MET CS688 C1
Web Analytics and Mining

Zlatko Vasilkoski

Text Mining 2

Text Mining Exercises 1 & 2.R

Implementing Text Mining In R

1. **Loading/Accessing Files** (pdf, csv, txt, html, xml etc.).

2. **Extracting textual content** into electronic form (Create Corpus) – using **tm** package, specify tm **readers** and **sources**.

3. **Preprocessing** with **tm_map** - remove numbers, capitalization, common words, punctuation, and prepare your texts for analysis.

4. **Staging the Data** - create a document term matrix (**dtm**).

5. **Explore your data** - Organize terms by their frequency, export dtm to excel, clipboard etc.

6. **Analysis**

   – Analyze most frequent terms - Word Frequency

   – Plot Word Frequencies

   – Relationships Between Terms

   – Term Correlations

   – Word Clouds!

   – ML Analysis (Clustering by Term Similarity, Hierarchal Clustering, K-means clustering)

Step 4. Staging - TF-IDF

A similar term commonly used is TF-IDF (Term Frequency – Inverse Document Frequency)

$$TFIDF = TF * IDF$$

- TF-IDF is very useful in text classification and text clustering.

- It is used to transform documents into numeric vectors, that can easily be compared.

- TF-IDF is a method to generate features from text by multiplying the frequency of a term (usually a word) in a document (the Term Frequency, or TF) by the importance (the Inverse Document Frequency or IDF) of the same term in an entire corpus.

$$TF = \frac{\#\ times\ the\ word\ t\ appears\ in\ a\ Document}{Tot\ \#\ of\ wordt\ per\ Document}$$

- The second term (IDF) weights less important words (e.g. "the", "it", "and" etc.) down, and words that don't occur frequently up.

$$IDF = \ln \frac{Tot\ \#\ of\ Docs}{\#\ of\ Docs\ with\ word\ t}$$

Step 4. Staging - TF-IDF

Example of text to match:

- doc1 =
  'TMitogen-activated protein kinases (MAPKs)',
  'have the remarkable ability to engage substrates, activating MAPK kinases (MAP-KKs), and inactivating phosphatases in',
  'docking sites remote from the catalytic site',
  'structured tails on the substrate or enzyme.'

- doc2 =
  'sequential kinase-interacting motifs (KIMs),',
  'TMitogen-activated protein kinases (MAPKs)',
  'activating MAPK kinases (MAP-KKs), site in MAPKs is bipartite, with a negatively ',

- 3-gram tokenization: of the first doc1 sentence: 'TMitogen-activated protein kinases (MAPKs)'
  [' Tm', 'Tmi', 'mit', 'ito', 'tog', 'oge', 'gen', 'en ', 'n A', ' Ac', 'Act', 'cti', 'tiv', 'iva', 'vat', 'ate', 'ted', 'ed ', 'd P', ' Pr', 'Pro', 'rot', 'ote', 'tei', 'ein', 'in ', 'n K', ' Ki',
  'Kin', 'ina', 'nas', 'ase', 'ses', 'es ', 's M', ' Ma', 'Map', 'apk', 'pks', 'ks ']

Step 4. Staging - TF-IDF

The columns of TF-IDF represent the numeric vectors of all 7 sentences.

The rows of TF-IDF represent the all 3-gram tokenization of the entire text.

Doc1 sentences are the first 4 columns, Doc2 sentences are the last 3 columns

It can be seen that the same sentence (column indices 0 and 5) in the TFIDF is represented by the same vector.

	0	1	2	3	4	5	6
0	0.0	0.0	0.0	0.0	0.0	0.0	0.12743476011817573
1	0.0	0.10287356805522808	0.0	0.0	0.0	0.0	0.0
2	0.1360389406354569	0.06337211404366459	0.0	0.0	0.0	0.1360389406354569	0.07850228493096051
3	0.0	0.10287356805522808	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.12743476011817573
5	0.0	0.0	0.15597931334231135	0.0	0.0	0.0	0.0
6	0.0	0.0	0.15597931334231135	0.0	0.0	0.0	0.0
7	0.0	0.08539389619554588	0.0	0.13628486346577723	0.0	0.0	0.0
8	0.0	0.0	0.15597931334231135	0.0	0.0	0.0	0.0
9	0.0	0.10287356805522808	0.0	0.0	0.0	0.0	0.0
10	0.0	0.14598374328396616	0.0	0.0	0.12301477818663616	0.0	0.09041877159620955
11	0.0	0.0	0.0	0.0	0.0	0.0	0.12743476011817573
12	0.11916630784203766	0.05551219978230875	0.0	0.0	0.18711181914519395	0.11916630784203766	0.06876580638373458
13	0.0	0.08539389619554588	0.0	0.0	0.0	0.0	0.10578179490570076
14	0.1360389406354569	0.1267422808732919	0.0	0.0	0.0	0.1360389406354569	0.23550685479288153
15	0.0	0.0	0.0	0.0	0.17337504671277512	0.0	0.0
16	0.0	0.0	0.0	0.0	0.0	0.0	0.12743476011817573
17	0.0	0.0	0.0	0.1641816429658984	0.0	0.0	0.0
18	0.0	0.0	0.0	0.1641816429658984	0.0	0.0	0.0
19	0.0	0.10287356805522808	0.0	0.0	0.0	0.0	0.0

Step 4. Staging - TF-IDF

Similarity between the vectors is measured using *cosine similarity*.
 docs1=
   'TMitogen-activated protein kinases (MAPKs)',
   'have the remarkable ability to engage substrates, activating MAPK kinases (MAP-KKs), and inactivating phosphatases in',
   'docking sites remote from the catalytic site',
   'structured tails on the substrate or enzyme.'

docs2 =
   'sequential kinase-interacting motifs (KIMs),',
   'TMitogen-activated protein kinases (MAPKs)',
   'activating MAPK kinases (MAP-KKs), site in MAPKs is bipartite, with a negatively ',

The result (cosine similarity is the numeric value below) gives very good match

Doc 1	Cos Sim	Doc 2
TMitogen-activated protein kinases (MAPKs)	1.00	TMitogen-activated protein kinases (MAPKs)
have the remarkable ability to engage substrates, activating MAPK kinases (MAP-KKs), and inactivating phosphatases in	0.35	activating MAPK kinases (MAP-KKs), site in MAPKs is bipartite, with a negatively
docking sites remote from the catalytic site	0.17	activating MAPK kinases (MAP-KKs), site in MAPKs is bipartite, with a negatively
structured tails on the substrate or enzyme.	0.03	TMitogen-activated protein kinases (MAPKs)

Step 4. Staging - Creating a Document Term Matrix

- A document term matrix is a matrix with
  - documents as the **rows**
  - terms as the **columns**
  - a **count** of the frequency of words as the cells of the matrix.

- To create the DTM and TF-IDF matrix this code is used (in the "tm" package)

  ```
  > dtm <- DocumentTermMatrix(Docs.corpus)
  > tfidf <- DocumentTermMatrix(Docs.corpus, control = list(weighting = weightTfIdf))
  ```

- To inspect the document term matrix use

  ```
  > inspect()
  ```

- The transpose of the DocumentTermMatrix() is created with

  ```
  > TermDocumentMatrix(Docs.corpus)
  ```

Exercise: Explore the DTM of "tm.pdf"

Implement all the text mining steps to explore your DTM data

1. Access the "tm.pdf" file from your tm package

   – Use proper tm readers and sources

2. Create Corpus of the "tm.pdf" text content.

3. Skip preprocessing

4. Create the dtm (Document Term matrix).

5. Analyze term frequencies in "tm.pdf".

   – What is max appearance frequency of a term?

   – What are the most/least frequent terms?

