The Magic of Transformation: Modifying Data Format to Improve Evaluation Results

Yabo Gao

School of Professional Studies, Northwestern University

MSDS 458: Artificial Intelligence and Deep Learning

Dr. Syamala Srinivasan

November 30, 2020

Abstract

Today's society demands the following things from consumer products: fast and accurate. In the post-Industrial Revolution world, technology cannot just execute tasks quickly – the factory has to produce it quickly too. Thus, industries must ensure that their products meet those three standards. One impediment to this goal in the Language Processing World is dirty data. Because language itself is a very freeform concept, it does not always display patterns that machines can recognize. This project's primary goal is to examine a version of such dataset – Reuters News, and see if there are any ways to transform this dataset into something that is more machine-friendly. I examined the following types of frequencies in my input and used the results in my decision-making process: category frequency, word frequency, and document length frequency.

Keywords: Data Cleaning, Frequency, One-hot encoding, Outliers, Padding

The Magic of Transformation: Modifying Data Format to Improve Evaluation Results

According to Tang et al. (2005), "Many text mining applications need take emails as input. Email data is usually noisy and thus it is necessary to clean it before mining." Even though there are several products for email cleaning already, none of them can eliminate every noise possible (Tang et al., 2005).

The problem above poses the need for a deep dive into data cleaning for application purposes. Specifically, it poses the need to study how data cleaning impacts Natural Language Processing. In this study, I applied various transformations on the same dataset and ran it on four models I developed in a previous study. After trying out every possibility and combination, I gathered the best model for each subset of transformed data and compared its effectiveness with the untransformed data. This juxtaposition displays the real power of data cleaning in Natural Language Processing.

Literature Review

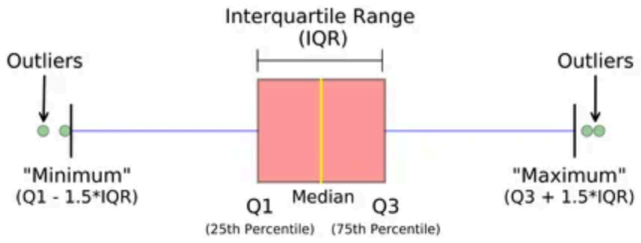
Box and whisker plot

According to McLeod (2019), "a box plot (also known as box and whisker plot) is a type of chart often used in descriptive data analysis to visually show the distribution of numerical data and skewness through displaying the data quartiles (or percentiles) and averages."

As shown in the image below from McLeod (2019), box plots demonstrate the dataset's minimum, first quartile, median, third quarter, maximum, and outliers on both ends.

Figure 1

Box and whisker plot



Embedding

According to Lebret (2016), “Word embedding is a feature learning technique which aims at mapping words from a vocabulary into vectors of real numbers in a low dimensional space. By leveraging large corpora of unlabeled text, such as continuous space representations can be computed for capturing syntactic and semantic information about words.” The images below from Chollet (2018) present a juxtaposition of one-hot encoding and word embedding.

Figure 2

Illustration of word embedding

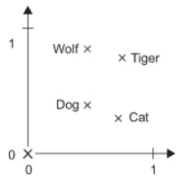
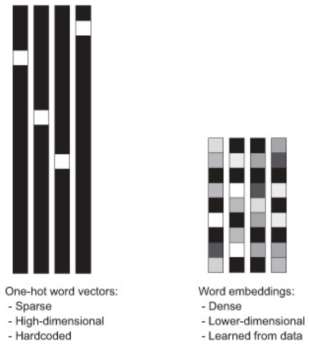


Figure 3

One-hot encoding vs. word embedding



One-hot encoding

According to Chollet (2018), “one-hot encoding is the most common, basic way to turn a token into a vector.” In this method, one assigns a unique integer to every word and turns that integer into a binary vector of the whole vocabulary (total number of unique words). This vector has all zeros except for that particular index (Chollet, 2018). Figure 3 contains a pictorial illustration of this concept. The figure below from Chollet (2018) provides the code for this.

Figure 5

Code for one-hot encoding

```
import numpy as np

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

token_index = {}
for sample in samples:
    for word in sample.split():
        if word not in token_index:
            token_index[word] = len(token_index) + 1

max_length = 10

results = np.zeros(shape=(len(samples),
                          max_length,
                          max(token_index.values()) + 1))
for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
        index = token_index.get(word)
        results[i, j, index] = 1.
```

Methods

I performed the following steps to obtain the results discussed in the next section:

1. Performed exploratory data analysis (EDA) on the dataset with Keras, Matplotlib, and Seaborn. More specifically, I examined the topic, word, and document length frequencies.
2. Performed the first round of data cleaning with Keras and built-in Python functions. This process includes removing all non-alphabetic words, words that appear too frequently based on the result in step 1, and words with length <=2
3. Reconstructed the document length EDA graphs with the data from step 2
4. Used Keras to set the lengths of all arrays based on the results from step 3
5. Ran all models described in the Appendix with Keras and data from step 2-4. I will use these models for the entire study.
6. Graphed the training/validation accuracy/loss for all models in step 3 with Matplotlib
7. Ran the best two models from step 3 using 20 epochs instead of 10
8. Graphed the training/validation accuracy/loss for all models in step 5 with Matplotlib
9. One-hot encoded the data from step 2 using built-in Python libraries and Keras
10. Ran all the models again, using the data from step 7 and scikit-learn's K Fold Cross Validation whenever possible

