AI 511 Machine Learning December 25, 2021

Project Report

Divyam Agrawal (IMT2019028) and Gagan Agarwal (IMT2019031)

1 Introduction to dataset

Importing the dataset was the first step, followed by checking for null attributes and duplicate rows.
Then we generated some dataset visualisations. The following visualisation approaches were used:

- Word cloud
- Word count plot
- Word count plot for bigram words

2 Preprocessing

In terms of preprocessing, we conducted the following:

- Removed ID column
- Checked for Null values → Found none
- Checked for duplicate rows → Found none
- Performed Feature Engineering to generate extra features
- Performed Data Cleaning

To aid the models in making more informed judgments, we created the following supplementary features:

- Word count
- Character count
- Sentence count
- Average Word length
- Average Sentence length
- Capital character count

- Capital character vs Total Character count
- Number Unique Words
- Word count vs Unique words

Later on, we attempted to use these traits, however they were ineffective in raising the models' f1 score.

We cleaned the data in the following ways:

- Lower case conversion
- Cleaning contractions
- Removing all punctuations
- Removing all single characters
- Removing multiple spaces
- Tokenization
- Removing stopwords
- Stemming or Lemmatization (one at a time)

However, We received the highest f1 score without performing any data preprocessing in our submission.

3 Feature Extraction

Machine learning algorithms cannot operate directly on raw text. To turn text into a vector of numbers, we employed the following feature extraction techniques:

1. **Bag of Words** - CountVectorizer:
   A bag-of-words is a representation of text that describes the occurrence of words within a document.
2. **TFIDF** - TfidfVectorizer:
   Term Frequency - Inverse Document Frequency (TFIDF) computes a score for each word to signify its importance in the document and corpus.

We have tried these both vectorizers with fine tuning ngram_range, min_df, max_df and max_features hyperparamters.

The hyperparameters min df, max df, and max features created features in less time, however they were unable to improve the f1 score.

Increasing the ngram range resulted in a higher f1 score because it could recognise false negatives more correctly, but it took longer than smaller ngram ranges.

CountVectorizer's ngram range = (1, 4) provided our best submission.

3. **Word2Vec**:
   We tried to use Word2Vec from the gensim library. We tried using 2 pretrained models: GoogleNews-vectors-negative300 and glove-twitter-25.

   We looped through each word a document, found it's corresponding weight vector and then took the mean of the vector weights and asssigned it to the word.
   We then split the data into test and training and applied AdaBoostClassifier.

   But this method didn't work out for us as it was taking extremely long and never finished running both locally and on colab.

4 Resampling

Standard machine learning algorithms are biased towards the majority class and tend to neglect the minority class. They tend to anticipate just the majority class, resulting in significant misclassification of the minority class in compared to the majority class. Even here, there is a significant class imbalance since there are many more data with class label 0 than data with class label 1.

We used a variety of typical resampling strategies in the hopes of getting better results:

1. Near Miss Version 1
2. Random Under Sampling
3. Random Over sampling + Under sampling
4. TomekLinks
5. Condensed Nearest Neighbour
6. Edited Nearest Neighbours
7. SMOTE
8. SMOTE + Random Undersampling

We believed that by using these strategies, we would be able to attain better outcomes, however most of these models could not provide us with any better results than previously. Some of these took several hours to complete. As a result, we had to interrupt execution for some of these.

SMOTE + Random Undersampling produced the closest results to the score without using any resampling techniques. As a result, in the hopes of improving the score, we proceeded to fine-tune the sampling method of the SMOTE and Random undersampling processes. However, we did not employ this process in our best submission.

5 Models

To get our results, we tested a variety of models. The models and descriptions of everything we tested with that model are as follows:

1. Naive Bayes - MultinomialNB
2. Logistic Regression - We tried fine tuning the following hyperparameters in Logistic regression using Randomized Search CV and Grid Search CV:
    (a) C
    (b) class weights
    (c) penalty
3. SVC - We tried fine tuning the following hyperparameters in SVC:
    (a) C
    (b) kernel = ”linear”, ”poly”, ”rbf”, ”sigmoid”
    (c) gamma
4. AdaBoost - We tried the following base estimators in AdaBoost:
    (a) Logistic Regression
    (b) SVC
    (c) Decision Tree
5. XGBoost - We tried working with XGBoost classifier in following ways:
    (a) RepeatedStratifiedKFold + Cross Validation Scores
    (b) RepeatedStratifiedKFold + GridSearchCV

Because we were aware of the class imbalance, we used predict proba instead of predict on these models. We fine-tuned the threshold up to four places after decimal for predict proba to divide the classes.

To have a better picture of how our model would perform on test data, we divided the training data into train data and validation data. We even tried to discover the ideal validation train split size to benefit from both sides, i.e. to acquire a better model and a thorough grasp of how the model works before deploying it on test data.

We achieved the highest f1 score on local validation by using the Logistic Regression model with the following hyperparameters: solver = ”liblinear” and max iter = 1000.

6 Best submission stats

We came in second position on the private scoreboard, barely 0.0001 points behind first.
On the private leaderboard, our top su bmission has a f1 score of 0.63819.

We have used the following things for this submission-

1. No Preprocessing
2. Feature extraction using Count Vectorizer with n gram range = (1, 4)
3. No Resampling technique involved
4. Validation train split size = 0.1
5. Model - Logistic regression with hyperparameters: solver = "liblinear" and max iter = 1000.

We feel and are certain that if we just add the validation data to our training data, our model's accuracy will improve and we will be able to get a higher f1 score and hence be first on the leaderboard. However, due to time and submission limits on the penultimate day, we were unable to do so.

7 Code

Google Colab notebook