MUSINSA

상품 리뷰 감성분석

및 사이즈 예측

한국품질재단 8조

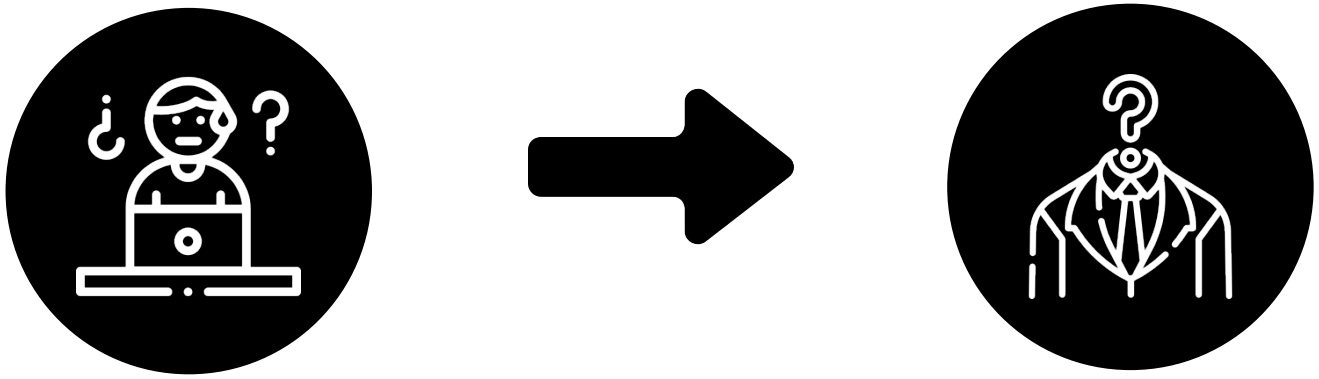
김정기, 제소현

목차

MUSINSA ?

- 대한민국의 온라인 패션커머스 기업
- 국내 최대 규모의 온라인 편집샵
- 패션 플랫폼 월간 활성 사용자 1위에 선정

리뷰 파악의 한계점



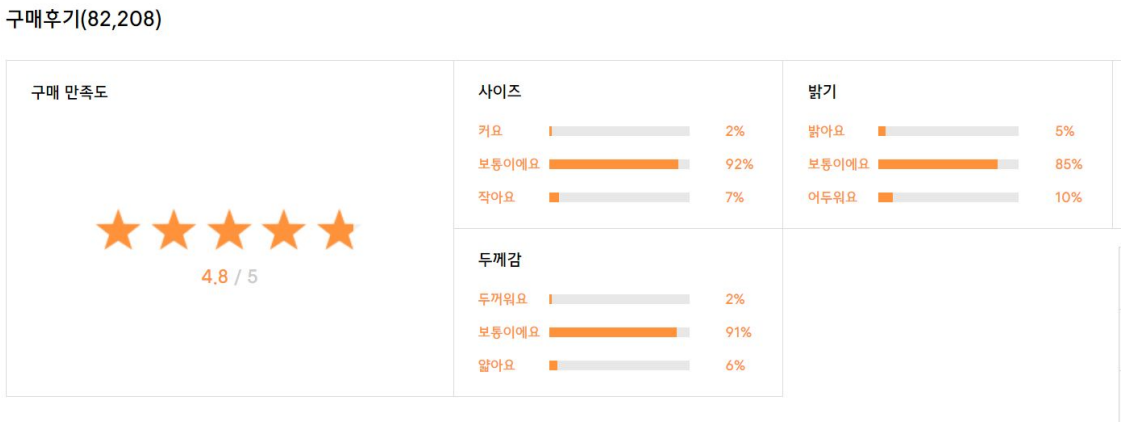
리뷰를 보고 상품에 대한 장단점 파악이 어려움

별점으로만 파악하는 것에 대한 한계점

- 상품에 대한 별점을 높게 매겨도 텍스트 리뷰에 단점이 수록된 경우 다수.
- 광고성 리뷰와 실구매자들의 리뷰가 섞여 많은 양의 리뷰 속 파묻혀 단점 리뷰만 찾기 어려움.
- 많은 리뷰를 모두 파악하기 어려움.

리뷰 파악의 한계점 예시

① 별점으로만 상품을 판별하는 것이 어려움



② 카테고리별 후기가 너무 많아 오히려 보기 어려움

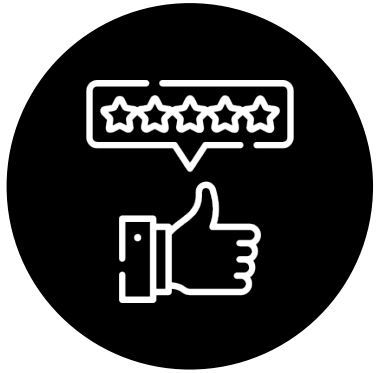
③ 별점이 상당히 높은 경우에도 텍스트리뷰에 단점이 들어감

④ 광고 별점이 포함되어 있어 별점이 낮은 경우(단점이 포함되어 있는 경우)만을 보기 어려움

⑤ 상품의 단점만을 보기 어려움

[Musinsa Standard] 테이퍼드 히든 밴딩 크롭 슬랙스 후기 中

진행 방법



상품 리뷰 크롤링

해당 카테고리에서
리뷰수가 많은 상품
리뷰 크롤링

리뷰 데이터 감성 분석

리뷰 데이터를 형태소로
잘라 필터링을 한 뒤,
각 리뷰에 대한 긍정-부정
퍼센트를 분석

분석 결과 도식화

분석 결과를 바탕으로
상품의 최종적인 감정
퍼센트 비율과 감정별
대표 키워드 추출 도식화

진행 과정

김정기: 상품 리뷰 크롤링

① 여러 카테고리 중 한 품목을 선정하여 리뷰수가 많은 임의의 인기 상품 url을 추출

```
wd.find_element(By.XPATH, '//*[@id="goods_list"]/div[2]/div[1]/div/div/a[3]').click()
a = wd.find_elements(By.CLASS_NAME,'img-block')
for j in a:
    urlList.append(j.get_attribute('href'))
```

```
url = []
url.append(urlList[0])
url.append(urlList[1])
url.append(urlList[6])
url
```

```
['https://www.musinsa.com/app/goods/1778404',
 'https://www.musinsa.com/app/goods/640839',
 'https://www.musinsa.com/app/goods/1778408']
```

```
wd.get(url[0])
```

```
url
```

```
['https://www.musinsa.com/app/goods/1778404',
 'https://www.musinsa.com/app/goods/640839',
 'https://www.musinsa.com/app/goods/1778408']
```