11. Divided the documents based on the frequency of its category with built python functions and results from step 1
12. Transformed the frequent and infrequent dataset based on the most optimal result from the previous experiments with the Keras and built-in python functions
13. Ran the best three models on the frequent and infrequent dataset
14. Compared the training and execution time for the best models with all dataset, frequent dataset, infrequent dataset, and the untransformed dataset

Results

Exploratory Data Analysis

First, I decided to learn a bit more about the dataset I have and examine patterns that I have not noticed before. I started by taking a look at the frequencies of categories.

Figure 1

Histogram of Categories

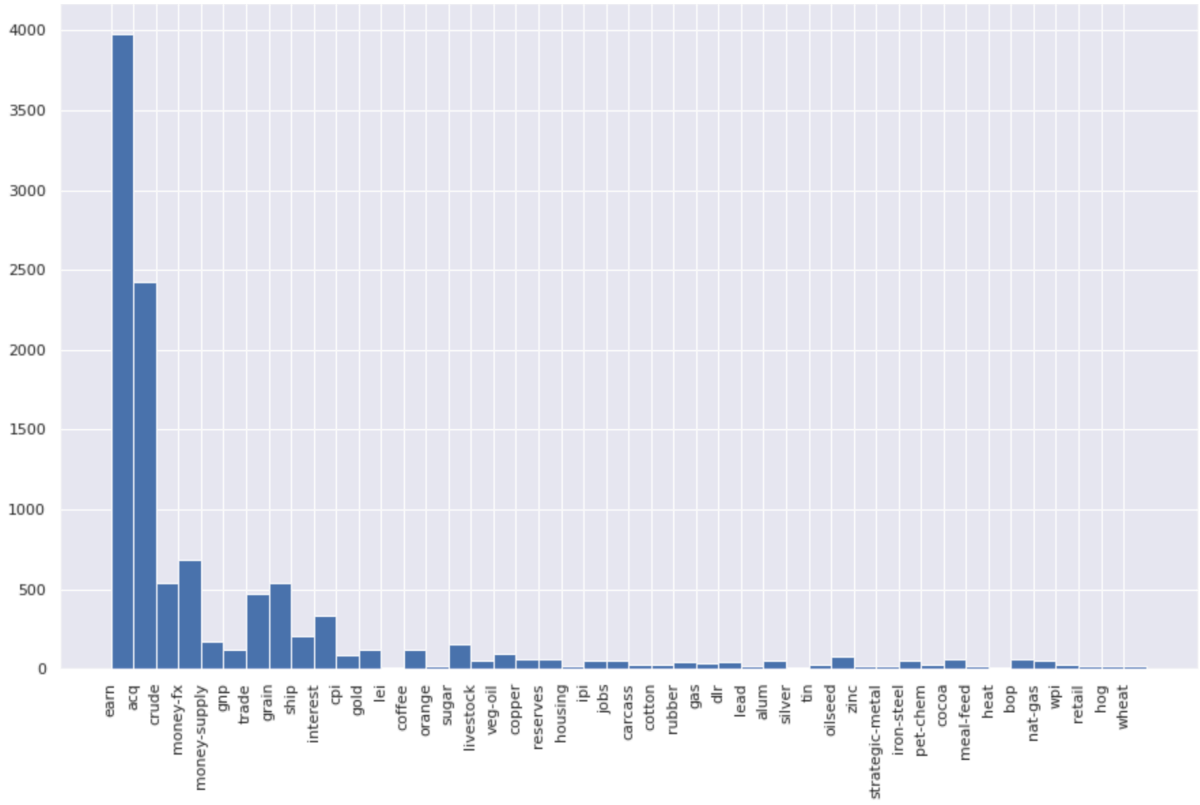
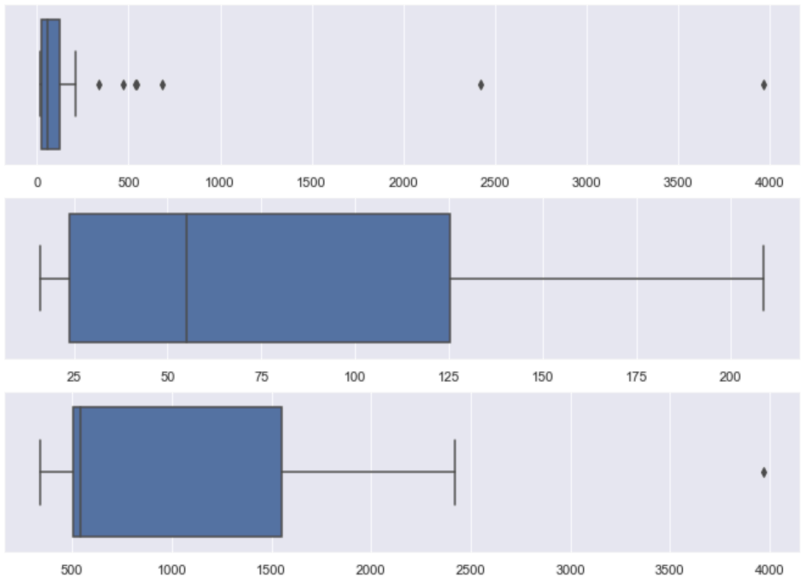


Figure 2

Boxplots of the above diagram. The top is the boxplot of the overall data. The middle is the boxplot for the overall data with outliers removed, and the bottom is the boxplot with outliers.



The two figures above show that most categories appear in less than 225 documents. For those with more than 225 documents, all but one of them has less than 2500 documents.

Figure 3

Histogram for word frequencies

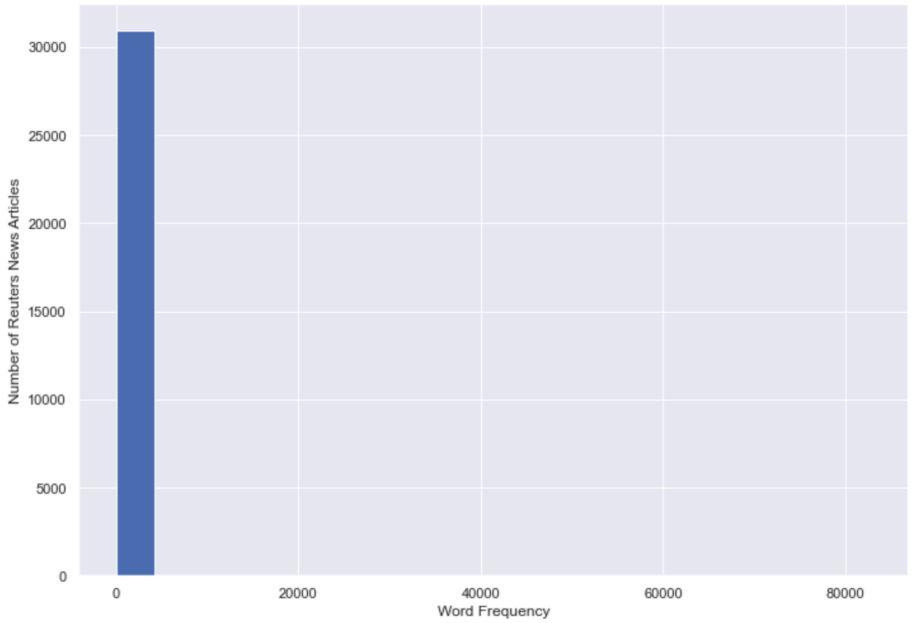
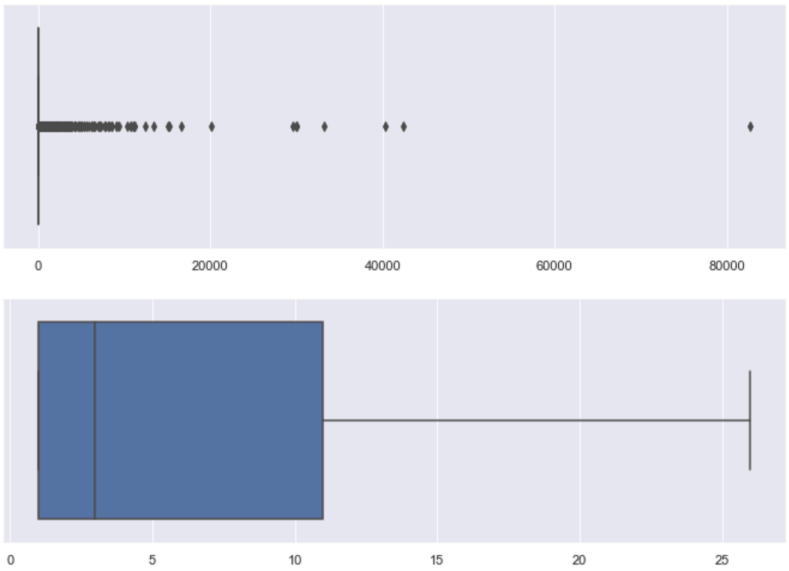


Figure 4

Boxplots of the diagram above. The top is the boxplot of the overall data. The bottom is the boxplot for the overall data with outliers removed.



Most of the words occur less than 30 times. Even though there are words that appear more than 80000 times, there are so few of them that they are not even on the histogram.

Figure 5

Histogram for the number of words in each document

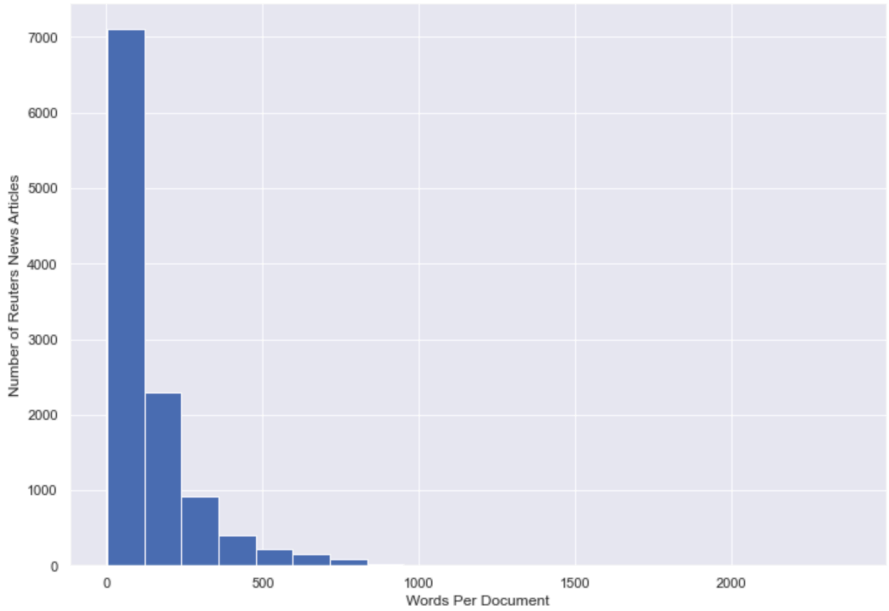
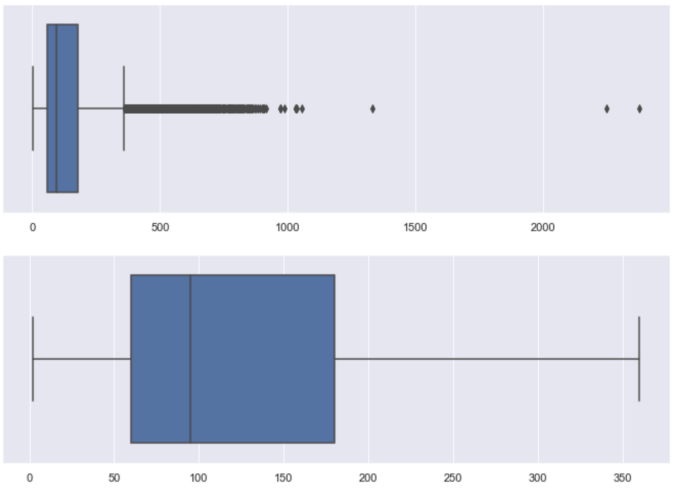


Figure 6

Boxplots of the diagram above. The top is the boxplot of the overall data. The bottom is the boxplot for the overall data with outliers removed.



The two figures above provide useful information on the dataset. It shows that most of the documents have less than 350 words. This result is not the final result, as the numbers will change after I apply the first transformation. However, it shows that I can truncate each document's length without reducing the overall result's accuracy too much.

Figure 7

Histogram for the number of words in each document after the first transformation.

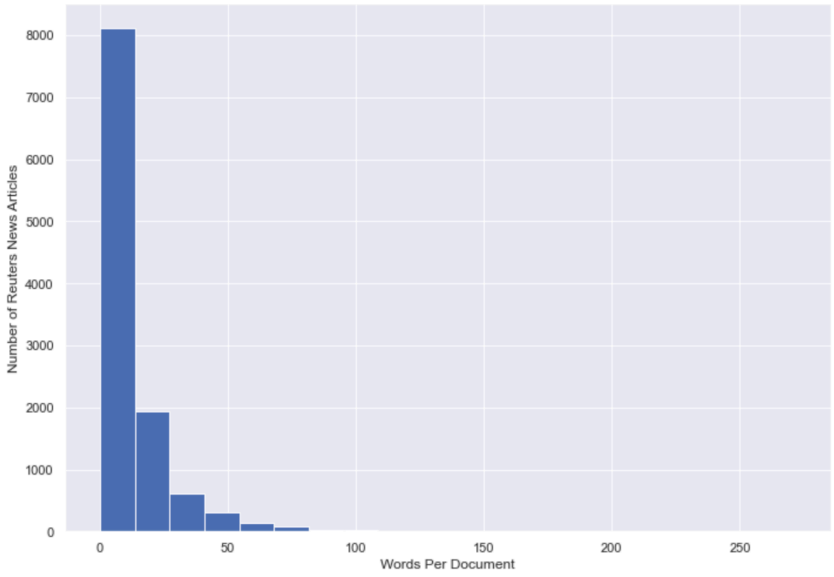
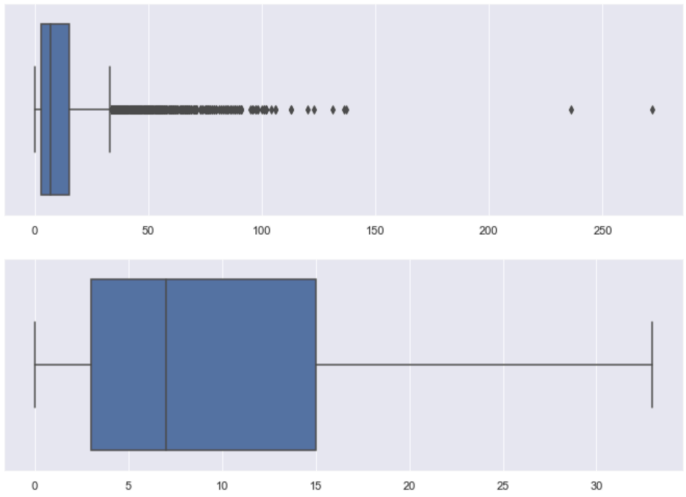


Figure 8

Boxplots of the diagram above. The top is the boxplot of the overall data. The bottom is the boxplot for the overall data with outliers removed.



After I cleaned the dataset according to the previous section's procedure, the document sizes changed considerably – as shown in the two figures above. Since most documents now have less than 35 words, I made all documents to be exactly 35 words. I used the previous section's techniques to pad all the remaining parts of the shorter documents with zeros and truncate the remaining parts of the longer document.

Analyses on Models

Table 1

Results for all models before document cleaning and unification

Model Name	Accuracy	Loss
DNN	0.3820	2.2845
CNN	0.3691	2.2848
RNN	0.3620	3.1900
LSTM	0.6759	1.1407

Table 2

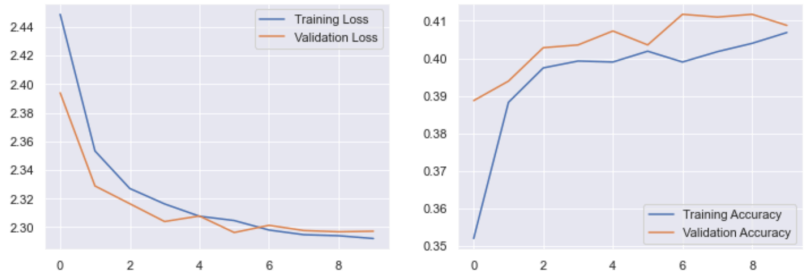
Results for all models after document cleaning and unification on the test dataset

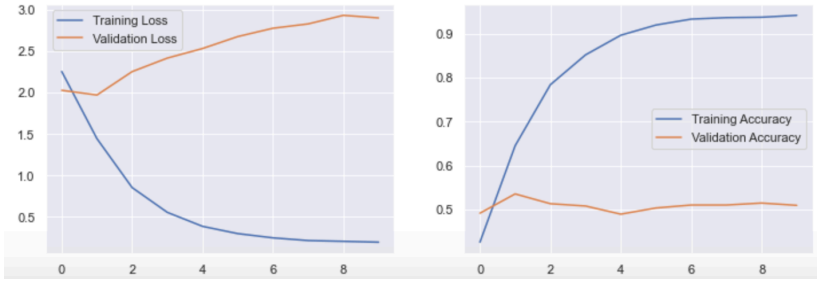
Model Name	Accuracy	Loss
DNN	0.4003	2.2996
CNN	0.4052	2.3005
RNN	0.4706	2.2945
LSTM	0.5223	2.8557

The juxtaposition of the two tables above shows that the cleaning procedure improved all models' accuracy except the LSTM. It also reduces loss for all models except for DNN and LSTM. However, the increase in loss for DNN, from 2.2845 to 2.2996, is minimal compared to that of LSTM – from 1.1407 to 2.8557. RNN experienced the most significant boost in accuracy– from 0.3620 to 0.4706, and reduction for loss--from 3.1900 to 2.2945.

Table 3

Training/Validation Accuracy and loss graph for all models with all of the above transformations

Model Name	Graph
DNN	
CNN	

Model Name	Graph
RNN	
LSTM	

The table above shows that over-fitting is not so much of a problem in CNN and DNN models. However, it is a noticeable problem in the RNN and LSTM models. For those models, the training set's accuracy is much higher than that of the validation set; the training set's loss is much lower than that of the validation set. However, this trend is not too unexpected. Most topics appear in less than 225 documents. Therefore, there may be categories that appear in the test set but not in the training set.

To further improve accuracy and reduce the loss of RNN and LSTM models, I increased the number of epochs for each training model. The tables below show that doing so increased accuracy by a bit for both models. It also reduced loss for the RNN model.

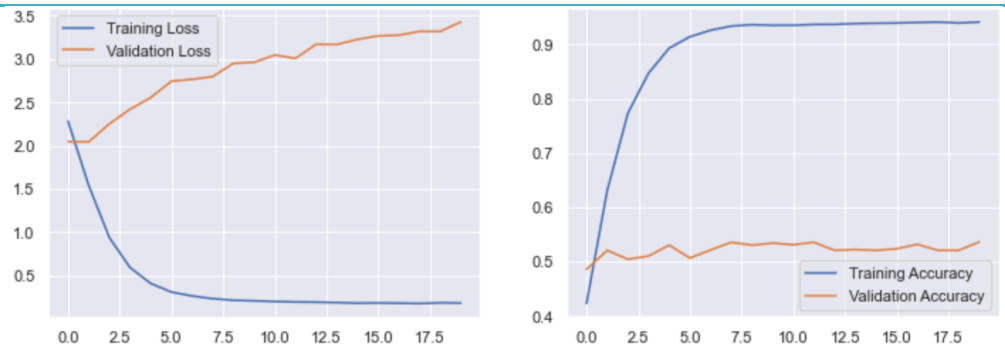
Table 4

Accuracy and Loss for RNN and LSTM models with the 20 epochs on the test dataset

Model Name	Accuracy	Loss
RNN	0.4907	2.2441
LSTM	0.5285	3.2249

Table 5

Training/Validation Accuracy and loss graph for the above models with 20 epochs

Model Name	Graph
RNN	
LSTM	

Because increasing the number of epochs is not a feasible solution for the long hall, I sought ways to transform the dataset further to increase accuracy for all models. Since the testing set can be different from the training dataset, I used the K Fold Cross-Validation for the remaining of this experiment to have a holistic understanding of the model's effectiveness.

I one-hot encoded my transformed dataset using the steps described in the previous section. After modifying the code to accommodate K Fold Cross-Validation and the new dataset, I realized that my computer does not have the hardware capacity to run the RNN and LSTM with the one-hot encoded data. Furthermore, K Fold Cross Validation cannot work on the 3D data structure for the CNN model. Thus, the tables below contains results for DNN with K Fold

Cross-Validation and one-hot encoded data, CNN with reshaped one-hot encoded data, and RNN/CNN with K Fold Cross-Validation and the cleaned dataset from the previous experiments.

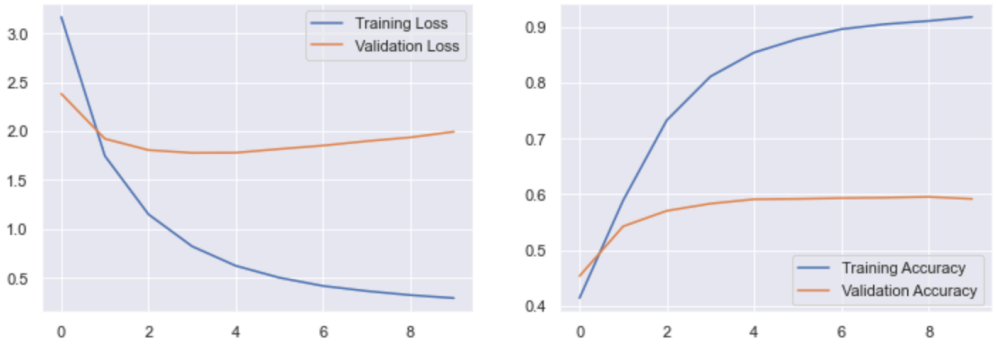
Table 6

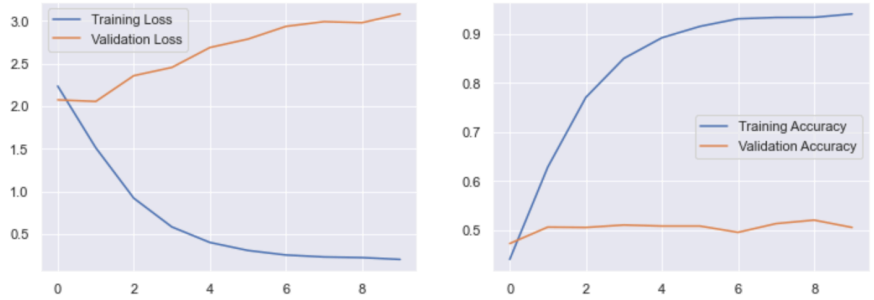
Accuracy for all models with one-hot encoded data and K Fold Cross-Validation

Model Name	DNN	CNN	RNN	LSTM
Accuracy	[59.72222089767456, 59.775638580322266, 61.41101121902466, 59.91448163986206, 60.82308888435364, 59.54035520553589]	0.3433	[42.68162250518799, 44.07051205635071, 46.87333106994629, 41.52859449386597, 44.25441026687622, 39.28380608558655]	[82.42521286010742, 82.47863054275513, 83.43132138252258, 82.09513425827026, 50.98877549171448, 39.28380608558655]

Table 7

Training/Validation Accuracy and loss graph for selected K Fold iterations for models above

Model Name	Graph
DNN	
CNN	

Model Name	Graph
RNN	
LSTM	

As the tables above show, one-hot encoding data significantly improves the accuracy of the DNN models. K Fold shows a bit of potential for the RNN model and a lot of potential for the LSTM model. However, the CNN model performs much better on the cleaned dataset without this transformation. The training/validation accuracy/loss looks very much alike across all iterations. As I said earlier, it is not surprising that the training accuracy is much higher than that of testing accuracy. After all, most topics appear in less than 225 documents. Therefore, there may be categories that appear in the test set but not in the training set.

To even out the distribution of data among the training and testing set, I divided the dataset based on the frequency of the document's topic. I put the documents whose topic appears more than 225 times into one category and the others into another category. Instead of plotting the training/validation accuracy/loss, I created confusion matrices to compare the training dataset's accuracy on the most frequent documents. These matrices allow me to see if

there are any topics that are not in the training set. Because the CNN network's performance has not improved much throughout the study, I will discontinue it for the study's remainder.

Table 8

Accuracy for all models with K Fold Cross-Validation on the frequent data. Data transformation is the same as the previous experiment.

Model Name	DNN	RNN	LSTM
Accuracy	[88.80688548088074, 87.33087182044983, 90.03689885139465, 88.19188475608826, 89.66789841651917, 89.65517282485962]	[82.77983069419861, 84.7478449344635, 83.0258309841156, 84.87085103988647, 87.20787167549133, 83.86699557304382]	[94.46494579315186, 93.8499391078949, 96.18695974349976, 94.46494579315186, 86.83887124061584, 83.86699557304382]

As the table above shows, the accuracy improved significantly for all models. In fact, for all models, the testing accuracy has been in the 80% and 90% range. LSTM provides the most promising result as four out of the six iterations resulted in more than 90% accuracy. However, the DNN provides the most stable results as it has the least range among all of them.

Figure 9

Confusion matrix for the first K Fold iteration of the DNN model

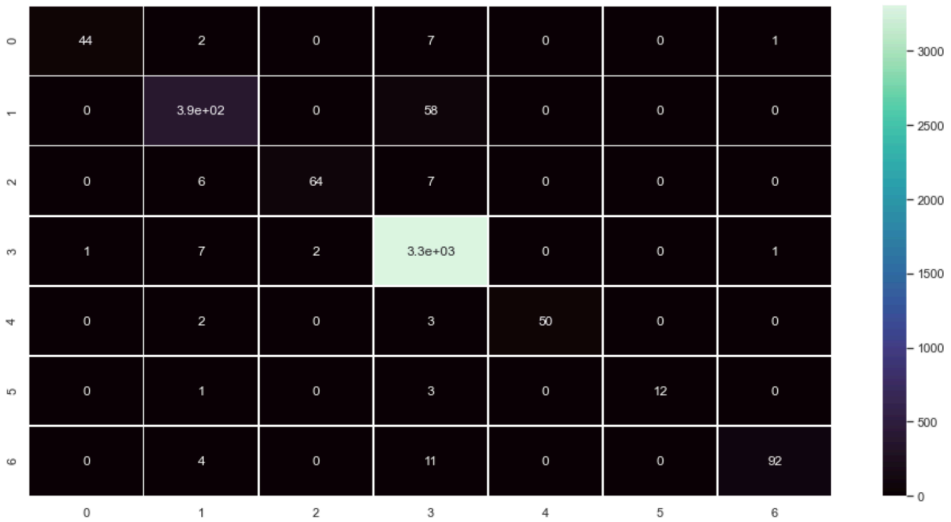


Figure 10

Confusion matrix for the second K Fold iteration of the RNN model

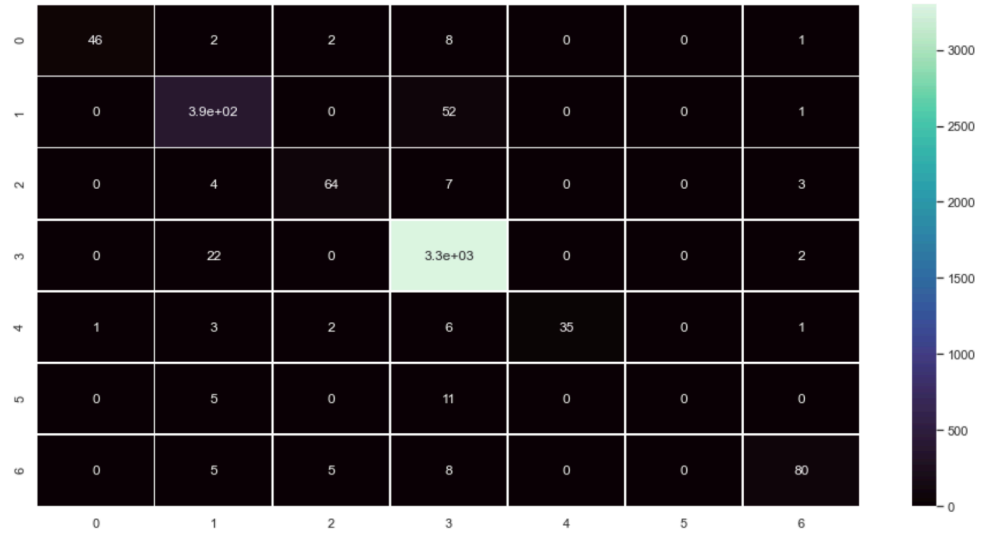
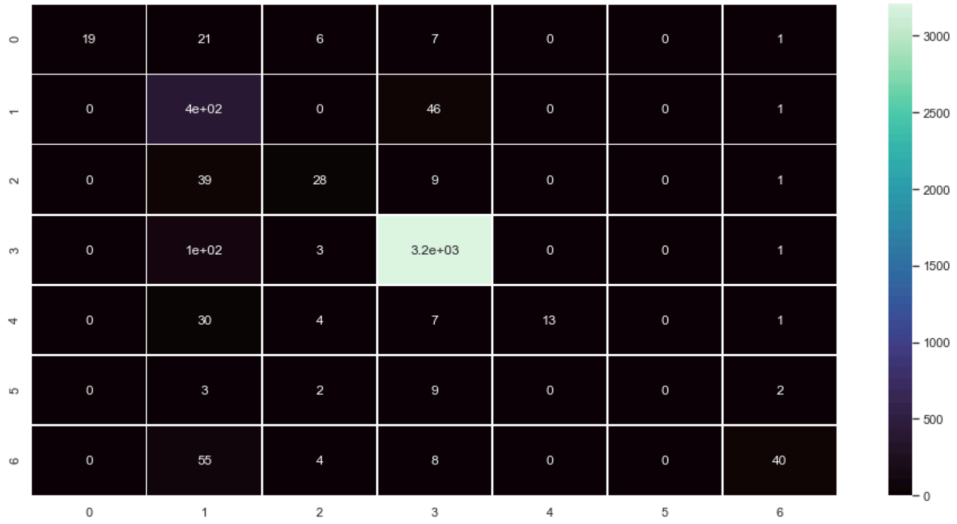


Figure 11

Confusion matrix for the first K Fold iteration of the LSTM model



The high numbers along the diagonal of the three confusion matrices above indicate that the model classified most items correctly. It is a good sign that the LSTM model has more non-zero grids on both sides of the diagonal even though it has the best accuracy on the test dataset. It shows that it might not have a strong over-fitting problem. The distribution of wrongly

classified items seems to be relatively similar between the other models. Furthermore, LSTM and RNN models did not pick up any items from one category in all K Fold training iterations. This can be a concern when the model encounters such data in the testing set.

Table 9

Accuracy for all models with K Fold Cross-Validation on the infrequent data. Data transformation is the same as the previous experiment.

Model Name	DNN	RNN	LSTM
Accuracy	[62.60623335838318, 59.112370014190674, 61.09537482261658, 60.01890301704407, 58.41209888458252, 55.860114097595215]	[44.75920796394348, 44.098204374313354, 28.5174697637558, 22.58979231119156, 40.73724150657654, 26.654064655303955]	[76.01510882377625, 75.1652479171753, 73.46553206443787, 75.99244117736816, 40.92627465724945, 26.654064655303955]

Even though the result for DNN remained relatively stable, the RNN and LSTM model's accuracy values have fluctuated with different train/test datasets. This trend shows that DNN is the best option for the infrequent categories.

Table 10

Training/Running time for the control models and its best counterparts after data cleaning

Model Name	Training Time	Execution Time
DNN - control	0s	0s
LSTM - control	206s	2s
DNN - altogether	11s	0s
LSTM - altogether	70s	0s
DNN – most frequent	10s	0s
LSTM – most frequent	30s	0s
DNN – least frequent	10s	0s
LSTM – least frequent	40s	0s

Since the most accurate model is between DNN and LSTM after data cleaning, I examined the their training/execution time before and after data cleaning. As the table above shows, it is crucial to perform data transformation before training a model with LSTM. It significantly reduces both training and execution for this model. Even though the training time increased by 10 seconds with the one-hot encoded and transformed data, it is still much faster than the improved version of LSTM. Thus, I can say that DNN is still the most touted model for faster production time and better stability.

Conclusion

This study demonstrates how data cleaning and transformation are crucial for machine learning. Removing the most frequent words, non-alphabetic words, and words with length less than or equals two improves the DNN, CNN, and RNN models' accuracy. Increasing the number of epochs further improved the result of the RNN model. By one-hot encoding the input for the DNN model, the accuracy went up by about 20%. K Fold shows how the hidden potential/instability of the LSTM model and the relative stability of the DNN model. RNN, DNN, and LSTM provide a promising result for the frequent categories, with DNN and LSTM being the best options. It also comes down to those two for the least frequent categories. However, DNN is the most recommended model as the LSTM model tends to fluctuate.

References

Chollet François. (2018). *Deep learning with Python*. Manning Publications Co.

Lebret Rémi Philippe. (2016). *Word Embeddings for Natural Language Processing*. Ecole Polytechnique Fédérale de Lausanne.

Mcleod, S. (2019, July 19). *Box plots (also known as box and whisker plots)*. What does a box plot tell you? https://www.simplypsychology.org/boxplots.html.

Tang, J., Li, H., Cao, Y., & Tang, Z. (2005). Email data cleaning. *Proceeding of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining - KDD '05*, 489–498. https://doi.org/10.1145/1081870.1081926

Appendix

Table 11

RNN, LSTM, DNN, CNN models description

Model Name	Description
RNN	• Embedding with input dimension of 30980, output dimension of 128, and input length of 1032 • Simple RNN with 256 nodes • DNN layer with 46 output nodes and softmax activation function
LSTM	• Embedding with input dimension of 30980, output dimension of 128, and input length of 1032 • LSTM with 128 nodes • Dropout layer with 20% nodes dropped out • DNN layer with 46 output nodes and softmax activation function
DNN	• Hidden DNN layer with 64 nodes in that layer • DNN layer with 46 output nodes and softmax activation function
CNN	• Convolution with 32 filters using Kernels of size 3 and stride of 1 • Max pooling taken from subsets of 2 • Convolution with 64 filters using Kernels of size 3 and stride of 1 • Max pooling taken from subsets of 2 • Flattened the data • Hidden DNN layer with 1024 nodes in the layer • Dropout layer with 20% nodes dropped out • DNN layer with 46 output nodes and softmax activation function