AI 511 - Machine Learning
Prof. Neelam Sinha
Prof. Dinesh Babu

24 Dec 2021

Final Project Report

AskReddit Troll Question Detection

—

Team: dev

Shrey Tripathi (IMT2019084)
Gunin Jain (IMT2019034)

Project Description

Reddit is a less well-known social media website built on anonymity and somewhat democratic principles. The probability of a post/comment "going viral" on Reddit is the same irrespective of whether a person is a celebrity or if they're a commoner. 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. Their official description: r/AskReddit is the place to ask and answer thought-provoking questions.

Of course, not all questions here are thought-provoking. Some are very obvious "troll" questions.

The Wikipedia entry for Internet trolls/ trolling:

> In internet slang, a troll is a person who posts inflammatory, insincere, digressive, extraneous, or off-topic messages in an online community (such as social media (Twitter, Facebook, Instagram, etc.), a newsgroup, forum, chat room, or blog), with the intent of provoking readers into displaying emotional responses or manipulating others' perception. This is typically for the troll's amusement, or to achieve a specific result such as disrupting a rival's online activities or manipulating a political process.

The moderators of AskReddit continuously strive to remove such questions, however, it is not possible to manually inspect every question and do this -- thousands of questions are added every day.

This project aims to create a model capable of automatically detecting troll questions so that they can be flagged and removed.

Data

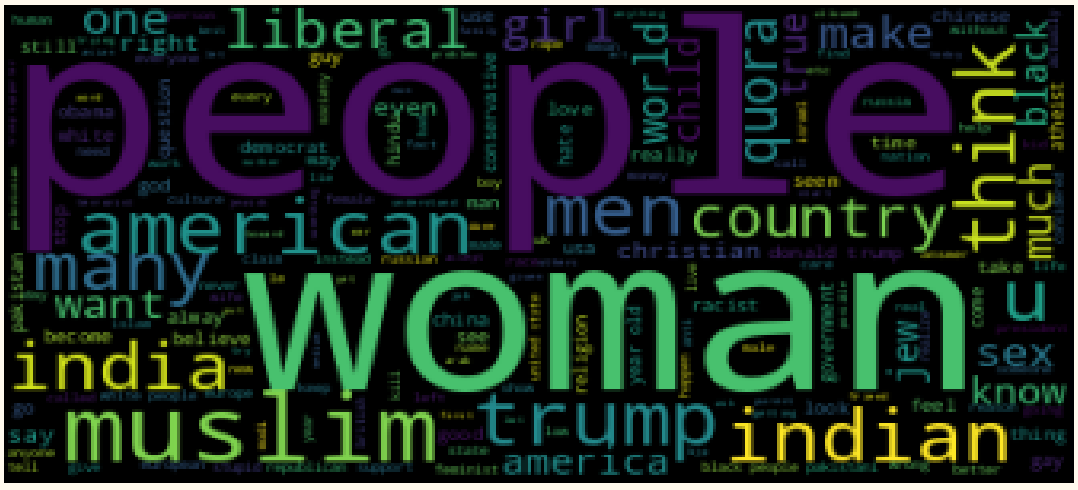
The train and test datasets were provided to us.

Train dataset had three columns: “qid”, “question_text”, “target”. “qid” is an ID string, “question_text” is the text of a given AskReddit question, and “target” is the ground truth: 0 if a question is not a troll question and 1 if the question IS a troll question. The test dataset had two columns: “qid” and “question_text” and we had to predict the target in the form of a CSV file with the “qid” and the corresponding predicted target values.