진행 과정

② 상품의 카테고리 별 리뷰 크롤링
(Style, Photo, Goods)

```
from types import NoneType

# 시작 url loop
for j in range(3):
    wd.get(urlList[j])
    print('상품변경')
    result=[]
    time.sleep(1)
    for k in range(1,4):
        reviews = ['스타일 후기', '상품 후기','일반 후기']
        # 스타일 후기 , 상품 후기 , 일반 후기
        wd.find_element(By.XPATH,f'//*[@id="estimateBox"]/div[2]/ul/li[{k}]').click()
        reviews = ['스타일 후기', '상품 후기','일반 후기']
        print(reviews[k-1])
        time.sleep(0.5)
        req = wd.page_source
        soup = BeautifulSoup(req,'html.parser')
        page = soup.select_one('div.box_page_msg')
        # 리뷰가 하나도 없을 때 처리
        if type(page) == NoneType:
            print('No data')
            continue
        # 페이지 숫자 뽑아내기
        page = page.text.replace(' ','').replace('\n','').replace('페이지','')
        a = page.split('중')
        num = int(a[0])
```

진행 과정

③ 크롤링 데이터를 CSV 파일로 변환하여 저장

```
# 총 한 블록당 페이지수가 5이므로 5로 나누어준다.
num = int(num/5)
for _ in range(num + 1):
    try:
        # 페이지별로 돌기
        for i in range(3,8):
            try:
                wd.find_element(By.XPATH,f'//*[@id="reviewListFragment"]/div[11]/div[2]/div/a[{i}]').click()
                print(i-2,'페이지',end=" ")
                req = wd.page_source
                soup = BeautifulSoup(req,'html.parser')
                bodysizeList = soup.select('div.review-profile__information')
                print("리뷰수: ", len(bodysizeList))

                for i in range(len(bodysizeList)):
                    # 바디사이즈
                    bodysizeList = soup.select('div.review-profile__information')
                    bodysize =  bodysizeList[i].text.replace('\n','')
                    bodysize =  bodysize.replace('신고','')
                    if bodysize == '':
                        bodysize = '남성,175cm,75kg'
                    sex,height,weight = bodysize.split(',')
                    # 옷이름
                    clothnameList = soup.select('div.review-goods-information__item > a')
                    clothname = clothnameList[i].text
                    # 옷사이즈
                    sizeList = soup.select('div.review-goods-information__item > p > span')
                    size = sizeList[i].text.replace('\n','').replace(' ','')
                    # 리뷰
                    reviewList = soup.select('div.review-contents > div.review-contents__text')
                    review = reviewList[i].text
                    result.append([sex]+[height]+[weight]+[clothname]+[size]+[review])
                # 페이지 내에서 에러가 났을때
                except Exception as e:
                    print(e)
                    continue
            # 페이지 블록넘기기
            wd.find_element(By.XPATH,'//*[@id="reviewListFragment"]/div[11]/div[2]/div/a[8]').click()
            time.sleep(1)
            print('pageblock change')

        # 페이지블록을 넘기면서 에러가 났을때
        except Exception as e:
            print(e)
            continue
    columns = ['sex','height','weight','clothname','size','review']
    musinsa_df = pd.DataFrame(result,columns=columns)
    musinsa_df.to_csv(f'./musinsa{j}_csv',index=True, encoding='utf-8')

print('complete')
```

	Unnamed: 0	sex	height	weight	clothname	size	review
0	0	남성	177cm	88kg	엔젤 와펜 집업 후드 그레이	XL	두께감도있고 꽤 오버핏에 편하게 입기 좋아요~무난하게 입을듯
1	1	남성	185cm	74kg	엔젤 와펜 집업 후드 그레이	XL	생각했던것보다 핏이 더 좋게 나옵니다. 로고도 귀여워요 :)
2	2	남성	168cm	65kg	엔젤 와펜 집업 후드 그레이	M	간절기 말고 초겨울에도 따뜻하게 입을 수 있을거 같습니다.생각보다 두꺼워요
3	3	여성	168cm	70kg	엔젤 와펜 집업 후드 블랙	L	사이즈도 좋고 여러옷에 다 잘맞아요.와핏이즌 옷은 실패한적이 없어요.딱 좋아요.이쁜...
4	4	남성	168cm	52kg	엔젤 와펜 집업 후드 그레이	M	진짜이뻐요 로고랑 뒤에 프린팅도 이쁘고 자주자주입을거같아요
...	...	...	...	...	...	...	...
13236	13236	남성	175cm	75kg	엔젤 와펜 집업 후드 그레이	L	옷도 이쁘고 질도 괜찮은데 후드끈이 없이 배송이 됐네요 비싼돈주고 샀는데 후드끈이 ...
13237	13237	남성	175cm	75kg	엔젤 와펜 집업 후드 그레이	M	이쁘고 기모 있어요 오버하게 입기 좋은것 같아요
13238	13238	남성	175cm	75kg	엔젤 와펜 집업 후드 그레이	S	배송이 정말 느리네요... 9월2일에 주문했는데 9월 7일에 왔어요ㅠ
13239	13239	남성	175cm	75kg	엔젤 와펜 집업 후드 블랙	S	키가 큰 편이라 사이즈 s 사면서 혹시 작을까 걱정했는데 하나도 안 작고 넉넉해요 ...
13240	13240	남성	175cm	75kg	엔젤 와펜 집업 후드 그레이	L	좀 클 줄 알았는데 널널하게 오버핏 나오고 많이 안길어요! 기모도 잘 들어가있고 지...

13241 rows × 7 columns

진행 과정

제소현: 리뷰 데이터 감성 분석 및 결과 도식화

① 학습할 리뷰 데이터를 정제 (200000개 중 중복을 제외한 총 샘플의 수: 199908)

※ 무신사에는 부정 리뷰가 거의 존재하지 않아 학습하기에 한계점 발생. 따라서 라벨링 작업이 되어있는 네이버 쇼핑 리뷰를 바탕으로 학습함.

```
train_data, test_data = train_test_split(total_data, test_size = 0.25, random_state = 42
print('훈련용 리뷰의 개수 :', len(train_data))
print('테스트용 리뷰의 개수 :', len(test_data))
```

```
훈련용 리뷰의 개수 : 149931
테스트용 리뷰의 개수 : 49977
```

```
test_data.drop_duplicates(subset = ['reviews'], inplace=True) # 중복 제거
test_data['reviews'] = test_data['reviews'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","") # 정규 표현식 수행
test_data['reviews'].replace('', np.nan, inplace=True) # 공백은 Null 값으로 변경
test_data = test_data.dropna(how='any') # Null 값 제거
print('전처리 후 테스트용 샘플의 개수 :',len(test_data))
```

```
전처리 후 테스트용 샘플의 개수 : 49977
```

