18 March 2022

Toxic Comment Classifier
Using Supervised Machine Learning Model

Bhavini Kadiwala, Jose Lopez, Ammon Bhatti, Selamawit Berekat

Member Contribution
Member 1 Contribution: Introduction + word2vec implementation
Member 2 Contribution: Word2vec + MLP methodology + implementation
Member 3 Contribution: Naive Bayes Code + Summary
Member 4 Contribution: Data Description + Neutral Networks methodology

1 Introduction

With 4.4 billion people worldwide using social media platforms, it is evident that the way we conduct ourselves online has a huge impact on society. Social media has made it normal to see hateful and vulgar comments daily, which leads to a negative impact on human behavior. People are afraid of threats and abuse online, which can often lead them to not express themselves and their opinions. These toxic comments have very real consequences as we can see in today's society with the attack on the White House by Trump supporters and the Indian government silencing the farmers' protests. On the other side, social media platforms are having a hard time regulating these toxic comments, which can lead to shutting down online communities or vastly restricting them. To help better detect these toxic comments, tools are created to focus on negative online behaviors, like toxic comments, which can be defined as comments that are rude, disrespectful, or otherwise likely to make someone leave a discussion. These tools still make errors and do not take into account the type of toxicity a comment could have. In our project, we will be focusing on building models that can detect whether a comment is toxic, severe toxic, obscene, threat, insult or identity hate. The data we will use for our project is sourced from Wikipedia's talk page edits. We conduct the Naive Bayes method, word2vec and MLP Classification to help solve the issue at hand and see what method best predicts the type of toxic comment it is. We hypothesize that the word2vec and MLP Classifier will be the best method for predicting the toxic comments by having the smallest error and fastest run time. We found that the word2vec model was better run time wise and the Naive Bayes had a higher accuracy rate. By taking a dive into the classification of toxic comments, we hope to see better regulations of hateful comments on the internet.

2 Methodology

Dataset

The dataset is imported from Kaggle and it contains four data training, testing, labels, and samples. [1]. In this project, training and testing data were used. The training dataset contains 159571 rows and 8 columns. The testing dataset contains 153164 rows and 2 columns. The predictor variables are both user ID (represented as id) and user comment (represented as comment_text). The response variable has six classification labels indicating the toxicity severity as 1=toxic, 2=severe_toxic, 3=obscenes, 4=threats, 5=insults, and 6=identity_hates. The below table represents total count and percentage of the label in the data. The percentages in the table indicate how the response discrete values are distributed in the data. Most of the comments were labeled as toxic (9.5%), and the label with the least comments were threats (0.3%). In addition, the total count in the table indicates the total count for each word separated based on the six labels.

The two basic approaches in machine learning are supervised and unsupervised learning of solving problems. Supervised learning uses the classified data sets to train the algorithms and get accurately predicted outcomes, while the unsupervised does the same analyzes but for unlabelled datasets.[3]. This report focuses on solving the toxic comment detection using a supervised machine learning algorithm called Naive Bayes. It also includes a deep learning algorithm Word2Vec method, and observes how each method performs at classifying the dataset. This section aims at providing a general idea about the algorithms in terms of the dataset.

Naive Bayes Method: Naive Bayes is a classification algorithm that uses Bayes' formula. In order to get the probability of the label given the data it applies Bayes rule and writes it as the probability of the label itself multiplied by the joint conditional distribution of the features given the label. This product is then divided by the predictor prior distribution as seen in Figure 1. The word naive in naive bayes comes from the fact that the joint conditional distribution of the features is considered conditionally independent. This assumption in the toxic comments case translates to each word of a particular comment being independent of any other word for a particular label. For example, if a comment labeled toxic has the word "hate" in it, we won't get any information about the probability of any other word being in the comment.

Word2Vec + MLP Classification: One of our approaches to label the comments in the dataset, utilized Word2Vec and Multi-Layer Perceptron Classifier (from the sci-kit learn package). After cleaning the comments in the dataset, by removing stopwords, non-letters and lowercase all words, we split the comments into arrays of words and used Word2Vec to transform these arrays into vectors. This is also known as word embedding. Word2Vec takes individual words (from an inputted text), creates a numerical representation of each word, and maps each of them to a vector. Each vector is then learned within the context of the inputted text, to try connecting definitions, semantic relationships, etc.

