인공지능 프로젝트 보고서

161655 허정화

1. 프로젝트 주제

금융뉴스 헤드라인 감정 분석

2. 기존 연구 소개(접근방법) 및 한계점

현재 우리 사회는 금융위기에 대한 불안감이 무척 큽니다. 이러한 금융 위기 예측 및 대비를 위해 시장 정서 파악에 대한 다양한 연구가 필요합니다. 즉, 신뢰할 수 있으며 시기적절하게 시장의 정서를 파악할 수 있는 방법이 필요합니다. 기존의 방법으로는 정기적으로 설문조사를 하여 계산된 신뢰도 지표와 같은 방법이 있는데 이는 정보가 무척 제한되며 결과가 시간이 지날수록 빠르게 뒤처진다는 한계가 있습니다. 따라서 설문조사에 기반한 정서 조사의 한계를 극복하기 위해 경제발전을 실시간으로 모니터링할 수 있는 기술에 대한 관심이 높아지는 추세입니다.
이를 극복하기 위한 가장 유망한 접근법 중 하나는 미디어를 투자자 입장에서 바라보는 것입니다. 월별로 최근 경제 발전에 대한 의견을 묻는 대신 뉴스를 활용하여 시장 정서의 진화를 시각화할 수 있습니다. 투자 정보는 주로 뉴스를 기반으로 하기에 미디어가 투자자의 감정과 시장 참여자의 행동에 영향을 준다고 가정하는 것은 매우 타당합니다.
따라서 본 프로젝트에서는 발전된 자연어 처리 기술을 이용하여 금융뉴스 헤드라인의 감정을 분류해볼 것입니다.

3. 사용 데이터 (수집 데이터)

데이터 이름 : Sentiment Analysis for Financial News

데이터 특징 : 소매 투자자 관점에서 금융 뉴스 헤드라인에 대한 감정이 라벨링 되어있음.

데이터 용도 : 텍스트 감정 분류 학습을 목적으로 사용

데이터 출처 : kaggle https://www.kaggle.com/ankurzing/sentiment-analysis-for-financial-news

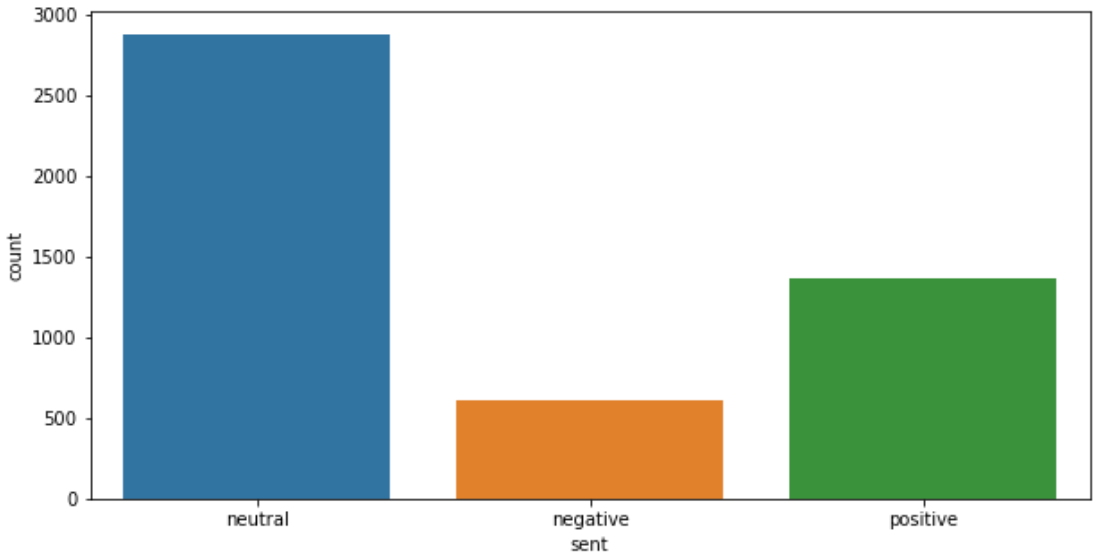
참고 : Malo, P., Sinha, A., Takala, P., Korhonen, P. and Wallenius, J. (2014): “Good debt or bad debt: Detecting semantic orientations in economic texts.” Journal of the American Society for Information Science and Technology.

4. 사용 데이터의 통계 (변수 별 평균, 분산 등) 및 전처리

	sentiment	text
0	neutral	According to Gran , the company has no plans t...
1	neutral	Technopolis plans to develop in stages an area...
2	negative	The international electronic industry company ...
3	positive	With the new production plant the company woul...
4	positive	According to the company 's updated strategy f...
...	...	...
4841	negative	LONDON MarketWatch -- Share prices ended lower...
4842	neutral	Rinkuskiai 's beer sales fell by 6.5 per cent ...
4843	negative	Operating profit fell to EUR 35.4 mn from EUR ...
4844	negative	Net sales of the Paper segment decreased to EU...
4845	negative	Sales in Finland decreased by 10.5 % in Januar...

