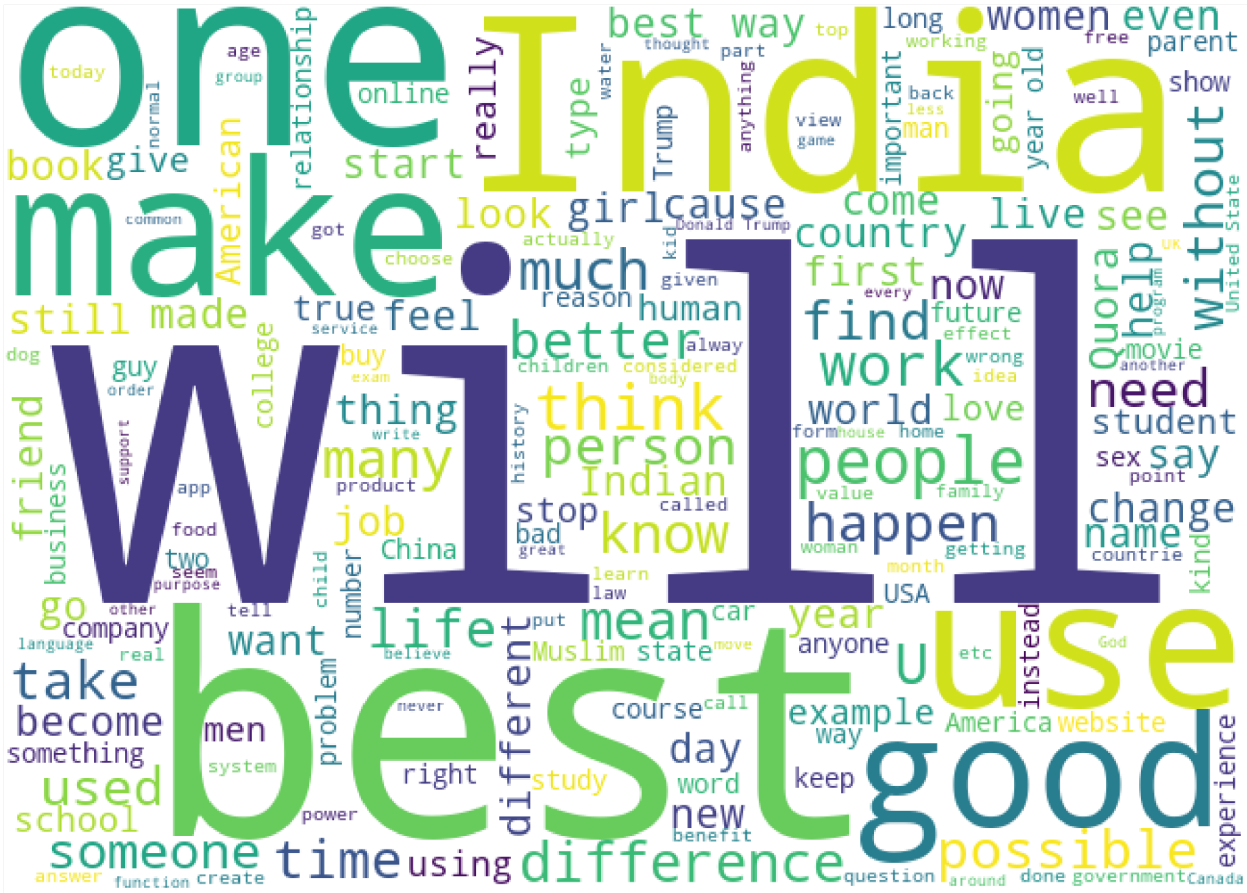
T1-21-22- AI 511 / Machine Learning

ASK REDDIT

Troll Question Detection Challenge



Word Cloud of the given training data

Submitted by:

Nandakishore S Menon (IMT2019057)

Rachna S Kedigehalli (IMT2019069)

About the challenge

The challenge asks us to detect troll questions (predict '1' if the question is a troll question and '0' if the question is not a troll question). This is to be done based on over 653000 questions given, along with information about whether or not they are troll questions.

Dataset details

The given dataset has three columns:

- *qid:* Represents the question ID which is a string.
- *question_text:* Text of a given AskReddit question, which is a string.
- *target:* Ground truth - 0 if the question is not a troll question and 1 if the question is a troll question.

The given dataset has 653061 questions, which we use to detect if a new question is a troll question.

Pre-processing

All pre-processing was done after combining the train and test dataset.

Missing values and duplicate entries

There are no missing values (null, ?, spaces) or duplicate entries.

