CSCI 315 - RNN Text Analysis

Professor Cody Watson
Fady Gouda and Abhi Jha

Introduction

For Assignment 4, we performed the extensive preprocessing steps that are required before we can feed data into a recurrent neural network model. In order for the model to understand textual data, we need to ensure that it is optimized for the model. For this assignment, we use the consumer_complaint.csv file to preprocess textual data. The dataset contains complaints that actual people have expressed for products that they've bought and used before. The dataset also contains information on the products that the consumers were talking about. The preprocessing of textual data, which in this case is consumer complaints, includes removing null values, special characters, tokenizing and more - we will get into the details. We had to utilize libraries such as numpy, pandas, re, nltk, keras, and tensorflow in order to implement our strategy.

Elements of Learning

Input

The input in this case is the 'consumer_complaint_narrative' column from the 'consumer_complaints.csv' file. There are about 555957 lines in the complaint column. We also have a product column, which corresponds to the product that the consumers are referring to. The complaints are the regular consumer complaints that talk about the product and the user's experience with it.

Output

Since we are only dealing with the preprocessing of data and we have not yet created a model to analyze the data, we don't have an output. In the future, we could use the preprocessed data to create an output, such as the sentiment. We could also set the product as an output to try to predict the product type based on the complaints.

Target Function

Again, since we don't have a model and clear idea of how we are going to use the model, they don't yet know our target function. The target function is normally f: X -> Y. While we know that X is the consumer complaints, we have not determined our Y, and therefore, we do not yet have the target function.

Data

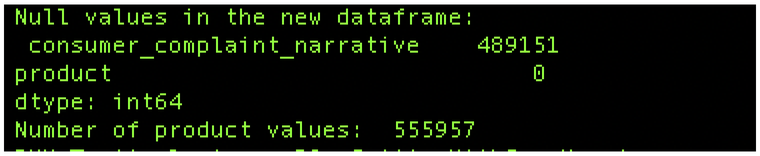
The consumer complaints data set came with 555957 lines of data. Currently, for the preprocessing step, we are using two columns from the data set: product and consumer complaints narrative.

Hypothesis
Since we don’t know the target function, it wouldn’t make sense for us to determine the hypothesis, just because we don’t know which function from our hypothesis set would best approximate the target function, because we don’t know the target function to begin with.

Code Functionality

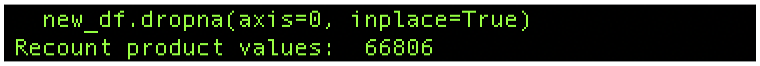
Our codebase can be divided into three parts: retrieving the data, cleaning the data, implementing the tokenizer. We will delve into each part in detail.

Retrieving the Data
We use pandas to import the consumer_complaints.csv dataset. The dataset is stored in a dataframe. For the preprocessing, we only want to use the product and consumer complaint narrative, and so we slice the dataframe to have only those two columns and store it in a new dataframe. Before we can actually clean the data, we want to ensure that there are no null values in the dataframe. So, we first count the null values in the dataframe using the isnull() method. We also count the number of products that exist using the count() method. We then drop the null values from the new dataframe using dropna() method. We make sure the dropping is done row wise by setting the parameter axis=0 and to retrieve the position the existing values have in the dataframe, we set the parameter inplace = True. After we have dropped the null values, we want to reset the index, and we do so by using the reset_index() method. Finally, we recount the number of products in the dataframe. Here are the results we achieved so far:



```
Null values in the new dataframe:
 consumer_complaint_narrative    489151
product                             0
dtype: int64
Number of product values:  555957
```

There were 489,151 null values in the conumer_complaint_narrative column and 0 null values in the product column. The number of product values before cleaning was 555,957.



```
  new_df.dropna(axis=0, inplace=True)
Recount product values:  66806
```

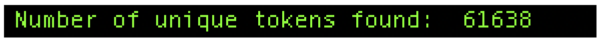
After dropping the null values and recounting the number of product values, we get 66,806. So, essentially, we can only use 66,806 lines from the dataset.

Cleaning the Data
To clean the text, we have the clean_text function defined. In the clean_text, we pass the slice of the dataframe that contains the consumer complaints narrative column. First, we lower the values using str.lower() method. Then, we compile all the special characters in a string and use the string and the replace() method to replace the characters with empty space. Furthermore,

we removed the ‘x’ characters with empty space too. The next step is to remove the stop words. Stop words are the words in english like he, she, myself that occur very commonly and don’t necessarily add meaning to the sentence. In order to do so, we split the text and loop through every complaint. For each complaint, we loop through the word that has been splitted, check if it’s a word and if it exists in the list nltk.stopwords, then we remove it from the dataset. Lastly, we join the splitted text and return the cleaned data.

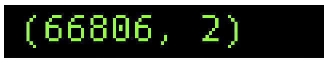
Tokenizer

The tokenizer is used to vectorize a text, and encode each word into an integer, and the entire text as a list of integers. We use the tokenizer function from the keras library and we pass to it the maximum number of words (vocabulary), the filters, which are all the characters to be removed from the text (such as all punctuation). We then fit the complaint data to the tokenizer, giving us all the different words used in the data set. Furthermore, we use the texts_to_sequence function to get the sequence representation of texts, as a list of integers that represent each word. As a test, we use the sequence_to_text function to basically reverse the last step to make sure our preprocessing has worked and the data is filtered and tokenized for learning.
Our model found the number of unique tokens to be 61,638, as shown below:



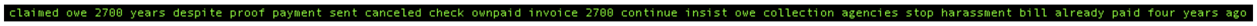
```
Number of unique tokens found:  61638
```

The shape of the data was (66806, 2) as shown below:



```
(66806, 2)
```

Furthermore, here is the first example of the tokenizer object to the sequences to text:



```
claimed owe 2700 years despite proof payment sent canceled check ownpaid invoice 2700 continue insist owe collection agencies stop harassment bill already paid four years ago
```

Output: “claimed owe 2700 years despite proof payment sent canceled check ownpaid invoice 2700 continue insist owe collection agencies stop harassment bill already paid four years ago”

As we can see, the special characters are removed, along with the stopwords and the x’s.