# Toxic Comment Classification
## CMPT413 Final Project Report

**Ao Tang**
tangaot@sfu.ca

**Shaoqiang Zou**
sza93@sfu.ca

## 1 Abstract

This report introduces different deep learning approaches applied to the challenge of toxicity classification of online comments. We evaluate the effect of the Logistic Regression and Naive Bayes in Term Frequency–Inverse Document Frequency (TF-IDF), and Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) in Word Embedding on identifying toxicity in comments. We analyzed our approaches on Wikipedia comments form Kaggle Toxic Comments Classification Challenge dataset. Our Recurrent Neural Network - Long Short-Term Memory model achieved the highest performance on test dataset. Overall, our word embedding method is outperformed our TF-IDF method.

## 2 Introduction

The internet has allowed individuals to connect and learn from each other. It requires a comforting and friendly environment to learn from others and simultaneously communicate one's feelings and thoughts to others. Online hate, harassment and toxicity discourage individuals from opening up and taking full advantage of the opportunities provided by online communication. In this project, six types of toxicities were predefined: toxic, severe toxic, obscene threat, insult, identity hate. The dataset of comments from Wikipedia's talk page edits is also used (AI, 2017). The training data includes 159,571 labeled comments for toxic actions labeled by human raters, and we split 20% of training data as validation set. The test data containing 63,978 unlabeled comments. Our goal is to use machine learning techniques to classify toxicity from the six types of toxicities, Which could be used to prevent users from posting potentially toxic posts, create more respectful comments when communicating with others, and to assess the toxicity of comments from other users.

This project aims to implement different deep learning approaches. We implemented two methods to study the effect of different models. To compare our approach, first, we convert comments to Term Frequency–Inverse Document Frequency (TF-IDF) to build Logistic Regression and Naive Bayes classification models. Then, we convert comments to word vector using the word embedding model (Word2vec, GloVe) to build Convolutional Neural Nets (CNN) and Recurrent Neural Nets (RNN) - Long Short-Term Memory (LSTM) classification models. We use accuracy metric to measuring the performance of each model. The challenging thing in this task is that one comment can have multiple types of toxicities at the same time. Therefore, this is a multi-label classification problem where one instance can have more than one target class labels.

## 3 Background/Related Work

Over the last few years, the sentiment classification of toxicity has been intensively studied, especially in the context of social media data. Researchers have used various machine learning systems to try and solve the toxicity problem. In 2017, Google and Jigsaw have released a project called Perspective, which uses machine learning to automatically identify toxic sentences. A demo site (Google, 2017) has also been launched that allows anyone to type a phrase in the interface and immediately see toxicity ratings. Existing research on sentiment classification and neural network approaches for sentiment classification are reviewed in the following section:

### 3.1 Effective LSTMs for Target-Dependent Sentiment Classification

Duyu and Bing (Tang et al., 2016) developed two target dependent long short-term memory (LSTM)

models on a benchmark dataset from Twitter. The first model is standard LSTM, the second is target-dependent LSTM models. The results shows target-dependent LSTM attained a best prediction accuracy of 0.715. Their finding indicate the advantages of target-dependent LSTM model on the target-dependent sentiment classification.

### 3.2 Convolutional Neural Networks for Sentence Classification

Yoon Kim (Kim, 2014) has proposed a series of experiments with convolutional neural networks (CNNs) that have been trained on top of pre-trained word vectors for sentence-level classification tasks. It shows that a simple CNN with a limited hyper-parameter tuning and a static vector achieves excellent results on multiple benchmarks. Moreover, a simple modification of the architecture was documented to allow for the use of both task-specific and static vectors. The findings show that on 4 out of 7 tasks, including sentiment analysis and query classification, the CNN models addressed herein enhance the state of the art.

### 3.3 Exploring Data Augmentation for Toxic Comment Classification

A combination of data augmentation techniques and machine learning algorithms was suggested by Chetanya and Nikka (Rastogi et al., 2020). In their experiment, they examined three common learning algorithms with Easy Data Augmentation and Back-translation, including Logistic Regression, Support Vector Machine (SVM), and Bidirectional Long Short Term Memory Network (Bi-LSTM). They use the Wikipedia Toxic Comments dataset to exploring the benefits of data augmentation. The idea of Back-translation was translate text to another language then back translate to original language. In the results, they found that data augmentation techniques can be used to significantly improve classifier efficiency, and it is an outstanding strategy to tackle unstructured data in nature language problems.