② 토큰화 작업으로 필요 없는 토큰 삭제

```
#train, test 데이터에 Mecab을 사용하여 토큰화 작업 중 '의', '이' 등 필요없는 토큰 삭제
mecab = Mecab()
stopwords = ['도', '는', '다', '의', '가', '이', '은', '한', '에', '하', '고', '을',
```

```
train_data['tokenized'] = train_data['reviews'].apply(mecab.morphs)
train_data['tokenized'] = train_data['tokenized'].apply(lambda x: [item for item in
test_data['tokenized'] = test_data['reviews'].apply(mecab.morphs)
test_data['tokenized'] = test_data['tokenized'].apply(lambda x: [item for item in x
```

③ 긍-부정 단어 중 빈도수가 높은 상위 단어 출력

```
#부정적인 단어들 중 빈도수가 높은 상위 20개 단어 출력
negative_word_count = Counter(negative_words)
print(negative_word_count.most_common(20))
```

```
[('네요', 31823), ('는데', 20095), ('안', 19748), ('어요', 14869), ('있', 13200),
('너무', 13056), ('했', 11796), ('좋', 9797), ('배송', 9623), ('같', 8995), ('거', 8
904), ('어', 8896), ('구매', 8884), ('없', 8695), ('아요', 8636), ('습니다', 8436),
('그냥', 8353), ('되', 8350), ('잘', 8029), ('않', 7986)]
```

```
#긍정적인 단어들 중 빈도수가 높은 상위 20개 단어 출력
positive_word_count = Counter(positive_words)
print(positive_word_count.most_common(20))
```

```
[('좋', 39365), ('아요', 21151), ('네요', 19906), ('어요', 18672), ('잘', 18612),
('구매', 16175), ('습니다', 13322), ('있', 12380), ('배송', 12263), ('는데', 11579),
('합니다', 9850), ('했', 9806), ('먹', 9453), ('재', 9252), ('너무', 8398), ('같', 7
867), ('만족', 7225), ('거', 6485), ('기', 6341), ('어', 6317)]
```

진행 과정

④ 분석에 영향을 미치지 않는 희귀 단어 비율 확인 후 제거

```
#등장 횟수가 1인 단어들 배제하기 위해 단어들의 비중 검색

threshold = 2
total_cnt = len(tokenizer.word_index) # 단어의 수
rare_cnt = 0 # 등장 빈도수가 threshold보다 작은 단어의 개수를 카운트
total_freq = 0 # 훈련 데이터의 전체 단어 빈도수 총 합
rare_freq = 0 # 등장 빈도수가 threshold보다 작은 단어의 등장 빈도수의 총 합

# 단어와 빈도수의 쌍(pair)을 key와 value로 받는다.
for key, value in tokenizer.word_counts.items():
    total_freq = total_freq + value

    # 단어의 등장 빈도수가 threshold보다 작으면
    if(value < threshold):
        rare_cnt = rare_cnt + 1
        rare_freq = rare_freq + value


print('단어 집합(vocabulary)의 크기 :',total_cnt)
print('등장 빈도가 %s번 이하인 희귀 단어의 수: %s'%(threshold - 1, rare_cnt))
print("단어 집합에서 희귀 단어의 비율:", (rare_cnt / total_cnt)*100)
print("전체 등장 빈도에서 희귀 단어 등장 빈도 비율:", (rare_freq / total_freq)*100)
```

```
단어 집합(vocabulary)의 크기 : 40263
등장 빈도가 1번 이하인 희귀 단어의 수: 18600
단어 집합에서 희귀 단어의 비율: 46.19625959317488
전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 0.9139056642988648
```

```
# 전체 단어 개수 중 빈도수 2이하인 단어 개수는 제거.
# 0번 패딩 토큰과 1번 OOV 토큰을 고려하여 +2
vocab_size = total_cnt - rare_cnt + 2
print('단어 집합의 크기 :',vocab_size)
print(X_train)
```

```
단어 집합의 크기 : 21665
```

⑤ 텍스트를 처리할 수 있도록 정수 인코딩 작업

```
#텍스트를 숫자로 변환
tokenizer = Tokenizer(vocab_size, oov_token = 'OOV')
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
print(X_train[:3])
print(X_test[:3])
```

```
[[67, 2067, 301, 14197, 261, 73, 6, 238, 170, 135, 807, 2939, 624, 2, 76, 62, 206, 4
0, 1356, 155, 3, 6], [479, 405, 53, 8473, 2597, 2373, 338, 2940, 248, 2387, 39, 472,
2], [44, 24, 817, 103, 35, 2374, 160, 7, 10, 8007, 4, 1326, 30, 139, 321, 45, 59, 16
0, 139, 7, 1935, 2, 113, 161, 1388, 302, 120, 134]]
[[14, 711, 772, 118, 186, 249, 12], [338, 3902, 62, 3817, 1607], [11, 70, 2, 49, 16
5, 3, 27, 15, 6, 513, 289, 17, 93, 110, 585, 59, 7, 2]]
```

⑥ 적당한 길이들의 리뷰로 작업할 수 있게 정제

```
#길이가 80이하인 샘플을 제외하고 삭제
def below_threshold_len(max_len, nested_list):
  cnt = 0
  for s in nested_list:
    if(len(s) <= max_len):
        cnt = cnt + 1
  print('전체 샘플 중 길이가 %s 이하인 샘플의 비율: %s'%(max_len, (cnt / len(nested_l

max_len = 80
below_threshold_len(max_len, X_train)
X_train = pad_sequences(X_train, maxlen = max_len)
X_test = pad_sequences(X_test, maxlen = max_len)
```

```
전체 샘플 중 길이가 80 이하인 샘플의 비율: 99.99933302652553
```

진행 과정

⑦ GRU를 이용해서 감성 분석 (모델 학습)

```
#GRU를 이용하여 감성 분석(데이터를 학습시킴)
model = Sequential()
model.add(Embedding(vocab_size, 100))
model.add(GRU(128))
model.add(Dense(1, activation='sigmoid'))
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(X_train, y_train, epochs=15, callbacks=[es, mc], batch_size=60, v
```

```
Epoch 1/15
1999/2000 [============================>.] - ETA: 0s - loss: 0.2731 - acc: 0.8984
Epoch 00001: val_acc improved from -inf to 0.91826, saving model to best_model.h5
2000/2000 [==============================] - 137s 68ms/step - loss: 0.2731 - acc: 0.
8984 - val_loss: 0.2254 - val_acc: 0.9183
Epoch 2/15
1999/2000 [============================>.] - ETA: 0s - loss: 0.2135 - acc: 0.9233
Epoch 00002: val_acc improved from 0.91826 to 0.92323, saving model to best_model.h5
2000/2000 [==============================] - 135s 67ms/step - loss: 0.2135 - acc: 0.
9233 - val_loss: 0.2123 - val_acc: 0.9232
Epoch 3/15
1999/2000 [============================>.] - ETA: 0s - loss: 0.1978 - acc: 0.9296- E
TA: 1s -
Epoch 00003: val_acc improved from 0.92323 to 0.92710, saving model to best_model.h5
2000/2000 [==============================] - 139s 70ms/step - loss: 0.1978 - acc: 0.
9296 - val_loss: 0.2047 - val_acc: 0.9271
```

```
#테스트 정확도 출력
loaded_model = load_model('best_model.h5')
print("₩n 테스트 정확도: %.4f" % (loaded_model.evaluate(X_test, y_test)[1]))
```

```
1562/1562 [==============================] - 18s 12ms/step - loss: 0.2127 - acc: 0.9
251
```

