Data Wrangling and Data Analysis
Text Processing

Hakim Qahtan

Department of Information and Computing Sciences

Utrecht University

Topics for Today

- Shell scripting for text processing
- Preprocessing textual data
- Text Similarity
- Information extraction and retrieval from textual data

Shell Scripting for Text Processing

Make Your Life Easier by Shell Scripting

- Where?
  - UNIX-like systems, Mac OS, Windows PowerShell
- What?
  - Commands, e.g., ls, less, cut, grep, sed, ….
- Why?
  - Quick to use

Open Text File with any Extension

- To list the files in the current directory use: ls or ls –l
- If you want to open a specific file that contains textual data
  - In windows: search for the application that can read such file format
    - Examples: file.data, file.dat, file.txt, file.names, file.anything
  - In UNIX/LINUX like:
    - Use the command less
    - You can also use more, cat, vim, …

Summary Statistics of the Data File

- wc
  - wc file: returns the number of lines (including the empty lines), the number of words and the number of bytes for the file
  - wc –l file: returns the number of lines only (including the empty lines)
  - wc –w file: returns the number of words only
  - wc –c file: returns the number of bytes only

Get Several Lines of the Data File

- head -15 file (get the first 15 lines of file)
- sed -n ‘101,110p’ file (get the lines from 101 to 110 of file)
- sed -n ‘101p;111p;121p’ file (get the 101th, 111th, and 121th line of file)

Be careful when copying the command to fix the single quotation mark

Get Lines that Have Specific Keyword

- grep “Utrecht” article.txt (get the lines having ‘Utrecht’)
- grep -i “Utrecht” article.txt (get the lines having ‘Utrecht’ or ‘utrecht’, or ‘UTRECHT’)
- grep -n “Utrecht” art*.* (get the lines having ‘Utrecht’ and show line number in all files with names that start with art)
- grep --help for more option

Replace A with B

- sed ‘s/Female/Woman/’ file (replace ‘Female’ by ‘Woman’, only the first ‘Female’ in each line)
- sed ‘s/Female/Woman/g’ file (replace all ‘Female’ by ‘Woman’)
- sed ‘1d’ file (delete the first line of file)

- man sed for more option

Vim Commands for Text Processing

Vim is a highly configurable text editor built to make creating and changing any kind of text very efficient. It is included as "vi" with most UNIX systems and with Apple OS X.

Searching	
Command	**Explanation**
/computer	Search for the word “computer”; use / and then *n* to continue searching for next occurrences. Search forward for the word
?computer	Similar to /computer but the search is performed backward
/c[ao]n	Search for words that starts with can or con
/can\\|con	Search for the words that starts with can or con
:set ignorecase	Used to perform case insensitive search
etc.	..

Vim Commands for Text Processing (Cont.)

Replacing	
Command	**Explanation**
:%s/old/new/g	Replace all occurrences of old by new in file
:%s/onward/forward/gi	Replace onward by forward, case unsensitive
:%s/old/new/gc	Replace all occurrences with confirmation
:2,35s/old/new/g	Replace all occurrences between lines 2 and 35
:5,$s/old/new/g	Replace all occurrences from line 5 to EOF
:%s/^/hello/g	Replace the beginning of each line by hello
:%s/$/Harry/g	Replace the end of each line by Harry

Vim Commands for Text Processing (Cont.)

Replacing	
Command	**Explanation**
:%s/ $//g	Delete all white spaces at the end of each line
:%s/ //g	Delete all white spaces in the text
:%s/\t//g	Delete all tab spaces in the text
:g/^$/d	Delete all empty lines
:g/string/d	Delete all lines that contain the string
:v/string/d	Delete all lines that do not contain the string
:%s/^/\=printf(‘%-4d’, line(‘.’))	Insert the line number at the beginning of each line

awk more than sed and grep

- Sytax: awk '/search_pattern/ { action_to_take_on_matches; another_action; }' file
- Examples:
  - awk '{print;}’ file
  - awk ‘/Jiawei//Jianpei/‘ file
  - awk '{gsub("\t",""); print;}’
  - awk '{print $2,$5;}’ file
  - awk '$4 ~/Technology/’ file
- More on http://www.grymoire.com/Unix/Awk.html

Get-content in Windows

There will be an exam question on these commands (incl. following slides)

- Examples:
  - Get-content file
  - Get-Content .\file -TotalCount 5 # reads the first 5 lines
  - (Get-Content .\file -TotalCount 26)[-1] # reads the line 25
  - Get-Content .\file -Tail 1 # reads the last line
  - Get-Content -Path C:\Temp* -Filter *.log # use filter to display content of specific set of files

- More on https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-content?view=powershell-7

Get Several Columns of the Data File

- cut -d',' -f1 file | less

  (get the 1st column of file with delimiter ',')

- cut -d',' -f1-5 file > F_c1_c5

  (get 1st to 5th column of file with delimiter ',')

- cut -d',' -f1,5 file

  (get 1st and 5th column of file with delimiter ',')

Get Several Columns of the Data File

- cut -d',' -f2 file | sort | uniq -c

  (count the frequency of distinct values in column 2)
- cut -d',' -f2 file | sort | uniq | wc -l

  (count the number of distinct values in column 2)
- cut -d',' -f1 file | sort

  (sort the values in column 1 by increasing order)
- man sort, man uniq for more option

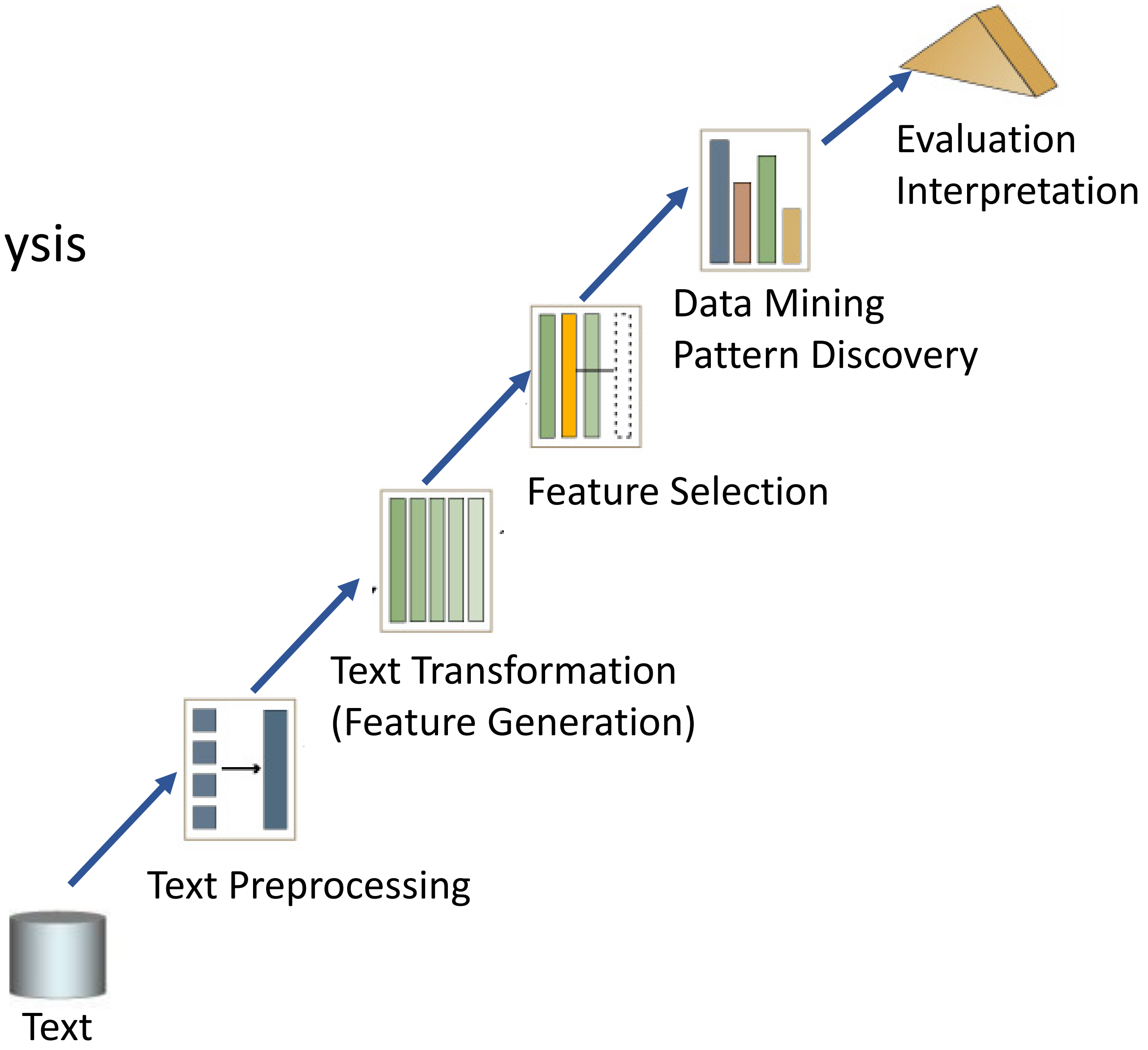
More Commands

- paste f1 f2 f3 > file # put columns together
- cat file1 >> file2 # write file1 into file2
- diff file1 file2 # compare file1 and file2
- Regular Expressions used with sed, grep, awk
  - grep “*ood” file # match good, wood, blood, etc. like % in SQL
  - grep “^good” file # match the line beginning with good
  - grep “good$” file # match the line ending with good

Text Preprocessing

Text Mining Process

- Text Preprocessing
  - Syntactic/semantic text analysis
- Features generation
  - Bags of words
- Feature selection
  - Simple counting
  - Statistics
- Text/data mining
- Analyzing the results



Know Your Data

- Item Code
- Item Name
- Item Price

Structured (attribute/Value pairs)

- Abstract
- Citations
- References
- Keywords
- etc.

Unstructured/Semi-structured

?

Structured Data

Text Preprocessing – Tokenization

- Convert a sentence into a sequence of tokens, i.e., words.
- Tokenizing English sentences is quite straightforward:
  - Just use spaces and punctuation as boundaries.
- For exceptions, use some heuristics
- Examples:
  - Tom’s ? a Possessive ending? or
  - Tom is? or
  - Tom has?

Text Preprocessing – Tokenization (Cont.)

- Convert a sentence into a sequence of tokens, i.e., words.
- Tokenizing English sentences is quite straightforward:
  - Just use spaces and punctuation as boundaries.