Removing URL, HTML, brackets, digits & underscore

- URLs, HTML, brackets and digits in text do not give any information about the sentiment of the question.
- URLs were removed by using regex expressions to remove any substring containing *'https://'*.
- Any text containing HTML was removed. This was done using the **BeautifulSoup** library which parses HTML and XML.
- Digits were removed by using regex expressions.

- All occurrences of underscore in the question text were removed using the 'replace' method.

Expanding contractions

All contractions such as "can't", "won't", "it's" etc were expanded to "can not", "will not", "it is" and so on using the **contractions** module.

Tokenization, converting to lowercase, removing punctuations and stopwords

- The sentences are first tokenized, i.e sentence strings are broken down into a list of words. This was done using the nltk module.
- These words are then converted into lowercase. All punctuations are removed using regex expressions.
- Words like "a", "he" and "she" don't give out any significant information about the sentence, thus these **"stopwords"** are removed.

Lemmatization

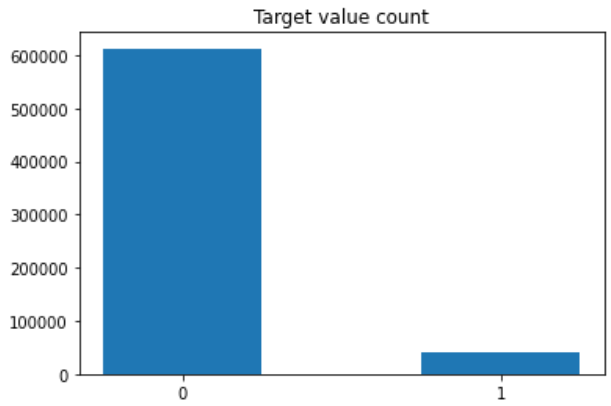
Lemmatization is used to get the base form of each word. We chose lemmatization over stemming because stemming doesn't give accurate results. Eg: for the word "scaling", lemmatization gives "scale" while stemming gives "scal".

This was done using **WordNetLemmatizer()** from the **NLTK** module. Since NLTK's word lemmatizer requires information about the part of speech (POS) of a word (in wordnet format) to lemmatize it, this was done using "nltk.tag.pos_tag" and "wordnet" from **nltk.corpus**.

Exploratory Data Analysis (EDA)

Target values

Among the questions in the training dataset, 40405 questions are troll questions and 612656 questions are not troll questions.



Language Analysis

Since the dataset has languages that are not English, like Hindi, Portuguese, etc, we thought of translating all those questions to English before training to get a better model. Though we tried a lot of libraries for this, none of them gave good results. We tried the following to translate the questions:

Pycld2

Though this module completed execution in a comparatively less amount of time, it predicted the language of a lot of questions in the dataset as unknown 'un'. So, it wasn't used.

Langid

langid did not complete execution even after running for hours.

Langdetect

langdetect did not complete execution even after running for hours.

Google translate

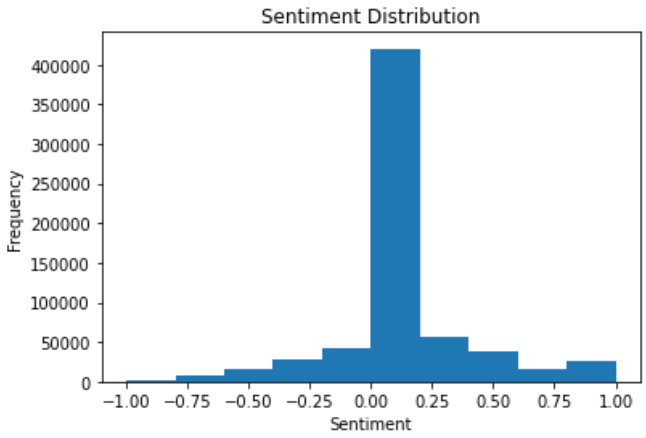
googletrans did not complete execution even after running for hours.
google_trans_new has an error in its library, which is still an open issue on Github. So, it couldn't be used.

itranslate

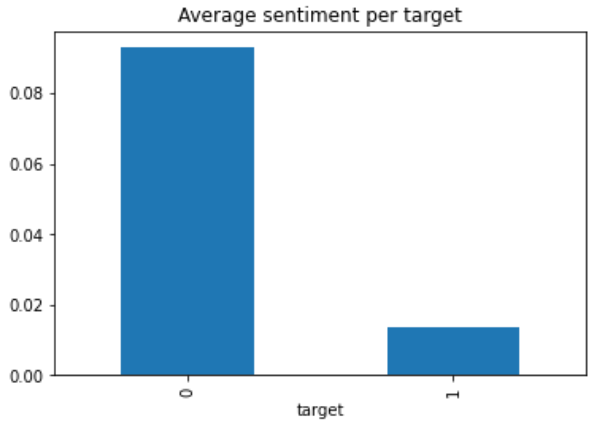
This module corrects the error of google translate, even though it is not as efficient as google translate.
But, it blocked IP after a few requests. So it couldn't be used, given the size of the dataset.

