Ensemble Learning [2022] - MSc in Data Sciences and Business Analytics

Cyberbullying Classification

Gitlink: https://github.com/prajwal-lohan/ensemble_learning

Aashima CHUGH(aashima.chugh@student-cs.fr)

Karin ISOGAI(karin.isogai@student-cs.fr)

Youjin JEONG(youjin.jeong@student-cs.fr)

Hannah HO-LE(hannah.ho-le@student-cs.fr)

Prajwal LOHAN(prajwal.lohan@student-cs.fr)

March 2022

Table of Contents

1. Introduction

Due to a misunderstanding of the concept of freedom of speech, the introduction of social media, notably Twitter, has raised a slew of concerns. Cyberbullying is one of these challenges, which is a serious global problem that impacts both individuals and societies. Many attempts have been made to intervene in, prevent, or reduce cyberbullying in the literature; nevertheless, these methods are impractical since they rely on the victims' interactions. As a result, detection is vital to stop cyberbullying without the victims' involvement. We try to answer these questions in this study. By taking a dataset of 48,000 tweets, we were able to investigate this issue. We utilized Decision Trees, Light Gradient Boosting Method (LGBM), and other Ensemble methods to classify the tweets into five different categories of cyberbullying and identify the harmful tweet as cyberbullying using binary classification.

1.1 Objective

Through this study, our team aims to achieve the following: 1) create a multi-class classification model for predicting the potential cyberbullying type - age, ethnicity, gender, religion, and other types of cyberbullying, - and 2) create a binary classification model to discern the tweets potentially containing hateful expressions.

Ensemble learning combines multiple weak learners to perform the tasks with greater accuracy, which has advantages of good flexibility and higher generalization performance. We have used several ensemble learning techniques for constructing the multi-class classification model. Random Forest has been used as the baseline model to select the features. Based on this, we have explored Decision Trees, LightGBM, Gradient Boosting, XGBoost, CatBoost, and Adaboost to achieve better classification results.

1.2 Dataset

We have used the cyberbullying dataset generated by Dynamic Query Expansion (DQE) process to extract natural language data from Twitter containing about 48,000 texts. (J. Wang et al., 2020) This dataset includes labels of the five different cyberbullying types which are equally distributed among classes.

1.3 Methodology

We divided our work into four main steps: (1) Data preprocessing, (2) Feature engineering, (3) Learning algorithm and hyperparameter tuning, and (4) Evaluation. Their details will be described in the next sections. We compared the performance of seven Ensemble tree-based classifiers that are commonly used for multi-class classification. We also used the CountVectorizer, Frequency-Inverse Document Frequency (TF-IDF), and Word2Vec models for feature extraction.

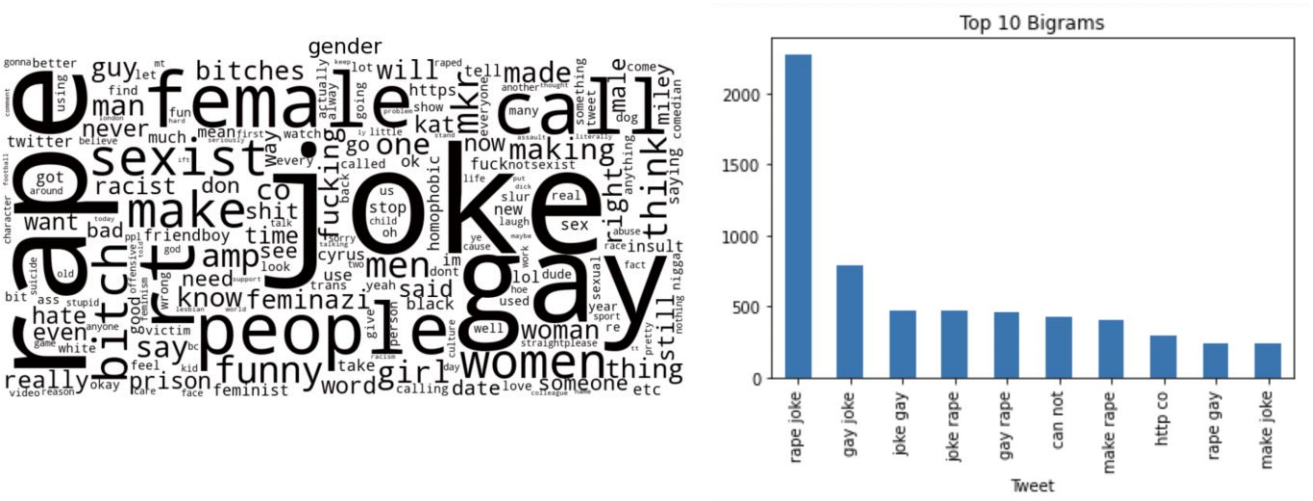
2. Exploratory Data Analysis and Data Preprocessing

2.1 Data Visualization and Exploration

Word Cloud displays the most frequent words in the text so that we can identify which words are prominent in a text. We created the Word Cloud for each cyberbullying type category. In addition, we examined the text in terms of bigrams because they can also represent characteristics of the text.

