Chapter2. 단어빈도분석과 군집분석

1. 단어 빈도 분석

1) 실습예제

```
# 단어 빈도 분석 : 전체 문서 또는 문서별 단어의 출현 빈도
```

```
# 트럼프 연설문 파일 로딩
f=open("d:/data/text/trumph.txt","r") # 읽기 모드로 파일 오픈
lines=f.readlines()[0] # 라인별로 읽어서 리스트로 저장
# print(lines)
f.close() #파일 닫기
lines[0:100]
```

' Chief Justice Roberts, President Carter, President Clinton, President Bush, President Obama, fellow'

```
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
tokenizer=RegexpTokenizer("[₩w]+") # 숫자, 특수문자 제거
stop_words=stopwords.words("english") # 영어 불용어 사전
# print(stop_words)
words=lines.lower() # 소문자로 변환
tokens=tokenizer.tokenize(words) # 단어 단위로 토근화
# print(tokens)
# 불용어 제거
stopped_tokens=[i for i in list((tokens))
                if not i in stop_words]
# 길이가 1인 단어들 제거
stopped_tokens2=[i for i in stopped_tokens if len(i)>1]
#stopped_tokens2 # 최종적으로 추출된 내용
```

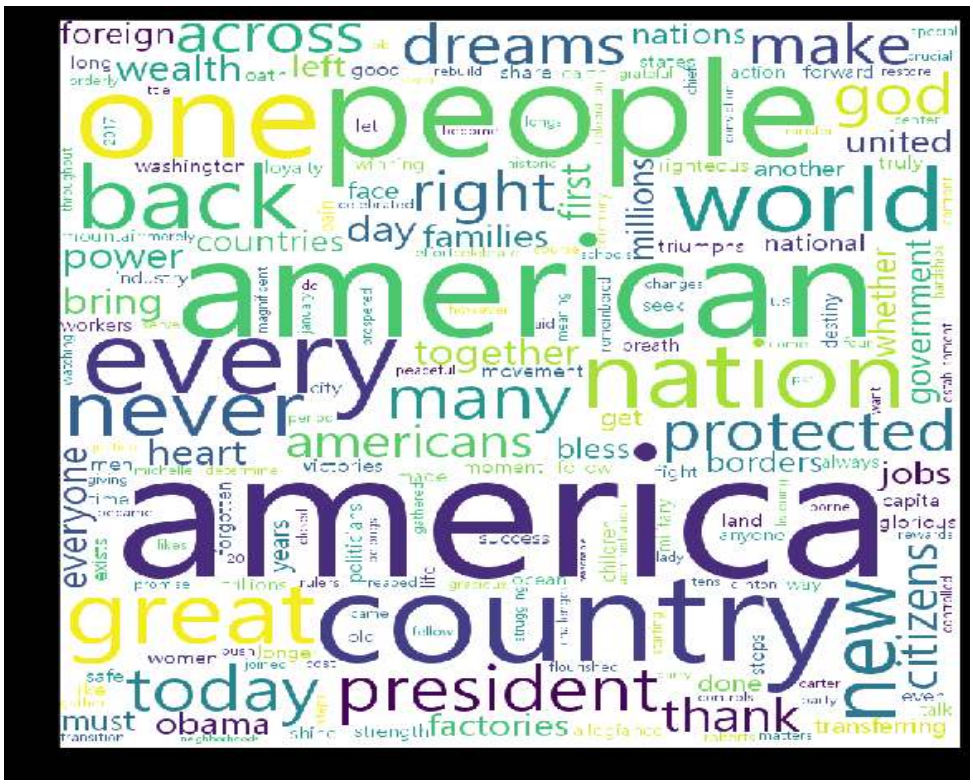
```
import pandas as pd
# 연설문에 포함된 단어들의 출현 빈도
pd.Series(stopped_tokens2).value_counts()
```

```
# 한글 텍스트 파일 분석
f=open("d:/data/text/news1.txt","r") # 파일을 읽기 모드로 오픈(내용이 있는 페이지 카피)
lines=f.readlines() # 라인별로 읽어서 리스트에 저장
# print(lines)
f.close() # 파일 닫기

# 한나눔 형태소 분석기
from konlpy.tag import Hannanum
han=Hannanum()

temp=[]
for i in range(len(lines)):
    temp.append(han.nouns(lines[i])) # 명사만 추출

# print(temp)

# 중첩 리스트를 하나의 리스트로 변환하는 함수
def flatten(l):
    flatList=[]
    for elem in l:
        if type(elem) == list: # 자료형이 리스트 타입이면
            for e in elem:
                flatList.append(e) # 리스트에 요소 추가
        else:
            flatList.append(elem)
    return flatList

word_list=flatten(temp)
word_list
```

※ 웹사이트에서 뉴스 내용을 스크랩한 후 news1.txt 파일로 저장한 후 작업을 수행한다.

```
# 2글자 이상인 단어만 추출
word_list=pd.Series([x for x in word_list if len(x) > 1])
# word_list[:10]

# 단어별 출현 빈도가 높은 순으로 출력
word_list.value_counts().head(10)
```

```
# 트럼프 연설문을 워드클라우드로 출력
%matplotlib inline
from wordcloud import WordCloud
from collections import Counter
import matplotlib.pyplot as plt

wordcloud=WordCloud(
    font_path="c:/windows/fonts/malgun.ttf",
    width=800, height=800, background_color="white"
)
count=Counter(stopped_tokens2) # 단어별 출현횟수
# print(count)
wordcloud=wordcloud.generate_from_frequencies(count) # 워드클라우드 객체

def to_array(self):
    return np.array(self.to_image()) # 넘파이 배열로 변환 - 플로팅을 위해

array=wordcloud.to_array()
fig=plt.figure(figsize=(10,10))
plt.imshow(array)
plt.show()
```



```
# 한글 뉴스를 워드클라우드로 출력
# 단어의 출현빈도를 계산
count=Counter(word_list)
# print(count)
# 워드클라우드 생성
wordcloud=wordcloud.generate_from_frequencies(count)
# 플로팅을 위하여 넘파이 배열로 변환
array=wordcloud.to_array()
fig=plt.figure(figsize=(10,10))
plt.imshow(array)
plt.show()
```

2. 군집분석

1) 텍스트 군집분석(Kmeans)

```
# 텍스트 분할 군집
# TF-IDF : 여러 문서로 이루어진 문서 군에서 어떤 단어가 특정 문서 내에서 얼마나 중요한
#          것인지를 나타내는 통계적 수치
# TF(Term Frequency) : 특정한 단어가 문서 내에서 얼마나 자주 등장하는지를 나타낸 값
# DF(Document Ffrequency) : 해당 단어가 문서 군내에서 얼마나 자주 사용되는지 나타내는
                            지표, 단어가 자주 등장할수록 중요한 단어가 아니라는 의미
# 예를 들어 신문기사에서 xx기자 라는 단어는 단어 자체가 중요한 단어는 아님
# DF의 역수인 IDF(Inverse Document Frequency)를 구하고, TF와 IDF의 값을 곱한 TF-IDF
  를 산출함
```

```
from konlpy.tag import Hannanum
import pandas as pd
han=Hannanum()
df=pd.read_csv("d:/data/text/군집분석데이터.csv",engine="python")
df
```

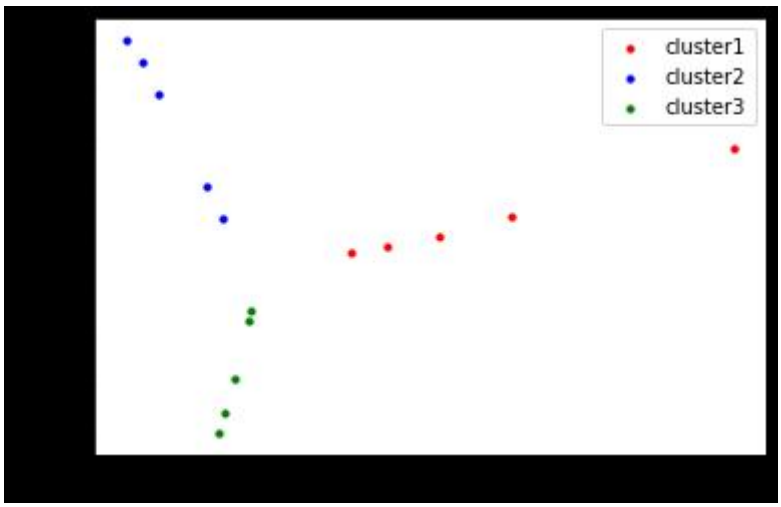
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import KMeans

docs=[]
for i in df["기사내용"]: # 기사내용 필드에서
    docs.append(han.nouns(i)) # 명사만 추출
#print(docs)
for i in range(len(docs)):
    # 명사들 사이에 공백을 붙여서 열거
    docs[i]=" ".join(docs[i]) # 공백을 추가하여 한 문장으로 연결

print(docs[:1])
# 문서-단어 행렬 생성
# 문서 집합에서 단어 토큰을 생성하고 각 단어의 수를 세어서 인코딩한 벡터를 생성
vec=CountVectorizer() # 문장별로 단어의 빈도를 벡터화하여 저장
X=vec.fit_transform(docs)
df2=pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
print(df2.head())
# print(df2.columns) - 변수를 확인하기 위해 출력
# 군집 개수를 3으로 설정
kmeans=KMeans(n_clusters=3,random_state=10).fit(df2)
print(kmeans.labels_)
```

```
%matplotlib inline
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
# 주성분분석(차원을 줄이는 방법)
pca=PCA(n_components=2,random_state=10) # 차원을 2차원으로 축소
components=pca.fit_transform(df2)
df3=pd.DataFrame(data=components,
                 columns=["component1","component2"])
print(df2.head())
print(df3.head())
df3.index=df["검색어"]
plt.scatter(df3.iloc[kmeans.labels_==0,0],
   df3.iloc[kmeans.labels_==0,1],s=10,c="red",label="cluster1")
plt.scatter(df3.iloc[kmeans.labels_==1,0],
   df3.iloc[kmeans.labels_==1,1],s=10,c="blue",label="cluster2")
plt.scatter(df3.iloc[kmeans.labels_==2,0],
   df3.iloc[kmeans.labels_==2,1],s=10,c="green",label="cluster3")
plt.legend()
```

```
[5 rows x 581 columns]
component1    component2
```

0	-2.245408	-4.167447
1	-3.613940	-9.898629
2	-2.932638	-7.159346
3	-2.142776	-3.757097
4	-3.397887	-8.904249

<matplotlib.legend.Legend at 0x25ce880e390>




```
import pandas as pd
df=pd.read_csv("c:/gurujjang/200_Program/data/news.csv",engine="python",
encoding="utf-8", error_bad_lines=False)
df.head()
```

```
import pandas as pd
df=pd.read_csv("c:/gurujjang/200_Program/data/news.csv",engine="python",
encoding="utf-8", error_bad_lines=False)
df.head()
df.tail()
```

```
df["category"].value_counts()
```

```
2    3707
1    2344
3    1835
Name: category, dtype: int64
```

```
import re
# 전처리 함수, 한글, 영문자 외에는 필터링
def preprocessing(sentence):
    sentence=re.sub("[^가-힣ㄱ-ㅎㅏ-ㅣa-zA-Z]"," ",sentence)
    return sentence

df["content_cleaned"]=df["content"].apply(preprocessing)
content=df["content_cleaned"].tolist()
```

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer=CountVectorizer() # 단어 갯수를 세어서 벡터로 저장
X=vectorizer.fit_transform(content)
df2=pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
df2.head()
```

```
from sklearn.preprocessing import normalize
X=normalize(X) # 0에 가까운 값으로 수렴하도록 처리(l2 기법)
```

L1 Regularization 과 L2 Regularization 모두 Overfitting(과적합) 을 막기 위해 사용

```
# 군집의 갯수를 3으로 설정한 모형
from sklearn.cluster import KMeans
kmeans=KMeans(n_clusters=3,random_state=10).fit(X)
```

```
# 실루엣 포인트 확인(원본과 비교하여 군집화가 얼마나 잘 되었는지 파악)
from sklearn.metrics import silhouette_score
silhouette_score(X,kmeans.labels_)
```

2) 구조적 군집분석

```
# 구조적 군집 분석 : 트리 형태의 군집으로 나누는 방법
# 개별 대상간의 거리에 의하여 가장 가까이에 있는 대상들로부터 시작하여 결합한 후
# 트리 모양의 계층구조를 형성하는 방법
```


```
# 구조적 군집 분석 : 트리 형태의 군집으로 나누는 방법
# 개별 대상간의 거리에 의하여 가장 가까이에 있는 대상들로부터 시작하여 결합한 후
# 트리 모양의 계층구조를 형성하는 방법
```

```
from konlpy.tag import Hannanum
import pandas as pd

han=Hannanum()
df=pd.read_csv("C:/gurujjang/200_Program/data/군집분석데이터.csv", engine="python")
df
```

```
from sklearn.feature_extraction.text import CountVectorizer

docs=[]
for i in df["기사내용"]: # 기사내용 필드에서
    docs.append(han.nouns(i)) # 명사만 추출

for i in range(len(docs)):
    docs[i]=" ".join(docs[i])

vec=CountVectorizer() # 문장별로 단어의 빈도를 벡터화하여 저장
X=vec.fit_transform(docs)

df2=pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
df2.head()
```

```
%matplotlib inline
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as shc
import matplotlib.pyplot as plt

cluster=AgglomerativeClustering(n_clusters=3)
cluster.fit_predict(df2)

plt.figure(figsize=(10,7))
plt.title("Customer Dendograms")
result=shc.linkage(df2)
print(result)
dend=shc.dendrogram(result)
```