Exercise Hints

- To find the term frequency you can use

```
freq <- colSums(as.matrix(dtm)) # Term frequencies
ord <- order(freq) # Ordering the frequencies
freq[tail(ord)] # Most frequent terms
freq[head(ord)] # Least frequent terms
findFreqTerms(dtm, lowfreq=10) # List terms (alphabetically) with frequency higher than 10

# Plot Histogram of Word Frequencies
wf <- data.frame(word=names(freq), freq=freq)
head(wf)
library(ggplot2)
p <- ggplot(subset(wf, freq>7), aes(word, freq))
p <- p + geom_bar(stat="identity")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))
p
```

Step 5. Explore your DTM data

Exploring the Document Term Matrix (*Text Mining Exercises 1 & 2.R*)

- The term frequencies can be obtained as a vector by converting the document term matrix into a matrix and summing the column counts. Obtaining the most frequent terms in the "tm.pdf" is illustrated with the following script.

```
# Example 5: Creating a Document Term Matrix
Docs.pth <- system.file(file.path("doc", "tm.pdf"), package = "tm") # Path to tm.pdf
Docs.corpus <- Corpus(URISource(Docs.pth), readerControl = list(reader = readPDF(engine = "xpdf")))
dtm <- DocumentTermMatrix(Docs.corpus) # Document Term Matrix
freq <- colSums(as.matrix(dtm)) # Term frequencies
ord <- order(freq) # Ordering the frequencies
freq[tail(ord)] # Most frequent terms
```

- The output of this script lists the most frequent terms

```
> freq[tail(ord)]
 which  corpus      for    text   and    the
   18      26       27      27    31     92
```

- For example the most frequent terms in the corpus that appear at least 10 times can be seen with

```
> findFreqTerms(dtm, lowfreq=10)
```

Histogram of Word Frequencies

- Words with freq > 7 are alphabetically ordered.

```
# Create a Histogram
library(ggplot2)
# Organize Histogram Data
WordFreq <- data.frame(word=names(freq), freq=freq)
p <- ggplot(subset(WordFreq, freq>7), aes(word, freq));
p <- p + geom_bar(stat="identity")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))
p # Display Histogram
```
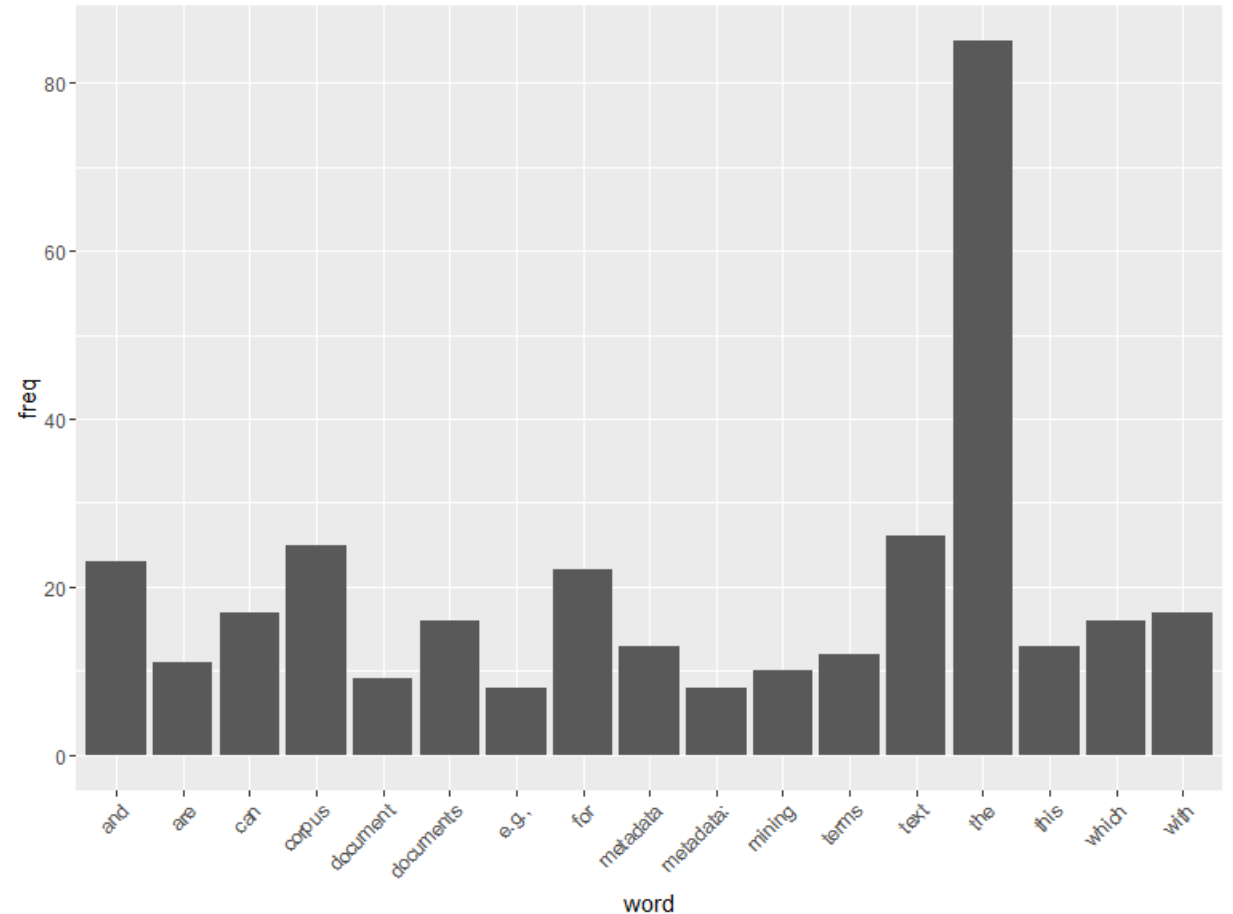
Distribution of Term Frequencies

- For a quick visual overview of the frequency of words in a corpus we can generate a word cloud with:

    > library(wordcloud)

    > set.seed(123)

    > wordcloud(names(freq), freq, min.freq=5, colors=brewer.pal(6, "Dark2"))

- If needed the document term matrix can be converted to a simple matrix and saved.

    > matdtm <- as.matrix(dtm)

    > write.csv(matdtm, file=" dtm.csv")

Step 6. Analysis

Regular and fuzzy string matching

- These operations most commonly refer to strings in the indexed text documents.

- Regular String Matching
  - Please refer to next few slides for the common packages for regular string manipulations in R.
  - Some low-level regular string manipulations include:
    - Splitting a String (try *strsplit()*)
    - Counting the Number of Characters in a String (try count.chars() # From parser package)
    - Detecting a Pattern in a String; Detecting the Presence of a Substring (many ways)
    - Try these on string 'ECfg_AHU_Design_Supply_Airflow'

- Fuzzy String Matching
  - Strings comparison process – similarity between strings, such as in a spell checker.
  - Algorithms reduce to linear algebra concepts such as similarity between vectors (dot product and cosine similarity for example).
  - We'll consider these three types of measures:
    - Character-overlap measures (Jaccard measure, Jaro-Winkler etc.)
    - Edit-distance measures (the minimum operations needed to transform one string into another)
    - N-gram edit distance (similar to the previous, transforms q-characters instead of letters).
- All of these are already implemented in R, you just need to be familiar with them.

Regular String Matching

- Splitting a string

strsplit() : Split the elements of a character vector 'x' into substrings according to the matches to substring 'split' within them.

See also str_split() (stringr package).

```
> unlist(strsplit("a.b.c", "\\."))
[1] "a" "b" "c"
```

tokenize() (tau package) split a string into tokens.

```
> tokenize("abc defghk")
[1] "abc"    " "     "defghk"
```

- Counting the number of characters in a string

nchar() gives the length of a string.

See also str_length() (stringr package).

```
> nchar("Web Analytics")
[1] 13
> str_length("Web Analytics")
[1] 13
> nchar(NA)
[1] 2
> str_length(NA)
[1] NA
```

Regular String Matching

•   Detecting a pattern in a string. Detecting the presence of a substring.

grepl() returns a logical expression (TRUE or FALSE).

str_detect() (stringr package) does a similar job.

```
> string1 <- "Web Analytics"
> string2 <- "Data Analytics"
> str_detect(string1, "Analytics")
[1] TRUE
> str_detect(string2, " Web")
[1] FALSE

> string <- "12 may 2014"
> string2 <- "1 may 2014"
> regexp <- "([[:digit:]]{2}) ([[:alpha:]]+) ([[:digit:]]{4})"
> grepl(pattern = regexp, x = string)
[1] TRUE
> str_detect(string, regexp)
[1] TRUE
> grepl(pattern = regexp, x = string2)
[1] FALSE # the day in the regexp was defined to have 2 digits
```

Fuzzy String Matching

- Queries may have typos and that is a challenge with exact matches of strings. Especially for phrases that use foreign expressions. Fuzzy string matching is similar to regular string matching. It is the process of finding strings that are similar, but not necessarily exactly alike. Spell-checking is just one example of fuzzy string matching.

- Fuzzy string matching immediately opens up a number of questions for which the answers aren't so clear. For instance:
  - How many characters need to match?
  - What if the letters are the same but not in the same order?
  - What if there are extra letters?
  - Are some letters more important than others?

- Different approaches to these questions. These approaches can be broken down into three measures:
  1. Character overlap measures
  2. Edit distance measures
  3. N-gram edit distance

Fuzzy string matching

- For the tasks of fuzzy string matching the R package **stringdist** is a useful tool. Among other functions it contains the function amatch() where the matching algorithm to use can be specified as an argument. Currently, the following distance metrics are supported by stringdist.

