데이터 수집 / 전처리

Data Preprocessing

데이터 수집

- Naver Sentimental Movie Corpus 란?
  - 한국어로 된 네이버 영화 리뷰를 웹 스크래핑(Web Scraping)한 데이터
  - GitHub 주소 - https://github.com/e9t/nsmc
  - 20만 개의 리뷰를 15만 개의 트레이닝 데이터와 5만 개의 테스트 데이터로 구성
  - 데이터 특성
    - 모든 리뷰는 140자 미만으로 구성
    - 긍정/부정은 동일한 비율로 샘플링
    - 긍정 리뷰는 평점이 9점 이상으로 구성
    - 부정 리뷰는 평점이 4점 이하로 구성
  - 데이터 샘플
    - id : 네이버에서 제공하는 리뷰 ID값
    - document : 실제 리뷰 내용
    - label : 리뷰의 감정 분류(0 : 부정, 1 : 긍정)
    - 각 컬럼 탭으로 구분

```
$ cat ratings_train.txt | head -n 6
id document label
9976970 아 더빙.. 진짜 짜증나네요 목소리 0
3819312 흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나 1
```

데이터 수집

- Naver Sentimental Movie Corpus 데이터 수집 구현
  - GitHub에 존재하는 ratings_train.txt와 ratings_test.txt 다운로드
  - 데이터 다운로드 주소
    - GitHub에서 파일의 이름을 클릭하여 이동한 페이지의 “View Raw”링크를 클릭하여 확인
    - HTTP Method 중 GET Method를 사용
    - 응답의 상태 코드 200으로 정상 호출 확인
    - 응답 온 바이너리 데이터를 파일로 저장

```
8   def nsmc_data_download(file_list) :
9       # 데이터 다운로드 주소
10      nsmc_url = 'https://raw.githubusercontent.com/e9t/nsmc/master/'
11      source_dir ='./data/'
12
13      # 데이터 다운로드 폴더 생성
14      if not(os.path.isdir(source_dir)) :
15          os.makedirs(os.path.join(source_dir))
16
17      # 바이너리 데이터를 파일로 쓰기
18      for file in file_list :
19          response = requests.get(nsmc_url + file)
20          print('file name : ' + file)
21          print('status code : ' + str(response.status_code))
22          with open(source_dir + file,'wb') as f:
23              # 바이너리 형태로 데이터 추출
24              f.write(response.content)
25          f.close()
```

데이터 수집

- Naver Sentimental Movie Corpus 데이터 수집 구현
  - requests 패키지의 간단 사용법
    - Get 요청

      req_params = {'Param1': 'Value1', 'Param2': 'Value2' }

      response = requests.get('http://examples.com', params=req_params)
    - POST 요청

      data = {'Param1': 'Value1', 'Param2': 'Value2' }

      response = requests.post('http://examples.com', data=data)
    - DELETE, HEAD, OPTIONS 요청

      response = requests.delete('http://examples.com/delete')

      response = requests.head('http://examples.com/get')

      response = requests.options('http://examples.com/get')

데이터 전처리

- 네이버 맞춤법 검사기 란?
  - 비격식(Informal) 문장인 영화 리뷰의 맞춤법과 띄어쓰기를 교정함으로써 가독성을 높임
  - 네이버 포털(https://www.naver.com) 화면에서 “네이버 맞춤법 검사기”로 검색
    - 500자 이내의 문장을 “맞춤법, 표준어 의심, 띄어쓰기, 통계적 교정”에 대한 교정 가능