- Potentially many exceptions
- Examples:
  - Tom’s ? a Possessive ending? Or Tom is? Or Tom has?
  - Medicine is not nearly as evidence-based as we'd like
- The assumption that words are separated by non-letters is not always true
  - it is useful in practice
- The assumption that a word equals a token is not always true
  - **New York** is a **U.S.** city

Text Preprocessing – Stop Words Removal

- Many words are not informative and thus irrelevant for document representation:
  - the, and, a, an, is, of, that, may, off, be, by, for, from, it, will, was, with, were, …
- Typically about 400 to 500 such words
- For an application, an additional domain specific stop words list may be constructed
- Benefits of removing stop words
  - Reduce data file size: stop words accounts 20-30% of total word counts
  - Improve efficiency,
    - Stop words are not useful for searching or text mining
    - Stop words always have a large number of hits

because they're in all documents

Text Preprocessing – Stemming

- Reducing words to their root form
- A document may contain several occurrences of words like fish, fishes, fisher, fishing and fishers
- Different words share the same word stem and should be represented with its stem instead of the actual words.
- Benefits
  - Improving effectiveness of text mining: matching similar words
  - Reducing indexing size: combing words with same roots may
  - Reducing indexing size as much as 40-50%.

Text Preprocessing – Normalization

- Equivalence classing of terms, e.g., {USA, U.S.A.}, {dataset, data set}, {anti-discriminatory, antdiscriminatory}, ….
- Synonym list, e.g., {car, automobile}, {cat, kitty}, ….
- Name Entity Recognition, e.g., names of persons, organizations, locations, etc.
- Depending on the text files you are processing.

unifying the structure of the data/words so that we can find the same representations of it byw applying text processing - techniques used are text similarity

Text Similarity

Text Similarity

- People can express the same concept (or related concepts) in many different ways. For example,
  - “the plane leaves at 12pm” vs
  - “the flight departs at noon”
- Text similarity is a key component of Natural Language Processing
- If the user is looking for information about cats, we may want the NLP system to return documents that mention kittens even if the word “cat” is not in them.