Method name	Description
osa	Optimal string aligment, (restricted Damerau-Levenshtein distance).
lv	Levenshtein distance (as in R's native adist).
dl	Full Damerau-Levenshtein distance.
hamming	Hamming distance (a and b must have same nr of characters).
lcs	Longest common substring distance.
qgram	q-gram distance.
cosine	cosine distance between q-gram profiles
jaccard	Jaccard distance between q-gram profiles
jw	Jaro, or Jaro-Winker distance.
soundex	Distance based on soundex encoding.

- For example, finding the Jaccard measure between 2 strings dict1 and dict2 can be done using

```
> amatch(query, c(dict1, dict2), method ="jaccard", maxDist=1)
[1] 2
```
returning the second object (dict2) form the lookup list (c(dict1, dict2)) as a best match.

1. Character overlap measures

There are several similarity measures to this approach.

- Jaccard similarity $\qquad J(A, B) = \dfrac{|A \cap B|}{|A \cup B|}$

- Sorensen–Dice similarity $\qquad D(A, B) = \dfrac{2\,|A \cap B|}{|A|+|B|}$

- Ochiai similarity $\qquad O(A, B) = \dfrac{|A \cap B|}{\sqrt{|A||B|}}$

where $|A|$ and $|B|$ are the number of elements in each set

The Jaccard measure, or similarity coefficient measure is used in the context of string comparisons.
- It is computed as the percentage of unique characters that two strings share when compared to the total number of unique characters in both strings. Let A be a set of characters in the first string, and B is the set of characters in the second string. Then the Jaccard measure is always a number between 0 and 1 and it is calculated as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

where the $|A \cap B|$ is the intersect and $|A \cup B|$ is the union of the unique characters in the two strings A and B.
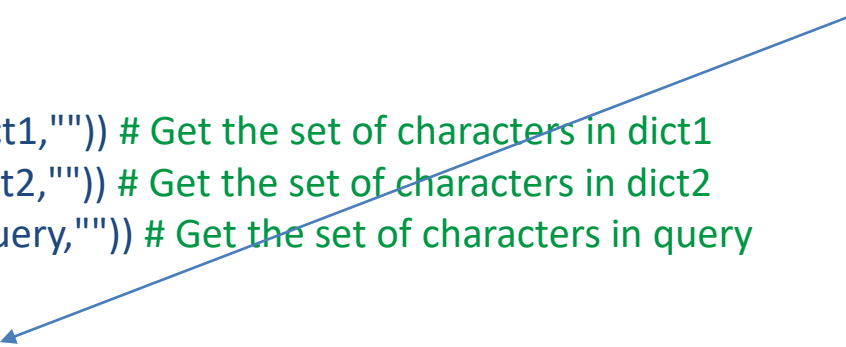
Sometimes the generalized Jaccard similarity is used which is the Jaccard similarity + a threshold.

Exercise

Example: Let the query be the misspelled query word "analisis" and let's have the 2 candidate words A="analytics" and B="analysis" obtained from the dictionary (or a lookup table) as a possible match. This is illustrated by the following R code:

```
# Example 6: The Jaccard measure
library(stringdist)
dict1 <- "analytics"
dict2 <- "analysis"
query <- "analisis"
A <- unlist(strsplit(dict1,"")) # Get the set of characters in dict1
B <- unlist(strsplit(dict2,"")) # Get the set of characters in dict2
Q <-  unlist(strsplit(query,"")) # Get the set of characters in query
UA <- union(A,Q)
IA <- intersect(A,Q)
JA <- length(IA)/length(UA) # The Jaccard measure
sprintf("%s %f","The Jaccard Q-A measure is", JA) # Display calculated measure
UB <- union(B,Q)
IB <- intersect(B,Q)
JB <- length(IB)/length(UB) # The Jaccard measure
sprintf("%s %f","The Jaccard Q-B measure is", JB) # Display calculated measure
```

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Note that the Sorensen–Dice similarity (0.588235) and Ochiai similarity (0.5892557) are the same for the AQ and BQ

The Jaccard distance

- The resulting Jaccard measure indicates that a better match to the misspelled query word "analisis" is the word B="analysis".

  [1] "The Jaccard Q-A measure is 0.625"
  [1] "The Jaccard Q-B measure is 0.833"

$$d_J(A, B) = 1 - J(A, B)$$

- The Jaccard distance can also be defined. It measures dissimilarity between set of characters A and B.
- The Jaccard measure treats all letters equally, and a common extension is to assign a weight to each character based on its frequency.

- Another measure is the Jaro-Winkler distance which matches window of characters from the first string to the second string.

- The character overlap technique does not model character order very well, which is an important feature of wording.

2. Edit distance measures

- This is another approach to determining how similar one string is to another. It uses the number of edit operations required to turn one string into the other string. Computing the minimum sequence of operations needed to transform one string into another can be done by performing nxm comparisons where n is the length of one of the strings and m is the length of the other. The distance matrix is formed for each pair of strings. The columns (rows) of the matrix are set by all the letters contained in the first string (second string).

- For example measuring the distance between the two strings: "Web Analytics" and "Data Analytics" is done by forming a matrix containing columns for each letter of the word "Web Analytics" and containing rows for each letter of the word "Data Analytics".

- This approach can be improved by setting a threshold for edit distances and introducing weights on the edit distance.

3. Q-gram edit distance

• This approach is similar to the previous one but instead of looking at columns and rows made of letters q-characters from the string are considered.

• For example measuring the distance between the two strings: "Web Analytics" and "Data Analytics" by using 3-gram edit distance is done by forming a matrix containing columns for each 3 letters of the word "Web Analytics" (i.e. "Web", "eb ", "b A" etc.) and containing rows for each 3 letters of the word "Data Analytics" (i.e. "Dat", "ata ", "at ", "t A" etc.).

Text Data Categorization

- The goal is to learn from text data, the process is subdivided as follows:
  - Supervised learning (know categories)
    - Classification (ranking sports teams, categories team names and scores being known)
  - Unsupervised learning (no knowledge of what categories are contained in data)
    - Clustering (grouping a set of emails without knowing what they contain)

- There are many different clustering algorithms.
- They all use some kind of similarity measure ("distance") to determine the clusters.
- Typically, the "distance" between two documents is measured as the distance between two vectors.
- There are many different "distance" measures available such as:
  - Euclidean distance (based on the Pythagorean theorem)
  - Manhattan distance (laid out on a square grid, like the streets of Manhattan in New York City)
  - Cosine distance (cosine similarity, as defined earlier)

Exercise - Text Clustering

Task: group (cluster) by similarity documents with the following 12 most frequent words

```
# Example 9: Word Clustering
doc1 <- c("Web Analytics", "Text Analysis", "Web Mining", "Text Mining")
doc2 <- c("Data Processing", "Machine Learning", "learn from data", "Big Data")
doc3 <- c("bedroom furniture", "dining room furniture", "diner chair", "new chairs")
```

- Implement preprocessing - stemming. This will affect the clustering, so you need some intuition to decide what particular pre-processing you need to implement for particular data to achieve better results. Keep 2 version of your corpus
  - Stemmed
  - Not stemmed
- Create the Document term matrix
- Create similarity dendrogram using similarity measures package ("*arules*") and *hclust()* (performs hierarchical cluster analysis).
- Create similarity dendrogram using optimal string alignment (restricted Damerau-Levenshtein distance).

Exercise - Text Clustering

Task: Assume that you have several documents with 12 most frequent strings in them and you would like to group (cluster) them together by similarity.

```
# Example 9: Word Clustering
doc1 <- c("Web Analytics", "Text Analysis", "Web Mining", "Text Mining")
doc2 <- c("Data Processing", "Machine Learning", "learn from data", "Big Data")
doc3 <- c("bedroom furniture", "dining room furniture", "diner chair", "new chairs")
doc <- c(doc1,doc2,doc3) # Merge all strings
dtm <- as.matrix(DocumentTermMatrix(Corpus(VectorSource(doc)))) # Document term matrix
```

Note

- To implement the clustering, we need to create the Document term matrix
- Implement any preprocessing we want to apply.
- This will affect the clustering, so you need some intuition to decide what particular pre-processing you need to implement for particular data to achieve better results.

Text Clustering Examples

- If necessary, stemming can be performed on the strings to improve the clustering performance.

```
corpus.temp <- tm_map(Corpus(VectorSource(doc)), stemDocument, language = "english")
dtm <- as.matrix(DocumentTermMatrix(corpus.temp))
```

- Now all of the terms contained in the document term matrix look like this:

```
colnames(dtm)
 [1] "analysi" "analyt"  "bedroom" "big"     "chair"   "data"    "dine"    "diner"   "from"
[10] "furnitur" "learn"   "machin"  "mine"    "new"     "process" "room"    "text"    "web"
```

- This will affect the clustering since if you perform stemming, the term "chair" and "chairs" will be stemmed to a single term "chair" for example.