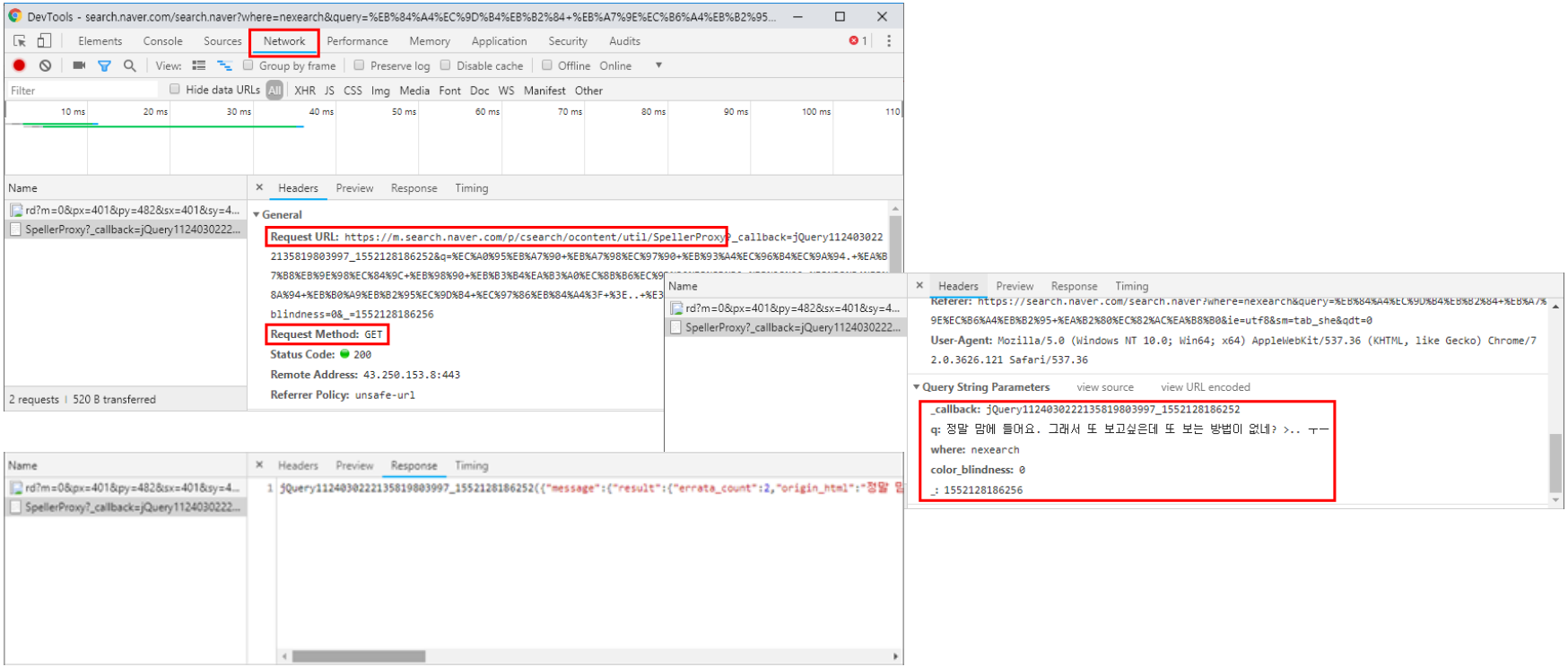
데이터 전처리

- 네이버 맞춤법 검사기 코드 구현
  - 네이버 맞춤법 검사기의 주소와 요청 파라미터, 응답 결과를 가져오는 방법(Chrome)
    - 네이버 맞춤법 화면 이동
    - F12 개발자 도구 생성
    - 텍스트 입력 후 검사하기 버튼 클릭
    - 개발자도구의 Network 탭에서의 왼쪽 Name에서의 마지막 요청 클릭
    - Request URL(? 앞까지), Request Method 확인
    - Query String Parameters 확인
    - Response 탭에서 응답값 확인

```python
def naver_spell_cheker(input) :
    source_dir ='./data/'
    # 네이버 맞춤법 검사기 주소
    spell_checker_url = 'https://m.search.naver.com/p/csearch/ocontent/util/SpellerProxy'

    def spell_cheker(object) :
        # request parameter 셋팅
        req_params = {'_callback': 'SpellChecker', 'q': object, 'color_blindness': 0}
        while True :
            response = requests.get(spell_checker_url, params=req_params)
            status  = response.status_code

            # 응답코드가 200일 때까지 반복
            if status == 200 :
                # 텍스트 형태로 데이터 추출
                response = response.text
                break

        # json 포멧으로 변경하기 위한 불필요 문자 제거
        response = response.replace(req_params.get('_callback')+'(', '')
        response = response.replace(');', '')

        data = json.loads(response)
        # json 포멧에서 필요 결과 값만 가져오기
        object = data['message']['result']['notag_html']
        object = html.unescape(object)

        return object
```

