Lesson 11 Text Mining

Outline

- Introduction to Text Mining
- Data Mining versus Text Mining
- Text Mining Process - Three steps
  - Establish the corpus
  - Create and discover DTM
  - Mine DTM using mining tools
    - Clustering
    - Classification

Motivation for Text Mining

- Approximately 80% of the World's data is held in Unstructured formats.
  - Web pages
  - Emails
  - Technical documents
  - Corporate documents
  - Books
  - Digital libraries
  - Customer complaint letters
- Growing rapidly in size and importance
- Text Mining is the process of deriving meaningful information from the natural language text.

Text Mining Applications

- Classification of news stories, web pages etc. according to their content
- Email filtering
- For search and retrieval (search engines)
- Clustering documents or web pages
- And many others…

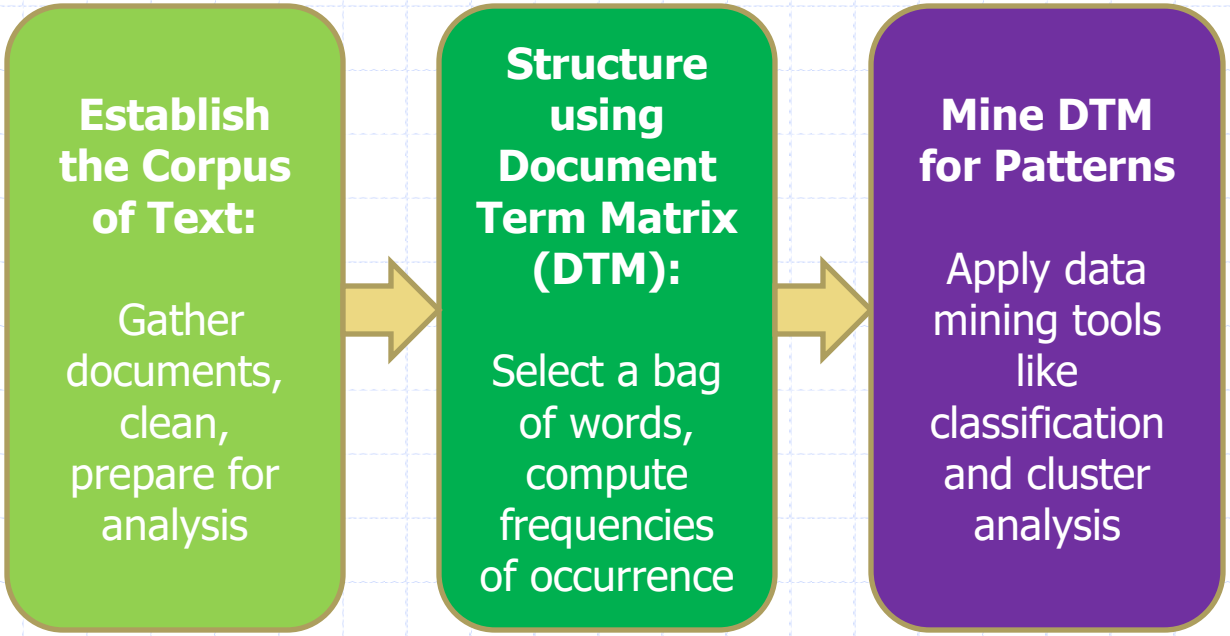
Data Mining versus Text Mining

- Both seek for novel and useful patterns
- Difference is the nature of the data:
  - Data Mining works on structured data stored in databases
  - Text Mining works on unstructured data in Word documents, PDF files, XML files, etc
- Text mining – first, impose structure to the data, then mine the structured data

Data Mining versus Text Mining

	Data Mining	Text Mining
Data Object	Numerical & categorical data	Textual data
Data structure	Structured	Unstructured & semi-structured
Space dimension	< tens of thousands	> tens of thousands
Methods	machine learning, statistic…	Data mining, information retrieval, NLP …
Maturity	Broad implementation since 1994	Broad implementation starting 2000
Market	10^5 analysts at large and mid size companies	10^8 analysts corporate workers and individual users

Text Mining Process – three steps



Text Mining Process

- Step 1: Establish the corpus
  - Collect all relevant unstructured data (e.g., textual documents, XML files, emails, Web pages, short notes, voice recordings…)
  - Digitize, standardize the collection (e.g., all in ASCII text files)
  - Place the collection in a common place (e.g., in a flat file, or in a directory as separate files)

Text Mining Process

- Step 2: Create the 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.

Document	computer	mouse	printer	monitor	keyboard
D1	0	3	1	5	0
D2	2	1	1	3	2
D3	1	0	3	0	6

Text Mining Process

- Step 2: Create the Document Term Matrix - Continued.
  - TDM is a sparse matrix. How can we reduce the dimensionality of the TDM?
    - Manual - a domain expert goes through it
    - Eliminate terms with very few occurrences in very few documents

Text Mining Process

- Step 3: Extract patterns/knowledge
  - Classification
  - Clustering
  - Association rules among the documents
  - And others…

Basic Text Mining Packages in R

The recommended text mining packages include:

- library(tm) - Framework for text mining
- library(SnowballC) - Provides wordStem() for stemming
- library(ggplot2) - Plots word frequencies
- library(wordcloud) - Create Word Cloud plots

Establish the Corpus of Text

- Use the appropriate tm source for the data you have
  - "DataframeSource”, "DirSource" , "URISource“, "VectorSource“, "XMLSource“
- Turn the merged document into a tm Corpus, using tm’s Corpus() function.
- **About corpus:** the main structure for managing documents in tm (Latin for “body”). It is a collection of electronically stored texts on which we would like to perform our analysis. Within each corpus, we may have separate articles. Corpora are R objects held fully in memory.

Example: Establish the Corpus of Text

- The script below illustrates extracting file content from a PDF file into corpus.

```
#the pdf file is stored in the work directory
pdf.loc <- file.path(getwd(), "somefile.pdf")
#create a uniform resource identifier source
#A uniform resource identifier source interprets each URI as a document.
cname <- URISource(pdf.loc)
corpus.pdf <- Corpus(cname, readerControl=list(reader=readPDF))
inspect(corpus.pdf)
corpus.pdf[[1]]$content #to view the content
corpus.pdf[[1]]$meta
```

Creating and Exploring the Document-Term Matrix

- Creating Document Term Matrix from corpus

```
dtm <- DocumentTermMatrix(corpus.pdf)
```

- Find the most frequent terms in the dtm

```
freq <- colSums(as.matrix(dtm)) #Term frequencies
ord <- order(freq) #Ordering the frequencies
freq[tail(ord)] #Most frequent terms
#get the terms that appear at least 10 times
findFreqTerms(dtm, lowfreq = 10)
```

Text Preprocessing

- We can see that the frequent terms discovered are not so interesting. So we will perform some preprocessing steps on the corpus before creating dtm.
- Pre-processing transformations such as:
  - Converting the text to lower case
  - Removing numbers and punctuation
  - Removing stop words
    - Stop words examples: the, and, a, an, is, of, that
  - Stemming and identifying synonyms
    - Stemming: reducing words to their root form, such as ‘fishes’ -> ‘fish’, ‘played’ -> ‘play’…

Text Preprocessing

- The basic transforms available within tm package can be seen with:
- >getTransformations()

  [1]"removeNumbers""removePunctuation""removeWords""stemDocument"

  [5]"stripWhitespace"
- These transformations are applied with the function "*tm_map()*".
- **Custom transformations** can be implemented by creating an *R* function wrapped within *content_transformer()* and then applying it to the corpus by passing it to *tm_map()*.

Preprocessing Steps

```
# Convert to lower case
corpus.tranf <- tm_map(corpus.pdf, content_transformer(tolower))
# Remove Punctuation
corpus.tranf <- tm_map(corpus.tranf, removePunctuation)
# Remove stop words
corpus.tranf <- tm_map(corpus.tranf, removeWords, stopwords("english"))
# specify your stopwords as a character vector
corpus.tranf <- tm_map(corpus.tranf, removeWords, c("type", "people"))
# Perform Stemmings
corpus.tranf <- tm_map(corpus.tranf, stemDocument, language = "english")
```

Word Cloud

- We can create a word cloud to present the most frequent words.

```
# Create a Word Cloud Plot
install.packages("wordcloud")
library(wordcloud)
wordcloud(names(freq), freq, min.freq=5,
                    colors=brewer.pal(6, "Dark2"))
```

Mining Text Data

- The goal is to learn from text data, the process is subdivided as follows:
  - Supervised learning (know categories)
    - Classification (Classification of news stories according to their content – ‘sport’, ‘politics’, ‘space’…)
  - Unsupervised learning (no knowledge of what categories are contained in data)
    - Clustering (grouping a set of emails when we don't know what they contain)

Text Clustering

- Given a set of documents, clustering is the task of coming up with a good grouping of the documents based on their contents.
- We can use any clustering algorithms discussed earlier.
- 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
  - Manhattan distance
  - Cosine distance (cosine similarity)

Cosine Similarity

Document	computer	mouse	printer	monitor	keyboard
D1	0	3	1	5	0
D2	2	1	1	3	2
D3	1	0	3	0	6