More specifically, Word2Vec groups together vectors of similar words, and makes very educated guesses about a word's meaning based on how they appear in a given text. Word2Vec does this in two different ways: Continuous bag of words (CBOW) and Continuous Skip-Gram Model. CBOW works by attempting to predict a target word based on a set of context words. Skip-Gram essentially does the opposite, as it tries to learn/predict words surrounding an input word. Skip-Gram was found to have higher prediction quality when working with large range word vectors. Since the average word length of comments data is nearly 400, we will choose the Skip-Gram model for our case. Skip-Gram generates training data by collecting pairwise combinations of a target word and all other words surrounding it (provided a window length). This is visualized in Figure 3.

Word2Vec uses a feedforward neural network for training the model (in other words, information travels in one direction). It primarily consists of three layers: input layer, a hidden layer, and output layer.

1. The input layer takes a word, makes it the "central word" and returns a vector containing all zeros except for one entry being "1", indicating the central word chosen. The size of the vector returned is the size of the text's vocabulary.
2. The hidden layer creates a matrix with the number of rows equal to the length of the vector returned by the input layer. Each row in this matrix contains the learned embedding representation of each word. The number of columns of this matrix corresponds to the number of features specified for each word.
3. The output layer yields a vector containing probabilities between the target word and other closely associated words in the inputted text. In other words, the k-th element of the vector contains the probability that the k-th word is in the context of the target word. This vector is the same size as the input vector.

We utilized MLP Classifier from the free Python machine learning library Scikit-Learn. MLP is an acronym for Multi-Layer Perceptron, and it is an algorithm that relies on an underlying neural network to perform classification. An MLP Classifier model essentially optimizes the log-loss function using stochastic gradient descent (SGD). SGD is a stochastic approximation of gradient descent, which is an algorithm that repeatedly takes steps in the opposite direction of the gradient of a function. This process leads to a local minimum of the differentiable function. A stochastic approximation of this process results in a reduction of computational complexity, leading to quicker iterations in exchange for low convergence rates (meaning more iterations are needed).

3 Implementation Details

Word2Vec+MLP:

After checking for null values in the data, we begin by using the function comment_to_array to get the text data in the correct format. The comment_to_array function allows us to remove STOP words, any non-letters and convert all words to lowercase and split the text into individual words. Our results from this process are arrays of the words for each comment, we save this as a new NumPy array (Figure A). Next, we set the values for the parameters of the Word2Vec model. We indicate the following parameters:

1. Word vector dimensionality (which is the number of dimensions that the features of a word are contained in)
2. Minimum word count
3. Number of threads running in parallel
4. Window size
5. Downsample setting for words that appear frequently

After setting these parameters, the Word2Vec model is trained using the Skip-Gram Method. The Functions MakeFeatureVec and getAvgFeatureVecs are then used for vector averaging. MakeFeactureVec is a function that averages all the words in a given comment and returns the average feature vectors (FeatureVec). FeatureVec is a pre-initialized empty NumPy array, which helps the speed time of the function. Index2word is a list that contains the names of the words in the model's vocabulary, which is then converted into a set to make the run time faster. Each word is reviewed through the for loop and added to the feature vector if it is in the model's vocabulary. Lastly, we divide the result by the number of words to get the average. The function getAvgFeatureVecs essentially executes a loop that goes through all the comments and runs them through the MakeFeatureVec function. Both of these functions work together to return a 2D NumPy array that contains averaged word feature vectors for all comments. We then trained 6 different MLP Classifier models with each having a different Y label (toxic, severe toxic, obscene…). We set the parameters of the algorithm, solver and hidden layer sizes, as "adam" (which is the stochastic gradient descent) and 30 layers in the neural network. Next we trained our models with a partial fit approach, which is a method that calls the dataset in chunks (this is used for runtime purposes for when the dataset is too big). Once all models were trained using the partial fit, we predicted the result for each toxicity label by using accuracy_score from the sklearn library, and obtained an accuracy score of 0.765.

Naive Bayes Method:

The python package used for the naive bayes modeling is called nltk (Natural Language Toolkit). This package contains many pretrained models and data to help with natural language processing. The implementation of this algorithm in nltk calculates the probabilities on the right-hand side of Bayes' rule using a frequency distribution. For example, the probability of a particular label (P(label)) is found by counting the number of occurrences of the label and dividing by the total number of examples. The conditional probabilities are calculated in a similar manner. The prior probability is calculated using the fact that the probabilities have to sum to one. A classification is carried out by calculating the probabilities of observing the words for each label. The model classifies the label that has the highest calculated probability.

For the data preprocessing phase, we used a word tokenizer from nltk that splits the string into a list of words. This is different from split which only splits around a space. For example “Hello World.” in the nltk will be [“Hello”, “World”, “.”] and in Python, split it will be [“Hello”, “World.”]. Then the words are normalized using lemmatization. Lemmatization actually looks at the context to find out the root word, and this is different from stemming which isn’t context sensitive. Then a regular expression is used to remove unwanted symbols and strings. Then we are left with an array of preprocessed words for every comment. The model in nltk wants the input data to be of the form dictionary([“word1”, True], [“word2, True],...). It wants the words to be paired with the boolean True. This preprocessing is done for the whole train.csv which has 150,000 plus rows. This process took 2 hours. The output dataframe was put into a csv file for use in the part.

4 Results and Interpretation:

Binary classification was done on each of the label types. A naive bayes model was trained on each one and the results are shown in Figure 2. The model had 90 plus percent accuracy on all the label types, except for Threat. On Threat it had about 88 percent accuracy. The test set was one-third the size of the whole dataset and was randomized using the train_test_split function from sklearn. The ten most informative words for each label are also shown in Figure 2. These words have the highest probability of feature given the label. Thus if a sentence had these words it would be much more likely to be labeled as that label. Preprocessing the data took around 2 hours; training a model took about 7 seconds. This is because Naive Bayes only requires one pass over the data to train. It doesn’t use iterative optimization. The highest accuracy on Kaggle was 98.8%, but this is in the multiclass-multilabel framework.

Note that even though the accuracies are individually good, they will not be so high if you are in the multilabel-multiclass framework. In this framework, you can have multiple labels assigned to each row. Therefore there are 2^(# of classes) possible labelings you can give to one comment. In this case, there are 2^6 or 64 possible labelings per comment. In this report, we treated each label as separate from the others and got accuracy’s for each. One possible approach is to use meta estimators. They use individual estimators and train them on each labeling. This is an area of active research. One way to address this is to treat each possibility of labeling as one label and train a classifier on each labeling. Having one classifier of each possibility is incredibly inefficient. In our case, there would be 64 different binary classifiers.

This report compared a machine learning algorithm (Naive Bayes) and a predictive deep learning model (word2vec) to find the best model to predict the presence of toxic comments in websites. From the experimental results, it is concluded that the Word2Vec + MLP Classifier algorithms are the most optimal for predicting toxicity in comments. The Word2Vec + MLP Classifier is chosen since it has the fastest computation time among both the algorithms (around 15 mins), and its average accuracy percentage is 76%. The Naive Bayes Classifier had an average run time of 2 hours, however had an average accuracy percentage of 90%. The report shows that machine learning and predictive deep learning techniques can be implemented to help detect toxic comments in websites given user comments.

