“Toxic Comment Challenge: NLP SUCKS” [1,0,0,1,0,0]

James Jiang Jeffrey Anderson Christie Ngo

James Jiang: NLP Research, Data cleaning/preprocessing, TF-IDF, ML-KNN

Jeffrey Anderson: NLP, Research, TF-IDF, Multi-Label, Visualization, Interpretation

Christie Ngo: Metrics Research, word2vec, Multi-output Logistic Regression, Visualization

I. Introduction

Team Project Goal

To understand the fundamental concepts underlying natural language processing, to explore some basic implementations, to explore more advanced implementations using either the word2vec word-embedding algorithm developed by researchers at Google or the GloVE model developed by members of Stanford University. We were interested in analyzing different methods of classification in order to determine the best technique for multilabel text.

Background/Dataset

The goal of this project is to tackle a problem that was posed in a Kaggle Dataset Competition. The data of this project consisted of a corpus of roughly 160,000 comments taken from wikipedia discussion forums that had undergone a labeling process. This labeling process assigned binary labels to six, non-mutually exclusive categories: “toxic”, “severe toxic”, “obscene”, “insult”, “threatening”, and “identity hate”. The competition included a set of 60,000 test comments with associated test labels.

II. Methodology

Natural Language Processing: Data Cleaning and Preprocessing

Data cleaning is necessary for clearing the corpus of useless words. It also strips down words so that multiple versions of a word would be considered just one word in the eyes of the model. Data cleaning is also useful for removing words, which will lighten the computational load. Data cleaning included lowercasing, de-punctuating, white space removal, stop word removal, and stemming, all of which will be covered in implementation details.

The test data came in two separate csv files; one for the comments and another for the labels. To make sure the test data was ready to go through the models, the test comments were merged with the labels. Also, not all of the test comments had labels assigned, and were subsequently given column values of -1. These were removed, since there was no way to test how well our model did with no answer key.

NLP: Feature Engineering: One Hot → Bag of Words → Tf-idf

One Hot Encoding

After the text data has been cleaned and preprocessed, we are left with a list of words which we will now denote as tokens or word–features(with each token more or less representing a single word).

The one hot encoding process is a primitive method of transferring this list of words into a numeric representation. If we imagine a series of row vectors used to represent each document in the training corpus, then the columns of those row vectors will represent the features of our data, and in our case the features are the list of all unique tokenized words. Imagining a comment with no text; this translates to a p-dimensional row vector of just 0’s. One hot encoding tells us that non-presence of a word-feature will be represented by a 0 and presence of a given word-feature will be represented by a 1. Thus we can imagine that if each document represents only a small subset of set words contained in the

entire corpus that we will be left with p-dimensional vectors containing many 0's and very few 1's. This formulation leads to what is known as a sparse data matrix.

Bag-of-Words

The next logical progression in the document vectorization process relates to the Bag of Words concept. The Bag of Words model tells us that not only is the list of word-features treated as unordered, but also that the vectorization ought to start taking multiplicities of words into account. Thus instead of vectors whose elements are only {0,1}, under the bag of words representation, nonzero entries will represent the number of occurrences for a given word-feature in its respective document.

TF-IDF

This brings us to the last step in our progression of text vectorization: tfidf. To recap, we started with binary row vectors in one hot encoding, and in bag-of-words these non-zero entries now represent the term-frequencies in each document. However, since each numeric representation of a document will be characterized by its most frequently occurring words – yet certain words are inevitably likely to occur more often across the entire corpus even after the stop-word removal process.These words will thus confer less meaning to any individual document, in the sense that they will certainly not help in differentiating a given document-vector from any other and may actively impede our ability to do so.

Thus tfidf attempts to modulate the multiplicities by introducing an inverse document frequency factor (the idf of tfidf). This idf factor is similar at least in philosophy to the penalty terms and shrinkage factors we have explored in depth in class: term frequency is not an unmitigated good, so it must be balanced by the inverse document frequency – a penalty for words that occur frequently across the set of ALL documents.