In Gender class, “joke,” “rape” and “gay” are displayed more frequently than other words while there are many harmful words that attack other people. (Figure 1) In terms of Bigrams, we observe that many bigrams contain the word “joke”, and that the bigram “rape joke” is more than three times frequent than other bigrams. In Ethnicity class, “fuck”, “nigger” and “dump” are the most frequently used in the texts, and the bigrams analysis indicates that many tweets use the combination of those words. The word cloud of Other CyberBullying class shows different characteristics from other cyberbullying classes. The most frequent words relate to a website, such as “rt”, “co”, and “https.” (Appendix 2)

[Figure 1. Word cloud and ten most frequent bigram of gender-related cyberbullying]



2.2 Data preprocessing

Data preprocessing was necessary as the first step so we could test the models afterward. We describe below the list of actions implemented in our code. How we evaluated each action is described after the list of actions.

Removing Duplicates: We removed all duplicate rows, keeping only the first row.

Removing Stop words: The stop words are the words that are most common in any language such as articles, conjunctions, and so on. These words do not necessarily add value to the sentiment of the sentence, hence removing them would reduce the size of our dataset and the training time as fewer tokens are involved in the training. There is no negative consequence on the model performance. We removed the stop words from the English Language via the NLTK library.

Lower-casing: The text has been lower-cased as it helps in the process of preprocessing and in later stages of the NLP application.

Expanding contractions: Contractions are words or combinations of words that are shortened by dropping letters and replacing them with an apostrophe, such as: you are - you're. Expanding contractions makes the tweets dataset more analyzable and thus predictable.

Removing numbers and punctuations: Since numbers and punctuations do not necessarily add any value to the language, these were removed to reduce the noise in the data.

Lemmatization and Stemming: Lemmatization is the method to normalize the tweets by keeping only the word root ("go", "going", "went" all change to "go"). The main goal of the normalization is to keep the vocabulary small, which helps to improve the accuracy of many languages modeling tasks. We have used lemmatization and stemming both to improve the performance of our model.

3. Features

3.1. Feature engineering

To decide on the new features, we realized that we would also need to make use of human judgment, so we applied our creative efforts and tried to develop features that might help in the feature engineering part. The features that we developed are mentioned below. This step was developed to feed in extra information to our model and help in improving the performance of models. We observed that by adding these features, the accuracy of the Random Forest Classifier went up by 3%.

- Number of Characters
- Number of Words
- Number of Capital Characters
- Number of Capital Words
- Count of punctuations
- Number of words in quotes
- Number of sentences
- Count of unique words
- Count of Hashtags (#StopCyberbullying)
- Count of mentions (@aashima)
- Count of stop words
- Average sentence length
- Unique words %age
- Stop words as a percentage of total words

3.2. Feature Extraction from Texts

Text data is required to be converted into representative vectors to be utilized as one of the inputs for Machine Learning models. We explored several embedding techniques for generating mathematical vector representations, CountVectorizer, Term Frequency - Inverse Document Frequency (TF-IDF), Word2Vec with Continuous Bag Of Words (CBOW), and Word2Vec with SkipGram.

CountVectorizer converts the collection of words into a matrix of numeric features based on the frequency of individual words appearing in the specific text. **TF-IDF** considers also the proportion of word occurrence in the whole document based on the count of each word. We have explored the **Word2Vec** method as well, based on CBOW and SkipGram. While the **CBOW based Word2Vec** model combines the representations of context words to predict the word, the **SkipGram based** model uses the center word for context words. For Word2Vec, tweet tokenizer is used, which generates tokens with hashtags(#) as it has a special meaning in tweeter.

With the Random Forest model as baseline, we explored the multi-label classification accuracy of embedding techniques above in order to see which text-preprocessing is the most appropriate for this dataset. CounterVectorizer with no stop words performed the best (85.1%). Based on this result, we have removed the stop words from the original text data and used the text without stop words as baseline text data.

[Table 1. Multi-class Classification result of different type of text, based on Random Forest Model]

Text type	CountVectorizer	TF-IDF	Word2Vec	
			CBOW	SkipGram
Original text	0.81004	0.81843	0.68644	0.52050
No stop words	**0.85148**	**0.82514**	0.65384	0.65384
Bigram	0.55897	0.69483	-	-
Trigram	0.32446	0.50142	-	-

4. Modeling and Evaluation

4.1 Learning algorithm and evaluation

In total, we tested seven algorithms and compared their F1-score as an evaluation metric. The algorithms and their results are as follows in Table 2 and Table 3:

[Table 2. Multi-class Classification results of different Ensemble techniques]

Model	CountVectorizer	TF-IDF	Word2Vec	
			CBOW	SkipGram
Random Forest	0.85376	0.86919	0.68644	0.67253
Decision Trees	0.82986	0.83670	0.59181	0.58855
LightGBM	0.86289	**0.88005**	0.80226	0.80096
Gradient Boosting	0.85115	0.86539	0.73598	0.73588
XGBoost	0.86169	0.85800	0.69796	0.69568
CatBoost	0.86028	0.87256	0.80465	0.80574
Adaboost	0.78781	0.81345	0.46426	0.46056

[Table 3. Binary Classification results of different Ensemble techniques]

Model	CountVectorizer	TF-IDF	Word2Vec	
			CBOW	SkipGram
Random Forest	0.87821	0.89581	0.88375	0.88233
Decision Trees	0.86908	0.86908	0.83638	0.83247

LightGBM	0.88418	0.89711	**0.89874**	0.89613
Gradient Boosting	0.86343	0.87277	0.87799	0.87940
XGBoost	0.85811	0.87071	0.87223	0.87245
CatBoost	0.88299	**0.89983**	0.89766	0.89809
Adaboost	0.84083	0.87462	0.84833	0.87082

4.2 Hyperparameter Tuning

LightGBM: We used a Gridsearch to optimize the parameters and improved the accuracy by 0.1% to 88.01% with the TF-IDF and the following hyperparameters: 'min_child_samples': 20, 'min_child_weight': 0.001, 'num_leaves': 34, 'reg_alpha': 0, 'reg_lambda': 0

CatBoost: For CatBoost, we used a Gridsearch method to search the optimum parameters while it does not improve the F1-score. CatBoost performs the best with TF-IDF and following hyperparameters: 'iterations': 40, 'l2_leaf_reg': 3, 'learning_rate': 0.1, 'max_depth': 10

XGBoost: Since gridsearch is computationally very expensive, we explored another option for hyperparameter tuning, hyperopt for XGboost. We defined a space of general parameters and ran a Bayesian Optimization on them to find the ones capable of minimizing the objective function MlogError. XGBoost yields the best results with TF-IDF and following hyperparmeters: 'colsample_bytree': 0.9308781563799851, 'gamma': 1.9365290511369784, 'max_depth': 5.0, 'min_child_weight': 7.0, 'reg_alpha': 43.0, 'reg_lambda': 0.9710120546082042

[Figure 2. Python code for Hyperparameter Tuning process for XGBoost using HYPEROPT]

```
[ ]  space={'max_depth': hp.quniform("max_depth", 3, 18, 1),
             'gamma': hp.uniform ('gamma', 1,9),
             'reg_alpha' : hp.quniform('reg_alpha', 40,180,1),
             'reg_lambda' : hp.uniform('reg_lambda', 0,1),
             'colsample_bytree' : hp.uniform('colsample_bytree', 0.5,1),
             'min_child_weight' : hp.quniform('min_child_weight', 0, 10, 1),
             'n_estimators': 180,
             'seed': 0
         }

[ ]  def objective(space):
         clf=xgb.XGBClassifier(
                         n_estimators =space['n_estimators'], max_depth = int(space['max_depth']), gamma = space['gamma'],
                         reg_alpha = int(space['reg_alpha']),min_child_weight=int(space['min_child_weight']),
                         colsample_bytree=int(space['colsample_bytree']))

         evaluation = [(X_train, y_train), (X_test, y_test)]

         clf.fit(X_train, y_train,
                 eval_set=evaluation, eval_metric="mlogloss",
                 early_stopping_rounds=10,verbose=False)
         pred= clf.predict(X_test)
         accuracy = accuracy_score(y_test,pred)

         print ("SCORE:", accuracy)
         return {'loss': -accuracy, 'status': STATUS_OK }
```

5. Conclusion

Among the different types of text data, pre-processing the text by removing stop words performs the best when used with the various vectorizing techniques. We also found out that TF-IDF provided good results for both multi-class and binary classification, while Word2Vec performed good only for binary classification.

In general, boosting method performs better than the bagging algorithm in this problem. Especially, LightGBM and CatBoost showed the best performance for binary classification as well as multi-classification of tweets as the Table 2 and 3 portray. As a result, we have yielded 89.98% accuracy in binary classification with CatBoost and TF-IDF, and 88.05% accuracy in multi-class classification with LightGBM and TF-IDF.

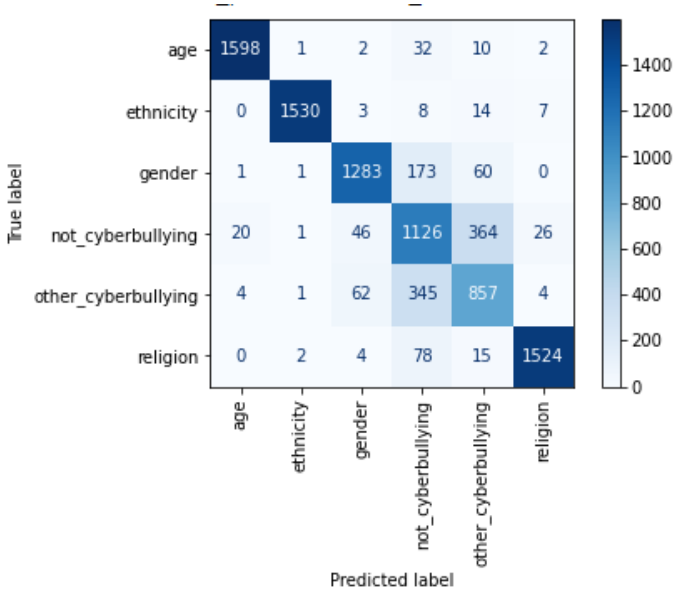
Appendix

Appendix 1. Confusion Matrix

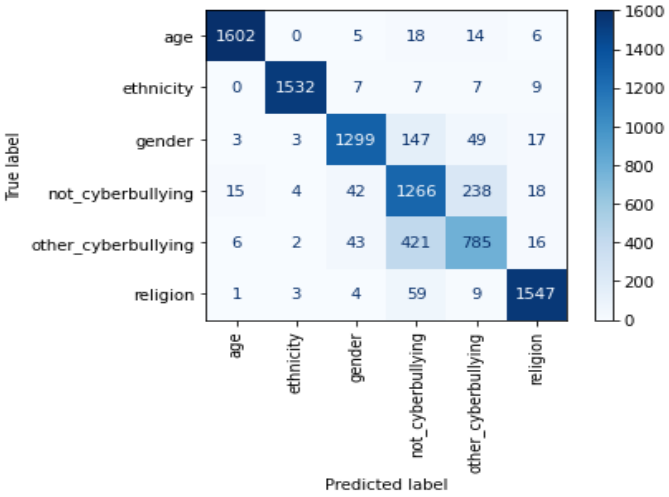
A. Multi-class Classification

i. CatBoost

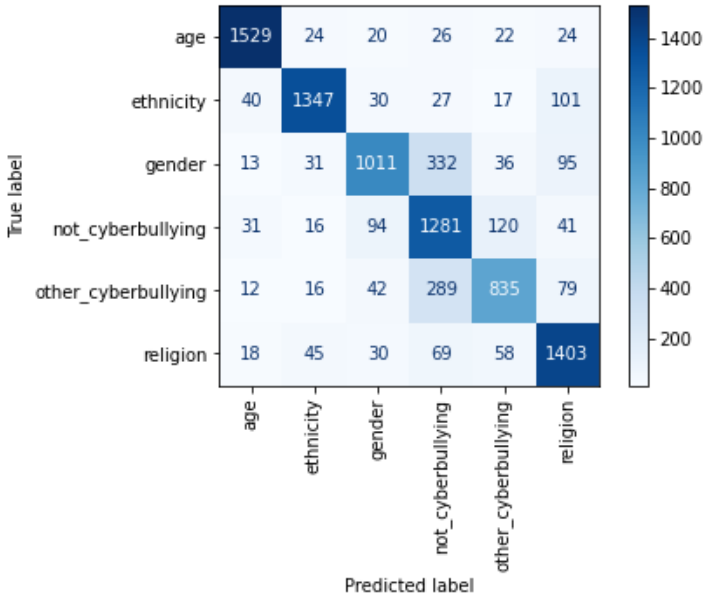
i. CountVectorizer



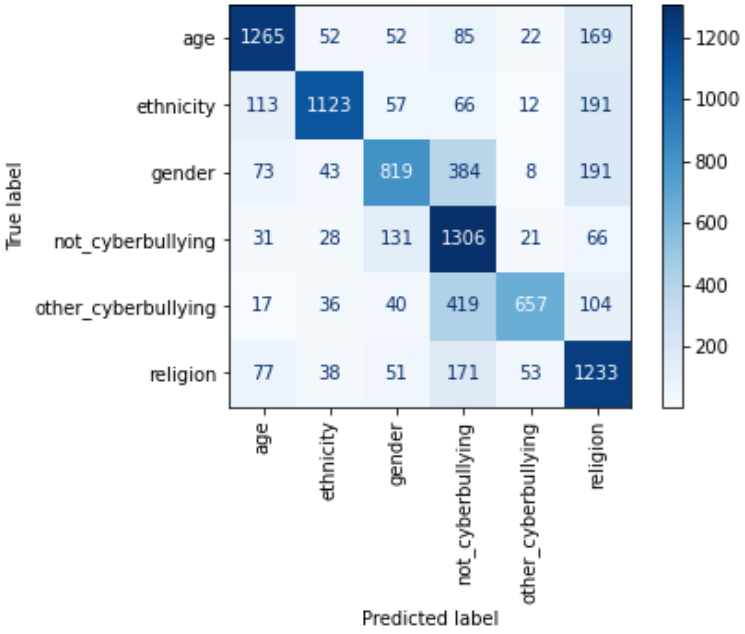
ii. TF-IDF



iii. CBOW

