Text Processing

2020.9

References:
- KPC, DSAC(Data Scientist Academy & Certificate) Manual, 2019
- many internet sites

Text Data

- Typical Un-structured data
- Token-based
  - One-Hot Encoding
  - Bag of Words (BOW): Document-Term Matrix, Tfidf (term frequency inverse document frequency)
- Word Embedding
  - Word Vector

텍스트 분석의 개요

- 텍스트 분석의 목적
  - 텍스트의 의미를 알아내는 것
    - 글의 목적
    - 글쓴이의 성향(찬성/반대)
    - 기분(기쁨/슬픔/우울함 등)
    - 제품 피드백 등
- 텍스트 자체는 대표적인 비정형 데이터
- 의미를 추출하려면 비정형 데이터에서 정형화된 정보를 먼저 얻어야 함
- 텍스트 구문을 분석하여 의미를 파악하고 이것을 정량적으로 측정함

텍스트 분석

- SNS(트위터, 블로그, 페이스 북 등) 글을 분석
  - 소비자들의 반응, 감성, 트렌드를 파악
  - 개인별 마케팅, 상품 피드백을 분석하는데 사용
- 이메일, 웹사이트 댓글, 신문기사, 콜센터 상담기록, 도서 등을 분석
  - 글의 주요 내용을 파악
  - 문서의 특징을 추출
  - 유사한 글이나 저자를 찾는 작업 등을 수행
- 참고문헌이나 본문 인용의 관계를 통해서 문서간의 연계성, 전문가들의 인적 네트워크 등을 파악하는데도 사용
- 인공지능 스피커, 챗봇 등에서도 기본적으로 텍스트 분석이 필요

텍스트 분석 응용

- 챗봇 (Chatbot)
  - 사람과 대화하듯이 음성, 키보드 입력으로 대화를 나누는 인공지능 서비스
  - 챗봇의 유형
    - 미리 답을 준비하여 관련 질문이 나오면 해당 답을 하는 간단한 방식 (저수준)
    - 신경망을 사용하여 최적의 답을 찾아주는 방식 (고수준)

- QA 시스템
  - 질문을 하면 검색을 통해 적절한 답을 찾아주는(대답) 서비스
    - 대한민국의 수도는?
    - 오늘 날씨는?

- 자연어 처리
  - 언어 모델을 사용
    - 가장 자연스러운 다음 문장을 완성
    - 문장을 번역
    - 문서 요약, 주제 분석, 감성 분석 등을 수행

텍스트 표현 방법

- 사람이 단어나 문장의 의미를 인식하듯이 컴퓨터가 단어 자체 의미를 직접 파악할 수는 없다

- 텍스트 데이터 처리
  - 대표적인 비정형 데이터
  - 먼저 비정형 데이터인 글자로부터 정형화된 데이터인 수치 데이터로 변환

- 토큰화(tokenize) : 텍스트 분석의 첫 단계
  - 컴퓨터가 다루는 텍스트의 단위 : 토큰
    - 단어 (word) or 글자(character)
  - 주어진 텍스트를 토큰으로 나누는 작업

코퍼스 (말 뭉치)

- 말뭉치(corpus)
  - 데이터 분석에 주어진 전체 문서 집합
- 문서(document)
  - 코퍼스 내의 한 단위의 텍스트
  - 예) 하나의 블로그는 문서이고, 분석할 대상 블로그가 1천개이면 이 1천개 블로그 집합이 말뭉치
- 파싱(parsing)
  - 코퍼스에서 의미 있는 단어를 추출하는 작업

토큰화

- 토큰화 단위 (크게 3가지)
  - 단어(word)
    - 사람이 말을 이해할 때, 단어 단위로 인식하기 때문에 많은 연구에서 선호
  - 글자(character)
  - n-gram
- 단어 단위로 정보를 표현하는 과정에서 많은 정보를 잃게 된다
  - “정말”, “정말로”, “정말은” 등 단어
    - 같은 단어로 취급, 아니면 각각 다른 단어로 처리할지에 따라 분석 결과 달라짐
    - 같은 단어로 취급하기 위해 단어를 어근(stem)으로 변환하면 어미 변화를 무시하거나 조사를 무시하게 되어 텍스트에 들어 있던 정보를 잃게 된다
- 일반적으로 단어의 종류는 보통 10만 단어 이상 (언어마다 상이)
  - 신조어, 특수한 단어 포함하면 수십만개로 확대

토큰화

- 글자 단위로 토큰화를 하면 어근으로 변환할 때 정보를 잃는 문제를 피할 수 있다.
  - “정”, “말 “, “로”, “은 “ 등
- 음절 (발음 가능한 최소단위: 자음+모음) 단위 토큰의 수
  - 영어
    - 음절 단위의 토큰의 수가 적다 : 알파벳이 26글자, 모음(단모음+복모음)
  - 한글
    - 음절의 수가 수천 가지 이상

토큰화 – n-gram

- n-gram
  - n개의 연속된 단어를 하나로 취급하는 방법

- 예를 들어 “러시아 월드컵” 이라는 표현을 “러시아” 와 “월드컵” 두 개의 독립된 단어로만 취급하지 않고 두 단어로 구성된 하나의 토큰으로 취급
  - n=2 경우, bi-gram
  - 단어의 수가 매우 크게 증가
    - 실제로는 빈도 수가 최소한 몇 개 이상인 것만 다룬다

토큰화 – n-gram (예)

텍스트: “어제 러시아에 갔다가 러시아 월드컵을 관람했다”

단어토큰: { “어제”, “러시아”, “갔다”, “러시아”, “월드컵”, “관람” }

2-gram 토큰: { “어제 러시아”, “러시아 갔다”, “갔다 러시아”, “러시아 월드컵”, “월드컵 관람” }

토큰화

- n-gram을 허용하면 토큰화 대상의 수가 매우 크게 증가
  - 이론적으로는 10만개의 단어를 두 개 붙여서 나올 수 있는 경우의 수는 10만의 자승이 된다.
- 실제로는 빈도수가 최소한 몇 개 이상인 것만을 다룬다.
- 토큰화한 결과를 수치로 만드는 방법
  - 원핫(one-hot) 인코딩
  - BOW(단어모음) : 각 문장을 벡터로 표시 (ex: document-term matrix)
  - 단어벡터(Word Vector) 방법 : 단어를 벡터로 표시

원 핫 (One-hot) 인코딩

- **원 핫 인코딩**
  - **토큰**에 고유 번호를 배정
  - 모든 고유번호 위치의 한 컬럼만 1, 나머지 컬럼은 0인 **벡터**로 표시

텍스트: "어제 러시아에 갔다가 러시아 월드컵을 관람했다"

토큰 사전: { "어제" :0, "러시아" :1, "갔다" :2, "월드컵" :3, "관람" :4}

원핫 인코딩:

어제 = [1, 0, 0, 0, 0]
러시아 = [0, 1, 0, 0, 0]
갔다 = [0, 0, 1, 0, 0]
월드컵 = [0, 0, 0, 1, 0]
관람 = [0, 0, 0, 0, 1]

BOW (Bag of Word, 단어 모음)

- 원핫 인코딩 방식으로 단어(토큰)을 표현하면
  - 단어의 수가 적을 때에는 문제가 안되지만
  - 단어가 모두 10만개이면
    - 모든 단어가 항목이 10만개인 (0과 1로 구성된) 벡터로 표시
  - 주어진 텍스트가 20개의 단어로 구성되어 있다면
    - 20 x 100,000개 크기의 벡터가 필요
- 텍스트 분석은 "문장" 을 단위로 하는 경우가 많다
- 단어 모음(BOW) 방식 : 한 문장을 하나의 벡터로 만드는 방법
  - 한 문장을 단어 사전 크기의 벡터로 표현하고 그 문장에 들어 있는 단어의 컬럼만 1로, 단어가 없는 컬럼은 모두 0으로 표현
- 먼저 단어 사전을 만들고 각 문장에 어떤 단어가 들어 있는지 조사하여 해당 컬럼만 1로, 나머지는 0으로 코딩

BOW

- 단어 사전: { “어제” :0, “오늘” :1, “미국” :2, “러시아” :3,
  “갔다” :4, “축구” :5, “월드컵” :6, “올림픽” :7,
  “관람” :8, “나는” :9, …, “중국” :4999 }
- Text_1: “어제 러시아에 갔다가 러시아 월드컵을 관람했다” 를 BOW로 표현하면

문장번호	0	1	2	3	4	5	6	7	8	9	10	…	4998	4999
Text_1	1	0	0	1	1	0	1	0	1	0	0	0	0	0
Text_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Text_3	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Text_4	0	0	0	0	0	0	0	1	0	0	0	0	1	0
…														
Text_50	0	0	0	0	0	0	1	0	0	0	0	0	0	0

BOW : document-term matrix

- **문서-단어(document-term) 행렬 (document-term matrix)**
  - 문장 단위로 어떤 단어들이 있는지를 나타내는 BOW의 확장
  - 문서(document) 단위로 어떤 단어들이 있는지를 표현
  - 같은 단어가 여러번 등장하면 1 이상의 값을 갖는다

문서번호	0	1	2	3	4	5	6	7	8	9	10	...	4998	4999
Doc_1	1	2	3	1	4	0	2	0	1	3	0	0	0	0
Doc_2	0	0	0	0	2	0	0	0	0	0	0	0	2	0
Doc_3	0	0	0	1	0	0	0	0	3	0	0	0	0	1
Doc_4	4	0	0	0	0	0	0	1	0	0	4	0	1	0
...														
Doc_100	0	2	0	0	0	0	1	4	0	1	0	0	0	0

td-idf

- term frequency-inverse document frequency
- tf : 단어가 각 문서에서 발생한 빈도
- df(document frequency) : 그 단어가 등장한 '문서' 의 빈도
- 적은 문서에서 발견될수록 가치 있는 정보
- 많은 문서에 등장하는 단어일수록
  - 일반적인 단어
  - 이러한 공통적인 단어는 tf가 크다고 하여도 비중을 낮추어야 분석이 제대로 이루어질 수 있다.
- 따라서 단어가 특정 문서에만 나타나는 희소성을 반영하기 위해서 idf(df의 역수)를 tf에 곱한 값을 tf 대신 사용

td-idf (example)

- From http://www.datasciencecourse.org/notes/free_text/
  - Doc1 = "The goal of this lecture is to explain the basics of free text processing"
  - Doc2 = "The bag of words model is one such approach"
  - Doc3 = "Text processing via bag of words"

$$X = \begin{matrix} \text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & & \text{approach} \\ \begin{bmatrix} 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & \cdots & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & & 0 \end{bmatrix} & \begin{matrix} \text{Document 1} \\ \text{Document 2} \\ \text{Document 3} \end{matrix} \end{matrix}$$

td-idf (example)

- Term frequency
  - Counts of each word in a document
  - $tf_{i,j}$ = frequency of word j in document i
- Inverse document frequency
  - Term frequencies tend to be "overloaded" with very common words ("the", "is", "of", etc)
  - Idea if inverse document frequency weight words negatively in proportion to how often they occur in the entire set of documents

$$\text{idf}_j = \log\left(\frac{\#\text{ documents}}{\#\text{ documents with word } j}\right)$$

td-idf (example)

$$X = \begin{bmatrix} \text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & \text{approach} \\ 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \ldots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{matrix} \\ \text{Document 1} \\ \text{Document 2} \\ \text{Document 3} \end{matrix}$$

$$\text{idf}_{\text{of}} = \log\left(\frac{3}{3}\right) = 0$$

$$\text{idf}_{\text{is}} = \log\left(\frac{3}{2}\right) = 0.405$$

$$\text{idf}_{\text{goal}} = \log\left(\frac{3}{1}\right) = 1.098$$

td-idf (example)

- Term frequency inverse document frequency = $tf_{ij} \cdot idf_j$
- Just replace the entries in the X matrix with their TFIDF score.

$$X = \begin{matrix} \text{the} & \text{is} & \text{of} & \text{goal} \\ \end{matrix}$$
$$\begin{bmatrix} 0.8 & 0.4 & 0 & 1.1 \\ 0.4 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \ldots$$

단어 임베딩

단어 임베딩의 정의

- 앞에서 소개한 세 가지 텍스트 코딩 방식인 원핫 인코딩, BOW(단어모음), 문서-단어 행렬방식은 단어마다 고유번호를 배정하여 사용
- 그러나 이 고유 번호 숫자에는 아무런 의미가 들어 있지 못하며 단지 인덱스의 성격만 갖는다.
- 단어를 인덱싱이 아니라, 의미 있는 숫자 들의 집합, 즉, 벡터로 표현하는 방법이 단어 임베딩 (Word Embeding)이다.

단어 벡터

- 단어 벡터
  - 각 단어를 50~300개 정도의 차원으로 구성된 벡터로 표현

    학교 = [0.23, 0.58, 0.97, … , 0.87, 0.95]

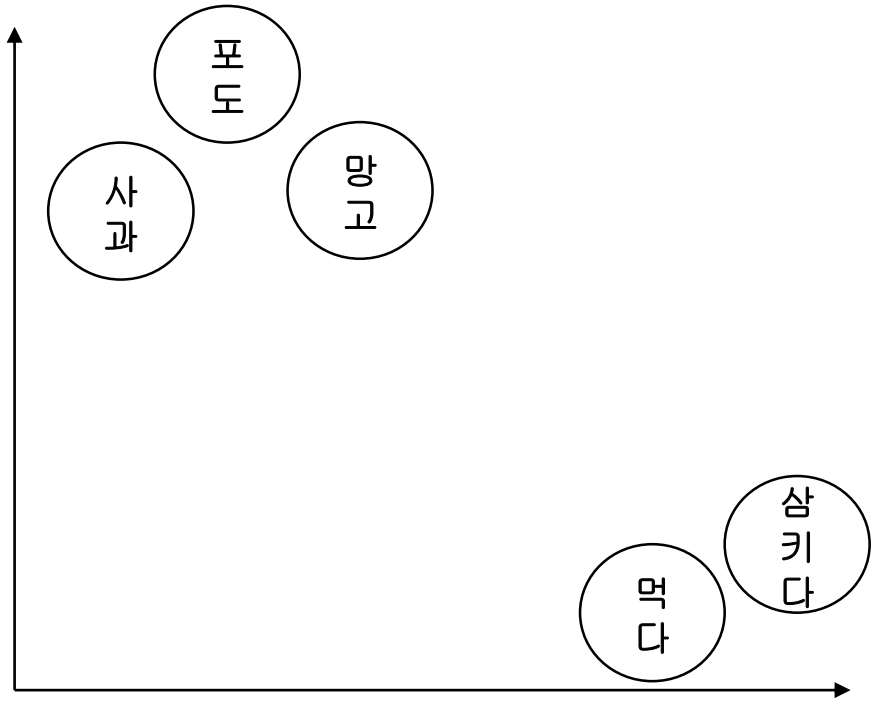
    바다 = [0.45, 0.37, 0.81, … , 0.22, 0.64]

- 단어 벡터를 사용하면
  - 각 단어들 사이의 "거리" 를 계산이 가능
  - 거리를 기반으로 유의어/반대어 등을 찾아낼 수 있다
  - 동물의 성별, 단수/복수, 동사/명사를 구분할 수도 있다
  - 그러나 각 벡터 값의 의미는 알 수 없다

단어 벡터

- 단어 벡터는 대형 말뭉치로부터 학습
  - 말뭉치의 문장들을 계속 입력하여 학습을 시키면 단어 벡터를 얻을 수 있다
  - 예를 들어 음식과 관련된 다음과 같은 문장들로 학습을 시키면 다음과 같은 단어 벡터를 얻을 수 있을 것이다.
  - 학습에 사용된 문장 예:

"나는 어제 바나나를 맛있게 먹었다"
"이 망고는 먹기가 힘들다"
"이 사과는 씹는 맛이 아주 좋다"
"바나나가 사과보다 맛있다"
"잘 씹어야 맛있게 먹을 수 있다"
…

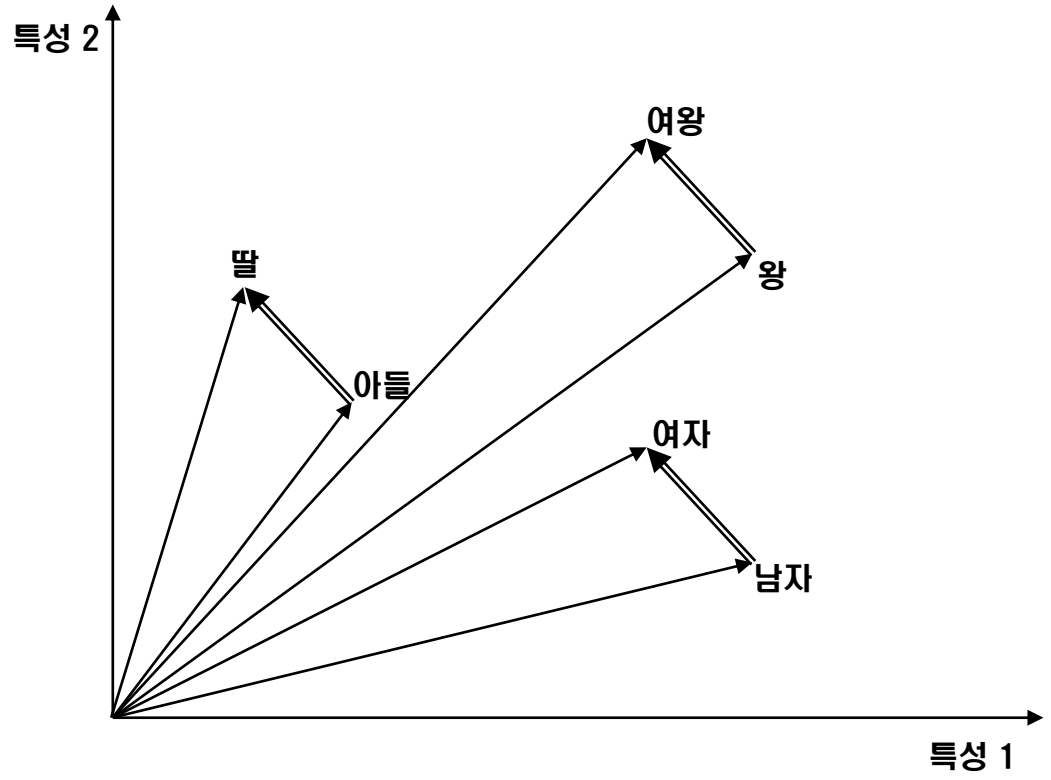


단어 벡터

- **이미 만들어져 있는 단어 벡터를 가져다 사용할 수도 있다.**
- **glove**
  - **2014년 스탠포드에서 만든** Global Vectors for Word Representations
  - **위키피디아 데이터로부터 학습**
  - **40만개 단어를 100차원으로 임베딩**
  - npl.stanford.edu/projects/glove **에서 다운로드**

단어 벡터

- 단어 벡터를 사용한 A:B = C: ?의 관계를 만족하는 ?를 찾을 수 있다.
  - 왕 : 여왕 = 아들 : ? → ? 부분 : 딸
  - 이러한 연산은 (B-A) 벡터, 즉 (왕 – 여왕) 성분을 구한 후 이를 벡터 C(아들)에 더하면 딸을 구할 수 있다.
  - 이들의 관계는 아래와 같다.



단어 벡터 생성

- **단어 벡터 만드는 과정을 소개**
  - **가장 널리 사용되는 라이브러리 : Gensim**
    - pip install gensim

```
from gensim.models.word2vec import Word2Vec
model = Word2Vec(sentense_list, min_count=1)
model.most_similar(positive="조선")

##
[('일본', 0.9953970909118652),
 ('관련', 0.9941188097000122),
 ('인물', 0.9938454031944275),
 ('러시아', 0.9931197166442871),
 ('주요', 0.9918481111526489),
 ('대원군', 0.9915156960487366),
...
```

문장 유사도 측정

- 단어의 유사도
  - 두 개의 문자열이 얼마나 다른지를 나타내는 편집 거리를 이용
- 편집 거리
  - 한 단어에서 다른 단어로 바꿀 때 필요한 최소한의 편집 행동의 횟수
  - 편집 행동
    - 글자를 추가, 제거, 변경
- 두 문장의 편집 거리 계산
  - NLTK 라이브러리를 활용

형태소 분석

- 단어 구분
  - 영어
    - 단어들이 대부분 스페이스로 구분, 단어 구분이 어렵지 않다
    - 예) I am a boy
  - 한글
    - 스페이스로 나눠진 단어가 조사를 포함하거나 복합명사인 경우 등이 있어 품사를 구분하는 작업이 영어처럼 간단하지 않다
    - 예) 나는 소년이다
      - 단어 구분 : '나는', '소년이다'
      - 추가적인 형태소 분석 : '나 ', '는', '소년 ', '이다 '
- 형태소 분석(morphological analysis)
  - 한글 문장을 처리하려면 단어를 다시 더 작은 단위인 형태소로 나누는 절차가 필요

형태소 분석기

- 형태소 분석기 - 한글
  - Hannanum (한나눔) : KAIST
  - Kkma (꼬꼬마) : 서울대
  - Komoran (코모란) : Shineware
  - Mecab (메카브)
    - 일본어용 형태소 분석기를 한국어를 사용할 수 있도록 수정
  - Okt (Open-korean-text) : twitter 개발
    - Twitter