Coronavirus Tweet Sentiment Analysis

Sandesh Gaikwad
Data Science Trainee,
AlmaBetter, Bangalore

Abstract: During time of pandemic to manage and keep law and order it is important to know what kind of sentiments people are having . It also helps to target people who are misinformed about the pandemic and help them.

This Ml project uses tweets from different part of world builds model to help them sort according to the sentiments of tweets.

Keywords : machine learning, ordinal error, NLP

1. Problem Statement:

Use the tweets data and build a machine learning model that will classify the tweet sentiments.

Data Summary

I. **User Id and Screen Id:** these are the id of the person tweeting and unique reference values which are not required for classification.

II. **Location:** most of the Twitter users are from the USA and UK. The column is irrelevant to the classification

III. **Tweet time:** date of the tweet, data is available for one month from 16/03/2020 to 14/04/2020.

IV. **Original text:** Original text of a tweet

V. **Sentiments:** Tweets are labeled into five categories
   - ‘Extremely Negative’
   - ‘Negative
   - ‘Neutral’
   - ‘Positive’
   - ‘Extremely Positive’

2. Data Cleaning and preprocessing:

We will only consider tweets and sentiment labels. There are no null values in these columns.

- **Remove user tags**: First, we will remove all user tags from the tweets, user tags don't give any information about the content of the tweet these are just usernames and are irrelevant to NLP

- **Remove links**: links of the websites do not give any specific

information about the content of the tweet. so we will remove all HTTP links.

- **Remove the stop words and punctuation:** In this classification, we are using a tokenization based machine learning approach, so stop words just don't add any value in predicting anything as they are present everywhere

- **Use stemming**: Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words "chocolates", "chocolatey" and "choco" to the root word, "chocolate" and "retrieval", "retrieved", "retrieves" reduce to the stem "retrieve".

3.Tokenization/vectorization

- **Parameters**
  - **Max_df:** Maximum document frequency - we have used the hyperparameter as 0.8 any word that is in 80% of tweets will not be considered for tokenization

  - **Min_df:** Minimum document frequency - we have used this hyperparameter as 20 any word that is in less than 20 tweets will not be considered for tokenization

- **Countvectorizer:** It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text

- **TFidfvectorizer:** Term frequency-inverse document frequency is a text vectorizer that transforms the text into a usable vector. It combines 2 concepts, Term Frequency (TF) and Document Frequency (DF).
  The term frequency is the number of occurrences of a specific term in a document. Term frequency indicates how important a specific term is in a document. Document frequency is the number of documents containing a specific term. Document frequency indicates how common the term is. Inverse document frequency (IDF) is the weight of a term, it aims to reduce the weight of a term if the term's occurrences are scattered throughout all the

Except for the Naive Bayes model all other models is fitted on Tf-Idfvectorizer

4. Misclassifications:

Some misclassifications are okay, but some aren't; let's take an example.
‘Extremely Negative’ to ‘Negative’ is okay, but ‘Extremely Positive’ is not okay. The same goes for positive cases also.
We will use three different methods to evaluate misclassifications.

- **Five Classes**: We will consider all classes independent and consider any misclassification as wrong. This approach lags behind the understating of how different misclassifications have different effects.

- **Three Classes**: We will merge ‘Extremely Negative’ and ‘Negative’ into one class ‘Negative’; ‘Extremely Positive’ and ‘Positive’ into one class ‘Positive’. We will keep Neutral as Neutral. It is important to remember that we are not creating a new classification model for this evaluation. We are using the same classifications model but evaluating it with three classes.

- **Ordinal Error**: Here we will encode five classes and then get the root mean square error and compare that with other models. Encoding
    - ‘Extremely Negative’: -3
    - ‘Negative’ : -2
    - ‘Neutral’ : 0
    - ‘Positive’ : 2
    - ‘Extremely Positive’ : 3

Classification Models:

- **Naive Bayes Algorithm:** - The Multinomial Naive Bayes algorithm is a Bayesian learning approach popular in Natural Language Processing (NLP). It calculates each sentiment's likelihood for a word and outputs the sentiment with the greatest chance. We will use this model as a baseline model.

- **Multiclass Logistic regression classification**: - Multi-class classification is implemented by training multiple logistic regression classifiers, C In the Multi-Class classification example, there are 5 classes. Hence, we need to train 5 different logistic regression classifiers. When training the classifier for label 1, we will treat input data with class 1 labels as +ve samples (y==1) and all other classes as -ve samples (y==0). When training the classifier for Class 2, we will treat input data with class 2 labels as +ve samples (y==1) and all other classes as -ve samples (y==0). This will continue for all the classes.

- **Ensemble of Trees**

Bagging

- **Random Forest Classifier: -** Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data and hence the output doesn't depend on one decision tree but multiple decision trees. In the case of a classification problem, the final output is taken by using the majority voting classifier. In the case of a regression problem, the final output is the mean of all the outputs. This part is Aggregation. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees. Random Forest has multiple decision trees as base learning models. We randomly perform row sampling

Boosting

- **Xgboost:** XGBoost stands for eXtreme Gradient Boosting. XGBoost implements Gradient Boosted decision trees. In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model.

- **Catboost:** "CatBoost" comes from two words "Category" and "Boosting". The library works well with multiple Categories of data, such as audio and text. "Boost" comes from the gradient boosting machine learning algorithm as this library is based on a gradient boosting library.

Results of all models

Model	RMSE	Accuracy (5 Labels)	Accuracy (3 Labels)
Naive Bayes (baseline model)	1.83	0.49	0.68
Xgboost	1.74	0.51	0.74
Random Forest	1.57	0.51	0.69
Logistic Regression	1.74	0.52	0.74
Catboost(best performing model)	1.5	0.59	0.76

Observation and Conclusion:

- Catboost gives best performance among all models
- Catboost outperforms all models in all categories
- The models from scikit learn does not support ordinal errors. The current classification weight every error in same way but some sentiments are closer to other that needs to be considered . Use of classification models with ordinal errors will improve performance.

References :

- Towards data science
- Median