4846 rows × 2 columns

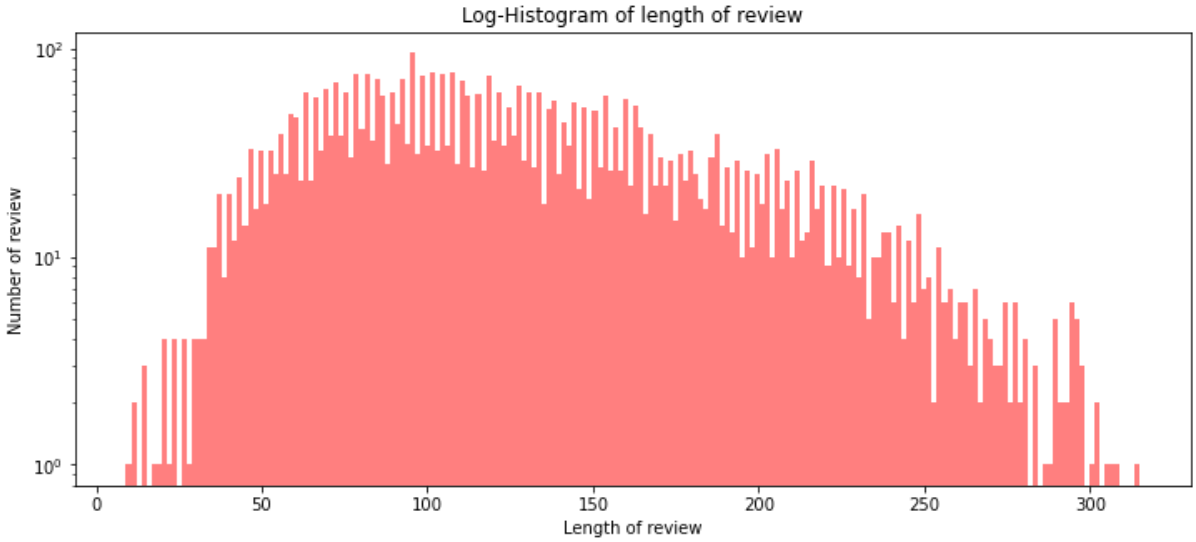
-데이터 분포



중립 개수: 2879
긍정 개수: 1363
부정 개수: 604

-Text 길이 분포

#그래프



길이 최대 값: 315

길이 최소 값: 9

길이 평균 값: 128.13

길이 표준편차: 56.52

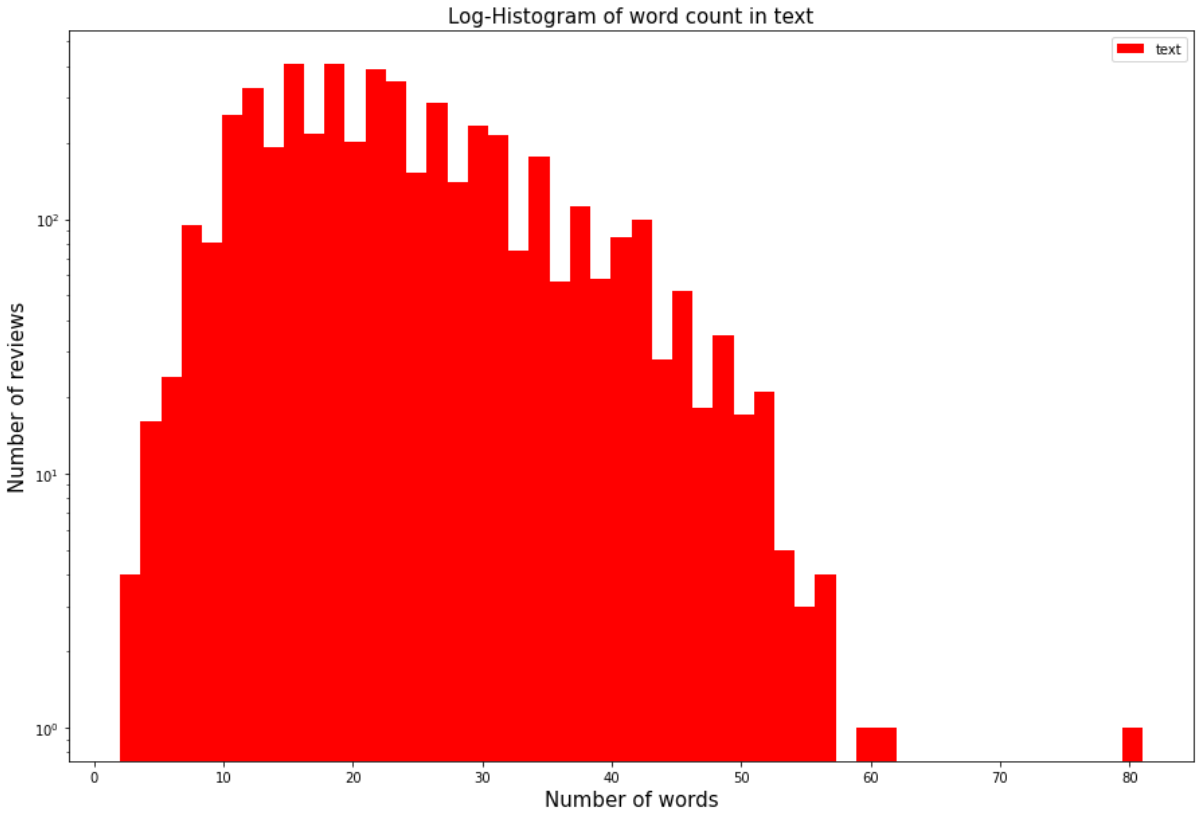
길이 중간 값: 119.0

길이 제 1 사분위: 84.0

길이 제 3 사분위: 163.0

-많이 사용된 단어

-리뷰 단어 개수 분포



단어 개수 최대 값: 81
단어 개수 최소 값: 2
단어 개수 평균 값: 23.11
단어 개수 표준편차: 9.96
단어 개수 중간 값: 21.0
단어 개수 제 1 사분위: 16.0
단어 개수 제 3 사분위: 29.0

-특수문자 및 대소문자 비율

물음표: 0.17%
마침표: 99.46%
첫 글자가 대문자: 92.88%
대문자: 99.83%
숫자: 53.32%
 # 학습에 방해되기 때문에 특수문자 제거 및 대문자->소문자 바꾸기가 필요함

-전처리

```
def preprocessing(text, remove_stopwords = False ):

    #1. 영어가 아닌 특수문자들을 공백(" ")으로 바꾸기
    text = re.sub("[^a-zA-Z]", " ", text)

    #2. 대문자들을 소문자로 바꾸고 공백단위로 텍스트들 나눠서 리스트로 만든다.
    words = text.lower().split()

    if remove_stopwords:
        #3. 불용어들을 제거
        #영어에 관련된 불용어 불러오기
        stops = set(stopwords.words("english"))
        # 불용어가 아닌 단어들로 이루어진 새로운 리스트 생성
        words = [w for w in words if not w in stops]
        # 4. 단어 리스트를 공백을 넣어서 하나의 글로 합친다.
        clean= ' '.join(words)

    else: # 불용어 제거하지 않을 때
        clean = ' '.join(words)

    return clean
```

```
clean = []
for d in data['text']:
    clean.append(preprocessing(d, remove_stopwords = True))

# 전처리한 데이터 출력
clean[10]
```

```
'teliasonera tlsn said offer line strategy increase ownership core business holdings
would strengthen eesti telekom offering customers'
```

1. 특수문자 공백으로 바꾸기 //특수문자는 문장의 의미에 크게 영향을 미치지 않는다
2. 대문자 -> 소문자 바꾸기 //불용어 제거 및 학습에 방해가 되기 때문에 소문자로 바꾼다.
3. 불용어 제거 //불용어란 문장에서 자주 출현하나 의미에 큰 영향을 주지 않는 데이터이다.

```
clean_df
```

	text	sentiment
0	according gran company plans move production r...	neutral
1	technopolis plans develop stages area less squ...	neutral
2	international electronic industry company elco...	negative
3	new production plant company would increase ca...	positive
4	according company updated strategy years baswa...	positive
...	...	...
4841	london marketwatch share prices ended lower lo...	negative
4842	rinkuskiai beer sales fell per cent million li...	neutral
4843	operating profit fell eur mn eur mn including ...	negative
4844	net sales paper segment decreased eur mn secon...	negative
4845	sales finland decreased january sales outside ...	negative

4846 rows × 2 columns

→전처리한 데이터는 csv파일로 저장하여 머신러닝에 사용한다.

