

## Classification of Texts Using LSTM and LDA

Seong-Yeon Park<sup>a</sup>, Hyun-Kyung Noh<sup>b</sup>, Seung-Yeon Hwang<sup>c</sup>, Jae-Kon Oh<sup>d</sup>, Jeong-Joon Kim<sup>e</sup>

<sup>a</sup>Dept of Computer Engineering, Korea Polytechnic University

<sup>b</sup>Dept of Computer Engineering, Korea Polytechnic University

<sup>c</sup>Dept of Computer Engineering, University of Anyang

<sup>d</sup>Director of JINWOO INDUSTRIAL SYSTEM CO.,LTD

<sup>e</sup>Dept of Software, University of Anyang

### Abstract

With the approach of the Fourth Industrial Revolution, information has become a powerful resource for society and the economy to operate and develop. In particular, as customized algorithms have become an essential service in most systems, the importance of big data-based information processing technology is also deepening. However, human languages have extreme variability compared to programming languages, so interpretation and processing are difficult. Efficient measures need to be taken to extract the desired information from these unstructured data. Therefore, in this paper, we identify and develop a more effective analytical system by classifying papers that are subdivided into five categories within the topic of 'technique' into LSTM and LDA techniques after learning.

**Keywords:** LSTM, LDA, NLP, Text Classification

### 1. Introduction

Since the Fourth Industrial Revolution, the world has changed rapidly, and technologies such as big data analysis, artificial intelligence, and natural language processing have become the basis for information processing. Among these, natural language processing boasts a much higher level of difficulty than in other fields. This is because human languages have extreme variability compared to programming languages. The programming language has a strict grammatical structure that is intuitive, clear, and easy to process. By comparison, human language is ambiguous, and the meaning changes continuously over time, making the process complicated and relatively less accurate. However, most of the data recently utilized are unstructured data that do not have a certain format, such as e-mail, SNS, etc. In order to extract and process useful information from these data, there is an increasing need to efficiently analyze and classify those resources.

Natural language processing refers to the process by which computers interpret human language. The areas in which this is utilized vary widely. It is possible to analyze the emotions revealed in posts on SNS and other sites, and it can be seen commonly in translators. Furthermore, it is also used to analyze documents and classify them into specific categories. Document classification can also be done in a variety of ways. Among them, many well-known methods are Latent Dirichlet Allocation (LDA) and Long Short Term Memory (LSTM), a type of Recurrent Neural Networks (RNNs). LDA is a type of unsupervised learning that classifies document content, while LSTM is supervised learning using a neural network. To compare the two performances, in this study, we classify papers that are subdivided into five categories again within a category called 'technique'.

This paper consists of the following. Chapter 2 describes the relevant research, Chapter 3 describes the implementation of the system, and Chapter 4, conclusion according to the results of the implementation.

## 2. Related Work

### 2.1 Text preprocessing

The process of preprocessing text before analyzing a document is essential. In the case of English, there is a ‘lowering technique’ that converts capital letters into lowercase letters. When processing data, computers can distinguish uppercase and lowercase letters, and thus need to unify texts. In addition, there is a ‘lemmatizing technique’ for regularizing refractive words, i.e., extracting basic dictionary words. For example, when there are words such as ‘studies’, ‘studying’, ‘studied’, the root verb ‘study’ is extracted from these. In addition, plural nouns can be replaced with singular nouns. Punctuation also affects data analysis, so preprocessing is necessary. Finally, we need to remove the words that do not have a special meaning, called ‘stopwords’, such as ‘and’, ‘you’, and ‘not’.

### 2.2 LSTM

Long Short-Term Memory (LSTM) is a deep learning technique used during natural language processing, especially for sentence learning, similar to RNN techniques that remember previously entered data and assign weights when multiple data are entered in order. However, LSTM is a model (Bengio et al., 1994) designed to solve the long-term dependency problem of RNNs (which using only one recursive layer), thus controlling information transfer and propagation through four layers: Forget, Input, Update, and Output. Thanks to these algorithms, LSTM has the advantage of being able to arrange given words sequentially to be predictable in advance, and is actively applied to language models that need previous data to process with current information.

### 2.3 LDA

Text mining process is required to derive meaningful information from unstructured data. Among text mining techniques, topic modeling is a method of extracting information from documents and then collecting words containing similar topics to find embedded topics in texts and the most algorithm of this is the Latent Dirichlet Allocation (LDA). LDA is also a model that can generate arbitrary documents, given the probability distribution of topics for a given document and the probability distribution of words for each topic, by repeating the probability choice of the topic and the probability selection of words present in the selected topic. LDA algorithms have been most commonly used in recent studies utilizing topic modeling, requiring text preprocessing to detect meaningful patterns in large text.

## 3. System Implementation

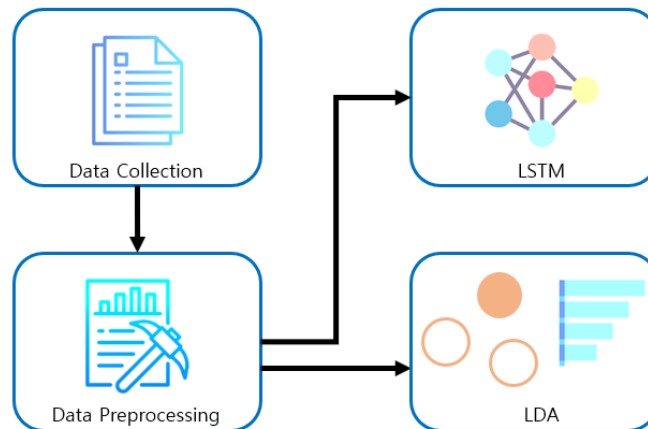


Fig 1: System Diagram

As shown in Figure 1, the system in this paper is proceeded by analyzing the results after data collection, data preprocessing, and data injection into LSTM and LDA.