Text Clustering Examples

- The non stemmed document term matrix for the 12 strings and the first 7 alphabetically ordered terms of this example looks like this:

```
> dtm[ ,1:7]
   Terms
Docs      analysis   analytics   bedroom   big   chairs   data   dining
 1        0          1           0         0     0        0      0
 2        1          0           0         0     0        0      0
 3        0          0           0         0     0        0      0
 4        0          0           0         0     0        0      0
 5        0          0           0         0     0        1      0
 6        0          0           0         0     0        0      0
 7        0          0           0         0     0        1      0
 8        0          0           0         1     0        1      0
 9        0          0           1         0     0        0      0
 10       0          0           0         0     0        0      1
 11       0          0           0         0     1        0      0
 12       0          0           0         0     1        0      0
```
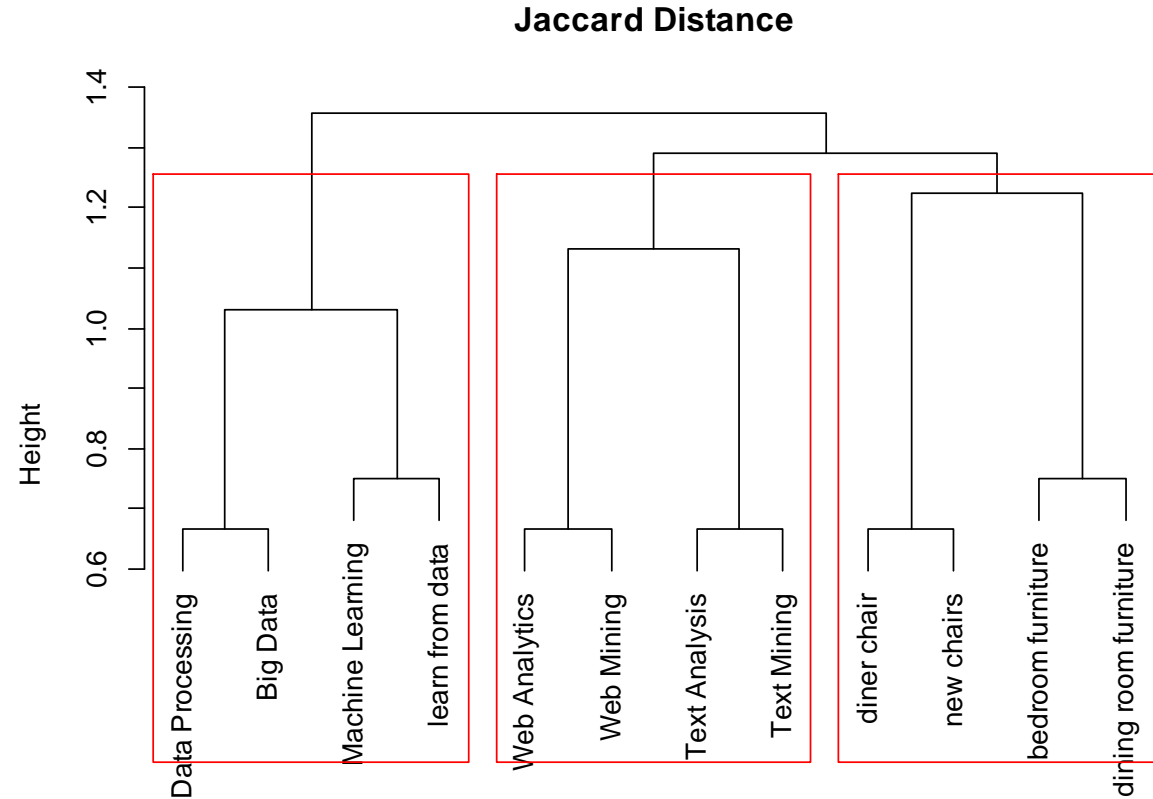
Text Clustering Examples

- The following code illustrates the text clustering.

```
# Using Jaccard Distance
library(cluster) # Load similarity measures package
d <- dist(dtm, method="binary") # Find distance between terms
cl <- hclust(as.dist(d)) # Perform clustering
cl$labels=doc # Assign labels to cluster leaves (documents we had)
plot(cl,main="Jaccard Distance") # Plot and set plot title
```

- Note:
  – The similarity measures package ("*cluster*") that was loaded for this purpose.
  – Obtaining the cluster leaves (*cl* objet) with *hclust()* where the argument was coerced as distance with *as.dist()*.
  – *hclust()* performs hierarchical cluster analysis.
  – The last two lines are related to plotting the dendogram (see lecture notes).
- Try different preprocessing to see the effect on clustering.
- Try different packages and different clustering functions and measures. Always consult help files/package for details.

Word clustering

- Words clustered by Jaccard distance similarity.

Jaccard Distance

Height

Data Processing
Big Data
Machine Learning
learn from data

Web Analytics
Web Mining
Text Analysis
Text Mining

diner chair
new chairs
bedroom furniture
dining room furniture

Term Clustering Dendrogram using
hclust (*, "ward.D2")

Word clustering with Stemming

- Words are stemmed and then clustered by Jaccard distance similarity.

Jaccard Distance with Stemming

Term Clustering Dendrogram using
hclust (*, "complete")
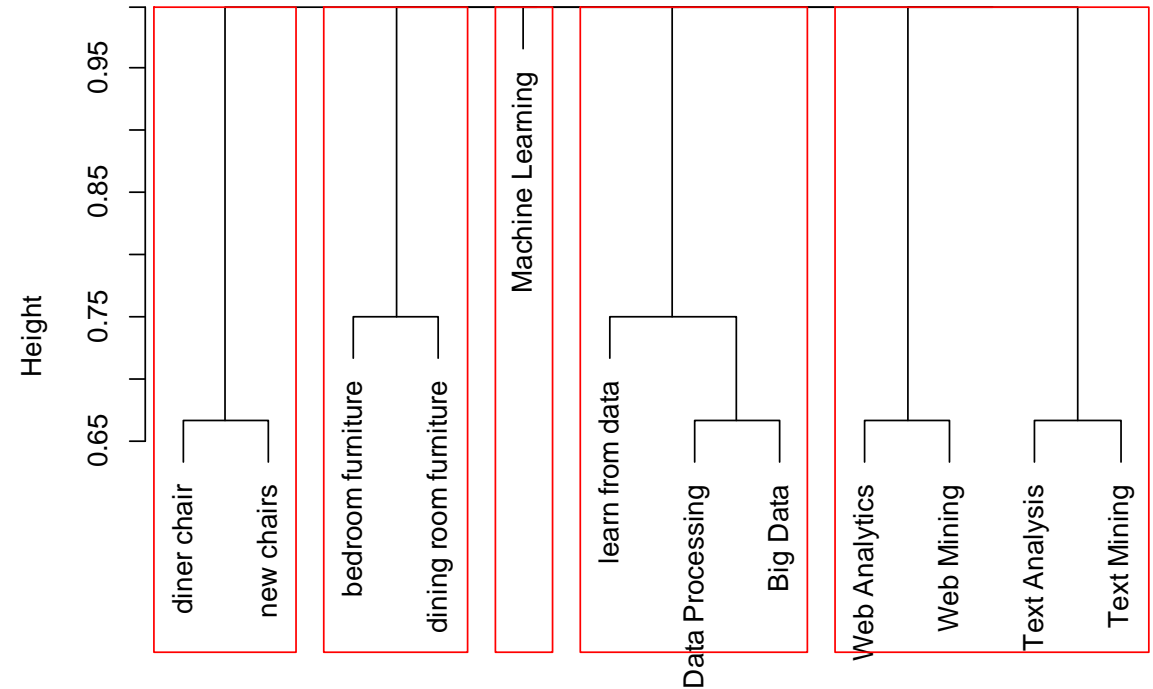
```
# With stemmed terms
corpus.temp <- tm_map(Corpus(VectorSource(doc)), stemDocument, language = "english")
dtm <- as.matrix(DocumentTermMatrix(corpus.temp))
d <- dist(dtm, method="binary") # Find distance between terms in dtm
cl <- hclust(as.dist(d)) # Perform clustering
cl$labels=doc # Assign labels (terms used) to cluster leaves
plot(cl,main="Jaccard Distance with Stemming",xlab="Term Clustering Dendrogram using") # Set plot title
rect.hclust(cl, k=5, border="red") # draw dendogram with red borders around the 5 clusters
```

Text Clustering Examples

- Yet another package (stringdist) can be used to perform clustering as illustrated with the following code:

```
# Hierarchical clustering
library(stringdist)
d <- stringdistmatrix(doc, doc) # Pairwise string distances (optimal string alignment)
cl <- hclust(as.dist(d)) # Perform hierarchical clustering
cl$labels=doc # Assign labels to cluster leaves
plot(cl) # Plot and set plot title
```

- Note that here within the function stringdistmatrix() we are using the default distance measure which is term optimal string alignment or the restricted Damerau-Levenshtein measure.

