Machine Learning Project Report

Task : AskReddit Troll question Detection Challenge

By Team **NAMELESS,**
Satwik Samayamantry (IMT2019077)
Manohar Suggula (IMT2019025)

<u>Project Description</u>:

Reddit is a social website which is built on anonymity and some democratic principles. In this website, your post can go viral irrespective of whether you are a celebrity or commoner. There are few rules enforced by reddit moderators. The admins will step in if there is any controversy or any rule being violated.
Their official description is "AskReddit is the place to ask and answer thought-provoking questions." But, there will be few troll questions also.

The Wikipedia entry for Internet trolls/ trolling:
"" In internet slang, a troll is a person who posts inflammatory, insincere, digressive,[1] 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,[2] 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. ""
So, the reddit moderators continuously strive to remove such questions although it is impossible to do it manually since there will be thousands of questions daily.

So, our job is to create a model capable of automatically detecting troll questions so that they can be flagged and removed.

<u>Dataset:</u>

- The dataset contains 2 files train.csv and test.csv.
- train.csv contains 3 columns namely
    - qid (id of the question).
    - question_text (a statement which is to be detected whether a troll or not).
    - target (1 if the statement is troll and 0 if it's not).
- test.csv contains 2 columns namely
    - qid (id of the question).
    - question_text (a statement which is to be detected whether a troll or not).

EDA and Data Pre-processing:

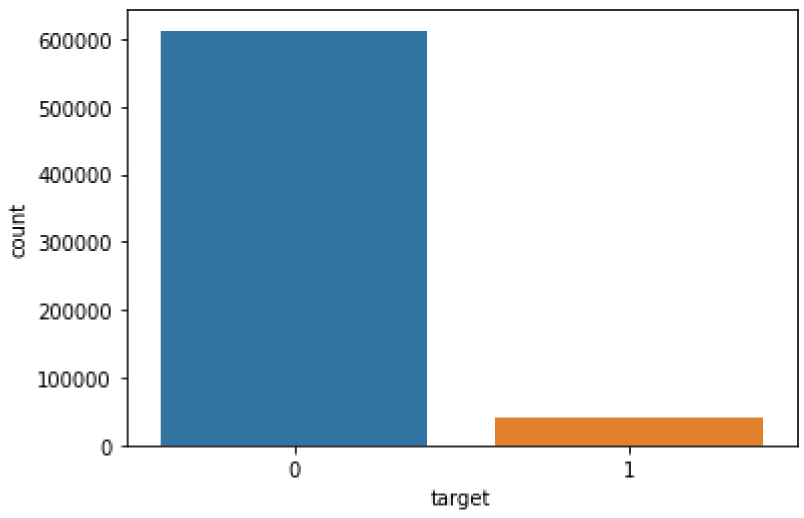
For Train Data

Function	**Conclusion**
train_data.head()	The data has three columns namely qid, questions and target. It has categorical data. So, no need of checking the outlier, correlation, doing normalization, etc.,
train_data.shape()	(653061, 3)
train_data.isna().sum()	No null values
(train_data == "?").sum()	No '?' marks
train_data.duplicated().sum()	No duplicate values
train_data['target'].value_counts()	There is a big difference between no.of 0's and 1's. So, we need to take care of it appropriately.

For Test data

Function	**Conclusion**
test_data.shape()	(653061, 2)
test_data.isna().sum()	No null values
(test_data == "?").sum()	No '?' marks
test_data.duplicated().sum()	No duplicate values

We used sns countplot for the target column in train data



For here on we combined the train and test data and applied the NLP techniques on the combined data.

We implemented the following NLP techniques:

- Converted the data to lowercase.
- Removed punctuations.
- Tokenization - Splits a sentence into a list of words.
- Lemmatization - converts a word into simple forms like plural→singular, past tense→present tense, etc.,
- Stemming - converts a word into some simple forms removing '-ing' in words, replacing 'y' with 'i' at end etc.,

But, we got better results without applying these NLP techniques.

So we continued without carrying any of the NLP preprocessing techniques as we also thought that they would be helpful in classifying spam questions as they indicate tone and context of the writer.

Then we implemented the CountVectorizer. We gave the hyper parameter ngram_range different values. We got better results when we gave ngram_range = (1,4). Above that value(i.e., (1,5), (1,6) etc.,), the session crashed.

The CountVectorizer returns a sparse matrix of token counts. By default it gives the number of features equal to the vocabulary size found by analyzing the data.

Feature Engineering:

- First of all, we have to vectorize the available data for better usage of data. Hence we tried using Word2vec to get a vector space but due to large data the computation is not able to complete within the available resources and sessions got crashed.
- Then we used CountVectorizer and TfidfVectorizer, where we found that CountVectorizer gave better results over TfidfVectorizer. So, we fixed up with using CountVectorizer to vectorize data.
- After CountVectorizer, we are left with lakhs of features which increases the complexity of fitting the model. So, we decide to reduce the features by picking the best out of them.
- Hence we first tried PCA. But PCA works only for dense data but our data is sparse (returned by CountVectorizer), so we need to convert it to a dense matrix. When we tried converting the sparse matrix to dense matrix using todense() and toarray(), it executed for a long time and the session got crashed.
- Then we used TruncatedSVD() to extract better features from the existing sparse matrix. This didn't improve the accuracy significantly as a result we ignored it.
- Hence, we took all features into consideration for final model training.
- Also, we tried implementing the models which accept the sparse matrix. For other models we reduced the number of features randomly and trained them.

Model Selection:

1. **Logistic Regression:** We got the best results for this model. This model ran successfully. We used many combinations of hyper parameters. We got good local scores for Logistic Regression. So, it made us stick to this model and tune different hyper parameter combinations.
2. **GaussianNB:** It takes only dense matrices as input. But we have a sparse matrix as we used a CountVectorizer on the data. Since, todense() and toarray() didn't work(due to large data), we tried reducing the number of features in CountVectorizer and trained the model. But, we got very little accuracy.

3. **SVC:** For SVC, the model fitting function executed so long (~ 2 hr) and eventually the session crashed.
4. **AdaBoostClassifier:** Same as SVC, it executed for a long time and the session crashed.
5. **BaggingClassifier:** We didn't get better results than previous models.
6. **DecisionTreeClassifier:** We didn't get better results than previous models.

So, Out of all models, We fixed it with LogisticRegression as it gave the best results when compared to remaining models.

We used Stratified train_test_split to split the data so that the proportion of values in the sample produced will be the same as the proportion of values provided to parameter stratify. Using this we are able to achieve better estimation of local score which is comparable to kaggle score.

The below table includes the trials which are submitted in kaggle and F1_Score corresponds to the private score in the kaggle contest.
The table contains very limited entries which don't show all the trials we've done.
We submitted the entries in kaggle which beat the benchmark in our local testing.

PreProcessing	**Model**	**HyperParameters**	**F1_Score**
Lowering, Removing punctuation and stopwords, Lemmatization CountVectorizer	LogisticRegression	CountVectorizer : Max_features = 500 LogisticRegression: max_iter=10000	0.30638

Lowering, Removing punctuation and stopwords, Lemmatization CountVectorizer	LogisticRegression	LogisticRegression: max_iter=10000	0.50246
CountVectorizer	LogisticRegression	LogisticRegression: max_iter=10000	0.54427
CountVectorizer	LogisticRegression	LogisticRegression: class_weight='balanced', solver = 'liblinear', max_iter=1000	0.55465
CountVectorizer	LogisticRegression	LogisticRegression: class_weight= {0:1,1:3}, solver = 'liblinear', max_iter=1000	0.61594
TfidfVectorizer	LogisticRegression	LogisticRegression:class_weight= {0:1,1:3}, solver = 'liblinear', max_iter=1000	0.61473
CountVectorizer	LogisticRegression	LogisticRegression: class_weight= {0:1,1:4}, solver = 'liblinear', max_iter=1000	0.61620
CountVectorizer	LogisticRegression	LogisticRegression: class_weight= {0:1,1:5}, solver = 'liblinear', max_iter=1000	0.61179
CountVectorizer	LogisticRegression	LogisticRegression: C=0.1, class_weight=	0.62562

		{0:1,1:4}, solver = 'liblinear', max_iter=1000	
CountVectorizer	LogisticRegression	LogisticRegression: C=0.1, class_weight= {0:1,1:4}, solver = 'liblinear', max_iter=1000	0.62668
CountVectorizer	LogisticRegression	LogisticRegression: C=0.2, class_weight= {0:1,1:4}, solver = 'liblinear', max_iter=1000, penalty = 'l1'	0.62483
CountVectorizer	LogisticRegression	LogisticRegression: C=0.18,class_weight= {0:1,1:4}, solver = 'sag', max_iter=1000,penalty='l2 '	0.62694
CountVectorizer	LogisticRegression	LogisticRegression: C=0.18,class_weight= {0:1,1:4}, solver = 'saga', max_iter=1000,penalty='l2 '	0.62699

CountVectorizer	LogisticRegression (did fit on 80% of data)	LogisticRegression: C=0.18,class_weight= {0:1,1:4}, solver = 'saga', max_iter=1000,penalty='l2 '	0.62152
CountVectorizer	LogisticRegression	LogisticRegression: C=0.77, class_weight= {0:0.25,1:1}, solver = 'liblinear', max_iter=10000, penalty = 'l1'	0.62483
CountVectorizer	LogisticRegression	CountVectorizer : (ngram_range=(1,3) LogisticRegression: C=0.77, class_weight= {0:0.25,1:1}, solver = 'liblinear', max_iter=10000, penalty = 'l1'	0.63543

CountVectorizer	LogisticRegression	CountVectorizer : (ngram_range=(1,3) LogisticRegression: C=0.2, class_weight= {0:1,1:4}, solver = 'liblinear', max_iter=10000, penalty = 'l1'	0.63489
CountVectorizer	LogisticRegression	CountVectorizer : (ngram_range=(1,4) LogisticRegression: C=0.77, class_weight= {0:0.25,1:1}, solver = 'liblinear', max_iter=10000, penalty = 'l1'	0.63579

HyperParameter Tuning :

For hyperparameter tuning, we've used **GridSearchCV**. It took so long to compute the best parameters. But at the end we found that the hyperparameters which came from GridSearchCV are not giving better accuracy than the hyperparameters which we fixed up manually on a trial and error basis.

Result :

The best result is achieved by applying CountVectorizer(ngram_range=(1,4)) and using LogisticRegression(C=0.77, class_weight= {0:0.25,1:1}, solver = 'liblinear', max_iter=10000, penalty = 'l1') for model fitting.

References :

1. https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/
2. https://towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b