### 1. Big Data Processing

```

import spacy
import gensim
import pandas as pd
from textblob import Word
  
```

```

from nltk.corpus import stopwords

from gensim.parsing.preprocessing import STOPWORDS

file = 'total.xlsx'

df = pd.read_excel(file, 'Sheet1', header=0,
engine='openpyxl')

print(df.category.value_counts())

df
    
```

Fig 2: Call Modules and Collected Data

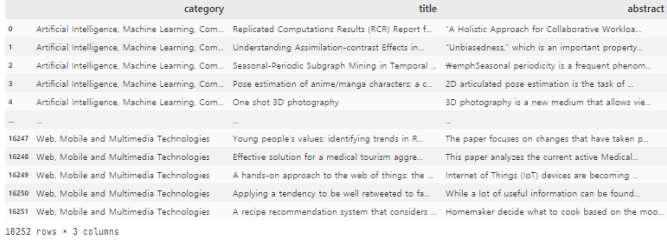
|   |
|---|
| <p>Networks and Communications 3417</p> <p>Information Systems, Search, Information Retrieval, Database Systems, Data Mining, Data Science 3376</p> <p>Artificial Intelligence, Machine Learning, Computer Vision, Natural language processing 3240</p> <p>Web, Mobile and Multimedia Technologies 3127</p> <p>Security and Privacy 3092</p> <p>Name: category, dtype: int64</p>  |
|---|

Fig 3: Collected Data Data Frame

Figure 2 is the process of 1) importing of all the modules needed and 2) calling the collected data. Figure 3 shows the information used in the file. There are 3,417 papers in ‘Networks and Communications’ category, 3,376 in ‘Information Systems, Information Retrieval, Database Systems, Data Mining, Data Science’ category, 3,240 papers in ‘Artificial Intelligence, Machine Learning, Computer Vision, Natural language processing’, 3,127 in ‘Web, Mobile and Multimedia Technologies’, and 3,092 in ‘Security and Privacy’ category— a total of 16,252 categories and abstracts of papers.

|   |   |
|---|---|
| 1 | <pre> def singularize(words):      word = Word(words)      word = word.lemmatize()      return word     </pre>  |
| 2 | <pre> def make_bigrams(texts):      return bigram_mod[texts]     </pre>   |
| 3 | <pre> def make_trigrams(texts):      return trigram_mod[bigram_mod[texts]]     </pre>   |
| 4 | <pre> ss = spacy.load('en_core_web_sm')  sp_stopwords = ss.Defaults.stop_words  gensim_stopwords = STOPWORDS  stopwords = stopwords.words('english')  stopwords.extend(sp_stopwords)     </pre> |

## Classification of Texts Using LSTM and LDA

```
stopwords.extend(gensim_stopwords)
stop_words = list(set(stopwords))
```

Fig 4: Data Preprocessing Preparation

Figure 4 is a preparation process for data preprocessing. 1) Using the textblob module in 'Word' class, lemmatize each word to create a function that changes the plural form to singular form. Bigram refers to two frequently attached words in a document, and Trigram refers to three frequently attached words. 2) Bigram\_mod and 3) trigram\_mod functions are made to extract bigrams and trigrams. 4) Finally, create a set of stopwords.

```
# title + abstract
df['data'] = df['title'] + ' ' + df['abstract']

# lowercase
df['data'] = df['data'].str.lower()

# tokenize
df['data'] = df['data'].map(gensim.utils.simple_preprocess)

# remove punctuations
df['data'] = df['data'].apply(lambda x: [item.replace("[^a-zA-Z]", "") for item in x])

# remove too short words (less than 3 letters)
df['data'] = df['data'].apply(lambda x: [w for w in x if len(w) > 3])

texts = df['data']
texts = texts.apply(lambda x: [item for item in x if item not in stop_words])
texts = texts.apply(lambda x: [singularize(item) for item in x])

bigram = gensim.models.Phrases(texts, min_count=5, threshold=100)
bigram_mod = gensim.models.phrases.Phraser(bigram)
texts = texts.map(make_bigrams)

trigram = gensim.models.Phrases(texts, threshold=100)
trigram_mod = gensim.models.phrases.Phraser(trigram)
texts = texts.map(make_trigrams)
```

Fig 5: Data Preprocessing

Figure 5 is a process of data preprocessing. First, combine the title and abstract and add them to a new column called 'data'. Then replace the uppercase letters to lowercase letters, and tokenize the sentence into words. Remove punctuation marks and words of less than three letters. Extract only the words that are not in the following set of stopwords and change the plural to singular with the singularize function. Then combine the words with make\_bigram and make\_trigram written in Figure 4. The sentence that completes the process is shown in Figure 6.

```
['replicated', 'computation', 'result', 'report', 'holistic', 'approach', 'collaborative', 'workload', 'execution', 'volunteer', 'cloud', 'holistic', 'approach', 'collaborative', 'workload', 'execution', 'volunteer', 'cloud', 'proposes',
```

```
'novel', 'approach', 'task', 'scheduling', 'volunteer', 'cloud', 'volunteer', 'cloud', 'decentralized', 'cloud', 'system',
'based', 'collaborative', 'task', 'execution', 'client', 'voluntarily', 'share', 'unused', 'computational', 'resource',
'simulation', 'based', 'statistical', 'analysis', 'technique', 'particular', 'statistical', 'model', 'checking', 'author',
'approach', 'outperform', 'existing', 'distributed', 'task', 'scheduling', 'algorithm', 'case', 'computation', 'intensive',
'workload', 'analysis', 'considered', 'realistic', 'workload', 'benchmark', 'provided', 'google', 'replicated',
'computation', 'result', 'report', 'focus', 'prototypical', 'tool', 'implementation', 'article', 'perform', 'analysis',
'software', 'straightforward', 'install', 'representative', 'experimental', 'result', 'article', 'reproduced', 'reasonable',
'time', 'standard', 'laptop']
```

Fig 6: First Value of 'data' Column

|   |   |
|---|---|
| 1 | <pre>clean = [] for text in texts:     sentence = ''.join(text)     clean.append(sentence)</pre>  |
| 2 | <pre>new_df = pd.DataFrame(columns=['category', 'content'])  new_df['category'] = df.category new_df['content'] = clean new_df.to_excel('LSTM.xlsx', index=False)</pre> |

Fig 7: Preprocessed Data to New Data Frame

Finally, Figure 7 shows 1) data stored in the form of a list are combined into a single string, and 2) stored as a new Excel file.

