자연어 텍스트 기반

감정 분석 딥러닝

PROJECT

김경태, 김재성, 안효준, 이민하

2024.09.23

주제 선정 배경

- 현대 사회는 소셜 미디어가 발달하면서 방대한 양의 텍스트가 생성 됨
- 텍스트 데이터를 통해 이용자들의 감정을 파악하는 것이 중요해짐

트위터(Twitter) 문장

감정 분석

안효준

데이터 출처

CHAITHANYA KUMAR A · UPDATED 5 YEARS AGO

142 | New Notebook | Download (11 MB)

Twitter and Reddit Sentimental analysis Dataset

Tweets and Comments extracted from Twitter and Reddit For Sentimental Analysis.

Data Card | Code (61) | Discussion (4) | Suggestions (0)

About Dataset

Context

This is was a Dataset Created as a part of the university Project On Sentimental Analysis On Multi-Source Social Media Platforms using PySpark.

There two datasets Respectively one Consists of Tweets from Twitter with Sentimental Label and the other from Reddit which Consists of Comments with its Sentimental Label.

Usability
10.00

License
CC BY-NC-SA 4.0

Expected update frequency
Never

Tags

출처: https://www.kaggle.com/datasets/cosmos98/twitter-and-reddit-sentimental-analysis-dataset

데이터 불러오기

```
# 데이터 불러오기
PATH = 'twitter/Twitter_Data.csv'
twitter = pd.read_csv(PATH)
twitter
```

✓ 0.2s

	clean_text	category
0	when modi promised “minimum government maximum...	-1.0
1	talk all the nonsense and continue all the dra...	0.0
2	what did just say vote for modi welcome bjp t...	1.0
3	asking his supporters prefix chowkidar their n...	1.0
4	answer who among these the most powerful world...	1.0
...	...	...
162975	why these 456 crores paid neerav modi not reco...	-1.0
162976	dear rss terrorist payal gawar what about modi...	-1.0
162977	did you cover her interaction forum where she ...	0.0
162978	there big project came into india modi dream p...	0.0
162979	have you ever listen about like gurukul where ...	1.0

162980 rows × 2 columns

> 162980 x 2 데이터

데이터 정보 확인 및 전처리

```
# 데이터 프레임 정보 확인
twitter.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 162980 entries, 0 to 162979
Data columns (total 2 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   clean_text  162976 non-null  object
 1   category    162973 non-null  float64
dtypes: float64(1), object(1)
memory usage: 2.5+ MB
```

```
# 결측치 확인
twitter.isna().sum()
```

```
clean_text    4
category      7
dtype: int64
```

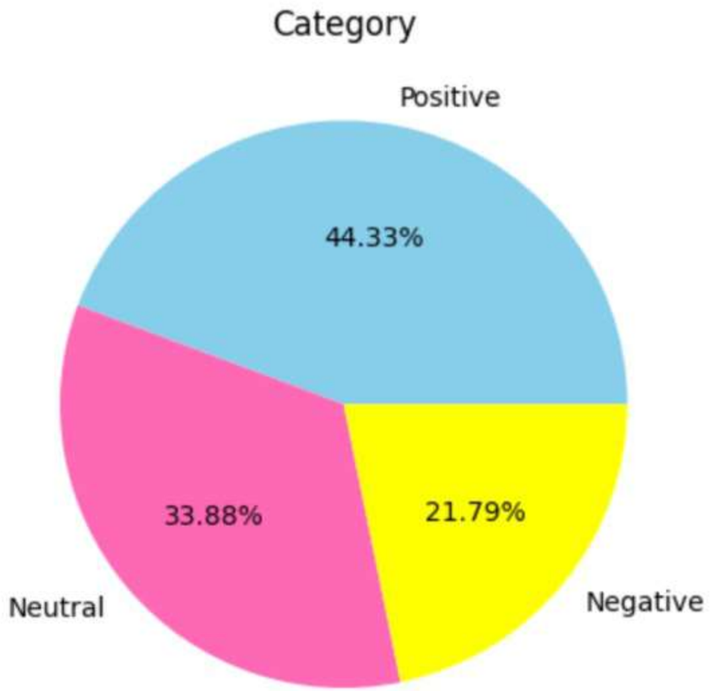
```
# 결측치 제거
twitter.dropna(inplace = True)
twitter.isna().sum()
```

```
clean_text    0
category      0
dtype: int64
```

> 두 컬럼 모두 결측치가 존재하여 제거

타겟 레이블 비율 시각화

```
# 타겟 레이블 비율 시각화
plt.pie(twitter['category'].value_counts(), labels = ['Positive', 'Neutral', 'Negative'],
        autopct = '%.2f%%', colors = ['skyblue', 'hotpink', 'yellow'])
plt.title('Category')
plt.show()
```



> 비교적 **균형 잡힌** 데이터

불용어 및 특수문자 제거

	clean_text	category
0	when modi promised “minimum government maximum...	-1.0
1	talk all the nonsense and continue all the dra...	0.0
2	what did just say vote for modi welcome bjp t...	1.0
3	asking his supporters prefix chowkidar their n...	1.0
4	answer who among these the most powerful world...	1.0

> 의미 없는 불용어와 특수문자가 존재

불용어 및 특수문자 제거

```python
# 불용어 및 특수문자 제거
stop_words = set(stopwords.words('english'))
total_text_list = []
for i in twitter['clean_text']:
    i = re.sub(r'[^A-Za-z0-9\s]', '', i)
    word_list = []
    for j in word_tokenize(i):
        if j not in stop_words:
            word_list.append(j)
    final_string = ' '.join(word_list)
    total_text_list.append(final_string)
```

✓ 9.1s

stop_words : 불용어 집합 목록

[^A-Za-Z0-9\s]: 영어, 숫자, 공백을 제외한 모든 글자

word_tokenize: 문장을 단어로 쪼개서 리스트화

```python
twitter['clean_text'] = total_text_list
twitter
```

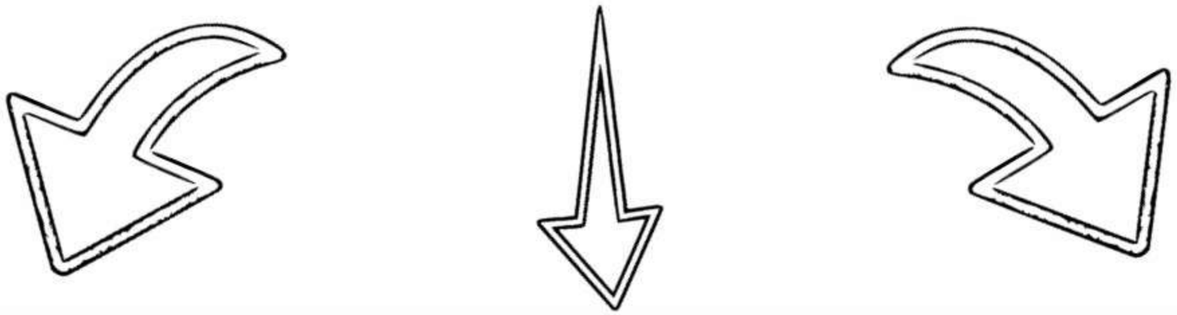
✓ 0.0s

	clean_text	category
0	modi promised minimum government maximum gover...	2.0
1	talk nonsense continue drama vote modi	0.0
2	say vote modi welcome bjp told rahul main camp...	1.0
3	asking supporters prefix chowkidar names modi ...	1.0
4	answer among powerful world leader today trump...	1.0
...	...	...
162964	456 crores paid neerav modi recovered congress...	2.0
162965	dear rss terrorist payal gawar modi killing 10...	2.0
162966	cover interaction forum left	0.0
162967	big project came india modi dream project happ...	0.0
162968	ever listen like gurukul discipline maintained...	1.0

162969 rows × 2 columns

동일한 의미의 단어 단수화

Come (기본형)



Came (과거형)　　Coming (현재분사)　　Comes (3인칭 단수)

> 같은 의미의 단어를 다른 단어로 인식

동일한 의미의 단어 단수화

동사 (Verbs):

- **VB**: 원형 동사 (e.g., run, go)
- **VBD**: 과거형 동사 (e.g., ran, went)
- **VBG**: 동명사 또는 현재분사 (e.g., running)
- **VBN**: 과거분사 (e.g., run, gone)
- **VBP**: 현재형 동사 (비3인칭 단수) (e.g., run, go)
- **VBZ**: 현재형 동사 (3인칭 단수) (e.g., runs, goes)

형용사 (Adjectives):

- **JJ**: 일반 형용사 (e.g., big, green)
- **JJR**: 비교급 형용사 (e.g., bigger, smarter)
- **JJS**: 최상급 형용사 (e.g., biggest, smartest)

부사 (Adverbs):

- **RB**: 일반 부사 (e.g., quickly, silently)
- **RBR**: 비교급 부사 (e.g., more quickly)
- **RBS**: 최상급 부사 (e.g., most quickly)

명사 (Nouns):

- **NN**: 단수 일반 명사 (e.g., dog, book)
- **NNS**: 복수 일반 명사 (e.g., dogs, books)
- **NNP**: 단수 고유 명사 (e.g., John, London)
- **NNPS**: 복수 고유 명사 (e.g., Americans)

> 같은 품사 내에서도 형태가 다양하다

동일한 의미의 단어 단수화

```python
# 단어를 단수화하는 객체
lemmatizer = WordNetLemmatizer()

# 복수형, 과거형 등 모두 단수형으로 변환
total_wordnet_list = []
for i in twitter['clean_text']:
    final_word_list = []
    pos_tag_list = pos_tag(word_tokenize(i))
    for word, pos in pos_tag_list:
        if pos[0] == 'N': final_word_list.append(lemmatizer.lemmatize(word, 'n'))
        elif pos[0] == 'J': final_word_list.append(lemmatizer.lemmatize(word, 'a'))
        elif pos[0] == 'R': final_word_list.append(lemmatizer.lemmatize(word, 'r'))
        elif pos[0] == 'V': final_word_list.append(lemmatizer.lemmatize(word, 'v'))
    total_wordnet_list.append(' '.join(final_word_list))
```

✓ 1m 5.1s

lematizer : 단어를 단수화하는 인스턴스

pos_tag: 단어의 품사를 파악해서 (단어, 품사) 튜플로 저장

```python
twitter['clean_text'] = total_wordnet_list
twitter
```

✓ 0.0s

	clean_text	category
0	modi promise minimum government maximum govern...	2.0
1	talk nonsense continue drama vote modi	0.0
2	say vote welcome bjp tell rahul main campaigne...	1.0
3	ask supporter prefix chowkidar name modi great...	1.0
4	answer powerful world leader today trump putin...	1.0
...	...	...
162964	crore pay neerav modi recover congress leader ...	2.0
162965	dear rss terrorist payal gawar modi kill musli...	2.0
162966	cover interaction forum leave	0.0
162967	big project come india modi dream project happ...	0.0
162968	ever listen gurukul discipline maintain even n...	1.0

162969 rows × 2 columns

방법1: 문장별 품사의 빈도수 데이터 프레임

```
# 각 문장별 명사, 형용사, 부사, 동사의 빈도수 세기
total_pos_dict_list = []
for i in twitter['clean_text']:
    pos_dict = {'NN' : 0, 'JJ' : 0, 'RB' : 0, 'VB' : 0}
    pos_tag_list = pos_tag(word_tokenize(i))
    for word, pos in pos_tag_list:
        if pos in pos_dict.keys():
            pos_dict[pos] += 1
    total_pos_dict_list.append(pos_dict)
```

✓ 58.4s

NN : 명사, JJ: 형용사, RB: 부사, VB: 동사

```
posDF = pd.DataFrame(total_pos_dict_list)
posDF
```

✓ 0.0s

	NN	JJ	RB	VB
0	14	3	0	1
1	5	0	0	0
2	8	2	0	1
3	9	6	1	0
4	5	1	1	0
...	...	...	...	...
162964	10	1	0	1
162965	12	13	2	2
162966	2	0	0	0
162967	5	2	0	0
162968	11	3	2	0

162969 rows × 4 columns

훈련, 검증, 테스트 데이터 분리

```
DF = posDF

# 훈련, 검증, 테스트 데이터 분리
featureDF = DF.iloc[:,:-1]
targetDF = DF[['category']]

train_inputDF, test_inputDF, train_targetDF, test_targetDF = train_test_split(featureDF, targetDF,
                                                             stratify = targetDF,
                                                             train_size = 0.8, random_state = 42)
train_inputDF, valid_inputDF, train_targetDF, valid_targetDF = train_test_split(train_inputDF, train_targetDF,
                                                               stratify = train_targetDF,
                                                               train_size = 0.8, random_state = 42)

print(f"[train] input : {train_inputDF.shape}, target : {train_targetDF.shape}")
print(f"[valid] input : {valid_inputDF.shape}, target : {valid_targetDF.shape}")
print(f"[test] input : {test_inputDF.shape}, target : {test_targetDF.shape}")
```

```
[train] input : (104300, 4), target : (104300, 1)
[valid] input : (26075, 4), target : (26075, 1)
[test] input : (32594, 4), target : (32594, 1)
```

> 훈련: 104300개, 검증: 26075개, 테스트: 32594개

데이터셋 및 데이터로더 생성

```
DIM = 1

trainDS = CustomDataset(train_inputDF, train_targetDF, feature_dim=DIM)
validDS = CustomDataset(valid_inputDF, valid_targetDF, feature_dim=DIM)
testDS = CustomDataset(test_inputDF, test_targetDF, feature_dim=DIM)

# 데이터셋 속성
print(f"trainDS shape : ({trainDS.n_rows}, {trainDS.n_features})")
print(f"validDS shape : ({validDS.n_rows}, {validDS.n_features})")
print(f"testDS shape : ({testDS.n_rows}, {testDS.n_features})")

trainDL = DataLoader(trainDS, batch_size=64)
validDL = DataLoader(validDS, batch_size=64)
testDL = DataLoader(testDS, batch_size=64)
```

<- 배치 사이즈 : 64개

```
trainDS shape : (104300, 4)
validDS shape : (26075, 4)
testDS shape : (32594, 4)
```

모델 인스턴스 생성

```
# 모델 인스턴스 생성
model = LinearModel(input_in=4, output_out=3,
                    hidden_list=[100, 80, 60, 40, 20], act_func=F.relu, model_type='multiclass')
print(model)
summary(model)
```

```
LinearModel(
  (input_layer): Linear(in_features=4, out_features=100, bias=True)
  (hidden_layer_list): ModuleList(
    (0): Linear(in_features=100, out_features=80, bias=True)
    (1): Linear(in_features=80, out_features=60, bias=True)
    (2): Linear(in_features=60, out_features=40, bias=True)
    (3): Linear(in_features=40, out_features=20, bias=True)
  )
  (output_layer): Linear(in_features=20, out_features=3, bias=True)
)
```

<- 은닉층 : 4개

방법1: 성능 확인

```python
# 옵티마이저 (모델의 가중치 절편 최적화)
adam_optim = optim.Adam(model.parameters(), lr=0.01)
# 가중치, 절편을 전달하고 학습률을 설정

# 모델 훈련
MODEL = model
EPOCH = 100
SAVE_PATH = 'model'
result = training(trainDL, validDL, MODEL, 'multiclass',
                  adam_optim, EPOCH, endurance_cnt=10,
                  num_classes=3, SAVE_PATH=SAVE_PATH)
```

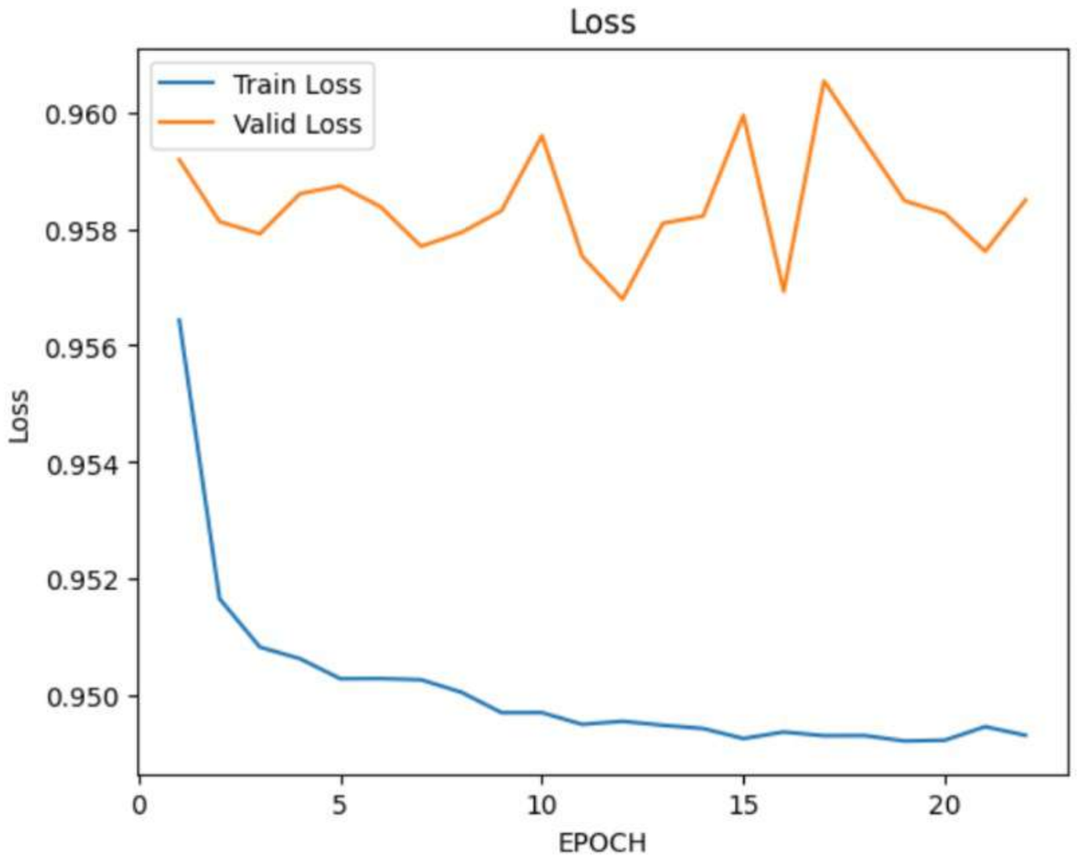
✓ 2m 21.2s

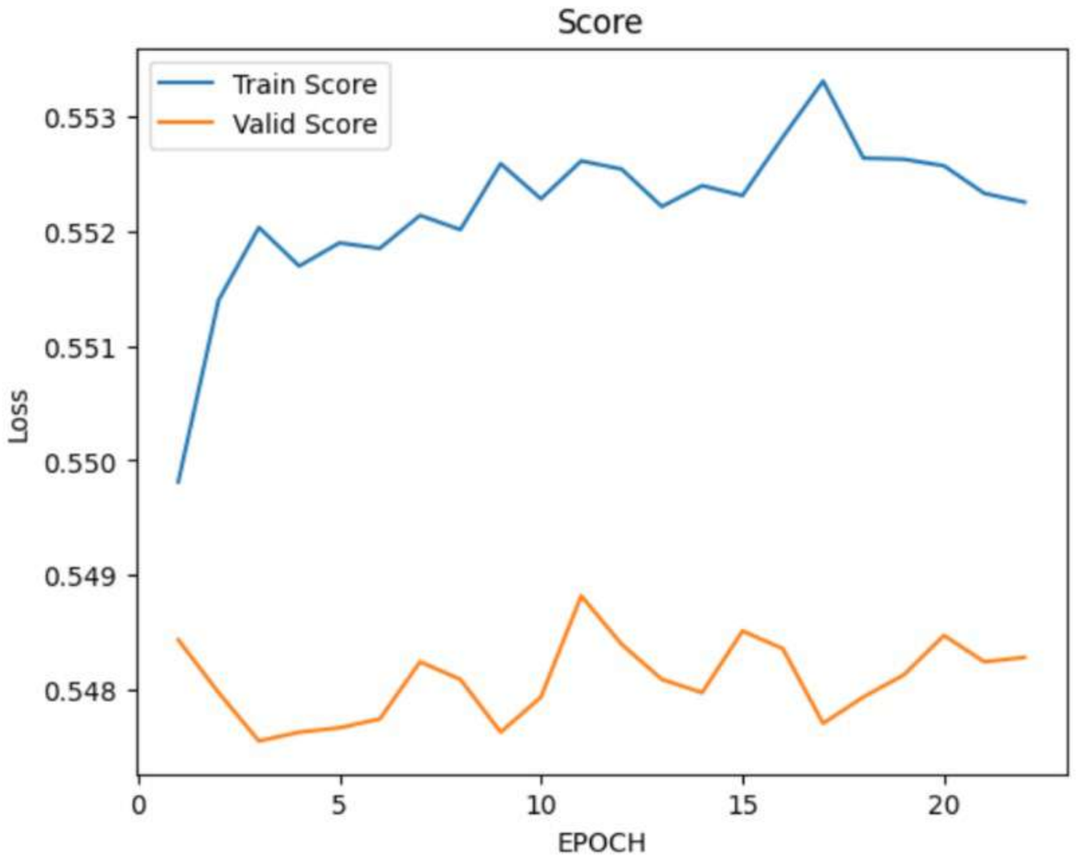
```
[EPOCH] : 1에서 모델 저장 완료.
[Loss : 1/100] Train : 0.9564, Test : 0.9592
[Score  : 1/100] Train : 0.5498, Test : 0.5484
[EPOCH] : 2에서 모델 저장 완료.
[Loss : 2/100] Train : 0.9517, Test : 0.9581
[Score  : 2/100] Train : 0.5514, Test : 0.5480
[EPOCH] : 3에서 모델 저장 완료.
[Loss : 3/100] Train : 0.9508, Test : 0.9579
[Score  : 3/100] Train : 0.5520, Test : 0.5476
[Loss : 4/100] Train : 0.9506, Test : 0.9586
[Score  : 4/100] Train : 0.5517, Test : 0.5476
[Loss : 5/100] Train : 0.9503, Test : 0.9587
[Score  : 5/100] Train : 0.5519, Test : 0.5477
[Loss : 6/100] Train : 0.9503, Test : 0.9584
[Score  : 6/100] Train : 0.5518, Test : 0.5477
[EPOCH] : 7에서 모델 저장 완료.
[Loss : 7/100] Train : 0.9503, Test : 0.9577
[Score  : 7/100] Train : 0.5521, Test : 0.5482
[Loss : 8/100] Train : 0.9501, Test : 0.9579
[Score  : 8/100] Train : 0.5520, Test : 0.5481
[Loss : 9/100] Train : 0.9497, Test : 0.9583
[Score  : 9/100] Train : 0.5526, Test : 0.5476
[Loss : 10/100] Train : 0.9497, Test : 0.9596
[Score  : 10/100] Train : 0.5523, Test : 0.5479
[EPOCH] : 11에서 모델 저장 완료.
...
[Score  : 20/100] Train : 0.5526, Test : 0.5485
[Loss : 21/100] Train : 0.9495, Test : 0.9576
[Score  : 21/100] Train : 0.5523, Test : 0.5482
[Loss]값의 개선이 이루어지지 않아 [22] EPOCH에서 학습을 종료합니다.
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

검증 데이터셋 성능: 54.8% ->

방법1: 손실값, 스코어 시각화





> 모델의 성능이 개선되지 않음

방법2: 문장별 품사의 비율 데이터 프레임

```
posDF2 = posDF.iloc[:,:-1].div(posDF.iloc[:,:-1].sum(axis=1), axis=0)
posDF2['category'] = posDF[['category']]
posDF2.dropna(inplace=True)
posDF2.reset_index(drop=True)
posDF2
```

✓ 0.0s

품사별 비율로 값을 변경 ->

	NN	JJ	RB	VB	category
0	0.777778	0.166667	0.000000	0.055556	2.0
1	1.000000	0.000000	0.000000	0.000000	0.0
2	0.727273	0.181818	0.000000	0.090909	1.0
3	0.562500	0.375000	0.062500	0.000000	1.0
4	0.714286	0.142857	0.142857	0.000000	1.0
...	...	...	...	...	...
162964	0.833333	0.083333	0.000000	0.083333	2.0
162965	0.413793	0.448276	0.068966	0.068966	2.0
162966	1.000000	0.000000	0.000000	0.000000	0.0
162967	0.714286	0.285714	0.000000	0.000000	0.0
162968	0.687500	0.187500	0.125000	0.000000	1.0

162568 rows × 5 columns

방법2: 성능 확인

```python
# 옵티마이저 (모델의 가중치 절편 최적화)
adam_optim = optim.Adam(model.parameters(), lr=0.01)
# 가중치, 절편을 전달하고 학습률을 설정

# 모델 훈련
MODEL = model
EPOCH = 100
SAVE_PATH = 'model'
result = training(trainDL, validDL, MODEL, 'multiclass',
                  adam_optim, EPOCH, endurance_cnt=10,
                  num_classes=3, SAVE_PATH=SAVE_PATH)
```

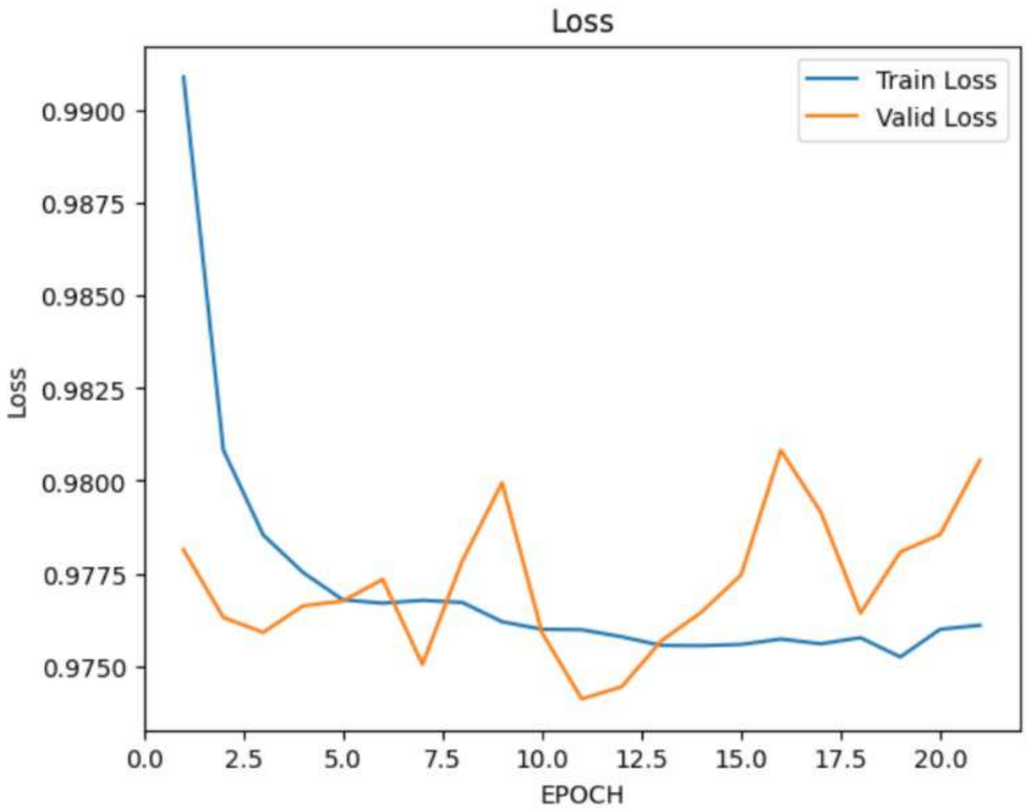
✓ 2m 21.2s

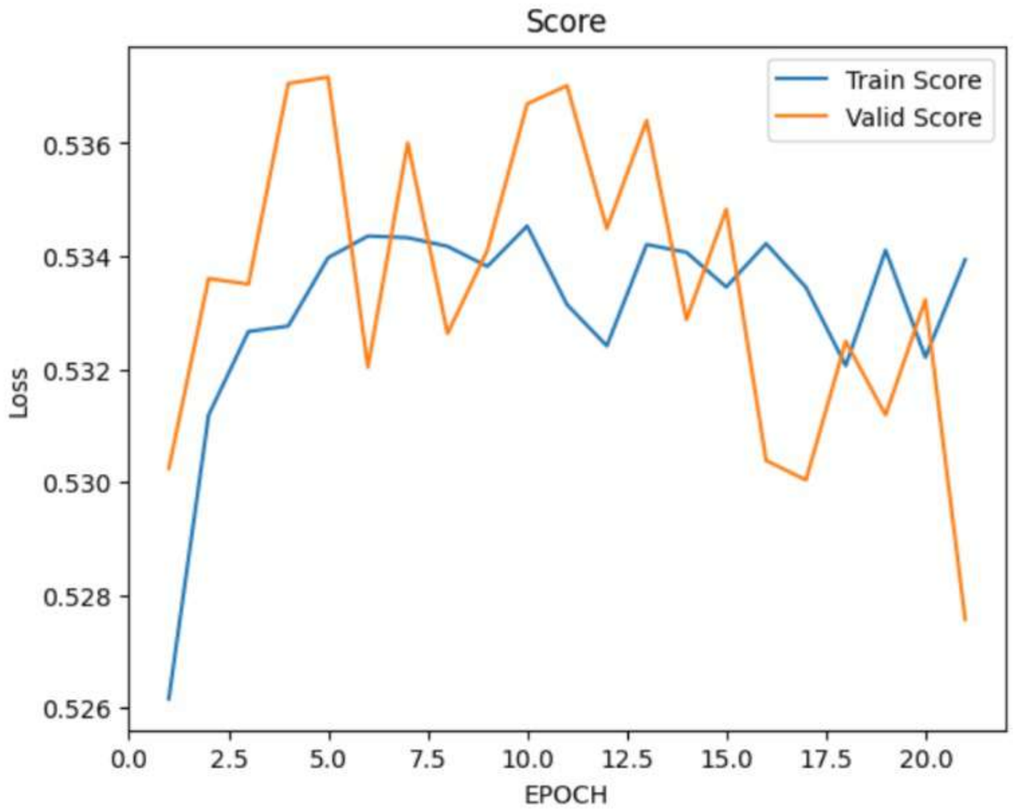
```
[EPOCH] : 1에서 모델 저장 완료.
[Loss : 1/100] Train : 0.9909, Test : 0.9781
[Score  : 1/100] Train : 0.5262, Test : 0.5302
[EPOCH] : 2에서 모델 저장 완료.
[Loss : 2/100] Train : 0.9808, Test : 0.9763
[Score  : 2/100] Train : 0.5312, Test : 0.5336
[EPOCH] : 3에서 모델 저장 완료.
[Loss : 3/100] Train : 0.9785, Test : 0.9759
[Score  : 3/100] Train : 0.5327, Test : 0.5335
[Loss : 4/100] Train : 0.9775, Test : 0.9766
[Score  : 4/100] Train : 0.5328, Test : 0.5371
[Loss : 5/100] Train : 0.9768, Test : 0.9768
[Score  : 5/100] Train : 0.5340, Test : 0.5372
[Loss : 6/100] Train : 0.9767, Test : 0.9773
[Score  : 6/100] Train : 0.5344, Test : 0.5320
[EPOCH] : 7에서 모델 저장 완료.
[Loss : 7/100] Train : 0.9768, Test : 0.9750
[Score  : 7/100] Train : 0.5343, Test : 0.5360
[Loss : 8/100] Train : 0.9767, Test : 0.9778
[Score  : 8/100] Train : 0.5342, Test : 0.5326
[Loss : 9/100] Train : 0.9762, Test : 0.9799
[Score  : 9/100] Train : 0.5338, Test : 0.5341
[Loss : 10/100] Train : 0.9760, Test : 0.9759
[Score  : 10/100] Train : 0.5345, Test : 0.5367
[EPOCH] : 11에서 모델 저장 완료.
...
[Score  : 19/100] Train : 0.5341, Test : 0.5312
[Loss : 20/100] Train : 0.9760, Test : 0.9785
[Score  : 20/100] Train : 0.5322, Test : 0.5332
[Loss]값의 개선이 이루어지지 않아 [21] EPOCH에서 학습을 종료합니다.
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

검증 데이터셋 성능: 53.3% ->

방법2: 손실값, 스코어 시각화





> 접근 방식에 근본적인 문제가 있다고 판단

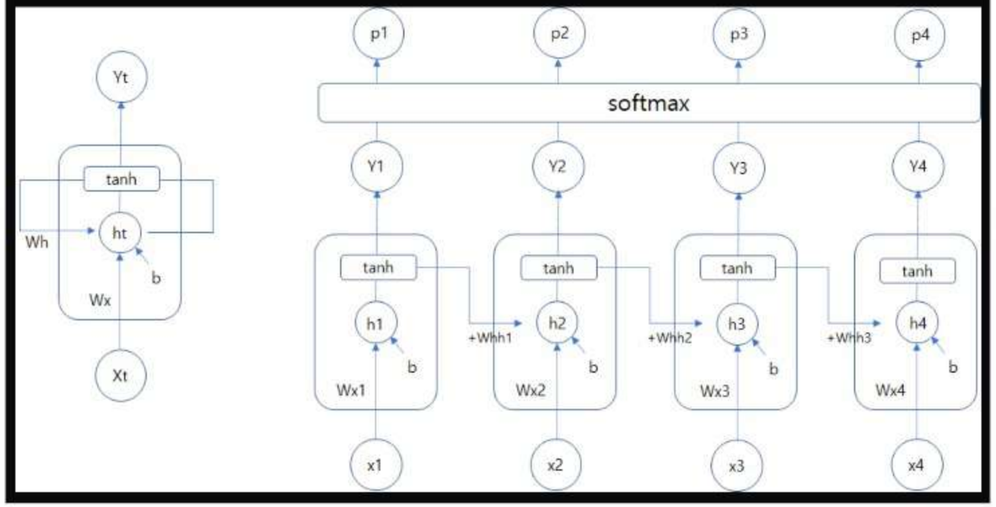
자연어 분석 모델 종류

RNN(Recurrent Neural Network)-순환신경망

RNN은 순환 신경망구조로써 기존까지 알고 있던 단순 퍼셉트론과는 약간 다른 개념을 지니고 있다.

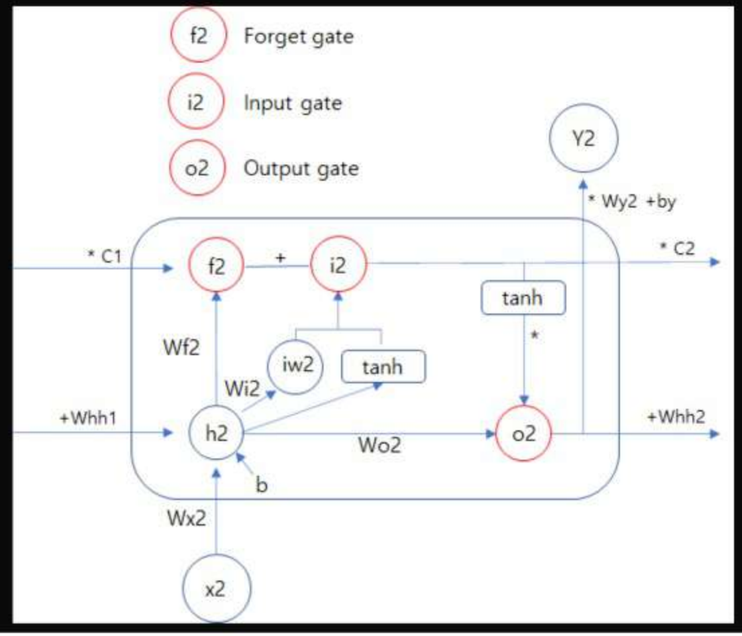
가장 큰 특징으로는 시퀀스 개념을 가지고 있어, 이전 정보를 토대로 다음 정보를 예측하는 방식이다.

RNN의 구조와 예측 방식



LSTM(Long Short Term Memory) 장단기 기억망:

RNN을 베이스로 **기울기 소실/폭발 관련 단점을 해결**한 순환 신경망의 응용으로 현재도 많이 사용하는 모델이라고 한다.



> LSTM을 분석 모델로 채택

자연어 분석 방법 종류

1. Bag of Words란?

Bag of Words란 단어들의 순서는 전혀 고려하지 않고, 단어들의 출현 빈도(frequency)에만 집중하는 텍스트 데이터의 수치화 표현 방법입니다. Bag of Words를 직역하면 단어들의 가방이라는 의미입니다. 단어들이 들어있는 가방을 상상해봅시다. 갖고있는 어떤 텍스트 문서에 있는 단어들을 가방에다가 전부 넣습니다. 그 후에는 이 가방을 흔들어 단어들을 섞습니다. 만약, 해당 문서 내에서 특정 단어가 N번 등장했다면, 이 가방에는 그 특정 단어가 N개 있게됩니다. 또한 가방을 흔들어서 단어를 섞었기 때문에 더 이상 단어의 순서는 중요하지 않습니다.

09-02 워드투벡터(Word2Vec)

앞서 원-핫 벡터는 단어 벡터 간 유의미한 유사도를 계산할 수 없다는 단점이 있음을 언급한 적이 있습니다. 그래서 단어 벡터 간 유의미한 유사도를 반영할 수 있도록 단어의 의미를 수치화 할 수 있는 방법이 필요합니다. 이를 위해서 사용되는 대표적인 방법이 워드투벡터(Word2Vec)입니다. Word2Vec의 개념을 설명하기 앞서 Word2Vec가 어떤 일을 할 수 있는지 확인해보겠습니다.

1. TF-IDF(단어 빈도-역 문서 빈도, Term Frequency-Inverse Document Frequency)

TF-IDF(Term Frequency-Inverse Document Frequency)는 단어의 빈도와 역 문서 빈도(문서의 빈도에 특정 식을 취함)를 사용하여 DTM 내의 각 단어들마다 중요한 정도를 가중치로 주는 방법입니다. 우선 DTM을 만든 후, TF-IDF 가중치를 부여합니다.

TF-IDF는 주로 문서의 유사도를 구하는 작업, 검색 시스템에서 검색 결과의 중요도를 정하는 작업, 문서 내에서 특정 단어의 중요도를 구하는 작업 등에 쓰일 수 있습니다.

TF-IDF는 TF와 IDF를 곱한 값을 의미하는데 이를 식으로 표현해보겠습니다. 문서를 d, 단어를 t, 문서의 총 개수를 n이라고 표현할 때 TF, DF, IDF는 각각 다음과 같이 정의할 수 있습니다.

> TF-IDF를 분석 방법으로 채택

모델 클래스 정의하기

K-Digital Training > MyModule > KDTModule.py > ...

```python
class LSTMModel(nn.Module):
    def __init__(self, input_size, output_size, hidden_list, act_func, model_type, num_layers=1):
        super().__init__()

        # LSTM 레이어 (입력 크기, 은닉 크기, 레이어 수, batch_first를 True로 설정하여 배치가 첫 번째 차원이 되게 함)
        self.lstm = nn.LSTM(input_size, hidden_list[0], num_layers=num_layers, batch_first=True)

        # 은닉층
        self.hidden_layer_list = nn.ModuleList()
        for i in range(len(hidden_list)-1):
            self.hidden_layer_list.append(nn.Linear(hidden_list[i], hidden_list[i+1]))

        # 출력층
        self.output_layer = nn.Linear(hidden_list[-1], output_size)

        self.act_func = act_func
        self.model_type = model_type

    def forward(self, x):
        # LSTM 레이어를 통과 (x: 배치 크기, 시퀀스 길이, 입력 크기)
        lstm_out, (hn, cn) = self.lstm(x)  # lstm_out은 모든 타임스텝의 출력을 포함, hn은 마지막 타임스텝의 출력

        # LSTM의 마지막 타임스텝 출력만 사용
        x = lstm_out[:, -1, :]  # 마지막 타임스텝의 출력을 사용

        # 은닉층
        for layer in self.hidden_layer_list:
            x = layer(x)
            x = self.act_func(x)

        # 출력층
        if self.model_type == 'regression':  # 회귀
            return self.output_layer(x)
        elif self.model_type == 'binary':  # 이진 분류
            return torch.sigmoid(self.output_layer(x))
        elif self.model_type == 'multiclass':  # 다중 분류
            return self.output_layer(x)  # CrossEntropyLoss에서 log-softmax 처리
```

> LSTM 모델 클래스 생성

방법3: TF-IDF 벡터화

```python
vectorizer = TfidfVectorizer(max_features = 8000) # 최대 8000개의 특징 선택
train_vectors = vectorizer.fit_transform(train_inputDF['clean_text']).toarray()
valid_vectors = vectorizer.transform(valid_inputDF['clean_text']).toarray()
test_vectors = vectorizer.transform(test_inputDF['clean_text']).toarray()
```

✓ 2.6s Python

```python
train_vectorsDF = pd.DataFrame(train_vectors)
valid_vectorsDF = pd.DataFrame(valid_vectors)
test_vectorsDF = pd.DataFrame(test_vectors)
train_vectorsDF
```

✓ 0.5s Python

	0	1	2	3	4	5	6	7	8	9	...	7990	7991	7992	7993	7994	7995	7996	7997	7998	7999
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
1	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
2	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
3	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
4	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
104295	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
104296	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
104297	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
104298	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
104299	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

104300 rows × 8000 columns

> TF-IDF 데이터 프레임 생성

성능 확인

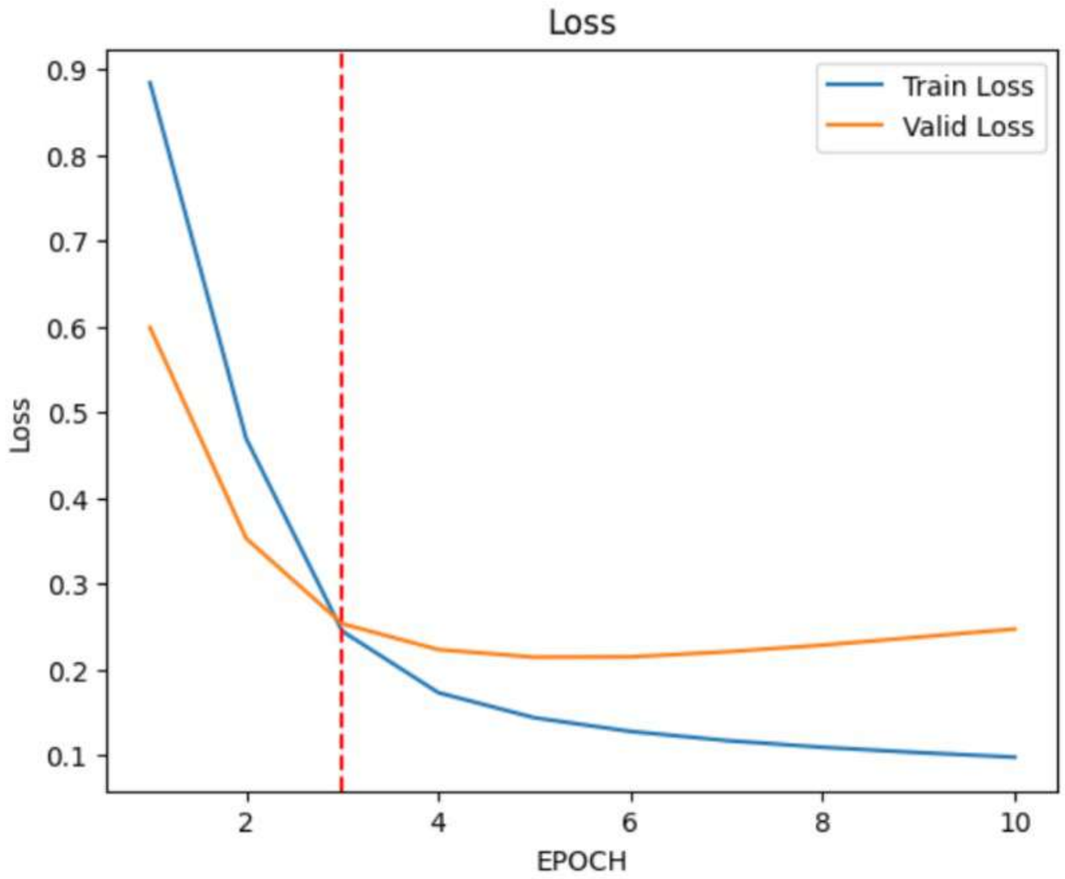
```python
# 모델 훈련
EPOCH = 100
SAVE_PATH = '/Users/anhyojun/VSCode/K-Digital Training/김소현 강사님/프로젝트/4번째 프로젝트/model'
result = training(trainDL, validDL, model, 'multiclass', adam_optim, EPOCH,
                  endurance_cnt=5, num_classes=3, SAVE_PATH=SAVE_PATH)
```

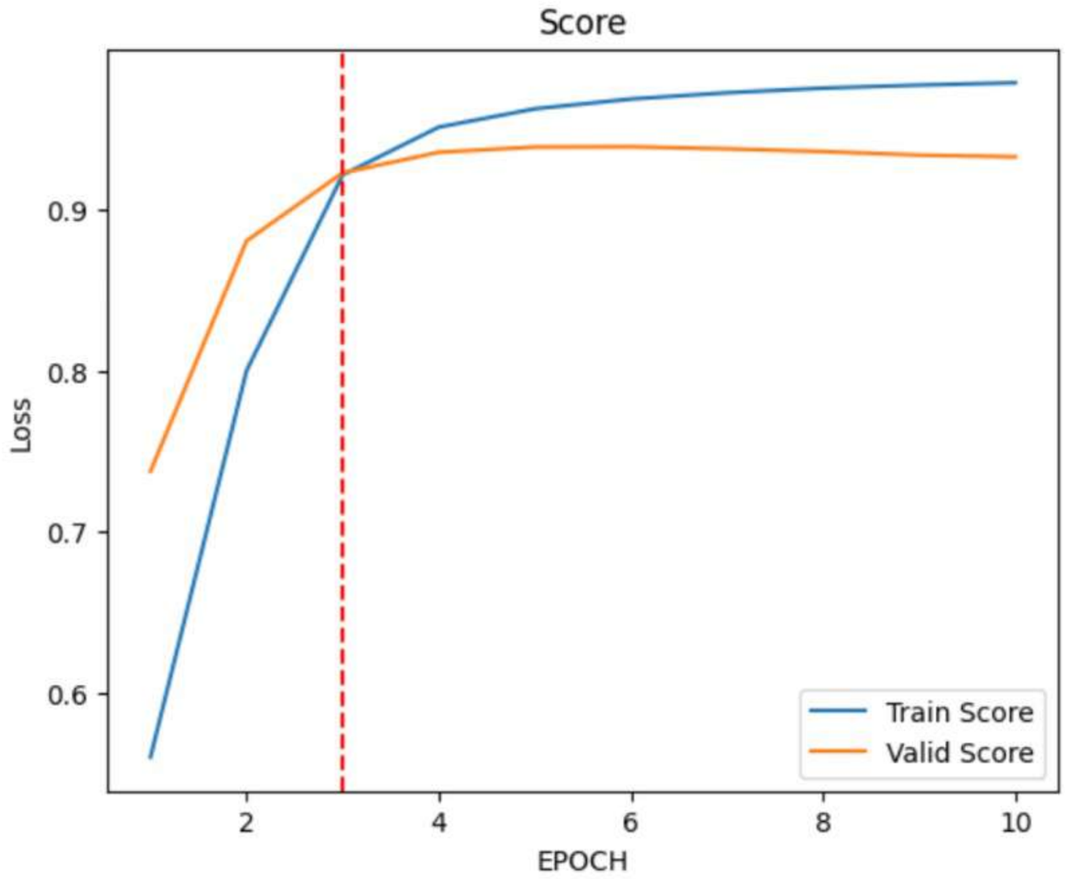
✓ 4m 10.0s Python

```
[EPOCH] : 1에서 모델 저장 완료.
[Loss : 1/100] Train : 0.8843, Test : 0.5985
[Score  : 1/100] Train : 0.5607, Test : 0.7376
[EPOCH] : 2에서 모델 저장 완료.
[Loss : 2/100] Train : 0.4692, Test : 0.3526
[Score  : 2/100] Train : 0.7998, Test : 0.8802
[EPOCH] : 3에서 모델 저장 완료.
[Loss : 3/100] Train : 0.2443, Test : 0.2529
[Score  : 3/100] Train : 0.9210, Test : 0.9223
[EPOCH] : 4에서 모델 저장 완료.
[Loss : 4/100] Train : 0.1725, Test : 0.2228
[Score  : 4/100] Train : 0.9506, Test : 0.9350
[EPOCH] : 5에서 모델 저장 완료.
[Loss : 5/100] Train : 0.1432, Test : 0.2140
[Score  : 5/100] Train : 0.9619, Test : 0.9383
[Loss : 6/100] Train : 0.1273, Test : 0.2143
[Score  : 6/100] Train : 0.9680, Test : 0.9384
[Loss : 7/100] Train : 0.1166, Test : 0.2202
[Score  : 7/100] Train : 0.9719, Test : 0.9372
[Loss : 8/100] Train : 0.1087, Test : 0.2280
[Score  : 8/100] Train : 0.9747, Test : 0.9355
[Loss : 9/100] Train : 0.1025, Test : 0.2373
[Score  : 9/100] Train : 0.9766, Test : 0.9333
[Loss]값의 개선이 이루어지지 않아 [10] EPOCH에서 학습을 종료합니다.
```

> 검증 데이터셋 성능: 93.3%

손실값, 스코어 시각화





> EPOCH = 3에서 최고 성능으로 판단

Best Model 저장 및 예측

```python
best_model = LSTMModel(input_size = 8000, output_size = 3, hidden_list = [100, 80, 60, 40, 20],
                       act_func=F.relu, model_type='multiclass', num_layers=1)
# num_layers : 재귀층(Recurrent Layer)의 개수 (기본적인 LSTM 모델에서는 1개)

best_model.load_state_dict(torch.load('Best_LSTM_Model.pth', weights_only=True))
```

✓ 0.0s Python

```
<All keys matched successfully>
```

```python
def predict_value(test_inputDF, model, dim):
    if dim == 2:
        test_inputTS = torch.FloatTensor(test_inputDF.values)
        return torch.argmax(model(test_inputTS), dim=1)
    elif dim == 3:
        test_inputTS = torch.FloatTensor(test_inputDF.values).reshape(1,1,-1)
        return torch.argmax(model(test_inputTS), dim=1)
```

✓ 0.0s Python

```python
test_vectorsTS = torch.FloatTensor(test_vectorsDF.values)
test_vectorsTS3D = test_vectorsTS.reshape(-1,1,8000)

predictTS = torch.argmax(best_model(test_vectorsTS3D), dim=1)
test_targetTS = torch.FloatTensor(test_targetDF.values)

answer_count = 0
for i in range(len(predictTS)):
    if predictTS[i] == test_targetTS.reshape(-1)[i]:
        answer_count += 1

print(f"[Test Accuracy] : {answer_count / len(predictTS)}")
```

✓ 1.3s Python

```
[Test Accuracy] : 0.923145364177456
```

> 테스트 정확도 : 92.3%

테스트 데이터 평가 지표

```
y_test = test_targetTS.reshape(-1).tolist()
y_pred = predictTS.tolist()

print(f"{'-'*18}테스트 세트 평가 지표{'-'*18}\n{classification_report(y_test, y_pred)}")
```

✓ 0.0s

```
------------------테스트 세트 평가 지표------------------
              precision    recall  f1-score   support

         0.0       0.94      0.94      0.94     11042
         1.0       0.93      0.96      0.94     14450
         2.0       0.89      0.82      0.85      7102

    accuracy                           0.92     32594
   macro avg       0.92      0.91      0.91     32594
weighted avg       0.92      0.92      0.92     32594
```

> 비교적 만족스러운 결과

AUC 값 확인

```
prob = F.softmax(best_model(test_vectorsTS3D), dim=1)
prob
```

✓ 0.7s

```
tensor([[9.4770e-01, 5.0048e-03, 4.7290e-02],
        [9.4138e-01, 1.8142e-02, 4.0473e-02],
        [1.5091e-04, 9.6940e-01, 3.0445e-02],
        ...,
        [5.5529e-04, 9.5091e-01, 4.8535e-02],
        [4.2552e-04, 9.5515e-01, 4.4422e-02],
        [6.8317e-02, 8.6519e-01, 6.6490e-02]], grad_fn=<SoftmaxBackward0>)
```

```
y_test_int = list(map(int, y_test))
y_test_onehot = np.eye(3)[y_test_int]
y_test_onehot
```

✓ 0.0s

```
array([[1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       ...,
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.]])
```

```
roc_auc_score(y_test_onehot, prob.detach().numpy(), multi_class='ovr')
```

✓ 0.0s

```
0.9781819498267373
```

> AUC : 0.978

최종 결론 및 소감

- 자연어 분석이 이렇게 어려운 줄 몰랐다
- 하지만 처음 해본 자연어 분석에서 만족스러운 결과를 도출하였다
- 선행학습을 통해 이후에 배울 자연어 분석의 기초를 맛볼 수 있었다