Anti-Vaccination Image Classification

Colin Williams

Project URL: <https://github.com/will3298/CSE482HonorsProject>

**ABSTRACT**

This project utilizes Python and various classification methods to scan images found on the internet and determine if they carry anti-vaccination messages in their contents. Ideally, this project aims to act as a method of flagging images online that contain misinformation based on the text that appears within the images. Through this process, images were able to be classified with reasonable accuracy in order to possibly act as a first measure in the censorship of false information.

# INTRODUCTION

1. During today’s universal access to information via the Internet, it is simpler than ever to spread misinformation disguised as a reliable source. Specifically, the recent rise of anti-vaccination sentiment within communities in the United States has become apparent as these groups make their way onto popular social media sites such as Twitter and Facebook. This project specifically focuses on images that are posted on the Internet, carrying misinformation with them. Many of the images are in the form of satirical posts aimed to mock the current imposition of mandatory vaccination for children and adults in many areas of the United States. It is important to uncover this misinformation on high-traffic sites in order to limit its exposure and ability to spread. Lack of immunization in recent years has led to an increase in preventable diseases, such as whooping cough. Children too young to receive the proper vaccinations for these diseases are at risk when others contract it due to failure to immunize. The leading misconceptions behind the rise of antivaccination sentiment have been debunked by health officials yet many communities and parents continue to refuse the vaccination of themselves and their children.
2. The goal of this project is to process images found on the Internet and perform classification methods on them to identify whether they contain antivaccination messages. The text within the image will be extracted and used as the data for the classification algorithm to run with. Using this classification as a flagging system for images posted online would be useful in the attempt to stop the spread of the misinformation. Ideally the accuracy will be reasonable enough to use as a preliminary measure when scanning images for antivaccination sentiment.
3. In order to acquire a successful means of classifying images, a model must be trained. Specifically, within this project, a linear regression model as well as a Multinomial Naïve Bayes model will be trained and compared with the testing set of data. After tuning the respective hyperparameters, the leading model can be used to classify images as they are posted.
4. Images sampled from the Internet will be processed using Python to extract the text within them. Using this text, a classifier can be trained to determine the sentiment within the images.
5. Challenges that occurred during this project included handling the massive amounts of image data that were provided for use. Many of the preprocessing tasks were run overnight, so the results could be gathered the next day.
6. The findings of this project conclude that is possible to use Python to scan images and perform classification with reasonable accuracy. Using the text found within the project served as a somewhat reliable way of identifying the underlying messages within an image. This process could be used in image censorship to stop the spread of misinformation.

# DATA

The data consists of two folders, one for pro-vaccination images and one for anti-vaccination images. Each image was in JPG format. Data was gathered from multiple social media sites such as Facebook, Twitter, and Instagram. All images were provided by the Stony Brook University Department of Computer Science/Engineering and the Michigan State University Department of Media and Information, Communication Arts and Sciences.

|  |  |
| --- | --- |
| Total Items | 27290 (655 MB) |
| Text Files | 20412 |
| Image Files | 6878 |
| Anti-Sentiment Images | 3690 |
| Pro-Sentiment Images | 3188 |

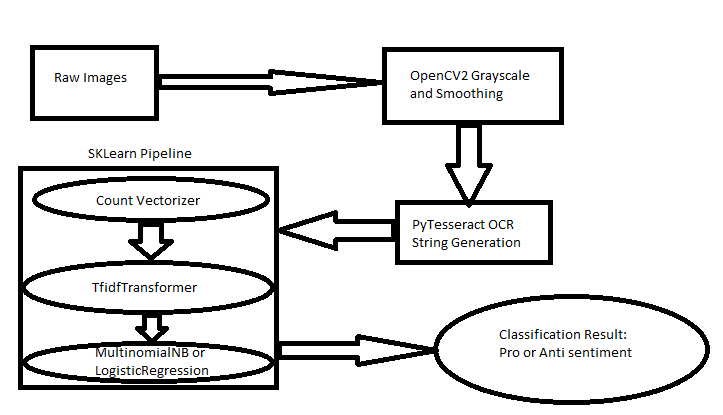
**Table 1:** Information for the raw data.

Preprocessing steps started with removing the extraneous text files from the folder that contained meta data related to the images. Once the folders contained only the image files, the text within each image had to be extracted for use in the classification. This process was completed using the Python packages OpenCV for image handling and PyTesseract for the text extraction. Using OpenCV, each image is read into Python. The image is then converted to grayscale and then smoothed slightly for image clarity. PyTesseract image reading is then used on the new image to grab the text found within. Through the PyTesseract wrapper, the image is fed into Google’s Tesseract-OCR-Engine and the recognized text is returned as a string. Any images that return no text are assumed to contain no words and are discarded from the data set. 4136 images remain in the dataset following this removal. The images are also labeled with their respective folder, “Pro” or “Anti”.

|  |  |  |
| --- | --- | --- |
| Attribute Name | Type | Description |
| File | Nominal | The file path of the image |
| Text | Nominal | The text extracted from the image |
| Class | Nominal | Pro or Anti sentimental – from raw data |

**Table 2:** Attributes for the final data set.

# METHODOLOGY



**Figure 1:** The process of moving data from raw image to the result of the classification

In order to use a classifier on string data, they must undergo some transformation into a feature vector. Data was split into a test size of 25% for the process in this project. As shown in Figure 2, this process is done within the SKLearn Pipeline prior to being inputted into the classification method. To start, the string data is sent through SKLearn Count Vectorizer. This takes the data as a sequence of strings and returns a matrix of token counts for each individual string found. Each feature is given an integer key and an occurrence value within the provided data. Following this, SKlearn TfidfTransformer is used to convert these occurrence values into Term Frequencies. This process accounts for strings of different lengths, as two examples may share the same underlying sentiment, but have large differences in overall word counts. This function also downscales words commonly found in general text, such as conjunctions, so that their impact on the final classification is not major. This deafening is referred to as term frequency times inverse document frequency. Finally, a classifier is trained using the data that has been sent through both the Count Vectorizer and the TfidfTransformer. Two classifiers were used in this project in separate pipelines, the Multinomial Naïve Bayes classifier and a Logistic Regression classifier. The Multinomial Naïve Bayes classifier considers all features independent of each other when determining the classification of an item, which serves as a good baseline model. Logistic Regression generates a set of parameters that define a line of regression when classifying examples of data. Both classifiers were tuned with their respective hyperparameters.

The code within this project includes:

* Preprocessing.ipynb – This file includes all of the preprocessing done for the images as well as the PyTesseract string testing. For the images, they are run through PyTesseract and the output is recorded. The script for the manual PyTesseract testing is provided and the results are compiled into a single excel file.
* Results.ipynb – This file includes the training and testing of the Multinomial Naïve Bayes classifier and the Logistic Regression classifier. The results of the hyperparameter tuning are displayed. The results from the PyTesseract string testing is also in this file.

Specific steps must be followed in order to use the PyTesseract package successfully in a Python script. Details can be found at <https://github.com/madmaze/pytesseract>.

# EXPERIMENTAL EVALUATION

## Experimental Setup

1. The computing platform for this project is Windows OS using Jupyter Notebooks for writing Python code. The Google Tesseract OCR engine is used within the Python code and requires installation.
2. Two different classification methods were used to compare the results with. The Multinomial Naïve Bayes serves as a good baseline, and the Logistic Regression takes it a step further.
3. Accuracy is used to identify the percentage of items correctly classified.

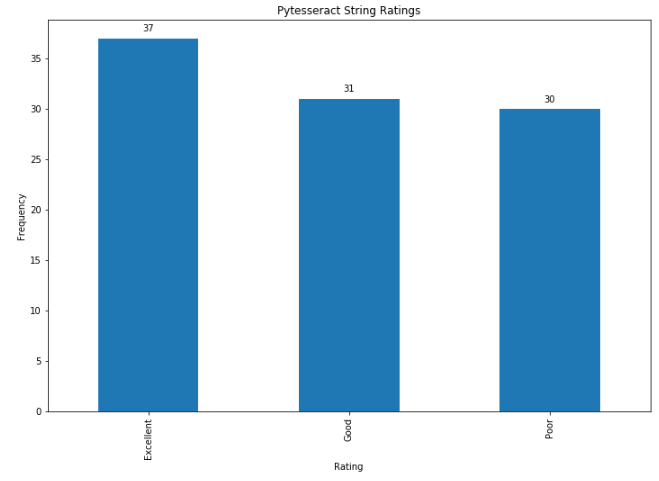
## Experimental Results

### **PyTesseract Results**

PyTesseract is the Python wrapper for Google’s Tesseract-OCR-Engine. This allows for images to be scanned and the text within the image to be returned in the form of a string. In order to test the validity and accuracy of the OCR (Optical Character Recognition) within the engine, 98 images were randomly selected, split evenly between “Pro” and “Anti” folders for a manual reliability test. This consisted of the recording of the strings that appeared within the images manually, then comparing them to the result of the PyTesseract string. The accuracy was graded on a three-level scale in which the levels are as follows: “Excellent”, “Good”, and “Poor”. A string received an “Excellent” score if the PyTesseract string contained all or almost all the text within the image with minimal spelling errors and miscellaneous noise. A “Good” was awarded when the PyTesseract string contained a fair amount of the text with some noise or misspellings, but many of the words on the image were present and formatted correctly. A “Poor” grade was given to PyTesseract strings that included large amounts of noise, many misspellings and missed words, and were overall difficult to trace the original image from. The results were compiled in an excel dataset.

|  |  |  |
| --- | --- | --- |
| Attribute Name | Type | Description |
| Actual String | Nominal | The string recorded manually from the image |
| PyTesseract Result | Nominal | The string returned by PyTesseract |
| ImagePath | Nominal | The image path |
| Rating | Ordinal | The manual rating of the quality of the PyTesseract Result |

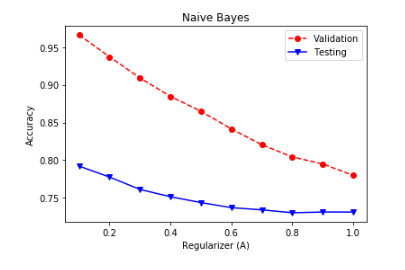
**Table 3:** The attributes for the data collected on the PyTesseract string outputs.



**Figure 2:** The results of the manual string quality testing.

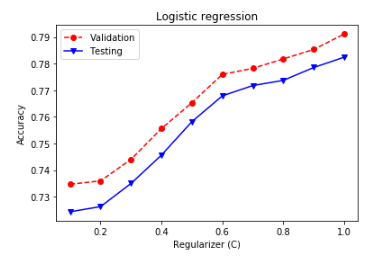
According to the manual test, the quality of the string was split roughly even between the three levels. 37 (~37.8%) of the images scored “Excellent”, 31 (~31.6%) scored “Good” and 30 (~30.6%) scored “Poor”. Through some inspection, images that featured clear dark words against a light background populated the majority of the “Excellent” scores. Images cluttered with words and other items impacted the performance of the OCR engine greatly. The poorer performing samples may be due to images consisting of an image background and text overlay, which could muddle the ability of the OCR. Other reasons could fall on poor image quality or less-clear fonts and sporadic formatting.

**4.2.2 Classification Results**

Both classifiers used were tested for accuracy while adjusting hyperparameters. 

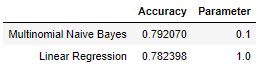
**Figure 3:** Validation and accuracy of the Multinomial Naïve Bayes classifier.

As shown in figure 3, the Multinomial Naïve Bayes classifier hit a peak accuracy of 79% on the testing set. As the hyperparameter, in this case the variable alpha, increased, the accuracy decreased, with the best results at an alpha value of 0.1.



**Figure 4:** Validation and accuracy of the Logistic Regression classifier.

As shown in figure 4, the logistic regression hit a maximum accuracy of 78.2% at a hyperparameter value of 1.0. The results are summarized in the table below.



**Table 4:** The results of the classifier testing.

Surprisingly, the Naïve Bayes classifier, which is considered a baseline for most feature vector classification, performed slightly better than the Linear Regression model. Furthermore, the validation accuracy of the Naïve Bayes classifier hit ranges of over 95% accuracy, while the Linear Regression models’ accuracies were closely coupled.

The classifiers both included coefficient lists for the feature importance used to determine the sentiment behind each photo. Even after the pass through the TfidfTransformer, many the words in the top ten for both the pro and anti classes of the Naïve Bayes classifier consist of common words such as “the” and “you”. However, the logistic regression coefficients contained more specific words for each class. “Vaccine” and “autism” are the two most impactful words when classifying an anti-vaccination image, whereas “surviving” and “health” land in the top five for pro-vaccination images. The logistic regression model does a better job of putting importance on key words to look for when performing this classification.

Overall, the two classification methods performed well and were able to identify the majority of the images correctly. The results may be improved upon if the common words that were flagged by the Naïve Bayes classifier were removed, allowing it to run solely on the specific words that the Logistic Regression model found important.

# CONCLUSIONS

Through the methods of this project, using OCR techniques to identify possibly dangerous information can be done with reasonable accuracy. However, this is just one approach to image classification and can be coupled with others to produce a stronger and more reliable firewall to flag misinformation within images. Social media sites can utilize these methods to keep their platform safe from the spread of harmful information.

# REFERENCES (at least 3 references)

1. Writers, Staff. “Vaccine Myths Debunked.” PublicHealth.org, PublicHealth.org, 23 Apr. 2020, www.publichealth.org/public-awareness/understanding-vaccines/vaccine-myths-debunked/.
2. Young Anna Argyris, PhD, Department of Media and Information, Communication Arts and Sciences, MSU
3. Zhaozheng Yin, PhD, Department of Computer Science/Engineering, Stony Brook University
4. Zuhui Zhang, Department of Computer Science/Engineering, Stony Brook University