테스트 정확도: 0.9251

⑧ 각 리뷰의 긍정-부정도 분석

```
#긍정, 부정의 비율을 알기 위해 긍정, 부정 데이터의 개수 저장
positive_ave=[]
negative_ave=[]

#각각 긍정, 부정데이터에서의 빈출 단어를 출력하기 위한 list
positive_data=[]
negative_data=[]

#학습된 데이터를 가지고 리뷰가 긍정인지 부정인지 예측
def sentiment_predict(new_sentence):
  print(new_sentence)
  sentence_noun=mecab.nouns(new_sentence)

  new_sentence = mecab.morphs(new_sentence) # 토큰화
  new_sentence = [word for word in new_sentence if not word in stopwords] # 불용어 제거
  encoded = tokenizer.texts_to_sequences([new_sentence]) # 정수 인코딩
  pad_new = pad_sequences(encoded, maxlen = max_len) # 패딩
  score = float(loaded_model.predict(pad_new)) # 예측
  if(score > 0.5):
    print("{:.2f}% 확률로 긍정 리뷰입니다.".format(score * 100))
    positive_ave.append(score * 100)
    positive_data.append(sentence_noun)
  else:
    print("{:.2f}% 확률로 부정 리뷰입니다.".format((1 - score) * 100))
    negative_ave.append((1 - score) * 100)
    negative_data.append(sentence_noun)
```

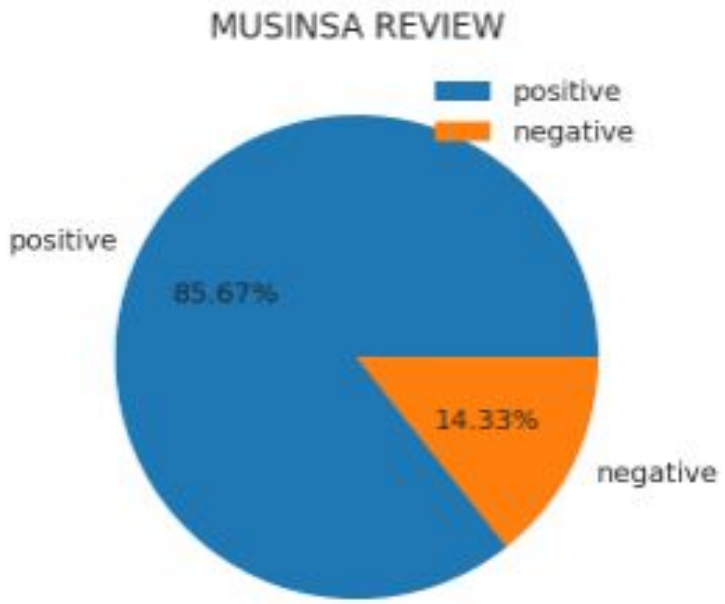
```
진짜 별로에요옷 봉제 문제로 번 교환받았고짜증나서 안입다 오늘 입으려고 다시 봤는
데앞목 부분에 봉제가 안되어 있네요이벳필드 진짜 별로입니다마감 진짜 구려요실밥 다
풀려있고 지금 확인한거는 날짜가 주 이상이 지났기 때문에 교환 환불도 불가능하고 스
트레스 엄청 받네요공장이 문제인건지 뭐가 문제인건지 모르겠지만 사면 절대 안되는
브랜드 같아요참다참다더보기
99.80% 확률로 부정 리뷰입니다.

티셔츠가 너무 귀엽고 이쁘네요 사이즈도 딱 적당한거같이용
99.23% 확률로 긍정 리뷰입니다.

프린팅이 아주 이쁘고 재질도 아주 좋습니다 아주 만족하고 있어요
99.06% 확률로 긍정 리뷰입니다.
```

진행 과정

⑨ 긍정, 부정 퍼센트 계산 후 도식화

```
#긍정, 부정 데이터의 비율 계산
total_len=len(positive_data)+len(negative_data)
ratio_list=[]

p_rate=(len(positive_data)/total_len)*100
#print("긍정 : {:.2f}%".format(p_rate))
ratio_list.append(p_rate)

n_rate=(len(negative_data)/total_len)*100
#print("부정 : {:.2f}%".format(n_rate))
ratio_list.append(n_rate)
```

```
# import matplotlib.pyplot as plt
labels=['positive', 'negative']

title = 'MUSINSA REVIEW'
plt.pie(ratio_list,labels=labels,autopct='%.2f%%')
plt.legend(labels)
plt.title(title)
plt.show()
```