Types Of Text Similarity

- Many types of text similarity exist:
  - Morphological similarity(e.g., respect-respectful)
  - Spelling similarity (e.g., theater-theatre)
  - Synonymy (e.g., talkative-chatty)
  - Homophony (e.g., raise-raze-rays)
  - Semantic similarity (e.g., cat-tabby)
  - Sentence similarity (e.g., paraphrases)
  - Document similarity (e.g., two news stories on same event)
  - Cross-lingual similarity (e.g., Dutch-Flemish-Afrikaans)

Morphological Similarity

- Words with the same root:
  - scan (base form)
  - scans, scanned, scanning (inflected forms)
  - scanner (derived forms, suffixes)
  - rescan (derived forms, prefixes)
  - rescanned (combinations)

Porter’s Stemming Method

- Porter’s stemming method is a rule-based (i.e. symbolic) algorithm introduced by Martin Porter in 1980
- The paper (“An algorithm for suffix stripping”) has been cited more than 10,000 times
- The input is an individual word. The word is then transformed in a series of steps to its stem
- The method is not always accurate
  - Utilizes suffix stripping, not addressing prefixes

Porter's Algorithm

- Example 1:
  - Input = computational
  - Output = comput
- Example 2:
  - Input = computer
  - Output = comput
- The two input words end up stemmed the same way
- Note: Stem is not(necessarily) the morphological root

Porter's Algorithm (Cont.)

- The measure of a word is an indication of the number of syllables in it
  - Each sequence of consonants is denoted by C
  - Each sequence of vowels is denoted as V
  - The initial C and the final V are optional
  - So, each word is represented as [C]VCVC ... [V], or [C](VC){m}[V], where m is its measure

initially a set of consonants that are after each other they all will be ONE C and not multiple C's

Examples of Measures

CV-> initial C aand final V are optional -> 0

- m=0: I, AAA, CNN, TO, GLEE

VC VC CVC CVC CVCV

- m=1: OR, EAST, BRICK, STREET, DOGMA

VCVC VCVC CVCVC -> initial C optional -> 2

- m=2: OPAL, EASTERN, DOGMAS

VCVCVC CVCVCVC

- m=3: EASTERNMOST, DOGMATIC
- STREET →[C]VC → m= 1

Porter's Algorithm

- The initial word is then checked against a sequence of ~60 transformation patterns, in order.
- An example pattern is:
  - (m>0) ATION -> ATE(e.g. medication -> medicate)
  - Note that this pattern matches medication and dedication, but not nation [m("n") == 0].
- Whenever a pattern matches, the word is transformed and the algorithm restarts from the beginning of the list of patterns with the transformed word.
- If no pattern matches, the algorithm stops and outputs the most recently transformed version of the word.

Examples

- Example 1:
  - Input = computational
    - Replace ational with ate: computate
    - Replace ate with ø: comput
  - Output = comput
- Example 2:
  - Input = computer
    - Replace er with ø: comput
  - Output = comput
- The two input words end up stemmed the same way
- Demo: http://text-processing.com/demo/stem/

Spelling Similarity

a VC is a set of vowels followed by a set of consonants

- Typos:
  - Brittany Spears -> Britney Spears
  - Catherine Hepburn -> Katharine Hepburn
  - Reciept -> receipt
- Variants in spelling:
  - Theater -> theatre   CVCVC - CVCV
  - Center -> Centre
  - Color -> Colour

Mississippi be m = 3. [c] vcvcvc[v]
because initial C and end V are optional
3 "sets" of VC

Computing Edit Distance

- Example: compute the edit distance between intention and execution

I	N	T	E	*	N	T	I	O	N
|	|	|	|	|	|	|	|	|	|
*	E	X	E	C	U	T	I	O	N
d	s	s		i	s				

- If each operation has cost of 1
  - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
  - Distance between them is 8

Computing Edit Distance Cont.)

- Dynamic programming: A tabular computation of D(n,m)
- Solving problems by combining solutions to subproblems.
- Bottom up
  - We compute D(i,j) for small i,j
  - And compute larger D(i,j) based on previously computed smaller values
  - i.e., compute D(i,j) for all i ($0 < i < n$) and j ($0 < j < m$)

Defining Minimum Edit Distance (Levenshtein)

- Initialization

$$D(i,0) = i$$

$$D(0,j) = j$$

- Recurrence Relation:

For each i = 1…M

For each j = 1…N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2 & \text{if } X(i) \neq Y(j) \\ 0 & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

D(N,M) is distance

Edit Distance Table – Example

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

9x9 comparisons

Edit Distance Table – Example (Cont.)

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

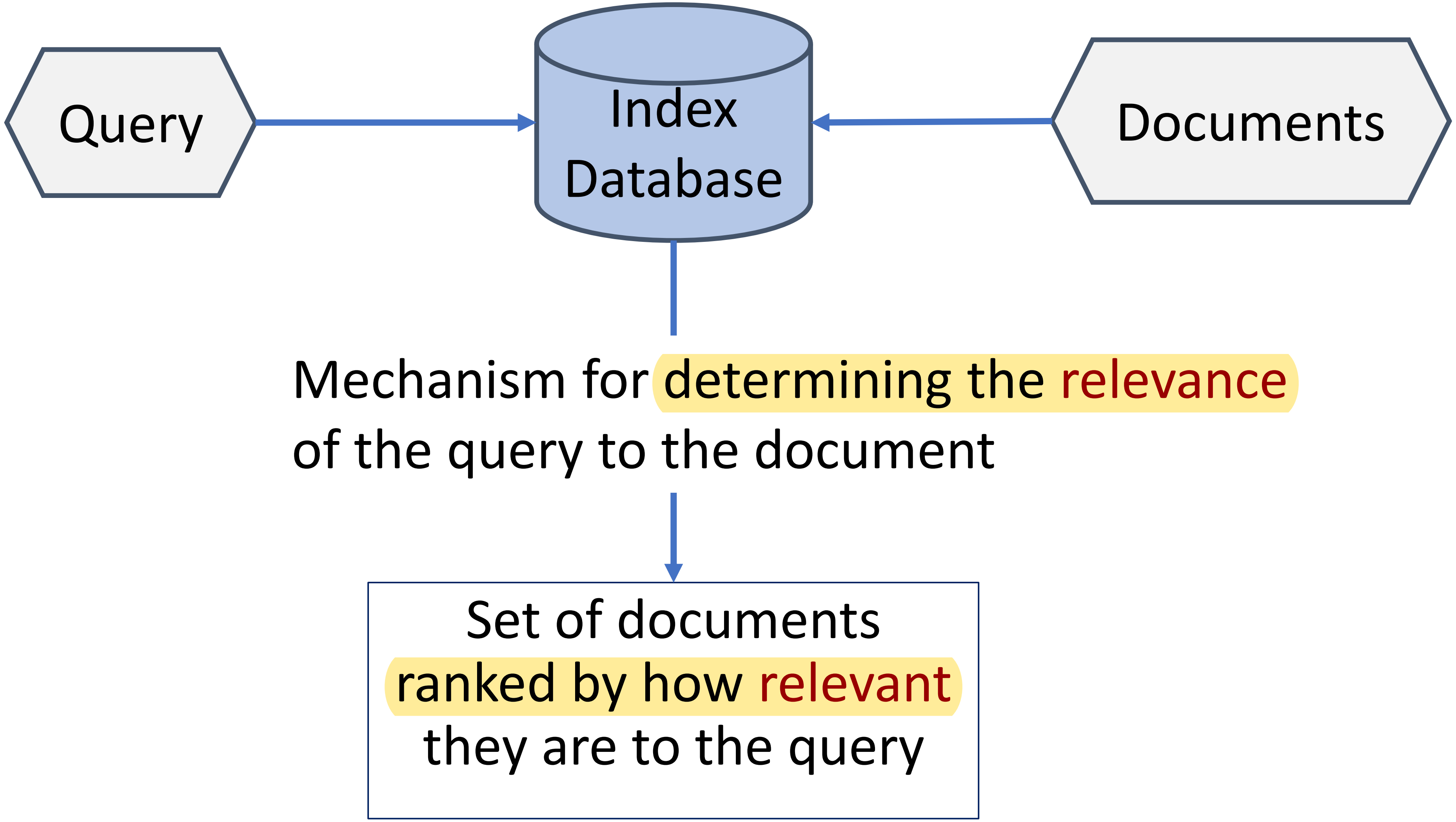
$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2 & \text{if } w1(i) \neq w2(j) \\ 0 & \text{if } w1(i) = w2(j) \end{cases} \end{cases}$$

Edit Distance Table – Example (Cont.)