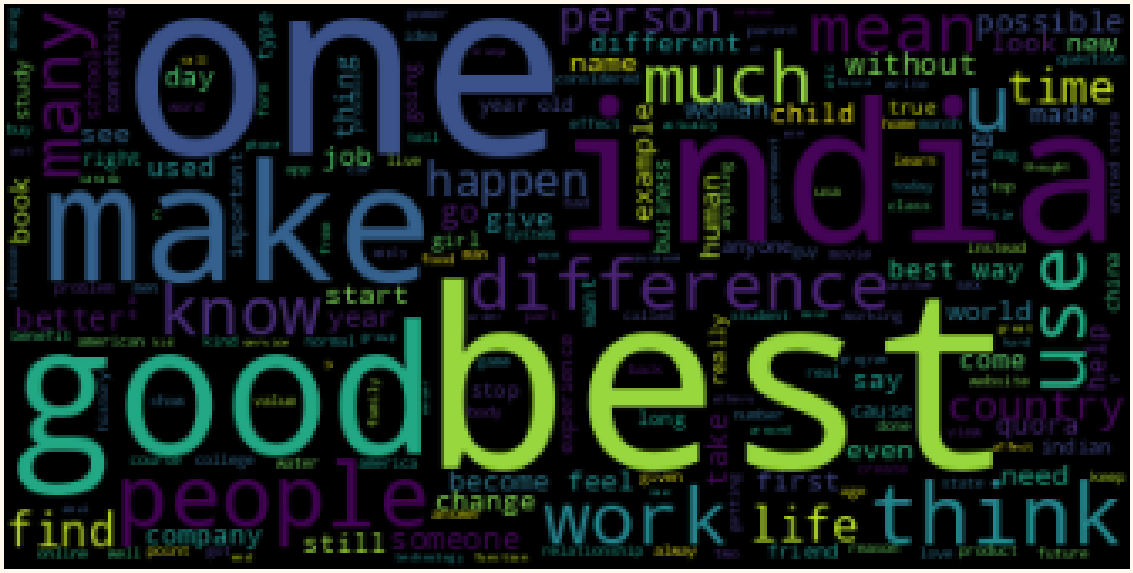
The training and testing datasets both had 653061 data points.

Exploratory Data Analysis

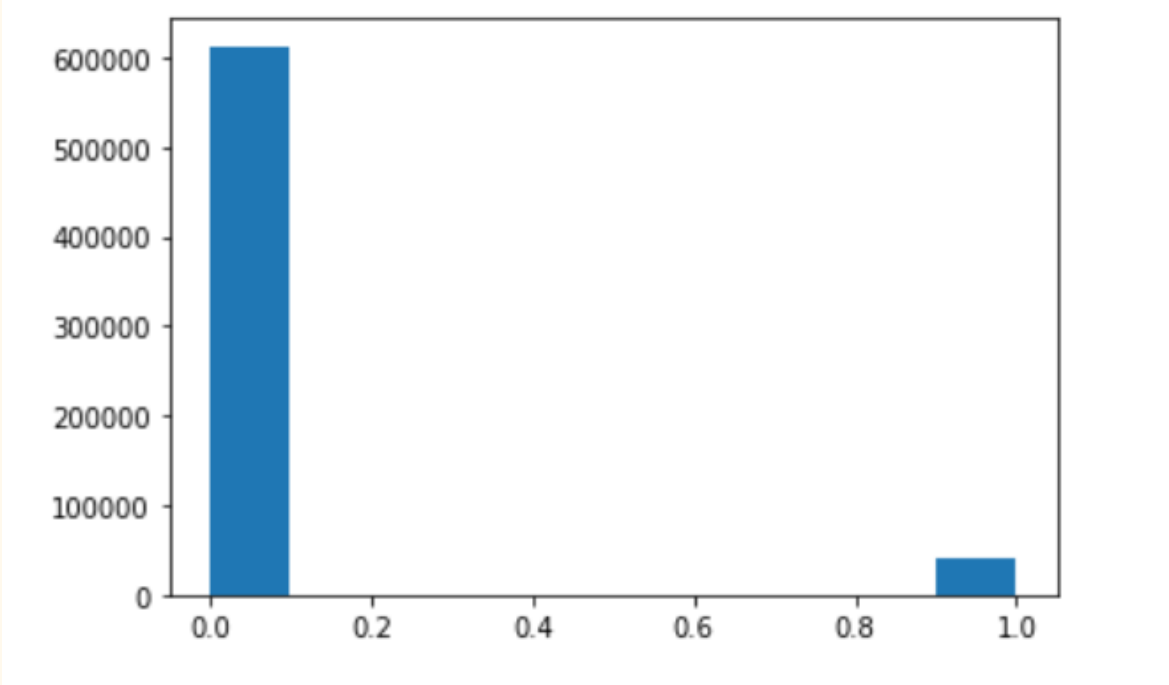
Wordcloud denoting frequently occurring words in troll questions:



Wordcloud denoting frequently occurring words in non-troll questions:



Distribution of troll vs non-troll questions:

