //product of word counts from A and B

  //Calculate Pearson
  num = pSum – (sumA * sumB / n)
  den = sqrt((sumAsq – sumA**2 / n) * (sumBsq – sumB**2 / n))
  //Return inverted Pearson score
  return 1.0 - num/den
```

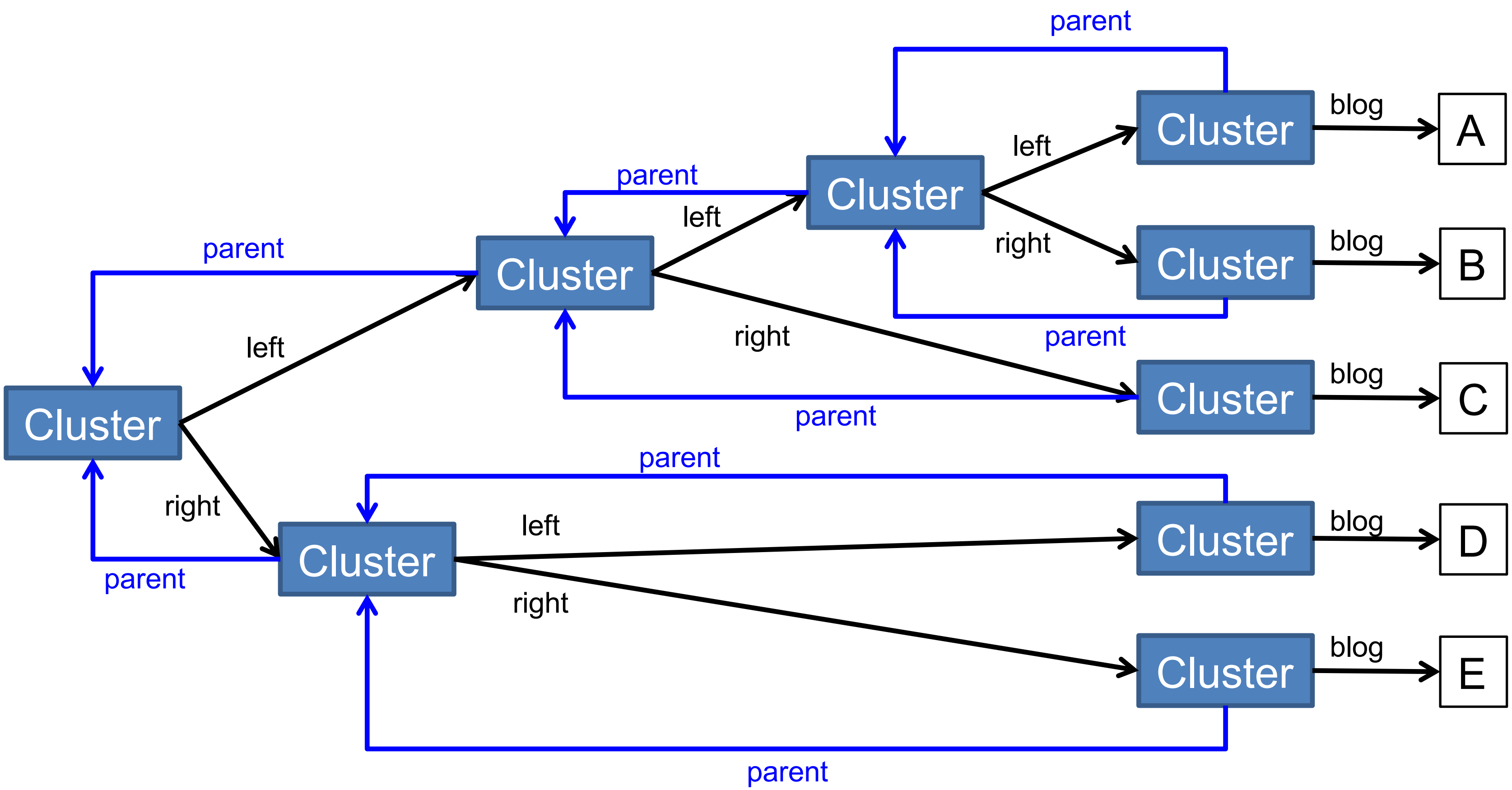
Tree data structure

- It is recommended to model the clusters as a tree
- Each node (called cluster) in the tree contains either a Blog (leaf node), or two branches to other nodes
- We will also save the distance measure between the two branches (or 0 if node is a leaf node with a Blog)
- … and a reference to the parent node
- The data structure we will use looks like this:

Cluster

```
Cluster left; //null if leaf node
Cluster right; //null if leaf node
Blog blog;
double distance; //0 if leaf node
```

The tree data structure



Merging

- Merging two clusters A and B is a bit tricky
- Do the following steps:

```
Create a new cluster P
Set P.left to A
Set P.right to B
Create a new blog bP for cluster P
Fill the blog by averaging word counts for each word
from the blogs in A and B
Set A.parent to P
Set B.parent to P
Set the distance score to P
```

Merge algorithm

```
//Merge two clusters A and B
Cluster merge(Cluster A, Cluster B, double distance)
  //Number of words
  n = 706
  //Create new cluster
  Cluster P
  //Fill data
  P.left = A
  A.parent = P
  P.right = B
  B.parent = P

  //Merge blog data by averaging word counts for each word
  Blog nB
  for (i = 0 to n)
    double cntA = A.blog.word_count(i)
    double cntB = B.blog.word_count(i)
    //Average word count
    double cnt = (cntA + cntB) / 2
    //Set word count to new blog
    nB.set_word_count(i, cnt)

  //Set blog to new cluster
  P.blog = nB
  //Set distance
  P.distance = distance

  //Return new cluster
  return P
```

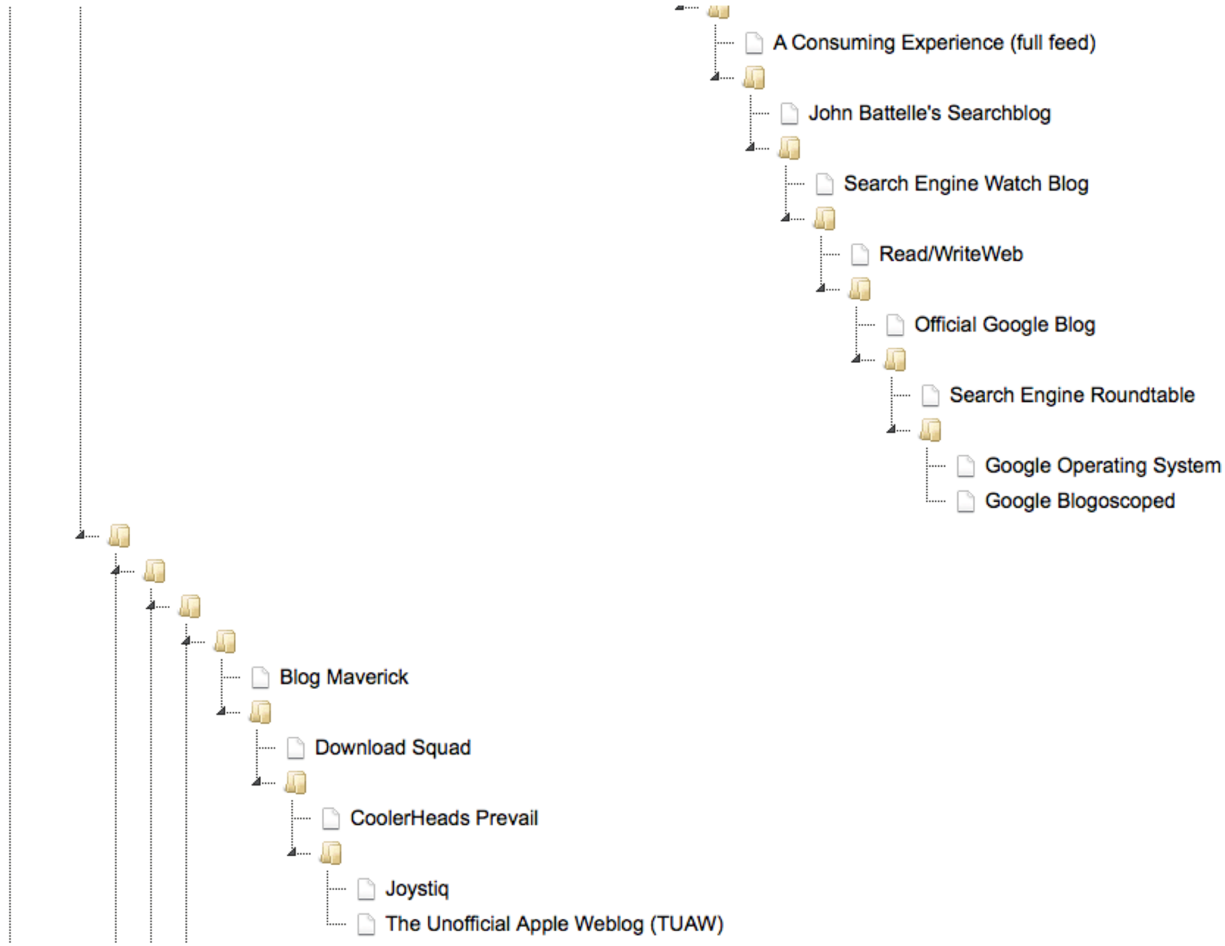
Hierarchy generation algorithm

- Start by generating one Cluster for each blog
- Then iteratively merge to clusters one at a time until only one cluster remains:

Hierarchy generation algorithm

```
//Iterate as long as there are more than one Cluster in the
//Clusters list
void iterate()
  //Find two closest nodes
  double closest = Double.MAX_VALUE
  Cluster A
  Cluster B
  foreach (Cluster cA : clusters)
    foreach (Cluster cB : clusters)
      double distance = pearson(cA.blog, cB.blog)
      if (distance < closest && cA != cB)
        //New set of closest nodes found
        closest = distance
        A = cA
        B = cB

  //Merge the two clusters
  Cluster nC = merge(A, B, closest)
  //Add new cluster
  clusters.add(nC)
  //Remove old clusters
  clusters.remove(A)
  clusters.remove(B)
```

Displaying the result



Performance issues

- Calculating the distance score between two blogs of 706 words is rather time consuming
- And the algorithm iterates several times, 98 in our example data set
- In total 161700 similarity measures are calculated

- It might take a while to generate and display the tree…
- Can you find any performance improvements?

K-means Clustering

K-means Clustering

- Hierarchical Clustering has some drawbacks:
  - Data is not broken down into distinct groups without additional computation
  - The algorithm is very slow
- An alternative is to use *K-means clustering*
- It is quite different from HC because we tell the algorithm in advance how many clusters we want
- The algorithm will then determine the size of each cluster based on the data

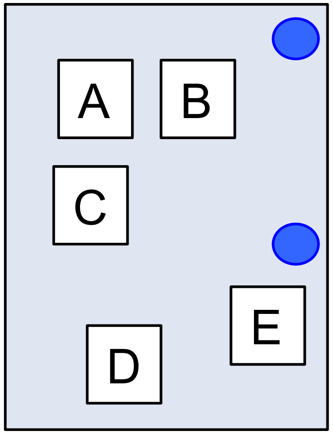
Centroid

- Central to the algorithm is *centroids*
- A centroid is a point in n-dimensional space that represents the center of a cluster
- Each centroid is placed at a random point at the beginning of the algorithm
- This means that for the blog data example we have 706 words
- Each centroid must then have 706 randomly generated counts ranging from min to max for that specific word
- For example the word “china” occurs between 0 and 11 times in the blogs, so the random count must be between 0 and 11

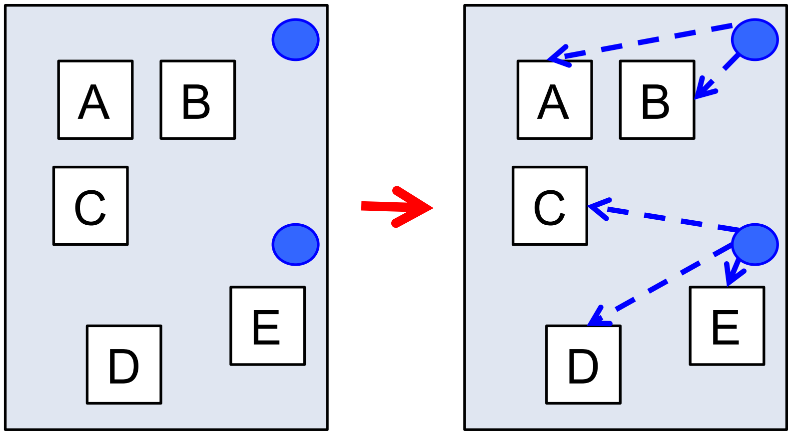
K-means clustering

- The algorithm works as follows:
  - Decide how many clusters that are needed, *k*
  - Randomly place *k* centroids
  - Assign every Blog to the closest centroid
  - After all blogs have been assigned, each centroid is moved to the average location of all blogs assigned to the cluster
  - Repeat for *i* iterations:
    - Clear assignments
    - Assign every Blog to the closest centroid
    - Move centroid to average of cluster

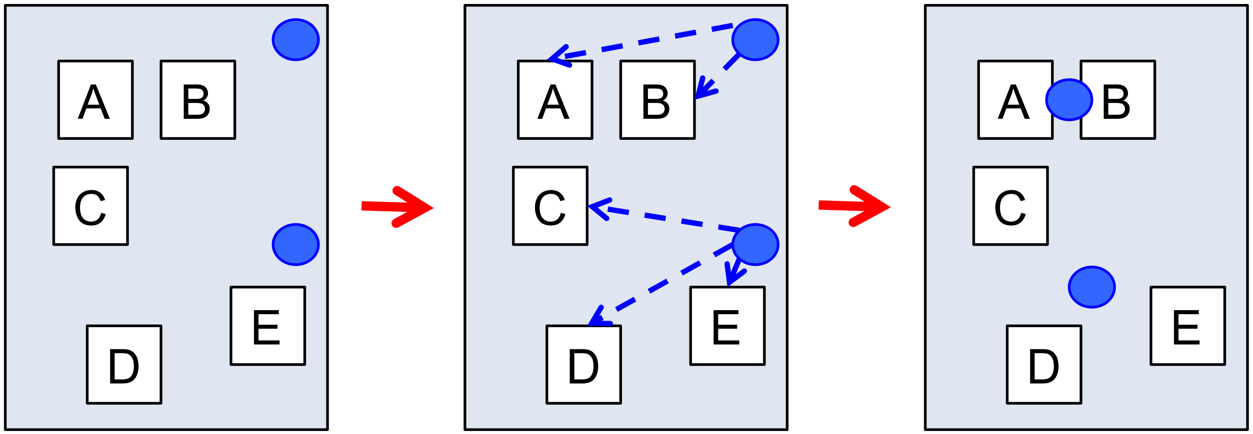
Randomly place 2 centroids



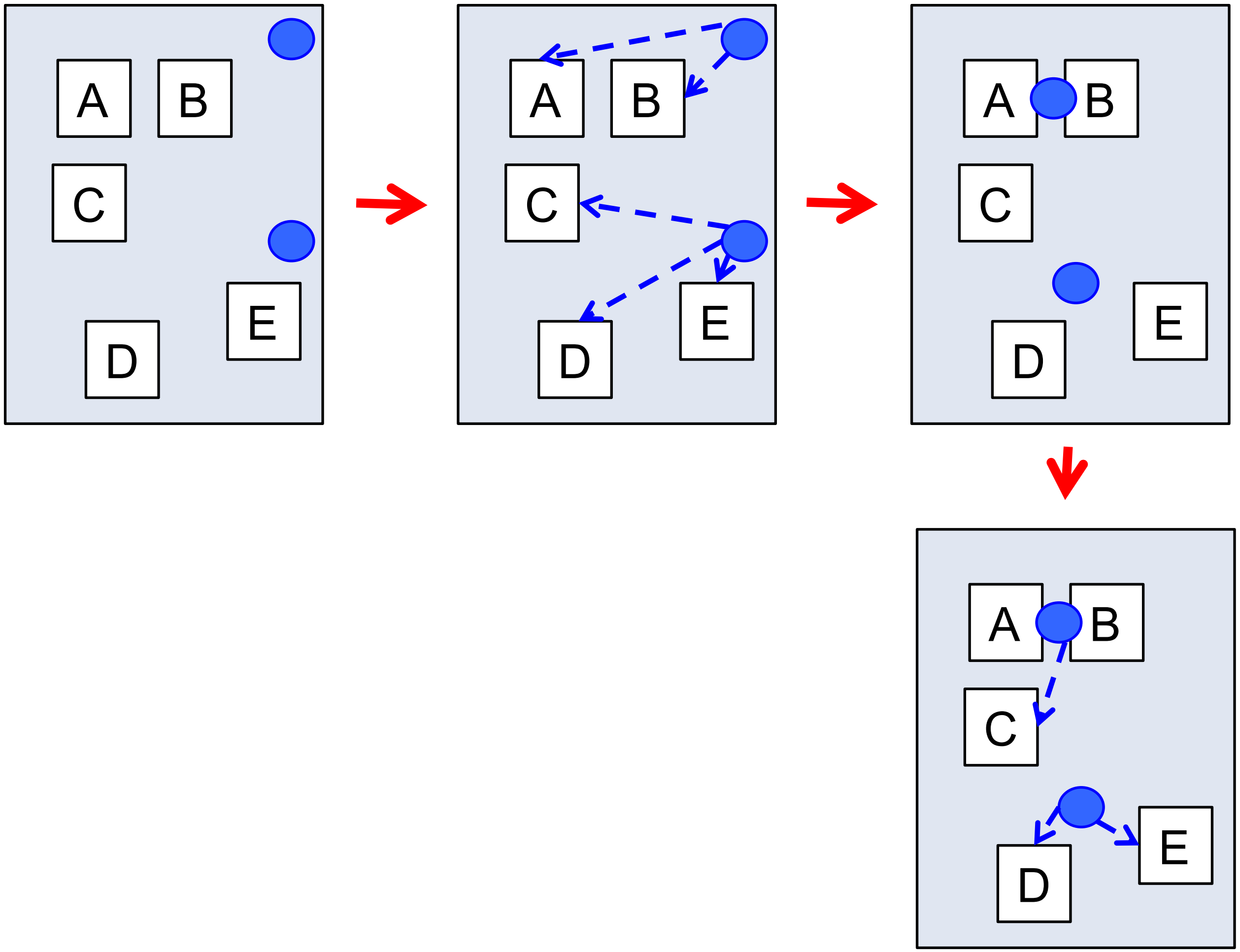
Assign blogs to the closest centroid



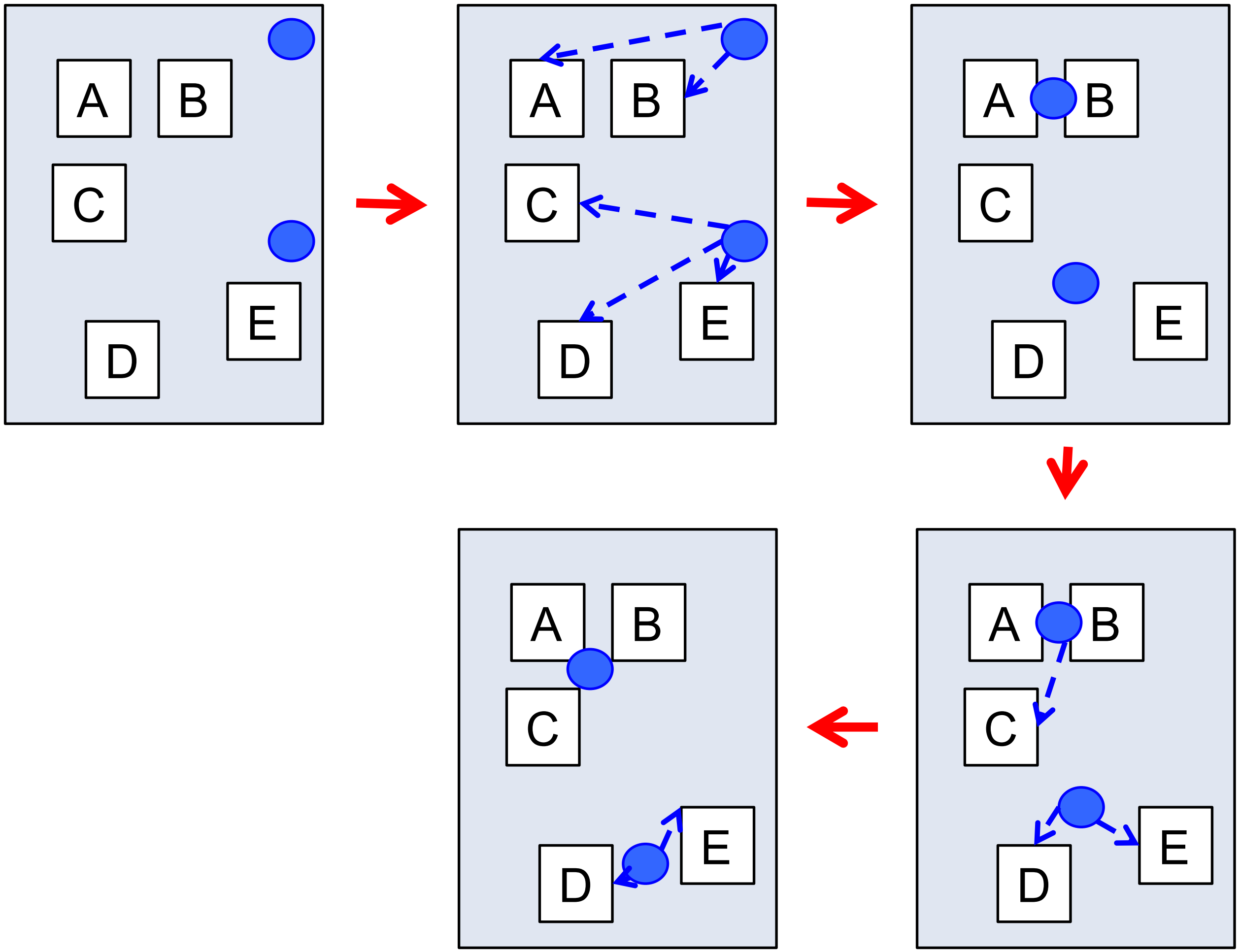
Move centroids to center of cluster



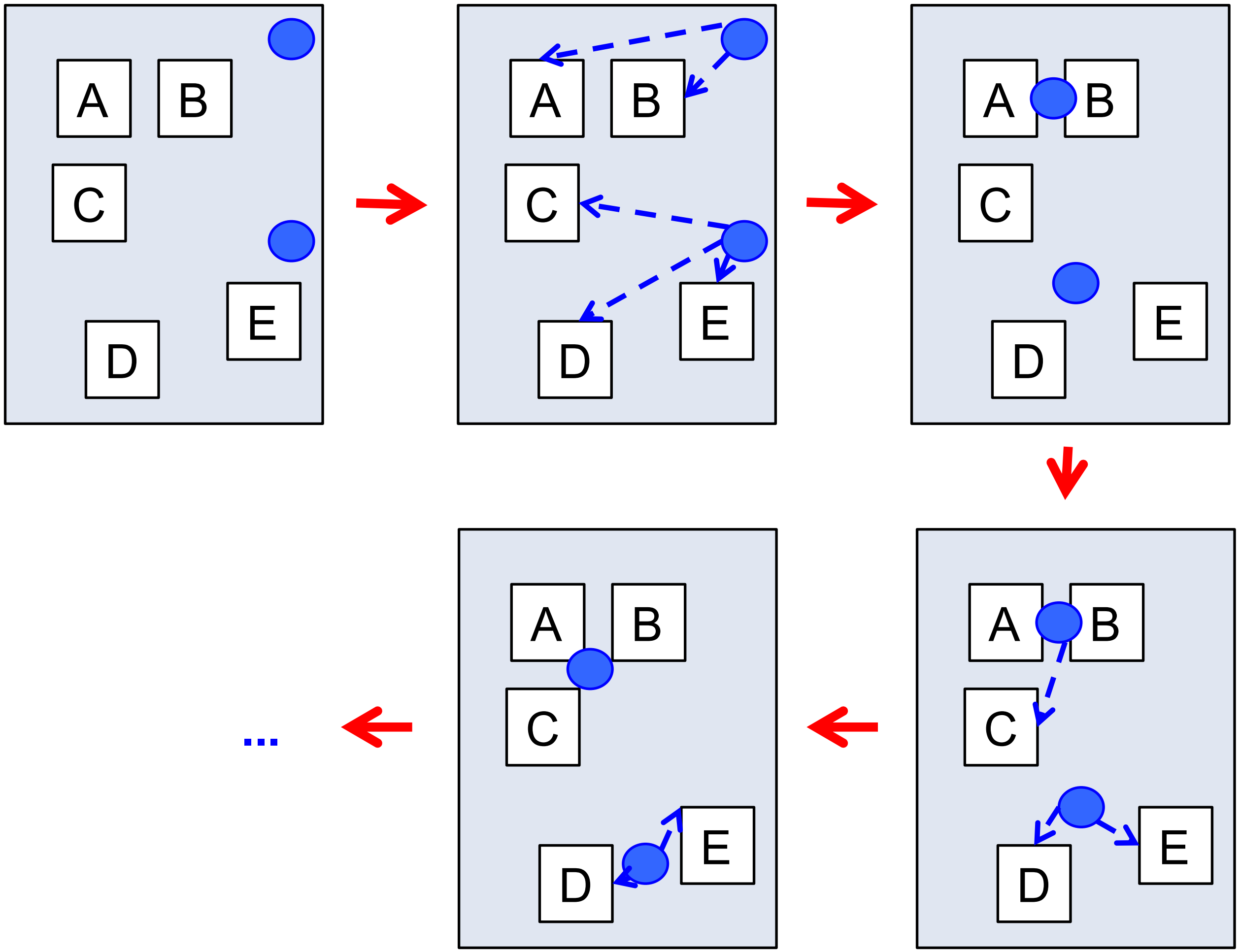
Re-assign blogs to closest centroids



Move centroids to center of cluster



Continue iterating until we reach a stable solution



The algorithm

```
//Number of words
n = 706
//Generate K random centroids
centroids = List()
for (c = 0 to K)
  Centroid c
  for (i = 0 to n)
    c.set_word_count(i, random(min[i], max[i]))

//Iteration loop
for (i = 0 to MAX_ITERATIONS)
  //Clear assignments for all centroids
  centroids.clearAssignments()

  //Assign each blog to closest centroid
  foreach (Blog b : blogs)
    double distance = Double.MAX_VALUE
    Centroid best
    //Find closest centroid
    for (Centroid c : centroids)
      double cDist = pearson(c, b)
      if (cDist < distance)
        best = c
        distance = cDist
    //Assign blog to centroid
    best.assign(b)
```

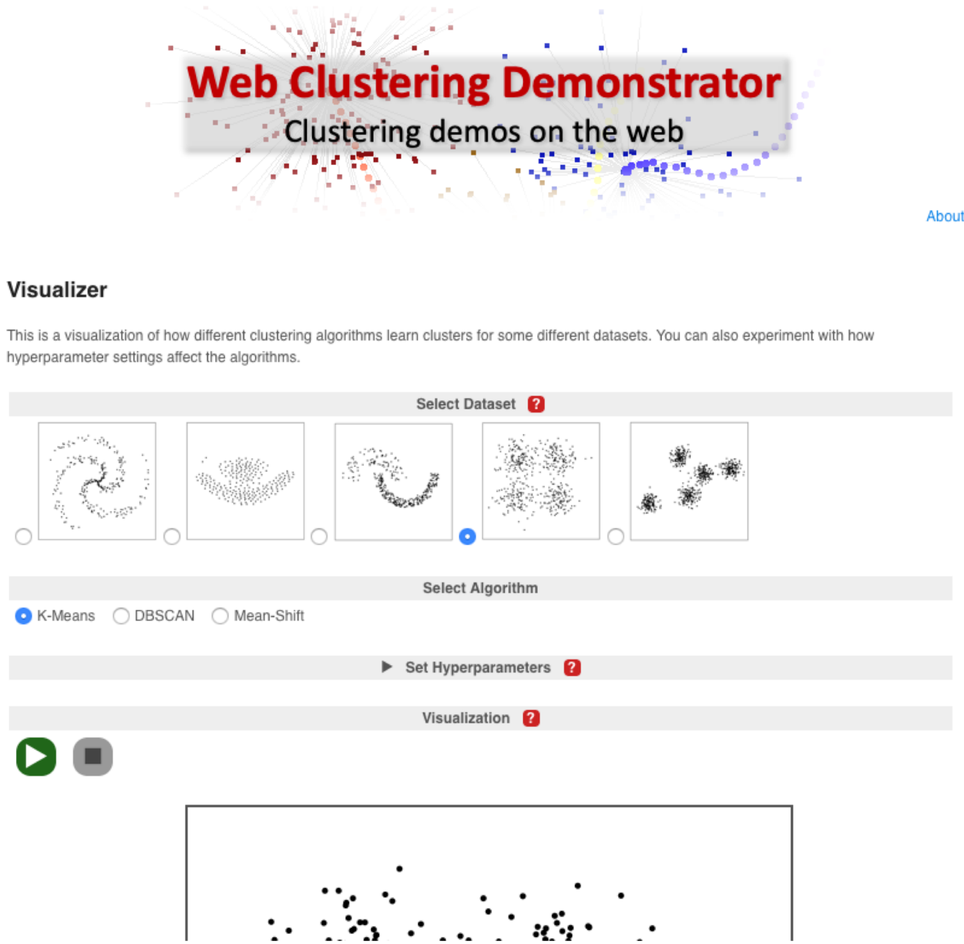
The algorithm

```
  //Re-calculate center for each centroid
  foreach (Centroid c : centroids)
    //Find average count for each word
    for (i = 0 to n)
      double avg = 0
      //Iterate over all blogs assigned to this centroid
      foreach (Blog b : c.assignments)
        avg += b.word_count(i)
      avg /= c.assignments.length()

      //Update word count for the centroid
      c.set_word_count(i, avg)

//End of iteration loop – all done
```

When is it finished?

- The algorithm must stop the iteration at some point
- To do this you:
  - Define a maximum number of iterations, for example 20
  - Always stop if the previous assignment is identical to the new assignment

K-means clustering



http://aiguy.org/webclust

Hierarchical vs. K-means

- K-means requires very few iterations compared to Hierarchical, and is significantly faster
- K-means is initialized with randomly placed centroids, therefore the result can differ between executions
- Hierarchical is always consistent

K-means, two runs

- Cluster 1 (22)
- Cluster 2 (22)
  - Publishing 2.0
  - Mashable!
  - GigaOM
  - Online Marketing Report
  - John Battelle's Searchblog
  - O'Reilly Radar
  - Lifehacker
  - Google Operating System
  - Topix.net Weblog
  - Micro Persuasion
  - Search Engine Watch Blog
  - Techdirt
  - Official Google Blog
  - Search Engine Roundtable
  - TechCrunch
  - PaulStamatiou.com
  - A Consuming Experience (full feed)
  - Matt Cutts: Gadgets, Google, and SEO
  - Bloglines | News
  - Quick Online Tips
  - Google Blogoscoped
  - Read/WriteWeb
- Cluster 3 (24)
- Cluster 4 (24)
- Cluster 5 (7)
  - Blog Maverick
  - Joystiq
  - Download Squad
  - Engadget
  - CoolerHeads Prevail
  - The Unofficial Apple Weblog (TUAW)
  - Autoblog

- Cluster 1 (6)
  - Blog Maverick
  - Joystiq
  - Download Squad
  - Engadget
  - CoolerHeads Prevail
  - The Unofficial Apple Weblog (TUAW)
- Cluster 2 (25)
  - Publishing 2.0
  - Mashable!
  - GigaOM
  - John Battelle's Searchblog
  - O'Reilly Radar
  - Lifehacker
  - Google Operating System
  - Valleywag
  - Slashdot
  - Topix.net Weblog
  - Micro Persuasion
  - Search Engine Watch Blog
  - Techdirt
  - Official Google Blog
  - Search Engine Roundtable
  - Shoemoney - Skills to pay the bills
  - Signum sine tinnitu--by Guy Kawasaki
  - TechCrunch
  - Wired News: Top Stories
  - A Consuming Experience (full feed)
  - Matt Cutts: Gadgets, Google, and SEO
  - Bloglines | News
  - Quick Online Tips
  - Google Blogoscoped
  - Read/WriteWeb
- Cluster 3 (9)
- Cluster 4 (47)
- Cluster 5 (12)

Clustering

Dr. Johan Hagelbäck

johan.hagelback@lnu.se

http://aiguy.org