N	9	8	9	10	11	12	11	10	9	8	
O	8	7	8	9	10	11	10	9	8	9	
I	7	6	7	8	9	10	9	8	9	10	
T	6	5	6	7	8	9	8	9	10	11	
N	5	4	5	6	7	8	9	10	11		10
E	4	3	4	5	6	7	8	9	10		9
T	3	4	5	6	7	8	9	8	9		8
N	2	3	4	5	6	7	8	7	8	7	
I	1	2	3	4	5	6	7	6	7	8	
#	0	1	2	3	4	5	6	7	8	9	
	#	E	X	E	C	U	T	I	O	N	

Information Extraction and Retrieval from Textual Data

Document Retrieval



Describing Documents

- Find important terms in a document remove stop words
- These terms can be matched to a search query match words from search query to words in document
- Which terms are important?
  - Terms with high frequency e.g. repeating "Computer Science"
    - Terms that occur in many documents are less distinctive and therefore less important
- Task: find terms with a high frequency within a document, but a low frequency in other documents

Describing Documents

- Assume a document with the following sentences
  - John sits inside. The cat walks inside. John can see the cat, but the cat cannot see him.
- The Document-Term Matrix is constructed as follows

words like "but, the" shouldn't be considered

Term	inside	him	John	the	cat	walks	but	can	cannot	see	sits
Frequency	2	1	2	3	3	1	1	1	1	2	1

- Term Frequency (TF) represents the frequency of the term in a specific document
- The underlying assumption: the higher the term frequency in a document, the more important it is for that document $tf(t, d) = c(t, d)$
- $c(t, d)$ – the number of occurrences of the term t in the document d

Feature Generation – Bag-of-Words

- Each document becomes a vector of terms
- Each term is a component (attribute) of the vector
  - the value of each component is the number of times the corresponding term occurs in the document

	team	coach	play	ball	score	game	won	lost	timeout	season
Doc. d_1	3	0	5	0	2	6	0	2	0	2
Doc. d_2	0	7	0	2	1	0	0	3	0	0
Doc. d_3	0	1	0	0	1	2	2	0	3	0

Feature Selection/Transformation

- Normalizing the document-term matrix

	team	coach	play	ball	score	game	won	lost	timeout	season
Doc. d_1	0.15	0	0.25	0	0.1	0.3	0	0.1	0	0.1
Doc. d_2	0	0.54	0	0.15	0.07	0	0	0.231	0	0
Doc. d_3	0	0.11	0	0	0.11	0.22	0.22	0	0.33	0

appx. 1

dviding frequency of each term over total numer of frequencies that exist in that document

Feature Selection/Transformation

- The underlying idea: assign higher weights to unusual terms, i.e., to terms that are not so common in the corpus
  - If a term occurs frequently in many documents it has less discriminatory power
- IDF is computed at the corpus level, and thus describes corpus as a whole, not individual documents

comparing it to other documents as well

- It is computed in the following way:

$$idf(t, d_i) = 1 + \log\left(\frac{N}{df(t)}\right)$$

total number of documents

number of documents containing the term

$df(t)$ = number of documents containing term t

N = total number of documents

Feature Selection/Transformation TF-IDF

high value IDF for rare words but also relatively high TF (term frequency)

- The underlying idea: value those terms that are not so common in the corpus (relatively high IDF), but still have same reasonable level of frequency (relatively high TF)
- General formula for computing TF-IDF

dash

$$TF - IDF(t, d_i) = tf(t, d_i) \times idf(t, d_i)$$

!!

$$TF - IDF(t, d_i) = tf(t, d_i) \times \log\left(\frac{N}{df(t)}\right)$$

term frequency

total number of documents

number of documents containing the term

TF

	team	coach	play	ball	score	game	won	lost	timeout	season
Doc. d_1	3	0	**5**	0	2	**6**	0	2	0	2
Doc. d_2	0	**7**	0	2	1	0	0	3	0	0
Doc. d_3	0	1	0	0	1	2	2	0	3	0

TF-IDF

	team	coach	play	ball	score	game	won	lost	timeout	season
Doc. d_1	**3.3**	0	**5.5**	0	0	2.43	0	0.81	0	2.2
Doc. d_2	0	2.84	0	2.2	0	0	0	1.22	0	0
Doc. d_3	0	0.41	0	0	0	0.81	2.2	0	3.3	0

A term is assumed to be “important” if it has a high TF and/or a high IDF

Estimating Similarity of Documents

- Key question: which metric to use for estimating the similarity of documents (i.e., vectors that represent documents)?
- The most well known and widely used metric is Cosine similarity

$$\cos(d_i, d_j) = V_i \times V_j \, / \, (||V_i|| \; ||V_j||)$$

check! dot product?

where V_i and V_j are vectors representing documents d_i and d_j

Cosine Similarity – Pros and Cons

- Advantages
  - Intuitive
  - Easy to implement
  - Empirically proven as highly effective
- Drawbacks
  - Based on the unrealistic assumption of words mutual independence
  - Tuning the model's parameters is often challenging and time consuming; this includes selection of method for:
    - Determining the terms' weights
    - Computing document (vector) similarity

Document Retrieval

- The Cosine similarity between d_i and d_j is defined as:

$$\cos(d_i, d_j) = \frac{\sum_{k=1}^{n} w(t_k, d_i) w(t_k, d_j)}{\|d_i\| \|d_j\|}$$

where d_i and d_j are the corresponding vectors in the document-term matrix and $\|d_i\|$ is the first norm of the document vector d_i, n is the number of terms. $t_1, t_2, \ldots, t_n$ are the terms in the matrix

$\|d_i\|$is computed as$\|d_i\| = \sqrt{\sum_{t=t_1}^{t_n} w^2(t, d_i)}$

$w(t_k, d_i)$ is the entry in the Document-Term Matrix at the column k and the row i

Document Retrieval – Example

TF

	team	coach	play	ball	score	game	won	lost	timeout	season
Doc. d_1	3	0	**5**	0	2	**6**	0	2	0	2
Doc. d_2	0	**7**	0	2	1	0	0	3	0	0
Doc. d_3	0	1	0	0	1	2	2	0	3	0

D1 3^2+5^2+2^2+6^2+2^2+2^2

- Let our query be Q = Coach and game
- We remove and as it is stop word
- We compute the cosine similarity between the query and each document

Document Retrieval – Example (Cont.)

TF

	team	coach	play	ball	score	game	won	lost	timeout	season
Q		1				1				
Doc. d_1	3	0	**5**	0	2	**6**	0	2	0	2
Doc. d_2	0	**7**	0	2	1	0	0	3	0	0
Doc. d_3	0	1	0	0	1	2	2	0	3	0

- Q = Coach and game
- $|Q| = \sqrt{\sum_{k=1}^{n} tf_{i,k}^2} = \sqrt{1^2 + 1^2} = \sqrt{2},$

$$|d_1| = \sqrt{82}$$
$$|d_2| = \sqrt{63}$$
$$|d_3| = \sqrt{19}$$

Document Retrieval – Example (Cont.)

TF

	team	coach	play	ball	score	game	won	lost	timeout	season
Q query	0	1	0	0	0	1	0	0	0	0
Doc. d_1	3	0	**5**	0	2	**6**	0	2	0	2
Doc. d_2	0	**7**	0	2	1	0	0	3	0	0
Doc. d_3	0	1	0	0	1	2	2	0	3	0

- Q = Coach and game
- $\cos(Q, d_1) = \frac{0+6}{\sqrt{2}\sqrt{82}} = 0.47$
- $\cos(Q, d_2) = \frac{7+0}{\sqrt{2}\sqrt{63}} = 0.62$
- $\cos(Q, d_3) = \frac{1+2}{\sqrt{2}\sqrt{19}} = 0.48$

cosine similarity is vector1 x vector2 = similarity between query and d1

The ranked results are: d_2, d_3, d_1

Exercise: redo the example using the TF-IDF

Reading Material

- Data Science at the command line
  - Chapters 3, 5
- Introduction to information retrieval
  - Chapter 2.2: Determining the vocabulary of terms
  - Chapter 6.2: Term frequency and weighting
  - Chapter 6.3: The vector space model for scoring
  - https://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf