Chapter1. 텍스트마이닝의 개요와 NLTK 패키지

1. 텍스트 마이닝의 개요

1) 데이터 마이닝과 텍스트 마이닝

① 데이터 마이닝
- 데이터에서 의미 있는 정보를 추출하는 기능
- 고급, 통계, 분석과 모델링 기법을 적용하여 데이터 안의 패턴과 관계를 찾아내는 과정

② 텍스트 마이닝
- 비정형 텍스트 데이터에서 분석 도구를 이용하여 패턴을 탐구하여 새롭고 의미 있는 정보를 찾아내는 과정 또는 기술
- 비정형 텍스트 데이터를 정량화 및 특징을 추출하는 과정이 요구됨
- 자연어 처리 기술에 기반을 둔 텍스트 데이터 가공 기술
- 모든 문서의 단어들을 하나하나의 변수로 보기 때문에 변수의 개수가 매우 많아지게 됨. 즉, 일반적인 데이터마이닝에 비해 데이터의 차원이 훨씬 커지게 됨

2) 실습예제

```
# 말뭉치(코퍼스, corpus) : 텍스트마이닝에 적용되는 텍스트 데이터의 집합
# 사전 처리(Preprocessing)
```

```
# 대소문자 통일
# 영문 텍스트 데이터는 대문자 또는 소문자로 변환하는 것이 좋음(보통 소문자로 변경)
s = “Hello World”
print(s.lower())    # 소문자로 변환
print(s.upper())    # 대문자로 변환
```

```
# 숫자, 문장부호, 특수문자 제거
# 단어가 아니므로 분석에 불필요한 경우가 대부분임
# 삭제할 경우 분석 결과가 왜곡된다고 생각될 때 남겨둘 필요도 있음
# 숫자의 경우 지우는 것이 일반적임
# 날짜, 수치, 백분율 등은 각각의 문장에서는 의미가 있지만, 전체 문서 집합에서는 크게
# 의미가 없는 경우가 많음
```

```
# 정규 표현식 모듈
import re
# 정규 표현식 숫자를 찾아내는 패턴, +반복
p = re.compile(“[0-9]+”)
# 문장에 포함된 숫자가 제거됨
result = p.sub(“”,“2019년 들어 대전 지역의 부동산 가격이 30% 하락했습니다.”)
print(result)
```

'년 들어 서울 지역의 부동산 가격이 % 하락했습니다.‘

```
# . , ? ! 등의 문장부호들은 삭제하는 것이 일반적
# 각 문장에서는 특수한 임무를 수행할 수 있으나 전체 말뭉치의 관점에서는 의미를 부여하기
# 어려운 경우가 대부분
# - () 등의 특수문자도 보통 삭제함
```

```
import re
def clean_text(input_data):
    # 텍스트에 포함된 숫자와 특수문자 제거
     p=re.compile("[0-9_!',.@#$%^&*]") # 정규 표현식 패턴
     result=p.sub("",input_data) # 문장에서 패턴을 찾아서 지움
     return result

txt = “2019년 들어 대전_지역의 부동산 가격이 30% 하락했습니다.!#$_$123”
print(txt)
print(clean_text(txt))
```

※ txt의 내용을 스크랩해서 다시 작업을 수행한다.

```
# 불용어 처리
# 불용어 : 빈번하게 사용되지만, 구체적인 의미를 찾기 어려운 단어들

# 영어의 경우 the, a, an 등의 관사는 많이 사용되지만, 텍스트 마이닝에서는 특별한 의미를
# 부여하기 힘든 경우가 많음
# NLTK : 파이썬에서 많이 사용되는 텍스트마이닝 패키지
# 언어별로 불용어 리스트 제공, 한국어는 지원하지 않음
# 한국어 불용어 리스트를 제공하는 패키지는 아직 없으며, 직접 만들거나 다른 분석가들이
# 작성한 리스트를 활용함
```

```
# 분석할 키워드 목록
words=["추석","연휴","민족","대이동","시작","늘어","교통량", “교통사고”, “특히”, “고장”,
        "자동차", “상당수”, "나타","것","기자"]

# 불용어 사전
stopwords=[“가다”, "늘어","나타","것","기자"]

# 키워드 목록에서 불용어를 제거한 리스트
[i for i in words if i not in stopwords]
```

['추석', '연휴', '민족', '대이동', '시작', '교통량', '자동차']

```
from nltk.corpus import stopwords

words=["chief","justice","roberts",",",".","1234","of","the",
        "thank","you","and","or","not"]

#print(stopwords.words("english"))

print([w for w in words if not w in stopwords.words("english")])
```

['chief', 'justice', 'roberts', ',', '.', '1234', 'thank']

```
# 불용어 사전 다운로드
import nltk #자연어 처리 패키지
nltk.download("stopwords")
```

```
from nltk.corpus import stopwords
words=["chief","justice","roberts",",",".","1234","of","the",
        "thank","you","and","or","not"]
#print(stopwords.words("english"))
print([w for w in words if not w in stopwords.words("english")])
```

['chief', 'justice', 'roberts', ',', '.', '1234', 'thank']

```
# 어근, 어미 정보 다운로드
import nltk
nltk.download('punkt')
```

True

```
# 어근 동일화 처리 : 비슷한 어근 처리(stemming)
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import word_tokenize
stm = PorterStemmer()
txt="cook cooker cooking cooks cookery"
#txt="python pythons pythoner Python Pythonweb Pythones"
words=word_tokenize(txt) # 문장에서 단어 구분
#print(words)
for w in words:
    print(stm.stem(w),end=" ")
```

cook cooker cook cook ingcookeri

```
# LancasterStemmer(랜커스터스테머) : PorterStemmer와 비슷하지만 좀더 나은 성능
from nltk.stem.lancaster import LancasterStemmer
stm=LancasterStemmer()
txt="cook cooker cooking cooks cookery"
#txt="python pythons pythoner Python Pythonweb Pythones"
words=word_tokenize(txt)
for w in words:
    print(stm.stem(w),end=" ")
```

cook cook cook cook cookery

```
# Porter나 Lancaster가 처리하지 못할 때는
# RegexpStemmer를 사용하여 특정한 표현식을 일괄적으로 제거함

from nltk.stem.regexp import RegexpStemmer
stm=RegexpStemmer("ing") # 문장에 포함된 ing를 제거
print(stm.stem("cooking"))
print(stm.stem("cookery"))
print(stm.stem("ingcook"))
```

위에 있는 Porter나 Lancaster에 마지막 cookery앞에 ing를 붙인다면 제거가 되지 않는다. 이럴 때 정규 표현식을 이용한다.

```
stm=RegexpStemmer("python") #문장에 포함된 python을 제거
txt="pythoning pythons Python pythoners pythoned"
words=word_tokenize(txt)
for w in words:
    print(stm.stem(w),end=" ")
```

ing s Python ers ed

```
# N-gram : n번 연이어 등장하는 단어들의 연쇄
# 2회 바이그램, 3회 트라이그램, 보편적으로 영어에만 적용되며, 바이그램이 주로 사용됨
```

```
# 바이그램(2글자) : 문자열의 끝에서 한 글자 앞까지만 반복함
txt="Hello"
for i in range(len(txt)-1):
    # 현재 문자와 그다음 문자 출력
    print(txt[i], txt[i+1], sep="")
```

He
el
ll
lo

```
# 트라이그램(3글자)
txt="Hello"
for i in range(len(txt)-2):
    print(txt[i], txt[i+1], txt[i+2], sep="")
```

Hel
ell
llo

```
# 단어 단위
txt="this is python script"
words=txt.split() # 공백을 기준으로 문자열을 나누어 리스트로 저장
print(words)
for i in range(len(words)-1):
    print(words[i],words[i+1])
```

['this', 'is', 'python', 'script']
this is
is python
python script

```
# zip 함수를 이용한 n-gram
txt="hello"
print(txt[1:])
two_gram=zip(txt, txt[1:])
print(list(two_gram))
for i in two_gram:
    print(i[0], i[1], sep="")
```

ello
[('h', 'e'), ('e', 'l'), ('l', 'l'), ('l', 'o')]

```
txt="this is python script"
words=txt.split()
list(zip(words, words[1:]))
```

[('this', 'is'), ('is', 'python'), ('python', 'script')]

```
from nltk import ngrams

sentence="I love you. Good morning. Good bye."
grams=ngrams(sentence.split(), 2) # two-gram(바이그램)
for gram in grams:
    print(gram, end=" ")
```

('I', 'love') ('love', 'you.') ('you.', 'Good') ('Good', 'morning.') ('morning.', 'Good') ('Good', 'bye.')

```
from nltk import ngrams

sentence="I love you. Good morning. Good bye."
grams=ngrams(sentence.split(), 3) # 트라이그램
for gram in grams:
    print(gram, end=" ")
```

('I', 'love', 'you.') ('love', 'you.', 'Good') ('you.', 'Good', 'morning.') ('Good', 'morning.', 'Good') ('morning.', 'Good', 'bye.')

```
# 품사 분석
# pos 태깅(Part-of-Speach)
# 모든 언어에 명사, 동사, 형용사, 부사는 공통적으로 존재함
```

```
# 한나눔 패키지
# pip install konlpy
from konlpy.tag import Hannanum
han = Hannanum()
txt="""
이날 오전 10시 26분 현재 아시아 주요국 주가는 전날에 이어 일제히 하락하며 장을 시작했다. 중국 상하이종합지수는 전날보다 1.58% 떨어진 2,776.99를 기록했다.
선전종합지수는 1,488.91로 1.87% 하락했다.
홍콩 항셍지수와 대만 자취안 지수는 각각 2.60%와 1.66% 떨어졌다.
일본 닛케이 225 지수는 전날 1.74% 하락 마감한 데 이어 현재 2.05% 떨어진 20,295.29를 기록했다.
"""
# 형태소 분석
print(han.morphs(txt))
```

```
print(han.nouns(txt))      # 명사 추출
```

```
# 형태소와 품사
print(han.pos(txt))
```

```
# 꼬꼬마 패키지
from konlpy.tag import Kkma
kkm=Kkma() #꼬꼬마 분석기
print(kkm.morphs(txt)) # 형태소 분석
```

```
print(kkm.nouns(txt))      # 명사 추출
```

```
print(kkm.pos(txt))
```

```
# 트위터 분석기
from konlpy.tag import Twitter
twit=Twitter()
print(twit.morphs(txt)) #형태소 분석
```

```
print(twit.phrases(txt)) #어구 추출 N-gram 비슷하게 추출해준다.
```

```
# 영어 품사 분석을 위한 패키지 다운로드
import nltk
nltk.download('averaged_perceptron_tagger')
```

```
# 영어 품사 분석
from nltk import pos_tag
a="I love you."
tags=pos_tag(a.split()) #단어 단위로 구분하여 품사 태깅
print(tags)
```

[('I', 'PRP'), ('love', 'VBP'), ('you.', 'RB')]

```
%matplotlib inline
import nltk
import urllib
import re
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"]=(20,16) #그래프 가로,세로 사이즈
plt.rcParams["font.size"]=15 #폰트 사이즈

res=urllib.request.urlopen("http://python.org") # url에 접속
html=res.read() # 원격 리소스를 읽어옴

tokens=re.split("₩W+",html.decode("utf-8")) # 특수문자 제거,인코딩설정
# print(tokens)
clean=BeautifulSoup(html,"html.parser").get_text() # 텍스트 내용
tokens=[token for token in clean.split()] # 단어 리스트
stop=set(stopwords.words("english")) # 불용어 사전
# 글자수가 1보다 크고 불용어가 아닌 단어들
clean_tokens=[token for token in tokens
   if len(token.lower())>1 and (token.lower() not in stop)]
tagged=nltk.pos_tag(clean_tokens) # 품사 태깅
# 명사와 고유명사만 선택
allnoun=[word for word,pos in tagged if pos in ["NN","NNP"]]
# print(allnoun)
# 단어의 출현빈도를 그래프로 출력
freq_result=nltk.FreqDist(allnoun)
freq_result.plot(50,cumulative=False)
```

2. NLTK 패키지

1) 실습예제

```
# 말뭉치(corpus) : 자연어 분석 작업을 위해 만든 문서 집합
```

```
import nltk
# NLTK 패키지에서 제공하는 샘플 말뭉치 다운로드, 시간이 오래 걸림
# 샘플 말뭉치 다운로드
nltk.download("book", quiet=True)
from nltk.book import *
```

```
# 저작권이 만료된 문학작품이 포함된 말뭉치 로딩
nltk.corpus.gutenberg.fileids()
```

```
# 제인 오스틴의 엠마 문서
emma_raw=nltk.corpus.gutenberg.raw("austen-emma.txt")
print(emma_raw[:500])
```

```
# 자연어 문서를 분석하기 위해서는 우선 긴 문자열을 분석을 위한 작은 단위로 나누어야
  한다.
# 이 문자열 단위를 토큰(token)이라고 하고
# 이렇게 문자열을 토큰으로 나누는 작업을 토큰 생성(tokenizing)이라고 함
# 영문의 경우에는 문장, 단어 등을 토큰으로 사용하거나 정규 표현식을 쓸 수 있다.
# 문자열을 토큰으로 분리하는 함수를 토큰 생성 함수(tokenizer)라고 한다.
# 토큰 생성 함수는 문자열을 입력받아 토큰 문자열의 리스트를 출력한다.
```

```
# 문장 단위로 토크나이징
from nltk.tokenize import sent_tokenize
print(sent_tokenize(emma_raw[:1000])[3:5])
```

```
# 단어 단위로 토크나이징
from nltk.tokenize import word_tokenize
print(word_tokenize(emma_raw[50:100])) # 50 ~ 99 사이의 단어만 이용
```

```
# 형태소 : 일정한 의미가 있는 가장 작은 말의 단위
# 형태소 분석(morphelogical analysis) : 단어로부터 어근, 접두사, 접미사, 동사 등 다양한
# 언어적 속성을 파악하고, 이를 이용하여 형태소를 찾아내거나 처리하는 작업
# 어간추출(stemming), 원형복원(lemmatizing), 품사부착(Part-of-tagging)
```

```
# 어간 추출(stemming) : 단어의 접미사나 어미를 제거
# 어간 추출법은 단순히 어미를 제거할 뿐이므로 단어의 원형을 정확히 찾아주지 않는다.
```

```
from nltk.stem import PorterStemmer, LancasterStemmer

st1=PorterStemmer()
st2=LancasterStemmer()

words=["fly","flies","flying","flew","flown"]

# 어간 추출(어미를 제거하는 작업)
print( [st1.stem(w) for w in words ])
print( [st2.stem(w) for w in words ])
```

```
# 원형복원 : 같은 의미가 있는 여러 단어를 사전형으로 통일하는 작업
# 동사를 지정하는 경우 좀 더 정확한 원형을 찾을 수 있다.
from nltk.stem import WordNetLemmatizer

lm=WordNetLemmatizer()
words=["fly","flies","flying","flew","flown"]
[lm.lemmatize(w,pos="v") for w in words] # 동사원형을 복원
```

```
# 각 단어의 사용 빈도를 그래프로 출력
%matplotlib inline
from nltk import Text
import matplotlib.pyplot as plt
# 정규 표현식을 이용한 어근 추출기
from nltk.tokenize import RegexpTokenizer
#print(retokenize.tokenize(emma_raw))

retokenize=RegexpTokenizer("[₩w]+") # 특수문자 제거
retokenize.tokenize(emma_raw[50:100]) # 토크나이징

text=Text(retokenize.tokenize(emma_raw))
plt.rcParams["figure.figsize"]=(15,10)
plt.rcParams["font.size"]=15
text.plot(20)
plt.show()
```

```
# 단어가 사용된 위치를 시각화
# 소설 엠마의 각 등장인물에 대해 적용
text.dispersion_plot([“Emma”, “Knightley”, “Frank”, “Jane”, “Harriet”, “Robert”])
```

```
# 단어가 사용된 위치를 표시
# 해당 단어의 앞뒤에 사용된 단어
text.concordance("Emma")
```

```
# 같은 문맥에서 주어진 단어 대신 사용된 횟수가 높은 단어들
text.similar("Emma")
```

```
# 불용어 제거
from nltk import FreqDist
from nltk.tag import pos_tag

# 불용어 리스트
stopwords=["Mr.","Mrs.","Miss","Mr","Mrs","Dear"]
emma_tokens=pos_tag(retokenize.tokenize(emma_raw)) # 품사 태깅

# NNP(고유대명사)이면서 필요 없는 단어(stop words)는 제거
names_list=[ t[0] for t in emma_tokens if t[1] == "NNP"
             and t[0] not in stopwords ]  # print(names_list)
# FreqDist : 문서에 사용된 단어(토큰)의 사용빈도 정보를 담는 클래스
# Emma 말뭉치에서 사람의 이름만 모아서 FreqDist 클래스 객체 생성
fd_names=FreqDist(names_list)
```

```
# 전체단어수, 키워드의 출현횟수, 출현비율
fd_names.N(), fd_names["Donwell"], fd_names.freq("Donwell")
```

```
# most_common 메서드를 사용하면 가장 출현 횟수가 높은 단어를 찾는다.
fd_names.most_common(5) # 가장 출현 빈도가 높은 단어 5개
```

```
#pip install wordcloud
from wordcloud import WordCloud

wc=WordCloud(width=1000,height=600,background_color="white",
             random_state=0)
plt.imshow(wc.generate_from_frequencies(fd_names))
plt.show()
```