-벡터화(Tokenizer 사용) : 각 단어를 인덱스로 벡터화한다.

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(clean)
text_sequences = tokenizer.texts_to_sequences(clean)
```

```
print(text_sequences[0])
```

```
[44, 3135, 2, 229, 524, 39, 84, 2485, 2, 609]
```

```
word_vocab = tokenizer.word_index
print(word_vocab)
```

```
{'eur': 1, 'company': 2, 'mn': 3, 'said': 4, 'finnish': 5, 'sales': 6, 'million':
7, 'net': 8, 'profit': 9, 'year': 10, 'finland': 11, 'group': 12, 'operating': 13,
'mln': 14, 'new': 15, 'business': 16, 'period': 17, 'quarter': 18, 'oyj': 19, 'sha
re': 20, 'market': 21, 'also': 22, 'services': 23, 'shares': 24, 'first': 25, 'eur
o': 26, 'helsinki': 27, 'loss': 28, 'compared': 29, 'today': 30, 'operations': 31,
'contract': 32, 'nokia': 33, 'mobile': 34, 'total': 35, 'per': 36, 'financial': 3
7, 'based': 38, 'production': 39, 'products': 40, 'corporation': 41, 'percent': 4
2, 'bank': 43, 'according': 44, 'companies': 45, 'technology': 46, 'hel': 47, 'cor
responding': 48, 'plant': 49, 'v': 50, 'service': 51, 'solutions': 52, 'constructi
on': 53, 'one': 54, 'increased': 55, 'capital': 56, 'well': 57, 'agreement': 58,
'investment': 59, 'increase': 60, 'customers': 61, 'rose': 62, 'value': 63, 'pct':
64, 'order': 65, 'oy': 66, 'stock': 67, 'end': 68, 'january': 69, 'second': 70, 't
wo': 71, 'would': 72, 'board': 73, 'usd': 74, 'unit': 75, 'omx': 76, 'part': 77,
'development': 78, 'industry': 79, 'building': 80, 'management': 81, 'earlier': 8
2, 'september': 83, 'russia': 84, 'equipment': 85, 'paper': 86, 'expected': 87, 'l
ast': 88, 'result': 89, 'third': 90, 'project': 91, 'decreased': 92, 'ceo': 93, 'e
mployees': 94, 'us': 95, 'maker': 96, 'deal': 97, 'signed': 98, 'price': 99, 'soft
ware': 100, 'media': 101, 'plc': 102, 'systems': 103, 'announced': 104, 'real': 10
5, 'markets': 106, 'annual': 107, 'number': 108, 'exchange': 109, 'including': 11
```

-패딩(pad_sequences 사용) : 일정한 길이의 입력값을 만들기 위해서 패딩한다.

```
MAX_SEQUENCE_LENGTH = 21

inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')

print('Shape of data: ', inputs.shape)
```

```
Shape of data:  (4846, 21)
```

Max_length는 단어개수의 중간값 사용한다.

이유 : 평균은 일부 데이터의 길이가 지나치게 길면 급격히 올라가기 때문에 자연어처리에서는 쓰이면 위험하기 때문이다.

→벡터화 및 패딩까지 마친 데이터는 딥러닝에 사용한다.

-데이터 저장

```
# 전처리 된 데이터를 넘파이 형태로 저장
np.save('data_input.npy', inputs)
np.save('data_label.npy', labels)

# 정제된 텍스트를 csv 형태로 저장
clean_df.to_csv('data_clean.csv', index = False)


# 데이터 사전을 json 형태로 저장
json.dump(data_configs,open('data_configs.json','w'), ensure_ascii=False)
```

5. 모델링 및 평가

<머신러닝>

1. TF-IDF로 text벡터화 한 경우

-TF-IDF : TF(특정 단어가 하나의 데이터에서 등장한 횟수) X I(역수)DF(특정단어가 여러 데이터에서 등장한 횟수)로 벡터화를 시키는 도구이다.

이를 통해 특정 단어가 하나의 데이터에서 아무리 많이 등장해도 여러데이터에서 많이 등장해버리면 값이 작아진다.

```
vectorizer = TfidfVectorizer(min_df = 0.0, analyzer="word"
                             , sublinear_tf=True, ngram_range=(1,3), max_features=10000)
```

*파라미터 설명

min_df	df값이 0.0보다 적게 나오면 제거
analyzer	"word" , 단어를 하나의 단위로
sublinear_tf	스무딩여부 TF->1 + ln(TF)
ngram_range(1,3)	단어 묶음을 1~3개
max_features	1000 벡터의 최대 길이

처음에는 analyzer를 char로 했으나 성능이 낮게 나와 word로 지정해주었다.

```
X = vectorizer.fit_transform(text).toarray()
y = np.array(sentiments)
```

```
X.shape
```

```
(4845, 10000)
```

-> text에 벡터화 적용

***이 때 데이터가 4845개로 줄어든 이유는 불용성 제거를 해주었을 때 한 개의 데이터가 null값이 되어 학습에 지장을 주기에 제거해주었다.**

```
train_data = pd.read_csv( 'data_clean.csv' )
train_data = train_data[train_data['text'].notnull()]
```

```
text = list(train_data['text'])
sentiments = list(train_data['sentiment'])
```

-data 나누기

```
RANDOM_SEED = 42
TEST_SPLIT = 0.2

X_train, X_eval, y_train, y_eval = train_test_split(X, y, test_size=TEST_SPLIT, random_state=RANDOM_SEED)
```

0.2의 비율로 train과 test data를 나누어 주었다.

-SVM

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear')
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_eval)
print("Accuracy: %f" % classifier.score(X_eval, y_eval))
print(classification_report(y_eval,y_pred))
```

```
Accuracy: 0.753354
              precision    recall  f1-score   support

    negative       0.78      0.45      0.57       113
     neutral       0.75      0.93      0.83       567
    positive       0.75      0.52      0.61       289

    accuracy                           0.75       969
   macro avg       0.76      0.63      0.67       969
weighted avg       0.75      0.75      0.74       969
```

-LogisticRegression

```
lgs = LogisticRegression(class_weight='balanced')
lgs.fit(X_train, y_train)
y_pred = lgs.predict(X_eval)
print("Accuracy: %f" % lgs.score(X_eval, y_eval))
print(classification_report(y_eval,y_pred))
```

```
Accuracy: 0.753354
              precision    recall  f1-score   support

    negative       0.64      0.60      0.62       113
     neutral       0.79      0.86      0.83       567
    positive       0.70      0.60      0.65       289

    accuracy                           0.75       969
   macro avg       0.71      0.69      0.70       969
weighted avg       0.75      0.75      0.75       969
```

-RandomForest

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators = 1000)

forest.fit( X_train, y_train )
y_pred = forest.predict(X_eval)
print("Accuracy: %f" % forest.score(X_eval, y_eval))
print(classification_report(y_eval,y_pred))
```

```
Accuracy: 0.734778
              precision    recall  f1-score   support

    negative       0.73      0.32      0.44       113
     neutral       0.73      0.96      0.83       567
    positive       0.76      0.45      0.57       289

    accuracy                           0.73       969
   macro avg       0.74      0.58      0.61       969
weighted avg       0.74      0.73      0.71       969
```

2. CounterVectorizer로 text벡터화 한 경우

-CounterVectorizer: 텍스트에서 횟수를 측정한 뒤 벡터로 만들어준다.

이때 횟수 기준을 word로 잡았다.

```
train_data = pd.read_csv( 'data_clean.csv' )
train_data = train_data[train_data['text'].notnull()]
```

```
text = list(train_data['text'])
y = list(train_data['sentiment'])
```

```
vectorizer = CountVectorizer(analyzer = "word", max_features = 1000)

train_data_features = vectorizer.fit_transform(text)
```

```
train_data_features.shape
```

```
(4845, 1000)
```

-데이터나누기

```
TEST_SIZE = 0.2
RANDOM_SEED = 42

train_input, eval_input, train_label, eval_label = train_test_split(train_data_features, y, test_size=TEST_SIZE, random_state=RANDOM_SEED)
```

0.2의 비율로 train과 test data를 나누어 주었다.

-SVM

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear')
classifier.fit(train_input, train_label)
y_pred = classifier.predict(eval_input)
print("Accuracy: %f" % classifier.score(eval_input, eval_label))
print(classification_report(eval_label,y_pred))
```

```
Accuracy: 0.706914
              precision    recall  f1-score   support

    negative       0.53      0.46      0.49       113
     neutral       0.75      0.84      0.79       567
    positive       0.66      0.55      0.60       289

    accuracy                           0.71       969
   macro avg       0.65      0.62      0.63       969
weighted avg       0.70      0.71      0.70       969
```

-LogisticRegression

```
lgs = LogisticRegression(class_weight='balanced')
lgs.fit(train_input, train_label)
y_pred = lgs.predict(eval_input)
print("Accuracy: %f" % lgs.score(eval_input, eval_label))
print(classification_report(eval_label,y_pred))
```

```
Accuracy: 0.710010
              precision    recall  f1-score   support

    negative       0.49      0.62      0.55       113
     neutral       0.81      0.77      0.78       567
    positive       0.64      0.64      0.64       289

    accuracy                           0.71       969
   macro avg       0.65      0.67      0.66       969
weighted avg       0.72      0.71      0.71       969
```

-RandomForest

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators = 1000)

forest.fit( train_input, train_label )
y_pred = forest.predict(eval_input)
print("Accuracy: %f" % forest.score(eval_input, eval_label))
print(classification_report(eval_label,y_pred))
```

```
Accuracy: 0.730650
              precision    recall  f1-score   support

    negative       0.63      0.42      0.51       113
     neutral       0.75      0.89      0.82       567
    positive       0.69      0.54      0.61       289

    accuracy                           0.73       969
   macro avg       0.69      0.62      0.64       969
weighted avg       0.72      0.73      0.72       969
```

-결과분석

TF-IDF로 벡터화 후 학습한 로지스틱회귀모델이 가장 정확도가 높았으며 f1스코어도 고루 높았다. 따라서 가장 성능이 좋은 모델이라고 보여진다.

모델	정확도
로지스틱회귀(TF-IDF)	0.75335
서포트벡터머신(TF-IDF)	0.75335
랜덤포레스트(TF-IDF)	0.73478
로지스틱회귀(CountVt)	0.71001
서포트벡터머신(CountVt)	0.70691
랜덤포레스트(CountVt)	0.73065

<딥러닝>

1. 데이터 불러오기 및 전처리

to_categorical을 이용해 원핫인코딩을 해준다.

1이있는 인덱스가 0인경우 negative, 1인경우 neutral, 2인경우 positive 이다.

```
input_data = np.load('data_input.npy',allow_pickle=True)
label_data = np.load('data_label.npy',allow_pickle=True)

prepro_configs = json.load(open('data_configs.json','r'))
```

```
from tensorflow.keras.utils import to_categorical
label_data_encoded = to_categorical(label_data)
```

```
#0=negative, 1=neutral, 2=positive
label_data_encoded
```

```
array([[0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       ...,
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]], dtype=float32)
```

2. 데이터 나눠주기

```
TEST_SPLIT = 0.2
RANDOM_SEED = 13371447

train_input, test_input, train_label, test_label = train_test_split(input_data, label_data_encoded,
                                                                    test_size=TEST_SPLIT, random_state=RANDOM_SEED)
```

```
print('훈련 샘플 본문의 크기 : {}'.format(train_input.shape))
print('훈련 샘플 레이블의 크기 : {}'.format(train_label.shape))
print('테스트 샘플 본문의 크기 : {}'.format(test_input.shape))
print('테스트 샘플 레이블의 크기 : {}'.format(test_label.shape))
```

```
훈련 샘플 본문의 크기 : (3876, 21)
훈련 샘플 레이블의 크기 : (3876, 3)
테스트 샘플 본문의 크기 : (970, 21)
테스트 샘플 레이블의 크기 : (970, 3)
```

***vocab_size 지정(토크나이저로 단어 인덱스화할 때 쓰인 단어 개수)**

```
VOCAB_SIZE = prepro_configs['vocab_size']+1
```

-RNN

```
def fit_and_evaluate(X_train, y_train, X_test, y_test):
    model = Sequential()
    model.add(Embedding(VOCAB_SIZE, 100))
    model.add(LSTM(150,activation='tanh'))
    model.add(Dense(3, activation='softmax'))

    mc = ModelCheckpoint('best_model_rnn.h5', monitor = 'val_acc', mode = 'max', verbose = 1, save_best_only = True)

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    hist=model.fit(X_train, y_train, batch_size=128, epochs=100, validation_data=(X_test, y_test),callbacks=[mc])

    plt.figure(figsize=(8, 4))
    plt.subplot(1, 2, 1)
    plt.plot(hist.history['loss'], 'b-', label="training")
    plt.plot(hist.history['val_loss'], 'r:', label="validation")
    plt.title("loss")
    plt.subplot(1, 2, 2)
    plt.title("accuracy")
    plt.plot(hist.history['acc'], 'b-', label="training")
    plt.plot(hist.history['val_acc'], 'r:', label="validation")
    plt.legend()
    plt.tight_layout()
    plt.show()
```

```
fit_and_evaluate(train_input,train_label,test_input,test_label) # 모델을 훈련하고 평가.
```

→input층: embedding층, Embedding(input_dim(첫번째 인자)=총 단어 개수, output_dim(두번째 인자)=임베딩 벡터의 출력 차원)

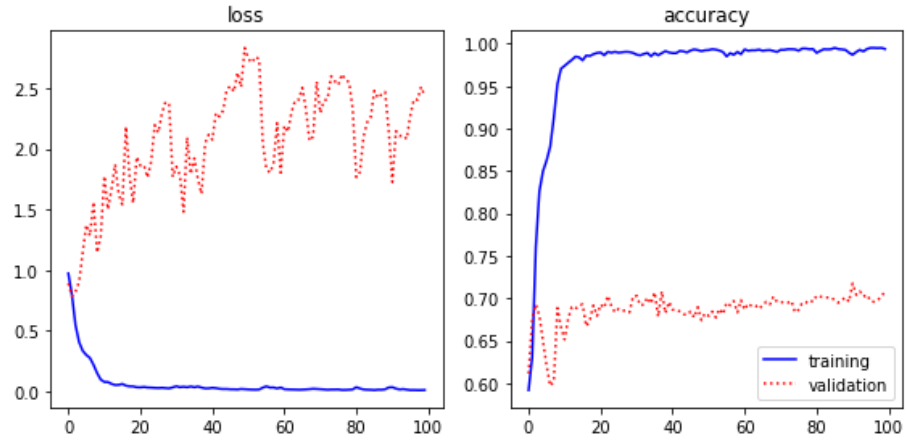
- ✓ input_length=입력 시퀀스의 길이(여기서는 21) 지정하던 안하던 결과는 같다.

→output층: Denselayer로 softmax함수 활용

[0.09909477, 0.8757712 , 0.02513401] 이런 형태로 출력한다.

이 출력값은 인덱스 1의 probability가 크므로 neutral로 판단한 것이다.

학습완료 후 결과:



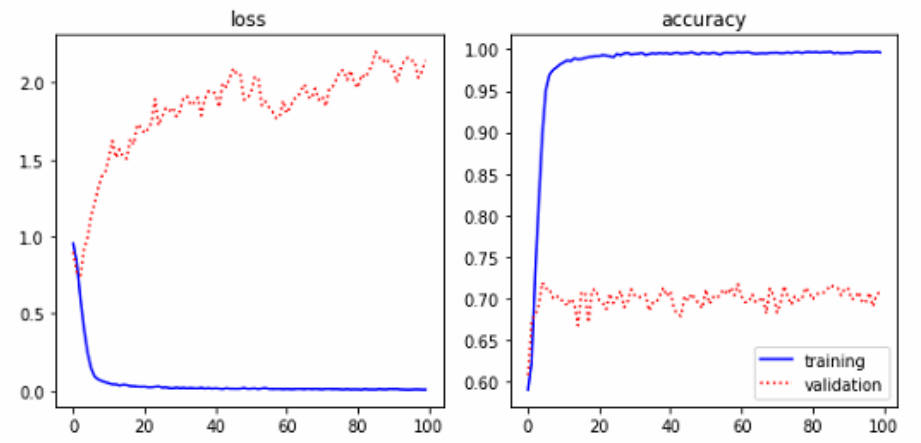

```
970/970 [==============================] - 1s 1ms/step

 테스트 loss: 1.7265
 테스트 정확도: 0.7175
```

-RNN+CNN

```
def fit_and_evaluate(X_train, y_train, X_test, y_test):
    model = Sequential()
    model.add(Embedding(VOCAB_SIZE, 100))
    model.add(Dropout(0.3))
    model.add(Conv1D(128,3, padding= 'valid', activation='relu'))
    model.add(MaxPooling1D(pool_size=4))
    model.add(LSTM(56,activation='tanh'))
    model.add(Dense(3,activation='softmax'))

    mc = ModelCheckpoint('best_model_cnn_rnn.h5', monitor = 'val_acc', mode = 'max', verbose = 1, save_best_only = True)

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    hist=model.fit(X_train, y_train, batch_size=128, epochs=100, validation_data=(X_test, y_test),callbacks=[mc])

    plt.figure(figsize=(8, 4))
    plt.subplot(1, 2, 1)
    plt.plot(hist.history['loss'], 'b-', label="training")
    plt.plot(hist.history['val_loss'], 'r:', label="validation")
    plt.title("loss")
    plt.subplot(1, 2, 2)
    plt.title("accuracy")
    plt.plot(hist.history['acc'], 'b-', label="training")
    plt.plot(hist.history['val_acc'], 'r:', label="validation")
    plt.legend()
    plt.tight_layout()
    plt.show()
```

```
fit_and_evaluate(train_input,train_label,test_input,test_label) # 모델을 훈련하고 평가.
loaded_model = load_model('best_model_cnn_rnn.h5')
print("₩n 테스트 정확도: %.4f" % (loaded_model.evaluate(test_input,test_label)[1]))
```

학습완료 후 결과:



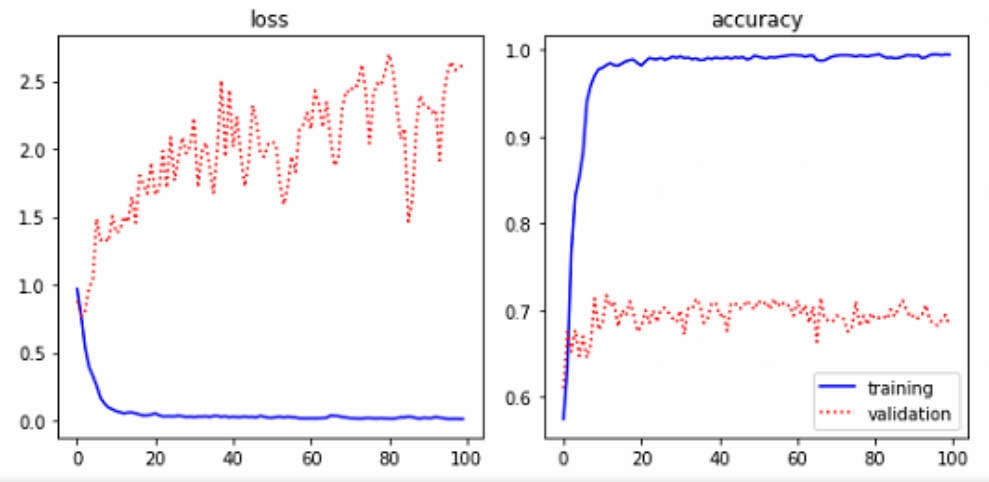

```
970/970 [==============================] - 1s 1ms/step

테스트 loss: 0.9917
테스트 정확도: 0.7196
```

-CNN(Conv1D 1개)

```
def fit_and_evaluate(X_train, y_train, X_test, y_test):
    model = Sequential()
    model.add(Embedding(VOCAB_SIZE, 100))
    model.add(Dropout(0.3))
    model.add(Conv1D(256,3, padding= 'valid', activation='relu'))
    model.add(GlobalMaxPooling1D())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(3, activation='softmax'))

    mc = ModelCheckpoint('best_model_cnn.h5', monitor = 'val_acc', mode = 'max', verbose = 1, save_best_only = True)

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    hist=model.fit(X_train, y_train, batch_size=128, epochs=100, validation_data=(X_test, y_test),callbacks=[mc])

    plt.figure(figsize=(8, 4))
    plt.subplot(1, 2, 1)
    plt.plot(hist.history['loss'], 'b-', label="training")
    plt.plot(hist.history['val_loss'], 'r:', label="validation")
    plt.title("loss")
    plt.subplot(1, 2, 2)
    plt.title("accuracy")
    plt.plot(hist.history['acc'], 'b-', label="training")
    plt.plot(hist.history['val_acc'], 'r:', label="validation")
    plt.legend()
    plt.tight_layout()
    plt.show()
```

```
fit_and_evaluate(train_input,train_label,test_input,test_label) # 모델을 훈련하고 평가.
```

학습완료 후 결과:




```
970/970 [==============================] - 1s 1ms/step

테스트 loss: 1.2098
테스트 정확도: 0.7402
```

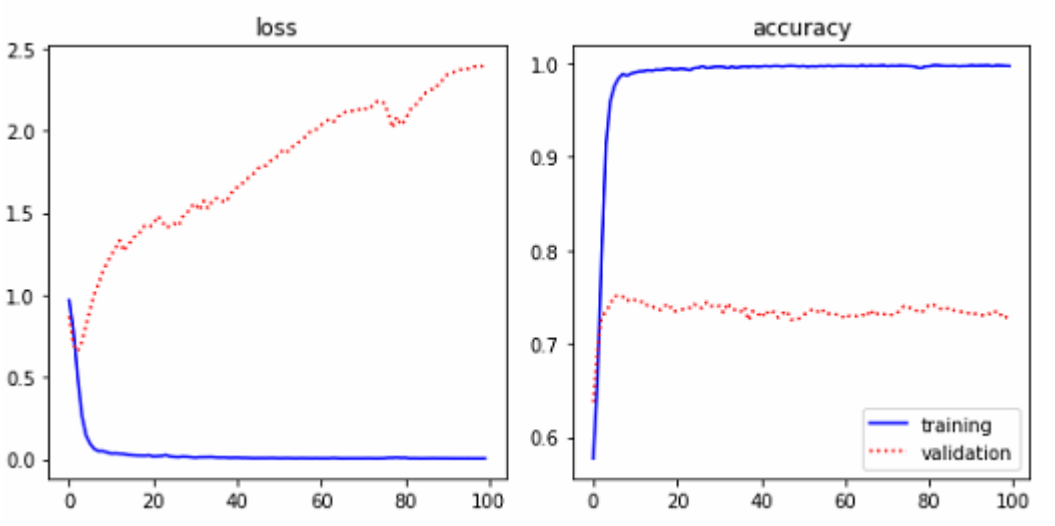
CNN만 사용시 이전보다 정확도가 크게 증가한 것을 볼 수 있다.

- CNN(Conv2D 1개)

```
def imdb_cnn_2():
    model = Sequential()

    model.add(Embedding(input_dim = num_features, output_dim = embedding_dimension,input_length = sequence_length))
    model.add(Reshape((sequence_length, embedding_dimension, 1), input_shape = (sequence_length, embedding_dimension)))
    model.add(Conv2D(filters = 128, kernel_size = (5, embedding_dimension), strides = (1,1), padding = 'valid'))
    model.add(GlobalMaxPooling2D())
    model.add(Dense(64))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(3))
    model.add(Activation('softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam' , metrics=['accuracy'])

    return model

model = imdb_cnn_2()
```

```
mc = ModelCheckpoint('best_model_cnn2.h5', monitor = 'val_acc', mode = 'max', verbose = 1, save_best_only = True)
hist= model.fit(train_input,train_label, batch_size = 128, epochs = 100, validation_data=(test_input,test_label),callbacks=[mc])

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(hist.history['loss'], 'b-', label="training")
plt.plot(hist.history['val_loss'], 'r:', label="validation")
plt.title("loss")
plt.subplot(1, 2, 2)
plt.title("accuracy")
plt.plot(hist.history['acc'], 'b-', label="training")
plt.plot(hist.history['val_acc'], 'r:', label="validation")
plt.legend()
plt.tight_layout()
plt.show()

loaded_model = load_model('best_model_cnn2.h5')
print("₩n 테스트 정확도: %.4f" % (loaded_model.evaluate(test_input,test_label)[1]))
```

학습완료 후 결과:




```
970/970 [==============================] - 1s 1ms/step

 테스트 loss: 1.0050
 테스트 정확도: 0.7515
```

위와 같은 구조에서 Conv1D대신 Conv2D를 사용했더니 loss가 감소하고 정확도가 증가했다.

- **CNN(Yoonkim박사의 CNN for sentence classification)**

출처: Convolutional Neural Networks for Sentence Classification - Yoon Kim (New York University)

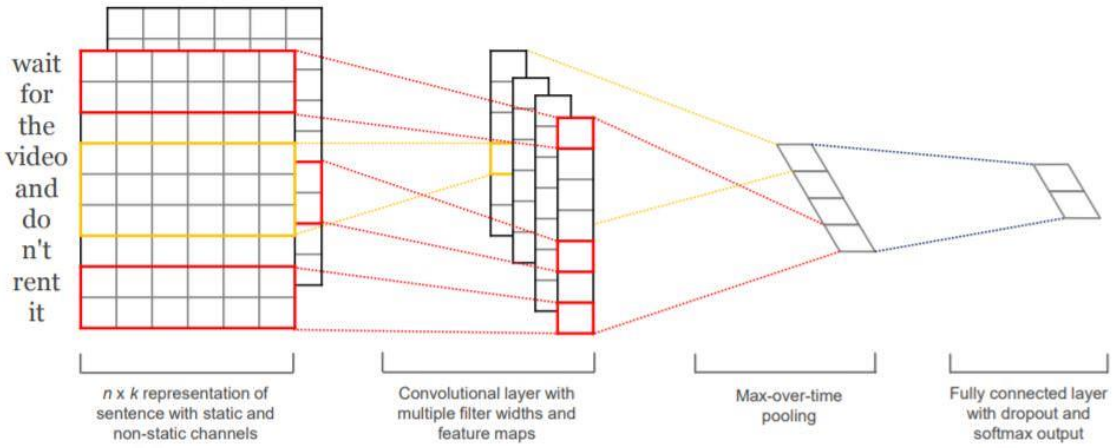
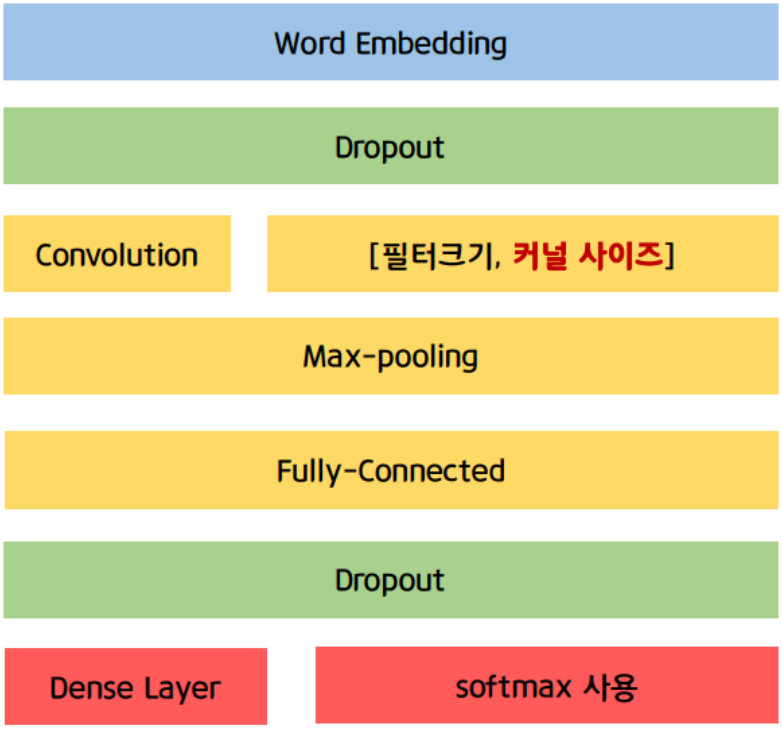


Figure 1: Model architecture with two channels for an example sentence.



커널 사이즈를 다르게 줌으로 써 N-gram의 효과 즉, 다양한 각도에서 문장을 보는 효과가 있다고 한다.

이 모델을 직접 구현해 보았으며, convolution layer에 filters=128, kernel_size=[3,4,5]로 주었다.

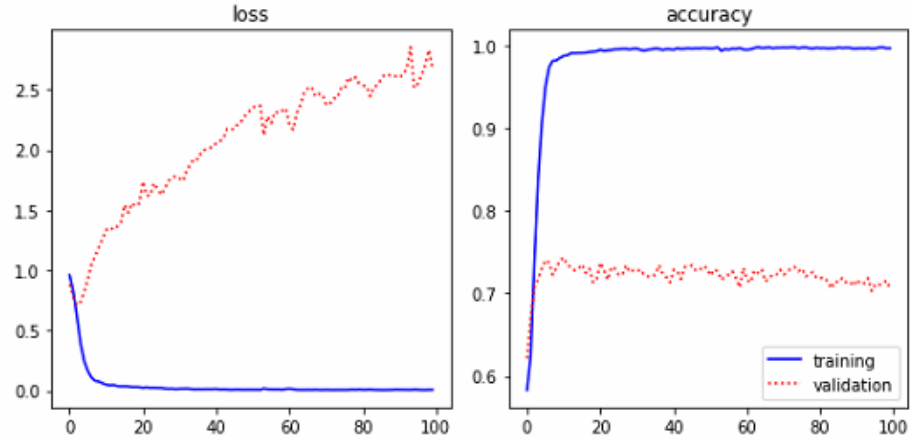
Conv1D와 Conv2D를 사용했을 때를 비교해 보았다.

-Conv1D 3개 사용

```
filter_sizes = [3, 4, 5]

def convolution():
    inn = Input(shape = (sequence_length, embedding_dimension))
    convolutions = []
    # we conduct three convolutions & poolings then concatenate them,
    for fs in filter_sizes:
        conv = Conv1D(filters = 128, kernel_size = fs, activation=tf.nn.relu , padding = "valid")(inn)
        maxpool = GlobalMaxPooling1D()(conv)
        convolutions.append(maxpool)

    outt = concatenate(convolutions)
    model = Model(inputs = inn, outputs = outt)

    return model

def imdb_cnn_4():

    model = Sequential()
    model.add(Embedding(input_dim = VOCAB_SIZE, output_dim = embedding_dimension, input_length = sequence_length))
    # call convolution method defined above
    model.add(convolution())

    model.add(Flatten())
    model.add(Dense(50))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(3))
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam' , metrics=['accuracy'])

    return model

model = imdb_cnn_4()
```

```
mc = ModelCheckpoint('best_model_cnn4.h5', monitor = 'val_acc', mode = 'max', verbose = 1, save_best_only = True)
hist= model.fit(train_input,train_label, batch_size = 128, epochs = 100, validation_data=(test_input,test_label),callbacks=[mc])

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(hist.history['loss'], 'b-', label="training")
plt.plot(hist.history['val_loss'], 'r:', label="validation")
plt.title("loss")
plt.subplot(1, 2, 2)
plt.title("accuracy")
plt.plot(hist.history['acc'], 'b-', label="training")
plt.plot(hist.history['val_acc'], 'r:', label="validation")
plt.legend()
plt.tight_layout()
plt.show()

loaded_model = load_model('best_model_cnn4.h5')
a=loaded_model.evaluate(test_input,test_label)
print("₩n 테스트 loss: %.4f ₩n 테스트 정확도: %.4f" % (a[0],a[1]))
```

학습완료 후 결과 :




```
970/970 [==============================] - 1s 648us/step

테스트 loss: 1.3348
테스트 정확도: 0.7433
```

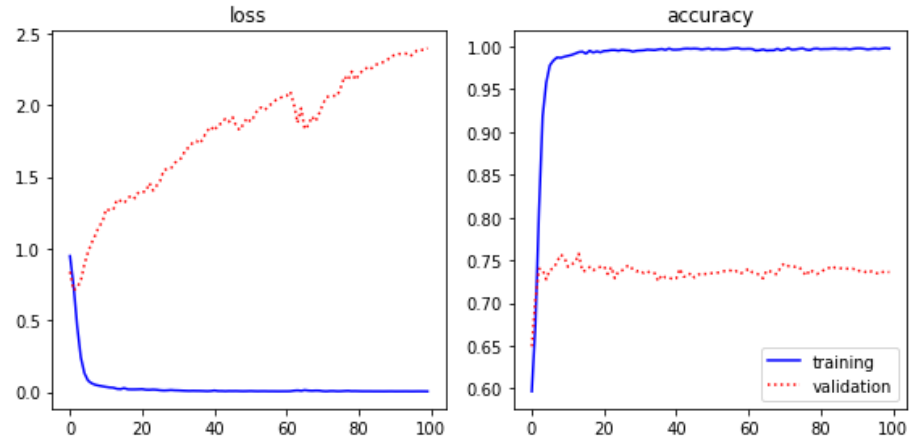
Conv1D를 1개 사용했을 때 보다 정확도가 올랐다.

-Conv2D 3개 사용

```
filter_sizes = [3, 4, 5]

def convolution():
    inn = Input(shape = (sequence_length, embedding_dimension, 1))
    convolutions = []
    # we conduct three convolutions & poolings then concatenate them.
    for fs in filter_sizes:
        conv = Conv2D(filters = 128, kernel_size = (fs, embedding_dimension), strides = 1, padding = "valid")(inn)
        maxpool = GlobalMaxPooling2D()(conv)
        convolutions.append(maxpool)

    outt = concatenate(convolutions)
    model = Model(inputs = inn, outputs = outt)

    return model

def imdb_cnn_3():

    model = Sequential()
    model.add(Embedding(input_dim = VOCAB_SIZE, output_dim = embedding_dimension, input_length = sequence_length))
    model.add(Reshape((sequence_length, embedding_dimension, 1), input_shape = (sequence_length, embedding_dimension)))

    # call convolution method defined above
    model.add(convolution())

    model.add(Flatten())
    model.add(Dense(50))
    model.add(Activation('relu'))
    model.add(Dropout(0.3))
    model.add(Dense(3))
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam' , metrics=['accuracy'])

    return model

model = imdb_cnn_3()
```

```
mc = ModelCheckpoint('best_model_cnn3.h5', monitor = 'val_acc', mode = 'max', verbose = 1, save_best_only = True)
hist= model.fit(train_input,train_label, batch_size = 128, epochs = 100, validation_data=(test_input,test_label),callbacks=[mc])

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(hist.history['loss'], 'b-', label="training")
plt.plot(hist.history['val_loss'], 'r:', label="validation")
plt.title("loss")
plt.subplot(1, 2, 2)
plt.title("accuracy")
plt.plot(hist.history['acc'], 'b-', label="training")
plt.plot(hist.history['val_acc'], 'r:', label="validation")
plt.legend()
plt.tight_layout()
plt.show()

loaded_model = load_model('best_model_cnn3.h5')
a=loaded_model.evaluate(test_input,test_label)
print("₩n 테스트 loss: %.4f ₩n 테스트 정확도: %.4f" % (a[0],a[1]))
```

학습완료 후 결과 :




```
970/970 [==============================] - 0s 453us/step

 테스트 loss: 1.3422
 테스트 정확도: 0.7577
```

이 때까지 학습한 모델 중 가장 높은 정확도를 가졌다.

-결과분석

이 데이터의 감정분류는 RNN보다 CNN이 적합하며, Conv2D를 3개를 이용할 때 정확도가 가장 높다.

모델	정확도
CNN2D 3개	0.7577
CNN1D 3개	0.7433
CNN2D 1개	0.7515
CNN1D 1개	0.7402
CNN1D+RNN	0.7196
RNN	0.7175

6. 기대효과

이 학습된 모델의 성능을 좀더 향상 시켜 실제로 사용한다면 지금도 무수히 쏟아져 나오고 있는 경제 뉴스에 대한 감정을 실시간으로 분류할 수 있을 것입니다.

더 나아가 분류된 데이터를 통해 앞으로 다가올 경제 위기 또는 호황을 예측할 수 있을 것이며 주가예측에도 사용될 수 있을 거라 기대됩니다.