Advanced Text Analysis for Business (IDS-566)

Lecture 2

Jan 26, 2018

Course Overview

- Instructor
  - Ehsan M. Ardehaly PhD, ehsan@uic.edu
  - Office hours: 4:45 - 5:45 pm F, BLC L270
  - Teacher assistant: 4:00 - 5:00 pm W, BLC L270
- Objectives:
  - Text mining
  - Applications for business decisions
  - Study of machine learning concepts
  - Design and implementation of text mining approaches

Course Overview

- Suggested text books:
  - Fundamentals of Predictive Text Mining (2nd Edition), Sholom M. Weiss, Nitin Indurkhya, Tong Zhang, 2015
  - Mining Text Data, Charu C. Aggarwal and ChengXiang Zhai, Springer, 2012
  - Mining of Massive Datasets, Jure Leskovec, Anand Rajaraman, Jeff Ullman

- Grading:
  - Final exam: 40%
  - 3 Assignments: 60% (3 x 20%)

Course Assignments

- Grade: 20%
- Loading textual data
- Building models
- Analysis
- Suggested programming language
  - Python 3
  - Scientific packages (e.g. scikit-learn)

Assignment policy

- Please read university regulations:
  - https://grad.uic.edu/university-regulations
- All assignment you turn must be done by you **alone**.
- The first violation will result in a failing grade for that assignment.
- The second will result in a failing grade for the course.
- **Late Submission Policy:**
  - 4% per hour
- **Grade dispute:**
  - Within 7 days of the receipt of the grade

Agenda

- Lexical analysis
  - Tokenization, Regular Expression
  - Unigram, Bigram, N-gram
  - Stop words
- Document to term matrix:
  - Vectorization
  - TF-IDF
- Sparse matrix:
  - LIL matrix, CSR matrix, CSC matrix
- Sentiment Analysis:
  - Lexicons

Lexical Analysis

- Text is unstructured
- Computer models need structure
- Solution:
  - Driving patterns from text
  - Creating structured data

Tokenization

- Sequence of characters → Sequence of terms

- Example:
  - Input: “Hello world!”

  - Word as a token:
    - Output: “hello”, “world”

  - Separate by space:
    - Output: “Hello”, “world!”

Tokenization

- Separating by special characters:
  - Space, tab, new-line, ...
  - What about Unicode?
  - Slow

Tokenization

- Separating by special characters:
  - Space, tab, new-line, ...
  - What about Unicode?
  - Slow

- Solution:
  - Using regular expression (regexp)
  - Python: package ‘re’

Regular Expression

- A sequence of characters that define a search pattern:
  - Find operation
  - Find and replace operation
  - Find and split operation

- Example:
  - [\n'()] → Match space, new-line, single quote, parenthesis
  - \w+ → Match multiple alphanumeric characters

Regular Expression use cases

- Tokenization to words:
  - \w\w+

- Recognizing URLs
  - http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+

- Recognizing hashtags (or mentions)
  - (?<=^|(?<=[^a-zA-Z0-9-_\.]))#([A-Za-z]+[A-Za-z0-9]+)

Unigram term

- Single word in sentence
- Without any space (or enter, tab, ...)
- Without special characters (!, ?, ...)
- Converting to lower case (optional)

- Sample: “this”, “is”, “i“

Bigram, n-gram

- Bigram: Contiguous sequence of two unigrams.
  - Example: “this is”, “is my”, “have you”, “text mining”

- N-gram: Contiguous sequence of N unigrams.
  - 3-gram: “text mining course”
  - 4-grams: “the president of us”
  - 5-grams: “the president of united states”

Tokenization to unigrams

- Input:
- “If I have enough money, I will go to Japan”

- Output:
- “if”, “i“, “have”, “enough”, “money”, “i“, “will”, “go”, “to”, “japan”

Tokenization to bigrams

- Input:
- “If I have enough money, I will go to Japan”

- Output:
- “if i”, “i have“, “have enough”, “enough money”, “money i”, “i will“, “will go”, “go to”, “to japan”

Unique unigram tokens:

- Input:
- “If I have enough money, I will go to Japan”

- Unique unigrams:
- “if”, “i“, “have”, “enough”, “money”, “will”, “go”, “to”, “japan”

Stop-words

- We often remove stop-words in tokenization.

- Some samples:
  - am, but, as, did, I, if, into, some, than, up, to, your, ...

Document

- Each sample of textual data:
  - Document 1: Shipment of gold damaged in a fire.
  - Document 2: Delivery of silver arrived in a silver truck.
  - Document 3: Shipment of gold arrived in a truck.
- Tokenization: unigram, lowercased, more than 1 letter.
  - arrived, damaged, delivery, fire, gold, in, of, shipment, silver, truck

Document to Term Matrix (DTM)

Each sample of textual data:

Document 1: Shipment of gold damaged in a fire.
Document 2: Delivery of silver arrived in a silver truck.
Document 3: Shipment of gold arrived in a truck.

Tokenization: unigram, lowercased, more than 1 letter.

arrived, damaged, delivery, fire, gold, in, of, shipment, silver, truck

Doc	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
D1	0	1	0	1	1	1	1	1	0	0
D2	1	0	1	0	0	1	1	0	2	1
D3	1	0	0	0	1	1	1	1	0	1

Count Vectorization vs. Binary Vectorization

Count vectorization (term frequency):

Doc	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
D1	0	1	0	1	1	1	1	1	0	0
D2	1	0	1	0	0	1	1	0	2	1
D3	1	0	0	0	1	1	1	1	0	1

Binary vectorization:

Doc	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
D1	0	1	0	1	1	1	1	1	0	0
D2	1	0	1	0	0	1	1	0	1	1
D3	1	0	0	0	1	1	1	1	0	1

Inverse Document Frequency

Term Frequency (TF):

Doc	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
D1	0	1	0	1	1	1	1	1	0	0
D2	1	0	1	0	0	1	1	0	2	1
D3	1	0	0	0	1	1	1	1	0	1

Inverse Document frequency (IDF) and log(IDF):

	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
IDF	3/2	3/1	3/1	3/1	3/2	3/3	3/3	3/2	3/1	3/2
log(IDF)	.41	1.1	1.1	1.1	.41	0	0	.41	1.1	.41

Smoothing IDF

$$idf = \log\frac{N}{DF} \xrightarrow{\text{Smoothing}} idf = 1 + \log\frac{N+1}{DF+1}$$

	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
IDF	3/2	3/1	3/1	3/1	3/2	3/3	3/3	3/2	3/1	3/2
log(IDF)	.41	1.1	1.1	1.1	.41	0	0	.41	1.1	.41
Smooth	1.29	1.69	1.69	1.69	1.29	1	1	1.29	1.69	1.29

TF-IDF

TF:

Doc	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
D1	0	1	0	1	1	1	1	1	0	0
D2	1	0	1	0	0	1	1	0	2	1
D3	1	0	0	0	1	1	1	1	0	1

IDF:

	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
Smooth	1.29	1.69	1.69	1.69	1.29	1	1	1.29	1.69	1.29

$$tf.idf = tf(1 + \log\frac{N+1}{DF+1})$$

Doc	arrived	damaged	delivery	fire	gold	in	of	shipment	silver	truck
D1	0	1.69	0	1.69	1.29	1	1	1.29	0	0
D2	1.29	0	1.69	0	0	1	1	0	3.39	1.29
D3	1.29	0	0	0	1.29	1	1	1.29	0	1.29

Normalizing TF-IDF

- Normalizing term vector
- Term vector: $[x_1 \quad x_2 \quad \cdots \quad x_n]$
- L1: $[\frac{x_1}{\sum x_i} \quad \frac{x_2}{\sum x_i} \quad \cdots \quad \frac{x_n}{\sum x_i}]$
- L2: $[\frac{{x_1}^2}{\sum {x_i}^2} \quad \frac{{x_2}^2}{\sum {x_i}^2} \quad \cdots \quad \frac{{x_n}^2}{\sum {x_i}^2}]$

Zipf's law

- Empirical law:
  - There are many rare words.
  - There are a small number of extremely frequent words.

- Let f_i be the frequency of the i-th most common term:
- $f_i \propto \frac{1}{i}$
- Or
- $f_i \propto i^{-b}$

Storing Document to Term Matrix

- Normal array:
  - Dense
  - Most of cells are zero
  - To much memory
  - To many matrix operations

Storing Document to Term Matrix

- Normal array:
  - Dense
  - Most of cells are zero
  - To much memory
  - To many matrix operations

- Solution:
  - Sparse Matrix

Sparse Matrix

Lists of lists (LIL)

- Good for incremental matrix construction

Compressed Sparse Row (CSR)

- Good for row operations

Compressed Sparse Column (CSC)

- Good for column operations

LIL Matrix

- Good for incremental matrix construction
- $\begin{bmatrix} 3 & 0 & 0 & 3 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$
- `Data:    [[3, 3], [2, 1], [1]]`
- `Rows:    [[0, 3], [1, 2], [1]]`

CSR Matrix

- Good for row operations
- $\begin{bmatrix} 3 & 0 & 0 & 3 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$
- `Data:          [3, 3, 2, 1, 1]`
- `Indices:       [0, 3, 1, 2, 1]`
- `Index pointer: [0, 2, 4, 5]`

CSC Matrix

- Good for columns operations
- $\begin{bmatrix} 3 & 0 & 0 & 3 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$
- `Data:          [3, 2, 1, 1, 3]`
- `Indices:       [0, 1, 2, 1, 0]`
- `Index pointer: [0, 1, 3, 4, 5]`

Example

- X[2, 4] = 5 ← Good for LIL matrix

- Z = X.dot(Y)
  - X: CSR matrix
  - Y: CSC matrix

Sentiment Analysis

Sentiment Analysis

- Classification of documents/messages by sentiment

- Applications:
  - Recommendation systems
  - Opinion mining
  - Customer relation management
  - Market sentiment

Approaches

- Lexicons
  - Dictionary base
  - General purpose
- Machine learning
  - Learning from data
  - Customized to the application

Lexicon approach

- Dictionary of words with score

- Example: AFINN http://neuro.imm.dtu.dk/wiki/AFINN
  - crying -2
  - excited 3
  - hell -4
  - winner 4
  - smiled 2

AFINN lexicons

- Congrats to @KingJames of the @cavs on joining the exclusive 30,000 Career Points Club!

- Congrats: 2
- Exclusive: 2

- Average score: 2 > 0

Challenges

- I do not dislike cabin cruisers. (Negation handling)
- Disliking watercraft is not really my thing. (Negation, inverted word order)
- I'd really truly love going out in this weather! (Possibly sarcastic)

https://en.wikipedia.org/wiki/Sentiment_analysis#Application_in_recommender_system