Sentiment Analysis

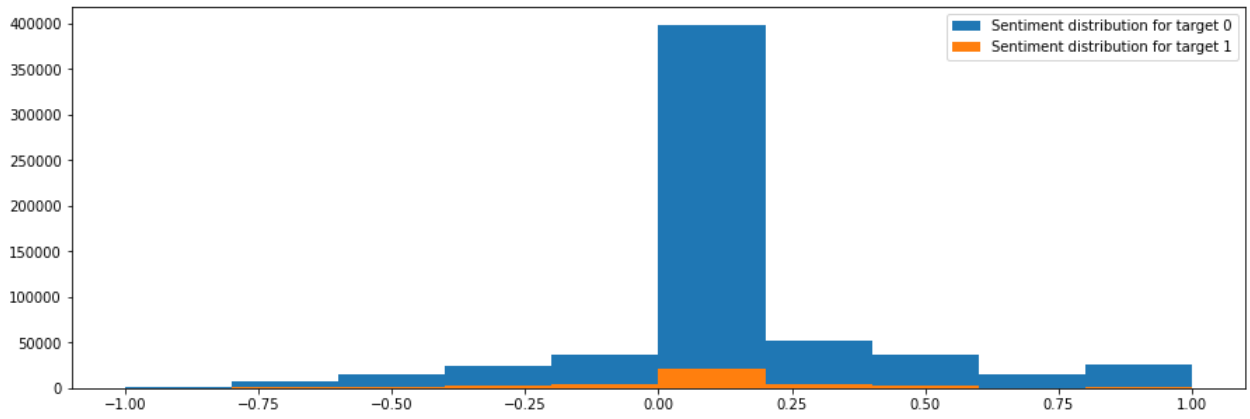
Sentiment analysis is done to get a sense of the tone/sentiment (from -1 for negative to 1 for positive tone) of the question text. This was done using the **TextBlob** library.



Histogram of sentiment for the entire dataset



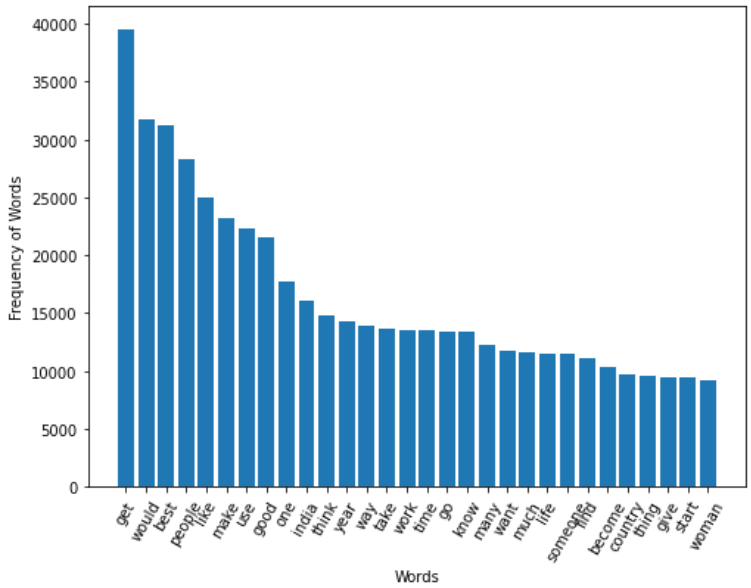
Average sentiment value for non-troll and troll questions



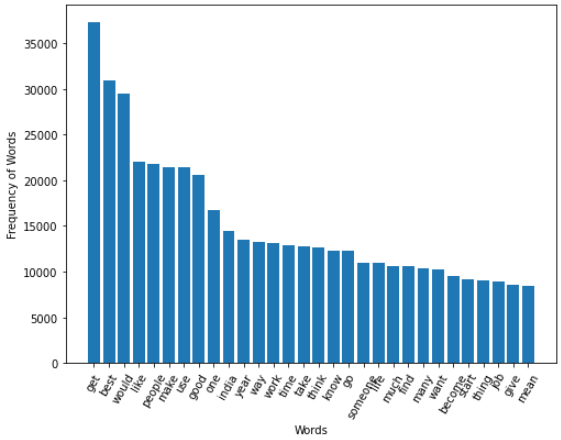
Histogram of sentiment for non-troll and troll questions