Cosine measure: If d1 and d2 are two vectors (e.g.,term-frequency vectors), then

$$\cos(d1, d2) = (d1 \cdot d2) / ||d1|| \, ||d2|| ,$$

where · indicates vector dot product, ||d||: the length of vector d

- The cosine similarity value is between 0 and 1. The closer the value is to 1, the more similar the two vectors are to each other.
- **Example:** the cosine similarity between D1 and D2 can be computed as follows:

  D1 · D2 = 0*2 + 3*1 + 1*1 + 5*3 + 0*2 = 19

  ||D1|| = sqrt(0^2+3^2+1^2+5^2+0^2) = 5.92 ||D2|| = 4.36

  cos(D1, D2) = 19 / (5.92 * 4.36) = 0.74

Text Clustering Using K-means

```
# text mining – clustering
#read all txt files in news folder of current working directory
dir = DirSource(paste(getwd(),"/news",sep=""))
corpus = Corpus(dir, readerControl=list(reader=readPlain))

# obtain dtm after preprocessing steps
dtm <- weightTfIdf(dtm)
dtm.matrix = as.matrix(dtm)

# normalize the vectors
norm_eucl <- function(m) m/apply(m, MARGIN=1,
                                 FUN=function(x) sum(x^2)^.5)
m_norm <- norm_eucl(dtm.matrix)

# cluster into 2 clusters
cl <- kmeans(m_norm, 2)
```

Notes About The Example

- Term frequency-inverse document frequency (tf-idf): How many times a given term appears in the document it belongs to is the TF (Term Frequency) part of TF-IDF. The higher the TF value is, the more important the term is for the document. However, if the given term appears in all the documents then it is not that important in order to identify the document. So we want to have a weighting system that would decrease the importance of a given term as the number of the documents it shows up increases. And this is the 'IDF (Inverse Document Frequency)' part of the 'TF-IDF'. So in a nutshell, TF-IDF is a weighting mechanism that calculates the importance of each term for each document by increasing the importance based on the term frequency while decreasing the importance based on the document frequency.
- See the following link on how exactly to calculate it: https://www.rdocumentation.org/packages/tm/versions/0.7-6/topics/weightTfIdf

Notes About The Example

- *Normalizing.* It is possible for two files to be similar even if one is very long and another very short. In that case, term frequencies will vary significantly, and the algorithm will fail to put such files in the same cluster. To avoid this confusion, we apply normalization before applying the algorithm. Normalization maps term frequency to a number between 0 and 1. Then the algorithm can use these normalized values.

```
norm_eucl <- function(m) m/apply(m, MARGIN=1,
                                 FUN=function(x) sum(x^2)^.5)
m_norm <- norm_eucl(dtm.matrix)
```

Text Classification

- Supervised learning
  - We know the groups (tags) into which we separate data.
  - The process of classification provide automated tagging.
- So typically the available data is split into several datasets.
  - Training dataset (we use 50% in this lecture.) –data for which we know the classification, used for training the algorithm.
  - Test dataset (50%) –data we analyze, to draw conclusions.

Text Classification Using KNN

- We have known several classification algorithms.
- Consider the KNN algorithm - the arguments of the function knn() are

  knn(train,test,cl,k)

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

Example: Text Classification Using KNN

```
# Populate eight documents and merge them
doc <- c(Doc1,Doc2,Doc3,Doc4,Doc5,Doc6,Doc7,Doc8)
corpus <- Corpus(VectorSource(doc))

# Preprocessing steps omitted
dtm <- as.matrix(DocumentTermMatrix(corpus)) # 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 = 3, prob=TRUE) # k-number of
                                                 neighbors considered
```

Notes About the Example

- The package "class" is used for implementing 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 consider 3 neighbors. The classification probability is returned for each test dataset.
  - prob.test<-knn(train.doc, test.doc, Tags, k = 3, prob=TRUE)

Analyzing the output of the knn()

- The classification probability is returned for each test dataset.
- Remember the test dataset contained:
  - doc3 related to cars.
  - doc4 related to cars.
  - doc7 related to space.
  - doc8 related to space.

```
> a <- c(3,4,7,8) #document ids
> b <- prob.test #predicts by the algorithm
> c <- attributes(prob.test)$prob #proportion of the votes for the winning class
> result <- data.frame(Doc=a, Predict=b,Prob=c)
> result
  Doc Predict      Prob
1   3    cars 0.6666667
2   4    cars 0.6666667
3   7   space 0.6666667
4   8   space 0.6666667
```

- This classification was 100% correct.