데이터 전처리

- 네이버 맞춤법 검사기의 주소와 요청 파라미터, 응답 결과를 가져오는 방법(Chrome)



데이터 전처리

- 네이버 맞춤법 검사기 코드 구현
  - 응답 값에 대한 샘플
    - message > result > notag_html 사용
    - HTML로 escape되어 있는 문자들에 대해 unescape 진행
  - 많은 문장 수행 시 시간이 오래 소요

```
SpellChecker({
  "message":{
    "result":{
      "errata_count":2,
      "origin_html":"정말 맘에 들어요. 그래서 또 <span class='result_underline'>보고싶은데</span>
       또 보는 방법이 없네? &gt;.. <span class='result_underline'>ㅜㅡ</span>",
      "html":"정말 맘에 들어요. 그래서 또 <em class='green_text'>보고 싶은데</em> 또 보는 방법이 없네?
      &gt;.. <em class='violet_text'>ㅜㅡ</em>",
      "notag_html":"정말 맘에 들어요. 그래서 또 보고 싶은데 또 보는 방법이 없네? &gt;.. ㅜㅡ"
    }
  }
});
```

데이터 전처리

- 자연어처리 란?
  - 사람들이 사용하는 언어에서 의미 있는 정보를 분석하고 추출하여 컴퓨터가 처리
  - 음성 인식, 정보 검색, Q&A 시스템, 문서 분류, ChatBot 등
  - python의 오픈 소스 라이브러리인 NLTK(Neural Language Toolkit)를 사용
    - OS에 관련 없이 설치가 가능하지만 한국어에 대한 지원이 지원되지 않음
- KoNLPy(코엔엘파이) 란?
  - 한국어 자연어 처리를 위해 만들어진 파이썬 오픈 소스 패키지
    - 어떤 대상 어절을 최소 단위인 “형태소”(단어 자체 또는 단어보다 작은 단위)로 분석
    - 분석된 결과에 대해 품사를 부착하는 기능
  - 형태소 분석기로는 Hannanum, Kkma, Komoran, Mecab, Okt가 제공
    (Okt는 v0.5.0 이전에는 Twitter 클래스로 제공)
    - 윈도우 환경에서는 Mecab을 지원하지 않음
  - 각 형태소 분석기별 비교 - http://konlpy.org/ko/latest/morph/

데이터 전처리

- 자연어처리 란?
  - 사람들이 사용하는 언어에서 의미 있는 정보를 분석하고 추출하여 컴퓨터가 처리
  - 음성 인식, 정보 검색, Q&A 시스템, 문서 분류, ChatBot 등
  - python의 오픈 소스 라이브러리인 NLTK(Neural Language Toolkit)를 사용
    - OS에 관련 없이 설치가 가능하지만 한국어에 대한 지원이 지원되지 않음
- KoNLPy(코엔엘파이) 란?
  - 한국어 자연어 처리를 위해 만들어진 파이썬 오픈 소스 패키지
    - 어떤 대상 어절을 최소 단위인 “형태소”(단어 자체 또는 단어보다 작은 단위)로 분석
    - 분석된 결과에 대해 품사를 부착하는 기능
  - 형태소 분석기로는 Hannanum, Kkma, Komoran, Mecab, Okt가 제공
    (Okt는 v0.5.0 이전에는 Twitter 클래스로 제공)
    - 윈도우 환경에서는 Mecab을 지원하지 않음
  - 각 형태소 분석기별 비교 - http://konlpy.org/ko/latest/morph/