Multi-Label

Nature of Our Labels: A Multi-Label Classification Approach

An important observation must be made about the nature of the classification problem: because the labels are not mutually exclusive, i.e. a comment can be labeled as "toxic", "insult", and "threatening" means this can be considered a multi-label classification problem.
There are traditionally three approaches to the muli-label paradigm. (citation)

1. Adapt an algorithm
2. Transfer to a multiclass classification (power label method)
3. Treat as binary classifiers

The methodologies considered in this report were mlk-nn which is an adaptation of the knn algorithm for multilabel settings, and classifier chain which treats the classification problem as a sequence of separate binary classification problems. Two methods which are suggested in the literature are one-vs-rest paradigm and the classifier chain paradigm (citation). The apparent difference between the two is that the classifier chain refines the idea behind one-vs-rest which assumes independence of binary labels and thus leads to independent binary classifiers. This might not always be an appropriate assumption, so the classifier chain allows for correlations between the labels to be exploited in order to return a more sophisticated classification scheme.

ML-KNN on the other hand attempts to adapt the knn algorithm that was covered in class to a multilabel context (cite paper). ML-KNN uses Bayes Theorem to find the probability of a test instance having each label. The value of each label is determined by the product of the prior and posterior probabilities. The prior is determined by results of the training labels and the Laplace Smoothing value. The posterior is determined by whether or not the k nearest neighbors of the test instance have the label or not. We go into detail on the nature of ML-KNN in the supplementary pages.

III. Implementation Details

Data Cleaning

To prepare the data, each sentence had to be simplified so that they only contained meaningful content, and so that computation would be easier and return only relevant information. The first steps were to lowercase all strings and remove punctuation. Other data cleaning and preparation primarily came from the natural language toolkit (nltk) package. Nltk included a list of stopwords, which are English filler words that do not contribute anything to our classification model. We used regular expressions (regex) to search the corpus for stopwords, and removed them.

The data was then stemmed, meaning that prefixes and suffixes were removed. Multiple versions of the same word would turn into just one version of the word. For example, “cried” and “crying” would both become “cri”. To stem the data, we used Snowball stemmer from nltk. We decided to use Snowball stemming over Porter Stemming because Snowball is essentially the new and improved Porter, and is often called Porter2. Porter has a simpler design, but is also often less precise.

We created an overall function that took in a dataframe of sentences, and returned one that had applied all data cleaning steps mentioned above.
To stem the data, each sentence for its row was run through a custom function, in which the sentence was split into individual words. Each word was stemmed using Snowball stemmer, then put into a new empty sentence, which replaced the original sentence.

Feature Engineering -- Vectorization

Vectorization was accomplished by utilizing the sklearn package vectorizer, which takes in a corpus and calculates the tfidf values for the documents. This outputs a sparse matrix whose rows represent the individual comments and whose columns represent the feature-space of all words in the dictionary. Outputs are sparse matrices, meaning the majority of entries are 0’s. Important to note is that the tf-idf vectorization is trained only on the training set of inputs, meaning any words it encounters in test data will be OOV (out of vocabulary) which is a shortcoming of this approach.

Classifier-Chain

Once the test and training inputs were vectorized with their tf-idf weightings, they were passed through a logistic regression classifier and then passed through the classifier chain function which applied the logistic regression to each label separately.. The method for solving the logistic function was liblinear and the order of the classifier was specified as random. The output will be an instance of the logistic

regression classifier chain model which will allow for predicted label sets to be generated on the test input data set. The input space will have the same dimensions as the training set since the tf-idf feature space is derived solely from the vocabulary of the training set.

ML-KNN

The test and training inputs were also passed through the mlknn function k=12 was used. Comparatively mlknn took much longer to run using the data. This could have been a result of using sparse-data matrix as an input. Further tuning could be explored using the gridsearch functionality to help identify the best value of the k-hyperparameter. Additionally, ml-KNN implementation might be more successful if certain data reduction techniques are employed in the future.

IV. Results and Interpretation

Evaluation Metrics from Implementation
Please see classification reports contained in supplementary sections for specific results pertaining to the implementation of the methods described in this report.