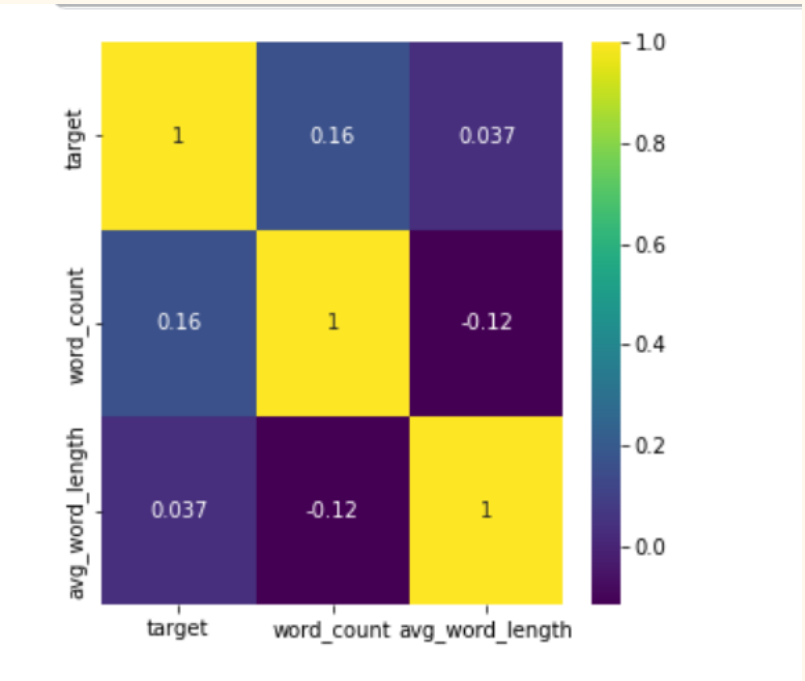


Feature Selection and Engineering

We generated the following features:

1. Number of words in a question
2. The average length of each word in the question

The correlation matrix between the target value and the above features is as follows:



Preprocessing

After EDA and feature engineering, the data preprocessing steps included methods and ways to remove noise and redundancies in the data. Cleaning the data is an important step in NLP, as it helps learn the models better.

Removing non-English questions

Removed non-English questions using the “langid” python module, since words/characters in a language may differ in meaning from the English language.

Converting the text to lower case

Since a question in uppercase or lowercase does not change its meaning. Hence, we converted all questions to lowercase to avoid the same questions in different cases to have different meanings.

Expanding contractions

Expanded contractions using a contractions dictionary from Wikipedia, since, for example, “aren’t” is equivalent to “are not”, but since both are not the same lexically, this again introduces redundancy.

Removing punctuation/special characters

Punctuation and numerical data were not important to the models used, and hence, were removed. Basically, we only needed English words to train the models, and hence, anything other than the 26 English characters was removed.

Removing extra spaces/tabs

This was done to basically clean the data. Extra spaces/tabs in data don’t provide any value to the training models.

Stop words removal

Words like *this, an, is,* which are inevitably present in almost every sentence do not make any contribution to the meaning of the sentence. We used the list of stop words from the NLTK module to identify and remove the stop words from the questions.

Tokenization

We “tokenized” the questions, that is, created a list of words that denotes every question. Similar to what the “split” function does in python. We used the “word_tokenize” module of the NLTK package for tokenization.

Stemming

Stemming is the process of converting the words of a sentence into their stem, basically removing the “suffix”. This is necessary because a stem may have various versions after adding different suffixes, which mean the same. We used the PorterStemmer that NLTK provides. PorterStemmer applies a set of five sequential rules (also called phases) to determine common suffixes from sentences.
Eg: amusing, amusement, amused -> amus

Lemmatization

Lemmatization is the process of converting any word to its base form. We used the WordNetLemmatizer that NLTK provides. Lemmatization converts the words based on the context of the word in a sentence(part of speech).

For eg: studying, studied, studies -> study

Vectorizers

1. **CountVectorizer**

   It uses the bag of words model to convert arbitrary text into fixed-length vectors by counting how many times each word appears.

   We used the CountVectorizer provided by SkLearn to implement the above technique.

2. **Tf- Idf Vectorizer**

$$TFIDF\ score\ for\ term\ i\ in\ document\ j = TF(i,j) * IDF(i)$$

where

$IDF = Inverse\ Document\ Frequency$

$TF = Term\ Frequency$

$$TF(i,j) = \frac{\text{Term i frequency in document j}}{\text{Total words in document j}}$$

$$IDF(i) = \log_2 \left(\frac{\text{Total documents}}{\text{documents with term i}}\right)$$

and

$t = Term$

$j = Document$

   The Tf-idf score for each word was calculated as given above to generate the feature vectors for each sample question.

   The Tf-idf technique is good as it gives importance to rarely occurring words.

We used the *TfIdfVectorizer* provided by SkLearn.

Word Embeddings

Word2Vec

This technique is used to represent each word as a vector. The vectors constructed by this technique are based on the context of each word in a sentence.