## 2. LSTM

We now proceed with classifying the above preprocessed data with LSTM.

|   |   |
|---|---|
| 1 | <pre>import pandas as pd import numpy as np import tensorflow as tf from collections import Counter import matplotlib.pyplot as plt from tensorflow.keras.optimizers import Adam from sklearn.preprocessing import LabelEncoder from tensorflow.keras.models import Sequential from tensorflow.keras.utils import to_categorical from sklearn.model_selection import train_test_split from tensorflow.keras.preprocessing.text import Tokenizer from tensorflow.keras.preprocessing.sequence import pad_sequences from tensorflow.keras.layers import Dense, Flatten, LSTM, Dropout, Activation, Embedding, Bidirectional, Conv1D, MaxPooling1D</pre> |
| 2 | <pre>file = 'LSTM.xlsx' df = pd.read_excel(file, header=0, engine='openpyxl')  content = df.content</pre>   |

## Classification of Texts Using LSTM and LDA

```
label = df.category
```

Fig 8: LSTM Preparation

Modules and file required for LSTM are imported. When implementing code with supervised learning, such as LSTM, it is necessary to change the label of the data to an integer. This can be solved by the LabelEncoder function as shown in Figure 9.

```
categories = np.unique(np.array(label))

label_encoder = LabelEncoder()
label = label_encoder.fit_transform(label)

for i in categories:
    labeled = label_encoder.transform([i])
    print(labeled, '<--', i)
```

Fig 9: Label Encoder

After converting categories to integers, we can see that the results have changed as follows.

```
[0] <-- Artificial Intelligence, Machine Learning, Computer Vision, Natural language processing
[1] <-- Information Systems, Search, Information Retrieval, Database Systems, Data Mining, Data Science
[2] <-- Networks and Communications
[3] <-- Security and Privacy
[4] <-- Web, Mobile and Multimedia Technologies
```

Fig 10: Result of Label Encoder

The following train\_test\_split function divides documents into learning sets and test sets.

```
X_train, X_test, Y_train, Y_test = train_test_split(content,label, shuffle=True, random_state=100,
test_size=0.1)
```

Fig 11: Split Train Set and Test Set

As a result of separating the test set to 0.1, the train set is divided into 14,626 and the test set into 1,626.

```
tokenizer = Tokenizer(filters='#')
tokenizer.fit_on_texts(X_train)

threshold = 2
total_cnt = len(tokenizer.word_index) # number of unique words
rare_cnt = 0 # words appearing less than threshold
total_freq = 0 # total word frequencies in train set
rare_freq = 0 # total frequency of appearance of words less than threshold

# word: key, number of appearance: value
for key, value in tokenizer.word_counts.items():
    total_freq = total_freq + value
```

```

if (value < threshold):
    rare_cnt = rare_cnt + 1
    rare_freq = rare_freq + value

vocab_size = total_cnt - rare_cnt + 2

print('size of vocabulary :',total_cnt)
print('words appearing only once (rare words) :', rare_cnt)
print('percentage of rare words in a word set: ', (rare_cnt / total_cnt)*100)
print('percentage of rare word appearance to total appearance frequency: ', (rare_freq / total_freq)*100)
print('word set size :',vocab_size)

```

Fig 12: Tokenizing and Frequency of Words

```

size of vocabulary : 29556
words appearing only once (rare words) : 8565
percentage of rare words in a word set: 28.978887535525782
percentage of rare word appearance to total appearance frequency: 0.6329127954793843
word set size : 20993

```

Fig 13: Frequency of Words Result

Figure 12 is a code for analyzing the frequency of words, and the results are shown in Figure 13. There are 27,703 unique words and 8.472 sparse words, and the final number of words to be used is the number of unique words minus the sparse words. The reason for subtracting sparse words is that they are not related to other words, which can be nothing but noise.

|   |  |
|---|--|
| 1 | <pre> tokenizer = Tokenizer(vocab_size, oov_token='OOV', filters='#') tokenizer.fit_on_texts(X_train) X_train = tokenizer.texts_to_sequences(X_train) X_test = tokenizer.texts_to_sequences(X_test) </pre>   |
| 2 | <pre> print('maximum length of document :', max(len(l) for l in X_train)) print('average length of document :', sum(map(len, X_train))/len(X_train)) plt.hist([len(s) for s in X_train], bins=50) plt.xlabel('length of samples') plt.ylabel('number of samples') plt.show() </pre>        |
| 3 | <pre> def below_threshold_len(max_len, nested_list):     cnt = 0     for s in nested_list:         if(len(s) &lt;= max_len):             cnt = cnt + 1     print('percentage of samples with length %s or less out of total samples: %s' % (max_len, (cnt / len(nested_list))*100)) </pre> |

## Classification of Texts Using LSTM and LDA

```

max_len = 300
below_threshold_len(max_len, X_train)

```

Fig 14: Encoding of Train Set, Test Set and Graphing of Document Information

1) The following is the process of encoding the train set and the test set into vectors. It selects frequently used words, builds indexes accordingly with the `fit_on_texts` function, and converts the words into a sequence of numbers using the `text_to_sequences` function. 2) Also, a graph is printed to know the number of documents and the length of each document. 3) With a maximum length of 300, documents not exceeding this value, i.e., all documents, are included in the analysis. The results are shown in Figure 15.

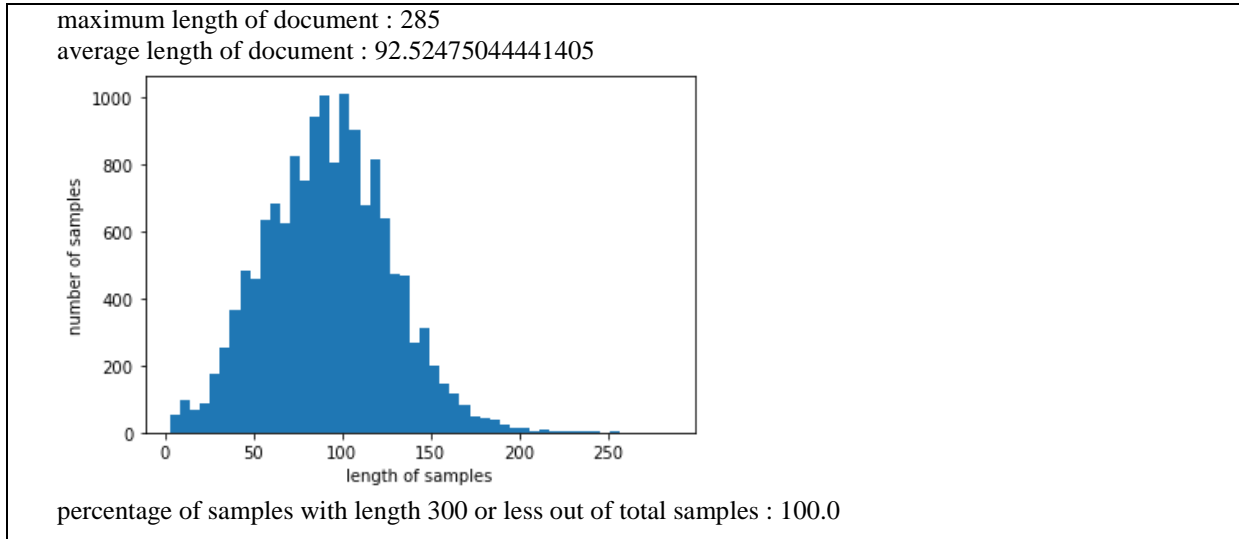


Fig 15: Result of Fig. 14 (2) and (3)

|   |  |
|---|--|
| 1 | <pre> X_train = pad_sequences(X_train, maxlen = max_len) X_test = pad_sequences(X_test, maxlen = max_len) </pre> |
| 2 | <pre> Y_train = to_categorical(Y_train) Y_test = to_categorical(Y_test) </pre>                                   |

Fig 16: Pad Sequence and One-Hot Encoding

Figure 16 shows 1) using the `pad_sequences` function from the sequence of integers (from Figure 14) that the padding 0 is placed in the remaining text relative to the longest text and transformed into a sequence of equal lengths. 2) In case of label, we transform the results of Label Encoder in Figure 9 into vectors using one-hot encoding techniques.

```

model = Sequential()
model.add(Embedding(vocab_size, 64, input_length=max_len))
model.add(Dropout(0.5))
model.add(Conv1D(64, 5, padding='valid', activation='relu', strides=1))
model.add(Dropout(0.5))
model.add(MaxPooling1D(pool_size=4))
model.add(Dropout(0.5))
model.add((Bidirectional(LSTM(100, recurrent_dropout=0.5))))
model.add(Dense(5, activation='softmax'))
model.summary()

```

Fig 17: Neural Network Model



Figure 17 shows the steps to model generation. Embedding Layer works to make words into dense vectors. It then extracts features with the Conv1D Layer and MaxPooling Layer, and uses the LSTM Layer to store information over the long term. At this time, the Bidirectional Layer allows forward as well as backward propagation. Finally, since it is classified into five categories, enter 5 as an argument in the Dense Layer and the activation function softmax. The generated model is shown in Figure 18.

```

Model: "sequential"
-----
Layer (type)                Output Shape              Param #
-----
embedding (Embedding)       (None, 300, 64)          1230912
dropout (Dropout)           (None, 300, 64)          0
-----
conv1d (Conv1D)              (None, 296, 64)          20544
dropout_1 (Dropout)          (None, 296, 64)          0
-----
max_pooling1d (MaxPooling1D) (None, 74, 64)           0
dropout_2 (Dropout)          (None, 74, 64)           0
-----
bidirectional (Bidirectional) (None, 200)              132000
-----
dense (Dense)                 (None, 5)                 1005
-----
Total params: 1,384,461
Trainable params: 1,384,461
Non-trainable params: 0
-----
    
```

Fig 18: Created Model

We then run the train the model 20 times with Adam optimizer and store the results of each learning in a history variable.

```

optimizer = Adam(lr=0.001, decay=1e-6)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=20, validation_data=(X_test, Y_test), batch_size=60)
    
```

Fig. 19. Train Set Fitting

```

Epoch 1/20
244/244 [=====] - 210s 820ms/step - loss: 1.5882 - accuracy: 0.2360 - val_loss: 1.4746 - val_accuracy: 0.3567
Epoch 2/20
244/244 [=====] - 230s 941ms/step - loss: 1.3671 - accuracy: 0.4093 - val_loss: 1.3987 - val_accuracy: 0.3948
Epoch 3/20
244/244 [=====] - 220s 901ms/step - loss: 1.1960 - accuracy: 0.5061 - val_loss: 1.4004 - val_accuracy: 0.3936
Epoch 4/20
244/244 [=====] - 210s 861ms/step - loss: 1.0479 - accuracy: 0.5929 - val_loss: 1.4610 - val_accuracy: 0.3905
Epoch 5/20
244/244 [=====] - 174s 712ms/step - loss: 0.9111 - accuracy: 0.6456 - val_loss: 1.5190 - val_accuracy: 0.3899
Epoch 6/20
244/244 [=====] - 194s 792ms/step - loss: 0.7846 - accuracy: 0.7038 - val_loss: 1.5954 - val_accuracy: 0.3641
Epoch 7/20
244/244 [=====] - 184s 755ms/step - loss: 0.6891 - accuracy: 0.7450 - val_loss: 1.6326 - val_accuracy: 0.3579
Epoch 8/20
244/244 [=====] - 173s 708ms/step - loss: 0.6144 - accuracy: 0.7741 - val_loss: 1.6017 - val_accuracy: 0.3438
Epoch 9/20
244/244 [=====] - 172s 704ms/step - loss: 0.5555 - accuracy: 0.7916 - val_loss: 1.6000 - val_accuracy: 0.3253
Epoch 10/20
244/244 [=====] - 205s 839ms/step - loss: 0.5110 - accuracy: 0.8025 - val_loss: 2.0013 - val_accuracy: 0.3339
Epoch 11/20
244/244 [=====] - 204s 837ms/step - loss: 0.4707 - accuracy: 0.8152 - val_loss: 2.0127 - val_accuracy: 0.3339
Epoch 12/20
244/244 [=====] - 221s 907ms/step - loss: 0.4447 - accuracy: 0.8320 - val_loss: 2.2073 - val_accuracy: 0.3253
Epoch 13/20
244/244 [=====] - 200s 820ms/step - loss: 0.4231 - accuracy: 0.8312 - val_loss: 2.2499 - val_accuracy: 0.3321
Epoch 14/20
244/244 [=====] - 180s 738ms/step - loss: 0.4007 - accuracy: 0.8361 - val_loss: 2.3548 - val_accuracy: 0.3223
Epoch 15/20
244/244 [=====] - 206s 843ms/step - loss: 0.3755 - accuracy: 0.8478 - val_loss: 2.4442 - val_accuracy: 0.3161
Epoch 16/20
244/244 [=====] - 192s 786ms/step - loss: 0.3553 - accuracy: 0.8554 - val_loss: 2.5131 - val_accuracy: 0.3284
Epoch 17/20
244/244 [=====] - 171s 689ms/step - loss: 0.3365 - accuracy: 0.8599 - val_loss: 2.6641 - val_accuracy: 0.3106
Epoch 18/20
244/244 [=====] - 168s 689ms/step - loss: 0.3388 - accuracy: 0.8565 - val_loss: 2.7061 - val_accuracy: 0.3186
Epoch 19/20
244/244 [=====] - 160s 655ms/step - loss: 0.3238 - accuracy: 0.8636 - val_loss: 2.7907 - val_accuracy: 0.3149
Epoch 20/20
244/244 [=====] - 136s 550ms/step - loss: 0.3144 - accuracy: 0.8615 - val_loss: 2.8575 - val_accuracy: 0.3161
    
```

Fig. 20. Train Set Fitting

For code visualizing Figure 20, see Figure 21.

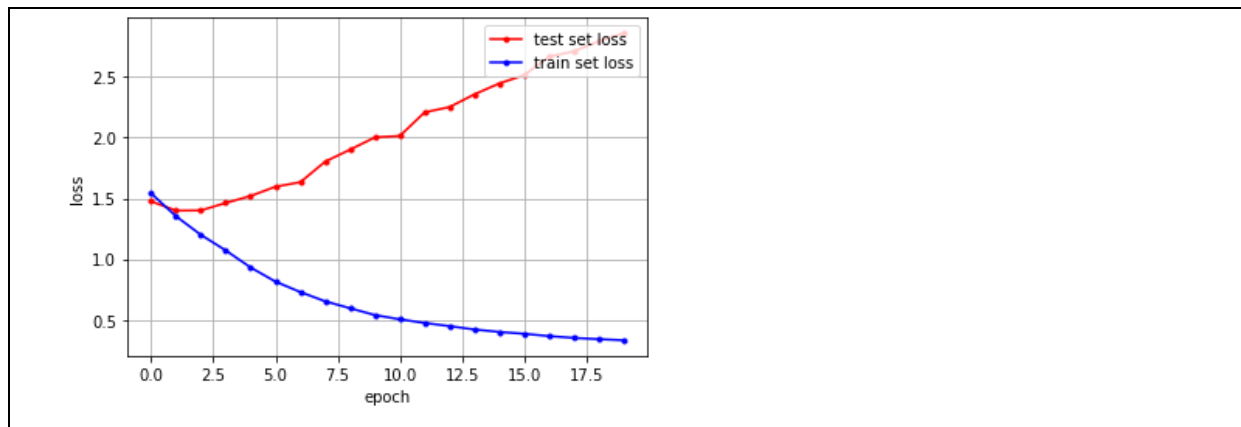
```

1 test_loss = history.history['val_loss']
    
```

## Classification of Texts Using LSTM and LDA

|   |  |
|---|--|
|   | <pre>train_loss = history.history['loss']  x_len = np.arange(len(train_loss)) plt.plot(x_len, test_loss, marker='.', c='red', label='test set loss') plt.plot(x_len, train_loss, marker='.', c='blue', label='train set loss')  plt.legend(loc='upper right') plt.grid() plt.xlabel('epoch') plt.ylabel('loss') plt.show()</pre>   |
| 2 | <pre>test_acc = history.history['val_accuracy'] train_acc = history.history['accuracy']  x_len = np.arange(len(train_loss)) plt.plot(x_len, test_acc, marker='.', c='red', label='test set accuracy') plt.plot(x_len, train_acc, marker='.', c='blue', label='train set accuracy')  plt.legend(loc='upper right') plt.grid() plt.xlabel('epoch') plt.ylabel('accuracy') plt.show()</pre> |

Fig. 21. Graphing Loss and Accuracy of Train Set and Test Set



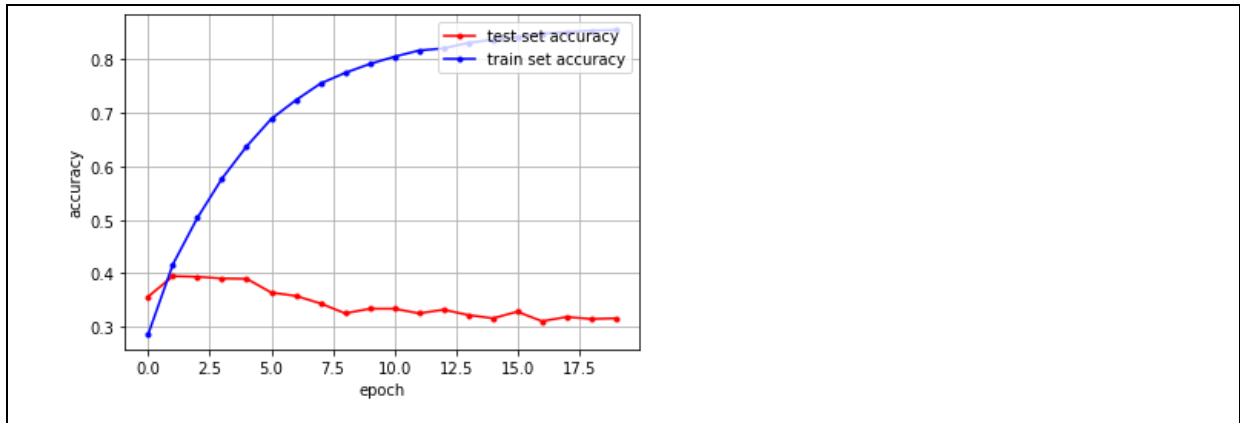


Fig. 22. Result of Loss and Accuracy

The results from Figure 22 show a typical overfitting graph despite training only 20 times, and its accuracy is very low. Based on the process, it may be caused by incorrect model creation or less data used for learning.

### 3. LDA

|   |  |
|---|--|
| 1 | <pre>import os import gensim import pyLDAvis import pandas as pd from gensim import corpora from pyLDAvis import gensim as gensimvis</pre> |
| 2 | <pre>file = 'LSTM.xlsx' df = pd.read_excel(file, 'Sheet1', header=0, engine='openpyxl')</pre>  |

Fig. 23. LDA Preparation

Call the modules required by the LDA, and retrieve preprocessed data.

|  |   |
|--|---|
|  | <pre>texts = df.content.str.split()  dictionary = corpora.Dictionary(texts) corpus = [dictionary.doc2bow(text) for text in texts]  num_tokens = len(dictionary) print('Number of unique tokens: %d' % num_tokens)  num_doc = len(corpus) print('Number of documents: %d' % num_doc)</pre> |
|--|---|

Fig. 24. Data Encoding and Word Frequency

|  |
|--|
| <pre>Number of unique tokens: 30924 Number of documents: 16252</pre> |
|--|

Fig. 25. Number of Unique Tokens and Documents

After each ID is given to the word, the frequency of word appearance in each document is determined. The result of the code in Figure 24 is in Figure 25.

## Classification of Texts Using LSTM and LDA

```

NUM_TOPICS = 5

ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics=NUM_TOPICS,
id2word=dictionary, passes=120)
    
```

Fig. 26. Training LDA Model

The next step is to train the LDA model. The document data (corpus) and the number of categories are passed by arguments, where corpus specifies 120 per batch.

```

pyLDAvis.enable_notebook()

vis = pyLDAvis.gensim.prepare(ldamodel, corpus, dictionary)

pyLDAvis.save_html(vis, 'LDA_topic.html')

pyLDAvis.display(vis)
    
```

Fig. 27. Visualization of LDA Model

Figure 27 shows the code for visualizing LDA. Each circle represents five categories, and when clicking on one circle shows information about the subject of the circle. A red bar graph represents the number of appearances of terms generated by a given topic, while a blue bar graph represents the overall frequency of each term within a document. If the circle is not selected, it shows the most important 30 words in the document, and the lambda value in the upper right corner determines the weight given to the probability of the term in the category. The closer the lambda value is to zero, the more words specialized for that topic.

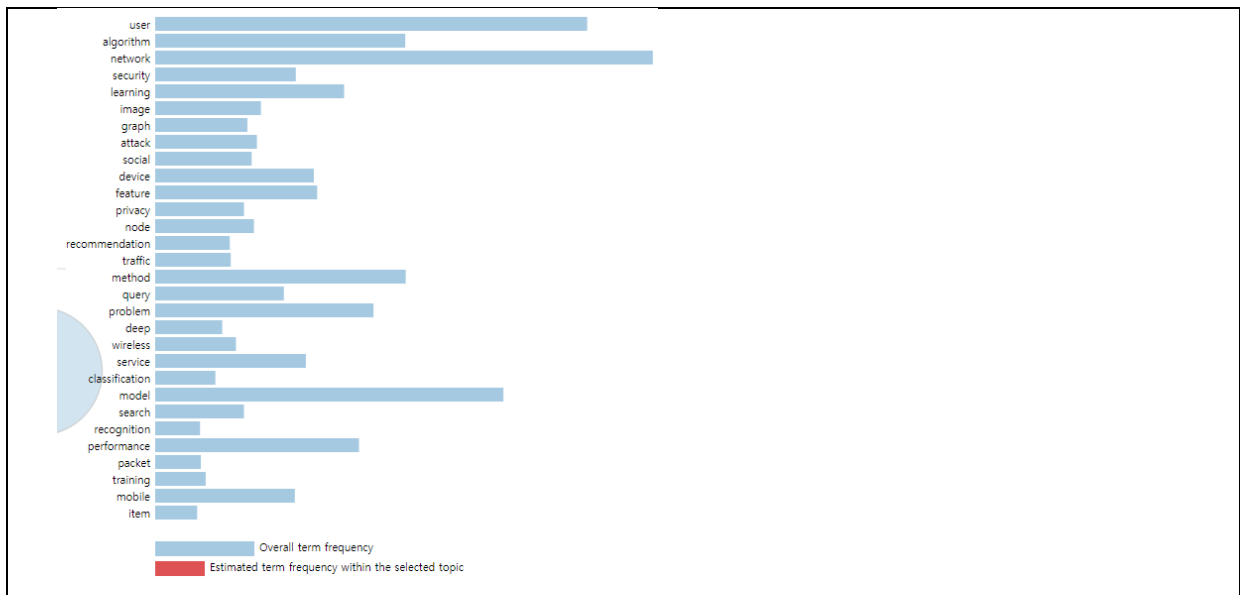


Fig. 28. No Topic Selected

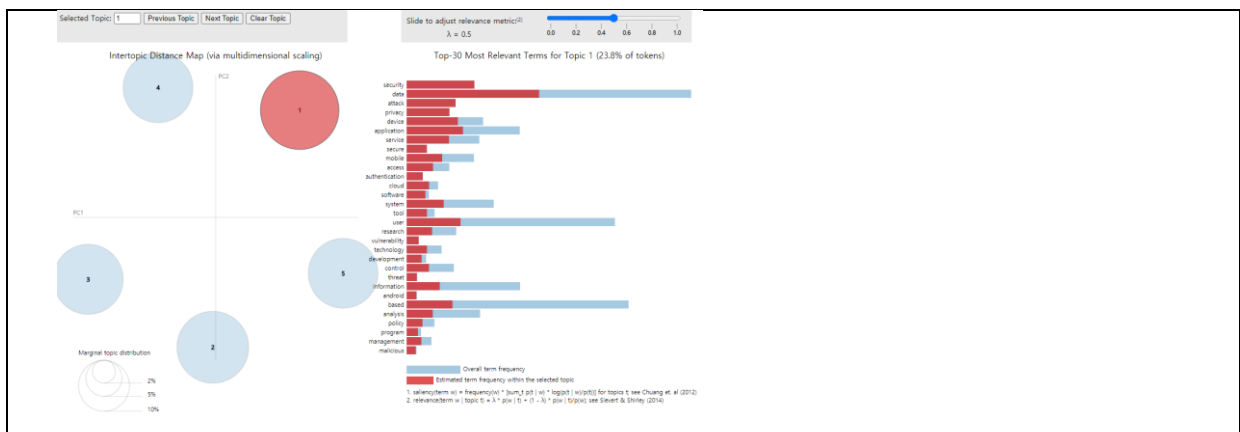


Fig. 29. Topic 1 Selected

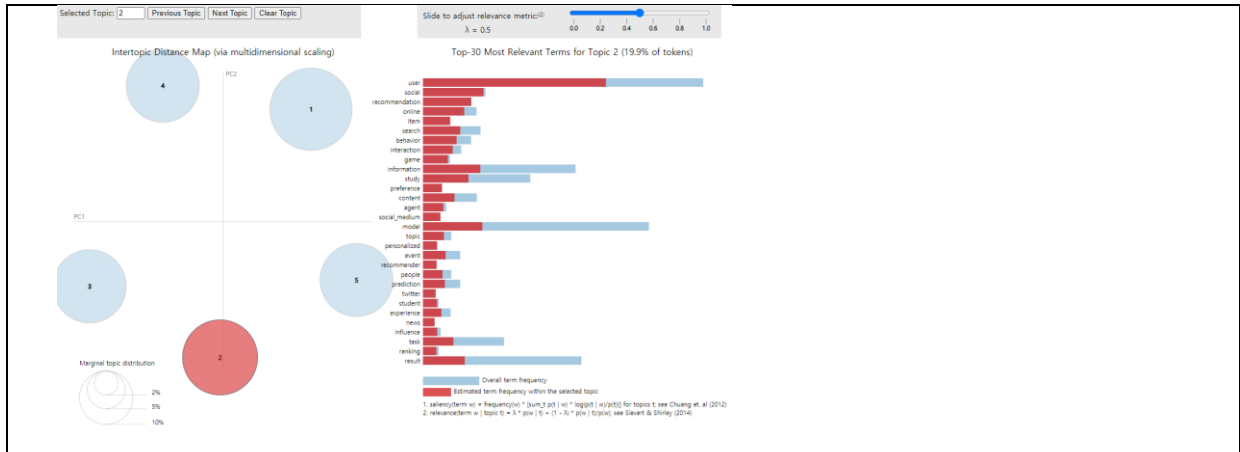


Fig. 30. Topic 2 Selected

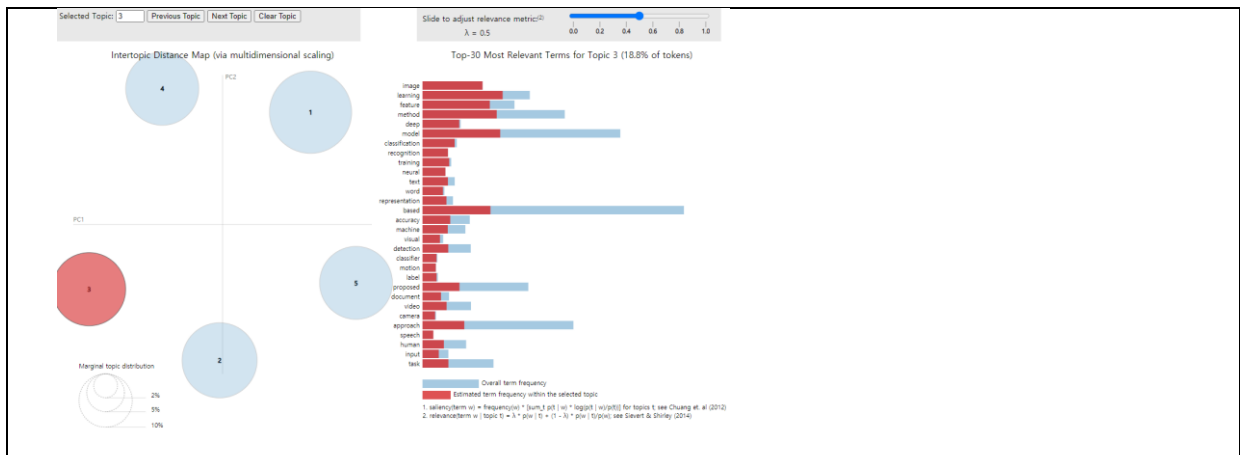


Fig. 31. Topic 3 Selected

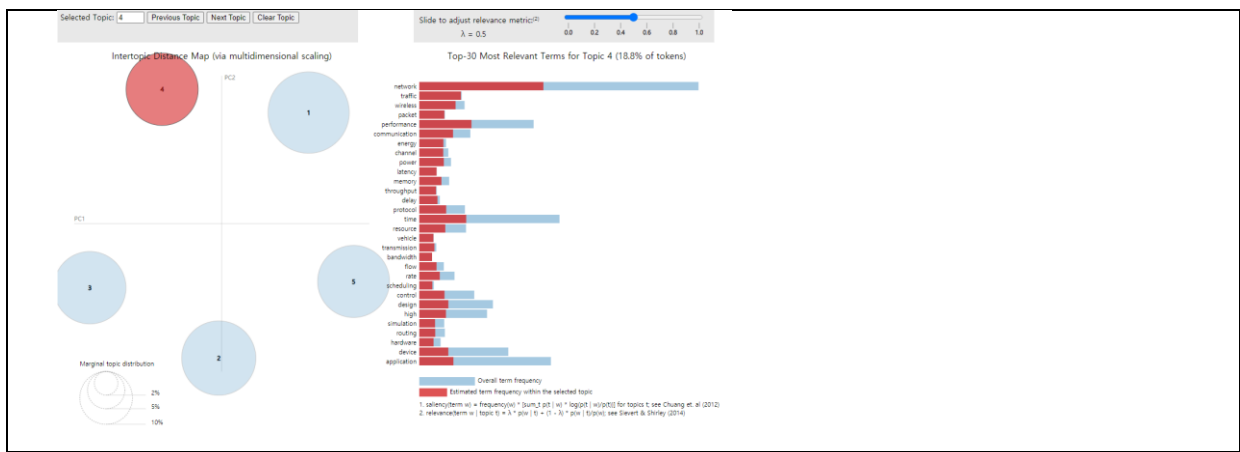


Fig. 32. Topic 4 Selected

## Classification of Texts Using LSTM and LDA

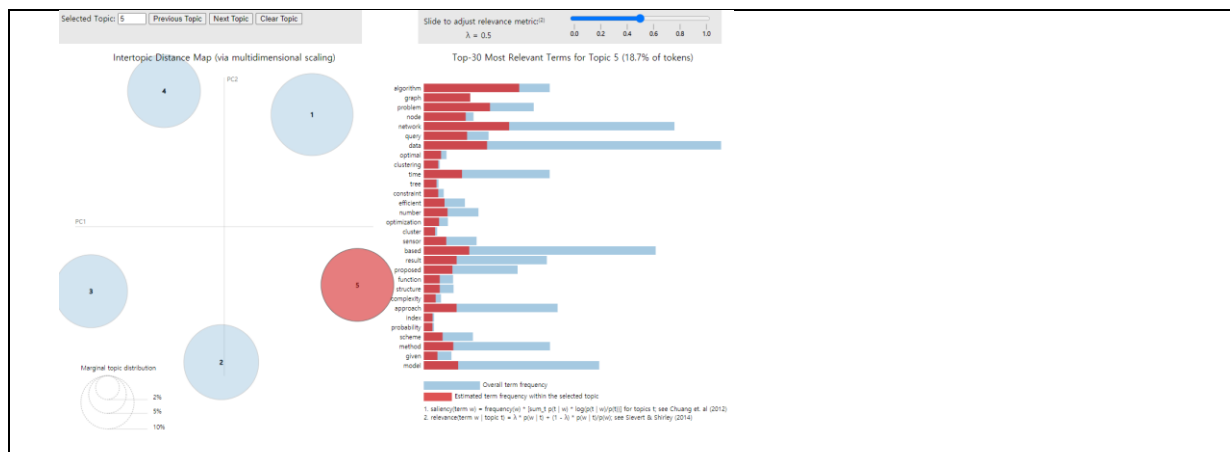


Fig. 33. Topic 5 Selected

The LDA only clusters documents and does not specify what category they are. However, if the clustering is a well-established model, users can fully predict it. Topic 1 uses words such as ‘security’, ‘privacy’, and ‘authentication’, thus one can predict that the category is ‘Security and Privacy’. Topic 2 corresponds to ‘Web, Mobile and Multimedia Technologies’ because words such as ‘social\_medium’, ‘twitter’, and ‘news’ appear. One can know that Topic 3 corresponds to ‘Artificial Intelligence, Machine Learning, Computer Vision, Natural Language Processing’ category through words like ‘image’, ‘classification’ and ‘recognition’. Topic 4 shows ‘Networks and Communications’ since ‘network’, ‘bandwidth’, and ‘wireless’ can be seen. Lastly, with the words ‘graph’, ‘algorithm’ and ‘shortest\_path’, one can see Topic 5 is about ‘Information Systems, Search, Information Retrieval, Database Systems, Data Mining, Data Science’.

### 4. Conclusion

In this study, we compared the performance of LSTM and LDA with natural language processing techniques. LSTM specifies which text and to what extent a text is predicted, but it is challenging and quite difficult to make direct constructions of the model. In contrast, LDA requires users to match clustered text directly to topics, but not much effort is needed when making a model.

### References

- [1] “Intelligent Records and Archives Management That Applies Artificial Intelligence,” Journal of Korean Society of Archives and Records Management, vol. 17, no. 4, pp. 225–250, Nov. 2017.
- [2] Kang Min-young. " Design and Implementation of Personal Talent Assessment System Based on Text Mining" Domestic Master's thesis Gachon University, 2014. Gyeonggi-do
- [3] H. Park and K. Kim, “Sentiment Analysis of Movie Review Using Integrated CNN-LSTM Mode,” Korea Intelligent Information Systems Society, vol. 25, no. 4, pp. 141–154, Dec. 2019.
- [4] Oh Young-taek, Kim Min-tae, and Kim Woo-joo (2019). Korean Movie-review Sentiment Analysis Using Parallel Stacked Bidirectional LSTM Model. Journal of Korean Institute Information Scientists and Engineers, 46(1), 45-49.
- [5] Seo Yang-Mo. “Analysis of Prediction Accuracy of Fine Dust Concentration for Seoul Region Using LSTM Model”, University of Sejong, 2019. Seoul
- [6] S. Ahn, Y. Chung, J. Lee, and J. Yang, “Korean Sentence Generation Using Phoneme-Level LSTM Language Model,” Korea Intelligent Information Systems Society, vol. 23, no. 2, pp. 71–88, Jun. 2017.
- [7] Park Ja-hyun and Song Min. (2013). A Study on the Research Trends in Library & Information Science in Korea using Topic Modeling. Journal of the Korean Society for Information Management, 30(1), 7-32.
- [8] Park Mi-sun. A Study on the Topic Analysis of the 4th Industrial Revolution News Articles using LDA and Word2vec. Seoul National University of Science and Technology, 2018. Seoul
- [9] T. Cho and J.-H. Lee, “Latent Keyphrase Extraction Using LDA Model,” Journal of Korean Institute of Intelligent Systems, vol. 25, no. 2, pp. 180–185, Apr. 2015.