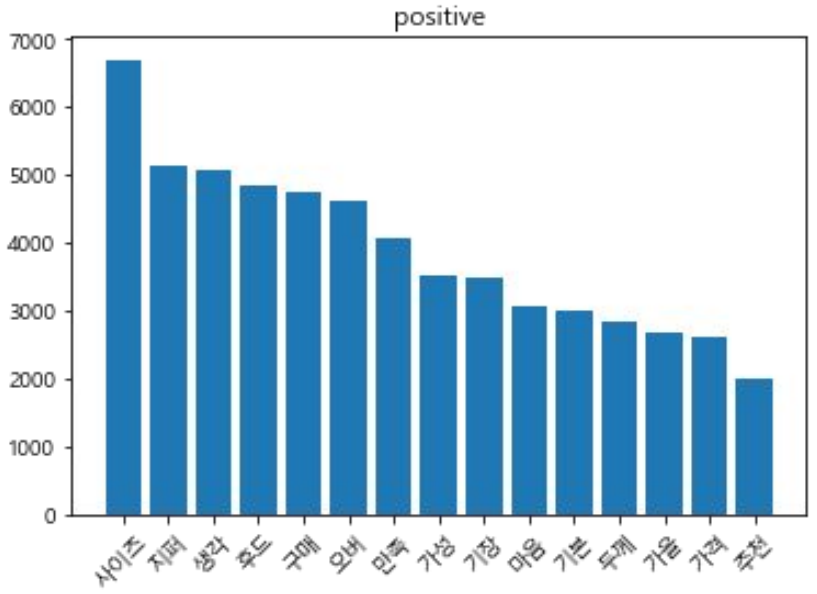
진행 과정

⑩ 긍정 데이터의 빈출 단어 도식화

```
# plt.rc('font', family='NanumBarunGothic')

# positive_data 를 한 배열에 모두 넣고 명사로 분리
text4 = []
text5 = []
nm=[]

# 형태소로 분리되어 리스트로 구분되어 저장되어 있는 것을 하나의 문자열로 저장
for i in range (len(positive_data)):
    text3 = ' '.join(positive_data[i])
    text4.append(text3)
text5 = ' '.join(text4)

# 저장된 문자열을 명사로 분리
nm = mecab.nouns(text5)

# Counter 명령어를 이용하여 문자당 사용된 빈도수를 계산
counter2 = Counter(nm)
available_counter2 = Counter({x: counter2[x] for x in counter2 if len(x) > 1})# 1글자 이하의 문자 제거
available_counter2.most_common(10)# 최다빈도 문자를 10개 출력
pos_real_avail=available_counter2.most_common(15)# 15개의 최다빈도 문자를 저장
#print(pos_real_avail)


positive_most_word=[]
positive_most_value=[]

#map을 이용하여 list에 묶여있는 데이터를 분리
real_avail=list(map(list,pos_real_avail))

#  word과 value를 따로 저장
for i in real_avail:
    positive_most_word.append(i[0])
    positive_most_value.append(i[1])

print(positive_most_word)
print(positive_most_value)
```

```
['사이즈', '지퍼', '생각', '후드', '구매', '오버', '만족', '가성', '기장', '마음', '기본', '두께', '가을', '가격', '추천']
[6692, 5135, 5057, 4829, 4744, 4612, 4085, 3531, 3501, 3054, 2990, 2833, 2693, 2627, 1997]
```

```
x1 = np.arange(len(positive_most_word))
plt.bar(x1, positive_most_value)
plt.title('positive')
plt.xticks(x1, positive_most_word)
plt.xticks(rotation=45)
plt.show()
```



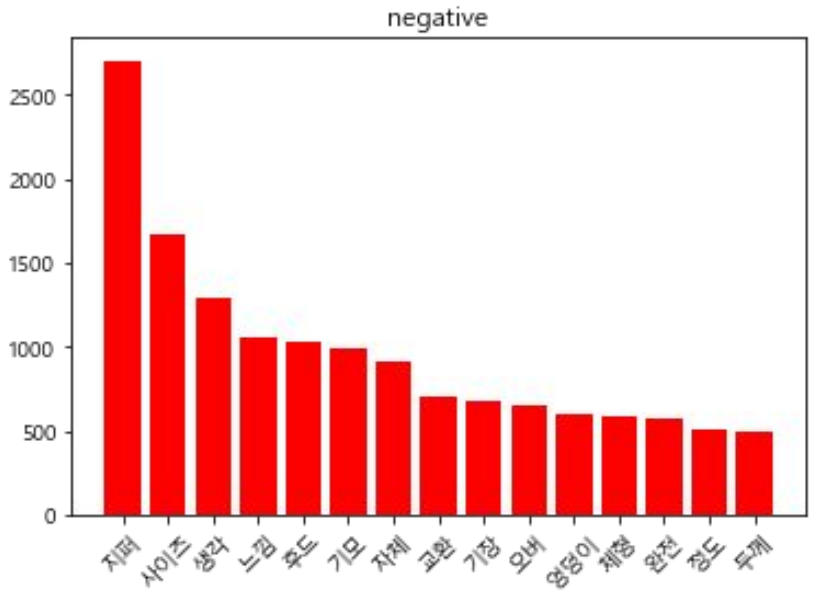
진행 과정

⑪ 부정 데이터의 빈출 단어 도식화

```
# negative_data 를 한 배열에 모두 넣고 명사로 분리
text7 = []
text8 = []
bl=[]
# 형태소로 분리되어 리스트로 구분되어 저장되어 있는 것을 하나의 문자열로 저장
for i in range (len(negative_data)):
    text6 = ' '.join(negative_data[i])
    text7.append(text6)
text8 = ' '.join(text7)
print("")

# 저장된 문자열을 명사로 분리
nl = mecab.nouns(text8)

                # Counter 명령어를 이용하여 문자당 사용된 빈도수를 계산
counter1 = Counter(nl)

# 1글자 제거
available_counter1 = Counter({x: counter1[x] for x in counter1 if len(x) > 1})# 1글자 이하의 문자 제거
available_counter1.most_common(10)# 최다빈도 문자를 10개 출력
neg_real_avail=available_counter1.most_common(15)# 15개의 최다빈도 문자를 저장
#print(neg_real_avail)

negative_most_word=[]
negative_most_value=[]

#map을 이용하여 list에 묶여있는 데이터를 분리
real_avail=list(map(list,neg_real_avail))

#  word과 value를 따로 저장
for i in real_avail:
    negative_most_word.append(i[0])
    negative_most_value.append(i[1])

print(negative_most_word)
print(negative_most_value)
```

```
['지퍼', '사이즈', '생각', '느낌', '후드', '기모', '자체', '교환', '기장', '오버', '엉덩이', '체형', '완전', '정도', '두께']
[2705, 1670, 1288, 1054, 1027, 985, 907, 705, 684, 655, 601, 581, 574, 510, 489]
```

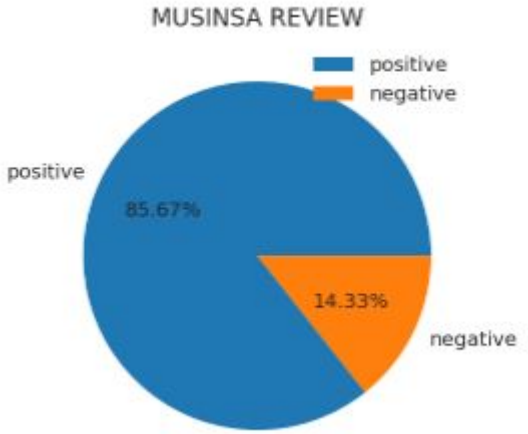
```
x2 = np.arange(len(negative_most_word))
plt.title('negative')
plt.bar(x2, negative_most_value)
plt.xticks(x2, negative_most_word)
plt.xticks(rotation=45)
plt.show()
```