Algorithms

MIT News: Searching big data faster

http://news.mit.edu/2015/searching-big-data-faster-0826

- Apply compressive algorithms to large-scale biological data.
- Techniques to biological and chemical data easier to analyze by, in some sense by compressing it.
- They identify properties of data sets that make them amenable to compression and present an algorithm for determining whether a given data set has those properties.
- To make searching more efficient, the Berger group's compression algorithms cluster together similar genomic sequences.

Document Classification

- Larger Text-Document Classification, more complicated
  - A "concept space" is created for each document from the document term matrix (DTM).
  - The DTM A is decomposed into 3 simpler matrices (A=USV) using a matrix concept called SVD (singular-value decomposition).
    - U relates to terms
    - V relates to documents
  - The "concept space " is created by choosing the largest values of S (which is diagonal and ordered from highest to lowest values).

  - If you want to create a more serious R application that will categorize documents and files you would need to implement similar kind of approach.
  - SVD is available in R's *base* package and is typically available in most of other language libraries.

$$A = [u_1 \quad u_2 \quad \cdots \quad u_k] \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_k \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_k^T \end{bmatrix}$$

Phases of the Process Used to Develop an Automatic Classifier

- Preparing the Data, Training and Testing the Model, and Deploying the Classifier into Production;

- Tuning Adjusts Training in Response to Test Results;

- Extension Extends a Classifier to Cover New Cases That Arise after Deployment
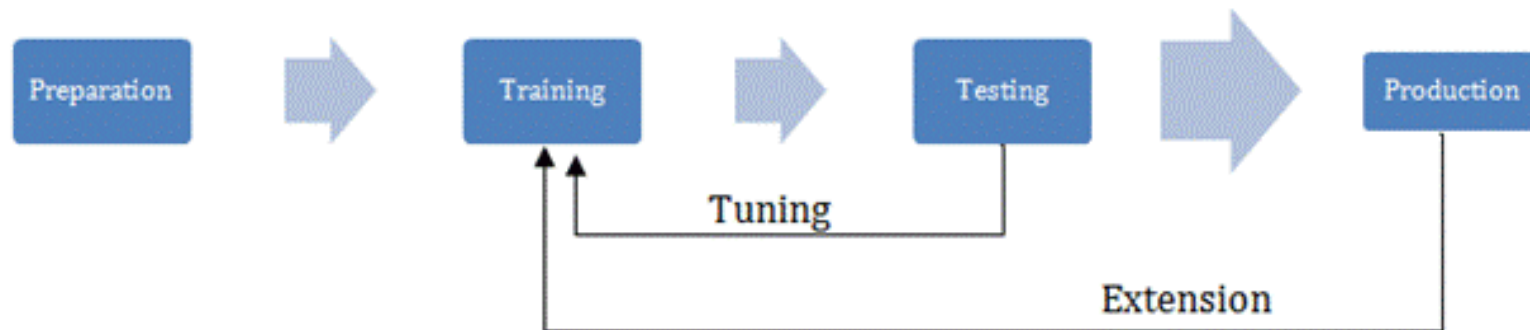
Tagging and Classification

- Classification - Supervised learning
  - We know the groups (tags) into which we separate data.
  - Tags are specific, relevant keywords by which we separate the text metadata.
  - The process of classification provide automated tagging.
  - The term "categorization" refers to assigning a category to an object.

- Training needed.
- So typically the available data is split into several datasets.
  - Training dataset (60%) – data for which we know the classification, used for training the algorithm.
  - Cross validation dataset (20%) – data on which we determine the statistical effectiveness of the classification.
  - Test dataset (20%) – data we analyze, to draw conclusions.

- **Note** in the next example, is good enough to split the data into only 2 datasets
  - Training dataset (50%)
  - Test dataset (50%)

Classification Algorithms

- There are several classification algorithms.

- Consider one of them, the k-NN algorithm. In the k-NN algorithm,
  - Uses (TF - IDF) as the default weighting measure which combines the term frequency (TF), with inverse document frequency (IDF) which is logarithmically scaled. IDF tells whether the term is common or rare across all documents.
  - Cosine similarity is used as the default similarity metric.

- The arguments of the function *knn()* are

  knn(train, test , Tags, k)

  - **train** is a dataset for which classification is already known.
  - **test** is a dataset you are trying to classify.
  - **Tags** is a factor of correct answers for the training dataset
  - **k** # number of neighbors considered per row of data.

- **Note** the function *knn()* expects same number of **train** and **test** datasets.

Exercise

Perform document classification using the *knn()* function (algorihm) from the *class* package.

- Classify the following documents (*Text Mining Exercises 1 & 2.R*):

# Example 10: Document kNN Text Classification
Doc1 <- "I spent 10K on my car. Compared to the prices of most cars in their class it was cheap. It is a red car so I like it and it has a lot of space."
Doc2 <- "I checked the car prices and I could not find a red car for under 10K. So the price was good even though it had a hole. I heard that it belonged to a movie star."
Doc3 <- "I like the red color, so I would buy a red car even if the car's price is over 10K."
Doc4 <- "I don't like red cars. The insurance for red cars is higher regardless of the price and I would not spend more than 10K. I like black cars."
Doc5 <- "A red giant star can curve the space to form a black hole. In absence of stars the space is flat."
Doc6 <- "With exception of the stars the space is filled with blackness making the black holes even harder to see."
Doc7 <- "Our sun is a small star and it will not end as a black hole. It does not have enough mass to curve the space."
Doc8 <- "Very few stars will end as black holes but still the space contains large number of black holes."

Hints using the k-NN algorithm

- There are 8 simple documents containing few sentences.
  - The first 4 documents (document ID from 1 to 4) mostly refer to subject related to cars.
  - The other 4 documents (document ID from 5 to 8) relate to space.
- There are some overlapping words between these 2 categories (space, star, red, black etc.).
- Note:
  - All documents are combined into a Corpus using R's *c()* function.
    - doc <- c(Doc1,Doc2,Doc3,Doc4,Doc5,Doc6,Doc7,Doc8) # Merge all strings
  - Preprocessing is implemented before *dtm* document term matrix is created.
  - The package "*class*" is used for implementing kNN.
    - library(class) # Using kNN
  - The data from *dtm* is equally split into test and train datasets (*train.doc* and *test.doc*).
    - train.doc <- dtm[c(1,2,5,6),] # Dataset for which classification is already known
    - test.doc <- dtm[c(3,4,7,8),] # Dataset you are trying to classify
  - The correct answers for the training dataset the tags (groups) are specified as factor data format required by the *knn()* classification function. Note the use of R's *c()* and *rep()* functions!
    - Tags <- factor(c(rep("cars",2), rep("space",2))) # Tags - Correct answers for the training dataset
  - Finally, the classification is implemented, specifying that we have k=2 classes. The classification probability is returned for each test dataset.
    - prob.test<- knn(train.doc, test.doc, Tags, k = 2, prob=TRUE) # k-number of neighbors considered

Example using the k-NN algorithm

```
# Example: kNN Text Classification
library("tm")
Doc1 <- "I spent 10K on my car. Compared to the prices of most cars in their class it was cheap. It is a red car so I like it and it has a lot of
space."
Doc2 <- "I checked the car prices and I could not find a red car for under 10K. So the price was good even though it had a hole. I heard that
it belonged to a movie star."
Doc3 <-"I like the red color, so I would buy a red car even if the car's price is over 10K."
Doc4 <-"I don't like red cars. The insurance for red cars is higher regardless of the price and I would not spend more than 10K. I like black
cars."
Doc5 <- "A red giant star can curve the space to form a black hole. In absence of stars the space is flat."
Doc6 <- "With exception of the stars the space is filled with blackness making the black holes even harder to see."
Doc7 <- "Our sun is a small star and it will not end as a black hole. It does not have enough mass to curve the space."
Doc8 <- "Very few stars will end as black holes but still the space contains large number of black holes."
doc <- c(Doc1,Doc2,Doc3,Doc4,Doc5,Doc6,Doc7,Doc8) # Merge all strings
corpus <- Corpus(VectorSource(doc))

# Preprocessing
corpus.temp <- tm_map(corpus, removePunctuation) # Remove Punctuation
corpus.temp <- tm_map(corpus.temp, stemDocument, language = "english")# Perform Stemming
dtm <- as.matrix(DocumentTermMatrix(corpus.temp)) # Document term matrix

# Text Classification
library(class) # Using kNN
train.doc <- dtm[c(1,2,5,6),] # Dataset for which classification is already known
test.doc <- dtm[c(3,4,7,8),] # Dataset you are trying to classify
Tags <- factor(c(rep("cars",2), rep("space",2))) # Tags - Correct answers for the training dataset
prob.test<- knn(train.doc, test.doc, Tags, k = 2, prob=TRUE) # k-number of neighbors considered
```

Analyzing the output of the knn()

The classification probability is returned for each test dataset.

Remember the test dataset contained:
1. doc3 related to cars.
2. doc4 related to cars.
3. doc7 related to space.
4. doc8 related to space.

The classification was correct.

```
> a <- 1:length(prob.test)
> b <- levels(prob.test)[prob.test]
> c <- attributes(prob.test)$prob
> result <- data.frame(Doc=a, Predict=b,Prob=c)
> result
  Doc Predict Prob
1   1    cars    1
2   2    cars    1
3   3   space    1
4   4   space    1
> sum(c)/length(Tags) # Overall probability
[1] 1
```

Note that if you change Doc8 to "My car is priced well." and rerun the script, the Doc8 is now classified with the "cars" tag with 50% probability.

Why?
You would need to change the training set and the Tags accordingly.
Finally the classification is not an exact process.

```
> a <- 1:length(prob.test)
> b <- levels(prob.test)[prob.test]
> c <- attributes(prob.test)$prob
> result <- data.frame(Doc=a, Predict=b,Prob=c)
> result
  Doc Predict Prob
1   1    cars  1.0
2   2    cars  1.0
3   3   space  1.0
4   4   space  0.5
> sum(c)/length(Tags) # Overall probability
[1] 0.875
>
> sum(prob.test==Tags)/length(Tags) # % Correct Classification
[1] 1
```

Effectiveness Analysis

- How do we measure effectiveness of an information retrieval system?

- Let's illustrate the effectiveness analysis with the following example.

- Example:
  - Consider the classification performance of an email spam filter based on a user specified set of "spam" words. The filter analyzed 100 emails out of which 27 were spam. The filter classified 21 emails as spam, out of which 11 were actual spam.

- What is the effectiveness of this spam filter?

Let's Define Some Terms

- **Positive** Event:
  - It is typical to choose as "positive" the rare event
  - So let's choose a spam email as a positive event.

- **Negative** Event:
  - Let's choose regular email as negative event.

- Two types of events (emails)
  - **Actual**
  - **Predicted** (classified)
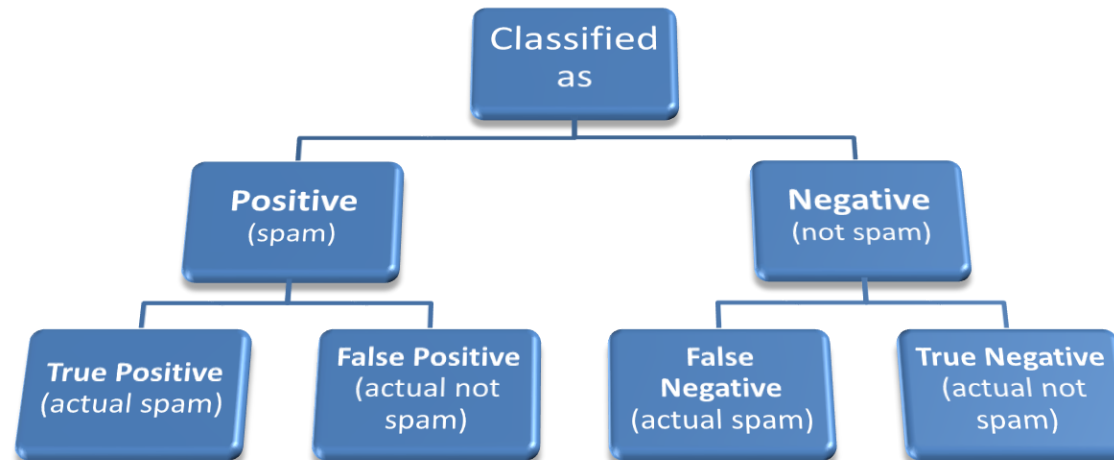
Confusion Matrix Terms

- In particular the terms related to the spam email classification example are:
  - TP - True Positive, actual spam emails (true) classified as spam (positive).
  - FP (Type 1 error) - False Positive, actual not spam emails (false) classified as spam (positive).
  - FN (Type 2 error) - False Negative, actual spam emails (false) classified as not spam (negative).
  - TN - True Negative, actual not spam emails (true) classified as not spam (negative).
- The 1 and the 0 refer to positive and negative events.

Useful links:

https://en.wikipedia.org/wiki/Precision_and_recall

https://en.wikipedia.org/wiki/Confusion_matrix

CM		Actual	
		1	0
Classified as	1	TP	FP
	0	FN	TN
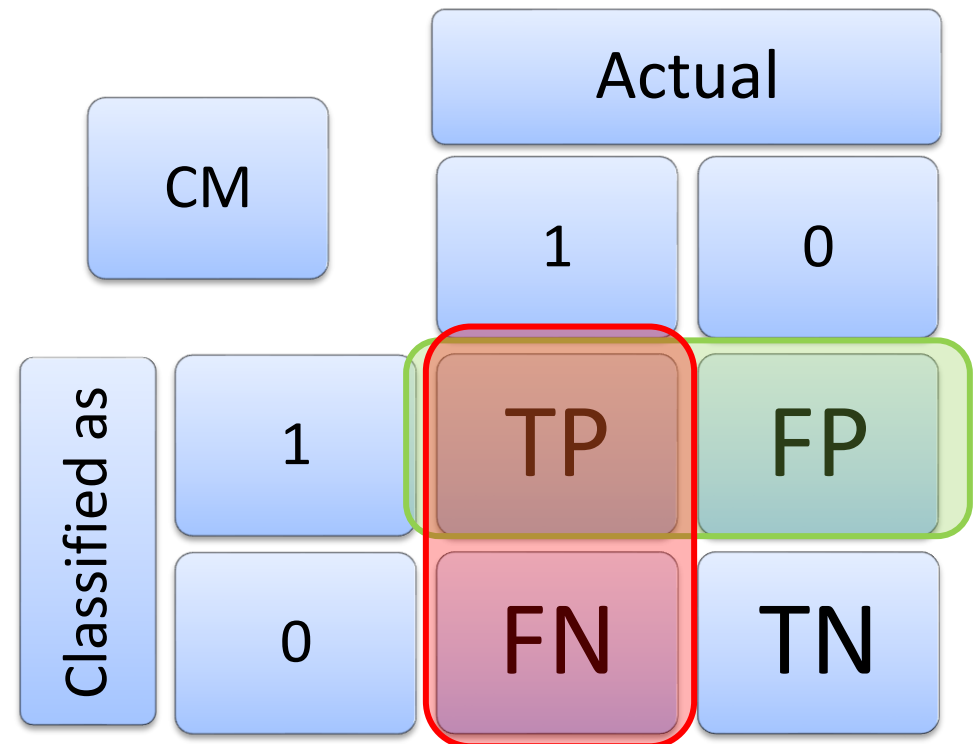
Precision and Recall

- A metric to assess the effectiveness of any classification process
  - **Precision** - is what fraction of the returned results are relevant to the information need.
  - **Recall** - is what fraction of the relevant documents in the collection were returned by the system.
- Mathematically the CM terms are related to precision and recall by the following formulas:

- Question: Why do we not consider TN?

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

	Actual	
CM	1	0

Classified as	1	TP	FP
	0	FN	TN
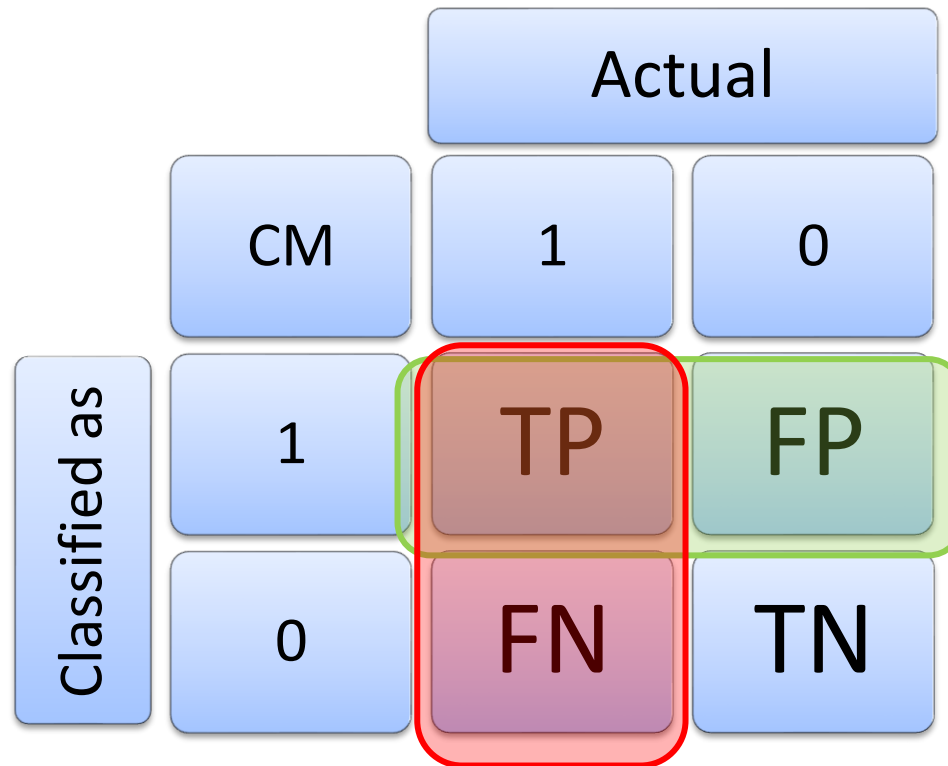
Effectiveness Analysis of a Spam Filter