데이터 전처리

- KoNLPy 중 Okt(Open Korean Text) 패키지 구현
  - 영화 리뷰에 대해 맞춤법 검사기를 통해 교정된 문장을 형태소 분석 및 품사를 부착
  - Okt 제공 함수
    - Okt().morphs(phrase, norm=False, stem=False)
      - 텍스트를 형태소 단위로 분리. 정규화, 어간 추출
    - Okt().phrase(phrase)
      - 텍스트에서 어절을 추출
    - Okt().nouns(phrase)
      - 텍스트에서 명사를 추출
    - Okt().pos(phrase, norm=False, stem=False, join=False)
      - morphs + 품사
      - join 사용 시 '형태소/품사' 표시

```
83      def pos_tagging(object) :
84          # 형태소 분석 및 품사 태깅(정규화, 어간추출, 품사합치기)
85          pos = Okt().pos(object, norm=True, stem=True, join=True)
86          # 명사 추출
87          noun = Okt().nouns(object)
88
89          return pos, noun
```

데이터 전처리

- KoNLPy 중 Okt(Open Korean Text) 패키지 구현
  - 자주 사용하는 단어에 대한 시각화를 위해 WordCloud 사용
    - generate_from_text 함수를 사용
      - 텍스트로 부터 WordCloud 생성
      - 단어와 빈도, 불용어 제거
    - 빈도가 많은 단어는 크게 표시
  - Counter 패키지를 통한 단어의 빈도 계산(상위 20개)
    - most_common 함수 사용

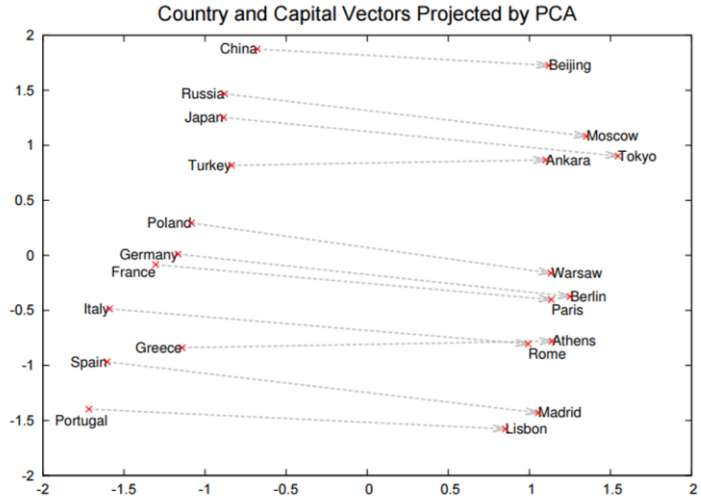
[('영화', 54418), ('이', 11734), ('정말', 10927), ('것', 10211), ('거', 9284), ('안', 8966), ('진짜', 8483), ('점', 8343), ('보고', 6865), ('연기', 6823), ('최고', 6341), ('평점', 6332), ('수', 6190), ('내', 5644), ('왜', 5602), ('말', 5447), ('스토리', 5377), ('생각', 5304), ('드라마', 5112), ('사람', 4969)]

데이터 전처리

- WordCloud 제공 함수
  - fit_words(frequencies)
    - generate_from_frequencies 대한 alias로 단어와 빈도를 통해 wordcloud를 생성
  - generate(text)
    - generate_from_text의 alias로 텍스트를 통해 wordcloud를 생성
    - process_text 함수와 generate_from_frequencies 함수를 호출
  - generate_from_frequencies(frequencies[, …])
    - 단어와 빈도를 통해 wordcloud를 생성
  - generate_from_text(text)
    - 텍스트를 통해 wordcloud를 생성
    - process_text 함수와 generate_from_frequencies 함수를 호출
  - process_text(text)
    - 텍스트를 단어로 분리하고 불용어를 제거합니다.
  - 파라미터 사용 예시
    - text = '텍스트 예제입니다 해당 형태로 작성이 필요합니다'
    - frequencies = {'단어':5, '빈도수':3, '표시':1}

데이터 전처리

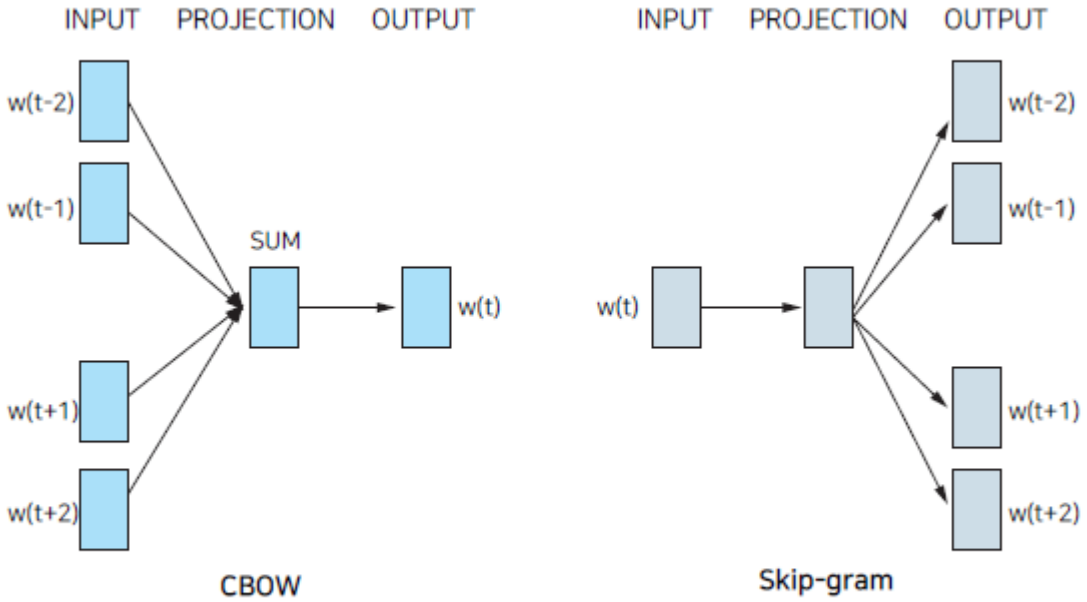
- 단어 임베딩(Word Embedding)
  - 자연어 처리에서 어휘의 단어를 컴퓨터가 처리할 수 있는 실수의 벡터로 변경
  - 구문 분석이나 감정 분석에 성능 향상
- Word2Vec
  - 기존의 one-hot vector 방식의 단어 표현은 단어 간 유사도를 표현할 수 없다는 단점
  - 여러 구글 엔지니어(2013년 Tomas Mikolov 외)에 의해 개발한 Neural Network 기반 알고리즘
    - “비슷한 위치에 등장하는 단어들은 비슷한 의미를 가진다”는 분포 가설(Distributional Hypothesis)
    - 독일과 베를린이 프랑스와 파리의 같은 방식으로 관련
    - ‘파리’-‘프랑스’+‘이탈리아’는 ‘로마’
    - ‘왕’-‘남자’+‘여자’는 ‘여왕’에 가까운 결과



"Distributed Representations of Words and Phrases and their Compositionality", Mikolov, et al. 2013

데이터 전처리

- Word2Vec 모델



"Efficient Estimation of Word Representation in Vector Space", Mikolov, et al. 2013

데이터 전처리

- Word2Vec 모델
  - CBOW(continuous bag-of-words)
    - 주변에 있는 단어들을 가지고, 중간에 있는 단어들을 예측
      - ex) “a barking dog never bites” - {“a”, “barking”, “never”, “bites”}를 통해 “dog”를 예측
      - 타겟 단어(Target Word) - 예측되는 단어
      - 주변 단어(Context Word) - 예측에 사용되는 단어
      - 윈도우(Window) - 주변 단어를 앞뒤로 몇 개까지 볼 수 있는지 지정
      - 슬라이딩 윈도우(Sliding window) - 윈도우를 옆으로 이동하면서 타겟 단어를 바꾸는 것
      - 크기가 작은 데이터셋에 적합, 속도가 빠른 장점
    - 윈도우가 2인 경우

데이터 전처리

- Word2Vec 모델
  - Skip-gram
    - 중간에 있는 단어로 주변 단어들을 예측
      - ex) “a barking dog never bites”
        윈도우 크기가 2일 때 타겟 단어가 “dog”라면 그 주변 {“a”, “barking”, “never”, “bites”} 단어를 예측
    - 타겟 단어의 주변 단어의 배수만큼 학습이 진행, 속도는 느리지만 성능이 더 좋은 결과