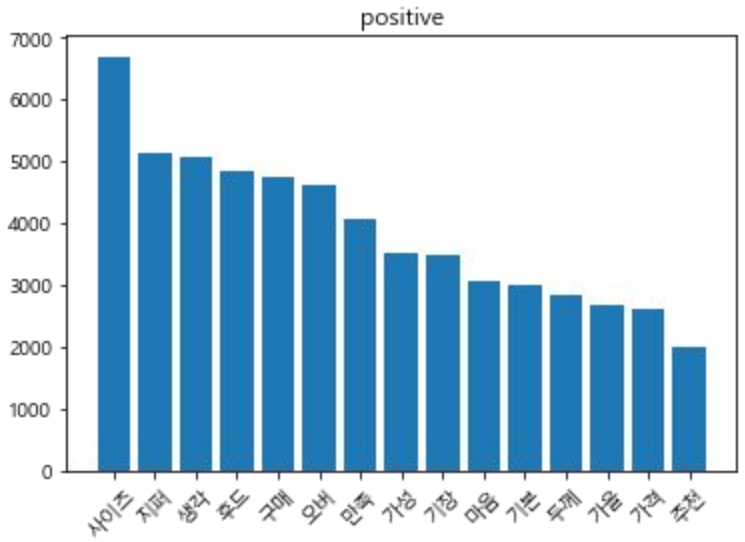


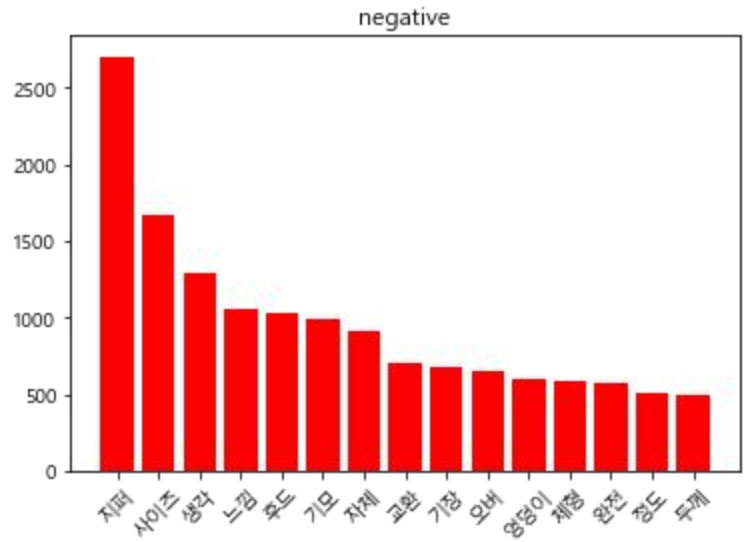
진행 결과

상품에 대한 리뷰의
전체적인 긍-부정 정도 파악

긍정 부정 리뷰 별
핵심 키워드 도식화



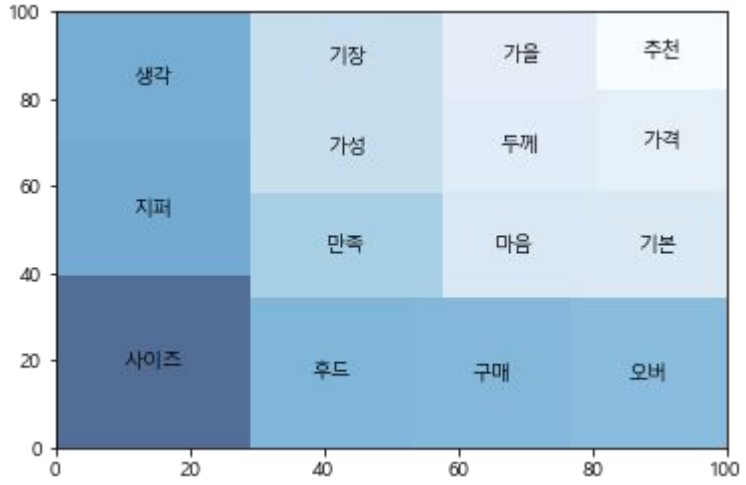


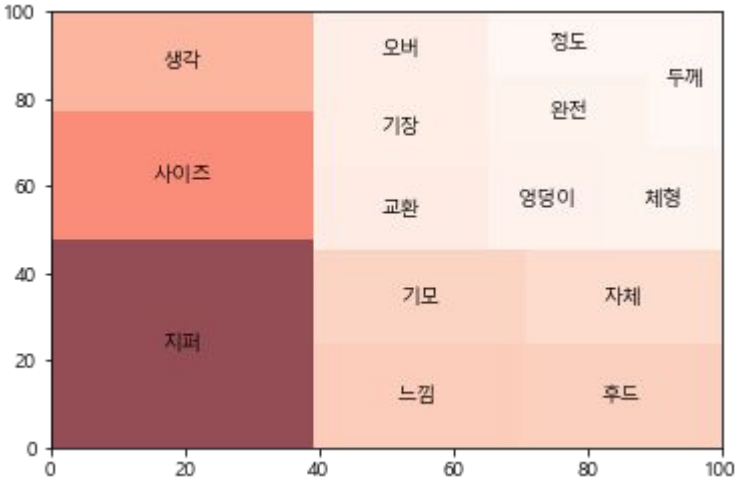


진행 결과

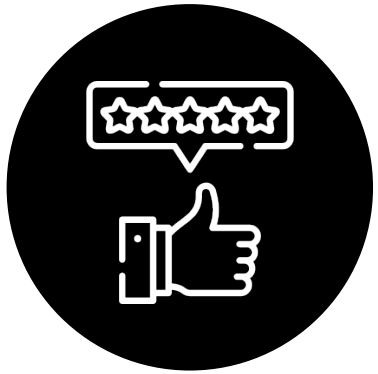

기장
만족
구매
가격
사이즈
색
추천
두께
생각
후드
가성
기본
어


후드
기장자체
오버
교환
지퍼
정도
생각
두께
기모
완전


생각
기장
가을
추천
가성
두께
가격
지퍼
만족
마음
기본
사이즈
후드
구매
오버

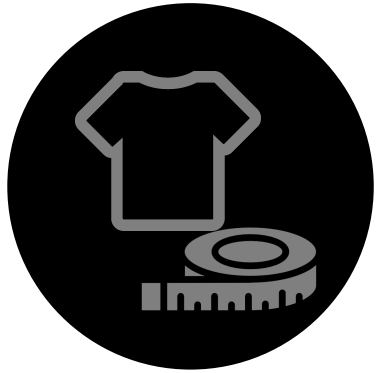

생각
오버
정도
두께
기장
완전
사이즈
교환
엉덩이
체형
기모
자체
지퍼
느낌
후드

상품 사이즈 예측 진행 방법



상품 리뷰 크롤링

앞서 크롤링한 리뷰수가
많은 상품 데이터 사용



KoBERT 다중 분류 모델

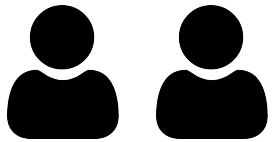
사용 사이즈 분류

실구매자의 성별, 키,
몸무게와 실제 구매
사이즈을 묶어서 학습시킴

분석 결과 도식화

학습 결과를 바탕으로
구매 희망자의 신체
사이즈를 입력값에 따른
추천 사이즈 도출

진행 과정



김정기: 실구매자의 구매사이즈 기반 사이즈 예측

① 학습할 리뷰 데이터를 전처리 (총 샘플의 수: 13000건)

※ 크롤링한 리뷰데이터의 성별, 키, 몸무게를 합치고
실 구매 사이즈를 정수형으로 타입 변경

```
musinsa_data['result']= musinsa_data['sex']+" "+musinsa_data['height']+" "+musinsa_data['weight']
```

```
musinsa_data.loc[(musinsa_data['size'] == "S"), 'size'] = 0  #S => 0
musinsa_data.loc[(musinsa_data['size'] == "M"), 'size'] = 1  #M => 1
musinsa_data.loc[(musinsa_data['size'] == "L"), 'size'] = 2  #L => 2
musinsa_data.loc[(musinsa_data['size'] == "XL"), 'size'] = 3  #XL => 3
```

	Unnamed: 0	sex	height	weight	clothname	size	review	result
0	0	남성	177cm	88kg	엔젤 와펜 집업 후드 그레이	3	두께감도있고 꽤 오버핏에 편하게 입기 좋아요~무난하게 입을듯	남성 177cm 88kg
1	1	남성	185cm	74kg	엔젤 와펜 집업 후드 그레이	3	생각했던것보다 핏이 더 좋게 나옵니다. 로고도 귀여워요 :)	남성 185cm 74kg
2	2	남성	168cm	65kg	엔젤 와펜 집업 후드 그레이	1	간절기 말고 초겨울에도 따뜻하게 입을 수 있을거 같습니다.생각보다 두꺼워요	남성 168cm 65kg
3	3	여성	168cm	70kg	엔젤 와펜 집업 후드 블랙	2	사이즈도 좋고 여러옷에 다 잘맞아요.와릿이즌 옷은 실패한적이 없어요.딱 좋아요.이쁘...	여성 168cm 70kg
4	4	남성	168cm	52kg	엔젤 와펜 집업 후드 그레이	1	진짜이뻐요 로고랑 뒤에 프린팅도 이쁘고 자주자주입을거같아요	남성 168cm 52kg
...	...	...	...	...	...	...	...	...
13236	13236	남성	175cm	75kg	엔젤 와펜 집업 후드 그레이	2	옷도 이쁘고 질도 괜찮은데 후드끈이 없이 배송이 됐네요 비싼돈주고 샀는데 후드끈이 ...	남성 175cm 75kg
13237	13237	남성	175cm	75kg	엔젤 와펜 집업 후드 그레이	1	이쁘고 기모 있어요 오버하게 입기 좋은것 같아요	남성 175cm 75kg
13238	13238	남성	175cm	75kg	엔젤 와펜 집업 후드 그레이	0	배송이 정말 느리네요... 9월2일에 주문했는데 9월 7일에 왔어요ㅠ	남성 175cm 75kg
13239	13239	남성	175cm	75kg	엔젤 와펜 집업 후드 블랙	0	키가 큰 편이라 사이즈 s 사면서 혹시 작을까 걱정했는데 하나도 안 작고 넉넉해요 ...	남성 175cm 75kg
13240	13240	남성	175cm	75kg	엔젤 와펜 집업 후드 그레이	2	좀 클 줄 알았는데 널널하게 오버핏 나오고 많이 안길어요! 기모도 잘 들어가있고 지...	남성 175cm 75kg

13241 rows × 8 columns

② train & test 데이터로 나누기

(사이킷런에서 제공하는 train_test_split 라이브러리를 이용하며, 4:1 비율로 나눔)

```
#train & test 데이터로 나누기
from sklearn.model_selection import train_test_split

dataset_train, dataset_test = train_test_split(data_list, test_size=0.25, random_state=0)
```

```
print(len(dataset_train))
print(len(dataset_test))
print(dataset_train[0])
print(dataset_test[0])
```

```
9930
3311
['남성  174cm  65kg', '1']
['남성 175cm 75kg', '2']
```

진행 과정

③ KoBERT 입력 데이터 만들기
(토큰화, 정수 인코딩, 패딩 등)

```
class BERTDataset(Dataset):
    def __init__(self, dataset, sent_idx, label_idx, bert_tokenizer, max_len,
                 pad, pair):
        transform = nlp.data.BERTSentenceTransform(
            bert_tokenizer, max_seq_length=max_len, pad=pad, pair=pair)

        self.sentences = [transform([i[sent_idx]]) for i in dataset]
        self.labels = [np.int32(i[label_idx]) for i in dataset]

    def __getitem__(self, i):
        return (self.sentences[i] + (self.labels[i], ))

    def __len__(self):
        return (len(self.labels))
```

(하이퍼 파라미터 조정)

```
## Setting parameters
max_len = 64  # 텍스트 데이터 최대 길이
batch_size = 64
warmup_ratio = 0.1
num_epochs = 10
max_grad_norm = 1
log_interval = 200
learning_rate =  5e-5
```

(BERTSentenceTransform로 토큰화와 패딩을 실행)

```
#토큰화
tokenizer = get_tokenizer()
tok = nlp.data.BERTSPTokenizer(tokenizer, vocab, lower=False)
```

```
data_train = BERTDataset(dataset_train, 0, 1, tok, max_len, True, False)
data_test = BERTDataset(dataset_test, 0, 1, tok, max_len, True, False)
```

(결과확인)

```
data_train[0]
```

```
(array([   2, 1419,  546,  157,  384,  620,  417,    3,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1], dtype=int32),
 array(8, dtype=int32),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       dtype=int32),
```

```
data_test[1]
```

```
(array([   2, 1419,  529,  214,  384,  529,  105,  417,    3,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1], dtype=int32),
 array(9, dtype=int32),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       dtype=int32),
```

(torch형식의 dataset을 만들기)

```
train_dataloader = torch.utils.data.DataLoader(data_train, batch_size=batch_size, num_workers=5)
test_dataloader = torch.utils.data.DataLoader(data_test, batch_size=batch_size, num_workers=5)
```

Python

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 5 worker processes in total. Our suggested max number of worker in current system is 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the worker number to avoid potential slowness/freeze if necessary.
  cpuset_checked))
```

진행 과정

④ KoBERT 학습모델 만들기

```
class BERTClassifier(nn.Module):
    def __init__(self,
                 bert,
                 hidden_size = 768,
                 num_classes=7,
                 dr_rate=None,
                 params=None):
        super(BERTClassifier, self).__init__()
        self.bert = bert
        self.dr_rate = dr_rate

        self.classifier = nn.Linear(hidden_size , num_classes)
        if dr_rate:
            self.dropout = nn.Dropout(p=dr_rate)

    def gen_attention_mask(self, token_ids, valid_length):
        attention_mask = torch.zeros_like(token_ids)
        for i, v in enumerate(valid_length):
            attention_mask[i][:v] = 1
        return attention_mask.float()

    def forward(self, token_ids, valid_length, segment_ids):
        attention_mask = self.gen_attention_mask(token_ids, valid_length)

        _, pooler = self.bert(input_ids = token_ids, token_type_ids = segment_ids.long(), attention_mask = attention_mask.float().to(token_ids.device))
        if self.dr_rate:
            out = self.dropout(pooler)
        return self.classifier(out)
```

진행 과정

⑤ BERT 모델 불러오기

```
model = BERTClassifier(bertmodel,  dr_rate=0.5).to(device)
```

```
# Prepare optimizer and schedule (linear warmup and decay)
no_decay = ['bias', 'LayerNorm.weight']
optimizer_grouped_parameters = [
    {'params': [p for n, p in model.named_parameters() if not any(nd in n for nd in no_decay)], 'weight_decay': 0.01},
    {'params': [p for n, p in model.named_parameters() if any(nd in n for nd in no_decay)], 'weight_decay': 0.0}
]
```

```
optimizer = AdamW(optimizer_grouped_parameters, lr=learning_rate)
loss_fn = nn.CrossEntropyLoss()
```

```
t_total = len(train_dataloader) * num_epochs
warmup_step = int(t_total * warmup_ratio)
```

```
scheduler = get_cosine_schedule_with_warmup(optimizer, num_warmup_steps=warmup_step, num_training_steps=t_total)
```

```
def calc_accuracy(X,Y):
    max_vals, max_indices = torch.max(X, 1)
    train_acc = (max_indices == Y).sum().data.cpu().numpy()/max_indices.size()[0]
    return train_acc
```

```
train_dataloader
```

```
<torch.utils.data.dataloader.DataLoader at 0x7fd48140d690>
```

진행 과정

⑥ KoBERT 모델 학습시키기

```
for e in range(num_epochs):
    train_acc = 0.0
    test_acc = 0.0
    model.train()
    for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(train_dataloader)):
        optimizer.zero_grad()
        token_ids = token_ids.long().to(device)
        segment_ids = segment_ids.long().to(device)
        valid_length= valid_length
        label = label.long().to(device)
        out = model(token_ids, valid_length, segment_ids)
        loss = loss_fn(out, label)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)
        optimizer.step()
        scheduler.step()  # Update learning rate schedule
        train_acc += calc_accuracy(out, label)
        if batch_id % log_interval == 0:
            print("epoch {} batch id {} loss {} train acc {}".format(e+1, batch_id+1, loss.data.cpu().numpy(), train_acc / (batch_id+1)))
    print("epoch {} train acc {}".format(e+1, train_acc / (batch_id+1)))

    model.eval()
    for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(test_dataloader)):
        token_ids = token_ids.long().to(device)
        segment_ids = segment_ids.long().to(device)
        valid_length = valid_length
        label = label.long().to(device)
        out = model(token_ids, valid_length, segment_ids)
        test_acc += calc_accuracy(out, label)
    print("epoch {} test acc {}".format(e+1, test_acc / (batch_id+1)))
```

```
epoch 10 batch id 1 loss 1.17534601688385 train acc 0.453125
epoch 10 train acc 0.5150240384615384
100% ████████████████████████████████████████ 52/52 [00:12<00:00, 4.57it/s]
epoch 10 test acc 0.49799253273322425
```

진행 과정

⑦ 모델 저장하기

```
import os
os.chdir('/content/drive/MyDrive/Colab Notebooks/미니프로젝트/res/')
os.getcwd()
```

'/content/drive/MyDrive/Colab Notebooks/미니프로젝트/res'

```
path = '/content/drive/MyDrive/Colab Notebooks/미니프로젝트/res/'
torch.save(model, path + 'size_model.pt') # 전체 모델 저장
```

```
torch.save(model.state_dict(), 'size_model_state_dict.pt') # 모델 객체의 state_dict 저장
```

```
torch.save({
    'model': model.state_dict(),
    'optimizer': optimizer.state_dict()
}, 'size_all.tar') # 여러 가지 값 저장, 학습 중 진행 상황 저장을 위해 epoch, loss 값 등 일반 scalar값 저장 가능
```

```
#★★★현재경로가 model이 있는 폴더여야함★★★
import os
os.chdir('/content/drive/MyDrive/Colab Notebooks/미니프로젝트/res/')

model1 = torch.load('size_model.pt') # 전체 모델을 통째로 불러옴, 클래스 선언 필수
model1.load_state_dict(torch.load('size_model_state_dict.pt')) # state_dict를 불러 온 후, 모델에 저장

checkpoint = torch.load('size_all.tar')  # dict 불러오기
model1.load_state_dict(checkpoint['model'])
optimizer.load_state_dict(checkpoint['optimizer'])
```

⑧ 결과 확인

```
[ ]  #질문 무한반복하기!
     end = 1
     while end == 1 :
         sentence = input("말을 입력해주세요 : ")
         if sentence == 0 :
             break
         predict(sentence)
         print("₩n")
```

```
말을 입력해주세요 : 남성 163cm 43kg
/usr/local/lib/python3.7/dist-packages/torch/ut
  cpuset_checked))
추천사이즈는
S입니다.
```

기대 효과

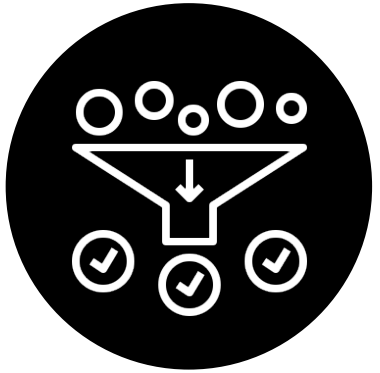
리뷰 감정 별 특징 파악

원하는 상품의 긍정 리뷰,
부정 리뷰의 각 키워드를
통해 감정 원인을
한눈에 파악할 수 있음.

상품 구매 결정에 도움

분석된 긍부정 퍼센트
비율과 키워드 도식화로
사용자 구매 결정에 도움

실구매자 리뷰 기반
사이즈 추천으로 교환 및
반품률 감소



별점의 한계점 극복과
광고성 리뷰 필터링 효과

제대로 매겨지지 않은 별점
한계 극복, 통계적 결과로
광고성 리뷰를 약간의
필터링이 가능

감사합니다