### 3.4 Comparison of Word Embedding for Tweet Classification Tasks

Hongmin and Xukun (Li et al., 2018) proposed a classifiers for filtering crisis tweets by using a simple feature based adaptation method where tweets are represented with word embedding or sentence encoding as dense numerical vectors of reduced dimensional. They compared tweet representation of different word embedding and sentence encoding in order to understand what embedding are more suitable for use in crisis tweet classification task. It appears in their results that the GloVe embedding-based tweet representations yield better results than the representations that use other embedding. In addition, the embedding of GloVe trained on crisis data yields better results while the embedding of GloVe pre-trained on a wide selection of general tweets.

### 3.5 TF-IDF vs Word Embedding for Morbidity Identification

Danilo and Rim (Dessì et al., 2020) proposed a Deep learning model based on Bidirectional Long-Short Term Memory layers to identifying sixteen morbidity types within textual descriptions of clinical records. The pre-trained word embedding GloVe and Word2Vec are employed for the LSTM. Furthermore, they have compared the performances of the deep learning approaches against the traditional TF-IDF using Support Vector Machine. In their results, it seems that the TF-IDF outperform the combination of Deep Learning approaches using any word embedding. This happens because the dataset they used to train the embedding is not large enough to allow the algorithm to learn domain peculiarities. And the results indicate that there are unique features that make the dataset skewed in favour of traditional machine learning approaches.

## 4 Approach

This project concentrates on various strategies - TF-IDF and Word Embedding - to study the effects of various deep learning approaches. A summary of the tasks performed is as follows:

1. Data Pre-processing and Split the data to Train, Validation and Test
2. Method 1:
   (a) Convert comments to TF-IDF on train data
   (b) Build Classification Models
      i. Logistic Regression
      ii. Naive Bayes
   (c) Assess the models on the validation results and tune the models

(d) Test performance of final model on Test dataset

3. Method 2:

(a) Train a word embedding model on a pre-trained model

(b) Using the word embedding model to transform text to word vector

(c) Build Classification Models

i. CNN - Word2Vec

ii. CNN - GolVe

iii. Long Short Term Memory Networks

(d) Assess the models on the validation results and tune the models.

(e) Test performance of final model on Test dataset.

Below, we detail these approaches:

### 4.1 TF-IDF

TF-IDF is a statistical measure that assesses in a series of documents how relevant a word is to a document. This is achieved by multiplying two metrics: how many times in a document a word appears, and the word's inverse document frequency over a set of documents (Stecanella, 2019). This is especially helpful because the model would see far more occurrences of a specific word and the corresponding target value than the sequence of terms.

#### 4.1.1 Logistic Regression

Logistics regression is common and is a useful regression method for solving the binary classification problem. Also, Logistic Regression is a Machine Learning algorithm which is used for the classification problems(Pant, 2019).Fit the data to a linear regression model, and then operate on it by predicting the logistic function of the target categorical dependent variable. In order to predict which category the data belongs to, a threshold can be set. Based on the threshold, the obtained estimated probabilities are classified.

#### 4.1.2 Naive Bayes

The Naive Bayes classifier is one of the simple and most effective Classification algorithms and is a simple classifier that classifies based on probabilities of events. It is the applied commonly to text classification. Though it is a simple algorithm, it performs well in many text classification problems which helps in building the fast machine learning models that can make quick predictions.

### 4.2 Word Embedding

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation(Brownlee, 2017). An embedding is a relatively low-dimensional space into which you can translate high dimensional vectors. Embedding make it easier to do machine learning on large inputs like sparse vectors representing words. Ideally, an embedding captures some of the semantics of the input by placing semantically similar inputs close together in the embedding space. We implement a function that transforms the text to word vectors. Basically, we put the word index into the pre-trained embedding layer before the neural network. The embedding vector is a pre-trained word vector space from a large corpus, and the output of the embedding layer is just a list of word coordinates in the vector space(He, 2019). We can also use the distance of these coordinates to detect relevance and context. Words with a closer meaning will be located closer in the word space.

#### 4.2.1 Word2vec CNN

In the absence of a large supervised training set, initializing with word vectors obtained from an unsupervised neural language model is a popular method to improve performance. We use publicly available word2vec vectors trained on 100 billion words in Google News. These vectors have a dimension of 300 and are trained using a continuous bag-of-words architecture, which randomly trains words that do not appear in the pre-trained word set. Once the Word2vec representation of the label is obtained, we can train a neural network that can predict the category of a given input label. They only accept fixed-size digital inputs. But we have some labels that are strings. And we got the word embedding from the Word2vec model. The Keras library has all the necessary conditions to bring all these together. It provides an embedding layer that can be used to train neural networks on text data. You can initialize it with random weights, or you can use word embeddings learned elsewhere. We use Tokenizer to encode tags, and use Embedding layer to tell NN the word embedding in the Word2vec model. In addition, we allow the NN to modify the embedding during training.

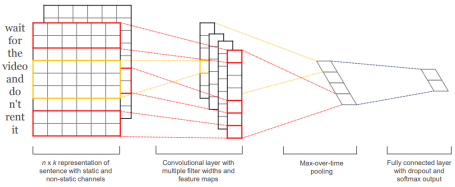


Figure 1: CNN model (Kim, 2014)

### 4.2.2 GloVe CNN

I use word embeddings from pre-trained Glove that was trained on a dataset of one billion tokens (words) with a vocabulary of 400 thousand words. First, we download the glove embedding. These files contain the mapping of each word to a 300-dimensional vector, also known as embedding. These embeddings are based on the probability of simultaneous occurrence of words. Then, we convert each text file into fixed length sequence of words by padding. We create embedding layer that contains mapping for words into glove vectors. we extract word embeddings to create an embedding matrix, and the embedded matrix contains valuable information for the CNN's embedding layer. Now, we can create a convolutional neural network. We chose CNN because the attributes that make CNN useful for images also apply to text. They preserve the relationship and order of the input, because pixel order is important in image processing, and text order is also important in classification.

### 4.2.3 RNN-LSTM

Recurrent Neural Network (RNN) is similar to CNN and has internal state (memory) to process the input sequence. We use long short-term memory (LSTM) to preserve the input order (words make up a sentence) and help understand the context for better classification. We use the default keras embedding layer to generate a vector of length 300. First encode the text data so that each word is represented by a unique integer. This data preparation step can be performed using the Tokenizer API included with Keras, and we add padding to make all vectors have the same length. The embedding layer requires the specification of the vocabulary size, the size of the real-valued vector space EMBEDDING-DIM = 300, and the maximum length of input documents max-length. Now we are ready to define the neural network model. The model will use the embedding layer as the first hidden layer. The embedding layer will be initialized with random

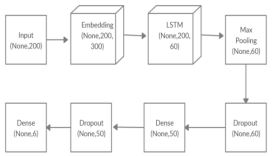


Figure 2: LSTM Architecture

| | id | comment_text | toxic | severe_toxic | obscene | threat | insult | identity_hate |
|---|---|---|---|---|---|---|---|---|
| 0 | 0000997932d777bf | Explanation\nWhy the edits made under my usern... | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 000103f0d9cfb60f | D'aww! He matches this background colour I'm s... | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 000113f07ec002fd | Hey man, I'm really not trying to edit war. It... | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0001b41b1c6bb37e | "\nMore\nI can't make any real suggestions on ... | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0001d958c54c6e35 | You, sir, are my hero. Any chance you remember... | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 3: Random data sample

weights and will learn the embedding of all words in the training data set during model training.

## 5 Experiments

### 5.1 Dataset

The dataset of comments from Wikipedia's talk page edits is also used (AI, 2017). The training data includes 159,571 labeled comments for toxic actions labeled by human raters, and we split 20% of training data as validation set. The test data containing 63,978 unlabeled comments. Figure 3 shows the random data sample of the comments, and the toxicity labels are: toxic, severe toxic, obscene, threat, insult and identity hate. The correlation matrix of the categories is given in Figure 4. The correlation matrix shows 'toxic' is correlated with 'obscene' and 'insult' (0.68 and 0.65). 'toxic' and 'severe_toxic' have a correlation factor of 0.74. This implies 'insult' and 'obscene' are highly correlated. The major concern of the data is that the bulk of comments are non-toxic. There are just a few observations on the training data for labels like 'threat,' as seen in Figure 5. This suggests that we need to deal with imbalanced classes and indeed,

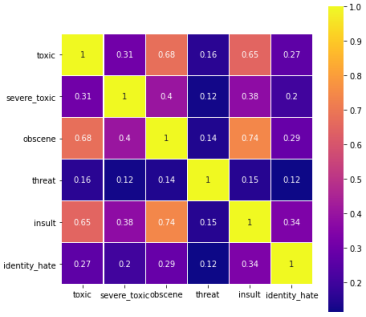


Figure 4: Correlation of features and targets

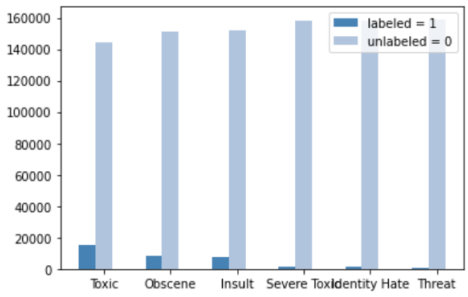


Figure 5: Number of labels

we use various methods to solve this problem, such as choosing robust models.

#### 5.1.1 Data Preprocessing and Augmentation

Prior to performing any machine learning on the dataset, it was required that we sift through the dataset and cleanse it of impurities and discrepancies. Our preprocessing steps included:

1. Translate Non-English word to English word use Google Translate API

   The idea is that use Google Translate API to translate comments to intermediate language and then translate it back to English. This is a good way to reduce overfitting and allow us to train deeper models (Chaudhary, 2020).

2. Convert All Character to Lowercase

   In order to facilitate subsequent data processing, we convert all text to lowercase letters at first.

3. Replace Emoticons with Word

   We found some emoticons such as ':)', ':P' and so on and we replaced them with 'smile'. For some emoticons such as ':(' and ':-(', we replaced them with 'sad'.

4. Rewrite Abbreviations

   In the text, there have a lot of abbreviations such as 'i'm', 'didn't', and 'won't'... We rewrite them to separate words. For example: 'i'm' to 'i am', 'didn't' to 'did not'...

5. Replace Date, Phone Number, and Website Links

   We replace various forms of dates with the word 'DATE', replace phone number with the word 'PHONE', and replace the website link with the word 'URL'.


Figure 6: Example of Feature Names

6. Remove all numbers and punctuation:

   We remove all numbers from 0 to 9, we remove all punctuation such as '?','@','!',etc...

7. Normalize Repeating Characters

   Some text in our dataset such as 'likeee' and 'ohhhhh' contain repeating characters in word. We normalize them to 'like' and 'oh'.

8. Remove Stopwords

   We removed the stopwords in text of our dataset, The set of stopwords we token from a Python library called 'nltk'. For later task, stopwords are removed from the given text so that more focus can be given to those words which define the meaning of the text.

### 5.2 Evaluation Metrics

We use an accuracy metric to measure the performance of the classifier. The accuracy score is the proportion of the total number of predictions that were correct. For this project, we considered two variants of accuracy assessment. One is reported accuracy on the basis of the correct classification of all 5 classes and the other one is reported label accuracy over all datasets.

### 5.3 TF-IDF

We have converted comments to the TF-IDF matrix using the sklearn provided libraries. TF-IDF converted the entire document into N number of unique terms T = t1,t2,...,tN and noted their respective frequency F = f1,f2,...,fN. In our case, the value of N is 136,829, which generates a feature space of that value. At first, it seems difficult to have a large number of unique terms in the comments, since we do not generally use that many words. But as you can see in Figure 6, many obscure words or phrases have no meaning, or only strings of the same characters are used in the comments.

To make the training process computational cost

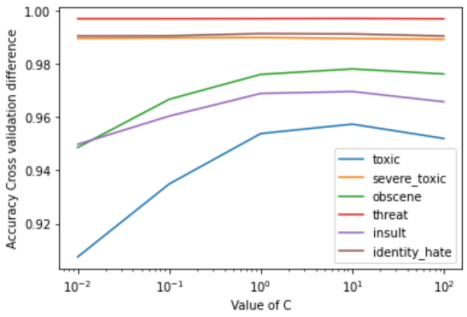


Figure 7: Accuracy v.s. Regularization Values

faster, we decided to remove some of the words from the text. We removed stop words like 'I,' 'and a' has no degrading meaning and is used to make the sentence grammatically correct. As a result, the number of features has been drastically decreased.

### 5.3.1 Logistic Regression

Logistic regression was the first model that we constructed. Each class was considered independent in this approach, therefore, we fed as input the converted comments by TF-IDF and use sklearn's default LogisticRegression package trained six models for six classes on training data. The total training time for six models around 57s. Predictions on validation were comparable but in order to solve overfitting, there was still scope for improvement. As Figure 7, we modified the regularization values to tackle overfitting and we find an optimal value that reduces overfitting and holds the accuracy score high enough. A test accuracy of 0.9722 was obtained by doing this.

### 5.3.2 Naive Bayes

Our second model is Naive Bayes. To do this, we fed as input the converted comments by TF-IDF into sklearn's naive_bayes package. The approximately training time is 0.6s, this is much faster than Logistic Regression. We tune the alpha parameter using cross validation to find the best performance for each toxicity label. Finally achieved an accuracy score 0.9708 on test data.

## 5.4 Word Embedding

Another advanced text representation we used is word embedding. The input data is prepared by tokenizing the text, convert words to index, padding the sequence to preserve a set size, and replacing the index with pre-trained word vectors using the Keras package in Python. This results in a 2-dimensional sequence representing the comments

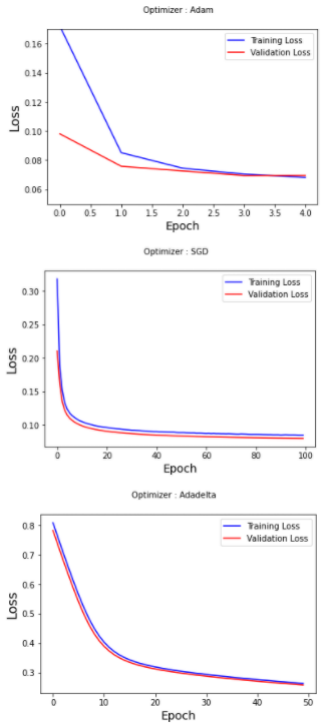


Figure 8: Performance vs Epoch for Various Optimises

data that would be used to train a Neural Net. The validation and test data is also converted to a 2-dimensional array that will be used for tuning and testing.

### 5.4.1 Word2vec CNN

To implement CNN architecture, we use Google Co-laboratory as a platform to run Jupyter Notebook with GPU support. For the CNN model we varied kernel size, pool size, the number of hidden layers, batch size and epoch. Different optimization algorithms are applied and the best the optimizer is selected by the loss function. As Figure 8, an example of performance vs epoch for different optimiser. We also used validation dataset to early stop the model train, so whenever the validation loss increases we stop training our model.

For input with Word2Vec embedding, training time for each epoch is around 12s using Google Co-laboratory with GPU support. We found Adam optimizer with learning rate = 0.001 gives has lowest loss on both training set and validation set. Adam optimization is a method of stochastic gradient descent focused on first-order and second-order moments' adaptive estimation (Kingma and Ba, 2017). Finally the accuracy on test dataset is 0.9724.

#### 5.4.2 GloVe CNN

For input with GloVe embedding, training time for each epoch is around 8s using Google Co-laboratory with GPU support. We found the Gradient descent optimizer (GSD) with learning rate = 0.01, momentum = 0.9, and = decay 0.0001 has the best performance on both training set and validation set. Finally it achieved 0.9730 accuracy score on test dataset.

#### 5.4.3 RNN-LSTM

A recurrent neural network is similar to a CNN in that it has an internal state to process the input sequences. We use LSTM because it preserves the input sequences and assist in understanding the context for better classification. To generate a vector of length 300, we used the default Keras embedding layer.

The data is prepared with Keras embedding. The 2-dimensional input is used to train a LSTM model with one hidden layer at 64 output units per layer. It took around 37s to train the model on Google Co-laboratory with GPU support. By incorporating additional layers, which would take more computational power, the model can be further improved. On the test dataset, this model achieved 0.9738 accuracy score, which is the best performance of all models.

### 5.5 Results

Table 1 is the accuracy results of the TF-IDF method we tried:

Table 1: Method 1 Results

| Models | Validation | Test |
|---|---|---|
| Logistic Regression | 0.9806 | 0.9722 |
| Naive Bayes | 0.9769 | 0.9708 |

Table 2 is the accuracy results of the word embedding method we tried:

Table 2: Method 2 Results

| Models | Embedding | Validation | Test |
|---|---|---|---|
| CNN | Word2Vec | 0.9793 | 0.9724 |
| CNN | GloVe | 0.9783 | 0.9731 |
| LSTM | Keras Embedding | 0.9787 | 0.9738 |

#### 5.5.1 Overall Quantitative Results

For TF-IDF method, we achieved a best performance of 0.9722 accuracy on test data from our Logistic Regression model. Besides, The Naive Bayes achieved 0.9708 accuracy on test data.

For word embedding method, we achieved a best performance of 0.9738 accuracy on test data from our LSTM. And CNN with Word2Vec embedding achieved 0.9724 accuracy, CNN with GloVe embedding achieved 0.9731 accuracy.

Overall, LSTM performed highest on our classification task. In addition, the word embedding method is outperformed the TF-IDF method.

#### 5.5.2 Overall Quantitative Evaluation

Compare Logistic Regression and Naive Bayes, Naive Bayes models the joint distribution of the feature and target label, and then predicts the posterior probability given as p(feature — target). Logistic regression directly models the posterior probability of p(target — feature) by learning the input to output mapping by minimizing the error (Ottesen, 2017). Naive Bayes assumes that all features are conditionally independent. However, in our case, the toxicity labels are correlated to each other, therefore, the performance of Naive Bayes is poor compare with other models.

Both word2vec and gloves enable us to represent a word in vector form. They are the two most popular word embedding algorithms that can reveal the semantic similarity of words and thus capture different aspects of word meaning (MLNerds, 2019). Word2vec embedding is based on training a shallow feed forward neural network, while glove embedding is learned based on matrix factorization technology (MLNerds, 2019). The structure of glove is simpler than word2Vec, so glove is faster than word2vec. Also, word2vec focuses on local information, and glove focuses on local and global information. In our case, we train CNN model using two different word embeddings, and the results of CNN using glove has better performance.

According to the result from Table 2, LSTM is the best performance in our test because when training the neural network for classification problems, we can learn the word embedding of our dataset. Before presenting the text data to the network, the text data is first encoded so that each word is represented by a unique integer. We perform this data preparation step using the tokenizer API that comes with keras, and we add padding so that all vectors have the same length. The model uses the embedded layer as the first hidden layer. The embedding

layer will be initialized with random weights, and will learn to embed all the words in the training data set during the model training(Nabi, 2018).

In general, from the obtained results Table 1 and Table 2, we can see the accuracy of all deep learning models fed by the word embedding representations is higher than model fed by TF-IDF. This indicates embedding representation enable the deep learning model to better classifying toxicities. Moreover, although the results in literature TF-IDF vs Word Embedding for Morbidity Identification (Dessì et al., 2020) shows the TF-IDF is outperformed pre-trained embedding. But our dataset is large enough to allow the algorithm to learn domain peculiarities. Therefore, the accuracy score of our word embedding model is higher than TF-IDF method.

### 5.6 Conclusion

In this paper, we have investigated two methods TF-IDF and word embedding. The LSTM with word embedding yielded the best performance. In general, the word embedding method is outperformed the TF-IDF method. It seems to be extracting semantics could help us hinder the antisocial content. Natural language processing is a vast area and there is a lot of scope for experimentation. We have done many research on the conversion of text data using techniques such as 'Word2Vec' and 'GloVe' were learned.

In the feature, we aim to achieve better performance by using more complex deep learning models.We would like to assess the effect of using Duyu and Bing's (Tang et al., 2016) target-dependent LSTM models.

## References


Jigsaw/Conversation AI. 2017. Toxic comment classification challenge.

Jason Brownlee. 2017. What are word embeddings for text?

Amit Chaudhary. 2020. Back translation for text augmentation with google sheets.

Danilo Dessì, Rim Helaoui, Vivek Kumar, D Reforgiato Recupero, and Daniele Riboni. 2020. Tf-idf vs word embeddings for morbidity identification in clinical notes: An initial study. In *1st Workshop on Smart Personal Health Interfaces, SmartPhil 2020*, volume 2596, pages 1–12. CEUR-WS.

Jigsaw Google. 2017. Perspective. https://www.perspectiveapi.com/.

Zhixing He. 2019. Toxic comment classification.

Yoon Kim. 2014. Convolutional neural networks for sentence classification.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization.

Hongmin Li, D Caragea, X Li, and Cornelia Caragea. 2018. Comparison of word embeddings and sentence encodings as generalized representations for crisis tweet classification tasks. *en. In: New Zealand*, page 13.

MLNerds. 2019. What is the difference between word2vec and glove ?

Javaid Nabi. 2018. Machine learning — word embedding sentiment classification using keras.

Christopher Ottesen. 2017. Comparison between naïve bayes and logistic regression.

Ayush Pant. 2019. Introduction to logistic regression.

Chetanya Rastogi, Nikka Mofid, and Fang-I Hsiao. 2020. Can we achieve more with less? exploring data augmentation for toxic comment classification.

Bruno Stecanella. 2019. What is tf-idf?

Duyu Tang, Bing Qin, Xiaocheng Feng, and Ting Liu. 2016. Effective lstms for target-dependent sentiment classification.