- Example: Effectiveness analysis of a spam filter
  - Consider the classification performance of an email spam filter based on a user specified set of "spam" words. The filter analyzed 100 emails out of which 27 were spam. The filter classified 21 emails as spam, out of which 11 were actual spam.

- This means
  - **TP** - True Positive, actual spam emails (**true**) classified as spam (**positive**). $TP = 11$
  - **FP** - False Positive, actual not spam emails (**false**) classified as spam (**positive**). $FP = 10$
  - **FN** - False Negative, actual spam emails (**false**) classified as not spam (**negative**). $FN = 16$
  - **TN** - True Negative, actual not spam emails (**true**) classified as not spam (**negative**). $TN = 63$

- Thus the

$$Precision = \frac{TP}{TP+FP} = \frac{11}{11+10} \cdot 100 = 52.4\%$$

$$Recall = \frac{TP}{TP+FN} = \frac{11}{11+16} \cdot 100 = 40.7\%$$

- High precision and recall gives us confidence in the classification effectiveness.
  - Note: before we assess the effectiveness of the classification, typically we divide the available data into 60% training, 20% cross validation and 20% test datasets.
  - Note that the precision and the recall need to be estimated on the **cross validation** dataset.

F score

- Note that these measures may not always be a good effectiveness indicator.
- Consider the following situation

$$TP = 2$$
$$FP = 34$$

- The Precision=5.6%

$$FN = 1$$

- The Recall=66.7%

$$TN = 63$$

- Is this good or bad?
- It is hard to quantify this performance. Better to have single measure.
- F score is defined in terms of precision and recall.

$$F_{score} = 2\frac{P\ R}{P + R}$$

- The F score for the first performance is 45.8%, while for the second is only 10.3%, indicating that the first performance is much more effective.

Classification & Ranking

- Binary **classification**, placing items into one of two types or classes.
- But what if the items in one class are not "created equally" and we want to rank the items within a class?
  - i.e. to say that one email is the most spammy, another is the second most spammy etc.
- Generating rules for **ranking** a list of items is a common task in machine learning.
- This kind of ranking is produced by a **recommendation system**, that we all have encountered.
  - Ecommerce websites benefit from leveraging data on their users to generate recommendations for other products their users might be interested in (i.e. books on a similar topic, accessories, movies etc.).
- Ranking falls into the supervised machine learning type.

Ranking Example: Rank Newsgroups

- Define content features by extracting it from the messages.

- Specifically, if there are common terms in the subjects and bodies of emails received by a user, then future emails that contain these terms in the subject and body may be more important than those that do not.

- This is actually a common technique, and it is mentioned briefly in the description of Google's priority inbox.

- Some of the features:
  - Who is it from?
  - Subject line content.
  - Is it active thread, and how many replies are there?
  - Message content.

Exercise - String Manipulation

- **Task1:** Create a corpus from the following text:

```
# Exercise: Corpus Manipulation.
library(tm)
Doc1 <- "From: ritterbus001@wcsub.ctstateu.edu"
Doc2 <- "Subject: Re: IR remote control receiver"
Doc3 <- "Nntp-Posting-Host: wcsub.ctstateu.edu"
Doc4 <- "Organization: Yale University, Department of Computer Science, New Haven, CT"
Doc5 <- "In article <wb9omc.735429954@dynamo.ecn.purdue.edu>, wb9omc@dynamo.ecn.purdue.edu (Duane P Mantick)
writes:"
```

- **Task2:** Use grepl() to create a meta field in your corpus called "Subject" and place the textual content of the subject line there. (i.e Doc.Corpus[[1]]$meta$Subject<-"…")

Exercise - String Manipulation

```r
### --- Exercise 1: Corpus Manipulation. ----
rm(list=ls()); cat("\014") # clear all
library("tm")
Doc1 <- "From: ritterbus001@wcsub.ctstateu.edu"
Doc2 <- "Subject: Re: IR remote control receiver"
Doc3 <- "Nntp-Posting-Host: wcsub.ctstateu.edu"
Doc4 <- "Organization: Yale University, Department of Computer Science, New Haven, CT"
Doc5 <- "In article <wb9omc.735429954@dynamo.ecn.purdue.edu>, wb9omc@dynamo.ecn.purdue.edu (Duane P Mantick) writes:"

doc <- c(Doc1,Doc2,Doc3,Doc4,Doc5) # Merge all text
Doc.Corpus <- VCorpus(VectorSource(doc))
Doc.Corpus[[1]]$meta

# Preprocessing -- Create corpus only from the "Subject:" line
Subject.List <- list(); cc <- 0
for (ff in 1:length(Doc.Corpus)) {
  TextLine <- unlist(Doc.Corpus[[ff]][1])
  if (grepl("Subject:",TextLine)) {
    cc <- cc +1
    Subject.List[cc] <- gsub("Subject: ","",TextLine)
    Doc.Corpus[[ff]]$meta$Subject <- gsub("Subject: ","",TextLine)
  }
}

Doc.Corpus[[2]]$meta$Subject # Subject line in Corpus
```

Newsgroups Assignment

Classify the Newsgroups data (**by date version data set**) from Blackboard:

- Save data in your "tm/text/" folder so you can specify path using **system.file()**

- Note that the data is separated into one test and one train folder, each containing 20 sub folders on different subjects.

Choose these 2 subjects to analyze (sci.space and rec.autos) and 100 documents from each. **Consider "rec.autos" as positive and "sci.space" as negative event. Note that kNN syntax expects (Positive First, Negative second).**

- For each subject select:
  - 100 documents for training from the train folder
  - 100 documents for testing from the test folder

- Obtain the merged Corpus (of 400 documents), please keep the order as
  - Doc1.Train from the "sci.space" newsgroup train data
  - Doc1.Test from the "sci.space" newsgroup test data
  - Doc2.Train from the " rec.autos" newsgroup train data
  - Doc2.Test from the " rec.autos" newsgroup test data

- Implement preprocessing (clearly indicate what you have used)

- Create the Document-Term Matrix using the following arguments (word lengths of at least 2, word frequency of at least 5)
  - – use: control=list(wordLengths=c(2,Inf), bounds=list(global=c(5,Inf)))

- Split the Document-Term Matrix into proper test/train row ranges
  - train range containing rows (1:100) and (201:300)
  - test range containing rows (101:200) and (301:400)
  - **Note knn expects the Positive ("Rec") event as first, so re-adjust your train/test range if necessary.**

- Use the words "Positive" and "Negative" as tag factors in your classification.
  - Check if the tag order is correct using table(Tags)
  - You should get

    Tags

Positive	Negative
100	100

  - If your order is not right, make proper changes.

- Classify text using the kNN() functionDisplay classification results as a R dataframe and name the columns as:
  - "Doc"
  - "Predict" - Tag factors of predicted subject ("Sci" or "Rec")
  - "Prob" - The classification probability
  - "Correct' - TRUE/FALSE

- What is the percentage of correct (TRUE) classifications?

- Estimate the effectiveness of your classification:
  - Clearly mark the values TP, TN, FP, FN
  - Create the confusion matrix and name the rows and columns with what is Positive/Negative event
 - Calculate Precision, Recall and F-score

Regarding the Assignment - Please note

Suggestion: It is good to save the data in your "tm/texts/" folder so you can specify path using **system.file()**

system.file("texts", "Folder Names to your files", package = "tm")

it easier to deal with the path in R.