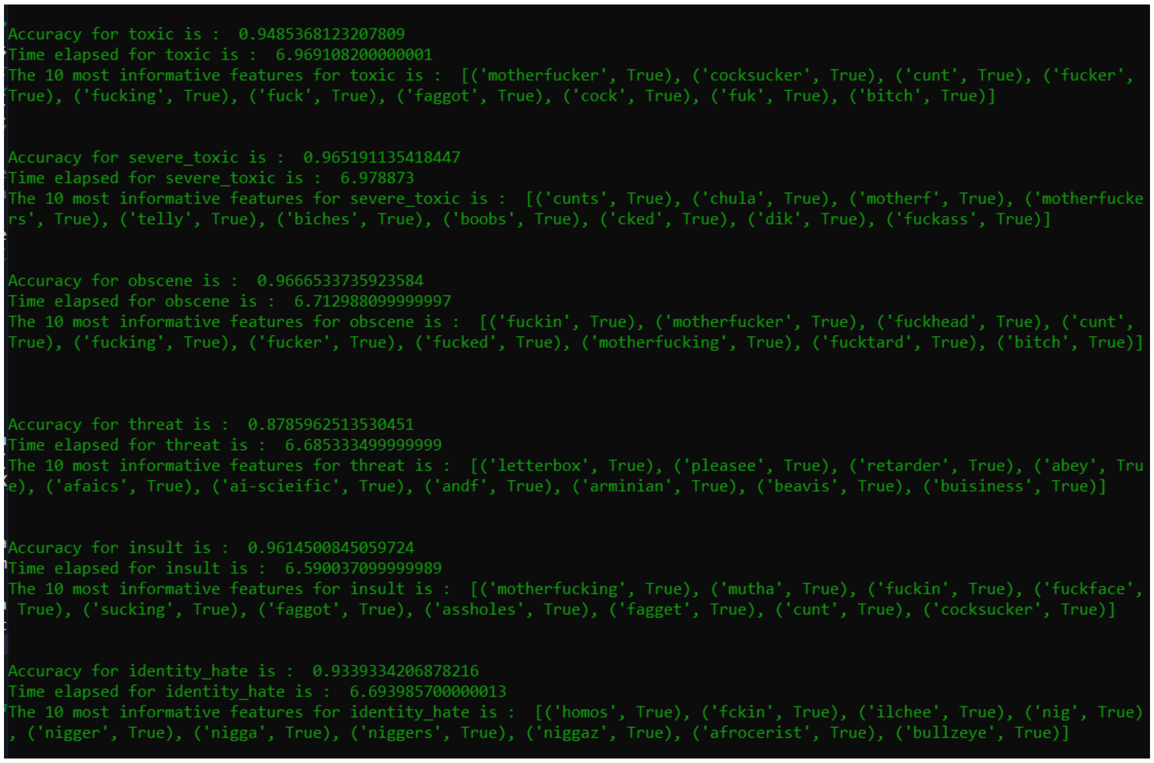
5 Supplementary

Label	TotalCount	Percent
1 = toxic	15294	9.5%
2 = severe_toxic	1595	1.0%
3 = obscenes	8449	5.2%
4 = threats	478	0.3%
5 = insults	7877	4.9%
6 = identity_hates	1405	0.8%

$$\underset{\text{Posterior Probability}}{\underset{\downarrow}{P(c \mid x)}} = \frac{\overset{\text{Likelihood}}{\overset{\nwarrow}{P(x \mid c)}}\,\overset{\text{Class Prior Probability}}{\overset{\nearrow}{P(c)}}}{\underset{\text{Predictor Prior Probability}}{\underset{\searrow}{P(x)}}}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Figure 1.
Naive bayes explanation, source: [2]



```
Accuracy for toxic is :  0.9485368123207809
Time elapsed for toxic is :  6.969108200000001
The 10 most informative features for toxic is :  [('motherfucker', True), ('cocksucker', True), ('cunt', True), ('fucker',
True), ('fucking', True), ('fuck', True), ('faggot', True), ('cock', True), ('fuk', True), ('bitch', True)]


Accuracy for severe_toxic is :  0.965191135418447
Time elapsed for severe_toxic is :  6.978873
The 10 most informative features for severe_toxic is :  [('cunts', True), ('chula', True), ('motherf', True), ('motherfucke
rs', True), ('telly', True), ('biches', True), ('boobs', True), ('cked', True), ('dik', True), ('fuckass', True)]


Accuracy for obscene is :  0.9666533735923584
Time elapsed for obscene is :  6.712988099999997
The 10 most informative features for obscene is :  [('fuckin', True), ('motherfucker', True), ('fuckhead', True), ('cunt',
True), ('fucking', True), ('fucker', True), ('fucked', True), ('motherfucking', True), ('fucktard', True), ('bitch', True)]



Accuracy for threat is :  0.8785962513530451
Time elapsed for threat is :  6.685333499999999
The 10 most informative features for threat is :  [('letterbox', True), ('pleasee', True), ('retarder', True), ('abey', Tru
e), ('afaics', True), ('ai-scieific', True), ('andf', True), ('arminian', True), ('beavis', True), ('buisiness', True)]


Accuracy for insult is :  0.9614500845059724
Time elapsed for insult is :  6.590037099999989
The 10 most informative features for insult is :  [('motherfucking', True), ('mutha', True), ('fuckin', True), ('fuckface',
 True), ('sucking', True), ('faggot', True), ('assholes', True), ('fagget', True), ('cunt', True), ('cocksucker', True)]


Accuracy for identity_hate is :  0.9339334206878216
Time elapsed for identity_hate is :  6.693985700000013
The 10 most informative features for identity_hate is :  [('homos', True), ('fckin', True), ('ilchee', True), ('nig', True)
, ('nigger', True), ('nigga', True), ('niggers', True), ('niggaz', True), ('afrocerist', True), ('bullzeye', True)]
```

Figure 2.
Naive Bayes results

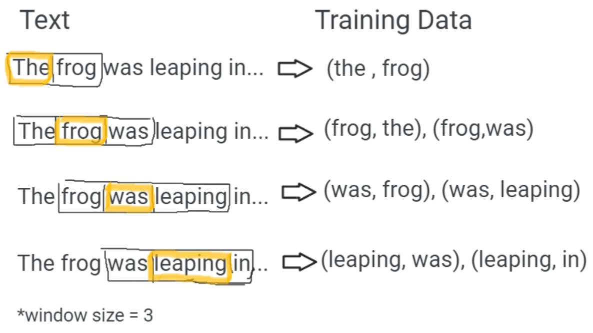
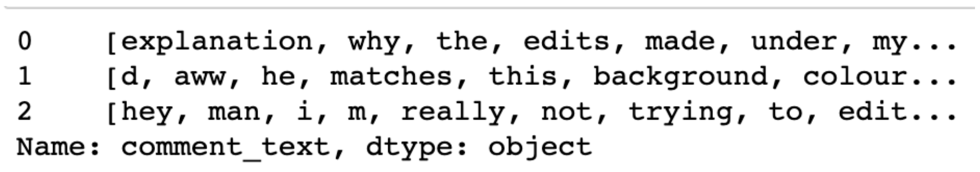
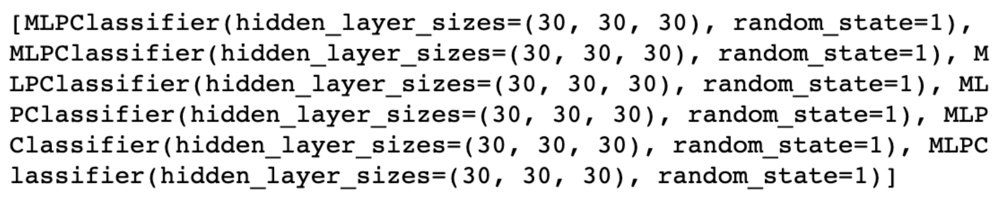

Text
Training Data
The frog was leaping in...
(the , frog)
The frog was leaping in...
(frog, the), (frog,was)
The frog was leaping in...
(was, frog), (was, leaping)
The frog was leaping in...
(leaping, was), (leaping, in)
*window size = 3

Figure 3. Word2Vec Skip-Gram process



```
0    [explanation, why, the, edits, made, under, my...
1    [d, aww, he, matches, this, background, colour...
2    [hey, man, i, m, really, not, trying, to, edit...
Name: comment_text, dtype: object
```

Figure A
Result of cleaning text data



```
[MLPClassifier(hidden_layer_sizes=(30, 30, 30), random_state=1),
MLPClassifier(hidden_layer_sizes=(30, 30, 30), random_state=1), M
LPClassifier(hidden_layer_sizes=(30, 30, 30), random_state=1), ML
PClassifier(hidden_layer_sizes=(30, 30, 30), random_state=1), MLP
Classifier(hidden_layer_sizes=(30, 30, 30), random_state=1), MLPC
lassifier(hidden_layer_sizes=(30, 30, 30), random_state=1)]
```

Figure B
Result of MLPClassifer

Work Citation

[1] Jigsaw/Conversation AI. (2018). Toxic Comment Classification Challenge, Original version. Retrieved [01 February 2022] from https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/overview.

[2] UC. (n.d.). Naïve Bayes Classifier. UC Business Analytics R Programming Guide. Retrieved March 14, 2022, from https://uc-r.github.io/naive_bayes

[3] Madhavan, Samaya, et al. "Supervised deep learning.", IBM Developer. Published on 21 July 2021. Retrieved from URL [https://developer.ibm.com/learningpaths/supervised-deep-learning/].

[4] C, Andrea. "How to train the word2vec model", Medium. Published on 21 July 2021. Retrieved from URL [https://towardsdatascience.com/how-to-train-the-word2vec-model-24704d842ec3].