Kathryn Egan & James Samford

Syracuse university

December 10, 2020

Classification of Anti-Vaccine Articles

ist 664, Natural Language Processing

Table of Contents

Introduction 2

Research Question 2

NLP Investigation:

Data Preparation 2

Data Pre-Processing 3

Analysis at the Semantic Level 4

Analysis at the Syntactic Level 5

Analysis at the Lexical Level 6

Watson Analysis 8

Classification Experiment 9

Discussion and Future Applications 11

References 12

Appendix of Code 13

**Introduction**

In a year marred by easily shared misinformation regarding COVID-19 and the vaccine that is expected to usher our exodus from the pandemic, it is more imperative than ever before to quickly identify and label misinformation before it has time to take hold and be mistaken as truth. Much of the misinformation shared regarding a COVID- 19 vaccine contains rhetoric from the Anti-Vaccine movement that has been gaining ground in the last few decades. The origin of the newest movement is not surprising – misinformation passed off as scientific research. In 1998 a researcher in the UK published findings that linked a common childhood vaccine with rising diagnoses of Autism in children. Though the author later confessed to falsifying his findings, the damage had been done (Omer, 2020).

His blatant lie, published as truth, grew its own legs and has run wild in the minds of fearful parents and conspiratorialists. Today, views regarding vaccine safety and legitimacy is polarizing; it comes down to AntiVaxxers vs. ProVaxxers (Northwestern University, 2020).

While there are misinformation filters being used by social media platforms like Twitter and Facebook to notify people that what they are reading has been flagged as False, the vast majority of articles, blog posts, and YouTube transcripts are readily available outside of social media’s domain(Shu & Shieber, 2020). For the people who make it beyond the reach of these misinformation filters online, how do they know that what they are reading is truth? Is there blind faith that if the author of an article refers to themselves as a doctor, it must be scientifically sound information that can be trusted? Perhaps there is a sentiment that pulls the reader in and builds their trust? Before training models for predicting if an article is Anti-Vaccine or not, a thorough analysis of the similarities and differences between Anti-Vaccine and Pro-Vaccine articles at various levels of the Synchronic Model of Linguistics, needs to be performed.

**Research Question**

Given how polarizing stances on vaccines are in our society, is polarity analysis of vaccine articles the strongest indicator of an article’s vaccine position? Is there a compelling enough partisanship between the two vaccine camps to make subjectivity analysis a greater predictor than polarity analysis? If, these do not appear to be strong predictors, will machine learning models and deep learning models have more success in their predictions?

**Data Preparation**

All articles were taken from websites and compiled in a csv file. Given the time constraints of the course, the corpus of Anti-Vaccine and Pro-Vaccine texts was limited, which only allows for a peak at sentiment differences between texts, but a much larger corpus would provide a stronger statistical analysis of the sentiments.

Five articles were converted to one data frame, using pandas, for Anti-vaccine texts. Then the same steps were taken for Pro-vaccine texts.

Data utilized in the tensorflow experiment was derived both from previously utilized datasets from prior experiments and additional research in order to create balanced models. This data was broken into raw text files stored on disk with a rating of either 0 for pro-vaccine and medical articles, or 1 for antivaccine articles and blog posts.

**Data Preprocessing**

To begin preprocessing the texts, a function was defined to clean the data by converting all text to lower-case, removing special characters and numbers. Next, English stop words were removed using NLTK’s stop words list. The cleaned text was then tokenized to words, followed by lemmatized to their roots, and finally tagged for part of speech. The output of each preprocessing step was saved in the data frame as new columns for ease of access for later analysis.

First rows of the cleaned Anti-Vaccine data frame:

Table

Description automatically generated

First rows of the cleaned Pro-Vaccine data frame:

Graphical user interface

Description automatically generated

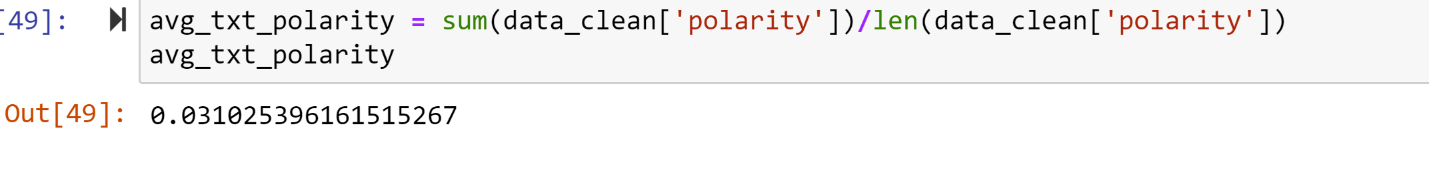
Data for Tensorflow was processed using the Keras library, which was configured to vectorize entire documents for analysis, with any unusual formatting and all punctuation removed, as well as any remnant html excluded.

**Semantic Analysis: Polarity & Subjectivity of Sentences in the Text**

The cleaned sentences were analyzed for polarity and subjectivity scores using the TextBlob library in Python. The scores were added up and then divided by their length to get the average scores for the sentences in each set.

Polarity is a measure of how positive or negative a sentiment is. The polarity scale runs from -1 being most negative to 1, most positive. As the polarity scores demonstrate below, the Anti-Vaccine sentences were more negative in sentiment than Pro-Vaccine sentences. Surprisingly, the difference is not very profound, but this is most likely due to the small corpus and with more documents added to the corpus, a stronger indication of the polarity differences should be evident

Anti-Vaccine Average Polarity:



Pro-Vaccine Average Polarity:



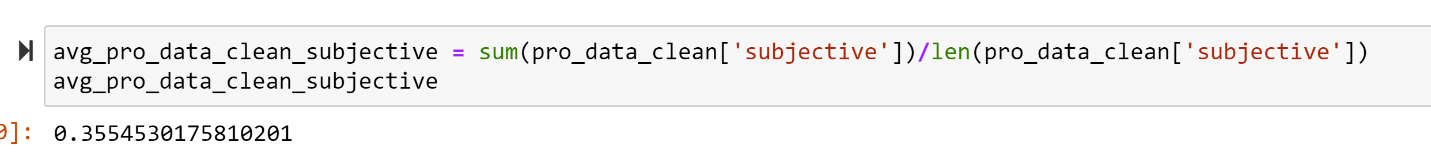
Subjectivity is a measure of prejudice, or lack of objectivity. The Subjectivity scale runs from most objective being 0 and most subjective being 1. There is a minute difference between the subjectivity of sentences from the two stances. Again, with more documents in the corpus, this difference may grow and become more obvious.

Anti-Vaccine Average Subjectivity:

A picture containing graphical user interface

Description automatically generated

Pro-Vaccine Average Subjectivity:



**Syntactic Analysis: Sentiment Analysis of N-Grams**

The cleaned sentences were next broken down to Bigrams and Trigrams. A matrix with frequency of each n-gram was made. Next, the same polarity and subjectivity analysis was done on the bigrams and trigrams as was on the sentences in the previous part.

Anti-Vaccine Most Frequent N-Grams Pro-Vaccine Most Frequent N-Grams

Table

Description automatically generatedTable

Description automatically generated

Looking at polarity of two 2 corpuses at the syntactic level appears to show a much more significant difference in polarity than it showed at the semantic level. Here Anti-Vaccine word combinations are clearly far more negative than Pro-Vaccine word combinations. This is not to imply that the Pro-Vaccine texts are positive, as they are also closer to -1 than 1 on the scale.

Average Anti-Vaccine N-Gram Polarity:

Graphical user interface

Description automatically generated

Average Pro-Vaccine N-Gram Polarity:

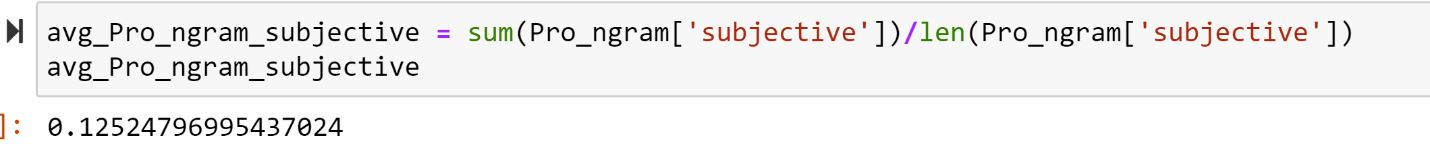


Just as with subjectivity analysis of the sentences in the corpuses, the results of the n-gram subjectivity analysis is not significantly different between the stances. They are both more objective than they are subjective.

Average Anti-Vaccine N-Gram Subjectivity: Graphical user interface, text, application, Word

Description automatically generated

Average Pro-Vaccine N-Gram Subjectivity:



**Lexical Analysis: Frequency Distribution of Words**

While examining word frequencies does not give insight into the importance of the words used in a text, it is a useful tool when comparing documents. For this part of the analysis, the lemmatized words created during the preprocessing phase were used. The NLTK FreqDist library and NLTK Brown Corpus were employed. The most common words in our corpuses were compared against the Brown corpus of most frequent words and the top 100 most common words that matched between our corpus and Brown corpus were removed, like stop words which would create a lot of noise in the data. Next, the frequency distribution was calculated, and the following lists were produced:

Anti-Vaccine Word Frequencies Pro-Vaccine Word Frequencies

Text

Description automatically generatedText

Description automatically generated

To visualize the difference in word choice and frequency, the WordCloud library was imported to create a word cloud for each corpus. The larger the word in the illustration, the more frequently it shows up in the corpus.

Anti-Vaccine Words

A picture containing text

Description automatically generated

Pro-Vaccine Words

A picture containing text

Description automatically generated

In the Anti-Vaccine word cloud, the word *supposed* jumps out as a more frequently occurring than. The word *supposed* has a connotation of distrust. Compare that to frequent Pro-Vaccine words like *tested, licensed*, and *research* which have a more trusting connotation. Further investigation into identifying distrust and trust words and then using those to build a feature for analysis. Perhaps a measure of trust will be a robust tool for classifying vaccine stance.

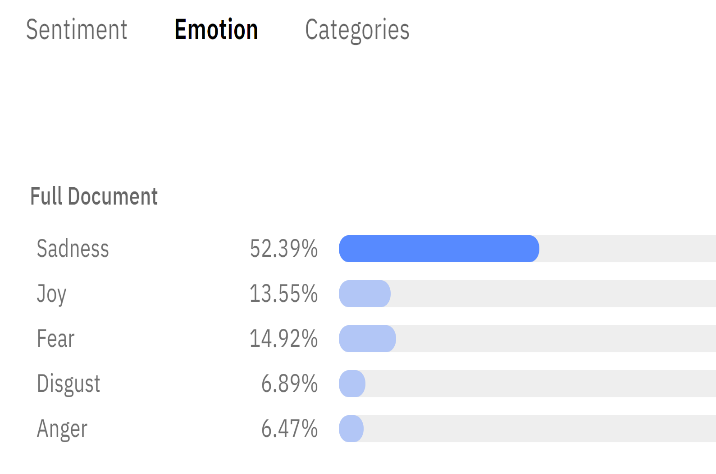
**Watson Analysis: Technology Exploration**

The IBM Watson Natural Language Understanding program is a useful tool for measuring emotion in texts. Each corpus was put into the IBM Watson NLU Demo for analysis of emotion and to compare our sentiment findings to the well-established algorithms of IBM Watson.

At first glance, seeing sadness as the emotion most found in the corpuses, is understandable since much of the discussions discuss death and epidemics. However, a deeper look at the words that IBM Watson tagged with the different emotions, it is extremely evident that this is not a useful tool for analyzing attitudes towards vaccines. For example, joy was ranked highly in the Anti-Vaccine corpus analysis, but it turns out that Watson considers vaccines to be a positive. If the analysis did a better job of identifying the emotion and sentiment of the sentences that the word *vaccine* was in, it would not tag them all as joy. Here is the most glaring example of the mislabeling: *Vaccines are NOT safe*. This is the very first sentence in the Anti-Vaccine corpus and it was wrongly labeled ‘joy’.

Anti-Vaccine Emotion Pro-Vaccine Emotion

Chart

Description automatically generated

Where IBM Watson’s assessment did not fail, was its classification of sentiment polarity. Just as our earlier analysis generated, both Anti-Vaccine and Pro-Vaccine corpuses were negative, with the Pro-Vaccine score a little less negative than Anti-Vaccine.

Anti-Vaccine Polarity Pro-Vaccine Polarity

A picture containing timeline

Description automatically generatedA picture containing chart

Description automatically generated

**Classification Experiment**

After analyzing both vaccine corpuses at different levels of the Synchronic Model of Language, machine learning and deep learning models were built to predict whether a given article is Anti-Vaccine or Pro-Vaccine.

The data used for training and testing the models was the original csv file that all articles were organized after being scrapped from various websites. Each article was labeled pro or anti, depending on the obvious stance after reading the article. The file contained 9 pro and 9 anti articles. After reading the file into python, the text was cleaned using the same code as discussed in the preprocessing phase of the previous corpuses. Once the data was cleaned, the sklearn library was imported. Next the data was separated into a training and a testing set following the 80/20 rule. All text was then transformed to numerical data using Bag of Words.

The first model used for prediction was Multinomial Naïve Bayes. This model classifies by calculating the previous probability of each label, in this case pro and anti, based on how frequent the label occurs in the training set. After using our training data set to train and fit the model, the test data was inputted into the model and the following classification report was generated:

Table

Description automatically generated

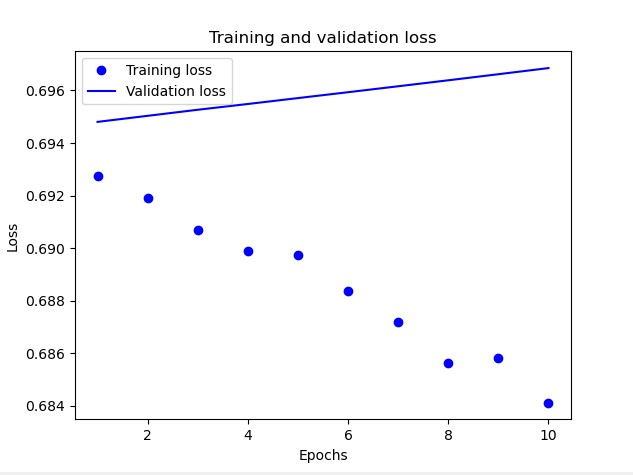
Focusing on the f1-score, which is a measure of accuracy with 1 indicating the highest accuracy, this model’s accuracy is low.

The second classification model used was Random Forest. Simply put, Random Forests classify by creating lots of decision trees with the training data. The steps for preparing data for Random Forest classification is the same as for Naïve Bayes. When training the model, one of the parameters (n\_estimators) had to be tuned multiple times due to overfitting. It is recommended to set the parameter between 500-1000, but this is for larger data sets than being used. Setting n\_estimators = 250 produced the best outcomes, with a f1-score of 0.8.

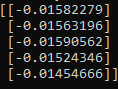
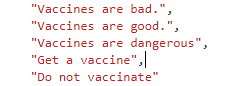
Table

Description automatically generated

Because Deep Learning is where Natural Language Processing is headed, the final model used to predict whether an article is anti-vaccine or pro-vaccine in this study is a Recurrent Neural Network. RNN works by processing sequences of patterns in the training data. Tensor Flow was used to build the RNN. The accuracy report generated by the neural network shows that the more cycles (epochs) that the algorithm ran, the lower the training loss and the higher the validation loss. Note that each iteration delivers different results, as the training model utilized is configured to randomly sample only part of the data. This was done specifically with the intent that larger datasets will become available and compute resources would be limited, or that the datasets would eventually become unbalanced, with antivax-negative texts vastly outweighing antivax positive datasets.



The tensorflow approach has some limitations in this experiment, largely due to the lack of data available for training – the model used would be highly accurate with a library of a hundred or so positive and negative articles, and could be even more accurate with articles being rated on a scale. However, due to the scope and time constraints present, this was not possible. Despite this, the model was able to report an accuracy of about 60%, which is not bad considering that it had only 16 examples to work with. Where the model falls short is with short, test texts, which it will rate close to negative regardless of input. The figures below show each phrase and their rating. Ratings close to 0 would be considered pro vaccine sentiment, where 1 is antivaccine.



**Future Development and Application**

Another evaluation of polarity and subjectivity should be performed on a much larger data set. There is just enough evidence here to suggest polarity is a better indicator of vaccine stance than subjectivity. Also, a deeper look at lexicons that show trust or distrust and their frequencies based on vaccine stance needs to happen. The best indicator of whether an article is anti-vaccine or pro-vaccine will most likely be found through a combination score between polarity and trust word frequencies. This combination should then be used to build a neural network for increased classification accuracy.

Creating an accurate classifier of anti-vaccine articles has a bigger potential application in identifying other misinformation laden or conspiratorial texts passed off as fact.

# **References**

Catherine Shu, J. S. (2020, March 16). *Facebook, Reddit, Google, LinkedIn, Microsoft, Twitter and YouTube issue joint statement on misinformation*. Retrieved from techcrunch.com: https://techcrunch.com/2020/03/16/facebook-reddit-google-linkedin-microsoft-twitter-and-youtube-issue-joint-statement-on-misinformation/

Northwestern University. (2020, December 15). *News Release: Vaccine hesitancy is fueled by polarization and mistrust in science, experts say*. Retrieved from news.northwestern.edu: https://news.northwestern.edu/stories/2020/12/vaccine-hesitancy-is-fueled-by-polarization-and-mistrust-in-science-experts-say/&fj=1

Omer, S. B. (2020, October 27). *The discredited doctor hailed by the anti-vaccine movement: Riveting biography of Andrew Wakefield is a cautionary lesson in the legacy of hubris.* Retrieved from Nature.com: https://www.nature.com/articles/d41586-020-02989-9

**Appendix**

Code

#!/usr/bin/env python

# coding: utf-8

## import sentences from 5 misinformation antivax texts to get avg scores

# In[151]:

MIStatement2 = [ ## See original code for sentences entered here, it added 10 pages to this appendix]

# In[152]:

import pandas as pd

MIdf2=pd.DataFrame(MIStatement2)

MIdf2.columns = ['text']

# In[42]:

#Clean the text

import re

def clean\_text(df, text\_field, new\_text\_field\_name):

df[new\_text\_field\_name] = df[text\_field].str.lower()

df[new\_text\_field\_name] = df[new\_text\_field\_name].apply(lambda elem: re.sub(r"(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)|^rt|http.+?", "", elem))

# remove numbers

df[new\_text\_field\_name] = df[new\_text\_field\_name].apply(lambda elem: re.sub(r"\d+", "", elem))

return df

data\_clean = clean\_text(MIdf2, 'text','clean\_txt')

data\_clean.head()

# In[43]:

#Remove Stopwords

from nltk.corpus import stopwords

stop = stopwords.words('english')

data\_clean['clean\_txt'] = data\_clean['clean\_txt'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))

data\_clean.head()

# In[44]:

#word tokenize

from nltk.tokenize import sent\_tokenize, word\_tokenize

data\_clean['text\_tokens'] = data\_clean['clean\_txt'].apply(lambda x: word\_tokenize(x))

data\_clean.head()

# In[46]:

#Lemmatize to get to root words

from nltk.stem import WordNetLemmatizer

def word\_lemmatizer(text):

lem\_text = [WordNetLemmatizer().lemmatize(i) for i in text]

return lem\_text

data\_clean['text\_tokens\_lemma'] = data\_clean['text\_tokens'].apply(lambda x: word\_lemmatizer(x))

data\_clean.head()

# In[45]:

#POS Tags

import nltk

def word\_pos\_tagger(text):

pos\_tagged\_text = nltk.pos\_tag(text)

return pos\_tagged\_text

nltk.download('averaged\_perceptron\_tagger')

data\_clean['text\_tokens\_pos\_tagged'] = data\_clean['text\_tokens'].apply(lambda x: word\_pos\_tagger(x))

data\_clean.head()

# In[9]:

## Now Polarity and Subjectivity Analysis at various levels of the Synchronic Model

# In[10]:

# Semantic Level

# In[47]:

from textblob import TextBlob

data\_clean['polarity'] = data\_clean['clean\_txt'].apply(lambda x: TextBlob(x).polarity)

data\_clean['subjective'] = data\_clean['clean\_txt'].apply(lambda x: TextBlob(x).subjectivity)

# In[48]:

data\_clean['polarity']

# In[49]:

avg\_txt\_polarity = sum(data\_clean['polarity'])/len(data\_clean['polarity'])

avg\_txt\_polarity

# In[50]:

data\_clean['subjective']

# In[51]:

avg\_txt\_subjective= sum(data\_clean['subjective'])/len(data\_clean['subjective'])

avg\_txt\_subjective

# In[52]:

#syntactic level with ngram analysis

# In[53]:

from nltk.corpus import stopwords

stoplist = stopwords.words('english')

# In[54]:

##create ngrams

from sklearn.feature\_extraction.text import CountVectorizer

c\_vec = CountVectorizer(stop\_words=stoplist, ngram\_range=(2,3))

# matrix of ngrams

ngrams = c\_vec.fit\_transform(data\_clean['clean\_txt'])

# count frequency of ngrams

count\_values = ngrams.toarray().sum(axis=0)

# list of ngrams

vocab = c\_vec.vocabulary\_

MI\_ngram = pd.DataFrame(sorted([(count\_values[i],k) for k,i in vocab.items()], reverse=True)

).rename(columns={0: 'frequency', 1:'bigram/trigram'})

# In[55]:

MI\_ngram

# In[56]:

#polarity and subjectivity of ngrams

MI\_ngram['polarity'] = MI\_ngram['bigram/trigram'].apply(lambda x: TextBlob(x).polarity)

MI\_ngram['subjective'] = MI\_ngram['bigram/trigram'].apply(lambda x: TextBlob(x).subjectivity)

# In[57]:

MI\_ngram['polarity']

# In[58]:

avg\_MI\_ngram\_polarity = sum(MI\_ngram['polarity'])/len(MI\_ngram['polarity'])

avg\_MI\_ngram\_polarity

# In[59]:

MI\_ngram['subjective']

# In[60]:

avg\_MI\_ngram\_subjective = sum(MI\_ngram['subjective'])/len(MI\_ngram['subjective'])

avg\_MI\_ngram\_subjective

# In[21]:

#Lexical analysis Word Frequency Distribution

# In[61]:

lists1 = data\_clean['text\_tokens\_lemma']

# In[62]:

words = []

for wordList in lists1:

words += wordList

# In[63]:

from nltk.corpus import brown

from nltk import FreqDist

#remove 100 most common words based on Brown corpus

fdist = FreqDist(brown.words())

mostcommon = fdist.most\_common(100)

mclist = []

for i in range(len(mostcommon)):

mclist.append(mostcommon[i][0])

words = [w for w in words if w not in mclist]

# In[64]:

#keep only most common words

fdist = FreqDist(words)

mostcommon = fdist.most\_common(100)

mclist = []# mclist contains second-most common set of 100 words

for i in range(len(mostcommon)):

mclist.append(mostcommon[i][0])

# In[65]:

words = [w for w in words if w in mclist]

# ALL occurrences of the words in mclist

words

# In[66]:

fdist1 = nltk.FreqDist(words)

fdist1.most\_common(10)

# In[67]:

topkeys = fdist1.most\_common(30)

for pair in topkeys:

print(pair)

numwords=len(words)

topkeysnormalized = [(word,freq/numwords)for(word,freq)in topkeys]

for pair in topkeysnormalized:

print(pair)

# In[68]:

## Now do same previous steps for 5 provax articles

# In[69]:

ProTexts = [ ## see original code for sentences added here, it added 10 pages to the appendix]

# In[70]:

ProDF=pd.DataFrame(ProTexts)

ProDF.columns = ['text']

# In[71]:

#clean text with function defined earlier

pro\_data\_clean = clean\_text(ProDF, 'text','clean\_txt')

pro\_data\_clean.head()

# In[72]:

#Remove Stopwords

from nltk.corpus import stopwords

stop = stopwords.words('english')

pro\_data\_clean['clean\_txt'] = pro\_data\_clean['clean\_txt'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))

pro\_data\_clean.head()

# In[73]:

#word tokenize

from nltk.tokenize import sent\_tokenize, word\_tokenize

pro\_data\_clean['text\_tokens'] = pro\_data\_clean['clean\_txt'].apply(lambda x: word\_tokenize(x))

pro\_data\_clean.head()

# In[74]:

#Lemmatize to get to root words

from nltk.stem import WordNetLemmatizer

def word\_lemmatizer(text):

lem\_text = [WordNetLemmatizer().lemmatize(i) for i in text]

return lem\_text

pro\_data\_clean['text\_tokens\_lemma'] = pro\_data\_clean['text\_tokens'].apply(lambda x: word\_lemmatizer(x))

pro\_data\_clean.head()

# In[75]:

#using POS tagger defined in previous step

pro\_data\_clean['text\_tokens\_pos\_tagged'] = pro\_data\_clean['text\_tokens'].apply(lambda x: word\_pos\_tagger(x))

pro\_data\_clean.head()

# In[69]:

# Pol and Subj analysis

# In[70]:

#Semantic level

# In[76]:

from textblob import TextBlob

pro\_data\_clean['polarity'] = pro\_data\_clean['clean\_txt'].apply(lambda x: TextBlob(x).polarity)

pro\_data\_clean['subjective'] = pro\_data\_clean['clean\_txt'].apply(lambda x: TextBlob(x).subjectivity)

# In[77]:

pro\_data\_clean['polarity']

# In[78]:

avg\_pro\_data\_clean\_polarity = sum(pro\_data\_clean['polarity'])/len(pro\_data\_clean['polarity'])

avg\_pro\_data\_clean\_polarity

# In[79]:

pro\_data\_clean['subjective']

# In[80]:

avg\_pro\_data\_clean\_subjective = sum(pro\_data\_clean['subjective'])/len(pro\_data\_clean['subjective'])

avg\_pro\_data\_clean\_subjective

# In[39]:

#Syntactic Level

# In[81]:

##create ngrams

from sklearn.feature\_extraction.text import CountVectorizer

c\_vec = CountVectorizer(stop\_words=stoplist, ngram\_range=(2,3))

# matrix of ngrams

ngrams = c\_vec.fit\_transform(pro\_data\_clean['clean\_txt'])

# count frequency of ngrams

count\_values = ngrams.toarray().sum(axis=0)

# list of ngrams

vocab = c\_vec.vocabulary\_

Pro\_ngram = pd.DataFrame(sorted([(count\_values[i],k) for k,i in vocab.items()], reverse=True)

).rename(columns={0: 'frequency', 1:'bigram/trigram'})

# In[82]:

Pro\_ngram

# In[83]:

#polarity and subjectivity of ngrams

Pro\_ngram['polarity'] = Pro\_ngram['bigram/trigram'].apply(lambda x: TextBlob(x).polarity)

Pro\_ngram['subjective'] = Pro\_ngram['bigram/trigram'].apply(lambda x: TextBlob(x).subjectivity)

# In[84]:

Pro\_ngram['polarity']

# In[85]:

avg\_Pro\_ngram\_polarity = sum(Pro\_ngram['polarity'])/len(Pro\_ngram['polarity'])

avg\_Pro\_ngram\_polarity

# In[86]:

Pro\_ngram['subjective']

# In[87]:

avg\_Pro\_ngram\_subjective = sum(Pro\_ngram['subjective'])/len(Pro\_ngram['subjective'])

avg\_Pro\_ngram\_subjective

# In[88]:

#Lexical analysis Word Frequency Distribution

# In[89]:

lists = pro\_data\_clean['text\_tokens\_lemma']

# In[90]:

words = []

for wordList in lists:

words += wordList

# In[91]:

from nltk.corpus import brown

#remove 100 most common words based on Brown corpus

fdist1 = FreqDist(brown.words())

mostcommon1 = fdist1.most\_common(100)

mclist1 = []

for i in range(len(mostcommon1)):

mclist1.append(mostcommon1[i][0])

words1 = [w for w in words if w not in mclist1]

# In[92]:

#keep only most common words

fdist2 = FreqDist(words1)

mostcommon2 = fdist2.most\_common(100)

mclist2 = []# mclist contains second-most common set of 100 words

for i in range(len(mostcommon2)):

mclist2.append(mostcommon2[i][0])

# In[93]:

words2 = [w for w in words if w in mclist2]

# ALL occurrences of the words in mclist

words2

# In[94]:

fdist2 = nltk.FreqDist(words2)

fdist2.most\_common(10)

# In[95]:

topkeys2 = fdist2.most\_common(30)

for pair in topkeys2:

print(pair)

numwords2=len(words2)

topkeysnormalized2 = [(word,freq/numwords2)for(word,freq)in topkeys2]

for pair in topkeysnormalized2:

print(pair)

# In[96]:

## Word Cloud using all antivaccine and pro vaccine documents collected for

import pandas as pd

df = pd.read\_csv('C:\\Users\katva\Desktop\Vaccine\_Texts.csv')

df.tail(5)

from PIL import Image

from wordcloud import WordCloud

import matplotlib.pyplot as plt

import numpy as np

from wordcloud import WordCloud,STOPWORDS

#set the stopwords list

stop\_words = ["s", "re", "t"] + list(STOPWORDS)

#generate the word cloud with parameters

wc = WordCloud(background\_color="white",

stopwords=stop\_words,

normalize\_plurals= True)

X = df['Body']

wordcloud1 = wc.generate(X[0]) # for antivax

print(df['Category'][0:10])

plt.imshow(wordcloud1, interpolation='bilinear')

plt.show()

wordcloud2 = wc.generate(X[11]) # for provax

print(df['Category'][11:21])

plt.imshow(wordcloud2, interpolation='bilinear')

plt.show()

# In[ ]:

## Deep Learning with code help from https://realpython.com/python-keras-text-classification/

# In[124]:

import pandas as pd

df = pd.read\_csv('C:\\Users\katva\Desktop\Vaccine\_Texts2.csv', names=['Category', 'Body'],skipinitialspace=True, skiprows=1, sep=',')

#df\_test = pd.read\_csv(test\_file, names=COLUMNS\_TEST, skipinitialspace=True, skiprows=1, engine="python")

df\_list = []

df\_list.append(df)

df = pd.concat(df\_list)

print(df.iloc[1])

# In[125]:

df.dtypes

# In[127]:

X=df.Body

y=df.Category

# In[128]:

documents = []

from nltk.stem import WordNetLemmatizer

stemmer = WordNetLemmatizer()

for sen in range(0, len(X)):

# Remove all the special characters

df = re.sub(r'\W', ' ', str(X[sen]))

# remove all single characters

df = re.sub(r'\s+[a-zA-Z]\s+', ' ', df)

# Remove single characters from the start

df = re.sub(r'\^[a-zA-Z]\s+', ' ', df)

# Substituting multiple spaces with single space

df = re.sub(r'\s+', ' ', df, flags=re.I)

# Removing prefixed 'b'

df = re.sub(r'^b\s+', '', df)

# Converting to Lowercase

df = df.lower()

# Lemmatization

df = df.split()

df = [stemmer.lemmatize(word) for word in df]

df = ' '.join(df)

documents.append(df)

# In[129]:

#Use Bag of Words to convert text to numbers

from sklearn.feature\_extraction.text import CountVectorizer

vectorizer = CountVectorizer(max\_features=1500, min\_df=5, max\_df=0.7, stop\_words=stopwords.words('english'))

X = vectorizer.fit\_transform(documents).toarray()

# In[131]:

from sklearn.feature\_extraction.text import TfidfTransformer

tfidfconverter = TfidfTransformer()

X = tfidfconverter.fit\_transform(X).toarray()

# In[132]:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# In[133]:

type(text)

# In[139]:

from keras.models import Sequential

from keras import layers

input\_dim = X\_train.shape[1] # Number of features

model = Sequential()

model.add(layers.Dense(10, input\_dim=input\_dim, activation='relu'))

model.add(layers.Dense(1, activation='sigmoid'))

# In[140]:

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

model.summary()

# In[141]:

history = model.fit(X\_train, y\_train,

epochs=100,

verbose=False,

validation\_data=(X\_test, y\_test),

batch\_size=10)

# In[142]:

loss, accuracy = model.evaluate(X\_train, y\_train, verbose=False)

print("Training Accuracy: {:.4f}".format(accuracy))

loss, accuracy = model.evaluate(X\_test, y\_test, verbose=False)

print("Testing Accuracy: {:.4f}".format(accuracy))

# In[106]:

## Random Forrest

from sklearn.ensemble import RandomForestClassifier

# In[107]:

RFcorpus = ('unvaccinated children','disease','vaccine injury','liscensed', 'vaccineautism connection','tested','vaccine industry','autism','aluminum', 'thimerosal',

'risk','research', 'mercury','safe and effective', 'injury', 'vaccine safety','infection','effective','instinct','not safe')

# In[108]:

Y = np.array([1,2,1,2,1,2,1,1,1,1,1,2,1,2,1,2,1,2,1,1])

# In[109]:

vectorizer = CountVectorizer(min\_df=1)

X = vectorizer.fit\_transform(RFcorpus).toarray()

# In[116]:

clf = RandomForestClassifier()

clf.fit(X, Y)

clf.predict(X)

# In[ ]:

## That works so now I will bring in a larger test

# In[117]:

Z = vectorizer.fit\_transform(pro\_data\_clean['clean\_txt']).toarray()

# In[118]:

clf = RandomForestClassifier()

clf.fit(X, Y)

clf.predict(Z)

# In[ ]:

#Naive Bayes

# Importing necessary libraries

from sklearn.preprocessing import LabelEncoder

y = df['Category']

labelencoder = LabelEncoder()

y = labelencoder.fit\_transform(y)

# In[ ]:

X = df['Body']

# In[ ]:

# Import libraries

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.model\_selection import train\_test\_split

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.Body, df.Category, test\_size=0.2,random\_state=109) # 80% training and 20% test

# In[ ]:

#Clean the text

import string

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

lemmatiser = WordNetLemmatizer()

# Defining a module for Text Processing

def text\_process(tex):

# 1. Removal of Punctuation Marks

nopunct=[char for char in tex if char not in string.punctuation]

nopunct=''.join(nopunct)

# 2. Lemmatisation

a=''

i=0

for i in range(len(nopunct.split())):

b=lemmatiser.lemmatize(nopunct.split()[i], pos="v")

a=a+b+' '

# 3. Removal of Stopwords

return [word for word in a.split() if word.lower() not

in stopwords.words('english')]

# In[ ]:

# defining the bag-of-words transformer on the text-processed corpus

bow\_transformer=CountVectorizer(analyzer=text\_process).fit(X\_train)

text\_bow\_train=bow\_transformer.transform(X\_train)

text\_bow\_test=bow\_transformer.transform(X\_test)

# In[ ]:

# transforming into Bag-of-Words and hence textual data to numeric..

text\_bow\_train=bow\_transformer.transform(X\_train)#ONLY TRAINING DATA

text\_bow\_train.shape

# In[ ]:

# transforming into Bag-of-Words and hence textual data to numeric..

text\_bow\_test=bow\_transformer.transform(X\_test)#TEST DATA

text\_bow\_test.shape

# In[ ]:

#train the NB classifier

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB()

clf = model.fit(text\_bow\_train, y\_train)

# In[ ]:

#performance of clf

clf.score(text\_bow\_train, y\_train)

# In[ ]:

from sklearn.metrics import classification\_report

predicted = clf.predict(text\_bow\_test)

# getting the Precision, Recall, F1-Score

print(classification\_report(y\_test,predicted))

# In[ ]:

## Random Forest

import numpy as np

import re

import nltk

import pandas as pd

nltk.download('stopwords')

import pickle

from nltk.corpus import stopwords

df = pd.read\_csv('C:\\Users\katva\Desktop\Vaccine\_Texts2.csv')

df.head(5)

# In[ ]:

y = df.Category

# In[ ]:

X=df.Body

## Clean the data

documents = []

from nltk.stem import WordNetLemmatizer

stemmer = WordNetLemmatizer()

for sen in range(0, len(X)):

# Remove all the special characters

df = re.sub(r'\W', ' ', str(X[sen]))

# remove all single characters

df = re.sub(r'\s+[a-zA-Z]\s+', ' ', df)

# Remove single characters from the start

df = re.sub(r'\^[a-zA-Z]\s+', ' ', df)

# Substituting multiple spaces with single space

df = re.sub(r'\s+', ' ', df, flags=re.I)

# Removing prefixed 'b'

df = re.sub(r'^b\s+', '', df)

# Converting to Lowercase

df = df.lower()

# Lemmatization

df = df.split()

df = [stemmer.lemmatize(word) for word in df]

df = ' '.join(df)

documents.append(df)

# In[ ]:

#Use Bag of Words to convert text to numbers

from sklearn.feature\_extraction.text import CountVectorizer

vectorizer = CountVectorizer(max\_features=1500, min\_df=5, max\_df=0.7, stop\_words=stopwords.words('english'))

X = vectorizer.fit\_transform(documents).toarray()

# In[ ]:

#BOWassigns a score to a word based on its occurrence in a particular document. It doesn't take into account the fact that the word might also be having a high frequency of occurrence in other documents as well. TFIDF resolves this issue by multiplying the term frequency of a word by the inverse document frequency. The TF stands for "Term Frequency" while IDF stands for "Inverse Document Frequency".

from sklearn.feature\_extraction.text import TfidfTransformer

tfidfconverter = TfidfTransformer()

X = tfidfconverter.fit\_transform(X).toarray()

# In[ ]:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# In[ ]:

classifier = RandomForestClassifier(n\_estimators=250, random\_state=0, max\_features='auto',criterion='gini')

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

#Evaluate the model

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

print(confusion\_matrix(y\_test,y\_pred))

print(classification\_report(y\_test,y\_pred))

print(accuracy\_score(y\_test, y\_pred))

#### RNN Experiment####

## Comment this out to enable TF messages for TF troubleshooting

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3'

import numpy as np

import re

import shutil

import string

import matplotlib.pyplot as plt

## I want to use Mongo to handle corpuses later, but have had trouble with it. I'm leaving this code intact for now.

#print("\nConnecting to the local MongoDB\n")

#from pymongo import MongoClient

#client=MongoClient('localhost',27017)

print("Starting up TensorFlow. This could take a moment.")

import tensorflow as tf

import keras.preprocessing.text

from tensorflow.keras import layers

from tensorflow.keras import losses

from tensorflow.keras import preprocessing

from tensorflow.keras.layers.experimental.preprocessing import TextVectorization

# Used these for doing some preliminary testing. Commenting out for now.

# import tensorflow\_datasets as tfds

# import tensorflow\_hub as hub

import matplotlib.pyplot as plt

# Validate that TF is Ok. See line 2 comment to enable info-level logging

print("Version: ",tf.\_\_version\_\_)

print("Eager mode: ",tf.executing\_eagerly())

# print("Hub version: ",hub.\_\_version\_\_)

print("GPU is", "available" if tf.config.list\_physical\_devices('GPU') else "NOT AVAILABLE -- you should fix this!")

#load and process the dataset

dataset\_dir = os.path.join('C:/users/james/OneDrive/Documents/python/corpus/train')

#os.listdir(dataset\_dir) # This line for troubleshooting data directory

train\_dir=(dataset\_dir)

test\_dir=os.path.join('C:/users/james/OneDrive/Documents/python/corpus/test')

raw\_train\_ds = tf.keras.preprocessing.text\_dataset\_from\_directory(

train\_dir, labels='inferred', label\_mode='binary',

class\_names=None, batch\_size=32, max\_length=None, shuffle=True, seed=42,

validation\_split=0.2, subset='training', follow\_links=False

)

raw\_val\_ds = tf.keras.preprocessing.text\_dataset\_from\_directory(

train\_dir, batch\_size=32, validation\_split=0.2, subset='validation', seed=42)

print("Label 0 corresponds to", raw\_train\_ds.class\_names[0],"or not Antivax Articles")

print("Label 1 corresponds to", raw\_train\_ds.class\_names[1],"or Known Antivax Articles")

raw\_test\_ds = tf.keras.preprocessing.text\_dataset\_from\_directory(test\_dir, batch\_size=32)

def custom\_standardization(input\_data):

lowercase = tf.strings.lower(input\_data)

stripped\_html = tf.strings.regex\_replace(lowercase, '<br />', ' ')

return tf.strings.regex\_replace(stripped\_html, '[%s]' % re.escape(string.punctuation), '')

max\_features = 10000

sequence\_length = 250

vectorize\_layer = TextVectorization(

standardize=custom\_standardization,

max\_tokens=max\_features,

output\_mode='int',

output\_sequence\_length=sequence\_length

)

# Create text-only dataset (without labels), then call adapt

train\_text = raw\_train\_ds.map(lambda x, y: x)

vectorize\_layer.adapt(train\_text)

def vectorize\_text(text, label):

text = tf.expand\_dims(text, -1)

return vectorize\_layer(text), label

## This is supposed to provide some test data, but there's a weird datatype issue with numpy. Shouldn't effect the end model.

# retrieve a batch from the dataset

#text\_batch, label\_batch = next(iter(raw\_train\_ds))

#first\_review, first\_label = text\_batch[0], label\_batch[0]

#print("Review", first\_review)

#print("Label", raw\_train\_ds.class\_names[first\_label])

#print("Vectorized review", vectorize\_text(first\_review, first\_label))

train\_ds = raw\_train\_ds.map(vectorize\_text)

val\_ds = raw\_val\_ds.map(vectorize\_text)

test\_ds = raw\_test\_ds.map(vectorize\_text)

AUTOTUNE = tf.data.experimental.AUTOTUNE

train\_ds = train\_ds.cache().prefetch(buffer\_size=AUTOTUNE)

val\_ds = val\_ds.cache().prefetch(buffer\_size=AUTOTUNE)

test\_ds = test\_ds.cache().prefetch(buffer\_size=AUTOTUNE)

embedding\_dim = 16

model = tf.keras.Sequential([

layers.Embedding(max\_features + 1, embedding\_dim),

layers.Dropout(0.2),

layers.GlobalAveragePooling1D(),

layers.Dropout(0.2),

layers.Dense(1)])

model.summary()

model.compile(loss=losses.BinaryCrossentropy(from\_logits=True), optimizer='adam', metrics=tf.metrics.BinaryAccuracy(threshold=0.0))

epochs = 10

history = model.fit(

train\_ds,

validation\_data=val\_ds,

epochs=epochs)

loss, accuracy = model.evaluate(test\_ds)

print("Loss: ", loss)

print("Accuracy: ", accuracy)

history\_dict = history.history

history\_dict.keys()

import matplotlib.pyplot as plt

acc = history.history['binary\_accuracy']

val\_acc = history.history['val\_binary\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(acc) + 1)

# "bo" is for "blue dot"

plt.plot(epochs, loss, 'bo', label='Training loss')

# b is for "solid blue line"

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.show()

examples = [

"Vaccines are bad.",

"Vaccines are good.",

"Vaccines are dangerous",

"Get a vaccine",

"Do not vaccinate"

]

examples=vectorize\_layer(examples)

prediction = model.predict(examples)

print(prediction)

Output:

(base) C:\Users\james>python vax\_engine.py

Starting up TensorFlow. This could take a moment.

Version: 2.3.1

Eager mode: True

GPU is available

Found 16 files belonging to 2 classes.

Using 13 files for training.

Found 16 files belonging to 2 classes.

Using 3 files for validation.

Label 0 corresponds to neg or not Antivax Articles

Label 1 corresponds to pos or Known Antivax Articles

Found 5 files belonging to 2 classes.

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding (Embedding) (None, None, 16) 160016

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout (Dropout) (None, None, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

global\_average\_pooling1d (Gl (None, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 1) 17

=================================================================

Total params: 160,033

Trainable params: 160,033

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Epoch 1/10

1/1 [==============================] - 0s 200ms/step - loss: 0.6931 - binary\_accuracy: 0.6154 - val\_loss: 0.6981 - val\_binary\_accuracy: 0.3333

Epoch 2/10

1/1 [==============================] - 0s 30ms/step - loss: 0.6924 - binary\_accuracy: 0.6923 - val\_loss: 0.6983 - val\_binary\_accuracy: 0.3333

Epoch 3/10

1/1 [==============================] - 0s 28ms/step - loss: 0.6906 - binary\_accuracy: 0.9231 - val\_loss: 0.6985 - val\_binary\_accuracy: 0.3333

Epoch 4/10

1/1 [==============================] - 0s 29ms/step - loss: 0.6902 - binary\_accuracy: 0.9231 - val\_loss: 0.6987 - val\_binary\_accuracy: 0.3333

Epoch 5/10

1/1 [==============================] - 0s 39ms/step - loss: 0.6880 - binary\_accuracy: 1.0000 - val\_loss: 0.6989 - val\_binary\_accuracy: 0.3333

Epoch 6/10

1/1 [==============================] - 0s 29ms/step - loss: 0.6879 - binary\_accuracy: 1.0000 - val\_loss: 0.6992 - val\_binary\_accuracy: 0.3333

Epoch 7/10

1/1 [==============================] - 0s 30ms/step - loss: 0.6865 - binary\_accuracy: 1.0000 - val\_loss: 0.6995 - val\_binary\_accuracy: 0.3333

Epoch 8/10

1/1 [==============================] - 0s 29ms/step - loss: 0.6858 - binary\_accuracy: 1.0000 - val\_loss: 0.6997 - val\_binary\_accuracy: 0.3333

Epoch 9/10

1/1 [==============================] - 0s 32ms/step - loss: 0.6841 - binary\_accuracy: 1.0000 - val\_loss: 0.6999 - val\_binary\_accuracy: 0.3333

Epoch 10/10

1/1 [==============================] - 0s 29ms/step - loss: 0.6825 - binary\_accuracy: 1.0000 - val\_loss: 0.7002 - val\_binary\_accuracy: 0.3333

1/1 [==============================] - 0s 997us/step - loss: 0.6905 - binary\_accuracy: 0.6000

Loss: 0.6905485391616821

Accuracy: 0.6000000238418579

[[0.02571638]

[0.02546135]

[0.0258476 ]

[0.02605585]

[0.02567069]]