- Word2Vec 이외의 단어 임베딩 모델
  - GloVe - 2014년 미국 스텐포드대학 연구팀에서 개발
  - Fasttext - 2016년 페이스북에서 개발

데이터 전처리

- Word2Vec 구현
  - Gensim 패키지 내 model 클래스 사용
  - Word2Vec 파라미터
    - size :단어를 벡터값으로 변환하기 위한 차원 수, 단어의 전체 개수에 따라 유동적으로 변경 필요
    - window : 문장 내 현재 단어와 예측 단어와의 최대 거리
    - min_count : 해당 값보다 낮은 빈도수 단어는 무시
    - workers : 모델을 학습하기 위한 병렬 처리 쓰레드 개수
    - sg : CBOW=0, Skip-gram=1
    - iter : 반복 학습 횟수

```
10  def apply_word2vec(file_list) :
11      source_dir ='./data/'
12      # 전체 문장을 담는 리스트 선언
13      total_sentences = list()
14
15      for file in file_list:
16          with open(source_dir + file, 'r', encoding='UTF-8') as f:
17              load_data = [line.split('\t') for line in f.read().splitlines()]
18              for data in load_data :
19                  total_sentences.append(data[0].split())
20
21      # word2vec로 단어 벡터로 변경 및 모델 저장
22      model = models.Word2Vec(total_sentences, min_count=3, window=5, sg=1, size=100, workers=4, iter=50)
23      model.save(source_dir + '3_word2vec_nsmc.w2v')
24      model.wv.save_word2vec_format(source_dir + '3_word2vec_nsmc_format.w2v', binary=False)
```

데이터 전처리

- Word2Vec 구현
  - save_word2vec_format 함수를 통해 단어가 벡터로 변경된 내용에 대해 확인 가능
    - 첫 번째 라인의 두 숫자 : 단어의 전체 개수와 벡터 차원 수
    - 두 번째 라인부터 단어에 대한 벡터 표현

```
22615 100
영화/Noun -0.32818502 0.27860388 0.1523643 0.06831967 0.19623944 -0.07209147
-0.18959798 -0.14979233 0.4294706 -0.38620764 -0.36520967 -0.005111015
-0.16903515 -0.39880487 0.19440751 -0.21463037 -0.0055839033 0.0191054
-0.13867551 -0.36297414 0.21187727 -0.12709628 -0.03174775 -0.041058134 0.14193992
-0.4235093 0.20149146 0.031723544 -0.09822699 0.19908044 -0.08718104 -0.046228327
0.09436537 0.13143788 -0.033787344 -0.21516575 0.11618206 0.27091342 0.05722208
-0.46929833 -0.074162 -0.1251334 0.15102917 -0.3119958 -0.02536161 -0.31003794
0.101542465 0.29173 -0.15800369 0.0266653 -0.08451466 0.07947255 -0.12946346
0.077833004 -0.18809474 0.022096505 0.23722965 0.019673629 -0.053121354 0.06628938
… 중략
```

데이터 전처리

- Word2Vec 모델 테스트
  - “배우, 엄마, 여자, 남자”의 단어를 사용 - “배우/Noun, 엄마/Noun, 여자/Noun, 남자/Noun”

```
34      # word2vec 모델 로드
35      model = models.Word2Vec.load(source_dir + w2v_name)
36
37      # 품사 태깅 된 데이터 추출 및 리스트 저장
38      data_list = list()
39      data1 = pre.konlpy_pos_tag('배우')
40      data_list.append(data1)
41      data2 = pre.konlpy_pos_tag('엄마')
42      data_list.append(data2)
43      data3 = pre.konlpy_pos_tag('여자')
44      data_list.append(data3)
45      data4 = pre.konlpy_pos_tag('남자')
46      data_list.append(data4)
47
48      # 모델에 적용하여 결과 출력
49      # model.doesnt_match, model.most_similar의 method는 4.0.0 버전에서 deprecated
50      print(model[data1])
51      print(model.wv.doesnt_match(data_list))
52      print(model.wv.most_similar(positive=[data1], topn=10))
53      print(model.wv.most_similar(positive=[data2, data4], negative=[data3], topn=1))
54      print(model.wv.similarity(data1, data2))
55      print(model.wv.similarity(data1, data3))
```