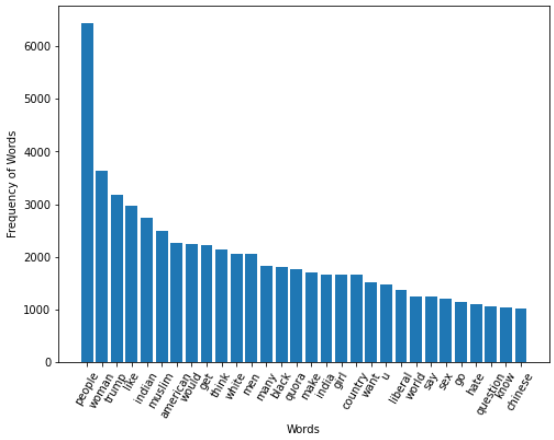
Word Frequency Analysis

We used **FreqDist** from **nltk.probability** to get a sense of the most common words in the dataset. We also used **WordCloud** to get the word cloud of all the words in the dataset.



30 most common words in the dataset





30 most common words in the non-troll questions 30 most common words in the troll questions

Training

Count Vectorizer

CountVectorizer from **sklearn.feature_extraction.text** module was used to convert the text into vectors. It performs something similar to one-hot encoding. Count vectorizer is fit on the entire data, so that the vocabulary of the model contains all possible words. Then the question text from train and test data are individually transformed. Countvectorizer has been used with ngrams=(1,3) for training the model which led to our highest score.

TF-IDF

Similar to CountVectorizer, TF-IDF converts text data to vectors. The advantage it has over using raw frequencies is that it reduces the bias due to tokens whose frequency is more. However, TF_IDF led to a lower score compared to CountVectorizer for the given data.

Word2Vec

Word2Vec, when trained on the vocabulary of the given data, failed to finish the vector transform even after 12 hrs. Thus we abandoned it. We also tried using the word embedding from Google and the same issue persisted.

Models

Gradient Boost

This took too long to train. It didn't complete execution even after 4 hours.

AdaBoost

This took too long to train. It didn't complete execution even after 4 hours.

XGBoost

This gave a very low score on Kaggle (0.16566 public score).

Linear Support Vector Classifier (LinearSVC)

This took too long to train. It didn't complete execution even after 3 hours.

SGDClassifier

This implements linear classifiers with stochastic gradient descent (SGD) learning.

Decision Tree and Random Forest

Both models gave very less score on Kaggle (less than 0.2)

Gaussian Naive Bayes

This gave decent results, but not better than Logistic Regression.

Multinomial Naive Bayes

We experimented with different alpha values. Similar to Gaussian Naive Bayes, this also gave decent results. But it was not better than Logistic Regression.

Multilayer Perceptron

This took a lot of time to train, but gave decent results (At one point, with 35 iterations, it gave our highest score on Kaggle).

But Logistic Regression, with some tuning, was able to perform better.

Perceptron

Though this took minimal time to train, it gave very poor results locally.

Logistic Regression

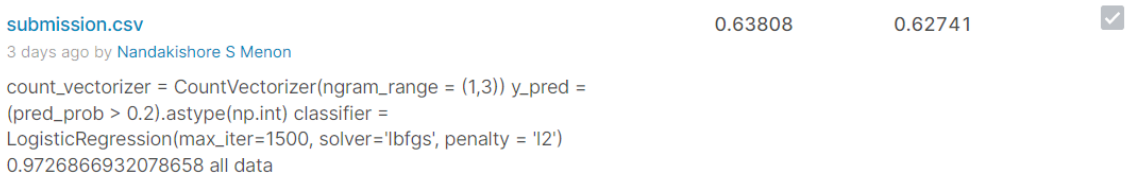
Logistic regression along with GridSearchCV gave good results locally, but did not give a good score on Kaggle.

Since GridsearchCV was overfitting the training data, we tried changing parameters manually. After a lot of tries, we got the best score with “lbfgs” solver and “l2” penalty over 1500 iterations.

Final Model

CountVectorizer along with LogisticRegression is our final model.

```
CountVectorizer(ngram=(1,3))
LogisticRegression(max_iter=1500, solver='lbfgs', penalty = 'l2')
```



To get better results, we trained the model on the entire data, rather than on a fraction of data.

Note: For all the models, we tried different threshold values for **predict_proba**. We used the one that gave the best results for Kaggle submissions.