Technical Note:

Classify the Masking tweets into four categories

Project conducted by:

San Diego State University
5500 Campanile Drive
San Diego, CA 92182-4493

Report prepared by:

Vaishnavi Bihare
vbihare@gmail.com
San Diego State University

July 6, 2020

Abstract

The rapid growth in social media data has motivated the development of a real time framework to understand and extract the meaning of the data. Text categorization is a well-known method for understanding text. Text categorization can be applied in many forms, such as authorship detection and text mining by extracting useful information from documents to sort a set of documents automatically into predefined categories.

Here, we propose a method for identifying those who posted the tweets into categories. The task is performed by extracting key features from tweets and subjecting them to a machine learning classifier. The research shows that this multi-classification task is very difficult, in particular the building of a domain-independent machine learning classifier. Our problem specifically concerned tweets about masks (during the coronavirus pandemic), most of which were noisy enough to affect the accuracy. The analytical technique used here provided structured and valuable information to understand public's opinion.

HDMA @SDSU

What is a tweet classifier?

Introduction

Text categorization is the process of automatically assigning one or more predefined categories to text documents. In the present study, the terms "documents" and "tweets" refer to a similar concept. We can treat each tweet as a document and use text-categorization concepts such as tokenization, stemming, term-frequency and document-frequency to encapsulate a flexible representation of the problem, making it easy for the text categorization algorithm to be efficiently applied to this problem. However, the machine-learning community has considered using other concepts as vectorize, TF-IDF, which we will be using here.

Our Aim

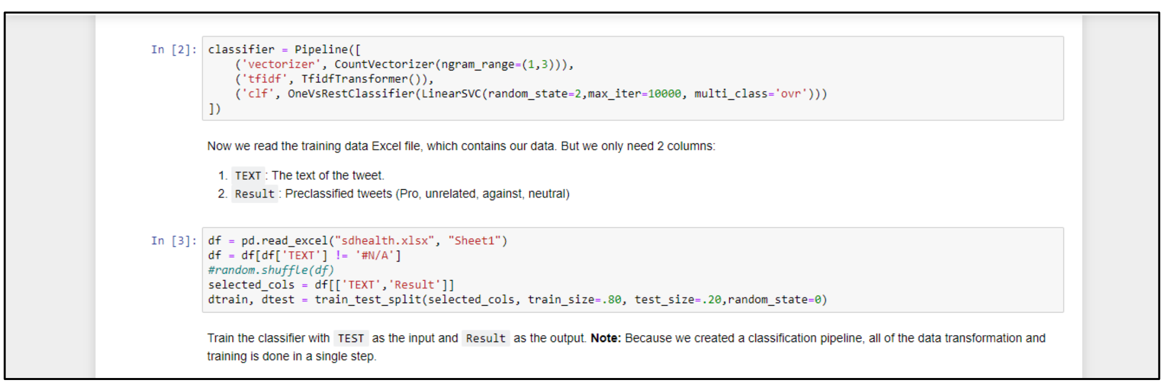
The aim of this work is to categorize incoming tweets automatically into several pre-defined classes. Hence, this project can be termed a multi-class categorization problem. The term multi-class refers to the machine-learning problem where the input instances/documents can be classified into more than two classes/categories. Multi-class categorization is difficult than binary-class classification (with only two output classes/categories). Certain tricks are available for converting multi-class problems into a series of binary-class problems and then predicting the output of classes by means of a voting scheme.

About the training dataset

- "Masking tweets"- The dataset is about the tweets collected over a period of time, describing the reaction of people in San Diego, CA towards wearing a mask during this COVID-19 pandemic.
- We are considering 4 categories to classify our tweets. Pro, Neutral, Against, and Unrelated. This is referred as Multi-class classification.
- We have divided our dataset into two parts, test and train. For a total number of 505 tweets in original "SDhealth.xlsx" file, we have taken 80% of the data to train the model, and we will testing the performance of our model on the remaining 20% of the data i.e. approximately 100 tweets.

How to achieve it:

1. First, we import necessary libraries. We are using scikit-learn to do most of the heavy lifting in terms of transforming and classifying data. Scikit-learn contains a wide range of functions for performing data mining and classification tasks. We also use the Pandas library for reading our data.
2. Before working with any of the training data, we need to implement a classification pipeline, that will combine all the necessary data transformation and modeling steps. Below are the 2 modeling steps we will be using here:
   - **Vectorize:** This step transforms text data into numerical data that can be used for classification. You can read more about it in the below mentioned link. https://en.wikipedia.org/wiki/Bag-of-words_model
   - **TF-IDF:** This is an additional transformation that is common when working with text data. It uses statistical properties of the dataset to assign weights to text terms. You can read more about it in the below mentioned link. https://en.wikipedia.org/wiki/Tf%E2%80%93idf
   - **Classifier:** Finally, we classify data that was transformed by the previous two steps. In this case we are using a linear support vector classifier, which is commonly used in text classification tasks. You can read more about it in the below mentioned link. https://en.wikipedia.org/wiki/Support_vector_machine#Linear_SVM
3. Below is the code on how to design the classifier, and read the data using pandas data frame.



In [2]:
```
classifier = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1,3))),
    ('tfidf', TfidfTransformer()),
    ('clf', OneVsRestClassifier(LinearSVC(random_state=2,max_iter=10000, multi_class='ovr')))
])
```

Now we read the training data Excel file, which contains our data. But we only need 2 columns:

1. `TEXT` : The text of the tweet.
2. `Result` : Preclassified tweets (Pro, unrelated, against, neutral)

In [3]:
```
df = pd.read_excel("sdhealth.xlsx", "Sheet1")
df = df[df['TEXT'] != '#N/A']
#random.shuffle(df)
selected_cols = df[['TEXT','Result']]
dtrain, dtest = train_test_split(selected_cols, train_size=.80, test_size=.20,random_state=0)
```

Train the classifier with `TEST` as the input and `Result` as the output. **Note:** Because we created a classification pipeline, all of the data transformation and training is done in a single step.

Figure 1. Dividing into test and train

4. We can also see that, we have randomly divided the data into two parts, test, and train. Here we will be training our model using 80% of our original dataset, and after passing the data through our classifier i.e. training our classifier, we will test the accuracy of our classifier using the test dataset.
5. Below is the code to execute it:

Train the classifier with `TEST` as the input and `Result` as the output. **Note:** Because we created a classification pipeline, all of the data transformation and training is done in a single step.

```
In [4]: dtrain = dtrain.values.astype(str)
        dtrain = pd.DataFrame(dtrain)
        dtest = dtest.values.astype(str)
        dtest = pd.DataFrame(dtest)
```

```
In [31]: # train the classifier
         classifier.fit(dtrain[0], dtrain[1]);
```

Finally, we can see how the trained classifier performs on new tweets by using the `predict` function.

```
In [10]: # try it out on some sample tweets
         texts = dtest[0]

         outputs = classifier.predict(texts)

         pd.set_option('display.max_rows', None)
         resultdf = pd.DataFrame()
         resultdf['Text']=texts
         resultdf['Result']=outputs

         resultdf
```

Figure 2: Implementing the classifier

F1-Score

- The F1 score, also referred as the F score or F measure, is basically a measure of a test's accuracy. It ranges from 0 to 1.
- The F1 score reaches the best value, meaning perfect precision and recall, at a value of 1. The worst F score, which means lowest precision and lowest recall, would be a value of 0. The below figure shows the calculation of our F1 score:

```
In [14]: #Importing library to calculate the f1 score, precision rate and recall rate
         from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix

         actual_dummy= pd.get_dummies(data=df['Actual results'], columns=['neutral','against','unrelated'])
         pred_dummy= pd.get_dummies(data=df['Result(ML)'], columns= ['neutral','against','unrelated'])

         print(f1_score(actual_dummy, pred_dummy, average="macro"))
         print(precision_score(actual_dummy, pred_dummy, average="macro"))
         print(recall_score(actual_dummy, pred_dummy, average="macro"))

         0.32469073647871116
         0.6202127659574468
         0.3254901960784314
```

Figure 3: F1-Score

- Our F1- Score is- 0.325, precision rate- 0.62, and recall rate- 0.325

Results

Multi-class classification problems suffer from class imbalance, whereby a class which has more training data is more likely to be predicted as an output class also. We can see that in the below images:

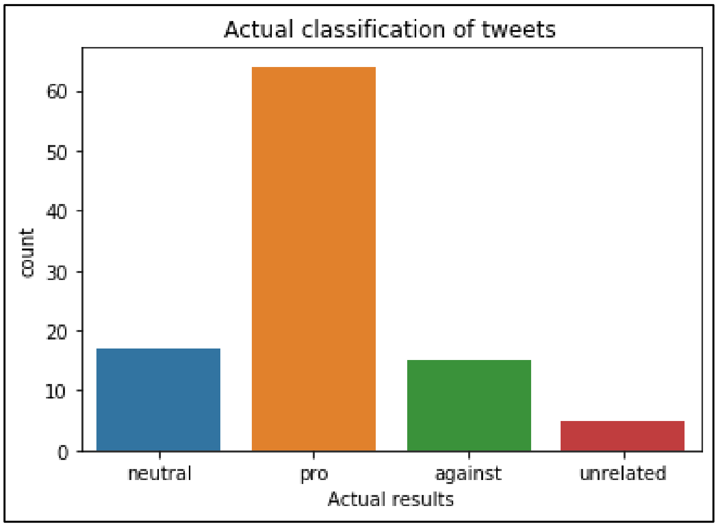


Figure 4: Actual Results of tweet classification

And the below graph shows our result i.e. testing dataset classified into 4 given categories.

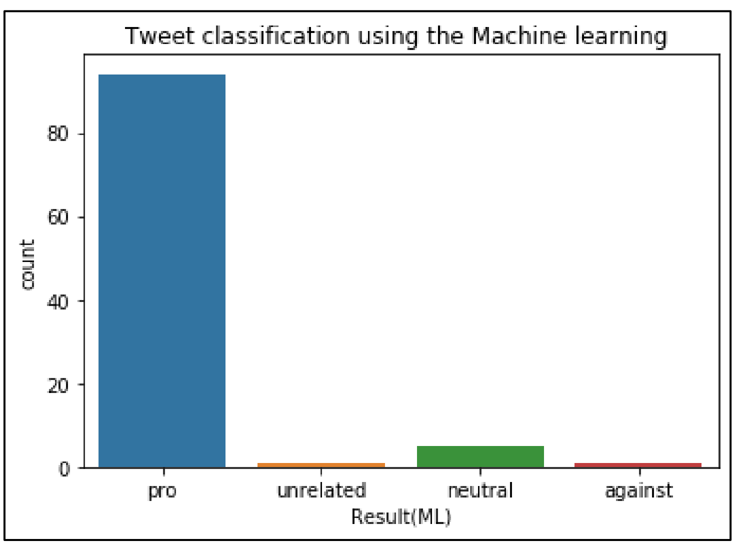


Figure 5: Predicted categories using classifier

As we can see that the original dataset (manually classified) had a greater number of "pro" tweets, hence our classifier turned out be biased. And it predicted almost 85% of the tweets fall into "Pro" category.