고급통계적딥러닝 Final Project

Spooky Author Natural Language Preprocessing

고려대학교 통계학과
2020021202 이혜원

1 Data Description

프로젝트에서 사용할 데이터는 **Spooky Author Identification** 데이터이다. 이 데이터는 세 명의 작가가 쓴 공포 소설의 문장들로 이루어져 있다. 본 프로젝트의 목적은 자연어처리 알고리즘을 적용해 소설 속 텍스트를 분석하여 해당 텍스트를 쓴 작가를 예측하는 것이다.

Spooky 데이터의 구성은 Table 1과 같다. 이 데이터는 19579개의 데이터와 세 변수로 구성되어있는데, 분석에 사용할 변수는 text, author 변수이다.

text 변수는 작가가 쓴 문장을 나타내는 변수이고, 목적변수인 author은 본 프로젝트에서 예측해야 할 세 명의 작가이다. 세 작가의 이름은 EAP(Edgar Allan Poe), HPL(HP Lovecraft), MWS(Mary Wollstonecraft Shelley) 이다.

Table 1: Spooky Author Identification Data

Variable	Description	Type
id	a unique identifier for each sentence	object
text	text written by authors	sentence
target	the author of sentence	class(EAP, MWS, HPL)

2 Exploratory Data Analysis

탐색적 데이터분석(EDA)를 진행하여 Spooky 데이터의 특징을 파악해보자.

2.1 Spooky Author distribution

Spooky 데이터의 author은 세 명으로 약자로 EAP, MWS, HPL 이다. Spooky 데이터의 각 author에 해당하는 데이터가 몇 개 있는지를 나타낸 그래프는 아래과 같다. EAP가 쓴 텍스트 데이터가 가장 많으며 HPL에 해당하는 텍스트 데이터가 가장 적다.

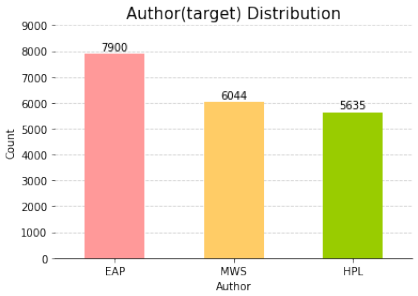


Figure 1: Spooky Authors

2.2 Frequency of words

각 author가 어떤 단어들을 가장 많이 썼는지, 문장에 사용한 단어들의 종류와 분포는 어떻게 되는지에 대해 분석해보자. 분석을 위해 불필요한 단어들은 제거했는데, 자세한 과정은 **??** 에서 자세히 다룬다. 불필요한 단어는 불용어(Stopwords)라 하는데, stopwords를 제거한 후 각 author이 가장 많이 사용한 20개의 단어의 빈도를 표현한 그래프는 Figure 2에 있다.

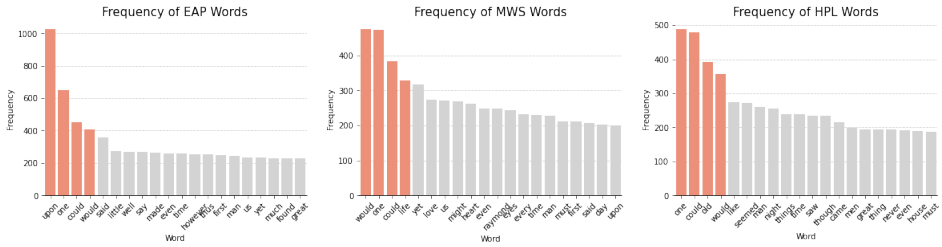


Figure 2: Frequency of words of authors

세 명의 author가 가장 많이 사용한 단어들은 **one, would, could** 이다. 이 단어들은 stopwords 목록에는 포함되어 있지 않지만, 문법적인 의미만 있을 뿐 툭별한 의미를 가지지 않는다. 따라서 이 단어들은 NLP 모델 성능을 저하시킬 수 있기 때문에 제거해야 한다.

2.3 Word Cloud

정제된 데이터를 이용해 각 author가 많이 쓰는 단어를 Word Cloud로 시각화해보자. 특정 단어의 크기가 클수록 author의 단어 사용 빈도가 높다는 뜻이다.

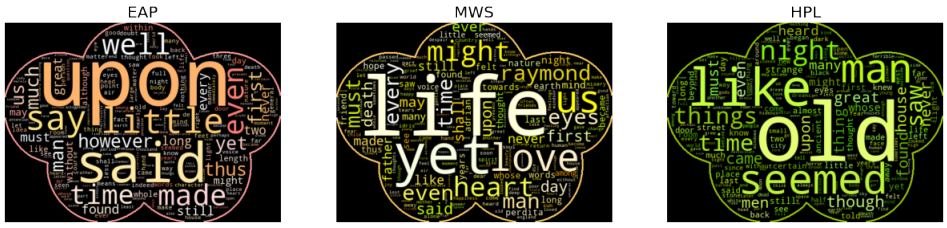


Figure 3: Word cloud for author's texts

3 Text preprocessing

Spooky 데이터에 있는 text 변수들을 그대로 NLP 모델에 사용하는 것은 불가능하다. 따라서 모델의 입력값으로 사용할 수 있도록 text 변수를 적절히 가공해야 한다. 텍스트 가공은 아래의 기법들을 이용해 진행한다. 텍스트 전처리 python code는 **??** 섹션을 참고하면 된다.

1. 텍스트 클렌징(Text Cleansing)

2. Stopwords 제거
3. 텍스트 정규화(Lemmatization)
4. 토큰화 및 Input 데이터 생성

3.1 Text Cleansing

텍스트 가공 전 불필요한 문자나 기호를 제거하는 텍스트 클렌징(Cleansing)을 한다. Spooky 데이터 클렌징을 위해 특수문자들을 제거한다. 특수문자는 문장 내 등장 빈도가 많지만 특별한 의미가 없는 문자이기 때문에 모델 성능 저하의 원인이 될 수 있다.
문장을 구성하는 단어들을 모두 소문자로 변환한다. horrible 이라는 단어를 예시로 들어보자. 만약 이 단어가 문장의 맨 앞에 나오면 Horrible, 문장의 중간에 나오면 horrible로 표기한다. 이후 단어 토큰화 시 Horrible과 horrible을 다른 단어로 인식하여 토큰화된 단어의 수가 늘어나는 현상이 발생할 수 있다. 이를 방지하기 위해 단어들을 모두 소문자로 변환한다.

3.2 Remove Stopwords

NLP 모델 성능 저하의 원인이 되는 stopwords를 제거한다. Stopwords는 텍스트 분석에 큰 의미가 없는 단어들을 지칭한다. 추가로 EDA 파트에서 찾은 one, would, could 와 같이 불필요한 단어들도 stopwords로 간주하여 Spooky 데이터에서 제거한다.

3.3 Lemmatization

표제어추출(Lemmatization)은 대표적인 텍스트 정규화 기법이다. 텍스트 정규화는 표현 방법이 다른 단어들을 통합하여 같은 단어로 만드는 과정이다. 표제어추출의 역할은 변형된 단어의 기본형을 찾는 것이다. 만약 텍스트 정규화를 적용하지 않으면 단어토큰화 시 running, ran, run을 모두 다른 단어로 인식한다. 같은 의미를 내포한 단어들을 각기 다른 단어로 가정하여 NLP 모델에 적용하면 토큰화된 단어의 수가 증가하기 때문에 모델 복잡도가 증가할 수 있다. 이러한 이유로 표제어추출을 적용하여 같은 단어에서 파생된 단어들을 하나의 단어로 통합한다.
표제어추출을 위해서는 표제어추출함수 WordNetLemmatizer 에 단어와 단어의 품사를 입력하면 된다. Python nltk 패키지에서는 pos_tag 라는 품사 추출 함수를 제공한다. pos_tag 는 단어를 입력하면 명사, 동사, 형용사, 부사 등등 해당 단어의 품사를 문자화하여 출력한다. 품사의 종류가 매우 많기 때문에 본 프로젝트에서는 주요 품사인 명사, 동사, 형용사, 부사에 해당하는 단어만 표제어추출을 하고 나머지 품사에 해당하는 단어는 그대로 둔다.

3.4 Tokenize and create input data

1. Predictor
이제 NLP 모델의 input 시퀀스를 만들기 위한 작업을 한다. Tokenizer을 이용해 Spooky 데이터의 문장을 단어토큰화(Wrod Tokenization)을 한 후 토큰화한 개별 단어에 고유한 인덱스를 부여한다. Spooky 데이터의 text 변수에 특정 단어가 포함되어있으면 단어의 인덱스를 입력하고, 그렇지 않으면 0으로 padding 한다. 이 때 최대 시퀀스의 길이는 100으로 설정한. 만약 문장을 구성하는 단어가 100개 보다 적으면 앞에서부터 0으로 padding하고, 100개보다 많으면 앞에서부터 단어를 제거하여 시퀀스 길이를 100으로 맞춘다.

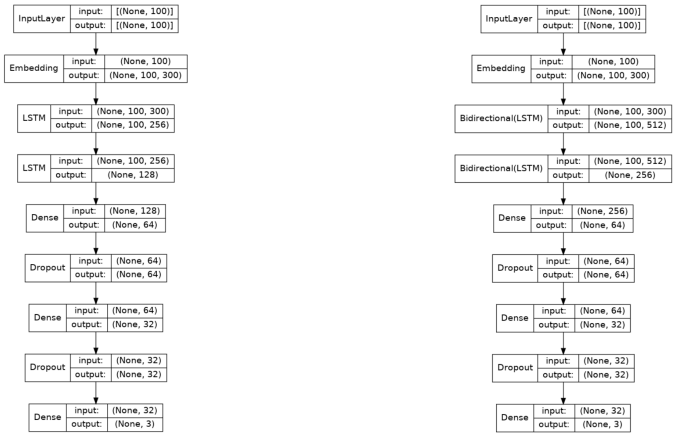
2. Target
Target 변수는 Spooky 데이터의 author 변수이다. author 변수는 class가 두 개 이상인 multiclass 변수인데, NLP 모델의 input으로 쓰기 위해 원핫벡터(One-hot vector) 형태로 가공해야 한다. author 변수의 class가 작가 이름인 object 형태이기 때문에 먼저 LabelEncoding을 이용해 integer class로 변경한 뒤, to_categorical 을 이용해 원핫벡터로 변환한다.

4 NLP Modeling

텍스트 전처리를 거친 Spooky 데이터를 이용하여 NLP 딥러닝 모델을 구축하고 학습시키자. 본 프로젝트에서는 두 종류의 모델을 비교 및 분석한다. 첫번째 모델은 Bidirectional LSTM 기반, 두번째 모델은 LSTM 기반 NLP 딥러닝 모델이다.

4.1 Model Structure

두 종류의 모델의 구조는 아래와 같다.



(a) Model 1: LSTM (b) Model 2: Bidirectional LSTM

Figure 4: Model Structure

두 NLP 모델 모두 Embedding - LSTM(256) - LSTM(128) - Dense(64) - Dense(32) - Dense(3, output layer) 의 구조를 갖는다. Output dense layer에서 3개의 class를 분류해야 하므로 node의 수는 3개, 활성화함수는 softmax를 사용한다. 각 layer을 구성하는 노드의 수는 모두 동일하며 유일한 차이점은 LSTM layer가 양방향, 단방향 이라는 것이다. NLP 모델을 구성하는 핵심적인 layer의 특징과 그 역할에 대해 알아보자.

Embedding layer
Embedding layer은 워드임베딩(Word Embedding)을 구현하는 layer이다. 워드임베딩은 단어 인덱스로 이루어진 input 시퀀스를 밀집벡터(dense vector)로 변환하는 기법이다. Spooky 데이터의 input 데이터의 dimension은 $(20170, 100)$ 이다. 여기서 20170은 unique한 단어 인덱스의 개수이고, 100은 앞서 Tokenize and create input data에서 설정한 sequence의 최대 길이다. Embedding layer은 input 시퀀스를 밀집벡터로 바꿔주지만, 단어 사이의 관계 및 유사도를 반영하지 못한다는 단점이 있다.

Glove embedding matrix
Embedding layer가 단어 사이의 관계를 반영하지 못하는 단점을 보완하기 위해 Glove를 embedding layer에 적용한다. Glove는 사전학습된 워드임베딩 기법 중 하나로, 단어 간의 유사도를 반영하면서 전체 텍스트에서의 단어 발생 빈도까지 반영하는 기법이다.
사전학습된 Glove의 단어별 가중치를 embedding matrix 형태로 Embedding layer의 weight로 설정한다. 이 때 Embedding matrix의 dimension은 $(20170, 300)$ 이다. Embedding matrix 생성 방법은 아래와 같다.

1. 단어가 Glove에 존재하면 이 단어의 가중치 벡터를 가져온다.
2. 단어의 인덱스에 해당하는 Embedding matrix의 row에 삽입한다.

예를 들어 pretty라는 단어가 있고 이 단어의 인덱스를 250이라 하자. 그러면 Glove에서 pretty가 가지는 가중치 벡터를 embedding matrix의 251번째 행에 넣는다. 자세한 코드는 Code 섹션을 참고하면 된다.

Dropout layer
NLP 모델의 overfitting을 방지하기 위해 LSTM layer과 Dense layer에 dropout layer을 추가한다. LSTM layer에서는 dropout과 recurrent_dropout 옵션을 각각 0.5, 0.3으로 설정한다. Dense layer에서의 dropout 비율은 0.5로 설정한다. 추가적으로 Dense layer에 kernel initizalier을 he_normal로 설정하여 Dense layer의 input과 output의 variance를 일정하게 유지해 overfitting을 방지하고자 했다.

4.2 Model Settings

두 NLP 모델의 공통적인 모델 세팅은 표와 같다. 모델 평가 metric은 accuracy를, loss는 multiclass 분류 모델이기 때문에 categorical crossentropy를 사용한다. 손실함수를 categorical crossentropy로 설정했을 때 metric에 accuracy를 입력하면 자동으로 category accuracy로 설정된다. Categorical accuracy는 예측한 class가 실제 class와 일치하는 비율이다.

Table 2: Settings for NLP models

Setting	Description
epochs/batch size	30/32
metrics	accuracy
loss	categorical crossentropy
optimizer	Adam
EarlyStopping	patience = 3
ReduceLROnPlateau	factor = 0.1

5 Result and Analysis

이 섹션에서는 구축한 두 종류의 NLP 딥러닝 모델을 학습하고 각 모델의 성능을 평가하고 결과를 해석한다.

5.1 Model perforamce result

Model 1: LSTM

LSTM layer을 사용한 Model 1의 학습 결과에 대한 그래프는 아래와 같다. Figure 5는 Model 1의 epoch 별 학습 결과를, Table 3은 Model 1의 최종 perforamce를 나타낸다.

- EarlyStopping에 의해 Model 1은 12번째 epoch까지만 학습했다.
- Epoch이 진행될수록 train set의 accuracy가 validation set의 accuracy보다 점점 커지는 현상이 발생한다.

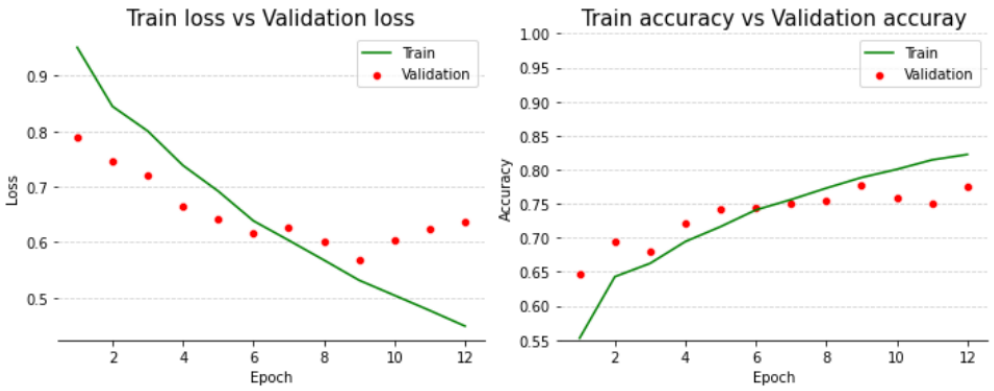


Figure 5: Model 1 Loss and Accuracy

Table 3: Model 1 Performance

Metric	Result
Loss	0.6369
Accuracy	0.7752

Model 2: Bidirectional LSTM

LSTM layer을 사용한 Model 1의 학습 결과에 대한 그래프는 아래와 같다. Figure 6는 Model 2의 epoch 별 학습 결과를, Table 4은 Model 2의 최종 perforamce를 나타낸다.

- EarlyStopping에 의해 Model 1은 15번째 epoch까지만 학습한다.
- Model 2 역시 epoch이 진행될수록 train set의 accuracy가 validation set의 accuracy보다 점점 커지는 현상이 발생한다.
- Model 1과 비교했을 때 Bidirectional layer을 사용한 Model2의 최종 performance는 Model 1보다 우수하다. Loss는 더 낮고, accuracy는 더 높다.
- Model 1와 Model 2 모두 epoch이 진행될수록 train set과 validation set의 loss 및 accuracy 차이가 커지는 경향을 보였는데, Model 2에서 그 차이가 좀 더 크게 나타난다.

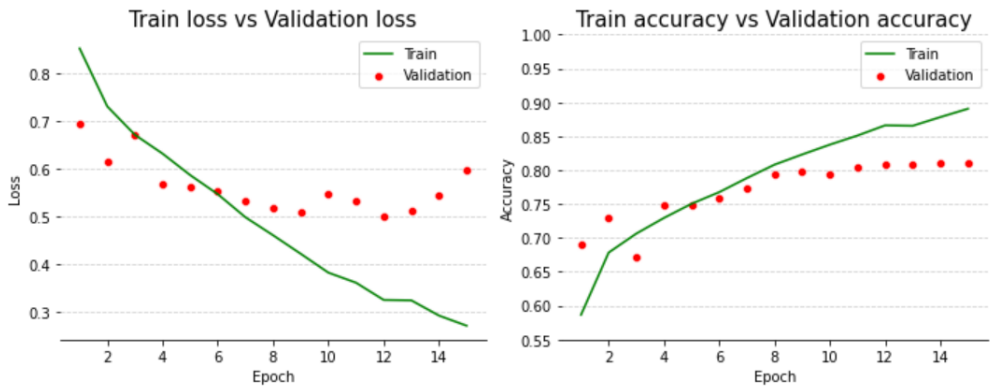


Figure 6: Model 2 Loss and Accuracy

Table 4: Model 2 Performance

Metric	Result
Loss	0.5979
Accuracy	0.8156

5.2 Analysis of result

Why is perforamce of Model 2 better than Model 1?
Model 1과 Model 2의 유일한 차이점은 LSTM layer의 Bidirectional 여부이다. 일방향 LSTM layer과 달리 Bidirectional LSTM의 경우 순방향으로 LSTM layer을 학습하고 역방향으로도 LSTM layer을 학습한다. 즉 이전 정보 뿐만이 아니라 이후 정보까지 모델이 학습하여 input 시퀀스에 대한 정보를 LSTM 보다 더 많이 반영한다.

NLP 분석에서는 텍스트를 구성하는 단어들 간의 문맥적 상관관계를 파악하는 것이 매우 중요하다. 이를 위해서는 NLP 모델이 텍스트에 대한 정보를 최대한 많이, 효율적으로 학습하는 것이 관건이데, 이러한 맥락에서 Bidirectional LSTM이 일반 LSTM 모델보다 더 우수하다. 따라서 Spooky 데이터를 구성하는 텍스트를 Bidirectional LSTM layer을 포함하는 Model 2가 더 우수하게 학습한다고 해석할 수 있다.

Why does overfitting occur in Model 1 and Model 2?
Model 1과 Model 2 모두 epoch가 진행될수록 train과 validation 성능(loss, accuracy 기준) 차이가 커지는 overfitting 현상이 발생하는 것을 확인할 수 있다. Dropout, kernel initializer를 사용했음에도 불구하고 overfitting이 발생하는 이유는 무엇일까?

가장 유력한 원인은 모델 복잡도 때문이라고 예상된다. Model 1과 Model 2 모두 LSTM layer 2개와 출력층 제외 Dense layer을 두 개 포함한다. 그리고 첫번째 LSTM layer에는 return_sequences = True 옵션을 적용했는데, 이는 LSTM layer에서 마지막 시퀀스 뿐만이 아니라 각 시퀀스에서 output을 출력하게 하는 설정이다. 각 LSTM layer에는 노드를 256개, 128개씩 부여했는데 그 결과 모델을 구성하는 모수의

수가 682만개 이상으로 input 시퀀스의 크기(15633×100)에 비해 지나치게 많아진다. 일반적으로 모델의 모수의 수가 지나치게 많을 경우 overfitting이 발생하는 경우가 빈번하다.

Model 2에서 Model 1보다 overfitting이 심하게 발생하는 이유도 같은 맥락이다. Model 2는 Bidirectional LSTM을 기반으로 하기 때문에 순방향, 양방향으로 학습을 두 번 진행한다. 따라서 Bidirectional LSTM에서의 노드의 수는 LSTM layer의 두 배가 된다. 즉 Model 2에서 LSTM layer의 노드 수를 256개, 128개씩 부여했지만 학습에는 512개, 256개의 노드가 이용된 것이다. 따라서 Model 2에서는 Model 1보다 모델에 사용된 모수의 수가 약 100만개 이상 더 많기 때문에 overfitting 현상이 더 두드러진다고 해석할 수 있다.

5.3 Simplified Bidirecional Model

앞서 분석한 overfitting 발생 원인을 고려하여 Model 2의 구조를 더 간단하게 변경하고 학습을 진행한다. 간소화된 NLP 모델 구조는 아래 그림과 같다. Bidirectional Layer을 두개에서 한개로 줄이고, return_sequences 옵션을 False로 설정했다.

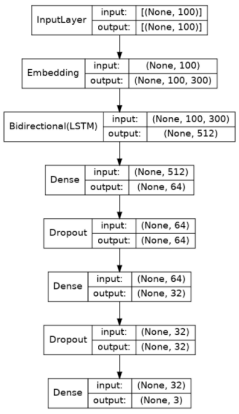


Figure 7: Simplified Model 2 structure

모델의 epoch별 loss와 accuray는 Figure 8와 같다. Model 2와 비교했을 때 성능은 더 나아지지 않았지만 overfitting 현상은 완화되었다. 모델 복잡도가 overfitting의 원인이라는 가설이 틀리지 않았다는 것을 확인할 수 있다.

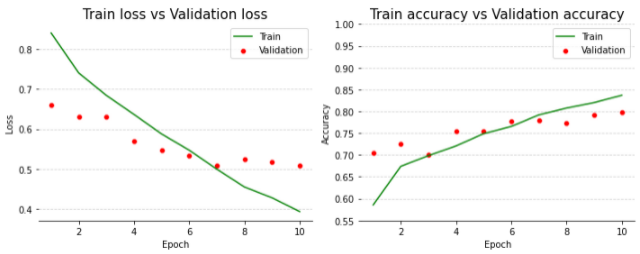


Figure 8: Simplified Model 2 Performance

6 Conclusion

본 프로젝트에서는 NLP 모델을 이용하여 Spooky Author Identification 데이터의 텍스트를 이용해 텍스트를 작성한 작가를 예측하였다. Text preprocessing을 통해 raw 텍스트를 모델의 input으로 가능한 형태로 가공하고, 다양한 모델링 기법을 적용하고 그 성능과 결과를 비교 및 분석을 했다는 것에 프로젝트의 의의가 있다. 양방향 LSTM과 일방향 LSTM의 차이를 비교할 수 있었다. 또한 deep learning 모델을 구축할 때 모델 복잡도가 지나치게 커지지 않도록 모델 파라미터 수와 layer의 수를 조절하여 overfittig이 발생하지 않는 최적의 모델을 만드는 것에 대한 중요성을 배울 수 있었다.
한 가지 아쉬운 점은 batch size, callback의 patience, 시퀀스의 최대 길이(maxlen) 등등을 임의로 조정했다는 것이다. 자료조사를 통해 이들 옵션에 대해서도 GridSearch를 적용할 수 있다고 하는데, deep learning 모델 최적화 방법론에 대해서 더 연구해볼 예정이다.

7 Code

```
% Text Preprocessing
stopwords = nltk.corpus.stopwords.words('english')
more_words = ['one', 'would', 'could']
for w in more_words:
    stopwords.append(w)

lemmatizer = WordNetLemmatizer()

def text_cleansing(text):
    tokens = text_to_word_sequence(text, filters = , lower = True, split = ' ')
    tokens = [token.strip() for token in tokens]
    new_tokens = [token for token in tokens if token not in stopwords]
    new_tokens_lem = []
    for word, tag in pos_tag(new_tokens):
        new_tag = tag[0].lower()
        new_tag = new_tag if new_tag in ['r', 'n', 'v', 'j'] else None
        lemma = lemmatizer.lemmatize(word, pos = new_tag) if new_tag in ['r','n','v']
    else (lemmatizer.lemmatize(word, pos = 'a') if new_tag == 'j' else word)
        new_tokens_lem.append(lemma)
    new_text = ' '.join(new_tokens_lem)
    return new_text

train['new_text'] = train['text'].apply(text_cleansing)

% Input
maxlen = 100
tokenizer = Tokenizer(num_words = None)
tokenizer.fit_on_texts(list(train['new_text'].values))
train_sequence = tokenizer.texts_to_sequences(train['new_text'].values)
train_data = pad_sequences(train_sequence, maxlen = maxlen)

encoder = LabelEncoder()
y_label = encoder.fit_transform(train['author'])
y = to_categorical(y_label)

% Embedding matrix
embedding_idx = {}
```

```
f = open('glove.42B.300d.txt', encoding = 'utf-8')
for line in f:
    values = line.split()
    word = values[0]
    seq = np.asarray(values[1:], dtype='float32')
    embedding_idx[word] = seq
f.close()

max_words = len(tokenizer.word_index) + 1
embedding_matrix = np.zeros((max_words, 300))
for word, i in tokenizer.word_index.items():
    if word in embedding_idx:
        embedding_vector = embedding_idx.get(word)
        embedding_matrix[i] = embedding_vector

% Model 1 & 2 (Model 1 - omit Bidirectional)
X_train, X_val, y_train, y_val = train_test_split(train_data, y, test_size = 0.2,
    random_state = 0)
embedding_dim = 300

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length = maxlen))
model.add(Bidirectional(LSTM(256, dropout=0.5, recurrent_dropout=0.3, return_sequences
     = True)))
model.add(Bidirectional(LSTM(128, dropout=0.5, recurrent_dropout=0.3)))
model.add(Dense(64, activation='relu', kernel_initializer = 'he_normal'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation = 'softmax'))

model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False

es = EarlyStopping(monitor = 'val_loss', patience = 3)
rp = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.1, patience = 3)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics =  ['acc'])

history = model.fit(X_train, y_train, epochs = 30, batch_size = 32, validation_data =
    (X_val, y_val), callbacks = [es, rp])


% Simplified Model
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length = maxlen))
model.add(Bidirectional(LSTM(256, dropout=0.5, recurrent_dropout=0.3)))
model.add(Dense(64, activation='relu', kernel_initializer = 'he_normal'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation = 'softmax'))

model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

Reference

https://www.kaggle.com/c/spooky-author-identification
https://www.aitimes.kr/news/articleView.html?idxno=15036
https://keras.io/ko/preprocessing/text/
https://wikidocs.net/21693
텐서플로 케라스를 이용한 딥러닝 Chapter 7