Vectorizing a Dataset of CDC Novel Coronavirus Reports

Robin Singh

Northwestern University

Abstract:

This is an extension of putting the CDC Novel Coronavirus Reports previously scraped into a .json file to use. Our objective is at this point in time, vectorization of the dataset in order to prepare the data to be fit into analytical models to be analyzed and categorized. We will be conducting three approaches for our model in order to compare the performance of the models on this dataset: analyst judgement, TF-IDF, and Neural Network approaches. With this we will see which vectorization method produces the best Random Forest score of categorizing the documents into two equivalence classes: early reports and recent reports.

Introduction:

This research was conducted in order to find the best way our CDC Novel Coronavirus Reports dataset, consisting of all of the reports scraped from the CDC website, could be vectorized in order to categorize them into equivalence classes in the end. This way, we can further determine which method performs the best in terms of score to analyze large document databases and why this may occur. The two equivalence classes this will split into would be early on reports and recent reports. This could be used to determine which information would be more applicable to today's day and age, as the structure and spread of the COVID-19 virus has been changing over time, with more variants of the virus coming up and different communities being affected at different times. It is important to distinguish between older reports and recent reports to determine whether or not the document is still relevant or not.

Literature Review:

A corpus regarding COVID-19 was also created by Erdal Baran and Dimitar Dimitrov. This corpus was called TweetsCOV19, a corpus consisting of Tweets conducted surrounding the COVID-19 pandemic (compared to my corpus which text data from CDC Novel Coronavirus

Reports). This corpus utilizes Tweets from the time period between October 2019 through April 2020. The corpus aimed at capturing online discourse about various aspects of the pandemic and its societal impact. However, this corpus was quite extensive, containing the Username, Tweet ID, Timestamp, the text within the Tweet, and hashtags. All-in-all, the dataset contains 8,151,524 tweets posted by 3,664,518 users.

This corpus itself was a subset from an existing public corpus, TweetsKB, which is an anonymized data for a large collection of annotated tweets. This corpus contains more than 7 billion tweets beginning from February of 2013. The way it was used to create the TweetsCOV19 corpus was by compiling a seed of 268 COVID-19-related keywords and filtering tweets from TweetsKB to get the ones relevant to COVID-19.

Methods:

To create the CDC Novel Coronavirus Reports corpus, I utilized Beautiful Soup, which is Python package for parsing HTML and XML documents. Beautiful Soup parses through websites and scrapes the information you dictate as relevant for your use. I started a Beautiful Soup project within Google Colab, which I used as my coding environment.

After scraping the websites from the CDC Novel Coronavirus Reports website, I conducted preprocessing to clean up the data using the NLTK library. This included removing punctuation from the data, or unnecessary characters that may be added to the text but brings no extra meaning when conducting such analysis. I was also able to remove stop words which add no additional meaning to the dataset. Afterwards, I had tokenized the data, separating the words from each other. This allowed me to stem the words using the Porter Stemmer from the NLTK library. This allowed me to simplify the words and group together similar words, such as simplifying defined to define and so on. This could be done by removing suffixes and/or prefixes

form the data. Lastly, as part of my preprocessing, I had removed numerical data from the text section so it may focus only on the words and not be thrown off by numerical characters, especially since we are characterizing on a time basis. This may also remove any advantages for the model when categorizing on time.

After the preprocessing was complete, we were able to vectorize our text data. Using the SKLearn library, I was able to import libraries that may vectorize for me. In the essence of time, due to my documents being so large (upwards of 3,000 words in some cases) I used the count vectorizer to count the occurrences of words in the dataset and assign them values depending on how frequently they occur in the data. Afterwards, I vectorized using the TF-IDF method and finally I used Doc2Vec for the neural network approach.

After all, three methods were complete, I separated their corresponding matrixes into data frames to use in the Random Forest classification model. I split them up into training and test sets, with 80% of the data in the training set and 20% in the test set. I also categorized the first 70 entries into recent documents and the rest into old documents out of the 258 documents total.

Results:

The results came to a surprise. I expected the Count Vectorizer and TF-IDF vectorizer to perform similarly, as they were using relatively similar methods. However, TF-IDF seems to be a lot more complicated, and those complications improved the result of the model when it came to categorization. When it came to the amount of zeros in the vectors, the number was the same with the percentage of zeros being 0.034, or 3.4% for both. The numbers also had similar weights, as a more frequently occurring number, such as “covid” had a 8498 value in the count vector and a 28.278477738532647 value in the TF-IDF vector, which was the highest in the test comparing several words values to each other. The more frequently it occurred, the higher the

value in both vectors. However, when it came to categorization performance, the Count Vector had a score of 0.7884615384615384, compared to the TF-IDF score of 0.9230769230769231. The TF-IDF performed significantly better, even compared to the Neural Network vector which assigned vector of a few numbers to the document as a whole instead of assigning a value to each word individually. The Doc2Vec vector yielded a score of 0.75, which was the poorest of the three models.

Conclusions:

From our experiment, we have figured out that the TF-IDF performed the best in our Random Forests analysis of the CDC Novel Coronavirus Reports corpus in terms of categorizing the documents between the most recent documents and old documents. It was followed by the Count vector in terms of performance and finally the Doc2Vec Neural Network approach or vectorization coming in last. This could be due to overgeneralization of such a large document in the Doc2Vec algorithm, as it may not count for a lot of the intricate details within the dataset, when it is simplified into such a small vector. The TF-IDF counts for both frequency in the document and frequency in the set of documents very well and creates a more precise way of optimizing the text data.

Appendix:

Sum of Values of words in Count Vector:

```
[ ] sum(df2['covid'])
    8498

[ ] sum(df2['california'])
    409

[ ] sum(df2['diseas'])
    2158

[ ] sum(df2['safe'])
    104

[ ] sum(df2['vaccin'])
    1849

    sum(df2['viral'])
    185

[ ] sum(df2['case'])
    3965

[ ] sum(df2['death'])
    1348
```

Sum of values in TF-IDF:

```
[ ] sum(df3['covid'])
    28.278477738532647

[ ] sum(df3['california'])
    2.5510964721845104

[ ] sum(df3['diseas'])
    7.191937468476215

[ ] sum(df3['safe'])
    0.8149711831734062

    sum(df3['vaccin'])
    12.184936861714164

[ ] sum(df3['viral'])
    1.3655553006971262

[ ] sum(df3['case'])
    15.63043815973914

[ ] sum(df3['death'])
    7.952874703212766
```

Score of Count Vector:

```
[ ] rfcmodel.score(x_test,y_test)
    0.7884615384615384
```

Score of TF-IDF Vector:

```
    rfcmodel.score(x_test,y_test)
    0.9230769230769231
```

Score of Neural Network Vector:

```
[ ]  rfcmodel.score(x_test,y_test)

     0.75
```