MACHINE LEARNING
Project Report

ASK REDDIT PROJECT REPORT

June 18, 2022

Manaswitha Reddy

1 Abstract

This is a NLP based classification problem, to detect troll questions in AskReddit.

2 Project idea

Reddit is a social media platform that allows users to stay anonymous. Each community or ”subreddit” has its own topics and rules enforced by subreddit moderators. The admins of Reddit only step in when there’s a public controversy or when their Terms of Service are violated. AskReddit is one of the most popular subreddits. This is the place to ask and answer thought provoking questions. With anonymity comes disadvantages. The disadvantage here is not all are useful questions, few of them are ”troll”. The moderators of AskReddit can’t remove every troll question randomly. So the aim of this project to train a model efficiently such that it detects all the troll questions and deletes them. It is a classification problem using Natural Language Processing.

2.1 Analysis of Dataset

Our given train data set contains 6,53,060 rows and 3 columns (QuestionID, Question, Label-Troll/NonTroll). Given test data set contains 6,53,060 rows and 2 columns. We need to predict the Label column. Our data set is imbalanced as we can see in the figure below. We have 6,12,656 non troll questions and 40,405 troll questions in our training data set. We can see that non-troll questions are about 93 % of the data.

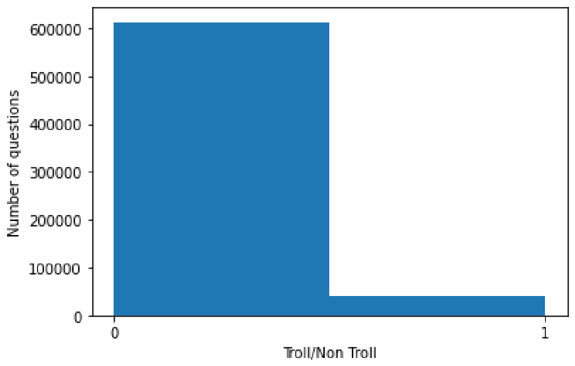


Figure 1: This is our data visualisation

2.2 Pre processing of Data

For better training of ML models we need to clean the data through preprocessing. We used the following preprocessing techinques of NLP.

2.2.1 Removing Punctuation

This step is to remove all the punctuation marks given in the text data.

2.2.2 Tokenization

This technique breaks the sentence into a list of words. This is a basic step to perform further preprocessing and feature engineering.

2.2.3 Removing stop words

This technique removes all the stop words mentioned in the 'corpus.stopword' library from nltk. The words like "the","these","is","are" etc which do not carry any significance are removed.

2.2.4 Stemming

This technique works by cutting off the common ends and beginnings of the word, It considers a list of common prefixes and suffixes that can be found in an inflected word.

2.2.5 Lemmatization

This technique on the other hand considers the morphological analysis of the words. We imported 'WordNetLemmatizer' from nltk library to perform this step.

We need not perform both Stemming and Lemmatization because they both do similar work. Lemmatization is an advanced version of Stemming.

2.3 Feature Engineering

Feature engineering is performed on the data after performing the above mentioned preprocessing steps. Feature engineering transforms raw data into features that can be used in machine learning algorithms. Models consist of an outcome variable and input variables, and it is during the feature engineering process that the most useful input variables are created and selected for the predictive model.
We extracted the following features from given dataset and performed Standardization using StandardScaler library from sklearn.preprocessing.

2.3.1 Sentiment Analysis

This is an approach that identifies the emotional tone behind a body of text. We imported 'textblob' library to give us sentiment value of each question. This tells us whether the underlying sentiment of a statement is positive,negative or neutral.

2.3.2 Word Count

Counts the number of tokens in the text (separated by a space)

2.3.3 Sentence Count

To count the number of sentences (separated by a period)

2.3.4 Average word length

Sum of words length divided by the number of words (character count/word count)

2.3.5 Average sentence length

Sum of sentences length divided by the number of sentences (word count/sentence count)

2.3.6 Punctuation Count

Count the number of punctuation marks in the text.

2.4 Exploratory Data Analysis

2.4.1 Sentiment Analysis

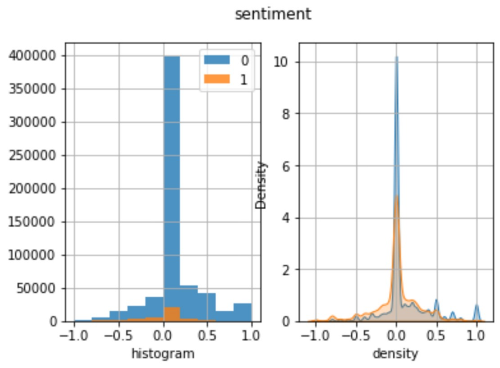


Figure 2: Density and Histogram analysis

2.4.2 Word Count

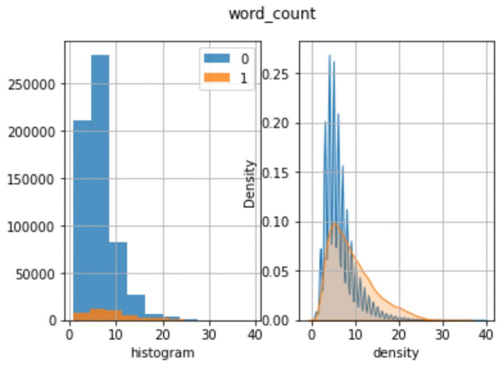


Figure 3: Density and Histogram analysis

2.4.3 Sentence Count

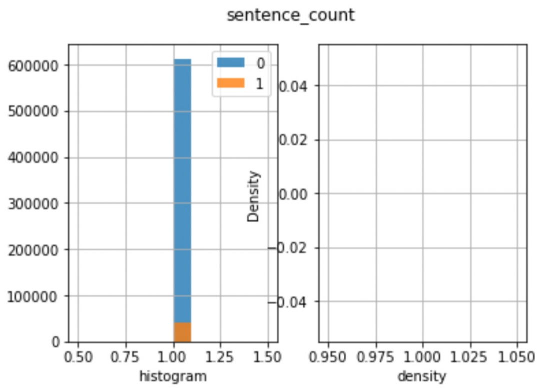


Figure 4: Density and Histogram analysis

2.4.4 Average Word Length

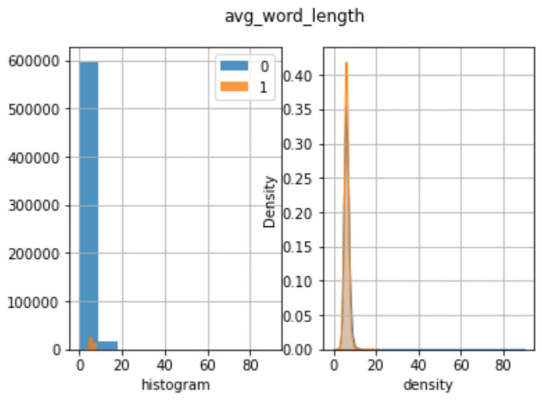


Figure 5: Density and Histogram analysis

2.4.5 Average Sentence Length

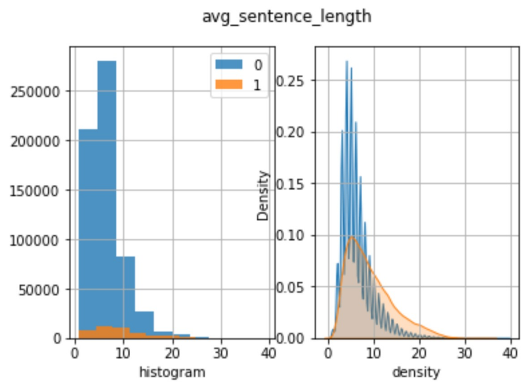


Figure 6: Density and Histogram analysis

2.4.6 Punctuation Count

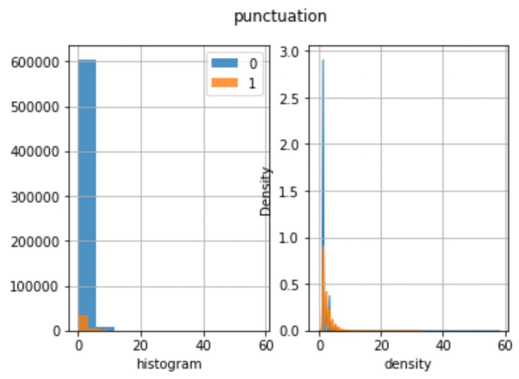


Figure 7: Density and Histogram analysis

2.5 Model Selection

We have used a pipelined model because we had both text and numeric data.

For text features we tried the following Bag of Words models to convert them into numerical data -

2.5.1 Count Vectorizer

CountVectorizer creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that particular text sample.

2.5.2 TFIDF Vectorizer

Term Frequency Inverse Document Frequency vectorizer, contains insights about the less relevant and more relevant words in a document. The importance of a word in the text is of great significance in information retrieval.

$$TF(w,d) = \frac{\text{occurence of a word w in document d}}{\text{total number of words in document d}}$$

2.5.3 Hash Vectorizer

Hashing vectorizer is a vectorizer which uses the hashing trick to find the token string name to feature integer index mapping. Conversion of text documents into matrix is done by this vectorizer where it turns the collection of documents into a sparse matrix which are holding the token occurence counts.

We sent both raw and preprocessed data to above three models and found that Count Vectorizer with raw data works best. Now this data is added to the numerical data which we got from feature engineering.

Pipelined models used on complete data-

2.5.4 Logistic Regression

2.5.5 Linear Support Vector Classifier

2.5.6 Stochastic Gradient Decent

2.5.7 Random Forest Classifier

2.5.8 AdaBoost Classifier

2.5.9 Gradient Boosting Classifier

Initially, we tried all the above classifiers without adding the numerical features. Then Logistic regression and Linear support vector classifier with Count Vectorizer gave the best results.
Later on, We have tried out all the above classifiers using GridSearchCV to find the best fitting hyper parameters for the given data. Tweaking the values of class weights has improved our classification significantly, this happened because of the imbalance in the given dataset. Logistic Regression with the following hyper parameters gave best accuracy and f1score.

class-weight = 0:0.19,1:0.81

maximum iterations = 7000

tolerance = 0.002

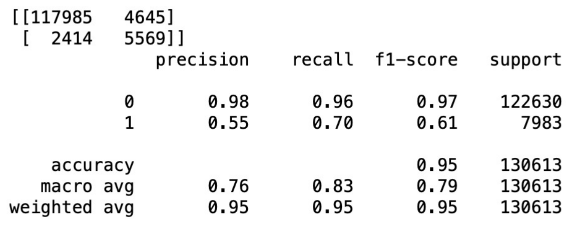
intercept scaling = 11

solver = liblinear

random state = 10

2.6 Results

Count Vectorizer with Logistic Regression Classifier worked best. This is the classification report corresponding to it when we split the train data in 80-20 ratio.



```
[[117985   4645]
 [  2414   5569]]
              precision    recall  f1-score   support

           0       0.98      0.96      0.97    122630
           1       0.55      0.70      0.61      7983

    accuracy                           0.95    130613
   macro avg       0.76      0.83      0.79    130613
weighted avg       0.95      0.95      0.95    130613
```

Figure 8: This is our classification report