데이터 전처리

- Word2Vec 모델 테스트
  - “배우, 엄마, 여자, 남자”의 단어를 사용 - “배우/Noun, 엄마/Noun, 여자/Noun, 남자/Noun”
    - model['배우/Noun'] : 100차원 벡터 표현
    - doesnt_match 함수 : 유사도가 없는 결과 표현
      - 결과 : “배우/Noun”
    - most_similar 함수 : 유사도가 높은 상위 N개의 단어 추출
      - positive :단어와 긍정적인 단어의 리스트
      - negative : 단어와 부정적인 단어 리스트
      - topn : 유사도가 높은 상위 n개의 단어를 반환
      - similarity : 두 단어 사이 유사도
        - “배우/Noun”과 유사도가 높은 10개의 단어
          - [('연기자/Noun', 0.7866206169128418), ('여배우/Noun', 0.7009295225143433), ('조연/Noun', 0.6413561701774597), ('영화배우/Noun', 0.6213721632957458), ('열연/Noun', 0.6110374927520752)]
        - “엄마/Noun”와 “남자/Noun”에 긍정적이고, “여자/Noun”에 부정적인 최상위 단어
          - 결과 : [('아빠/Noun', 0.7548638582229614)
        - “배우/Noun”와 “여자/Noun”의 유사도
          - 결과 : 0.39090595

데이터 전처리

- Word2Vec 모델 시각화
  - t-SNE(t-Stochastic Neighbor Embedding) 란?
    - 고차원 벡터 차원을 축소하여 표시
    - 차원 축소 시 축의 위치 변경에 따라 다른 모양으로 변환
    - 군집성은 유지
    - n_component : 축소할 차원 수
  - 한글에 대한 시각화
    - 폰트 검색 - matplotlib 패키지

```
['/usr/share/fonts/truetype/nanum/NanumGothicBold.ttf',
'/usr/share/fonts/truetype/nanum/NanumMyeongjo.ttf',
'/usr/share/fonts/truetype/nanum/NanumMyeongjoBold.ttf',
'/usr/share/fonts/truetype/nanum/NanumBarunGothic.ttf',
'/usr/share/fonts/truetype/nanum/NanumGothic_Coding.ttf',
… 중략
]
```

    - 폰트 설정 - matplotlib 패키지
      - rc() 함수

```python
# 단어 리스트 중 가장 많이 사용된 100개 단어 추출
counter = Counter(total_word_list).most_common(100)
word_list = [word[0] for word in counter]
print(word_list)

# 설정 가능한 폰트 리스트 출력
font_list = font_manager.get_fontconfig_fonts()
print([font for font in font_list if 'nanum' in font])

# 폰트 설정
rc('font', family=font_manager.FontProperties(fname=font_name).get_name())

# 단어에 대한 벡터 리스트
vector_list = model[word_list]

# 2차원으로 차원 축소
transformed = TSNE(n_components=2).fit_transform(vector_list)
print(transformed)

# 2차원의 데이터를 x, y 축으로 저장
x_plot = transformed[:, 0]
y_plot = transformed[:, 1]

# 이미지의 사이즈 셋팅
pyplot.figure(figsize=(10, 10))

# x, y 축을 점 및 텍스트 표시
pyplot.scatter(x_plot, y_plot)
for i in range(len(x_plot)):
    pyplot.annotate(word_list[i], xy=(x_plot[i], y_plot[i]))

# 이미지로 저장
pyplot.savefig(source_dir + fig_file)
```

데이터 전처리

- Word2Vec 모델 시각화
  - t-SNE 결과
    - 의미가 유사한 단어들이 거리가 가깝게 표시
    - “재미있다, 재밌다”, “감동, 재미”의 단어가 거리가 가깝게 표현
    - “재미없다, 아깝다, 지루하다”의 단어들도 가깝게 표시