Precision Recall
Because the data set was highly imbalanced (see Figure 1 in supplement) accuracy is not an appropriate evaluation metric. Precision and recall will offer more relevant metrics than accuracy. Since the majority of comments are labeled as 0's in all categories, a model assigning 0's to every label would still result in a model with high accuracy, but would confer no ability to correctly classify the labels of interest. Precision measures the proportion of true positives to everything that was predicted as a positive ($\frac{TP}{TP+FP}$). And recall measures the proportion of true positives to all items labeled as positive ($\frac{TP}{TP+FN}$).

From the classification report based on the classifier chain of individual binary logistic regressions, the algorithm received high precision and recall scores for toxic, obscene, and insult labels. The recall scores for the other three labels (severe toxic, threat, and identity hate) were drastically lower. This suggests that the classifier had difficulty in detecting a large portion of the true positive labels for these three label categories. Intuitively this might make sense seeing as detecting these labels from a human perspective requires a level of nuance not available to the basic machine learning scheme considered in this project (for example what differentiates a benign racial reference to a malicious racial reference).

ROC/AUC
The AUC-ROC (Area Under The Curve Receiver Operating Characteristics) measures how well the model can separate between classes. It plots the false positive rate (total number of false positives out of all true negatives, $\frac{FP}{TN+FP}$) against the true positive rate (total number of true positives out of all true positives, $\frac{TP}{TP+FN}$). When the AUC is 0.5 and the two ROC distributions of the positive class and negative class overlap completely, the model is unable to distinguish between classes.

1.Supplementary – Formulas and Algorithms

A. Word2Vec -- Word embedding

One method of vectorizing the text was to use word2vec from the gensim package. As a space-efficient method of formulating word embeddings, word2vec was considered as an alternative to the TF-IDF process. Contrasting TF-IDF, word2vec considers the context of each word based on its learned vocabulary when computing the word embeddings. As there is no sparse matrix involved, word2vec performs better when considering space complexity. Although word2vec itself as a model iis unsupervised, it implements a supervised classification model based on continuous bag of words and skipgrams to return the embeddings.

Continuous bag of words and skipgram

With the continuous bag of words approach, the representation of surrounding context is used to predict the middle word. The model learns which words co-occur. We first pass in context words into the embedding layer for the word embeddings to be inputted into the lambda layer. This layer takes the average of the word embeddings for the SoftMax layer to predict target words.

For the skipgram procedure, we are interested in the weights of the hidden layer from the neural network. Given a word, skipgram attempts to predict its surrounding context. The model receives input of positive and negative paired samples where negatives labeled with 0 imply that the pair is irrelevant. Dense word embeddings are passed onto the merge layer to compute dot products of word pair embeddings. The dot value goes into the sigmoid layer, which outputs either 1 or 0. After comparing this result with the true label, the embedding layer is updated.

Implementation of word2vec

We develop our word2vec model using our training data set. Each comment was split into words and using “Phrases” from gensim, we were able to group common phrases into one entity. Using the trained word2vec model, we used the learned vocabulary to retrieve word embeddings of each word in the testing comments. We summed up all the 100 dimensional arrays of each comment and these became our training dataset for the multi-output classifier using logistic regression. The probabilities of each label for the comment were returned. Using a simple threshold of >0.5 for a positive, we classified the probabilities into binary labels for assessment metrics. The classification report from sklearn.metrics (returns precision, recall, F1 scores) was used to evaluate our classifier. With extremely high recall and low precision, we recognize that the extreme class imbalance skewed our classifier to label most comments with positives.

Balanced accuracy score

When dealing with an imbalance in classes, the balanced accuracy score can be used to evaluate binary classifiers. It is the average of sensitivity and specificity, $(\frac{TP}{TP+FN} + \frac{TN}{FP+TN})/2$. Balanced scores around 0.5 indicate that the classifier tends to predict mostly negatives or mostly positives. Our mean balanced accuracy score across all labels was 0.5; we conclude that the training set had a strong class imbalance and our model tended to predict positive hits. In the future, we would train our classifier using less of the negative samples. We can randomize the samples to drop under this condition.

B. ML-KNN

ML-KNN is a multi-label classification method that draws inspiration from the original K-Nearest Neighbors. It solves the problem of how to label some test instances with the correct labels by finding the location of this test instance on a graph, and looking at the statistical data of the *k* nearest training instances around it.

The entire data has a set of labels Y. In our case, it would be $Y = \{toxic, severe\ toxic, \ldots\}$

Each instance has its own label classification vector $\overline{y} = \{\overline{y}(1), \ldots, \overline{y}(6)\}$.

A probabilistic approach, using Bayes Theorem, is used to determine the test instance labels. The general formula is as follows: the test instance t has each label $\overline{y}(l)$, determined by

$$\overline{y}_t(l) = argmax_{b\in\{0,1\}} \; P(H_b^{\,l})\; P(E^l_{\overline{C}_t(l)} | H_b^{\,l})$$

Where the first term is the prior probability and the second is the posterior probability. The prior probability $P(H_b^{\,l})$ looks at the probabilities of instance t having the label ($H_1^{\,l}$), or not having the label ($H_0^{\,l}$). The probability $H_1^{\,l}$ is a function of the summation of the training labels and Laplace Smoothing s.

$$P(H_1^{\,l}) = \frac{s + \sum_{i=1}^{m} \overline{y}_{x_i}(l)}{2s + m} \qquad P(H_0^{\,l}) = 1 - P(H_1^{\,l})$$

Laplace Smoothing is a term for Bayes Theorem that ensures that if and when the model encounters something it has never seen before, it does not assign it a prior probability value of 0. However, since it does artificially inflate values, the Laplace Smoothing term subsequently changes the actual probabilities in order for this failsafe to exist. In ML-KNN, this value is set to 1.

The posterior distribution $P(E^l_{\overline{C}_t(l)} | H_b^{\,l})$ is determined by KNN rules. It looks at the k nearest values to the test instance, and counts the number of training instances that have the label, $c[j]$, and the number of training instances that don't have the label, $c'[j]$. You then choose the higher value of the two:

$$P(E_j^{\,l} | H_1^{\,l}) = \frac{s + c[j]}{s(k+1) + \sum c[p]} \qquad P(E_j^{\,l} | H_0^{\,l}) = \frac{s + c'[j]}{s(k+1) + \sum c'[p]}$$

C. TF-IDF

$$idf(t, D) = log(\frac{N}{n_t})$$

$$tf\text{-}idf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

t: a given term

D: a set of documents

n_t: number of documents term appears in

N: total number of documents being assessed

Note: tf(t,df) denotes the relative frequency of a term in a given document; idf will approach zero as the ratio between N and n_t approaches one, resulting in the tf-idf weighting being zero. However, when n_t is small relative N the tf-idf will increase the tf-idf weighting. [8]

2.Supplementary – Graphics

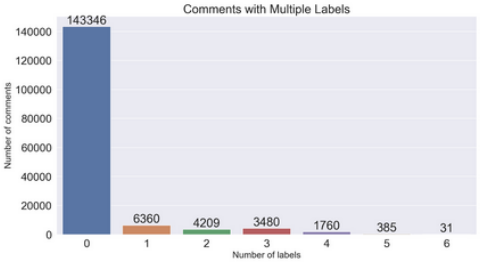


Figure 1

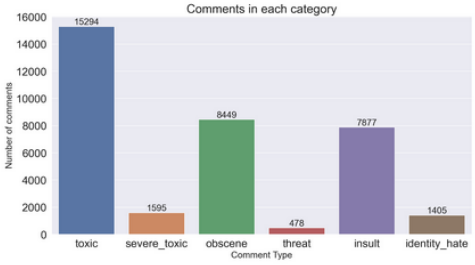
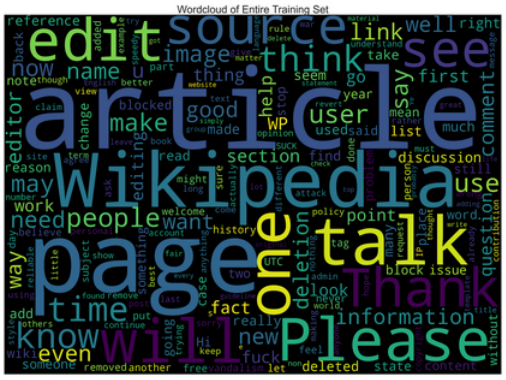
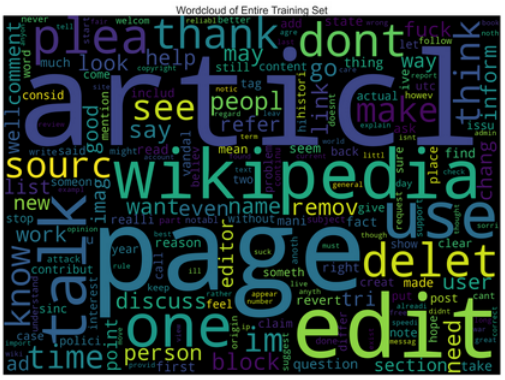


Figure 2



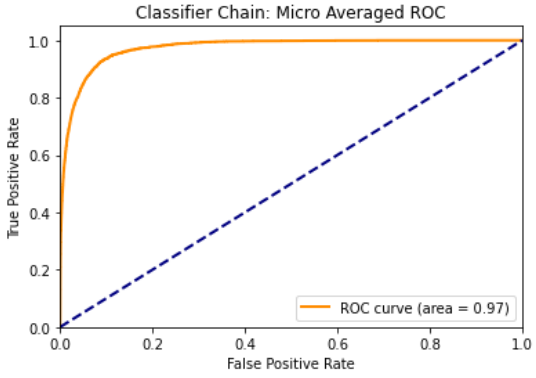


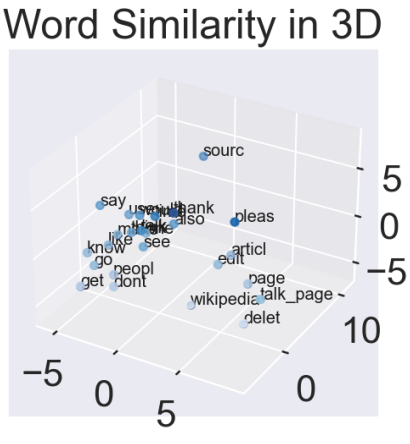
Classification Report: Classifier Chain

	precision	recall	f1-score	support
Toxic	0.70	0.63	0.66	6090
Severe Toxic	0.41	0.29	0.34	367
Obscene	0.75	0.62	0.68	3691
Threat	0.47	0.16	0.24	211
Insult	0.63	0.56	0.59	3427
Identity Hate	0.67	0.27	0.38	712
micro avg	0.69	0.58	0.63	14498
macro avg	0.61	0.42	0.48	14498
weighted avg	0.68	0.58	0.62	14498
samples avg	0.05	0.05	0.05	14498

Classification Report: MlkNN

	precision	recall	f1-score	support
Toxic	0.59	0.46	0.52	6090
Severe Toxic	0.14	0.25	0.18	367
Obscene	0.61	0.42	0.50	3691
Threat	0.22	0.14	0.17	211
Insult	0.52	0.39	0.44	3427
Identity Hate	0.54	0.20	0.29	712
micro avg	0.55	0.41	0.47	14498
macro avg	0.44	0.31	0.35	14498
weighted avg	0.56	0.41	0.47	14498
samples avg	0.04	0.04	0.04	14498





References

1. Toxic comment competition: https://www.kaggle.com/code/sravanneeli/toxic-comment-classification/notebook
2. Ml-knn: A Lazy Learning Approach to Multi-Label Learning: http://proceedings.mlr.press/v94/roseberry18a/roseberry18a.pdf
3. Classifier Chains: https://www.cs.waikato.ac.nz/~eibe/pubs/chains.pdf
4. word2vec to Logistic Regression ttps://medium.com/analytics-vidhya/text-classification-from-bag-of-words-to-bert-part-2-word2vec-35c8c3b34ee3
5. PCA for word2vec visualization https://towardsdatascience.com/visualization-of-word-embedding-vectors-using-gensim-and-pca-8f592a5d3354
6. scikit-multilearn: A scikit-based Python environment for performing multi-label classification: https://arxiv.org/pdf/1702.01460.pdf
7. Using TF-IDF to Determine Word Relevance in Document Queries: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1424&rep=rep1&type=pdf
8. Text Mining: Term vs. Document Frequency: https://afit-r.github.io/tf-idf_analysis
9. Case study of Toxic comments competition https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff