DATA, ALGORITHMS AND MEANING

THE STORY SO FAR…

Core concepts will be introduced (with minimal maths) along with worked applications using R

Topics include

> Linear regression, logistic regression, PCA, regularization, Bash Skills ✓

> Decision trees and ensemble models, Caret ✓

> Support vector machines, neural nets and AWS. ✓

> Text mining, clustering, network graphs, LSA, Topic Models and Word Embedding.

OVERVIEW OF TODAY'S SESSION

Text mining basics

Clustering

Network Graphs

Latent Semantic Analysis

Topic Modelling

Word Embedding

Naïve Bayes

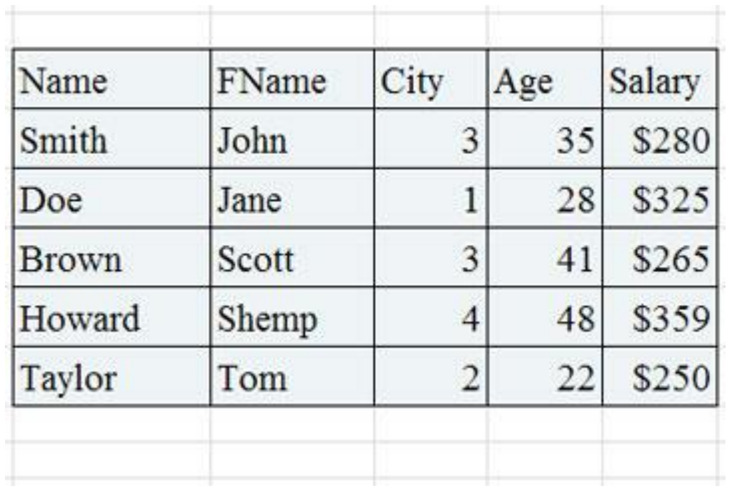
Assignment 3 Briefing

Wrap Up / Q&A

Basics of Text Mining

TEXT MINING – SOME PRELIMINARIES

Structured data



Name	FName	City	Age	Salary
Smith	John	3	35	$280
Doe	Jane	1	28	$325
Brown	Scott	3	41	$265
Howard	Shemp	4	48	$359
Taylor	Tom	2	22	$250

Unstructured data

“Joe Bloggs readily admits when he doesn’t know the answer to a particular query. He outlines the steps that he will take to resolve a problem. However, he has difficulty saying no or tactfully telling customers that they must wait their turn. He also tends to refers too many queries to management for final resolution.”

TEXT MINING (AKA TEXT ANALYTICS)

What is it?

> The process of deriving useful information from unstructured data using computer-based techniques.

Basics

> Frequencies – how often does a word appear in a corpus.

> Associations – which words appear together? how often?

> Simple visualisations (frequency histograms, wordclouds)

> Naïve measures of document similarity

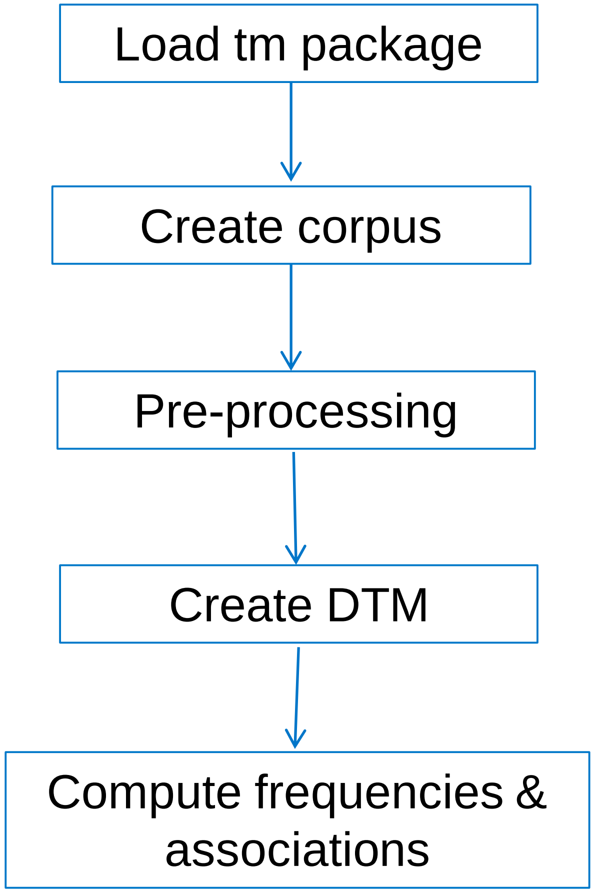
Advanced topics

> Latent semantic analysis –a reduced representation of a corpus (somewhat like PCA for text)

> Topic modelling – automatic, theme-based classification of documents

> Word embedding models - new class of algorithms that can (among other things) detect *semantic* similarities – e.g. synonyms, analogies etc.

THE PROCESS



R’s text mining package

Corpus = collection of docs

Remove punctuation, **stopwords** whitespace &, digits. **Stemming**

Document-term matrix (key step!)

Computed directly from the DTM

Reference: https://eight2late.wordpress.com/2015/05/27/a-gentle-introduction-to-text-mining-using-r/

PREPROCESSING STEPS

Preprocessing?

> **Clean up text** and **remove inessential information**

Steps

> Remove extraneous symbols (punctuation, special symbols)

> Transform to lower case

> Remove digits (unless you need them!)

> Remove stopwords (a, an, the etc.)

> Remove whitespace

> *Stem* to standardise variants (e.g organisation, organise, organising -→ organi). *Lemmatisation* is a more sophisticated technique.

The order matters!

DIFFERENCE BETWEEN STEMMING AND LEMMATISATION

Stemming

Form	Suffix	Stem
studies	-es	studi
studying	-ing	study

Lemmatisation

Form	Morphological information	Lemma
studies	Third person, singular number, present tense of the verb study	study
studying	Gerund of the verb study	study

DOCUMENT-TERM MATRIX

To gain quantitative insights from text we need to transform pre-processed text to a mathematical object.

Achieved by converting text corpus into a matrix where:

> Rows represent documents

> Columns represent terms (words or any other suitable tokens e.g. ngrams)

> Cells contain a number representing term counts (or some other metric)

Such a representation is called a Document-Term Matrix (DTM)

> Note that we can flip this around: i.e. represent terms as rows and documents as columns. Such a representation is called a **Term-Document Matrix (TDM).**

> **Some algorithms take DTMs as inputs, other take TDMs. Be sure to read the documentation to find out which one is needed.**

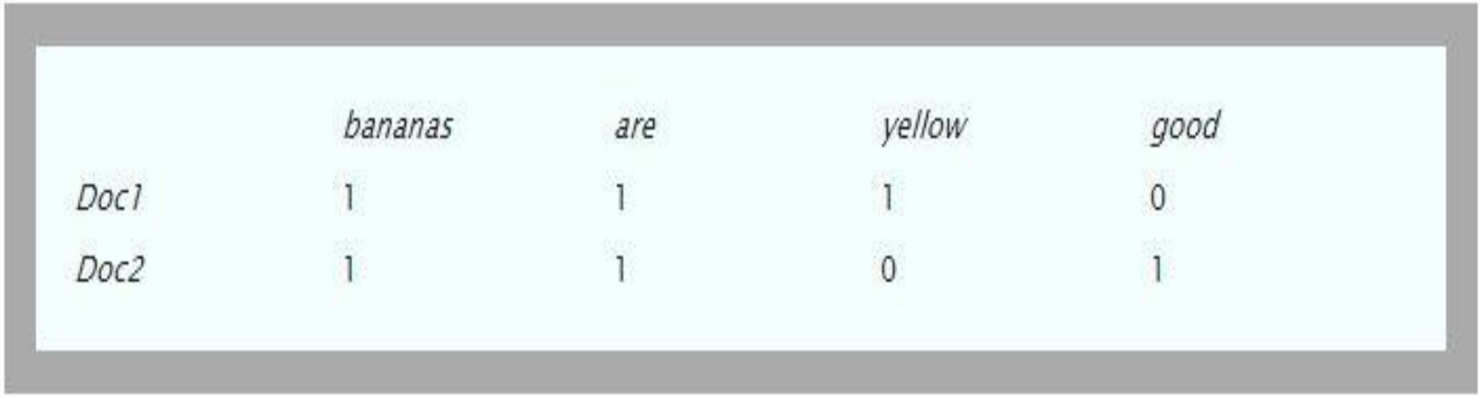
DOCUMENT-TERM MATRIX

A toy example using two documents:

> **Doc1**: bananas are good

> **Doc2**: bananas are yellow

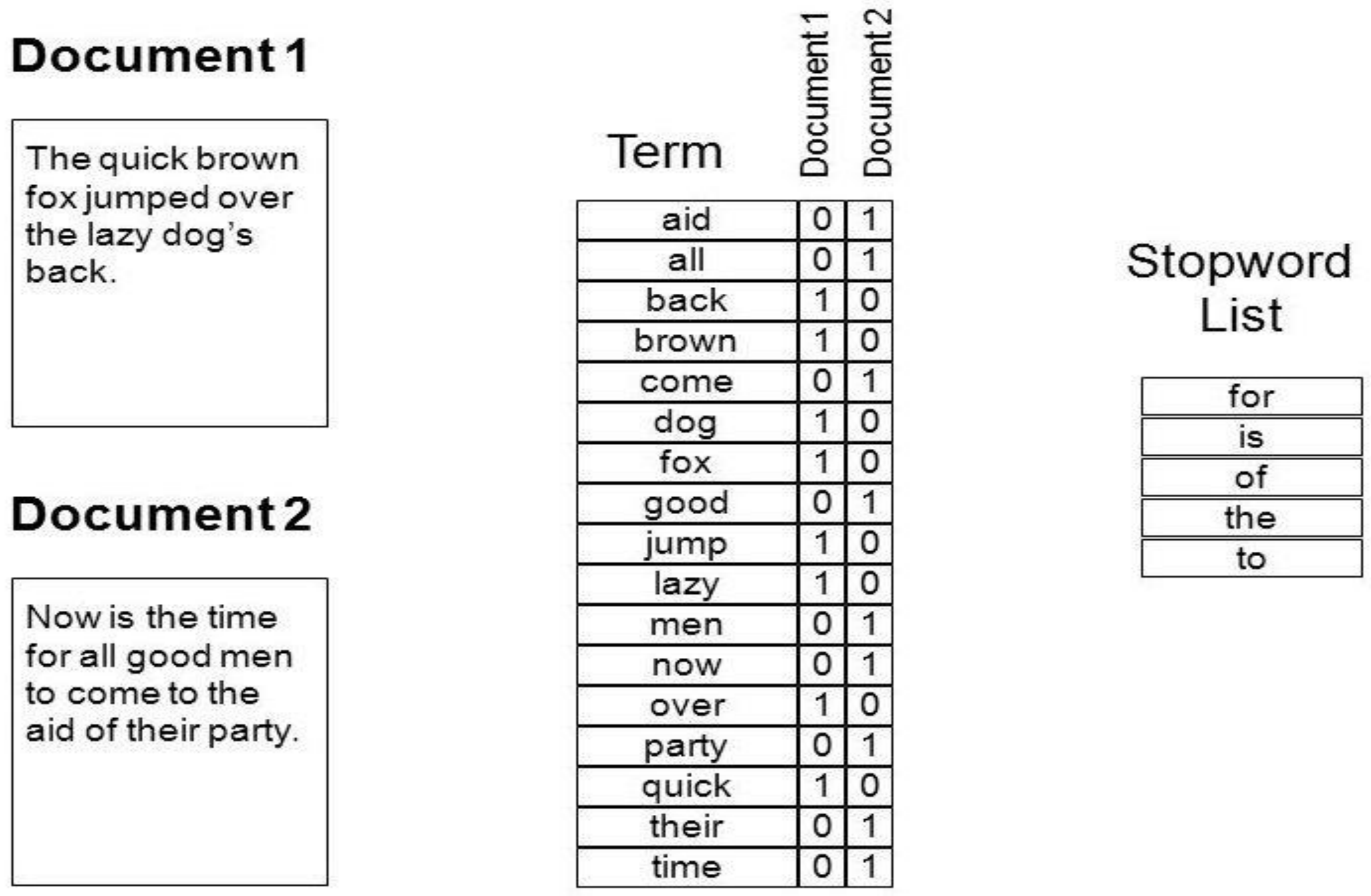
The DTM is:



	bananas	are	yellow	good
Doc1	1	1	1	0
Doc2	1	1	0	1

A DTM is a *mathematical representation of text*, but one in which *order does not matter* (“Bag of Words”)

DOCUMENT-TERM MATRIX – ANOTHER EXAMPLE



Document 1

The quick brown fox jumped over the lazy dog's back.

Document 2

Now is the time for all good men to come to the aid of their party.

Term	Document 1	Document 2
aid	0	1
all	0	1
back	1	0
brown	1	0
come	0	1
dog	1	0
fox	1	0
good	0	1
jump	1	0
lazy	1	0
men	0	1
now	0	1
over	1	0
party	0	1
quick	1	0
their	0	1
time	0	1

Stopword List
for
is
of
the
to

EXPLORING THE DTM – PART 1

The DTM is a quantitative (albeit highly simplified!) representation of the corpus.

Some of the things that can be done using the DTM:

> Document wordcounts

> Word frequencies

> Associations between words

> Building frequency histograms and wordclouds.

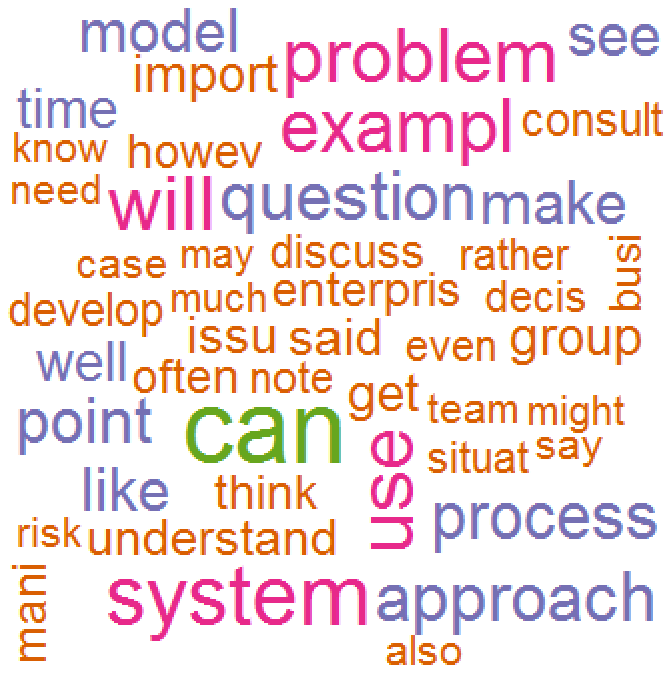
WORDCLOUDS

Use wordcloud package Control look using following parameters:

> scale

> min.freq and max.words

> colors



IN-CLASS EXERCISE

Given a corpus of 49 US presidential speeches.

> Preprocess it
> Create a DTM.
> Calculate word frequencies, find the most/least frequently occurring words.
> Find words that co-occur in documents their co-occurrence correlations.
> Create a frequency histogram & wordcloud of frequently occurring words.
> Figure out how to extract ngrams instead of individual words

Using following R packages:

> tm
> ggplot2
> Wordcloud

Exercise file **01–tm.R** docs in **docs-1** folder

IN-CLASS EXERCISE

Some tips

> Install the required packages before you proceed: **tm**, **ggplot2**, **wordcloud**

> Run the code for part 1 line by line and examine the results.

> Figure out what each step does.

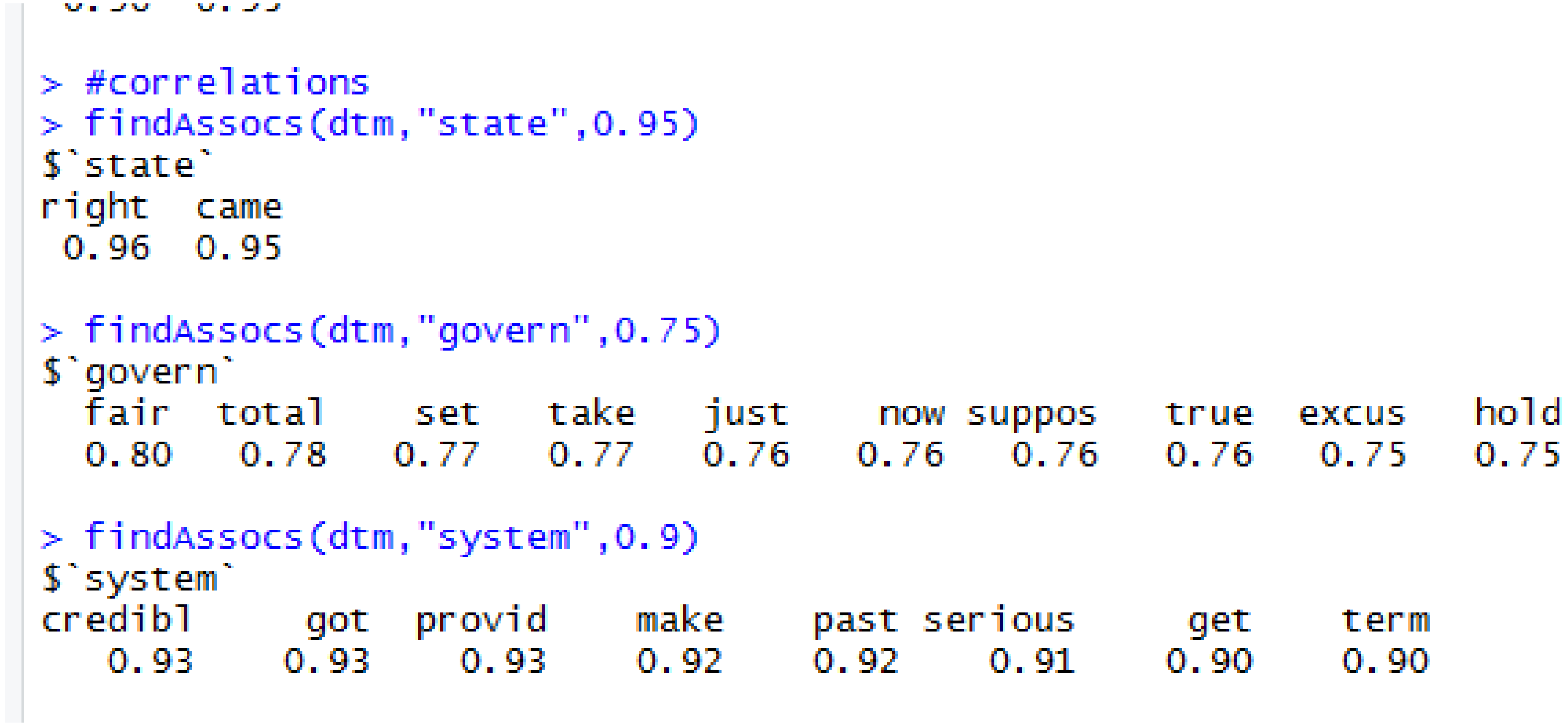
> Check the tm documentation for tm specific functions: **tm_map**, **DocumentTermMatrix**, **findFreqTerms** and **findAssocs**

> Experiment! For example, try with and without stemming; change plot parameters etc.

Reference

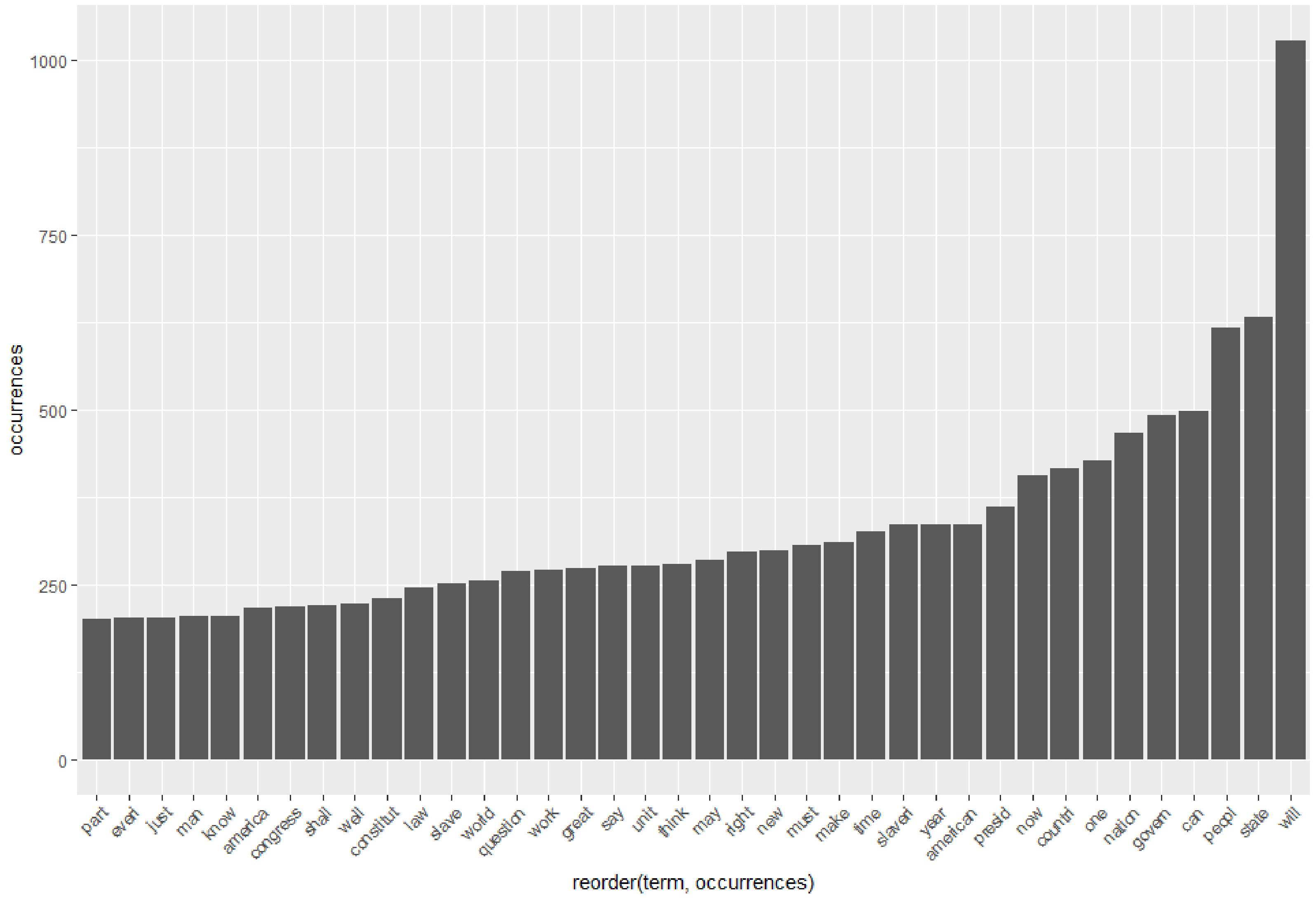
https://eight2late.wordpress.com/2015/05/27/a-gentle-introduction-to-text-mining-using-r/

IN-CLASS EXERCISES

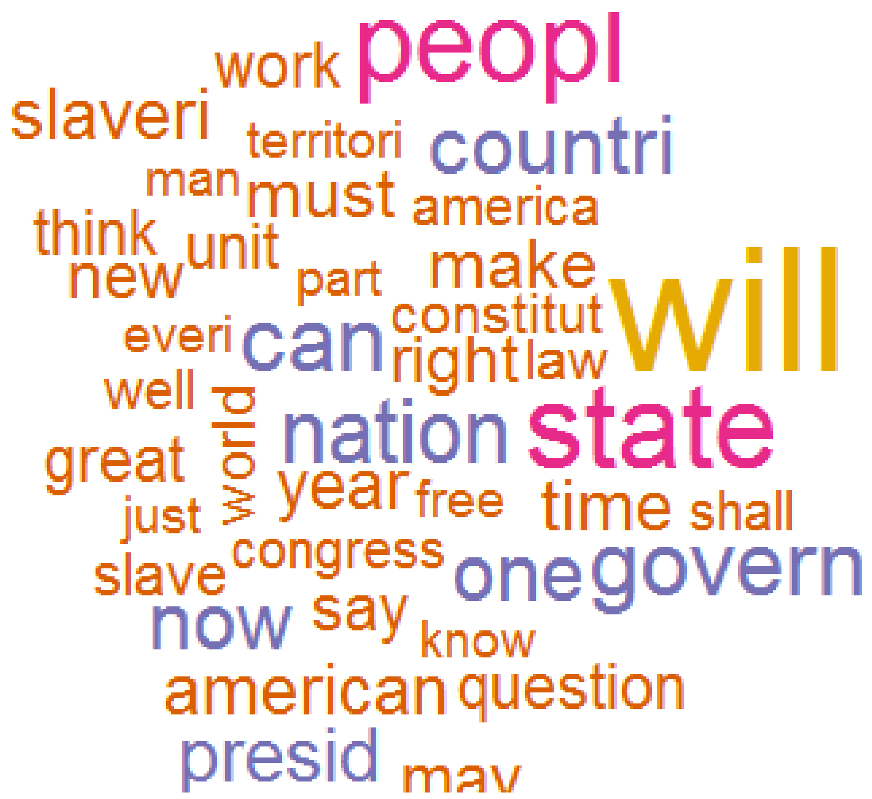


```
> #correlations
> findAssocs(dtm,"state",0.95)
$`state`
right  came
 0.96  0.95

> findAssocs(dtm,"govern",0.75)
$`govern`
  fair  total    set   take   just    now suppos   true  excus   hold
  0.80   0.78   0.77   0.77   0.76   0.76   0.76   0.76   0.75   0.75

> findAssocs(dtm,"system",0.9)
$`system`
credibl     got  provid    make    past serious     get    term
   0.93    0.93    0.93    0.92    0.92    0.91    0.90    0.90
```

FREQUENCY HISTOGRAM



WORDCLOUD


peopl
work
slaveri
territori
countri
man
must
america
think
unit
new
part
make
will
constitut
everi
can
right
law
well
world
nation
state
great
year
free
time
shall
just
congress
slave
one
govern
now
say
know
american
question
presid
may

VARIATIONS ON THE DTM

We’ve created the DTM based on word frequencies. This may not be the best way to represent documents:

> Important keywords may not be the most frequently occurring words

> Frequently occurring words may not be indicative of meaning

> There are several ways to address these issues. We’ll explore two.

On another note: did you notice an issue to do with stopwords?

> We’ll fix that by creating a custom stopword list.

1. EXCLUDE HIGHEST AND LOWEST FREQUENCY TERMS

This is based on the following insights

> Common words are not specific enough and very uncommon ones are too specific (by definition!).

> The same is true of very short and unusually long words.

So let's bound both:

> Can be done in tm using *wordLengths* and *bounds* options in *DocumentTermMatrix* method.

> **Example:**

dtmr <-DocumentTermMatrix(docs, control=list(wordLengths=c(4, 20), bounds = list(global = c(3,45))))

IN-CLASS EXERCISE

Using the same corpus you created in the previous exercise:

> Create a custom stopword list and remove those stopwords from the corpus.

> Create a DTM that contains only words that :

  > have between 4 and 20 characters.

  > Occur in at least 3 and no more than 45 speeches.

> Plot a histogram of words that occur at least 200 times.

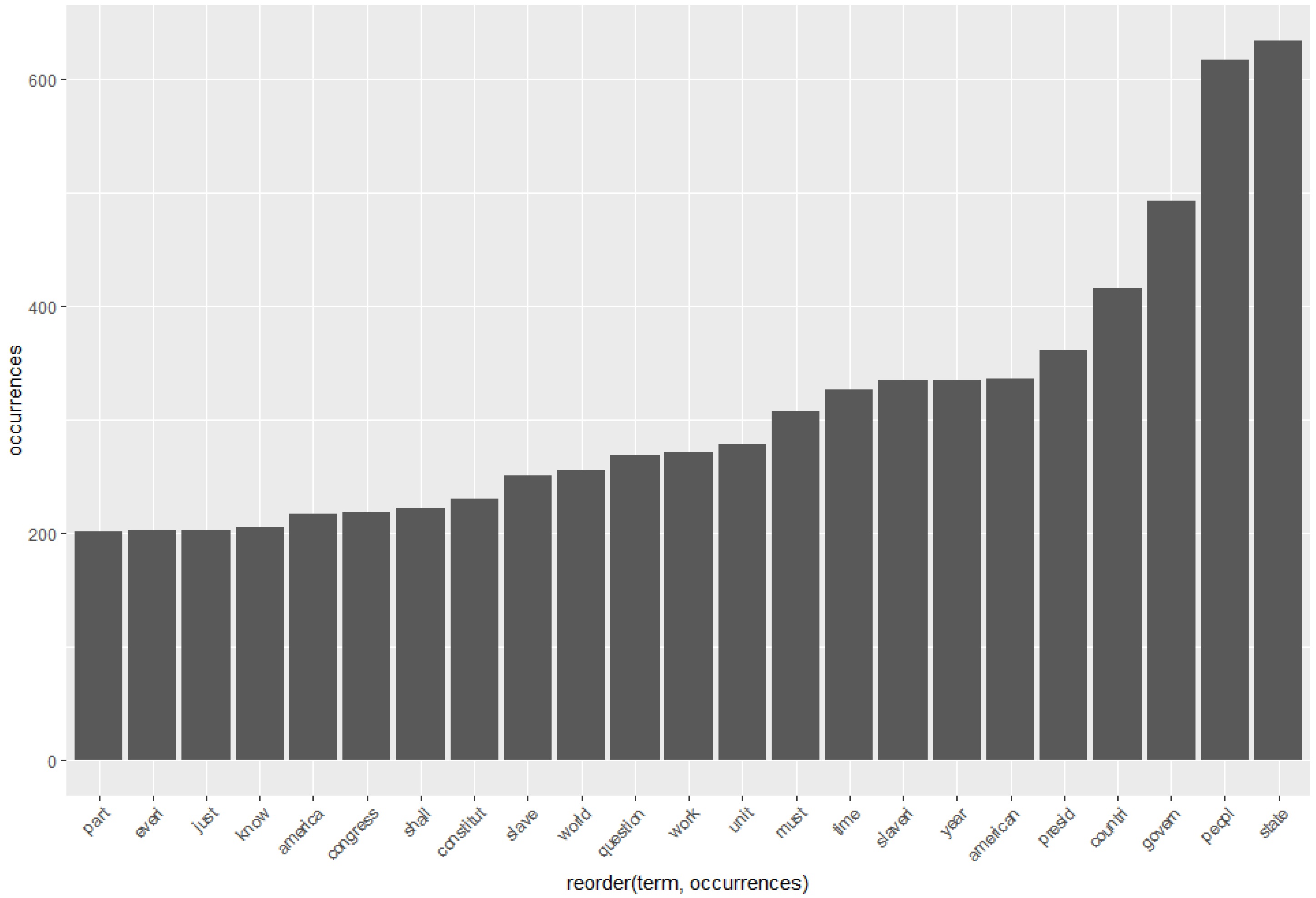
> Create a wordcloud

> Do terms associations on various words.

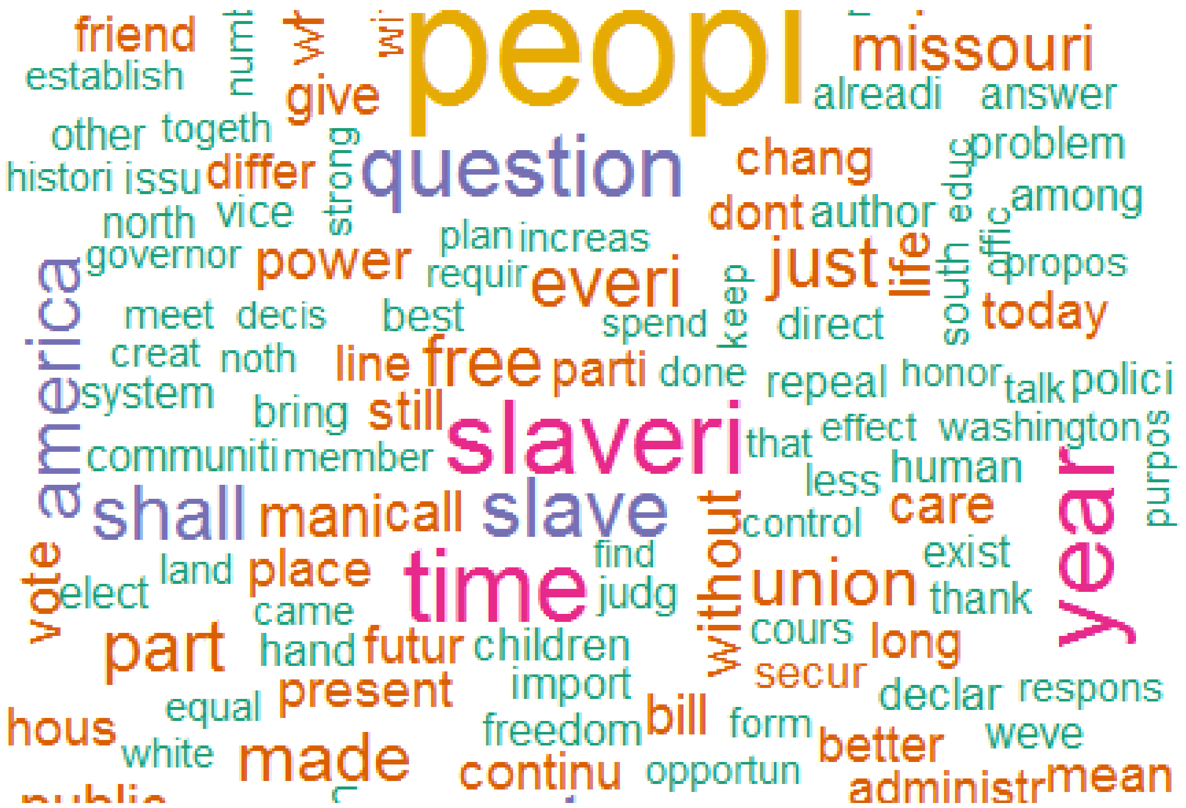
> Compare the total number of terms to that in Exercise 1.

Exercise file: **02-tm.R**

FREQUENCY HISTOGRAM



WORDCLOUD


friend
establish
give
peopl
missouri
alreadi
answer
other
togeth
histori
issu
differ
strong
question
chang
educ
problem
among
north
vice
dont
author
plan
increas
governor
power
requir
everi
keep
just
life
south
offic
propos
today
america
meet
decis
best
spend
direct
creat
noth
line
free
parti
done
repeal
honor
talk
polici
system
bring
still
slaveri
that
effect
washington
purpos
communiti
member
less
human
shall
manicall
slave
without
control
care
year
vote
elect
land
place
time
find
judg
exist
union
thank
came
cours
part
hand
futur
children
long
secur
present
import
declar
respons
hous
equal
bill
form
white
made
freedom
better
weve
continu
opportun
administr
mean

2. TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY

TF-IDF is a weighting method based on the following intuition:

terms that occur less often are more likely to be more descriptive of specific documents

> TF = number of times word occurs in a document. Need to normalize by doc wordcount

> > **TF = # of occurrences of word in doc** / **# of words in doc**

> IDF – weigh down frequent terms and scale up rare terms.

> > **IDF = Ln (# of docs / # of docs with word)**

> DocumentTermMatrix method has TF-IDF option

> See http://www.tfidf.com/ for an explanation of Tf-idf and look at tm documentation for details on how to get it working

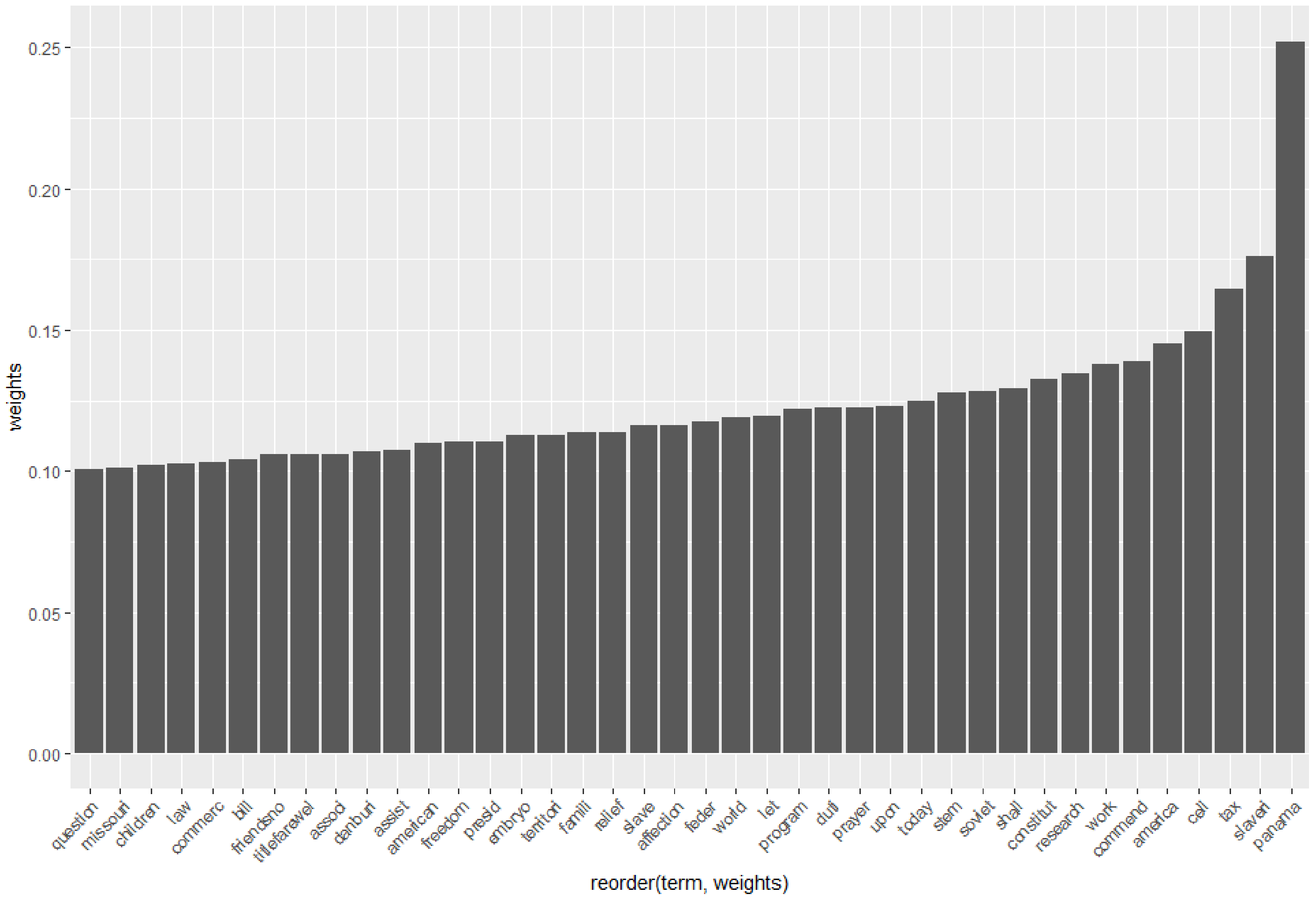
IN-CLASS EXERCISE

Using the same corpus you created in Exercise 1 (the first exercise)

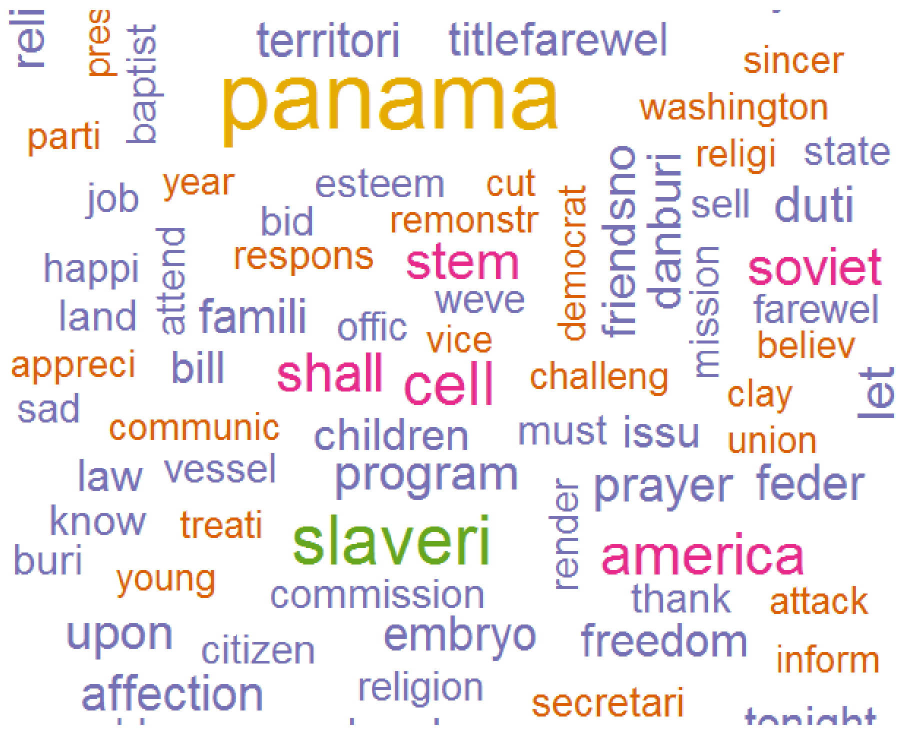
- Create a DTM that uses tf-idf weighting
  - have between 4 and 20 characters.
  - Occur in at least 3 and no more than 45 speeches.
- Create a histogram and wordcloud (setting appropriate cutoffs – examine the frequency table to see what these might be)
- Do terms associations on various words.

Exercise file: **03-tm.R**

WEIGHT HISTOGRAM



WEIGHT WORDCLOUD



NEXT STEPS

This is just a starting point. For more:

> Explore different parameter settings for tm functions.

> Explore other text mining packages. There's a zoo of packages, each with their a different design philosophy:

> ***tidytext -*** https://cran.r-project.org/web/packages/tidytext/vignettes/tidytext.html

> ***quanteda*** - https://quanteda.io/

> ***text2vec -*** https://cran.r-project.org/web/packages/text2vec/vignettes/text-vectorization.html

> ***textmineR***- https://cran.r-project.org/web/packages/textmineR/vignettes/a_start_here.html

> Play with other corpuses and text mining tools – see http://libguides.usc.edu/textmining/tools for possibilities.

Clustering

AN INTRODUCTION TO CLUSTERING

Given a set of objects, the basic idea is to create subsets (clusters) that contain similar objects.

Objects in a particular cluster are:

> Similar to each other

> Different from objects in other clusters

Obviously, how clusters formed depends on the criteria used for clustering. We'll look at 2 algorithms:

> Hierarchical clustering

> K-means clustering

There are many, many more. Google them, explore them.

Clustering is an *unsupervised* learning method. The unsupervised / supervised distinction covered later.

AN INTRODUCTION TO CLUSTERING

Many (but not all!) clustering algorithms are *distance-based*.

> Once we define a distance, the problem of clustering boils down to finding objects that are close to each other

So, how do we define a distance between documents?

> A DTM essentially describes a “document space”

> The dimension of this space = number of distinct words

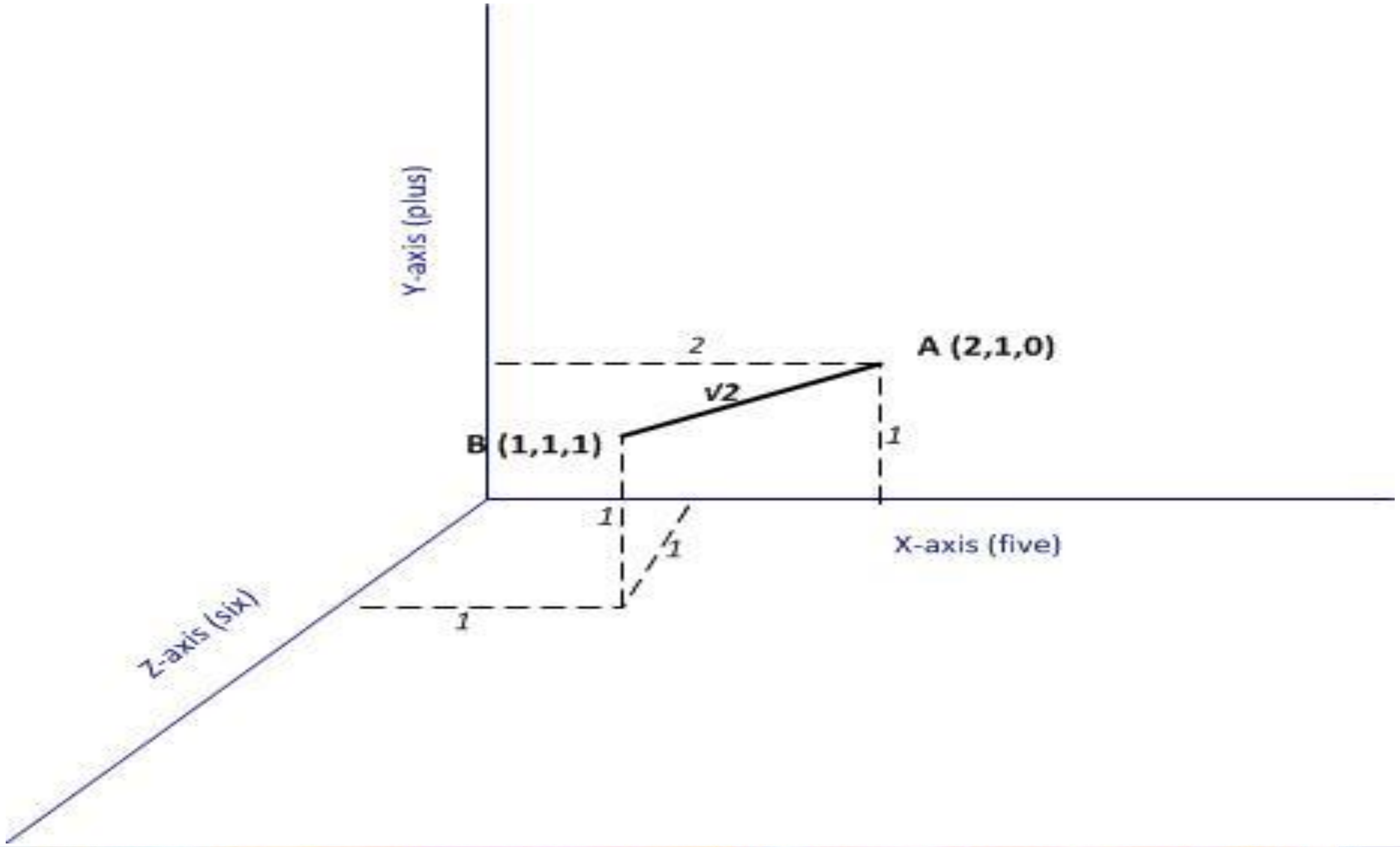
Let’s look at a simple example consisting of 2 documents

> **Document A**: “five plus five”

> **Document B**: “five plus six”

EUCLIDEAN DISTANCE

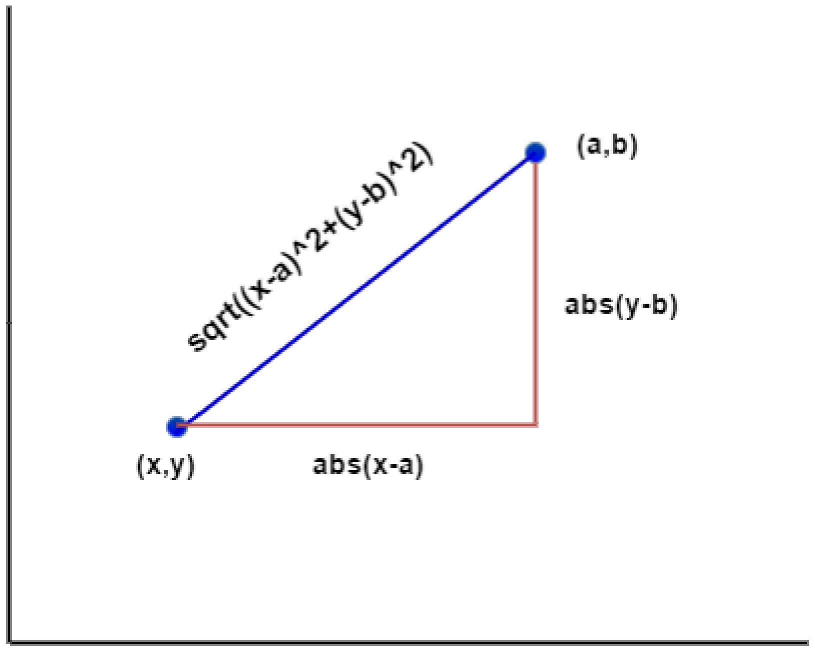
These two documents can be represented as points A and B in a 3 dimensional space that has the words “*five*” “*plus*” and “six” as the three coordinate axes



EUCLIDEAN VS MANHATTAN DISTANCE

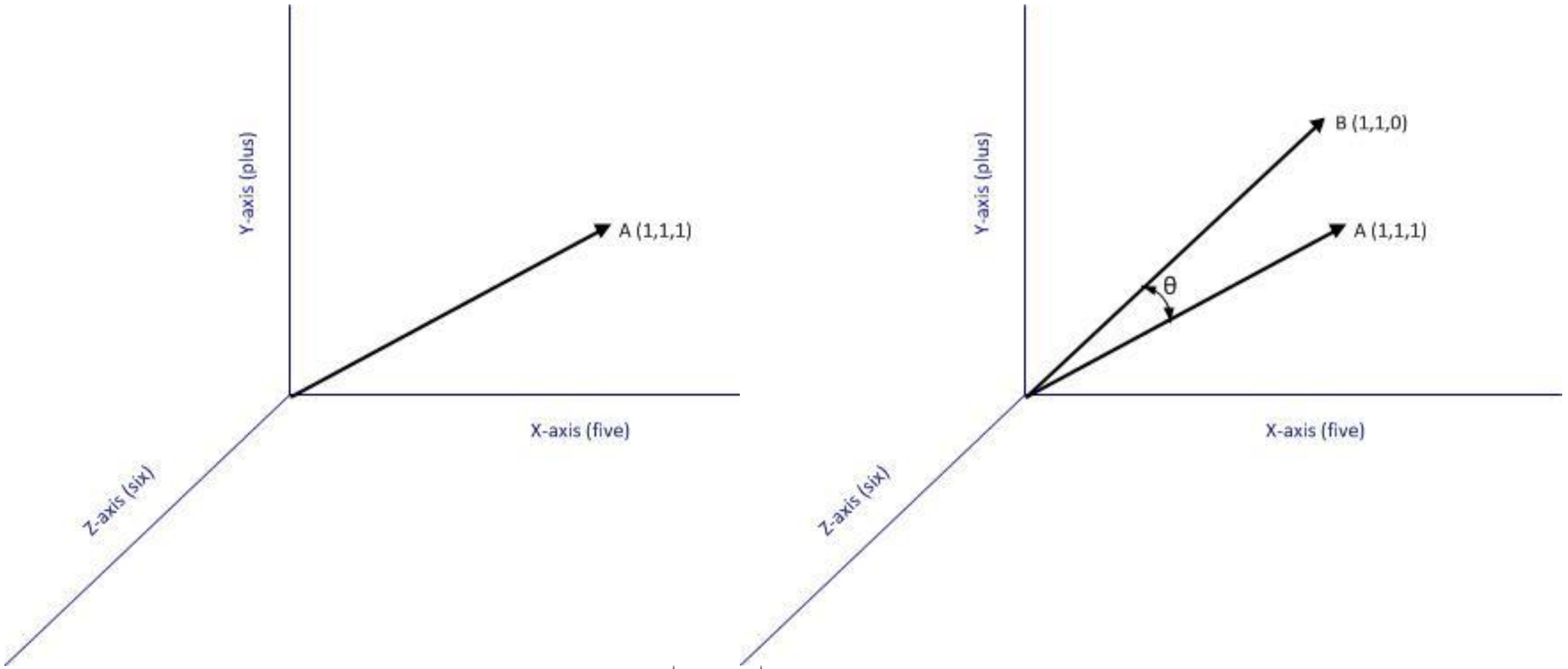
Euclidean distance = **sqrt((x-a)^2 + (y-b)^2)** *(“As the crow flies” distance*)

Manhattan distance = **|x-a| + |y-b|** (*Block distance*)



COSINE DISTANCE

Cosine **similarity** = dot product between vectors representing two datapoints



$$\cos(\theta) = \frac{x_1x_2 + y_1y_2 + z_1z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2}\sqrt{x_2^2 + y_2^2 + z_2^2}} \ldots (1)$$

Cosine distance = 1 – Cos(*θ*)

HIERARCHICAL CLUSTERING

Two approaches:

> **Agglomerative** – start with each object in its own cluster and build clusters starting with objects that are closest

> **Divisive** – start with all objects in one cluster and split clusters recursively until each document is in its own cluster

> Clusters are then determined via comparing the distances at which merges or splits occur. Widely separated merges / splits are indicative of a cluster.

We'll use the *hclust* algorithm which uses the agglomerative approach. Works as follows:

1. Assign each document to its own (single member) cluster
2. Find the pair of clusters that are closest to each other and merge them.
3. Compute distances between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until you have a single cluster containing all documents.

HIERARCHICAL CLUSTERING

A note on clustering mechanism

How do we decide to merge clusters? This is defined by the *linkage criteria* which is determined by the *linkage function* which computes the distance between clusters as a function of the observations within any two pairs of clusters.

Some possible linkage functions:

> The *single* linkage function will merge two clusters based on the closest two elements within both clusters

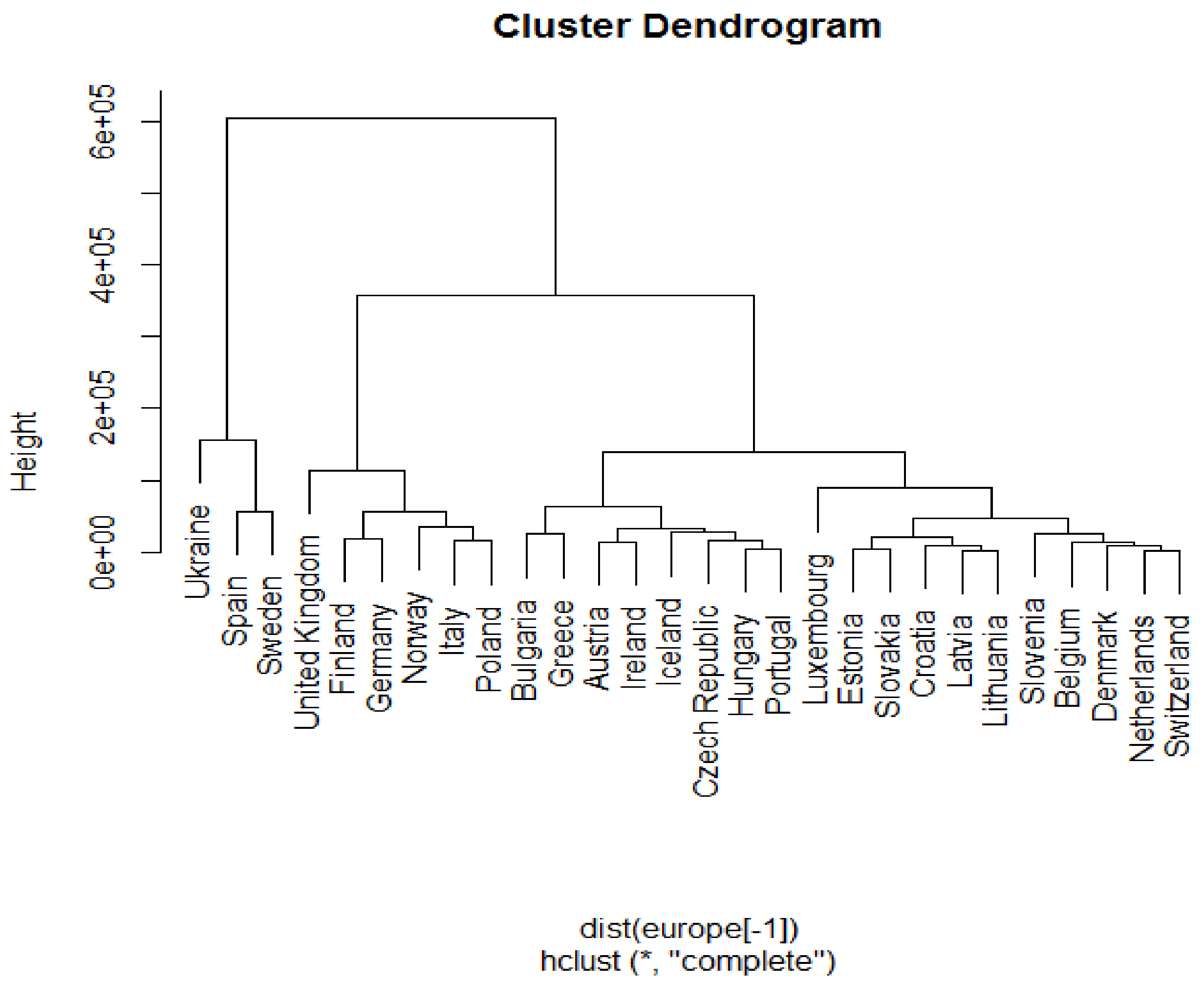
> The *complete* linkage function will merge two clusters based on the farthest two elements within both clusters.

> The ward linkage function is known as “Ward’s minimum variance method”, and works by minimizing the total within-cluster variance. The pair of clusters with the smallest cluster distance are then merged. (Generally a **good go-to**)

> There are also average, weighted, centroid, median

VISUALISING HIERARCHICAL CLUSTERS

▪



EXERCISE

Using the same corpus as before:

> Compute a Euclidean distance matrix from the DTM

Hierarchical Clustering

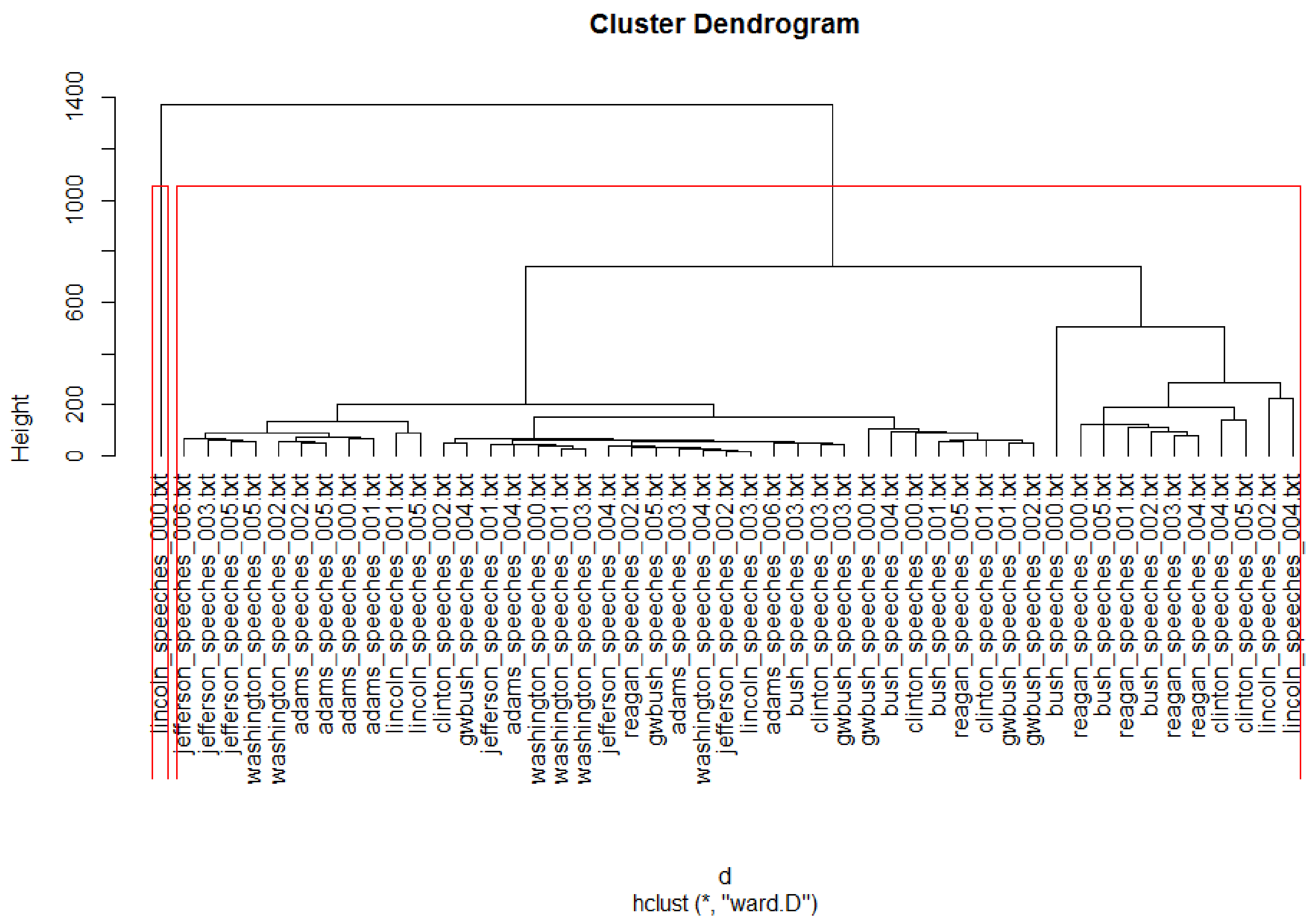
> Run hierarchical clustering algorithm hclust using Ward’s method

> Plot the results in a dendrogram and interpret them

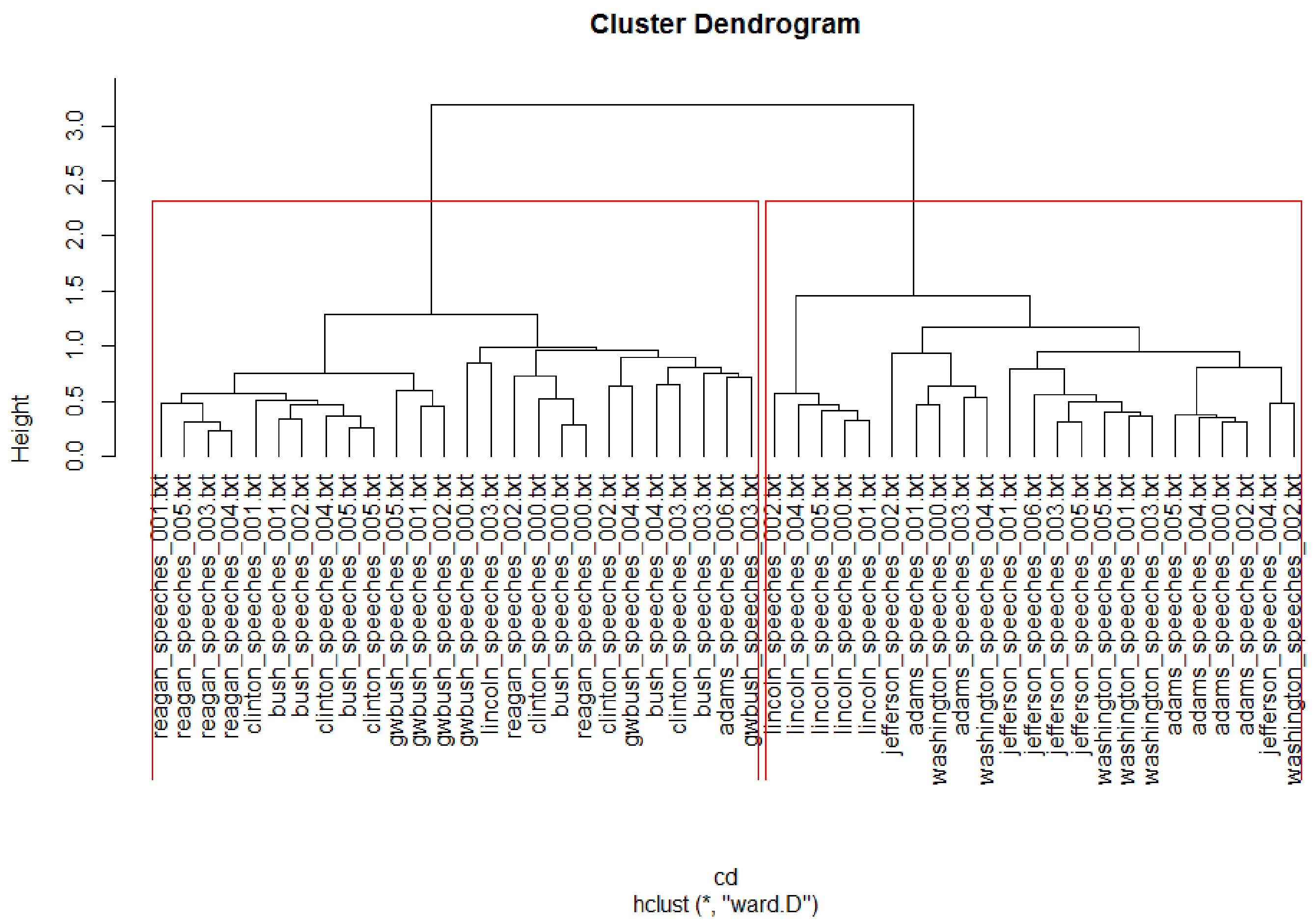
> Re-run using cosine distance

Exercise file: **04 - hierarchical clustering.R**

HIERARCHICAL CLUSTERING – EUCLID



HIERARCHICAL CLUSTERING – COSINE



K-MEANS CLUSTERING

Generates a specified number (K) of clusters centred around the average (Mean) of each cluster. We'll use the *kmeans* algorithm.

Works as follows:

1. Assign the documents randomly to k bins
2. Compute the location of the centroid (geometric center) of each bin.
3. Compute the distance between each document and each centroid
4. Assign each document to the bin corresponding to the centroid closest to it.
5. Stop if no document is moved to a new bin, else go to step 2.

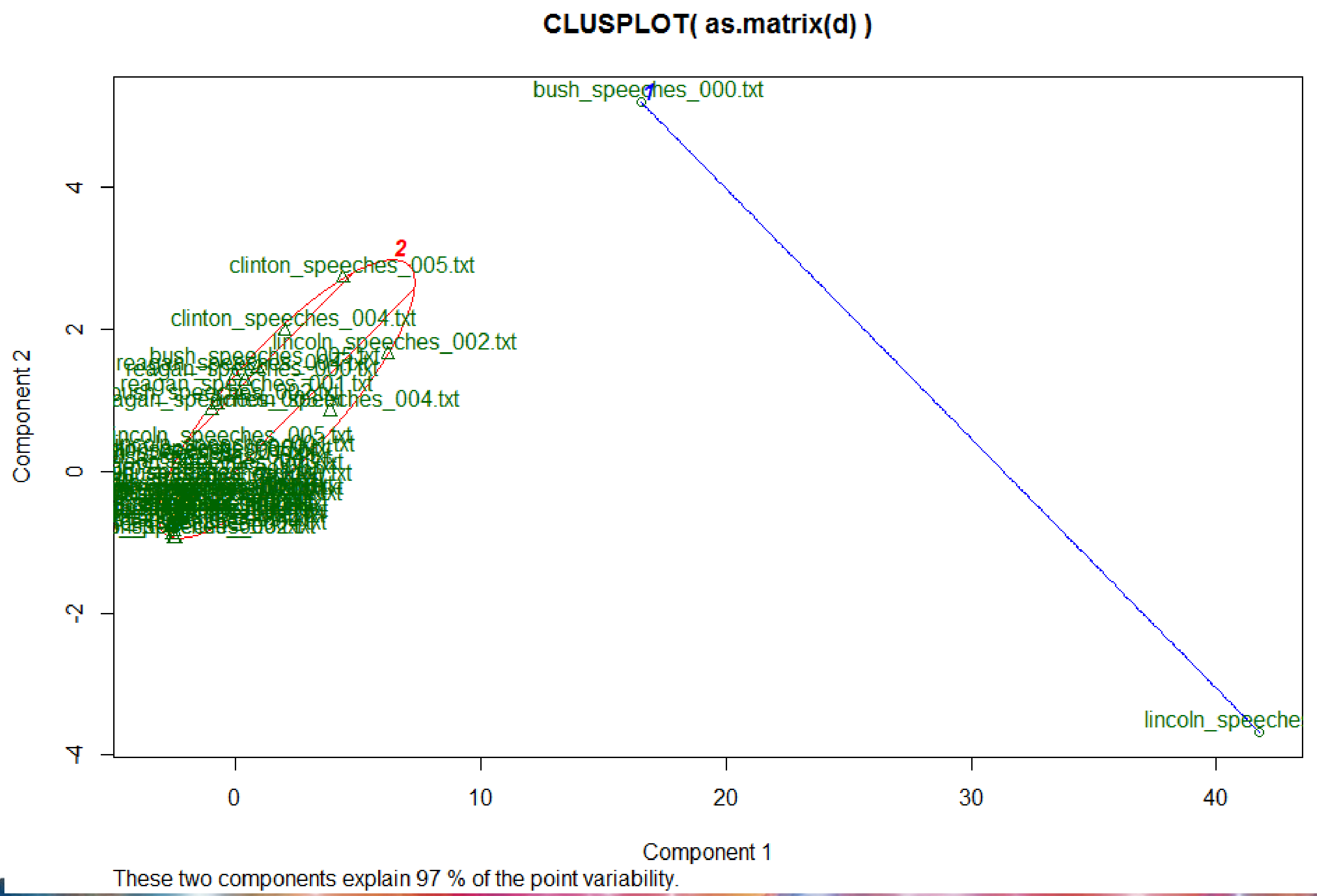
Limitations

> Local optimum – so try many different starting configs (controlled by *nstart* parameter)

> Difficult to figure out what K should be.

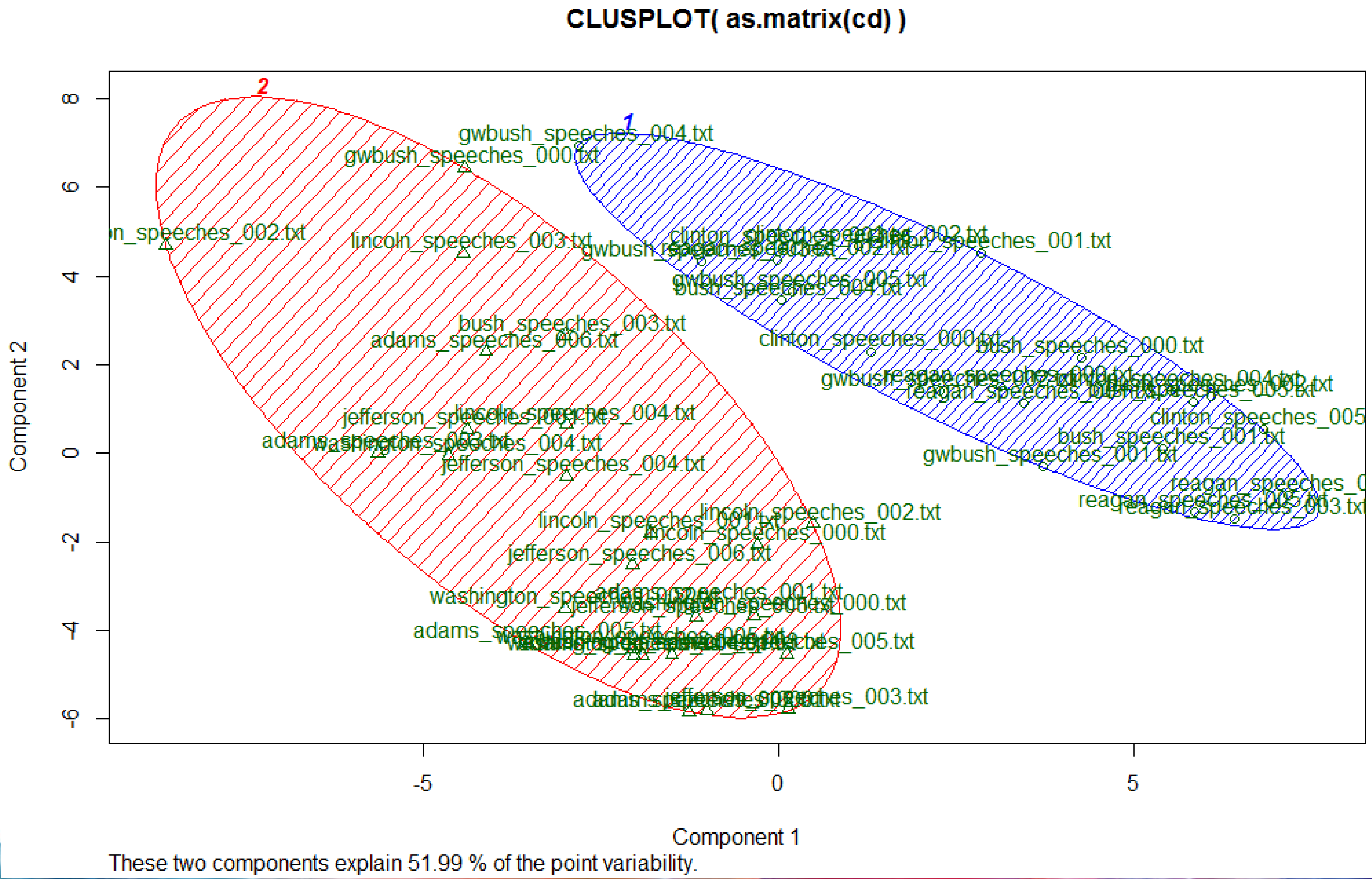
VISUALIZING KMEANS CLUSTERS - EUCLID

Many dimensions, so hard to visualize. The clusplot function projects along plane of maximum variability (PCA)



VISUALIZING KMEANS CLUSTERS - COSINE

Many dimensions, so hard to visualize. The clusplot function projects along plane of maximum variability



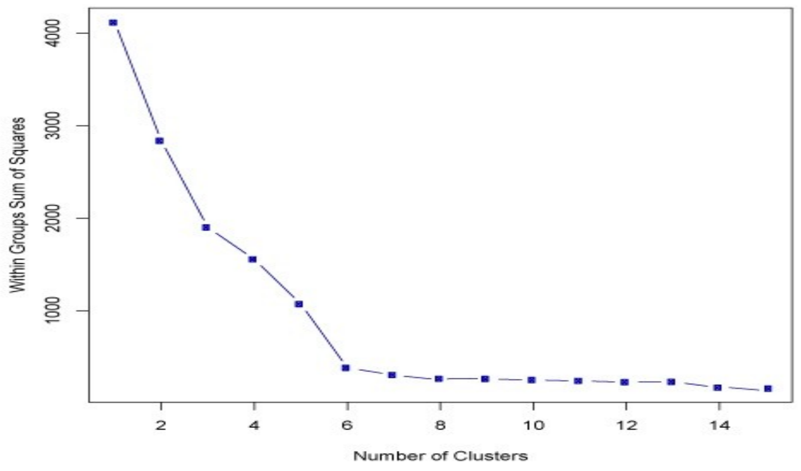
CHOOSING K

No fail safe method to choose K. The elbow method is a heuristic approach based on the following intuition:

> The quantity that is minimised by kmeans is the total of the within-cluster sum of squares (WSS) between each point and the mean.

> One expects that this quantity will be maximum when k=1 and then will decrease as k increases, sharply at first and then less sharply as k reaches its optimal value.

> There should be an "elbow" in the plot of k vs WSS.



CHOOSING K PT.2 (K-MEANS++)

The traditional K-means algorithm can result in bad clusterings based on poor choice of initial centroids.

> K-means++ places initial centroids far from each other to avoid the issue of poor initial centroid choice.

> Additionally the algorithm is configured to run 10 times with different centroid seeds and the best cluster inertia is recorded.

Note: Within cluster sum-of-squares (SSW) is often referred to as ’Cluster Inertia’

EXERCISE

Using the same corpus as before:

> Compute a (Euclidean) distance matrix from the DTM

K-means Clustering

> Run kmeans clustering using the default option

> Plot clusters in two dimensions using clusplot

> Use the elbow method to find an optimal value of k

Use the following packages:

> Cluster

Exercise File: **05 - kmeans clustering.R**

SOME THINGS TO THINK EXPERIMENT WITH

Some things to try

> Try different values of nstart and k, Use clusplot to plot the results. Interpret them,

> Find the optimal value of k using the elbow method. If there is no obvious optimum, make a reasonable guess.

> Do you have any ideas on how we can do better? (*Hint*: think about the potential effect of very long posts on distance measures)

  > Repeat exercise using cosine distance

Ref: https://eight2late.wordpress.com/2015/07/22/a-gentle-introduction-to-cluster-analysis-using-r/

SOME GENERAL POINTS ABOUT CLUSTERING

Unsupervised learning method

> Predefined category labels are not required.

> In contrast, most of the other algorithms you'll learn in DAM are *supervised* learning algorithms.

Very general.

> Can be applied to pretty much any kind of data (providing you can define a clustering criterion)

Clustering criteria are not necessarily distance-based.

> Our next algorithm is a good example….

ANALYSING CLUSTER ‘QUALITY’ - CROSSTABS

We will discuss later metrics to analyse classification tasks, but how do we say that the clustering was ‘good’?

> Depends on the business case. Remember that this is *unsupervised*

> Cross-tabulation allows for validation of the cluster labels aligning with the known class labels. Good clustering would see each column and each row distinctly separated.

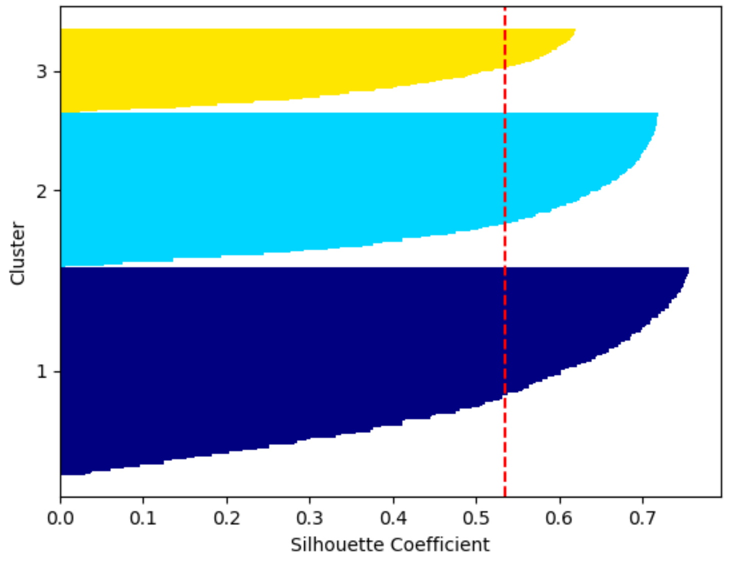
> Let us say that ’good’ would being able to separate ‘high value’ and ‘low value’ customers into distinct clusters. You have 2 clusters and 100 samples. The following crosstab would be a ‘perfect’ clustering.

	Value	
	High	Low
Cluster 1	50	0
Cluster 2	0	50

ANALYSING CLUSTER ‘QUALITY’ – SILHOUETTE COEFFICIENT

Silhouette plots provide the opportunity for visual and numerical analysis

> Visually, the width (y-axis length) of each coloured spike indicates the size of the cluster. For a balanced dataset, good clustering would have these as roughly equal in width. In contrast, in an analysis in which the dataset is unbalanced, good clustering would have clusters of differing sizes.



ANALYSING CLUSTER 'QUALITY' – SILHOUETTE COEFFICIENT

Silhouette plots provide the opportunity for visual and numerical analysis

> Numerically, silhouette plots also provide the silhouette coefficient. This metric is on a scale of [-1,1] and provides a numeric way to analyse the how close each point in one cluster is to points in the neighbouring clusters. Negative silhouette coefficient scores are interpreted as 'the observations within the cluster are probably in the wrong cluster'. Values closer to 1 indicate better clustering. A guide to interpreting different levels of this coefficient is provided in (source), page 88.

Silhouette Coefficient	**Interpretation**
0.71 - 1.00	A strong structure has been found
0.51-0.70	A reasonable structure has been found
0.26-0.5	The structure is weak and could be artificial; please try additional methods on this dataset
<= 0.25	No substantial structure has been found

CLUSTERING: DISCUSSION

Can you think of any areas (Perhaps in your own workplace) where clustering would be useful?

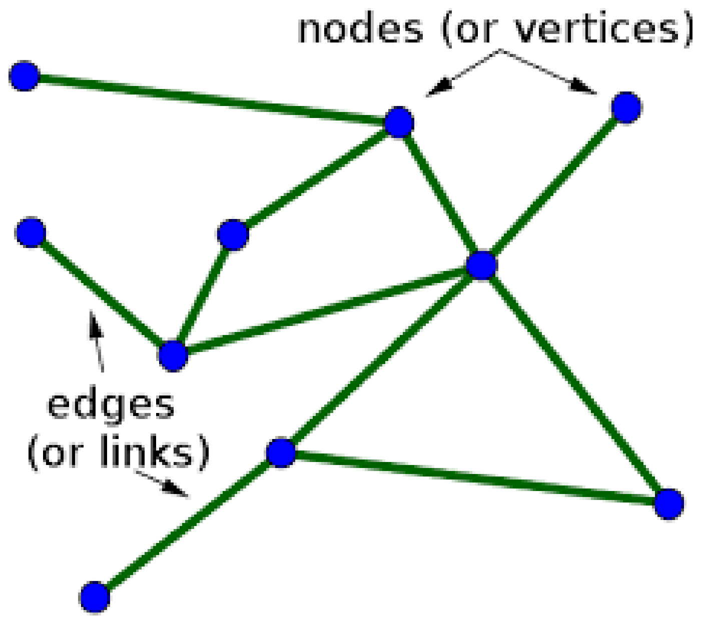
Network Graphs

WHAT IS A GRAPH?

A collection of connected objects:

> Objects are called **vertices (nodes)**

> Connections called **edges**

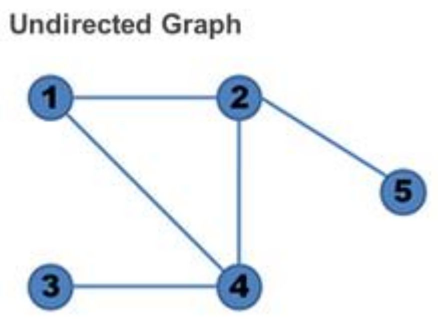


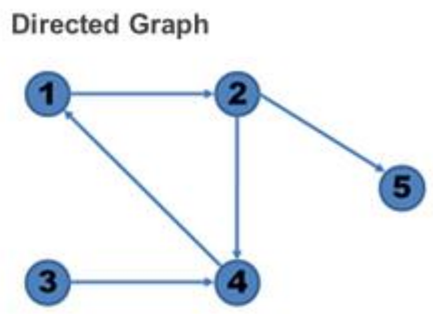
UNDIRECTED AND DIRECTED GRAPHS

Graphs can be undirected or directed

> Undirected: road networks between cities, friend networks on social media

> Directed: electrical circuits,





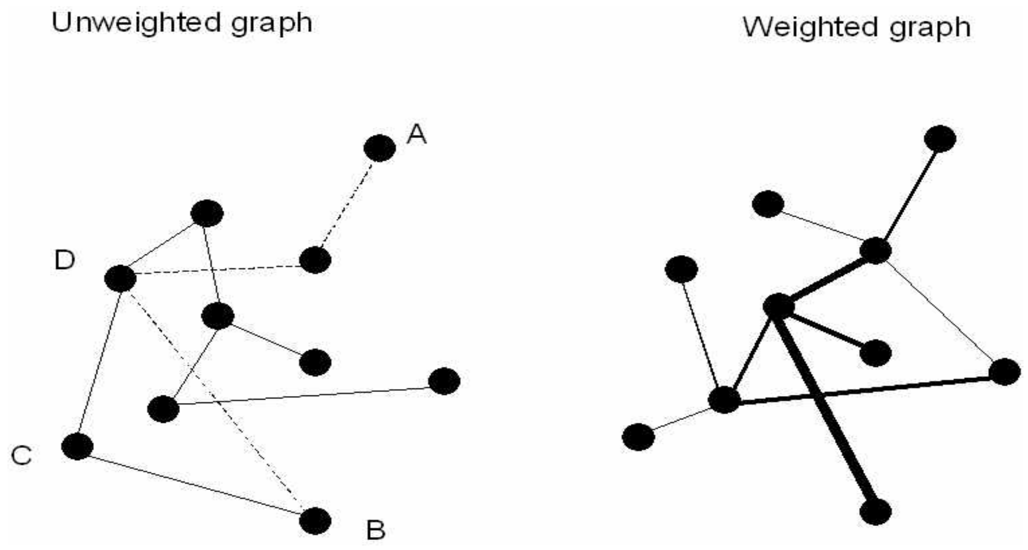
WEIGHTED AND UNWEIGHTED GRAPHS

Graphs (edges) can be unweighted or weighted

> Unweighted – where the connection between two vertices can take on only one of two values, 0 (no connection) or 1(connection)

> Weighted – where there is a numerical weight associated with each connection. Normally this can take on any value from a minimum to a maximum.

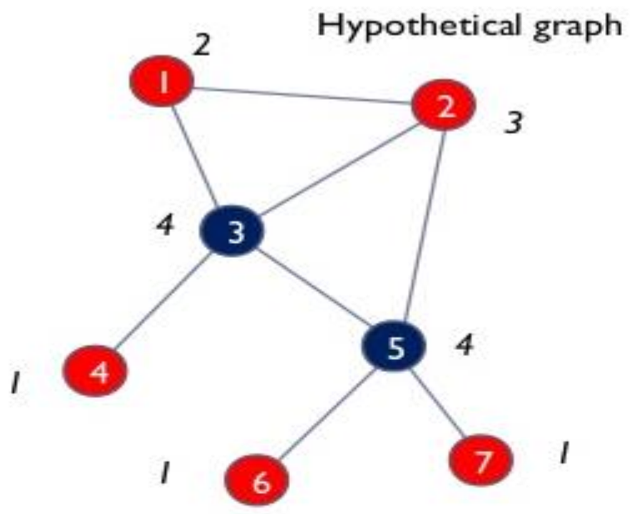
> Edges can also have a ‘rank’ or category



CENTRALITY

Measure of importance of a vertex

> **Degree centrality** – number of edges connected to vertex. Measures the "relatedness" of a vertex.



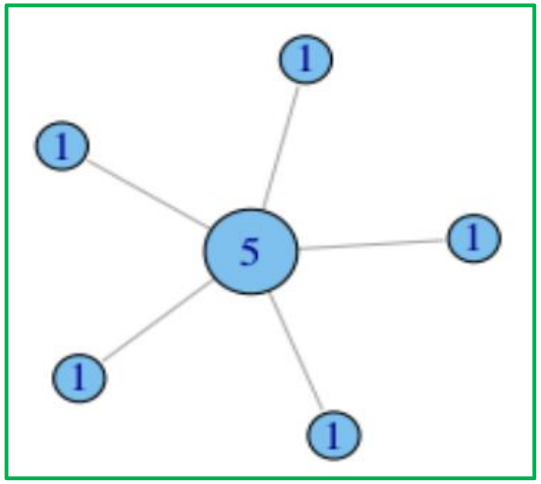
CENTRALITY

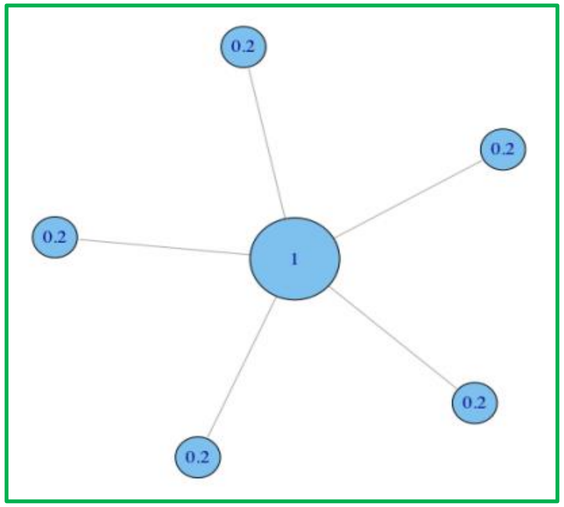
Degree Centrality - Normalised

> We can also normalise the degree centrality of nodes to allow for comparison between larger and smaller graphs.

> $C_D(i) = \frac{d(i)}{N-1}$

> The degree centrality of a node is the degree (number of edges connected to it) divided by the total number of edges.



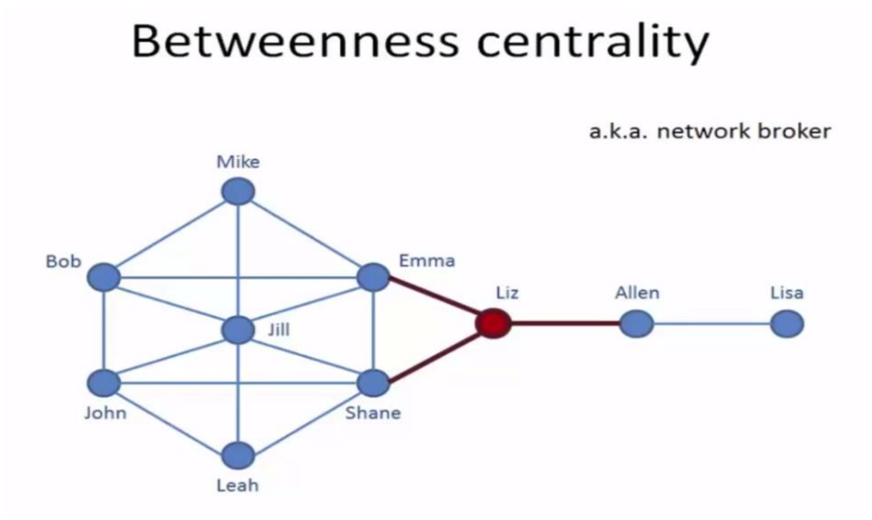


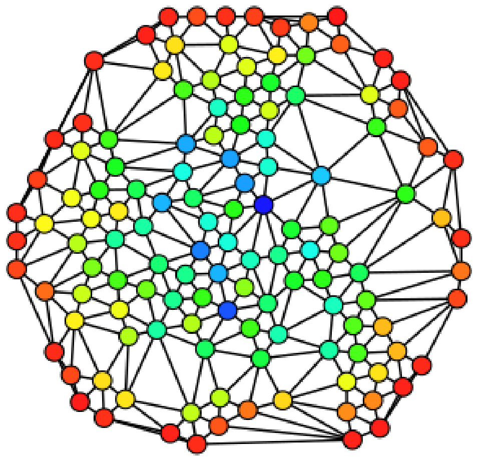
CENTRALITY

Measure of importance of a vertex

> **Betweenness centrality** – number of shortest paths between all pairs of vertices (excluding the one under consideration) that pass through a vertex. Measures overall **connectivity** of a vertex.

> In the example to the right, the nodes are coloured from red (least) to blue (most) betweenness centrality





CENTRALITY

Betweenness Centrality – calculations

The **formula** for betweenness centrality is:

$$C_B(i) = \sum_{j<k} \frac{g_{jk}(i)}{g_{jk}}$$

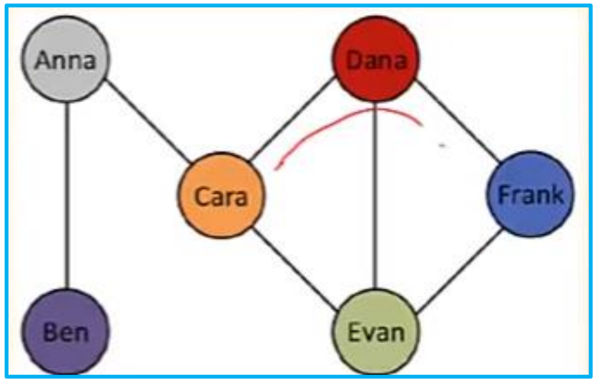
- Where g_{jk} is the number of shortest paths connecting jk and $g_{jk}(i)$ is the number that node i is on. Noting that I,j,k are different nodes.
- The formula is usually normalised (the red 2 is for undirected graphs):
  - $C'_B(i) = \frac{C_B(i)}{\frac{(n-1)(n-2)}{2}}$
- So to calculate for some node (i)
  - Find all the shortest paths between all nodes.
  - For each shortest path that (i) lies on (not start/end) it receives 1 unit.
    - BUT this is only allocated pro-rata for the proportion of shortest paths between nodes that (i) is on.

CENTRALITY

Betweenness Centrality – An example

The **formula** for betweenness centrality is:

$$C_B(i) = \sum_{j<k} \frac{g_{jk}(i)}{g_{jk}}$$



- Let us calculate the betweenness centrality of DANA (red)
- **Find all the shortest paths between all nodes**
  - AC, AB, AE, BA, AC, CE, EF and others all don’t matter as their shortest paths don’t contain Dana.
  - Consider AF. The shortest path length is 3 (ACDF or ACEF).
  - Consider CF. The shortest path length is 2 (CDF or CEF).
  - Consider BF. The shortest path length is 4 (BACDF or BACEF)
- **Allocate 1 point (pro-rata) for each shortest path that Dana lies on**
  - For AF, Dana lies on ½ of these so gets 0.5 points
  - For CF, Dana lies on ½ of these so gets 0.5 points
  - For BF, Dana lies on ½ of these so gets 0.5 points
- **Dana has a betweenness centrality of 1.5 (Which we could then normalise)**

CENTRALITY

Measure of importance of a vertex

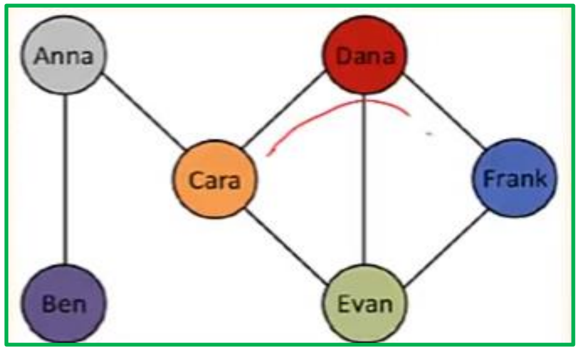
- **Closeness centrality** – The normalised closenesss centrality represents the average length of the shortest paths between that node and all other nodes.
- Whereas betweenness can be thought of as how important a vertex is at bridging between others, Closeness is more about how 'close' this node is to all other nodes.
- The formulae (Regular and normalised) are as follows:
  - $$\boxed{C_c(i) = \frac{1}{\sum_j d(i,j)}} \quad \boxed{C_c(i) = \frac{N-1}{\sum_j d(i,j)}}$$
  - Therefore we:
    - Find all the shortest path lengths in the graph from node (i) to all other nodes.
    - Inverse
    - Optional: normalise by multiply by n-1

CENTRALITY

Closeness Centrality – An example

The **formula** for closeness centrality is:

$$C_c(i) = \frac{N-1}{\sum_j d(i,j)}$$



- Let us calculate the closeness centrality of CARA (Orange)
- **Find all the shortest path lengths in the graph from node (i) to all other nodes.**
  - Shortest paths and associated lengths would be:
    - CA(1), CB(2), CD(1), CE(1), CF(2) = Total of (7)

- **Inverse**
  - 1/7

- **(Optional) Nomalise**
  - (n-1)/7 = 5/7 = **0.7142**

COMMUNITY DETECTION

A community is a well-connected subgraph (subset of nodes)

> **Modularity** – a measure of the relative denseness of intra and inter group connectivity.

Fast Greedy: Bottom-up, hierarchical approach

> Start with every vertex in a separate community

> Merge iteratively, such that each merge yields a maximum increase in modularity.

> Stop when not possible to increase modularity further.

Louvain method:

> Look for "small" communities by optimizing modularity locally (like fast greedy)

> Second, it aggregates nodes belonging to the same community and builds a new network whose nodes are the communities.

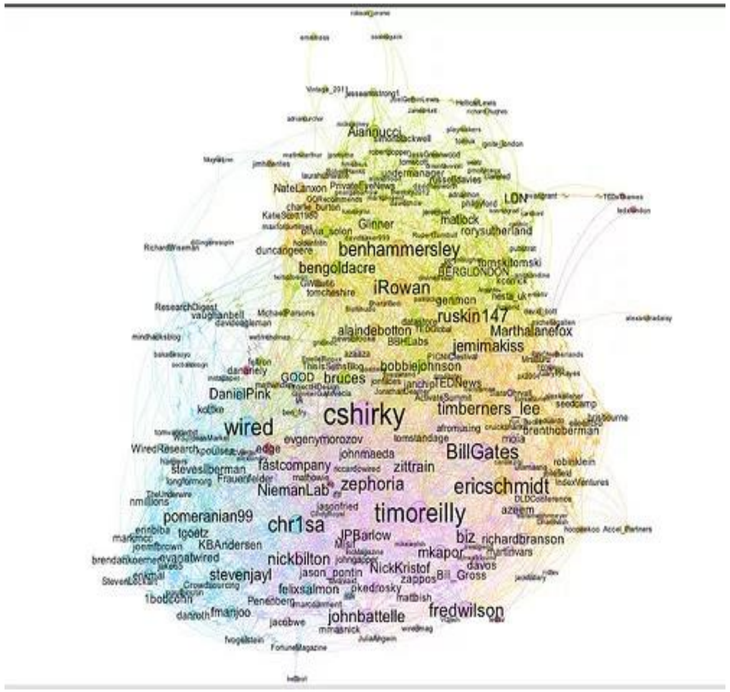
> Repeat until a maximum modularity is attained.

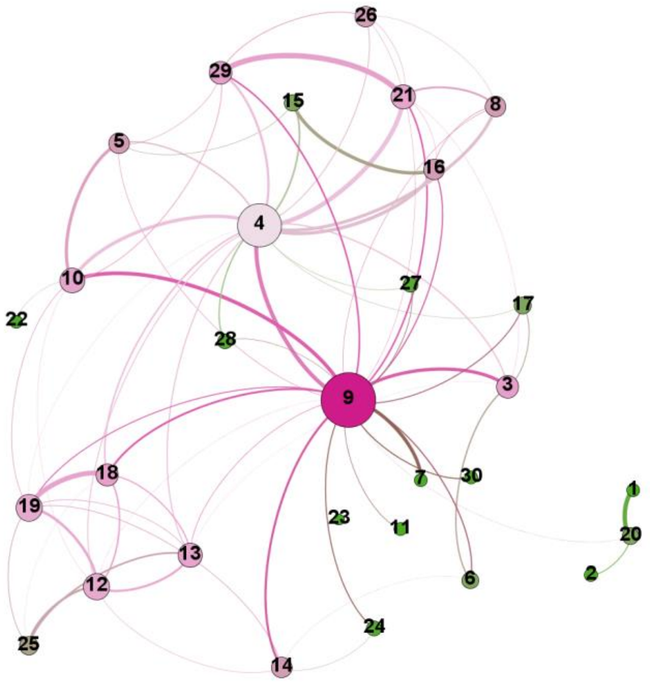
WHAT ARE THEY GOOD FOR?

An excellent way to visualise relationships between objects. Some examples:

> Social networks relationships (followers, friends, connections)

> Document relationships (similarity between documents in a corpus)



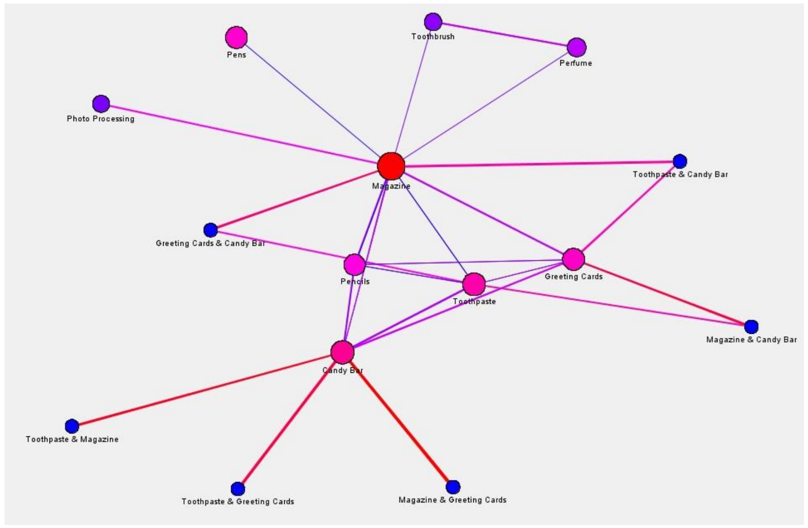


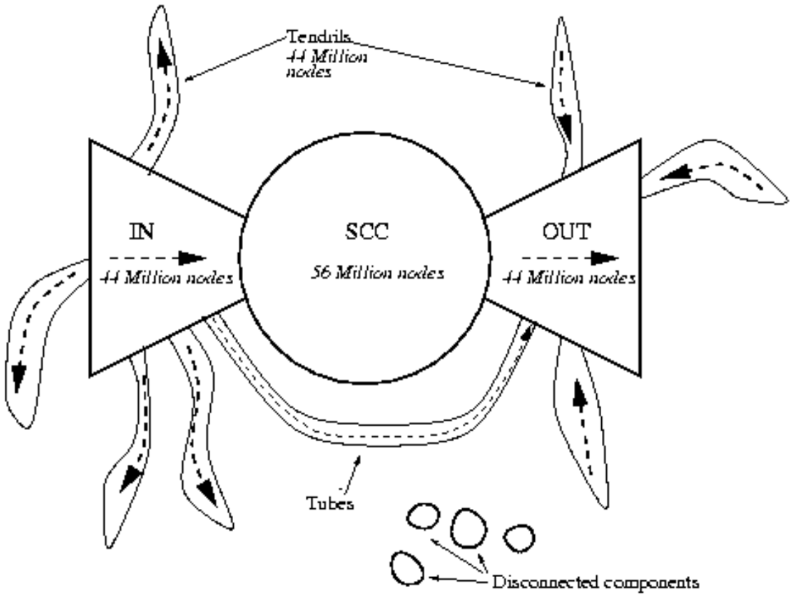
WHAT ARE THEY GOOD FOR?

Some examples:

> Websites (Links between websites is the backbone of SEO/Google Rankings). See here.

> E-commerce basket analysis

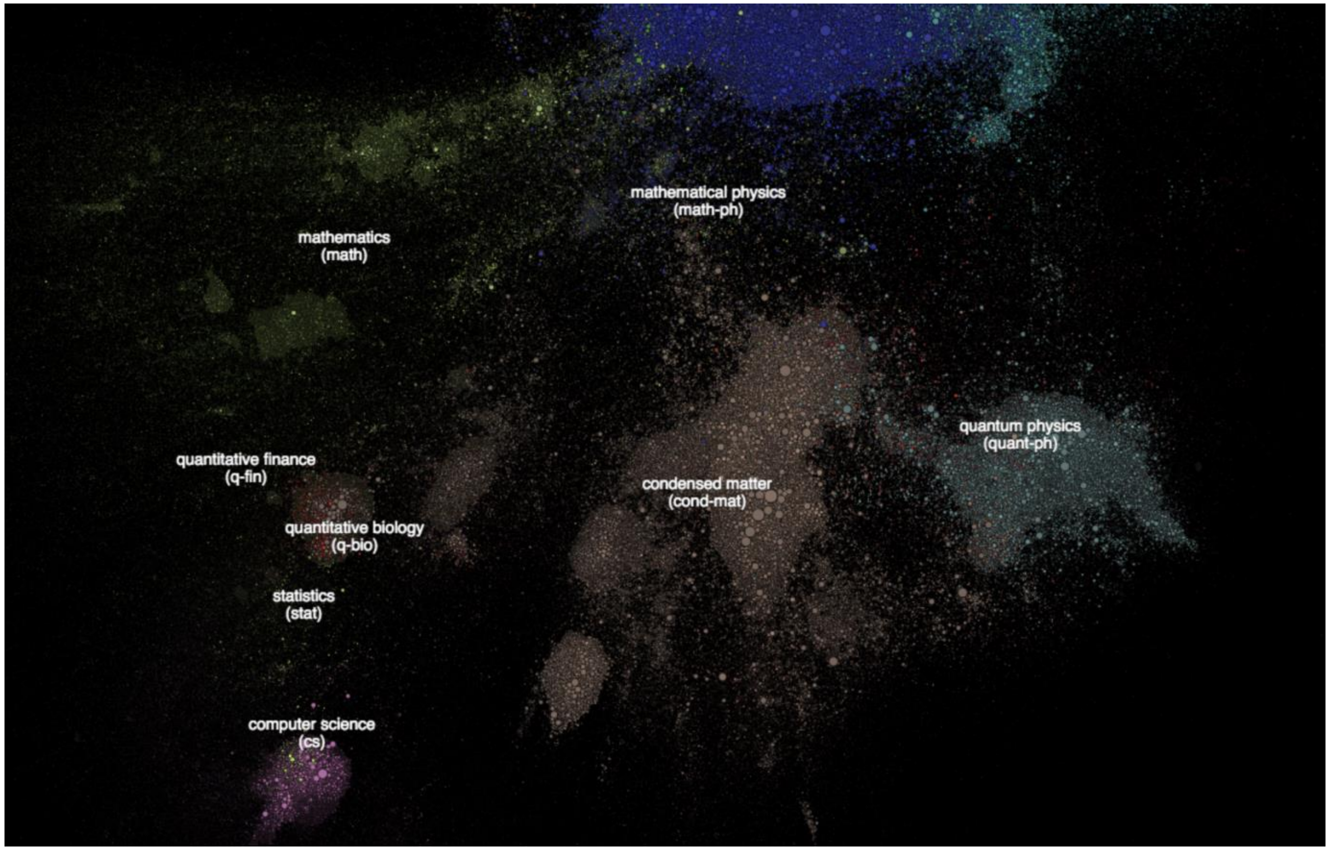




WHAT ARE THEY GOOD FOR?

Some examples:

> Academic discipline mapping (Citation graphs). [See]



ELECTIVE SPOTLIGHT

‘Social and Informational Network Analysis’

> Work through things like:

> > More granular definitions of edges, graph types (random, scale-free, small world), different representations of networks.

> > Calculating paths between nodes, analysing different groupings and sub-groupings in graphs.

> > More measures of properties and relationships between nodes, communities, cliques

> > Information cascade, diffusion and propagation through graphs

> > Recommendation engines

> > Information networks, Google and the WWW

NETWORK GRAPHS IN R

The best known network graphing library is **igraph**

http://igraph.org

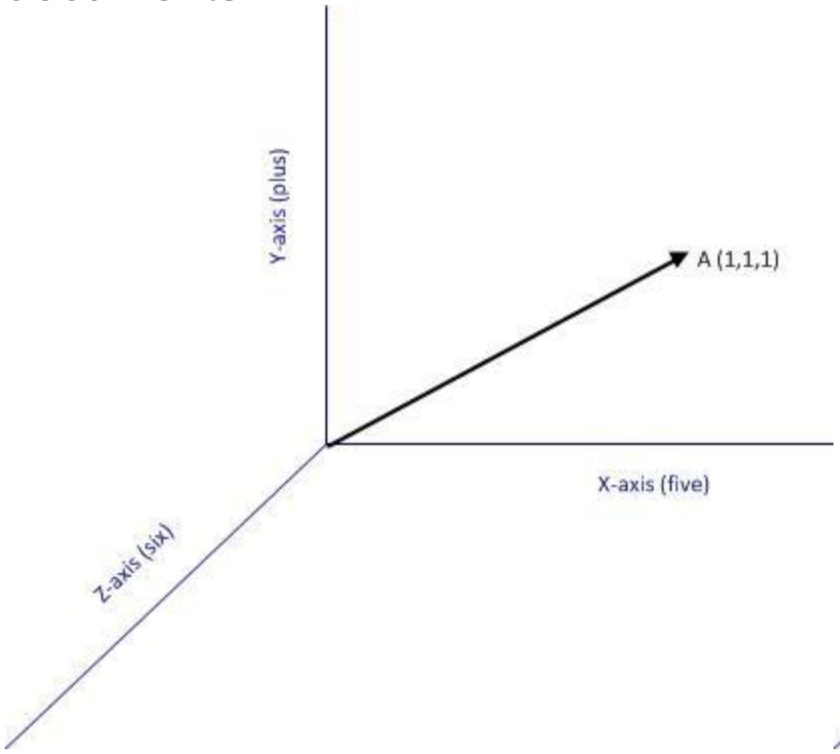
“*igraph is a collection of network analysis tools with the emphasis on **efficiency**, **portability** and ease of use. igraph is **open source** and free. igraph can be programmed in **R**, **Python** and **C/C++**.*”

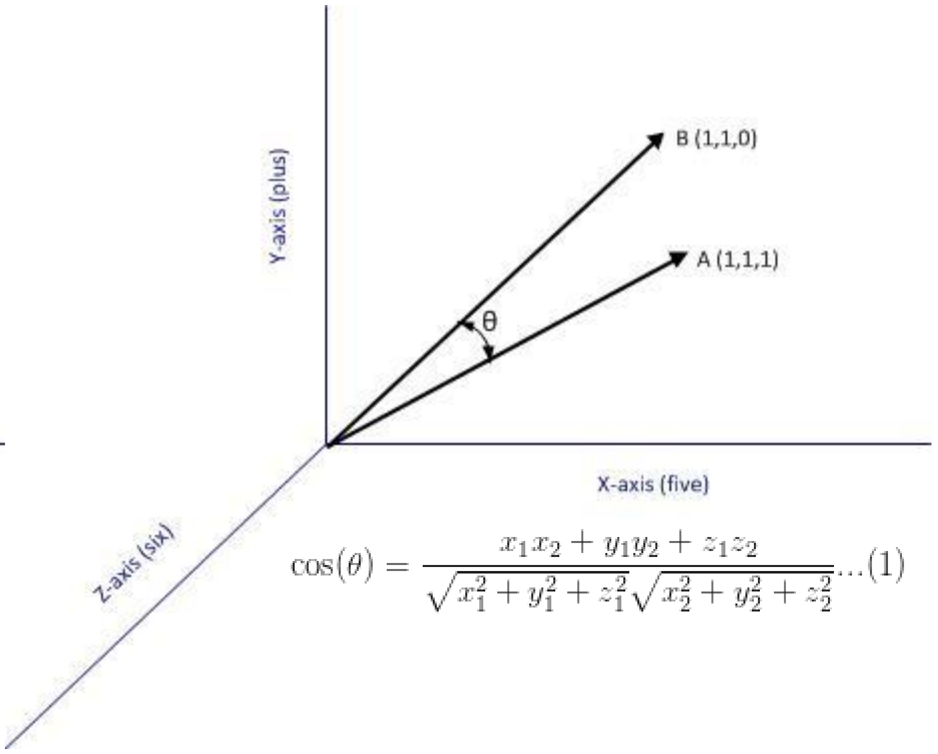
…very versatile, but as a consequence quite complicated and can take some time to learn.

We’ll do a simple exercise building on the work we’ve done so far today’s session…

COSINE SIMILARITY – A REMINDER

Cosine **similarity** = dot product between vectors representing two documents





$$\cos(\theta) = \frac{x_1x_2 + y_1y_2 + z_1z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2}\sqrt{x_2^2 + y_2^2 + z_2^2}} \ldots (1)$$

We can calculated cosine similarity between every pair of documents in our corpus. It lies between 0 and 1 (1=identical, 0=nothing in common)

VISUALISING DOCUMENT SIMILARITY

Based on the text corpus in docs-2.zip:

> Compute a **similarity matrix** from the DTM. In the world of network graphs, this is called the **adjacency matrix**.

> To simplify the graph, retain only similarities that lie between half the maximum and the maximum. Set all the others to zero (no relationship)

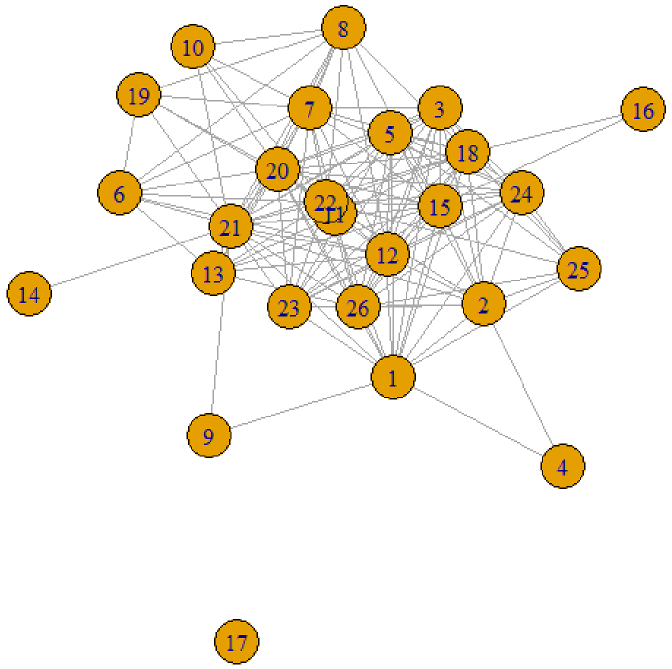
> Import the adjacency matrix into igraph and plot the graph

> Find communities (clusters) using a fast / greedy and Louvain methods

Exercise File: **06 – network_graphs.R**

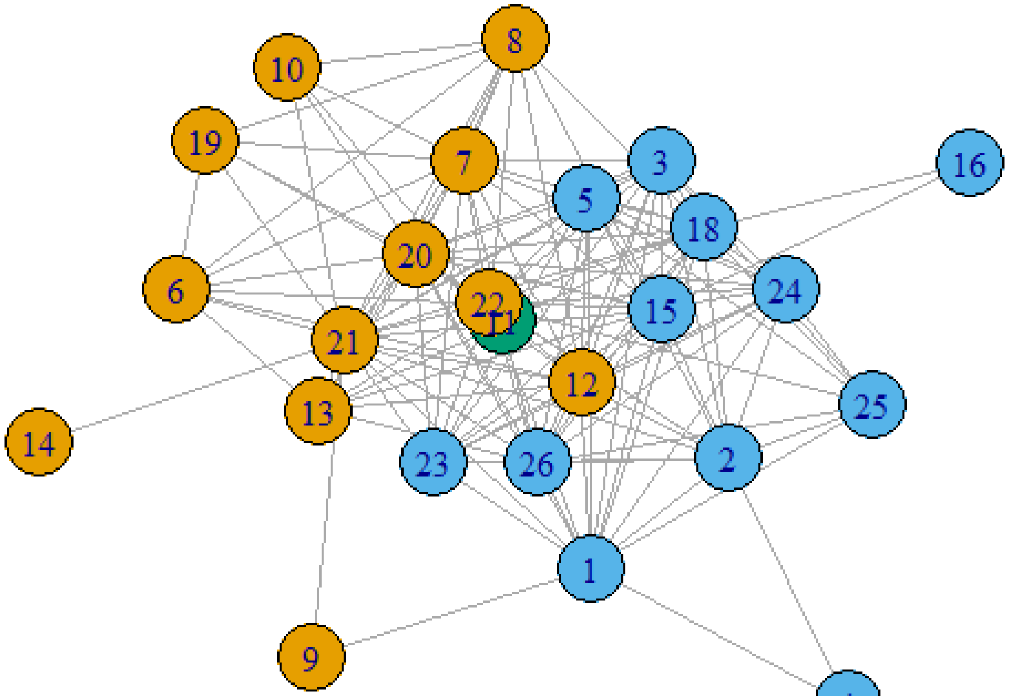
> Play around with different graph parameters (refer to the extensive igraph documentation a needed)

BASIC NETWORK GRAPH – KAWAI-KAMADA

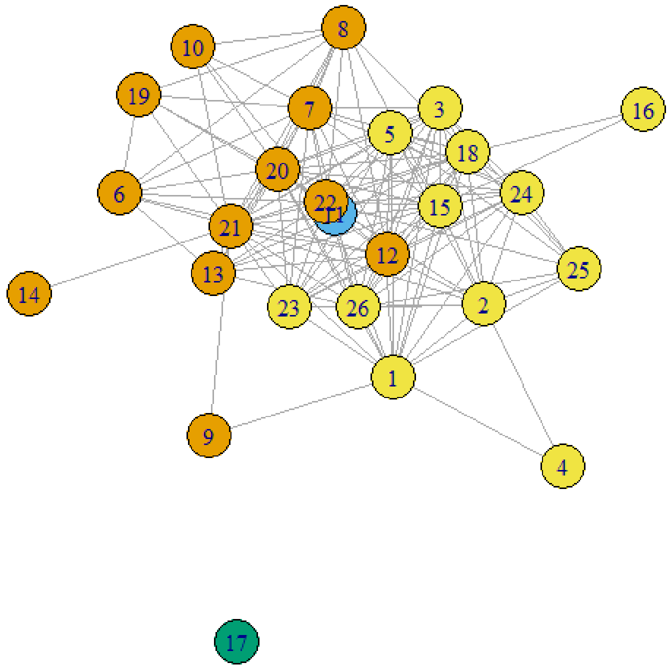


COMMUNITY DETECTION –FAST/GREEDY

You should get something like this…



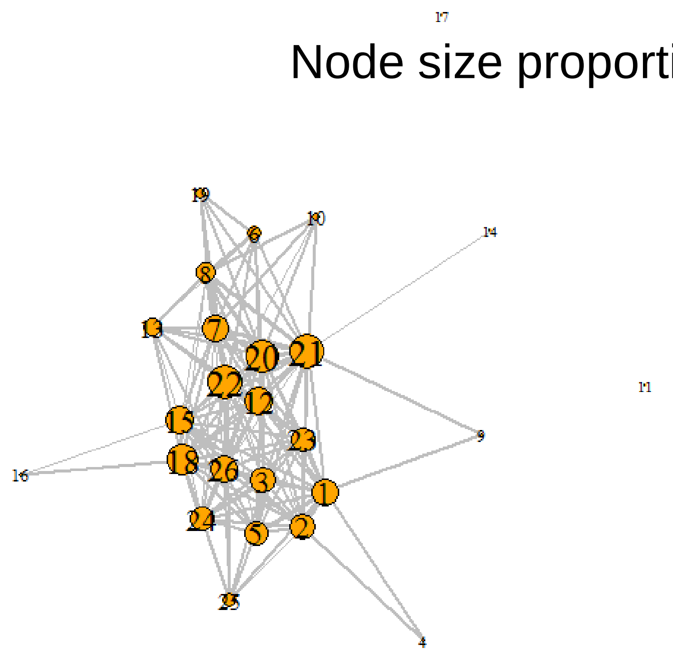
COMMUNITY DETECTION - LOUVAIN



EDGE AND NODE WEIGHTS VISUALISED

Edge weight proportional to similarity

Node size proportional to degree



A NOTE ON DOCUMENT REPRESENTATIONS

The methods we've seen so far deal with words as atomic entities. These result in sparse representations of documents

> Each word is encoded by its frequency of appearance in documents via a DTM (or TF-IDF) matrix.

> Conversely, each document is represented in terms of frequencies of words that appear in the document. A document is treated as a **bag of words** (BOW), with no notion of contiguity.

Limitation: There is no possibility of extracting **themes** or **meaning** via a purely BOW approach**.**

In particular, BOW approaches do not capture **synonymy** (e.g. kid and child) and **polysemy** (e.g. kid (child) and kid (lamb))

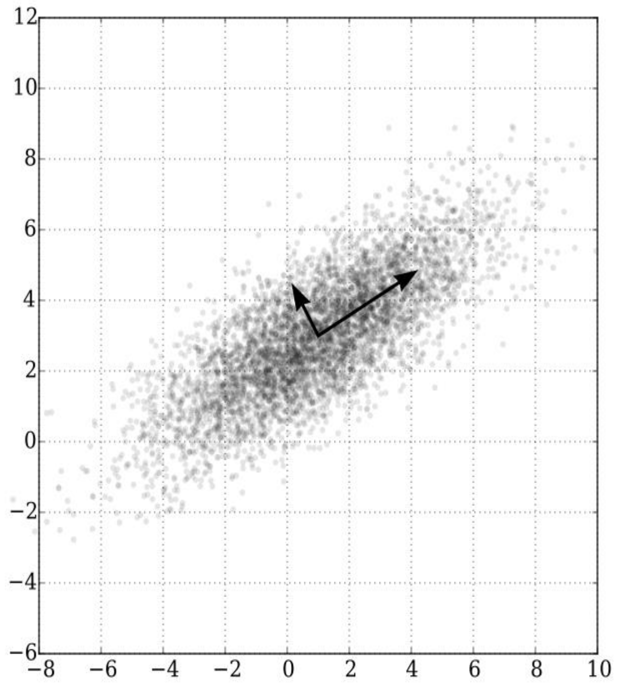
The methods we discuss next address some of these limitations.

Latent Semantic Analysis

DIMENSIONALITY REDUCTION REDUX

PCA

> Rotate the coordinate system so that the new “x axis” is aligned with the maximum variance in feature space. What does that mean?



- Turns out the new directions are aligned with the ***eigenvectors*** of the correlation matrix in original space.
- Eigenvectors are obtained by solving the matrix equation.

$$\mathbf{C}x = \lambda x$$

(the rotation *diagonalizes* C)

Where C is the correlation matrix, x the eigenvector and λ the eigenvalue.

- Eigenvectors are directions of *pure stretching* (since direction is preserved)
- Typically, a feature space with N dimensions will have N eigenvalues.
- The **eigenvector associated with the largest eigenvalue defines the first principal component.**

DIMENSIONALITY REDUCTION REDUX

Principal component decomposition can be carried out only on square (NxN) matrices.

In general, DTMs are ***not*** square! Typically:

# of terms >> # of docs

Is there a way to reduce dimensions in a similar way to PCA??

It turns out there is!

DIMENSIONALITY REDUCTION REDUX

A general theorem of linear algebra tells us that any matrix **C** can be decomposed into three components as follows:

A rotation - **U**

A stretch – **Σ** (a diagonal matrix)

Another rotation – **V**

$$\mathbf{C = U\Sigma V^T}$$

Where $\mathbf{V^T}$ is the transpose of **V**.

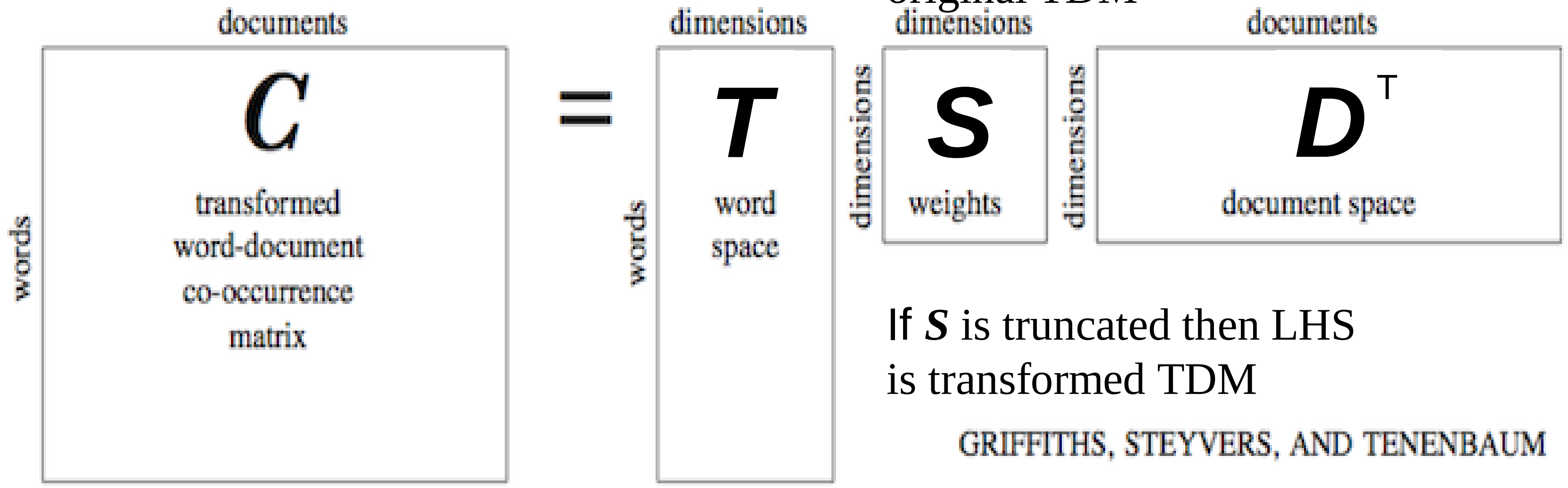
Called ***Singular Value Decomposition.***

The **diagonal entries of Σ are equivalent to eigenvalues in PCA**. They are called ***singular values*** rather than eigenvalues.

So, if C is a DTM (or TDM), we can **reduce its *effective* dimension by choosing to restrict our attention to the first k singular values** (k << # of terms), and zero out the others.

A GRAPHICAL REPRESENTATION

If S contains all singular values then LHS is original TDM



If ***S*** is truncated then LHS is transformed TDM

Note that the transformed matrix (after LSA) has the **same dimensionality as the original TDM**. However, every word in an original document is expressed in terms of a smaller number of “effective terms”.

If you compare a document vector in the original TDM with a document vector in the LSA space, you will see that the latter is much denser. We’ll see this in the exercise.

LATENT SEMANTIC ANALYSIS

The basic idea is as follows:

> Start with a TDM (terms in rows, documents in columns)

> Decompose TDM using SVD

> Reduce effective dimensionality of term space by choosing n highest singular values, where n << # of terms.

> Compute document vectors in this reduced space

> Use standard similarity measures (such as cosine similarity) to compute similarity between documents.

Notes:

> Dimensions do not have intuitive meanings because they no longer correspond to individual terms.

> Dimensionality reduction densifies vectors.

> Benefit is that it captures relationships between terms (reflected in the densification of vectors)

LATENT SEMANTIC ANALYSIS

Exercise (file: 07-lsa.R)

> Perform LSA on the small presidential speeches corpus (docs-1 folder)

> 1) Examine LS term vectors and compare them to those in TF space. What does this tell you about LSA?

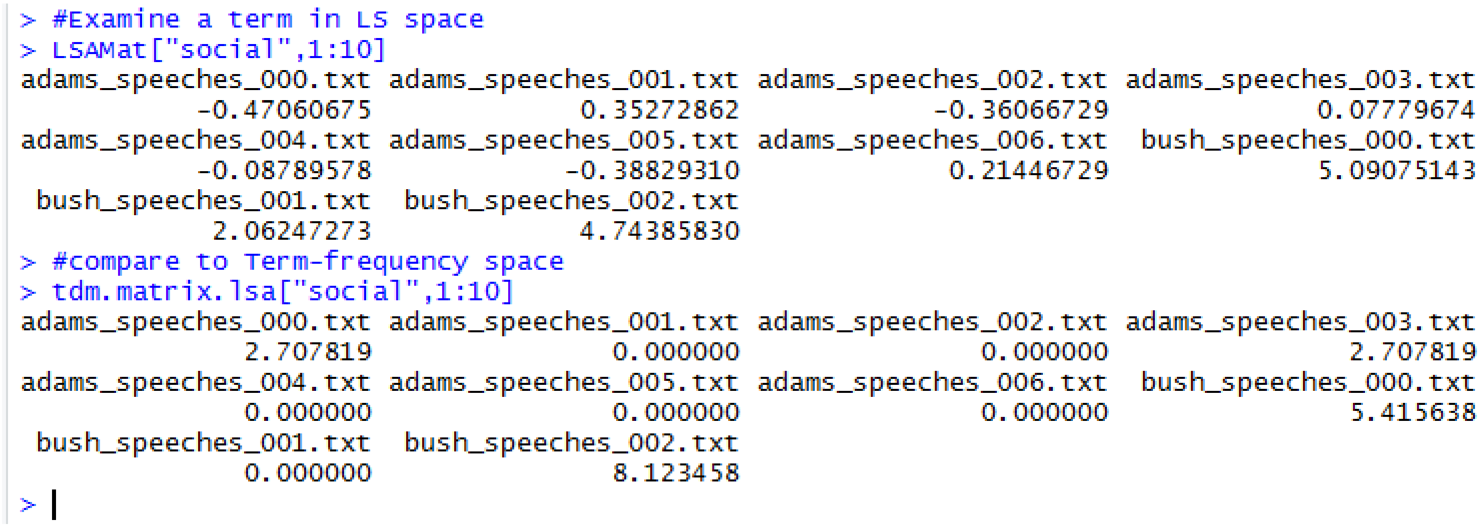
> 2) Examine document vectors in LS and TF space. Comment.

> 3) Calculate cosine similarity using the document vectors obtained. View the similarity matrix. Comment on your results.

> Exercise file: 07-lsa.R

LATENT SEMANTIC ANALYSIS

1) Comparison of terms in LS and TFIDF spaces



```
> #Examine a term in LS space
> LSAMat["social",1:10]
adams_speeches_000.txt adams_speeches_001.txt adams_speeches_002.txt adams_speeches_003.txt
           -0.47060675             0.35272862            -0.36066729             0.07779674
adams_speeches_004.txt adams_speeches_005.txt adams_speeches_006.txt  bush_speeches_000.txt
           -0.08789578            -0.38829310             0.21446729             5.09075143
 bush_speeches_001.txt  bush_speeches_002.txt
            2.06247273             4.74385830
> #compare to Term-frequency space
> tdm.matrix.lsa["social",1:10]
adams_speeches_000.txt adams_speeches_001.txt adams_speeches_002.txt adams_speeches_003.txt
              2.707819               0.000000               0.000000               2.707819
adams_speeches_004.txt adams_speeches_005.txt adams_speeches_006.txt  bush_speeches_000.txt
              0.000000               0.000000               0.000000               5.415638
 bush_speeches_001.txt  bush_speeches_002.txt
              0.000000               8.123458
> |
```

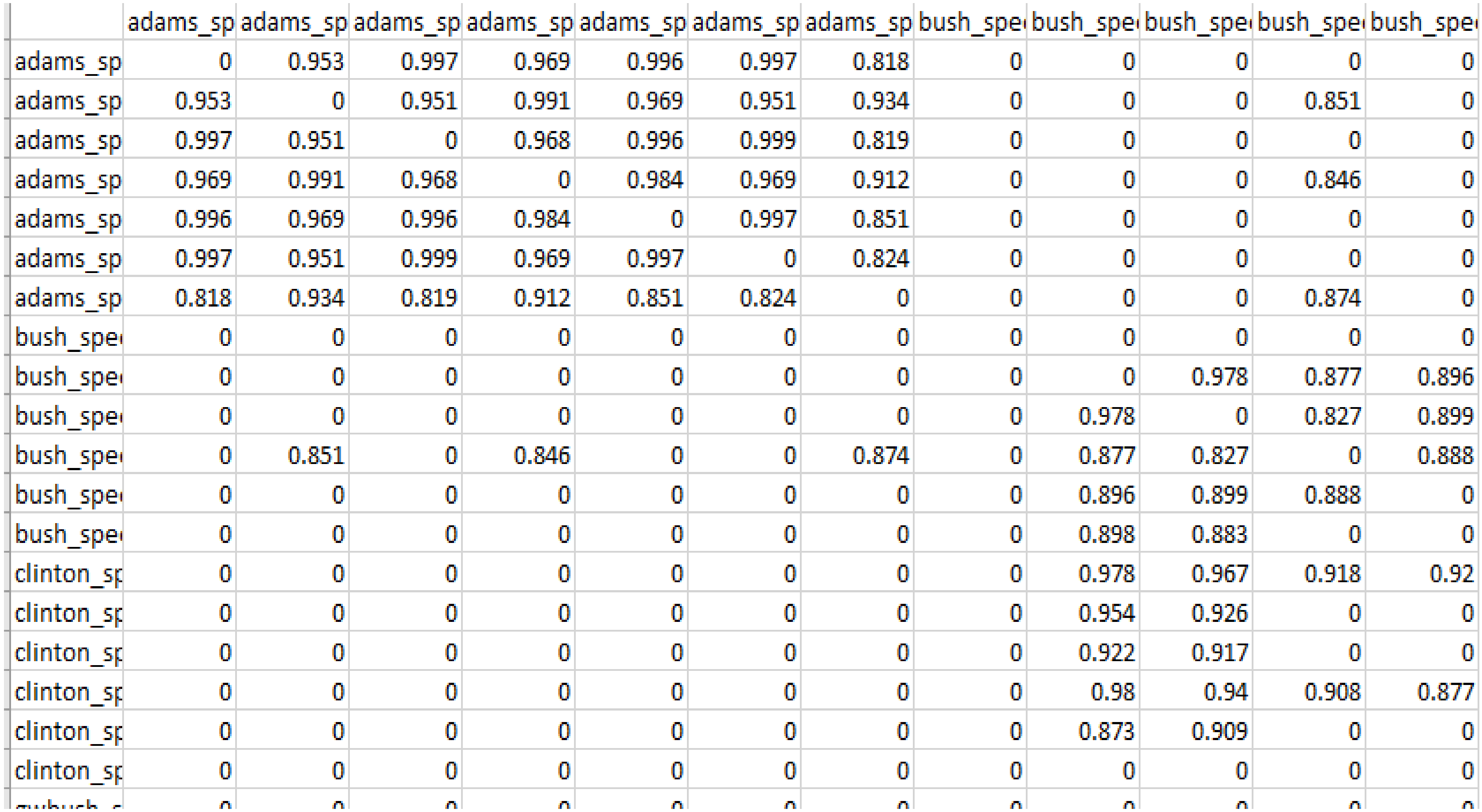
LATENT SEMANTIC ANALYSIS

2) Comparison of term vectors in LSA and TFIDF space

	X (LSA)		X (TFIDF)
abandon	1.131645	abandon	5.029747
abandoned	0.688049	abandoned	0
abandonment	0.388704	abandonment	0
abashed	0.165877	abashed	0
abatement	0.692273	abatement	0
abc	0.028799	abc	0
abide	1.326422	abide	0
abiding	-0.19423	abiding	0
abilities	1.190079	abilities	6.61471
ability	0.627718	ability	0
able	2.742677	able	2.292782
ablest	-0.07785	ablest	0
aboard	0.083131	aboard	0
abolish	0.22619	abolish	0
abolished	0.689818	abolished	0
abolition	0.363076	abolition	0
abolitionist	-0.1557	abolitionist	0
abolitionists	-0.21699	abolitionists	0
aboriginal	0.510162	aboriginal	0

LATENT SEMANTIC ANALYSIS

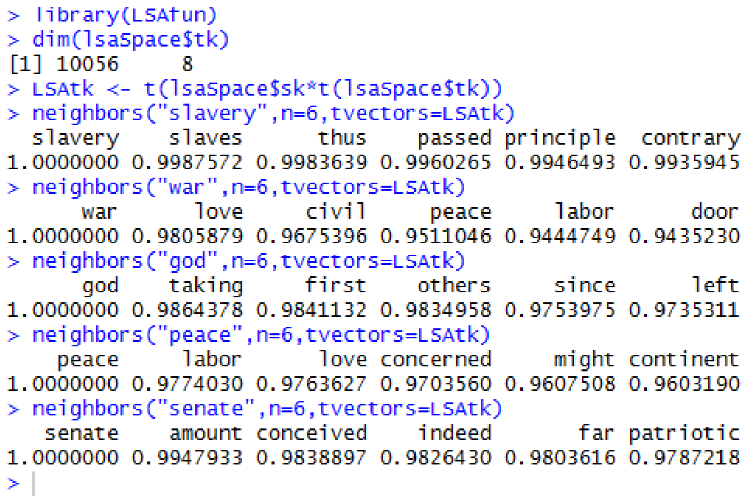
3) Truncated Similarity matrix



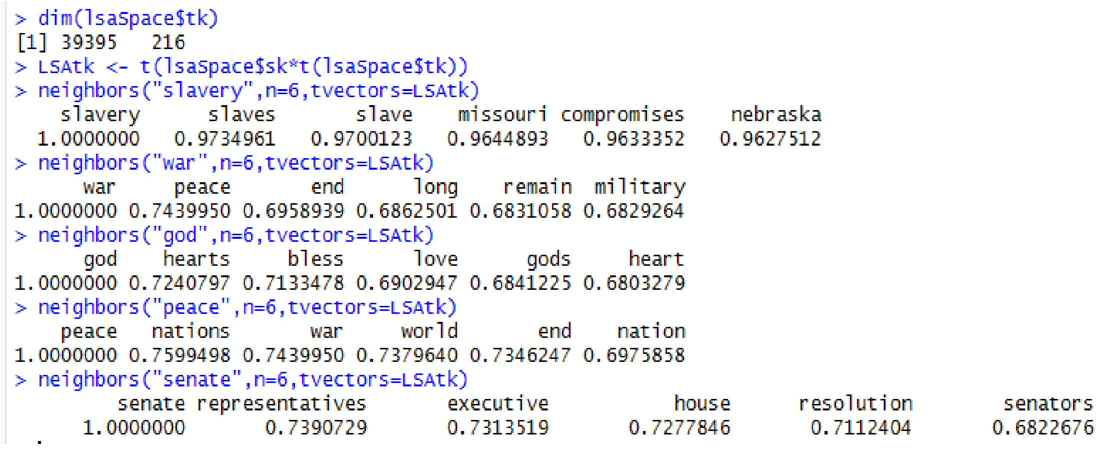
	adams_sp	adams_sp	adams_sp	adams_sp	adams_sp	adams_sp	adams_sp	bush_spe	bush_spe	bush_spe	bush_spe	bush_spe
adams_sp	0	0.953	0.997	0.969	0.996	0.997	0.818	0	0	0	0	0
adams_sp	0.953	0	0.951	0.991	0.969	0.951	0.934	0	0	0	0.851	0
adams_sp	0.997	0.951	0	0.968	0.996	0.999	0.819	0	0	0	0	0
adams_sp	0.969	0.991	0.968	0	0.984	0.969	0.912	0	0	0	0.846	0
adams_sp	0.996	0.969	0.996	0.984	0	0.997	0.851	0	0	0	0	0
adams_sp	0.997	0.951	0.999	0.969	0.997	0	0.824	0	0	0	0	0
adams_sp	0.818	0.934	0.819	0.912	0.851	0.824	0	0	0	0	0.874	0
bush_spe	0	0	0	0	0	0	0	0	0	0	0	0
bush_spe	0	0	0	0	0	0	0	0	0	0.978	0.877	0.896
bush_spe	0	0	0	0	0	0	0	0	0.978	0	0.827	0.899
bush_spe	0	0.851	0	0.846	0	0	0.874	0	0.877	0.827	0	0.888
bush_spe	0	0	0	0	0	0	0	0	0.896	0.899	0.888	0
bush_spe	0	0	0	0	0	0	0	0	0.898	0.883	0	0
clinton_sp	0	0	0	0	0	0	0	0	0.978	0.967	0.918	0.92
clinton_sp	0	0	0	0	0	0	0	0	0.954	0.926	0	0
clinton_sp	0	0	0	0	0	0	0	0	0.922	0.917	0	0
clinton_sp	0	0	0	0	0	0	0	0	0.98	0.94	0.908	0.877
clinton_sp	0	0	0	0	0	0	0	0	0.873	0.909	0	0
clinton_sp	0	0	0	0	0	0	0	0	0	0	0	0

LATENT SEMANTIC ANALYSIS

4) Nearest neighbours



```
> library(LSAfun)
> dim(lsaSpace$tk)
[1] 10056     8
> LSAtk <- t(lsaSpace$sk*t(lsaSpace$tk))
> neighbors("slavery",n=6,tvectors=LSAtk)
  slavery    slaves      thus    passed principle  contrary
1.0000000 0.9987572 0.9983639 0.9960265 0.9946493 0.9935945
> neighbors("war",n=6,tvectors=LSAtk)
      war      love     civil     peace     labor      door
1.0000000 0.9805879 0.9675396 0.9511046 0.9444749 0.9435230
> neighbors("god",n=6,tvectors=LSAtk)
      god    taking     first    others     since      left
1.0000000 0.9864378 0.9841132 0.9834958 0.9753975 0.9735311
> neighbors("peace",n=6,tvectors=LSAtk)
    peace     labor      love concerned     might continent
1.0000000 0.9774030 0.9763627 0.9703560 0.9607508 0.9603190
> neighbors("senate",n=6,tvectors=LSAtk)
   senate    amount conceived    indeed       far patriotic
1.0000000 0.9947933 0.9838897 0.9826430 0.9803616 0.9787218
> |
```



```
> dim(lsaSpace$tk)
[1] 39395   216
> LSAtk <- t(lsaSpace$sk*t(lsaSpace$tk))
> neighbors("slavery",n=6,tvectors=LSAtk)
    slavery        slaves         slave     missouri compromises      nebraska
  1.0000000     0.9734961     0.9700123    0.9644893   0.9633352     0.9627512
> neighbors("war",n=6,tvectors=LSAtk)
      war     peace       end      long    remain  military
1.0000000 0.7439950 0.6958939 0.6862501 0.6831058 0.6829264
> neighbors("god",n=6,tvectors=LSAtk)
      god    hearts     bless      love      gods     heart
1.0000000 0.7240797 0.7133478 0.6902947 0.6841225 0.6803279
> neighbors("peace",n=6,tvectors=LSAtk)
    peace   nations       war     world       end    nation
1.0000000 0.7599498 0.7439950 0.7379640 0.7346247 0.6975858
> neighbors("senate",n=6,tvectors=LSAtk)
          senate representatives       executive           house      resolution        senators
       1.0000000       0.7390729       0.7313519       0.7277846       0.7112404       0.6822676
```

Small corpus – 49 speeches

Full corpus – 962 speeches

See 07a- lsa-big corpus.R

Topic Modelling

INTRODUCTION

The standard way to search for documents is via keywords.

….but often, we don’t know what we’re looking for.

For example, you may have a large collection of documents, but you have know idea what they are about.

Topic modelling is a technique that can help in this situation.

WHAT IS TOPIC MODELLING?

A technique that enables the *unsupervised* discovery of topics in a collection of documents

We'll use an algorithm called *Latent Dirichlet Allocation* (LDA).

Why *latent*?

Because it discovers *hidden* topics by looking for important keywords, both in documents and across the entire corpus.

As in k-means clustering, the number of topics (also denoted by k) has to be specified upfront.

EXAMPLE

Say you have the following sentences:

> 1. I like ham and cheese

> 2. I ate a cheese sandwich for lunch

> 3. Rats and cockroaches are pests

> 4. Rats like cheese

> 5. Cats and rats don't get along

Classify into 2 topics (# of topics is user specified)

Here’s what you might get

> {cheese, ham, sandwich} topic A; {rats, cats cockroaches} (topic B)

> Sentences 1 and 2 - 100% topic A

> Sentences 3 and 5 - 100% topic B

> Sentence 4 - 50% topic A 50% topic B

BASICS

Aim is to discover automatically discover topics in a collection of documents.

Documents are known but

> Topics

> Per-document topic distributions

> Per-document per-word topic assignments

are *latent* (hidden). The word “topic” should not be taken literally – it is simply a bunch of words that occur together.

The computational problem is to generate a latent topic structure which is consistent with the (known) document structure.

HOW IT WORKS (SIMPLIFIED)

Randomly assign every word in the corpus the k topics.

For each document and word (in the document)

> Update the topic assignment for the word based on:

  > The relative importance of different topics in the current document.

  > How often the current word is associated with this topic across all documents.

> Iterate through the above steps until topic assignments become stable.

See the following article for more details (at an intuitive level):

> http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/

Why Dirichlet?

> Per-document topics and per-topic words are assumed to follow Dirichlet probability distributions.

TOPIC MODELLING IN R

Package *topicmodels*, algorithm *lda*

Need to set number of topics upfront (k)

Uses *Gibbs Sampling* – a probabilistic technique. See http://www.matthewjockers.net/2011/09/29/the-lda-buffet-is-now-open-or-latent-dirichlet-allocation-for-english-majors/ for a simple explanation.

Gibbs sampling uses a random walk to sample from a distribution. Requires a number of parameters:

> Burn-in period (burnin=1000)

> Thinning parameter (thin-500)

> Total number of iterations (iter=2000)

> Independent runs (nstart=5)

> Seed for each run (5 integers)

> Pick the run with the highest posterior probability (best=TRUE)

EXERCISE

Using the same corpus as before (docs-1):

> Run the lda algorithm for k=6

> For each value of k, list the following:

  > Top 8 words in each of the topics

  > Primary (top) topic assignment for each document

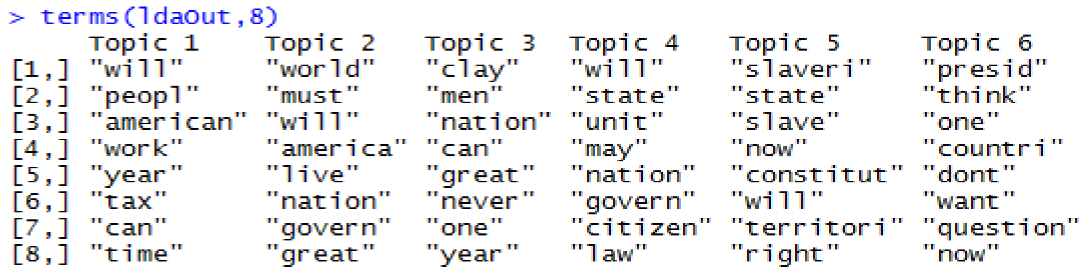
  > Per document probability of each topic

> Are the primary topic assignments reasonable? Discuss why or why not.

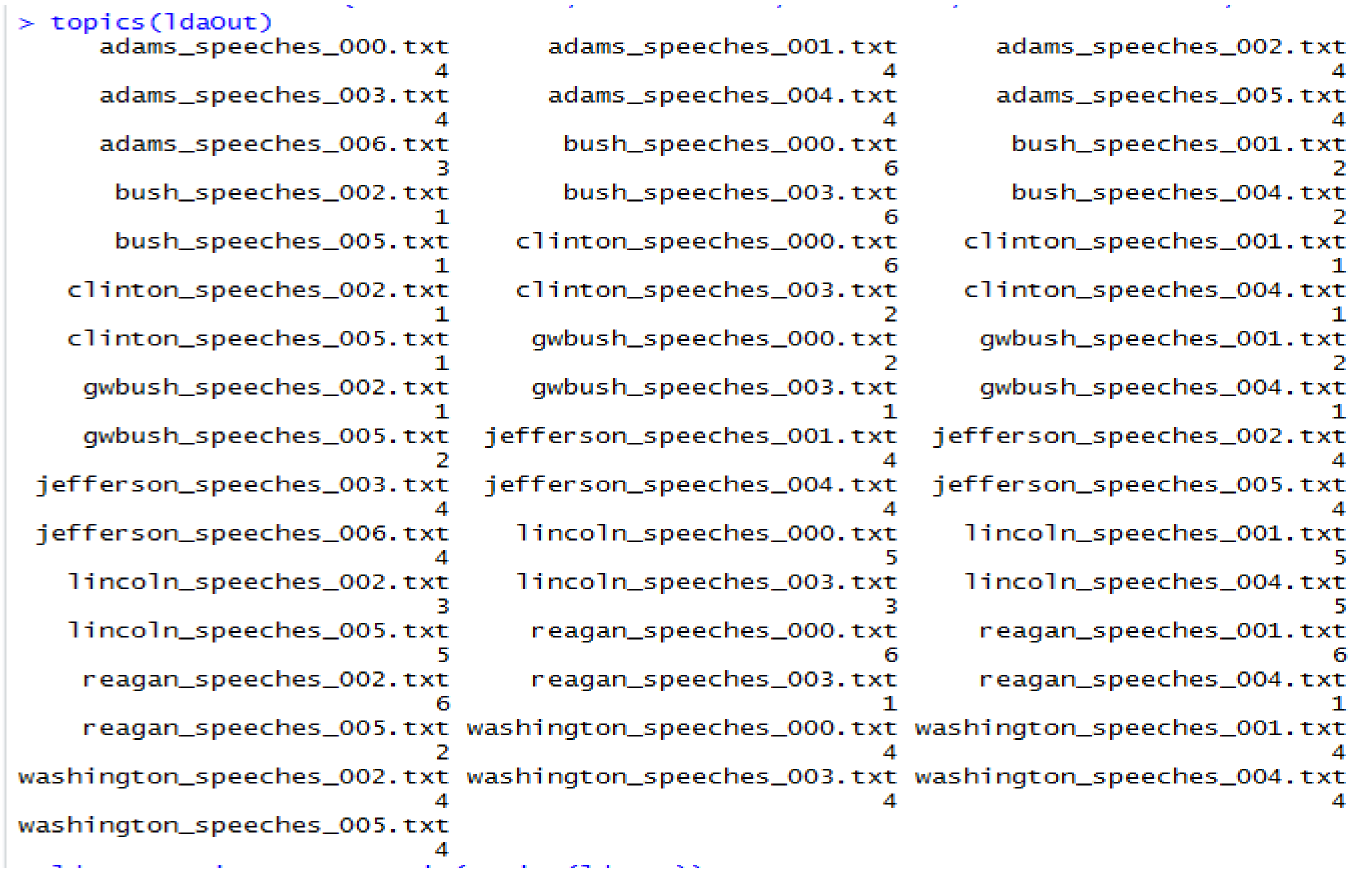
> Experiment with other values of k, say 4 to 8.

> Exercise file: **08-topicmodels.R**

TOPIC MODELLING



```
> terms(ldaOut,8)
     Topic 1    Topic 2    Topic 3  Topic 4   Topic 5      Topic 6
[1,] "will"     "world"    "clay"   "will"    "slaveri"    "presid"
[2,] "peopl"    "must"     "men"    "state"   "state"      "think"
[3,] "american" "will"     "nation" "unit"    "slave"      "one"
[4,] "work"     "america"  "can"    "may"     "now"        "countri"
[5,] "year"     "live"     "great"  "nation"  "constitut"  "dont"
[6,] "tax"      "nation"   "never"  "govern"  "will"       "want"
[7,] "can"      "govern"   "one"    "citizen" "territori"  "question"
[8,] "time"     "great"    "year"   "law"     "right"      "now"
```



```
> topics(ldaOut)
     adams_speeches_000.txt      adams_speeches_001.txt      adams_speeches_002.txt
                          4                           4                           4
     adams_speeches_003.txt      adams_speeches_004.txt      adams_speeches_005.txt
                          4                           4                           4
     adams_speeches_006.txt       bush_speeches_000.txt       bush_speeches_001.txt
                          3                           6                           2
      bush_speeches_002.txt       bush_speeches_003.txt       bush_speeches_004.txt
                          1                           6                           2
      bush_speeches_005.txt    clinton_speeches_000.txt    clinton_speeches_001.txt
                          1                           6                           1
   clinton_speeches_002.txt    clinton_speeches_003.txt    clinton_speeches_004.txt
                          1                           2                           1
   clinton_speeches_005.txt     gwbush_speeches_000.txt     gwbush_speeches_001.txt
                          1                           2                           2
    gwbush_speeches_002.txt     gwbush_speeches_003.txt     gwbush_speeches_004.txt
                          1                           1                           1
    gwbush_speeches_005.txt  jefferson_speeches_001.txt  jefferson_speeches_002.txt
                          2                           4                           4
 jefferson_speeches_003.txt  jefferson_speeches_004.txt  jefferson_speeches_005.txt
                          4                           4                           4
 jefferson_speeches_006.txt    lincoln_speeches_000.txt    lincoln_speeches_001.txt
                          4                           5                           5
   lincoln_speeches_002.txt    lincoln_speeches_003.txt    lincoln_speeches_004.txt
                          3                           3                           5
   lincoln_speeches_005.txt     reagan_speeches_000.txt     reagan_speeches_001.txt
                          5                           6                           6
    reagan_speeches_002.txt     reagan_speeches_003.txt     reagan_speeches_004.txt
                          6                           1                           1
    reagan_speeches_005.txt washington_speeches_000.txt washington_speeches_001.txt
                          2                           4                           4
washington_speeches_002.txt washington_speeches_003.txt washington_speeches_004.txt
                          4                           4                           4
washington_speeches_005.txt
                          4
```

Extension Topic:

Word Embedding Models

VECTOR REPRESENTATIONS

The methods we've seen so far deal with words as atomic entities – each word is encoded as a 1 or 0 in the DTM (or TF-IDF) matrix.

We call this a "one-hot" representation. E.g.,

shop is represented as [0 0 0 0 0 0 1 0 0 0]

store is represented as [0 0 1 0 0 0 0 0 0 0]

Problems:

- The representation does not carry information on the *meaning* of the word
- There is no notion of *similarity* between two words

Word embedding is based on the principle that we can gain knowledge of the **different contexts** in which a word is used by looking at its neighbours.

WORD MEANING

Definition: **meaning** (Webster dictionary)

- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.

➔ We can define the meaning of a word by other words commonly surrounding it – encoding by a dense vector (Similar to LSA, but different)

there have never been any shark attacks at places like

juvenile white sharks used to area closer to the entrance

inhabited with far more great white sharks than previously thought

tagged great white sharks, some up to 3.2 metres long

only one shark being detected in the western section

…

$$\text{shark} = \begin{pmatrix} 0.134 \\ -0.231 \\ 0.564 \\ 0.034 \\ -0.013 \\ 0.983 \\ \cdots \end{pmatrix}$$

WORD2VEC

- word2vec is a group of models used for word embedding, Mikolov et al. 2013 (google inc.)
- Neural network models are trained on a large corpus of text to reconstruct linguistic contexts of words.

Two algorithms:

1. Skip-grams (SG): predict the surrounding window of context words from the current word
2. Continuous bag-of-words (CBOW): predict the current word from a window of surrounding context words. Order does not matter

GLOVE

- GloVe (Global Vector Embedding) is another approach. Works as follows:
  - Build a word co-occurrence matrix based on a rolling window (5-10 words is common)\
  - The **co-occurrence ratios between two words in a context** are strongly connected to meaning.

  What does this mean?

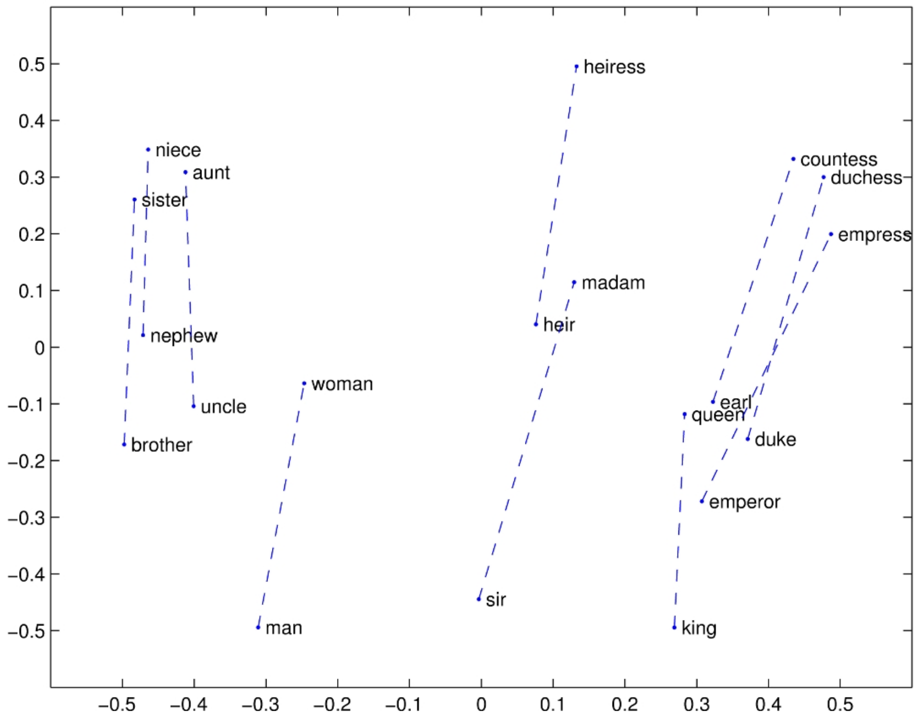
  > Consider the words “ice” and “steam”. Different, yet related because they are both forms of water.

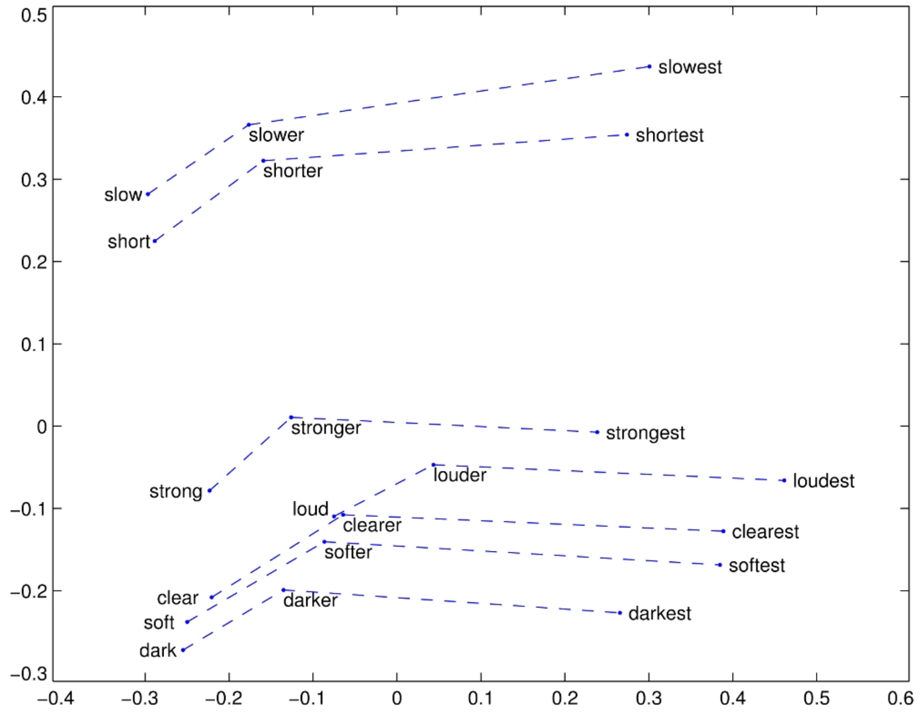
  > So, we would expect words related to water (like “water” and “wet”) to appear equally in the context of “ice” and “steam”.

  > In contrast, words like “cold” and “solid” would probably appear near “ice” but would not appear near “steam”.

INTERPRETATION

The model gives us a set of vectors (say 300) of length N (number of words). This is a 300 x N dimensional vector space. These vectors have some interesting properties:





Source: Socher 2016, lecture 3, http://web.stanford.edu/class/cs224n/syllabus.html

MODEL PARAMETERS

- Number of word vectors: Commonly set to 300 or 1000
- Context window size: Common value is 10. Larger window picks up more semantic relationships, smaller window focusses on syntax
- Sub-sampling: higher frequency words provide less information, can remove these
- Data size: a large corpus is required, commonly 1 billion words (approx. 40 GB of text)

Building these models takes time and requires a lot of data, but pre-built models are available online.

APPLICATIONS

- Known to improve accuracy of most NLP tasks (sentiment analysis, named entity recognition, topic modelling).
- Word vectors are passed to machine learning models as features
- Widely used in state-of-the-art research and in industry
- Building these models takes time and requires a lot of data, but pre-built models are available online
- Two main releases: GloVe and word2vec

Note: Python is currently better supported than R (due to this research coming from Google)

IN CLASS EXERCISE

File 09-text2vec_example.R

> Uses a 2006 snapshot of a subset of Wikipedia

> **Warning: may hang your machine. Run at your own risk!!!**

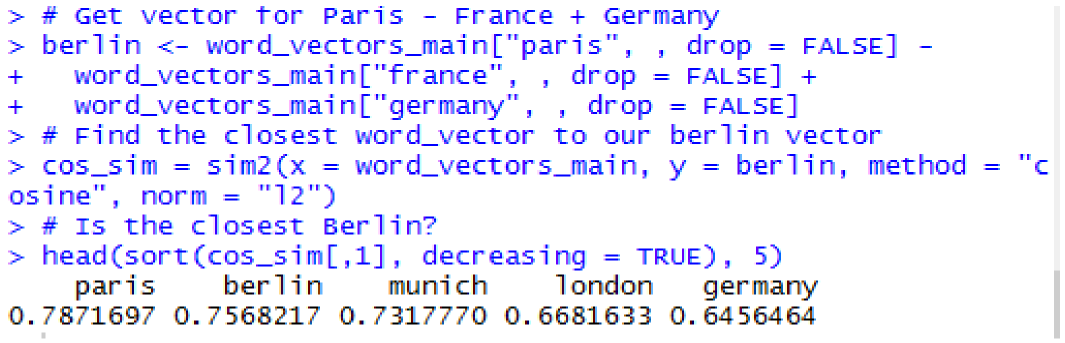
> Compute closest vectors to:

  > (paris – france + Germany)

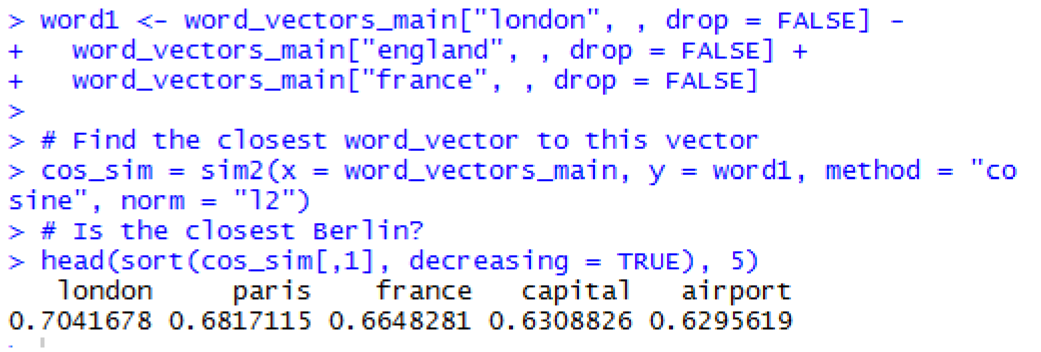
  > (london – england+ france)\

  > (book – author + artist)

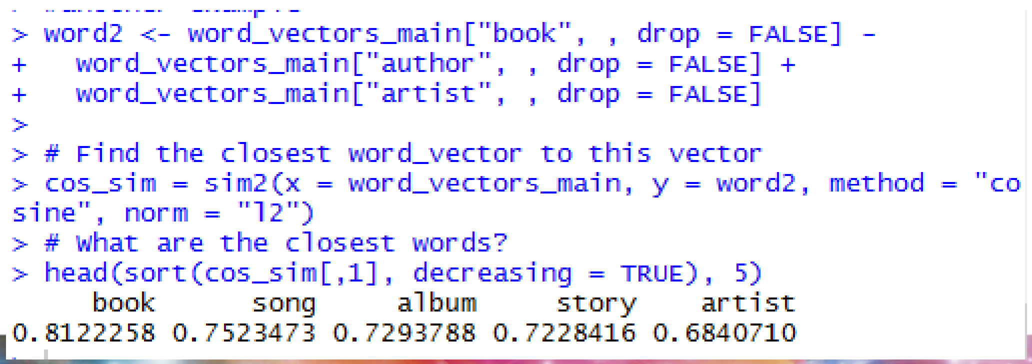
GLOVE VECTOR ARITHMETIC



```
> # Get vector for Paris - France + Germany
> berlin <- word_vectors_main["paris", , drop = FALSE] -
+   word_vectors_main["france", , drop = FALSE] +
+   word_vectors_main["germany", , drop = FALSE]
> # Find the closest word_vector to our berlin vector
> cos_sim = sim2(x = word_vectors_main, y = berlin, method = "c
osine", norm = "l2")
> # Is the closest Berlin?
> head(sort(cos_sim[,1], decreasing = TRUE), 5)
    paris    berlin    munich    london   germany 
0.7871697 0.7568217 0.7317770 0.6681633 0.6456464 
```



```
> word1 <- word_vectors_main["london", , drop = FALSE] -
+   word_vectors_main["england", , drop = FALSE] +
+   word_vectors_main["france", , drop = FALSE]
> 
> # Find the closest word_vector to this vector
> cos_sim = sim2(x = word_vectors_main, y = word1, method = "co
sine", norm = "l2")
> # Is the closest Berlin?
> head(sort(cos_sim[,1], decreasing = TRUE), 5)
   london     paris    france   capital   airport 
0.7041678 0.6817115 0.6648281 0.6308826 0.6295619 
```



```
> word2 <- word_vectors_main["book", , drop = FALSE] -
+   word_vectors_main["author", , drop = FALSE] +
+   word_vectors_main["artist", , drop = FALSE]
> 
> # Find the closest word_vector to this vector
> cos_sim = sim2(x = word_vectors_main, y = word2, method = "co
sine", norm = "l2")
> # What are the closest words?
> head(sort(cos_sim[,1], decreasing = TRUE), 5)
     book      song     album     story    artist 
0.8122258 0.7523473 0.7293788 0.7228416 0.6840710 
```

FURTHER READING

Main pages (documentation, code and pre-built models):
word2vec - https://code.google.com/archive/p/word2vec/
GloVe - https://nlp.stanford.edu/projects/glove/

Python:
https://radimrehurek.com/gensim/
https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-3-more-fun-with-word-vectors

R:
https://github.com/mukul13/rword2vec
https://github.com/bmschmidt/wordVectors

Stanford Deep NLP Course:
http://web.stanford.edu/class/cs224n/syllabus.html

Papers:
https://arxiv.org/abs/1301.3781
https://cs.stanford.edu/~quocle/paragraph_vector.pdf

Naïve Bayes

A CRASH COURSE ON PROBABILITY

Mathematical definition of probability

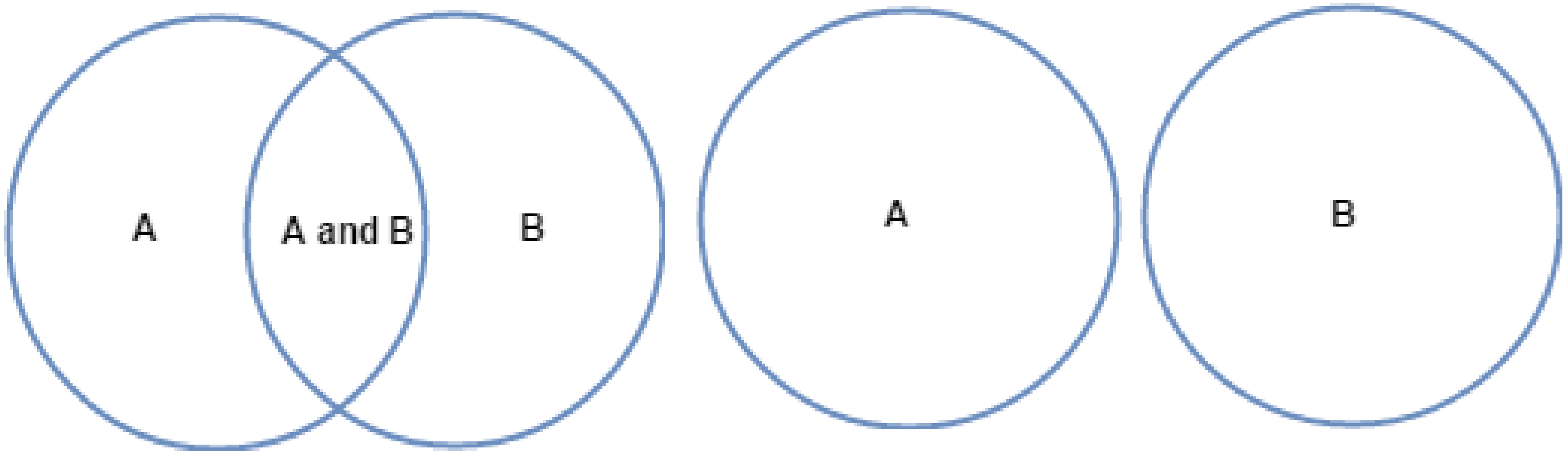
Probability of an event has the following properties:

1. $0 \leq \mathrm{P(A)} \leq 1$, for any event A

2. $\sum \mathrm{P(A_i)} = 1$ where the sum runs over all possible events

Let's look at more complex situations

PROBABILITY OF A OR B



P(A or B) = P(A) + P(B) – P(A and B)

If A and B are mutually exclusive then,

P(A or B) = P(A) + P(B)

Question: What about P(A and B)??

CHAIN RULE AND CONDITIONAL PROBABILITY

What is the probability of two events co-occurring?

P (A and B) = P(A) * P(B given that A has occurred)

P (A and B) written as **P(A ∩ B)**

P(B given that A has occurred) written as **P(B|A)**

P(A ∩ B) = P(A) * P(B|A) – Chain Rule

Example:

What is the probability that a student picked at random is blond and male?

P(Male ∩ Blond) = P(Male) * P(Blond|Male)

CONDITIONAL PROBABILITY

Question:

A bag contains 2 blue and 3 red marbles. What is the probability of picking:

1. A blue marble followed by a red marble?
2. A red marble followed by a blue marble?

Comment on your answers.

CONDITIONAL PROBABILITY

Probability of picking blue, P(B) = 2/5 = 0.4

Probability of picking red given that we already picked blue,

P(R|B) = 3/4 = 0.75 (**Conditional Probability**)

P(Blue and Red)

P(B ∩ R) = P(B) * P(R|B) = 0.4 * 0.75 = **<u>0.3</u>**

Probability of picking red, P(R) = 3/5 = 0.6

Probability of picking blue given that we already picked red,

P(B|R) = 2/4 = 0.5 (**Conditional Probability**)

P(Red and Blue)

P(R ∩ B) = P(R) * P(B|R) = 0.6 * 0.5 = **<u>0.3</u>**

BAYES THEOREM

$$P(B \cap R) = P(R \cap B)$$

Or

$$\mathbf{P(B) * P(R|B) = P(R) * P(B|R)}$$

(Bayes Theorem)

…but why is this useful?

BAYES THEOREM

Gives us a way to revise our beliefs based on known probabilities. i.e. it is an *approach to learning*.

Example:

Assume B is a set of beliefs about the world and D is data regarding those beliefs.

B = {Climate change is real **C,** Climate change is BS **~C**}

D = { There is factual support for climate change **F,** There are no facts supporting climate change **~F**}

P(C|F) = P(F|C)*P(C)/P(F) (posterior belief)

Assume reasonable numbers:

P(F|C) = 0.95, P(F|~C)=0.2, P(C)=0.5 (prior belief)

P(F)=P(F|C)*P(C) + P(F/~C)*P(~C)=**0.575**

P(F|C)*P(C) = 0.95*0.5=**0.475**

P(C|F) = 0.475/0.575 = 0.826

NAÏVE BAYES ALGORITHM

The **Naïve Bayes Algorithm** in machine learning assumes that the features (predictors) of a dataset are **conditionally independent** of each other.

For example, if $x_1, x_2 \ldots . x_n$ are words in an SMS, the conditional probability that the words are collectively indicative that the message is spam (y) is given by the product of the individual conditional probabilities:

$$P(x_1 \ldots x_n | y) = \prod_{i=1}^{n} p(x_i | y)$$

This is tantamount to assuming, for example, that the word "*congratulations*" is independent from "*winner*" although they are most likely not.

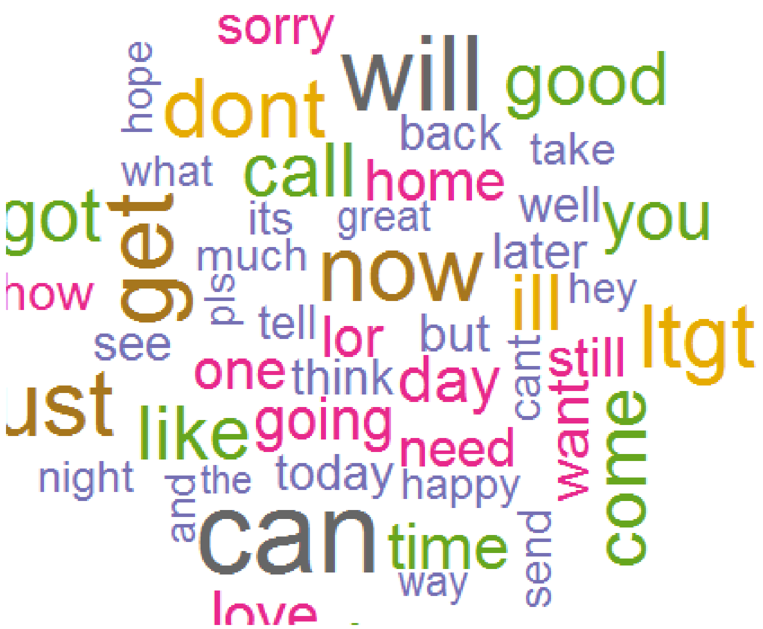
The probability of an email being spam $P(y|x_1 \ldots . x_n)$ is then easily calculated using Bayes theorem.

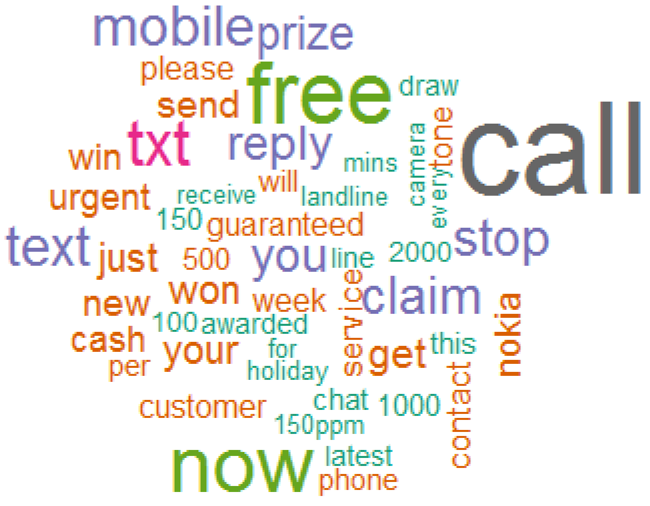
IN CLASS EXERCISE

File 10-naïve_bayes_spam_classifier.R

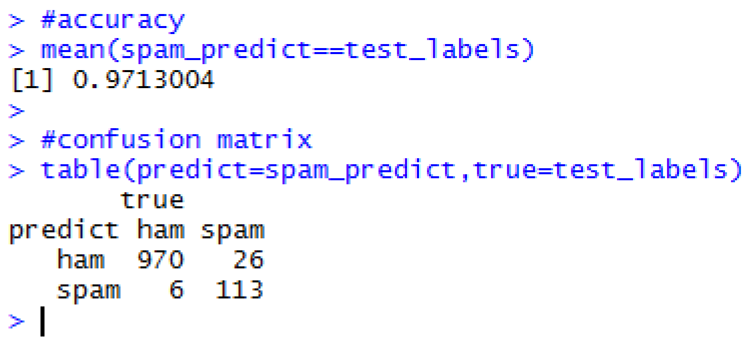
> Uses a dataset of 5572 SMS messages, classified as “ham” or “spam”
>
> > Create wordclouds for spam and ham subsets
>
> > Build a naïve Bayes model that classifies a previously unseen message as spam or ham. Use the **naiveBayes** function from **e1071**.

WORDCLOUDS


sorry
hope
dont
will
good
back
take
what
call
home
got
get
its
great
well
you
much
now
later
how
pls
ill
hey
tell
lor
but
ltgt
see
one
think
day
cant
still
ust
like
going
need
want
come
night
and
the
today
happy
can
time
send
way
love


mobile
prize
please
free
draw
send
call
win
txt
reply
mins
camera
everytone
urgent
receive
will
landline
150
guaranteed
text
just
500
you
line
2000
stop
new
won
week
service
claim
nokia
100
awarded
cash
your
for
get
this
per
holiday
customer
chat
1000
150ppm
contact
now
latest
phone

CONFUSION MATRIX



```
> #accuracy
> mean(spam_predict==test_labels)
[1] 0.9713004
>
> #confusion matrix
> table(predict=spam_predict,true=test_labels)
       true
predict ham spam
   ham  970   26
   spam   6  113
> 
```

Can calculate precision and recall from confusion matrix. Do it!

Which measure do you think is more useful here – precision or recall?

ASSIGNMENT 3

Title: Analysis of Unstructured Data

> Due Date: **Sunday 28th Oct**

> Weight: 30%

> Assignment brief and corpus available at: https://canvas.uts.edu.au/courses/609/assignments/3758

Deliverables:

> Part A: Text Analysis of an Unknown Corpus

> > A written report including visualisations and highlights/recommendations + working code in separate R file . **Length**: 500-1000 words. (25%).

> Part B: Use of Text Analytics in the Workplace

> > Write a **reflective** blog post on CIC Around reflecting on the use of text analysis in the workplace. For example: what can it be used for and what are its implications?. Tag your post with "**DAM-SPR2018**". **Length**: at least 500 words (5%). **Submit link to post on Canvas.**