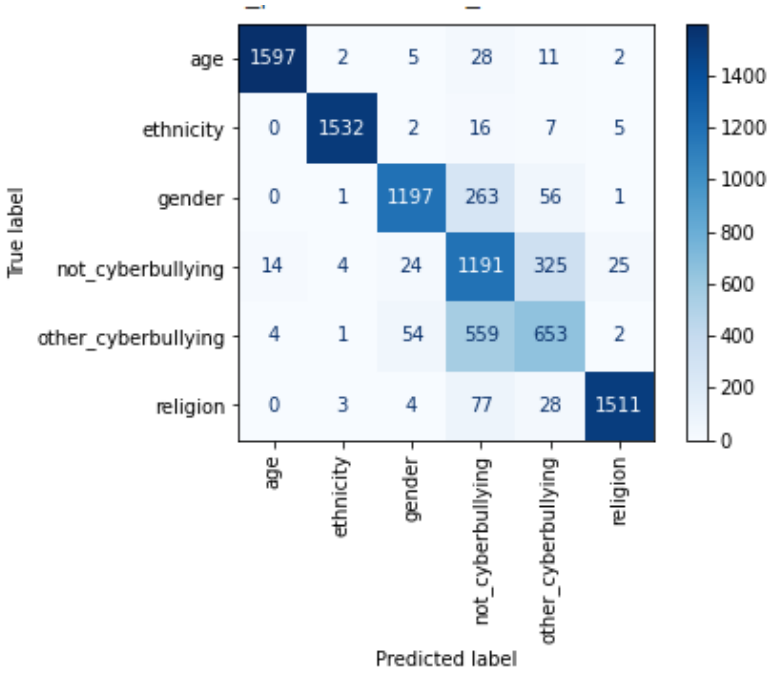


iv. SkipGram

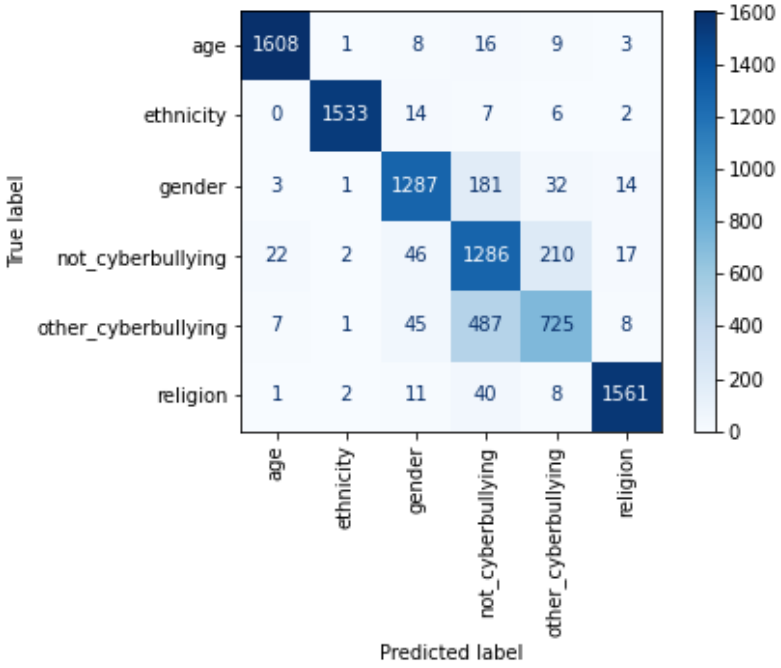


ii. XGBoost

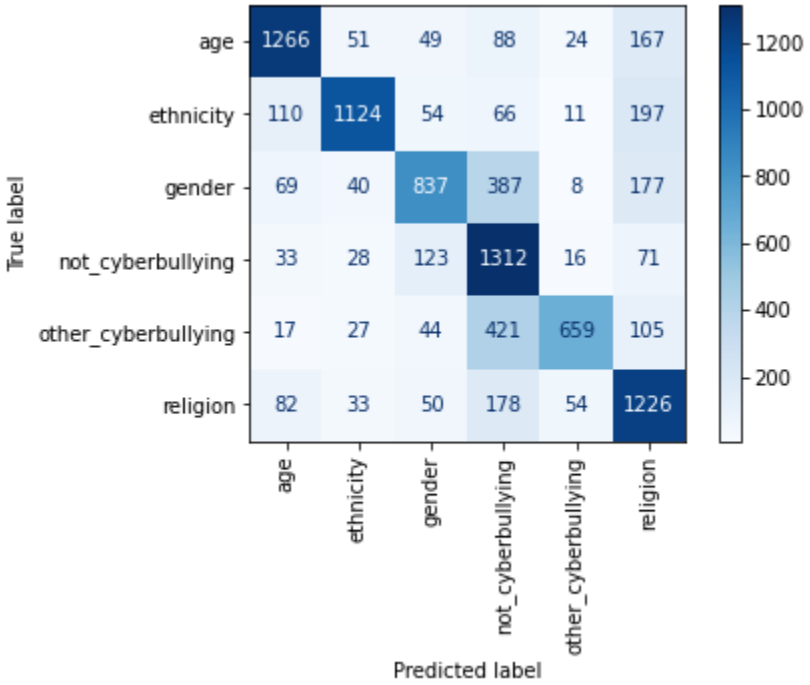
i. CountVectorizer



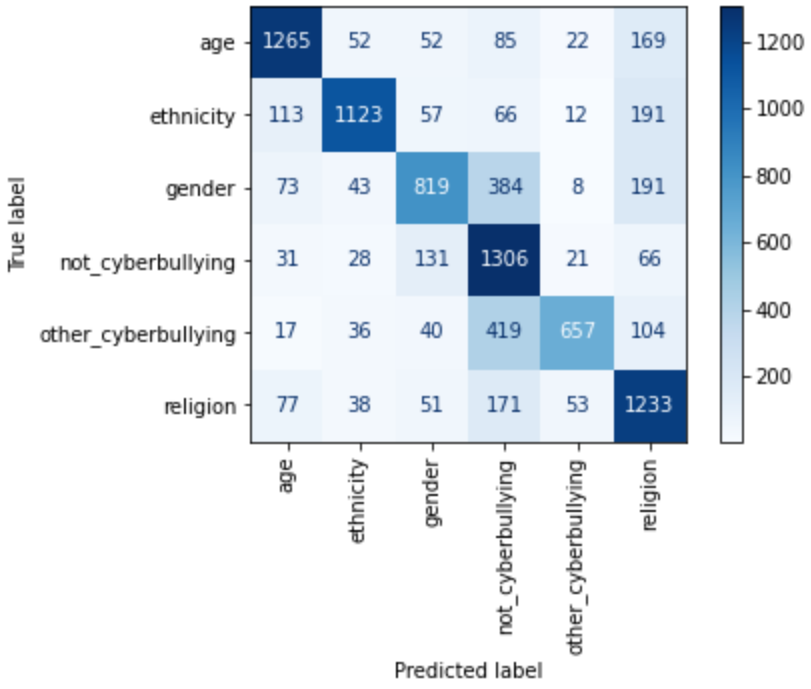
ii. TF-IDF



iii. CBOW



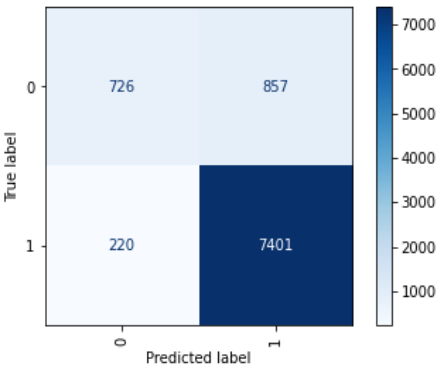
iv. SkipGram



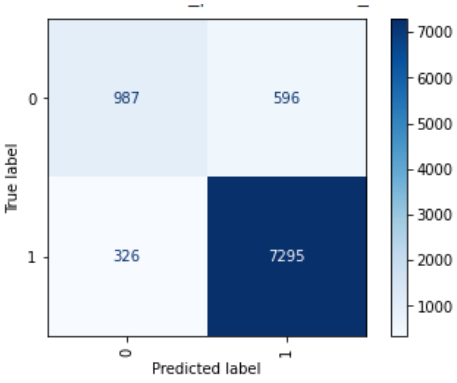
B. Binary Classification:

i. CatBoost

i. CountVectorizer

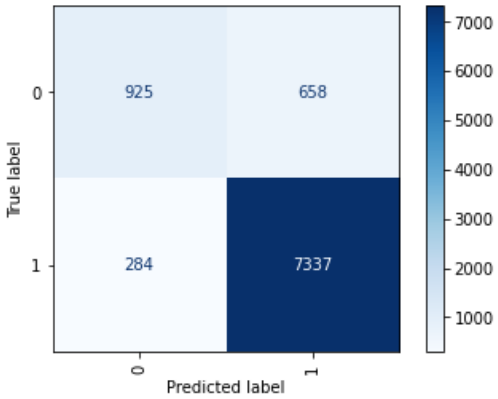


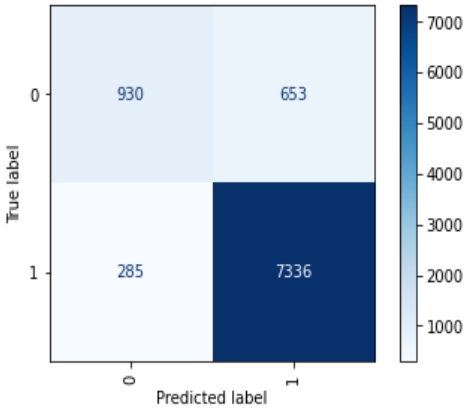
ii. TF–IDF



iii. CBOW

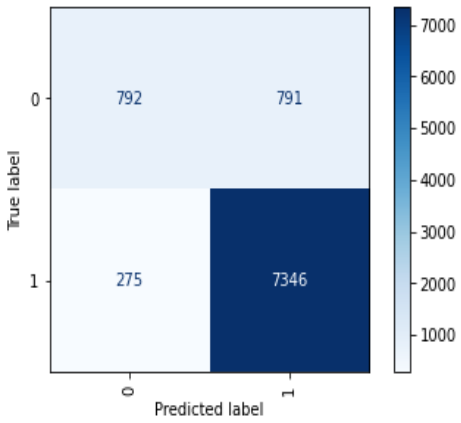
iv. Skipgram



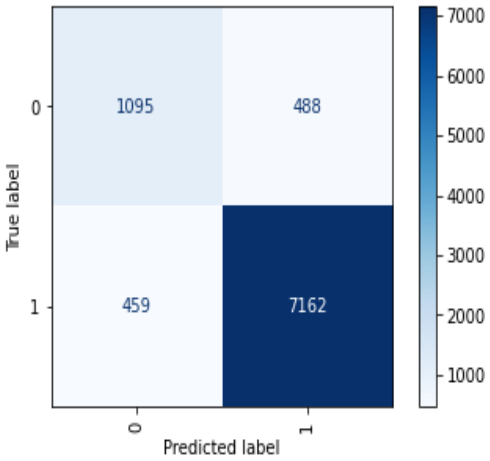


ii. LightGBM

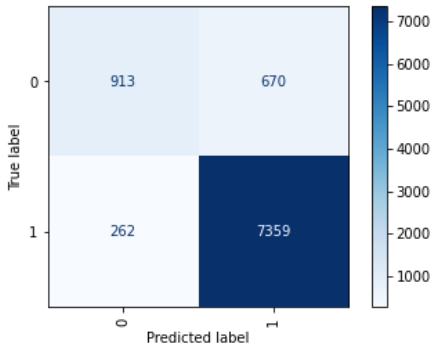
i. CountVectorizer



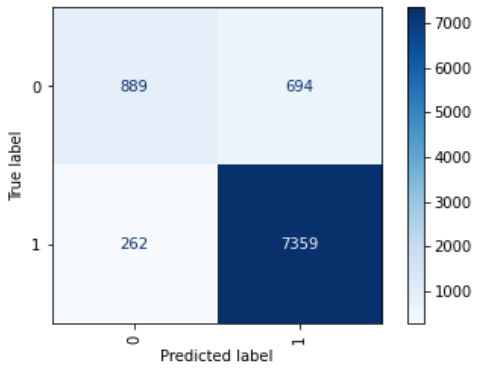
v. TF–IDF



vi. CBOW

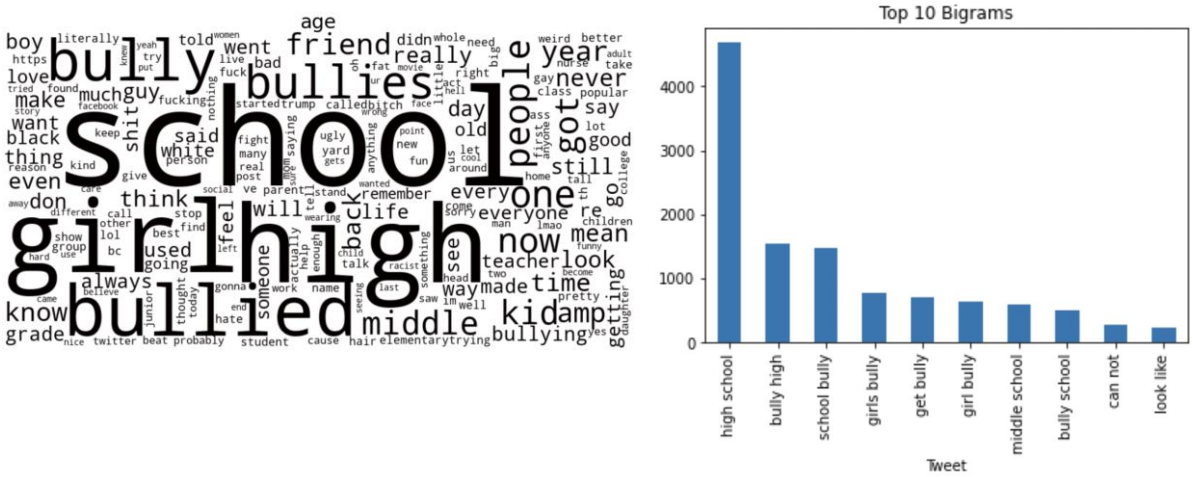


vii. Skipgram

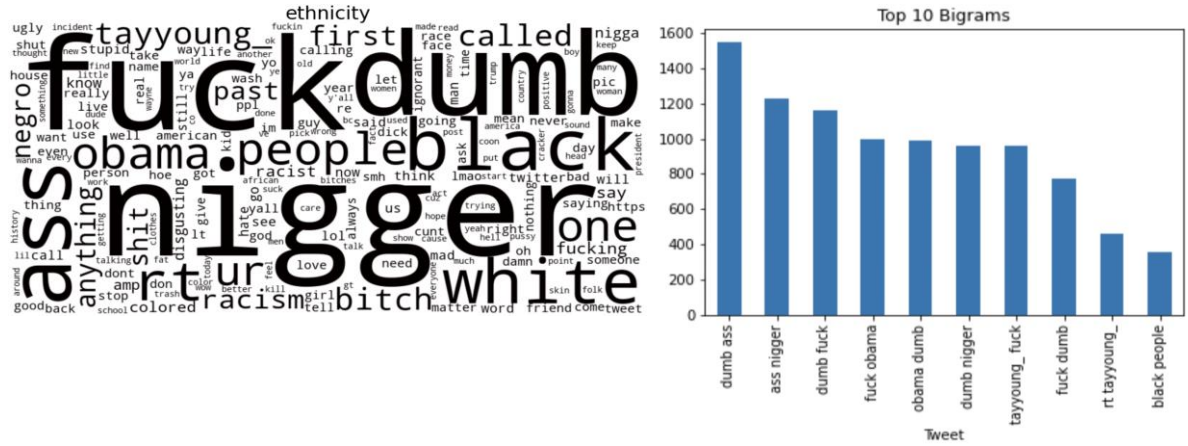


Appendix 2. Word Cloud and 10 most frequent bigrams for other topics

[Figure 3. Word cloud and ten most frequent bigram of gender-related cyberbullying]



[Figure 4. Word cloud and ten most frequent bigram of ethnicity-related cyberbullying]



[Figure 5. Word cloud and ten most frequent bigram of other cyberbullying]