Once you extract the files you will have 2 folders (with names "20news-bydate-test" and "20news-bydate-train") each containing 20 subfolders with files on different topics that you would need to perform text mining and analyze for similarity using R.
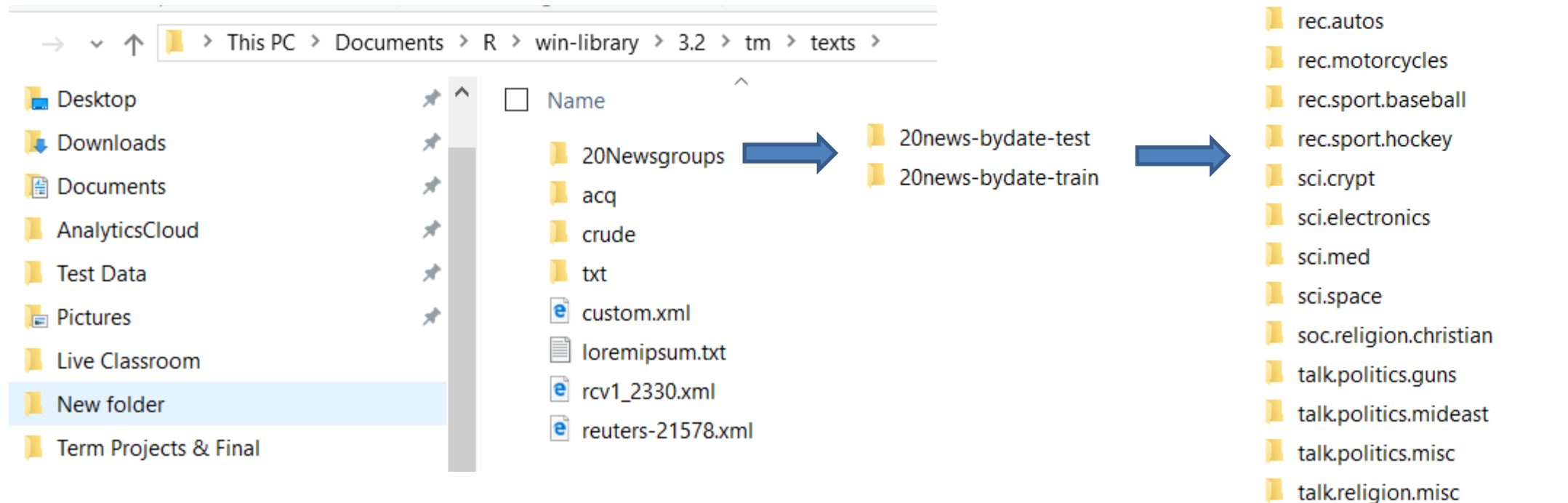
Newsgroups Assignment Hints

Get the Newsgroup data (**by date version data set**)

- Specify (the four) paths to saved Newsgroups data on your computer:

- Suggestion. Save data in your "tm/text/" folder so you can specify path using **system.file()**
    - system.file("texts", "Folder Names to your files", package = "tm")

- Specify *tm* source accordingly (for the 4 datasets) to create an R list of the paths
    - Note that you are going to import files from a folder, so you would need something like
    - > Temp1 <- DirSource(Doc1.TestPath)
    - Temp1$filelist[1] gives "~/R/win-library/3.1/tm/texts/20Newsgroups/20news-bydate-test/sci.space/61242"

- Note that the size of the 4 loaded Newsgroups is different
    - sci.space Test is 394 files
    - sci.space Train is 593 files
    - rec.autos Test is 396 files
    - rec.autos Train is 5934 files

- For each subject you need to select only 100 documents from Test and Train folders:
    - Many ways to do this, one is to just subset Temp1$filelist[1:100]

You can see the R objects you have created in the R Studio Workspace management Window

Note
-the paths to the data files
-The R list of the paths (Temp1)
-You can click on the blue dots to explore the R objects.

Note
-the Corpus properties, size and memory.
-You can explore them by click on the blue dots.

Newsgroups Assignment Hints

- Next create the Corpus
  - > Doc1.Train <- Corpus(URISource(Temp1$filelist[1:100]),readerControl=list(reader=readPlain))

- Please keep the order and the Corpus names as specified for easier grading
  - Doc1.Train from the "sci.space" newsgroup train data
  - Doc1.Test from the "sci.space" newsgroup test data
  - Doc2.Train from the " rec.autos" newsgroup train data
  - Doc2.Test from the " rec.autos" newsgroup test data

- Merge all of 4 Corpora into 1 Corpus keeping the above specified order
  - Using R's *c()* function for example.
  - Why 1 Corpus and not 4? So we can implement all the preprocessing steps at once and create one DTM.

- Implement some preprocessing (clearly indicate what you have used)
  - Note that the end classification result depends on this step.
  - Remember *tm* uses *tm_map()* for basic and custom transforms.
  - Note that you can implement some of the preprocessing inside DocumentTermMatrix , see
    - > ?DocumentTermMatrix

Newsgroups Assignment Hints

- Create the Document-Term Matrix using the following arguments (find out what is proper syntax for this for your version of tm)
  - word lengths of at least 2
  - word frequency of at least 5
- Carefully split the Document-Term Matrix into
  - **Train dataset** for the 2 newsgroups spans the rows (1:100, 201:300)
    - Place Positive Train data in the rows 1:100
    - Place Negative Train data in the rows 201:300
  - **Test dataset** for the 2 newsgroups spans the rows (101:200,301:400)
    - Place Positive Test data in the rows 101:200
    - Place Negative Test data in the rows 301:400

- Create Tags (R **factor** data type) using *rep()*, see lecture notes examples if needed.
  - First 100 are "Positive"
  - 101 to 200 are "Negative"
- Classify text using the kNN() function
- Display Classification Results (it can be the same as in lecture examples).

Newsgroups Assignment Hints

The result should look something like this:

What is percentage of correct (TRUE) classifications?
- It is the % of TRUE in the last column.

If the probability of classification was prob.test then the % is
    sum(prob.test==Tags)/length(Tags)

```
> result
   Doc Predict       Prob Correct
1    1     Sci 0.5000000    TRUE
2    2     Rec 0.5000000   FALSE
3    3     Sci 0.5000000    TRUE
4    4     Rec 0.5000000   FALSE
5    5     Sci 1.0000000    TRUE
6    6     Rec 0.5000000   FALSE
7    7     Sci 1.0000000    TRUE
8    8     Sci 0.5000000    TRUE
9    9     Rec 0.5000000   FALSE
10  10     Rec 0.5000000   FALSE
11  11     Sci 0.5000000    TRUE
12  12     Rec 0.5000000   FALSE
13  13     Rec 0.5000000   FALSE
14  14     Rec 1.0000000   FALSE
15  15     Sci 0.5000000    TRUE
16  16     Rec 0.5000000   FALSE
17  17     Rec 0.5000000   FALSE
18  18     Sci 1.0000000    TRUE
19  19     Sci 1.0000000    TRUE
20  20     Sci 0.5000000    TRUE
21  21     Sci 0.5000000    TRUE
22  22     Rec 0.5000000   FALSE
23  23     Rec 0.6666667   FALSE
```

Newsgroups Assignment Hints

- Use **set.seed(0)** before classification **prob.test <- knn(...)** .

- Create R code for the specified effectiveness measures by yourself to understand the process better and then you can check with the knn output if you did it right.

- You can get the Automatic Confusion Matrix (to compare with)

  table(prob.test, Tags) -> AutoCM

- Carefully choose events (CM depends on this)
  - Positive Events (typically rare events)
  - Negative Events

- Carefully determine what are TP, FN, FP, and TN. For example, I have:

  # If Considered " Sci" as Positive and " Rec" as Negative

  SciClassified <- (prob.test==Tags)[1:100] # Classified as "Sci" (Positive)

  TP <- sum(SciClassified=="TRUE") # Actual "Sci" classified as "Sci"

- One way of creating the CM (as a data.frame R object) would be:

  CM <- data.frame(Rec=c(TP,FN),Sci=c(FP,TN),row.names=c("Sci","Rec"))

General About Optimization

- What probability of classification % would you expect?

- What arguments would you have for the result?

- What is this classification based on?
  - Preprocessing, term frequency, word length.
  - Try changing it but it is very tricky to optimize

- The most typical fact about any classification/clustering (or any machine learning approach in general) algorithms is that you almost always need to adjust them by hand (customize) which is not straight forward. Experience & familiarity with data and the algorithms is necessary.

- Applying brute force and more training data is most common mistake.

Exploring the parameter landscape

- The effectiveness analysis of your analytics typically reduces to questions such as these:
  - How would you find the highest peak (optimal performance) without knowing the (2) parameter landscape?
  - Many statistical techniques (experimental design, ANOVA etc.) may help but there is no easy way.
  - You need to explore most of the combinations for this (2) parameter landscape.
  - Real time problems have multidimensional (100's or more) parameter space.
  - This is why machine learning and neural network techniques are used.