**감성 사전 활용 감성 분석**

1. 감성 사전 만들기

- 데이터 불러오기

**from** **sklearn.externals** **import** joblib

**with** open('amazon.pkl', 'rb') **as** f:

data = joblib.load(f)

locals().update(data)

tdm 을 확인해보자.

tdm

- 데이터 분할(학습용, 검증용)

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(tdm, sentiment, test\_size=.2, random\_state=1234)

- 모델 생성

**from** **sklearn.linear\_model** **import** LogisticRegressionCV

model 에 로지스틱 회귀분석을 불러온다.

model = LogisticRegressionCV()

.fit() 으로 tdm 과 점수를 나타내는 score 를 학습시킨다.

model.fit(X\_train, y\_train)

- score 확인

.score() 메소드로 으로 정확도를 확인할 수 있다.

model.score(X\_train, y\_train) #학습 데이터

model.score(X\_test, y\_test) #훈련 데이터

- 회귀계수 확인

model.coef\_ 로 회귀계수를 확인할 수 있다.

첫 10개 단어의 회귀계수를 확인해보자.

model.coef\_.shape #천개 변수의 회귀계수값을 가짐

(1, 1000)

model.coef\_[0, :10]

- 단어와 회귀 계수 프레임 생성

zip()를 사용하면 두개의 리스트 값을 짝지을 수 있다.

sent\_df = pd.DataFrame({'단어': vectorizer.get\_feature\_names(),

'계수': model.coef\_.flat})

sd.tail()

- 부정 단어 사전

0보다 작다 즉 음수는 부정이 되고, 양수는 긍정이 된다.

neg\_df = sent\_df[sent\_df['계수'] < 0].sort\_values(by='계수')

neg\_df.head()

- 긍정 단어 사전

pos\_df = sent\_df[sent\_df['계수'] > 0].sort\_values(by='계수', ascending=False)

pos\_df.head()

- 전체 감성 사전 저장

sent\_df.to\_csv('sent\_df.csv')

2. 감성 분석

- 사전 불러오기

**import** **pandas** **as** **pd**

sent\_df = pd.read\_csv('sent\_df.csv', index\_col=0)

- 극성 칼럼 추가

회귀분석을 통해 만든 감정 사전에서도 계수가 양수면 +1, 음수면 -1로 변환한다.

**import** **numpy**

sent\_df['극성'] = numpy.sign(sent\_df['계수'])

- 인덱스를 단어 칼럼으로 변경(매핑 편의를 위해서)

sent\_df.set\_index('단어', inplace=True)

sent\_df.head()

- 문장 극성 매핑

sentence = 'poor screen but reasonable price'

words = sentence.split() #형태소 분석으로 토큰화 해도 되지만 간단하게 하기위해

# 해당하는 사전에서 매핑

Sent\_df.reindex(words)

- OOV 즉 미 매핑 단어 삭제

sent = sent\_df.reindex(words).dropna()

sent

- 극성 계산

sent['극성'].sum() / sent['극성'].abs().sum()

**기계 학습을 통한 감성 분석**

1. 케라스 활용 로지스틱 회귀 분석

- 모형 만들기

**from** **keras** **import** Sequential

**from** **keras.layers** **import** Dense, Input

NUM\_WORDS = tdm.shape[1]

단어의 갯수(NUM\_WORDS)만큼 입력을 받아 1개의 출력을 내놓는 레이어를 추가한다. Dense는 선형 모형, sigmoid는 로지스틱 함수와 같다.

m1 = Sequential()

m1.add(Dense(1, activation='sigmoid', input\_shape=(NUM\_WORDS,)))

m1.summary()

- 학습

**from** **keras.optimizers** **import** Adam

데이터로 학습을 시킨다.

손실 함수는 크로스엔트로피를 쓴다. 문제에 따라 크로스엔트로피를 효과적으로 계산하는 방법이 달라지는데 둘 중에 하나로 분류할 때는 binary\_crossentropy를 사용한다. 3개 이상의 분류에는 categorical\_crossentropy를 사용한다.

최적화 알고리즘으로는 Adam을 사용한다.

m1.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=Adam())

m1.fit(X\_train, y\_train, epochs=30)

- 예측

y\_pred = m1.predict\_classes(X\_test)

- 결과

**from** **sklearn.metrics** **import** accuracy\_score

accuracy\_score(y\_test, y\_pred) #이전 실행한 사이킷런의 로지스틱 회귀랑 비교

2. 다층 신경망 생성

- 신경망 생성 및 학습

m2 = Sequential()

m2.add(Dense(100, activation='relu', input\_shape=(NUM\_WORDS,)))

m2.add(Dense(1, activation='sigmoid'))

m2.summary()

m2.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=Adam())

m2.fit(X\_train, y\_train, epochs=30, verbose=0)

<keras.callbacks.History at 0x1a5e2f88550>

y\_pred2 = m2.predict\_classes(X\_test)

accuracy\_score(y\_test, y\_pred2)

- 추가로 더 깊은 신경망 만들 때

m2.add(Dense(50, activation='relu', input\_shape=(100,)))

m2.add(Dense(100, activation='relu', input\_shape=(50,)))

Dense 층 추가

3. 네이버 영화 리뷰 별점 활용 감성 분석

- 데이터 불러오기(1주차 크롤링을 통해 저장)

import pandas as pd

review\_star = pd.read\_csv('movie\_review.csv',encoding='utf8')

- 전처리(정규식)

review\_star.tail()

review\_star['review']=review\_star['review'].str.replace(pat=r'[^ A-Za-z0-9가-힣+]',repl= r'', regex=True)

- 형태소 분석 및 벡트라이즈

from konlpy.tag import Komoran

tag = Komoran()

from sklearn.feature\_extraction.text import CountVectorizer

- TDM 생성

def kor\_noun(text):

words = []

for w in tag.nouns(text):

if len(w) > 1:

words.append(w)

return words

cv = CountVectorizer(tokenizer=kor\_noun, max\_features=1000)

tdm = cv.fit\_transform(review\_star['review'])

- 학습 및 검증 테이터 분리

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(tdm, review\_star['score'], test\_size=.2, random\_state =1234)

- 모델 생성

from sklearn.linear\_model import LogisticRegressionCV

logreg = LogisticRegressionCV()

logreg.fit(X\_train, y\_train)

- 예측 및 평가

y\_pred = logreg.predict(X\_test)

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test, y\_pred)

- 긍정 및 부정 단어 확인

words = cv.get\_feature\_names()

sent\_dict = sorted(zip(logreg.coef\_[0], words))

sent\_dict[:10]

sent\_dict[:-0]

- 분포 확인

%matplotlib inline

review\_star['score'].hist()

- split 재 수행

high\_low = review\_star['score'] > 5

X\_train, X\_test, y\_train, y\_test = train\_test\_split(tdm, high\_low, test\_size=.2, random\_state =1234)