It is a combination of the CBOW(Continuous bag of words) and Skip-gram technique.

The *ngram* range i.e. the number of words to be considered for context can be given by us.

We used the Word2Vec module provided by *gensim* to build a vocabulary from the given corpus.

This technique generated 100-dimensional vectors for each word in a question which were then added to get a single 100-dimensional vector for each question.

Balancing the dataset

The random oversampler provided by *imblearn* was used to balance the dataset as a majority of the samples were of the non-troll category.

The random oversampler added duplicate samples of the minority class to the dataset in order to make the count of troll and non-troll questions equal.

SMOTEToMek was another technique (based on generating synthetic samples of the minority class rather than duplicates) which we tried but was later discarded due to it taking too much time to balance the given data.

Top 3 Approaches

Approach 1: Logistic Regression

We used Logistic Regression with no preprocessing. After vectorization of the question text (using *CountVectorizer*) with hyperparameter tuning (*max_features = 150000*) and learning the vocabulary dictionary using *fit_transform*, we split the training dataset into train and test data. The model used was SkLearn's *LogisticRegression()* model with the *liblinear* solver.

The Kaggle score obtained was: 0.54451

The classification report on the validation data is as follows:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.96	0.99	0.98	92007
1	0.69	0.44	0.54	5953
accuracy			0.95	97960
macro avg	0.83	0.71	0.76	97960
weighted avg	0.95	0.95	0.95	97960

Approach 2: Logistic Regression

This approach was the same as the first one, with just the parameter difference in the *CountVectorizer,* with the parameter *max_features = 12000.*

The Kaggle score obtained was: 0.53494

The classification report on the validation data is as follows:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.96	0.99	0.98	92007
1	0.68	0.43	0.53	5953
accuracy			0.95	97960
macro avg	0.82	0.71	0.75	97960
weighted avg	0.95	0.95	0.95	97960

Approach 3: Multi-Layer Perceptron

After splitting the train data into train data and test data, we used the *X_train* and *y_train* to fit the SkLearn's *MLPClassifier()* and then used it for prediction on *X_test*. *MLPClassifier* trains iteratively since at each time step the partial derivatives of the loss function with respect to the

model parameters are computed to update the parameters. We tried hyperparameter tuning, but couldn't achieve much difference or a much better F1-score. We also tried changing/adding/removing some preprocessing steps, but couldn't improve the score much.

Preprocessing steps	**Tuned parameter (*hidden_layer_sizes*)**	**F1-score on Kaggle**
- Tokenization - Stemming - Converting text to lowercase - Stop words removal - Countvectorizer (4000 features)	(5, 2)	0.53793
- Tokenization - Stemming - Converting text to lowercase - Stop words removal - Word2Vec word embedding	(5, 2)	0.41217
- Tokenization - Stemming - Converting text to lowercase - Stop words removal - Word2Vec word embedding	(20,)	0.45311

The classification report on the validation data (for the best submission) is as follows:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     92007
           1       0.63      0.44      0.52      5953

    accuracy                           0.95     97960
   macro avg       0.80      0.71      0.74     97960
weighted avg       0.94      0.95      0.95     97960
```

Other Methods and Approaches

All the below approaches are applied after following the pre-processing steps as mentioned above.

Logistic Regression

We used logistic regression on the vectors provided by :

1. CountVectorizer
2. Tf-Idf Vectorizer
3. Word2Vec word embedding

The f1 scores obtained on train data and validation data are as follows:

CountVectorizer

```
print('Best params:', gscv.best_params_)
print('Best f1 score on train data:',gscv.best_score_)
print('f1 score on test data:',gscv.score(X_test_cv,y_test_cv))
```

```
Best params: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
Best f1 score on train data: 0.8462051713936278
f1 score on test data: 0.4306656101426307
```

Tf-Idf Vectorizer

```
print('Best params:', gscv.best_params_)
print('Best f1 score:',gscv.best_score_)
print('f1 score on test data:',gscv.score(X_test_tv,y_test_tv))
```

```
Best params: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
Best f1 score: 0.849005463659587
f1 score on test data: 0.4114882506527415
```

Word2Vec word embeddings

```
print('Best params:', gscv.best_params_)
print('Best f1 score:',gscv.best_score_)
print('f1 score on test data:',gscv.score(X_test_w2v,y_test_w2v))
```

```
Best params: {'C': 0.001, 'penalty': 'l2', 'solver': 'liblinear'}
Best f1 score: 0.8524775628556404
f1 score on test data: 0.4396206121052117
```

The train score is high while the validation score is low, showing clearly that the model is overfitting the data.

We observed that generally, vectors generated by Word2Vec word embeddings gave a better score for both test and validation data. We have used the Word2Vec embeddings generated vectors for some of the approaches discussed below.

Decision Tree Classifier with Word2Vec vectors

```
print('Best params:', gscv.best_params_)
print('Best f1 score:',gscv.best_score_)
print('f1 score on test data:',gscv.score(X_test_w2v,y_test_w2v))
```

```
Best params: {'max_depth': 10, 'min_impurity_decrease': 0.1}
Best f1 score: 0.745513168686671
f1 score on test data: 0.25838926174496646
```

Random Forest Classifier with CountVectorizer generated vectors

```
print('Best params:', gscv.best_params_)
print('Best f1 score:',gscv.best_score_)
print('f1 score on test data:',gscv.score(X_test_cv,y_test_cv))
```

```
Best params: {'criterion': 'gini', 'max_depth': 10, 'min_impurity_decrease': 0.2, 'n_estimators': 20}
Best f1 score: 0.5333333333333333
f1 score on test data: 0.11640768648375815
```

Gaussian Naive Bayes with Word2Vec vectors

```
print(classification_report(y_test_w2v, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.84      0.90    122541
         1.0       0.20      0.59      0.30      8072

    accuracy                           0.83    130613
   macro avg       0.58      0.72      0.60    130613
weighted avg       0.92      0.83      0.87    130613
```

Gradient Boosting Classifier with Tf-Idf vectors

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     92007
           1       0.61      0.43      0.50      5953

    accuracy                           0.95     97960
   macro avg       0.79      0.70      0.74     97960
weighted avg       0.94      0.95      0.94     97960
```

Hyperparameter Tuning for Gradient Boost:

N_ESTIMATORS	LEARNING RATE	MAX_DEPTH	RANDOM_STATE	MAX_FEATURES	F1-score (0)	F1-score (1)	Accuracy
500	1.5	7	0	3000	0.97	0.37	0.94
500	1.5	10	0	4000	0.95	0.31	0.91
500	1.5	1	0	4000	0.95	0.00	0.95

500	0.5	7	0	4000	0.97	0.50	0.95
500	0.5	7	1	4000	0.97	0.50	0.95
500	0.25	7	0	4000	0.97	0.49	0.95

XGBoost with Word2Vec vectors

```
from sklearn.metrics import classification_report
print(classification_report(y_test_w2v, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.89      0.94    122541
         1.0       0.33      0.83      0.47      8072

    accuracy                           0.88    130613
   macro avg       0.66      0.86      0.70    130613
weighted avg       0.95      0.88      0.91    130613
```

Perceptron with Word2Vec vectors

```
from sklearn.metrics import classification_report
print(classification_report(y_test_w2v, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.87      0.92    122541
         1.0       0.27      0.74      0.39      8072

    accuracy                           0.86    130613
   macro avg       0.62      0.80      0.66    130613
weighted avg       0.94      0.86      0.89    130613
```

Adaboost with Word2Vec vectors

```
print(y_pred)
print(classification_report(y_test_w2v, y_pred))
```

```
[0. 1. 0. ... 0. 0. 0.]
              precision    recall  f1-score   support

         0.0       0.98      0.78      0.87    122541
         1.0       0.19      0.80      0.31      8072

    accuracy                           0.78    130613
   macro avg       0.59      0.79      0.59    130613
weighted avg       0.93      0.78      0.84    130613
```

Ridge Classifier with Word2Vec vectors

```
print(y_pred)
from sklearn.metrics import classification_report
print(classification_report(y_test_w2v, y_pred))
```

```
[0. 1. 0. ... 0. 0. 0.]
              precision    recall  f1-score   support

         0.0       0.99      0.89      0.93    122541
         1.0       0.32      0.80      0.46      8072

    accuracy                           0.88    130613
   macro avg       0.65      0.84      0.70    130613
weighted avg       0.94      0.88      0.90    130613
```

AWS Usage

We used AWS just for running the Jupyter notebooks, in cases of high RAM or CPU usage on Kaggle, Colab, and localhost. Apart from this, using AWS didn't provide any computational advantage over the other options (Kaggle, Colab, etc.)

Required Libraries

- numpy
- pandas
- re
- sklearn
- nltk
- wordcloud
- matplotlib
- gensim
- imblearn
- xgboost

References

1. SkLearn documentation
2. Text Classification: Tips and Tricks
3. Gensim documentation
4. Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit