**Exploratory Data Analysis**

1. Is non-toxic really non-toxic?

2. Does the length of comments matter in making predictions?

3. Does the number of uppercase matter?

4. The frequent words only in toxic

**Word Similarity Score**

1. Word2Vec

2. FastText

**Data Pre-processing**

1. Contractions

2. Remove spaces

3. Lowercase

4. Stopwords

5. Tokenization

**Feature Engineering**

1. Bag of Words

2. Bag of n-Grams

3. TF-IDF

**Supervised Learning Methods**

1. Naive Bayes Classifier

2. Logistic Regression

* without tf-idf : Bag-of-Word, Bag of 2-grams (bigram), Bag of 3-grams (trigram)
* with tf-idf : Bag-of-Word, Bag of 2-grams (bigram), Bag of 3-grams (trigram)

3. RNN (bidirectional)

4. LSTM (bidirectional)

5. BERT

**Unsupervised Learning Methods**

1. Vector Encoding

- TF IDF

- Bag of n-grams/Bag of words

2. Dimensionality Reduction

- Truncated SVD

3. ML Models on raw data

- Self Organized Map, Gaussian Mixture Models, Hierarchical clustering, K-means, Density Based Scans

4. Clustering Models on BERT Encoded Vectors

- GMM, K-means, DB Scan,

**Results and Discussions**

**Exploratory Data Analysis**

**0. Dataset**

We use the dataset in a Kaggle competition.

The dataset includes 159,571 Wikipedia comments and six labels: toxic, severe\_toxic, obscene, threat, insult, and identity\_hate.

Our goal is to build a model which predicts the toxicity of the comment.

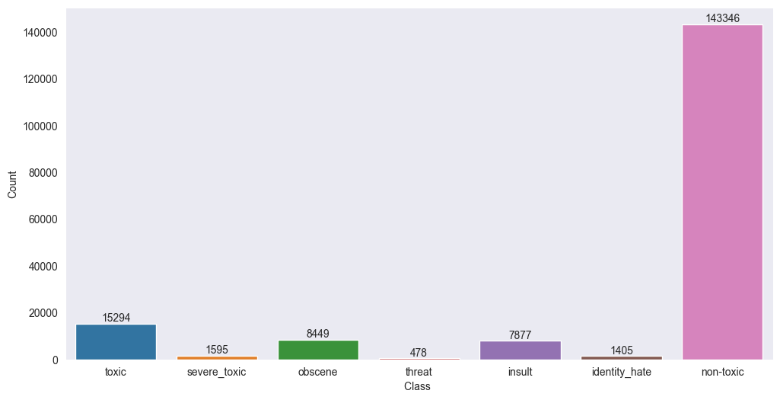
텍스트, 실내, 은색, 스크린샷이(가) 표시된 사진

자동 생성된 설명

The number of comments classified in each category is as follows. Most of the comments are classified as non-toxic as below, which indicates that the dataset is imbalanced and we need to handle this. Some classes such as “threat” or “identity\_hate” obviously have fewer comments than others.

텍스트이(가) 표시된 사진

자동 생성된 설명



**1. Is non-toxic really non-toxic?**

We discover that some comments are labeled as non-toxic even though they have other labels (obscene, threat, insult, identity\_hate). For example, there are 523 comments that are considered obscene but not toxic. This would not be ideal for us, so we decide to label those comments as toxic.

텍스트이(가) 표시된 사진

자동 생성된 설명

Figure: Before

텍스트, 모니터, 검은색, 화면이(가) 표시된 사진

자동 생성된 설명

Figure: After

Examples of toxic comments

\* Disclaimer: These comments have several bad words.

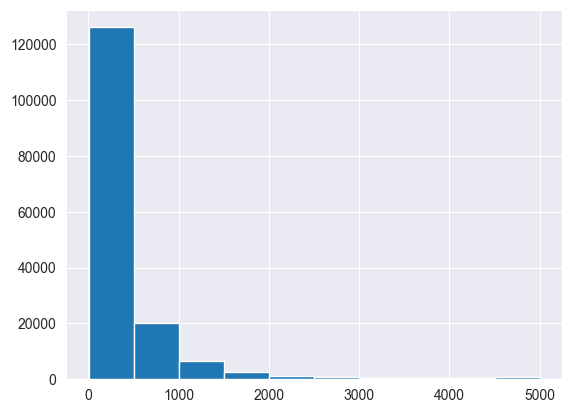
toxic: Hi Im a fucking bitch.

severe\_toxic: What a motherfucking piece of crap those fuckheads for blocking us!

Threat: I think that your a Fagget get a oife and burn in Hell I hate you 'm sorry we cant have any more sex i'm running out of conndoms

**2. Does the length of comments matter in making predictions?**

The length of most sentences is less than 2000 words.



We compare the distribution of the length of the comments for both toxic and non-toxic comments. The box plot and density plot for the length of comments showed different distributions for each category. In addition to that, the longest comments seem to have a higher chance of toxicity.

Figure: Length of comments

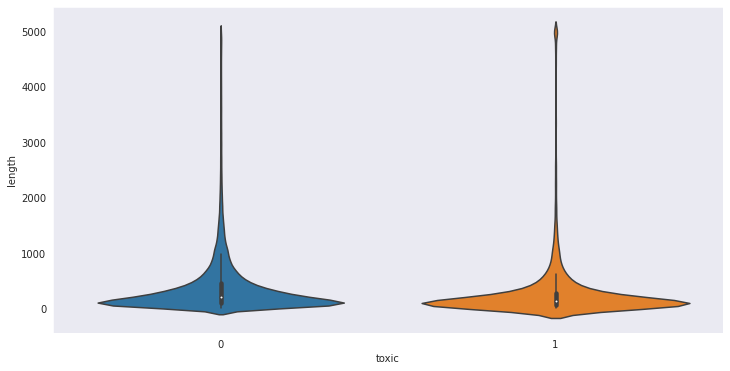


Figure: Violin plot for the length of comments

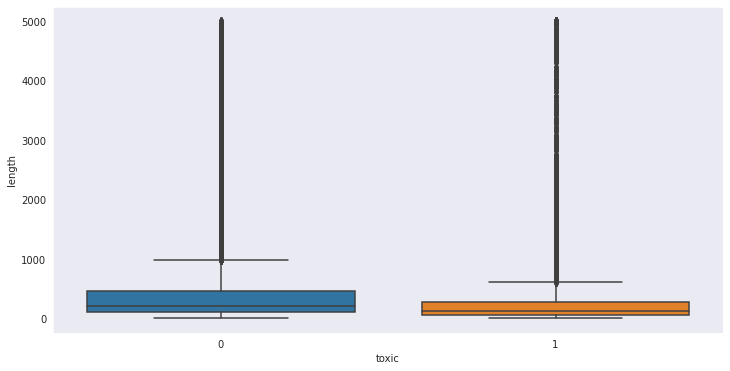


Figure: Box plot for the length of comments

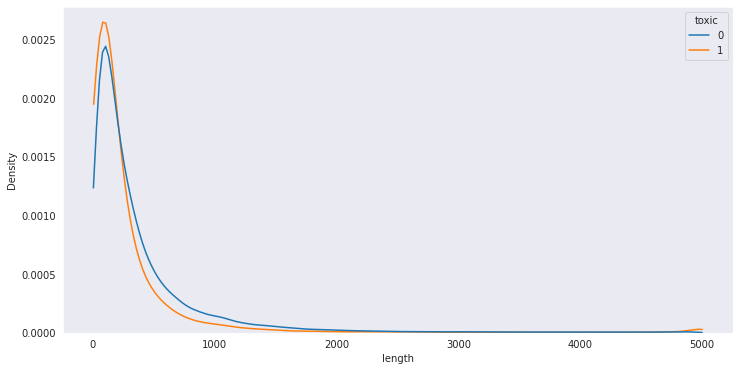


Figure: Density plot for the length of comments.

**3. Does the number of uppercase matter?**

We compare the number of uppercase because toxic comments may have more uppercase letters to stress their words. With violin and density plots, we conclude that the number of uppercase letters can be an effective feature since toxic comments tend to include a greater number of uppercase letters indeed.

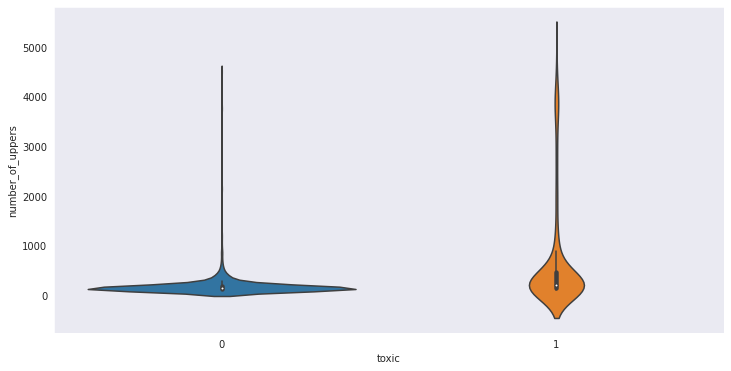


Figure: Violin plot for the number of uppercase letters

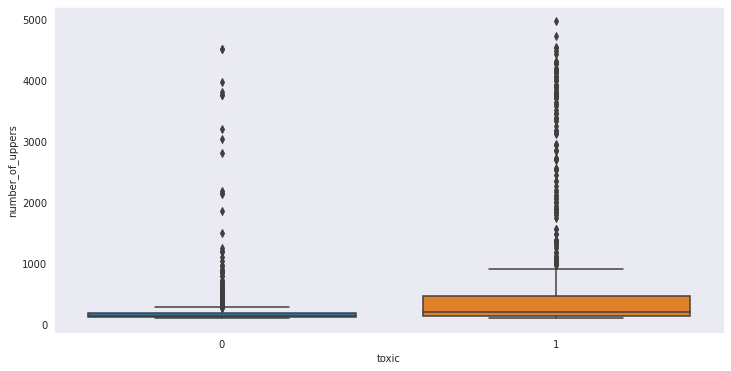


Figure: Box plot for the number of uppercase letters

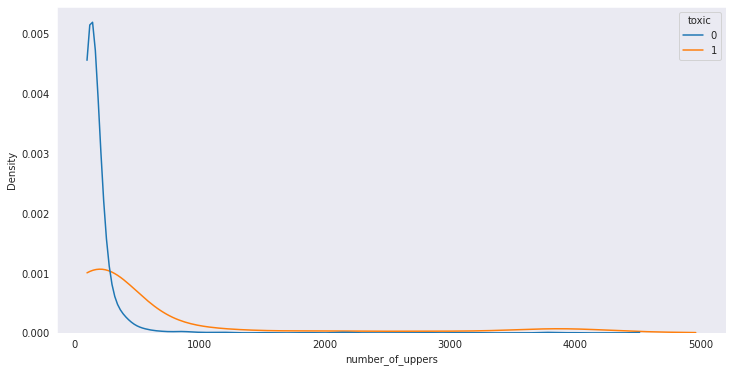


Figure: Density plot for the number of uppercase letters

**4. The frequent words only in toxic**

Looking into words’ frequency in each category, toxic and non-toxic comments, we exclude the most frequent common words such as “Wikipedia.” We include the number of occurrence of the most frequent words that are only in toxic comments as a feature.



Figure: Word Cloud for non-toxic comments



Figure: Word Cloud for toxic comments

The most frequent words only in toxic comments :

{'administrator', 'admins', 'aids', 'alone', 'american', 'anal', 'anti', 'ass', 'assad', 'asshole', 'attack', 'away', 'balls', 'ban', 'banned', 'bark', 'bastard', 'bastered', 'big', 'bitch', 'bitches', 'bollocks', 'boobs', 'bot', 'bullshit', 'bum', 'bunksteve', 'bush', 'buttsecks', 'call', 'care', 'chester', 'china', 'cock', 'cocks', 'cocksucker', 'cocksucking', 'computer', 'crap', 'criminalwar', 'cunt', 'cunts', 'd', 'damn', 'deleting', 'dick', 'dickhead', 'die', 'dog', 'dont', 'dumb', 'eat', 'en', 'everyone', 'f', 'face', 'fag', 'faggot', 'faggots', 'fart', 'fat', 'fggt', 'fool', 'freedom', 'fuck', 'fucker', 'fuckin', 'fucking', 'fucksex', 'gay', 'god', 'gonna', 'guy', 'guys', 'ha', 'hanibal911you', 'hate', 'head', 'heil', 'hell', 'hey', 'hitler', 'homo', 'huge', 'idiot', 'im', 'jew', 'jewish', 'kill', 'kind', 'lick', 'live', 'lol', 'loser', 'love', 'man', 'marcolfuck', 'mexicans', 'mitt', 'moron', 'mother', 'mothjer', 'nazi', 'nice', 'niggas', 'nigger', 'nipple', 'noobs', 'notrhbysouthbanof', 'offfuck', 'oxymoron', 'pathetic', 'penis', 'piece', 'pig', 'poop', 'power', 'prick', 'pro', 'pussy', 'racist', 'rape', 'retarded', 'romney', 'rules', 'self', 'seriously', 'sex', 'sexsex', 'sexual', 'shit', 'shut', 'small', 'stupid', 'suck', 'sucks', 'super', 'truth', 'twat', 'u', 'ur', 'useless', 'wanker', 'warning', 'whatever', 'whore', 'wo', 'yeah', 'yourselfgo'}

**Word Similarity Score**

We calculate similarity score to check the relationship between words. We remove words with frequency less than two to reduce noise.

1) Word2Vec

Word2Vec model compute the distance between two word vectors with Cosine distance metric.

2) FastText

FastText is an open-source, free library from Facebook AI Research(FAIR). It represents each word as an n-gram of characters. It even allows typo, so it is reputed to perform better than word2vec. We can see that FastText tends to capture words with similar spelling.

| input | Most similar words found by | |
| --- | --- | --- |
| Word2Vec | FastText |
| bitch | [('fuckin', 0.8765264749526978),  ('motherfucker', 0.8752584457397461),  ('fat', 0.8631212711334229),  ('scared', 0.8560019731521606),  ('dumbass', 0.8399372696876526),  ('youre', 0.8398669362068176),  ('goddamn', 0.8390710353851318),  ('wtf', 0.8365249633789062),  ('hes', 0.8347536325454712),  ('cunt', 0.8346415758132935)] | [('bitchh', 0.973738431930542),  ('bitchs', 0.9640547037124634),  ('bitchboy', 0.960981011390686),  ('bitchy', 0.9609492421150208),  ('bitchass', 0.9520701766014099),  ('bitchfuck', 0.9296033978462219),  ('bitches', 0.8995120525360107),  ('biitch', 0.8946561813354492),  ('sonofabitch', 0.8901281356811523),  ('bitchmother', 0.8769140839576721)] |
| fuck | [('cunt', 0.7634404897689819),  ('shut', 0.7433393597602844),  ('asshole', 0.7140934467315674),  ('vanalism', 0.7136352062225342),  ('rediculous', 0.7105764150619507),  ('shitbag', 0.6995687484741211),  ('supressing', 0.696487545967102),  ('bitch', 0.695846676826477),  ('aubie', 0.6932984590530396),  ('malaysians', 0.692956805229187)] | [('fuckıng', 0.9229608774185181),  ('fucky', 0.9079781770706177),  ('fuckon', 0.9017402529716492),  ('fuckup', 0.898465633392334),  ('dumbfuck', 0.8974303007125854),  ('fuckwit', 0.895268440246582),  ('fucka', 0.8930884599685669),  ('bitchfuck', 0.8902262449264526),  ('fuckwits', 0.8886486887931824),  ('fuckyourself', 0.8855721354484558)] |
| good | [('bad', 0.7980550527572632),  ('proselytising', 0.6285301446914673),  ('wcc', 0.6180676221847534),  ('nice', 0.5918679237365723),  ('decent', 0.5837923884391785),  ('weisz', 0.5761880278587341),  ('excellent', 0.5696313977241516),  ('fine', 0.5652056932449341),  ('great', 0.5581843852996826),  ('mundane', 0.5571675300598145)] | [('agood', 0.8400813341140747),  ('hapgood', 0.8359401226043701),  ('goodfaith', 0.8095809817314148),  ('goodkind', 0.7996166944503784),  ('goood', 0.7951282858848572),  ('goodluck', 0.7882312536239624),  ('goodytwo', 0.766779899597168),  ('bad', 0.7569311857223511),  ('goode', 0.7497248649597168),  ('goody', 0.7493317127227783)] |
| like | [('prefer', 0.5287191867828369),  ('fishy', 0.5216623544692993),  ('strange', 0.5142408609390259),  ('matsuda', 0.5005139112472534),  ('fine', 0.4932643175125122),  ('silly', 0.48215726017951965),  ('better', 0.47990554571151733),  ('nice', 0.4735766649246216),  ('expect', 0.4650725722312927),  ('peeves', 0.46453726291656494)] | [('likey', 0.8004260659217834),  ('ilike', 0.7885597944259644),  ('rlike', 0.7871559858322144),  ('liken', 0.7652401328086853),  ('godlike', 0.7432641983032227),  ('dreamlike', 0.7424696087837219),  ('likn', 0.7343519330024719),  'childlike', 0.7110942602157593),  ('suchlike', 0.7059680223464966),  'likeit', 0.7010021209716797)] |

**Data Pre-processing**

We use re - Regular expression operations - clean and handle the text data in Python.

**1. Contractions:** We replace apostrophe and short words for clear understanding.

i.e. what’s -> what is, can’t -> cannot, n’t -> not

**2. Remove spaces:** We remove unnecessary spaces before/between/after words.

**3. Lowercase:** We convert words to lower-case to prevent the algorithm from being confused by the same words with different cases of alphabets.

Result example:

'i think she meets notability guidelines and thanks for creating the page for her she was a big icon in the pageant drag community and in the drag gay community in general her death has been covered by several mainstream media outlets'

'hey man i am really not trying to edit war it just that this guy is constantly removing relevant information and talking to me through edits instead of my talk page he seems to care more about the formatting than the actual info'

'i think she meets notability guidelines and thanks for creating the page for her she was a big icon in the pageant drag community and in the drag gay community in general her death has been covered by several mainstream media outlets'

For deep learning architecture, we also do

**4. Stopwords**: We remove stopwords to reduce noise in the sentence. Stopwords are basically a set of commonly used words such as “I”, “am”, and “he”. For example, “i'm too tall but beautiful fuck” will be “tall beautiful fuck” after preprocessing.

**5. Tokenization**: We split the sentence into a smaller chunk to process text.

**Feature Engineering**

To transform text into numeric vectors, we try three methods.

**1. Bag of Words (BoW)**: It is a count-based method, which is simple and effective. Considering the number of times the word appears in the document, it creates a flat vector for the word. However, it does not take into account the order of the words.

**2. Bag of n-Grams**: It is an extension of Bag of Words, but it considers n-words together, so it can preserve the original sequence of the sentence. We call 1-gram unigram, 2-grams bigram, 3-gram trigram, etc. However, it is very expensive to compute.

**3. Term frequency-inverse document frequency (Tf-idf)**: TF is a ratio of (a specific word’s occurrence in certain sentence) to (the total number of words in that sentence). IDF is the natural log of the ratio of (the number of sentences in the dataset) to (the number of sentences that includes a specific word). It calculates whether a certain word is important or not. If a word appears frequently in a sentence, but also appears frequently in the whole dataset, it may not contain important clue. On the other hand, if a word appears frequently only in certain sentence, it is likely that it contains important clue.

**Supervised Learning Methods**

**1. Naive Bayes Classifier**

Naive Bayes is a simple, basic probablisic algorithm based on Bayes Theorem. It assumes all variables are conditionally independent. This assumption leads to significant reduction of model’s time complexity. However, such independence is not likely to happen in real-world. Also, the algorithm cannot work with the words not in the training dataset (zero-frequency problem).

We apply the classifier on tf-idf features. The accuracy is okay, but the f1-score is very low. The model does not handle imbalance data effectively, and tends to classify every data into non-toxic.

Training : Test :: 8 : 2

Using TF-IDF Features

| Result:  Accuracy: 91.77 %  F1-Score: 31.39 %  ROC-AUC Plot    Confusion Matrix |
| --- |

**2. Logistic Regression Classifier**

Logistic regression is a binary classification algorithm, which uses a sigmoid function that convert any real number to a number between 0 and 1. It is easy to implement, and efficient to train. However, it assumes the linearity between the dependent and the independent variables, and we cannot solve non-linear problems with it. Since we can rarely find linearly separable data, it is hard to apply it for the real-world problem.

We apply the classifier on six different features to compare their performance. Bag of 3-grams features show the highest accuracy and f1-score. Applying tf-idf does not significantly change the result.

Training : Test :: 8 : 2

|  | | Bag-of-Words | Bag of 2-grams  (bigram) | Bag of 3-grams  (trigram) |
| --- | --- | --- | --- | --- |
| without  Tf-idf | Accuracy | 90.11% | 92.58% | 95.12% |
| F1-Score | 6.9% | 49.56% | 72.89% |
| with  Tf-idf | Accuracy | 90.1% | 92.62% | 95.06% |
| F1-Score | 6.89% | 50.0% | 73.05% |

Without Tf-idf:

| 1) Using Bag-of-Words Features  ROC-AUC Plot    Confusion Matrix    2) Using Bag of 2-gram (bigram) features  ROC-AUC Plot    Confusion Matrix    3) Using Bag of 3-gram (trigram) features  ROC-AUC Plot    Confusion Matrix |
| --- |

With Tf-idf:

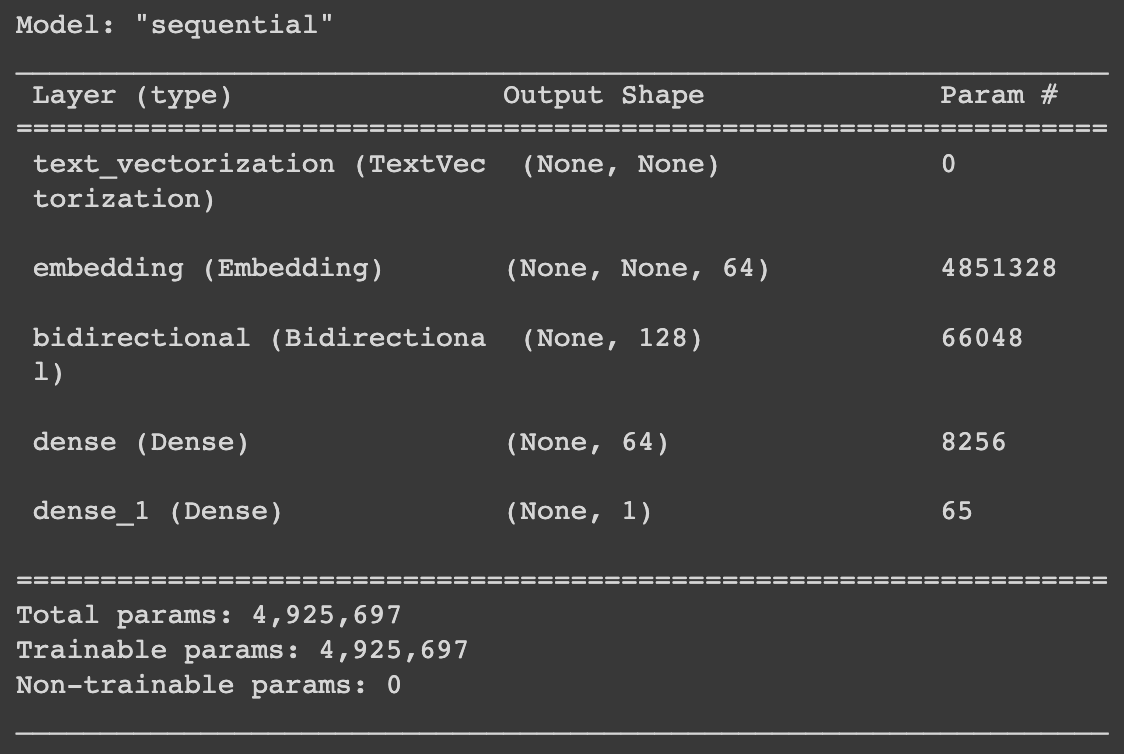
| 1) Using Bag-of-Words Features  ROC-AUC Plot    Confusion Matrix    2) Using Bag of 2-gram (bigram) features  ROC-AUC Plot    Confusion Matrix    3) Using Bag of 3-gram (trigram) features  ROC-AUC Plot    Confusion Matrix |
| --- |

**3. RNN (bidirectional)**

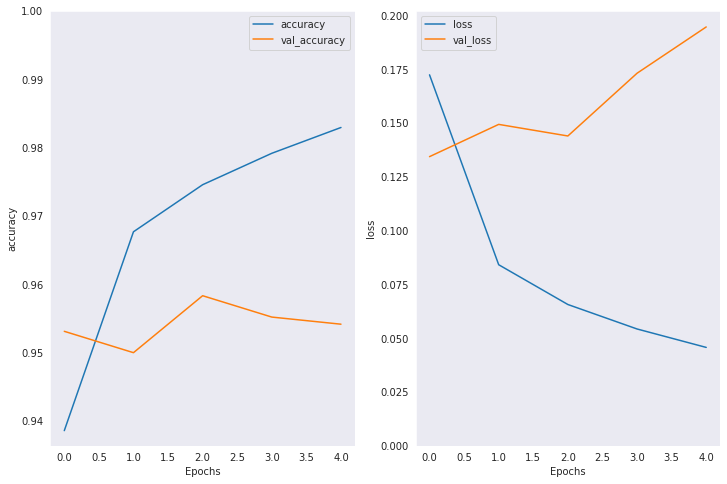
Recurrent Neural Network is a deep learning algorithm for sequential data. It remembers important information from the previous layer (input) to predict next layer (output). RNN uses backpropagation to optimize the model. Backpropagation calculates the gradient of an error function with respect to a weight of a neural network. However, this brings two problems: Exploding gradient and vanishing gradient.

We use bidirectional RNN and choose the values from epoch 3, which give relatively high accuracy for both train and validation datasets. The result shows better accuracy and f1-score compared to the previous models (Naive Bayes, Logistic Regression). However, it is still not precise. The model tends to classify sentences with bad words as toxic without considering the context. For example, for input like “Damn! I like your comment” and “This is fucking good!”, the model classify it as toxic (89.12%, 99.97%).

Model Summary



The change of Accuracy/Loss over Training Epoch



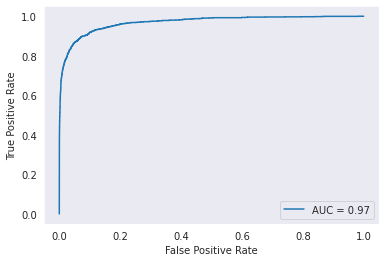
Training : Validation : Test :: 8 : 1 : 1

Result:

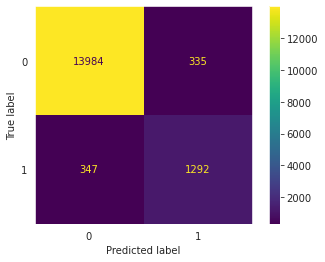
Accuracy: 95.73 %

F1-Score: 79.12 %

ROC-AUC Plot



Confusion Matrix



Predict Toxicity:

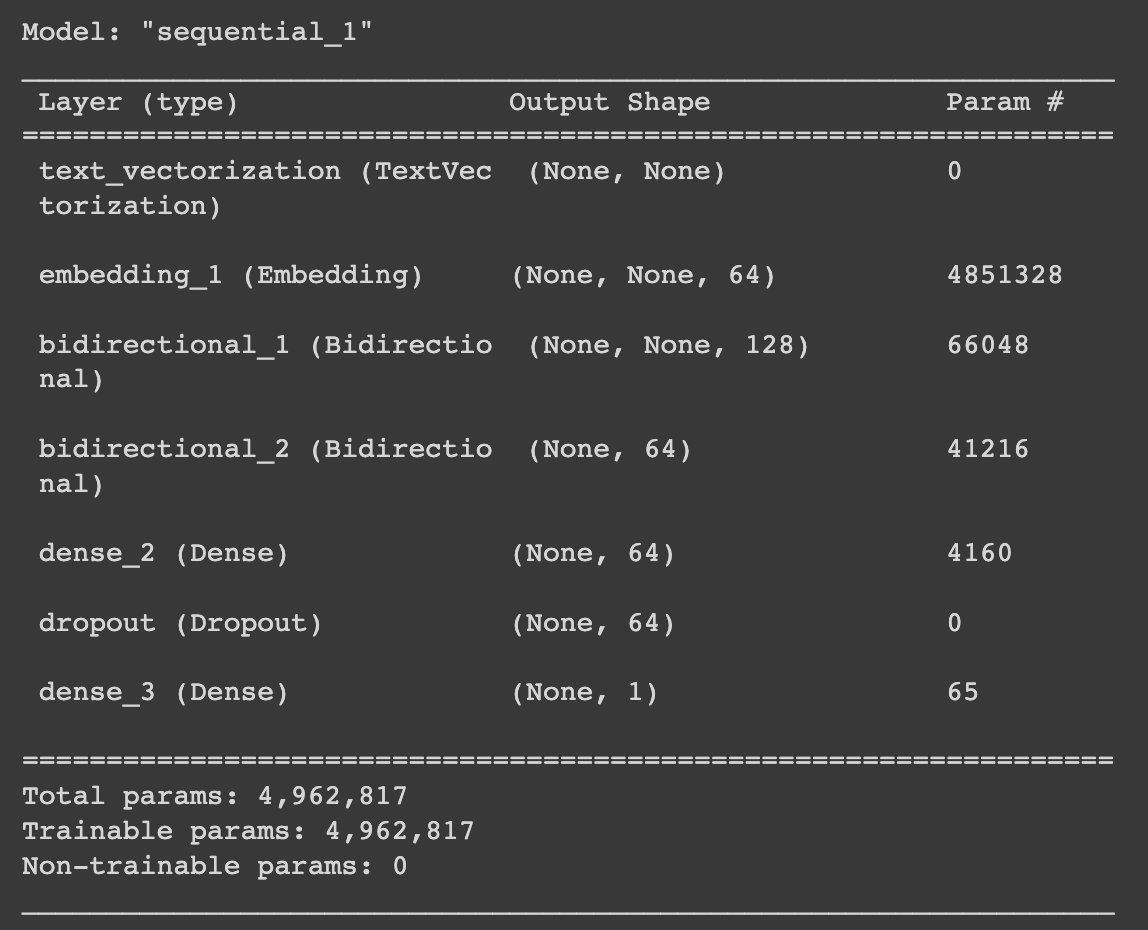
| input | RNN |
| --- | --- |
| I hate you Bitch | 99.99% |
| Fuck you | 99.98% |
| Damn! I like your comment | 89.12% |
| Okay I think it is good | 0.09% |
| This is fucking good! | 99.97% |

**4. LSTM (bidirectional)**

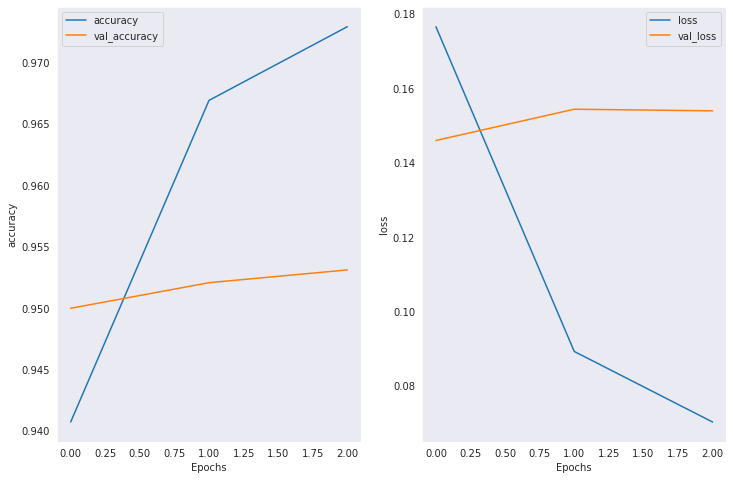
Long short-term memory is an extension of RNN, but it has a memory cell that decides which information to keep. Thus, it overcomes vanishing gradient issue of RNN.

We use bidirectional LSTM and choose the values from epoch 3, which give relatively high accuracy for both train and validation dataset. The result shows better accuracy and f1-score compared to the previous models (Naive Bayes, Logistic Regression, RNN). However, it is still not precise. The model tends to classify the sentence with bad words as toxic without considering the context. For example, for the input like “Damn! I like your comment” and “This is fucking good!”, the model classify it as toxic (86.44%, 99.54% - a bit lower than RNN).

Model Summary



The change of Accuracy/Loss over Training Epoch



Training : Validation : Test :: 8 : 1 : 1

Result:

Accuracy: 96.07 %

F1-Score: 80.5 %

| ROC-AUC Plot    Confusion Matrix |
| --- |

Predict Toxicity:

| input | LSTM |
| --- | --- |
| I hate you Bitch | 99.61% |
| Fuck you | 99.09% |
| Damn! I like your comment | 86.44% |
| Okay I think it is good | 0.18% |
| This is fucking good! | 99.54% |

**5. BERT**

Bidirectional Encoder Representations from Transformers is a deep learning model based on transformers and attention mechanism. It is one of the state-of-art technologies in the field of natural language processing.

Result:

|  | Small BERT  (original) | Remove  Stopwords | Give  Class-Weight | Add feature  # of Uppercase |
| --- | --- | --- | --- | --- |
| Accuracy | 96.64% | 96.42% | 96.11% | 96.6% |
| F1-Score | 83.3% | 81.96% | 81.73% | 82.97% |

Why BERT the BEST?

BERT encoder has the following advantages.

First, it can capture the sequence of words. Traditional ways of embedding such as bag-of-words or TF-IDF cannot incorporate the sequence of words, losing sequential information after encoding. BERT, on the other hand, uses Transformer blocks to encode, allowing embedding for each word to be affected by all the other words.

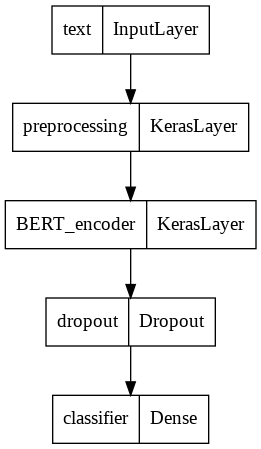
Second, pretrained models of many different versions of BERT are easily accessible. In general NLP tasks, one needs a large number of documents to train a model. Using pretrained BERT eliminates this expensive and time-consuming step.

1) Small BERT

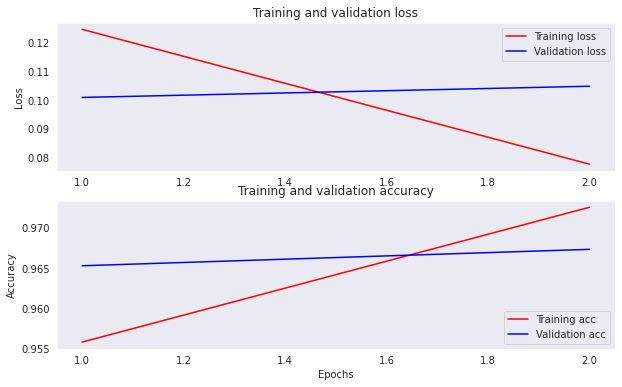
Among many different versions of BERT, we opt for “Small BERT” to effectively reduce training time and model complexity. Preprocessed comments are fed to the preprocessing layer and then to BERT encoder. Finally, we put the last output vector of the encoder into a Dense Block. We use Binary-Cross-Entropy as the loss function and train the model with AdamW optimizer.

One thing remarkable is that the model was able to classify non-toxic sentence with bad words as non-toxic. For example, for the input like “this is fucking great”, the model classify it as non-toxic (86.44%). This is a great improvement on previous deep learning models (LSTM, RNN).

Structure of Small BERT Model-1



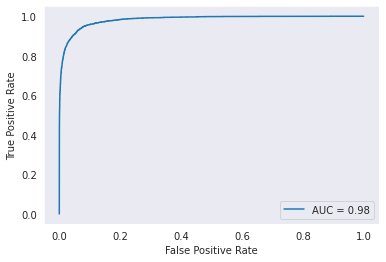
Loss & Accuracy of Small Bert Model-1



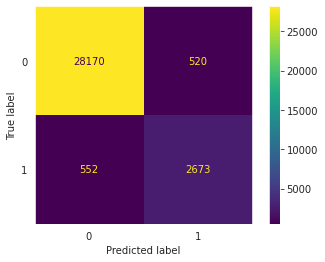
Predict Toxicity:

| input | BERT |
| --- | --- |
| i like the weather today | 0.51% |
| what do you want from me? | 0.36% |
| fuxk damn shit | 99.97% |
| what the fuck is wrong with u | 99.96% |
| your grandma wouldve done better than you | 4.55% |
| this is fucking great | 0.45% |

ROC-AUC Plot



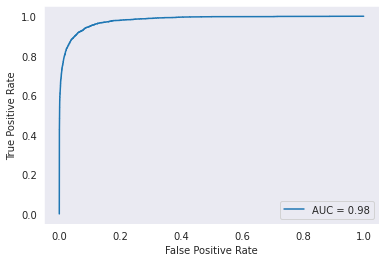
Confusion Matrix



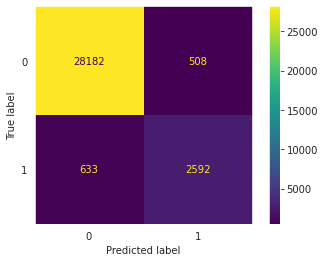
2) Small BERT (remove stopwords)

We also try the text without stopwords. Accuracy and AUC does not change, but F1-score drop significantly because the number of False-Negative cases increased. This means that the threshold for classification became more strict in Model-2, predicting less comments as toxic. We conclude that the model failed to capture the contextual sense of each comment, only relying on a set of words.

ROC-AUC Plot



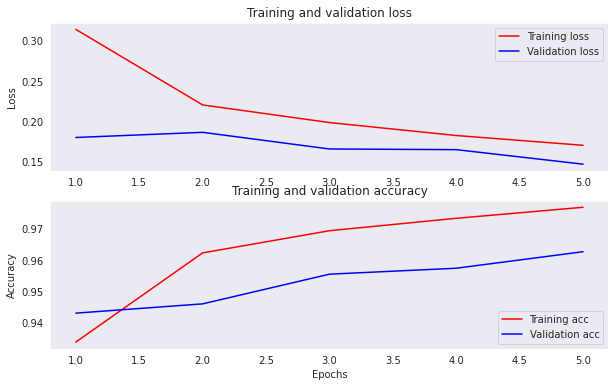
Confusion Matrix



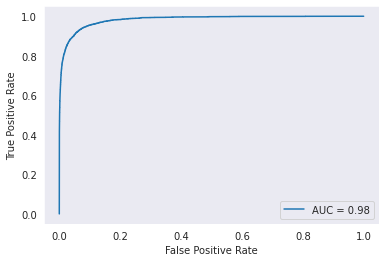
3) Small BERT (give class-weight)

To handle the imbalance of the dataset, we use class-weights approach, giving 10 times greater weight to toxic comments so that our model would focus heavily on toxic comments, predicting more comments as toxic.

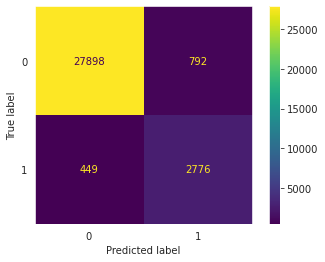
The validation accuracy and AUC is very similar to previous models but shows lower F1-score. This becomes obvious with the confusion matrix results. It predicts much more comments as toxic, while reducing the number of False-Negative. We concluds that class weights can be useful in handling False-Negatives.



ROC-AUC Plot



Confusion Matrix

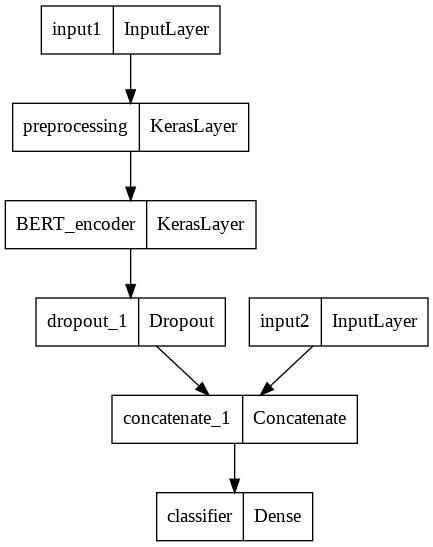


4) Small BERT (Add the number of uppercase letters as a feature)

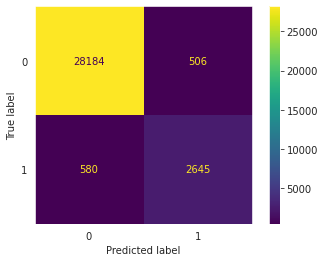
We add the number of uppercase letters in a comment as a feature to check whether it matters or not. Since BERT encoder only takes text inputs, we decide to add a side input (input2 layer) to incorporate the feature.

Unfortunately, the result does not improve significantly. Interestingly, the number of False-Positive decreased. The model seems to classify comments as toxic if they contain enough number of uppercase letters, leading to a smaller number of comments to be predicted as toxic.

Structure of Small BERT Model-4 with a side input (# of uppercase letters in a comment)



Confusion Matrix



5) Fun Try

We try multi-classification for each class (toxic, obscene, threat, insult, etc.) to meet with the goal of the competition. The only thing we change here is the number of outputs of the last dense layer.

Result:

Accuracy: 98.5 %

Finally, we add “occurrences of toxic words in comments” as a new feature. This model has a similar structure to model-4 with an additional side input. However, the result does not improve at all.

Result:

Accuracy: 96.39 %

F1-Score: 81.92 %

**Unsupervised Learning:**

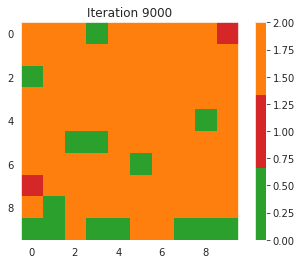
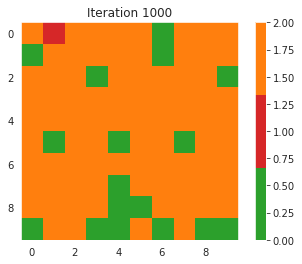
**Part 1:**

Vector Encoding

Both Term Frequency-Inverse Document Frequency and Bag of 2-gram vectorizers were used for the encodings of the K means and DB Scan approaches. For the self organized map, primarily a bag of 2-gram encoding was used. For dimensionality reduction, Truncated SVD was used for its superior performance on sparse datasets such as text.

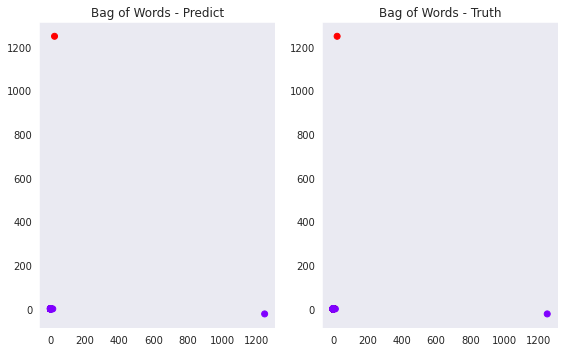
Self Organized Map

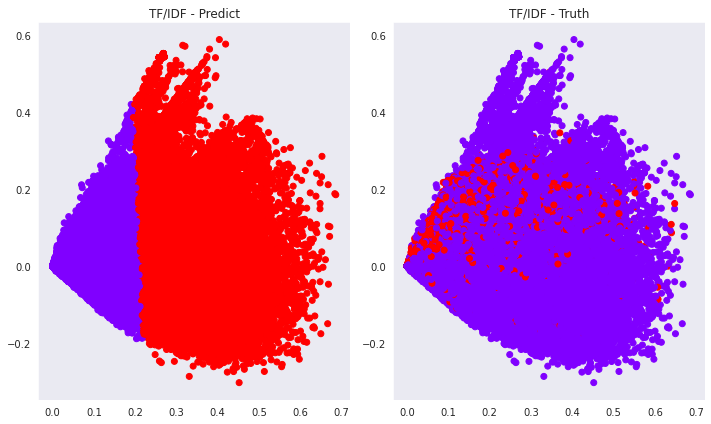
Self organizing map is an unsupervised machine learning algorithm for reducing a higher dimensional dataset to a lower dimensional feature space which can then be used for both classification and visualization. One can see how such an algorithm would lend itself to the binary classification problem of our toxic comments dataset. The data was cleaned using similar methods as discussed above ie removing capitalization, contractions etc. Next the sentences were encoded using 2-word gram vectorizer. A 10x10 grid was selected which results in 100 neurons. After 9000 iterations, there was still little convergence but more training time/compute resources would result in a better separated map.



K Means

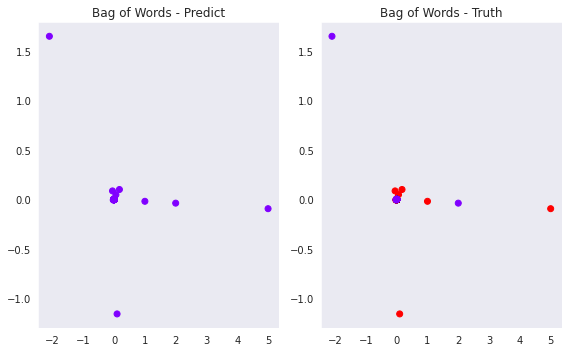
The K means algorithm was applied using both a Bag of 2-Grams vectorizer and a TF-IDF Vectorizer, the outputs are shown below

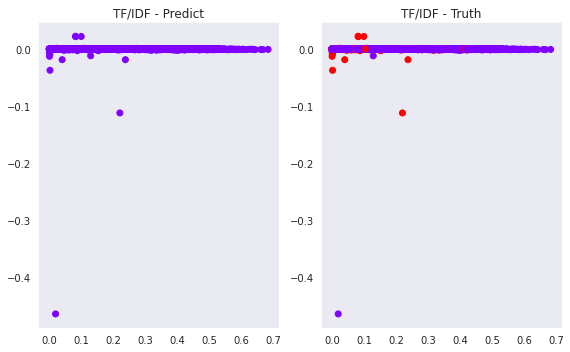




Density Based Scanning

The DB Scan algorithm was also applied to the dataset across both encodings. However, due to the space complexity ballooning, a random subset of the original dataset of about 20,000/150,000 was used. This number was chosen by slowly increasing the sample until the machine exhausted the runtime computational resources.





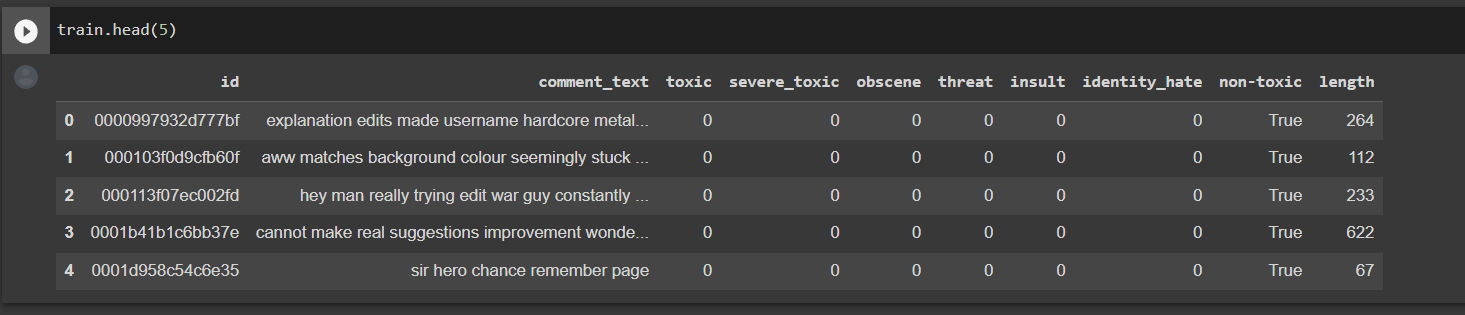
Metrics:

|  | K MEANS | | Density Based Scan | |
| --- | --- | --- | --- | --- |
| TF-IDF | Bag of 2-grams | TF-IDF | Bag of 2-grams |
| F1 Score | 8.89% | 0.01% | ~0% | ~0% |
| NMI | 1.24% | 9.3e-3% | ~0% | ~0% |
| FM | 65.3% | 90.9% | 90% | 90% |

**Unsupervised Learning (Part 2):**

Data Cleaning using Stop words:

We would not want the stop words such as [“the”, “a”, “an”, “in”] to take up space in our database, or take up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. We use stop\_words = set(stopwords.words('english')) and remove the stop words from the train[‘comment\_text’]. This ensures that the comments in the dataset are free from the not useful words. The output of the dataframe with the stopwords removed is as below



Vector Encoding using TF-IDF:

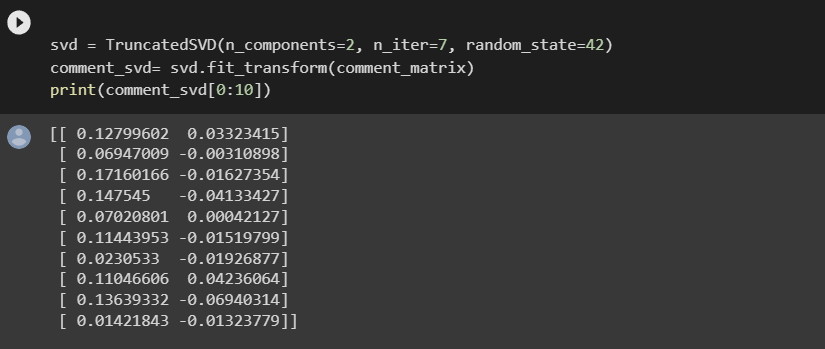
TF (Term Frequency): Term frequency is the ratio of the number of target terms in the document to the total number of terms in the document.

IDF(Inverse Document Frequency): Logarithm of the ratio of the total number of documents to the number of documents in which the target term occurs.

The TF-IDF value is obtained by TF\*IDF. We implemented this using the TfIdfVectorizer from the sklearn library. The output of the encoded text is a sparse matrix named as comment\_matrix in our implementation.

Dimensionality reduction using Truncated SVD:

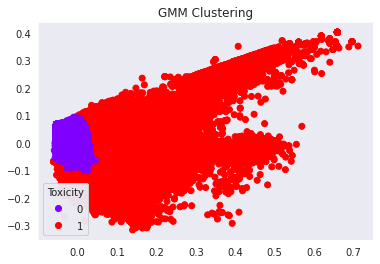
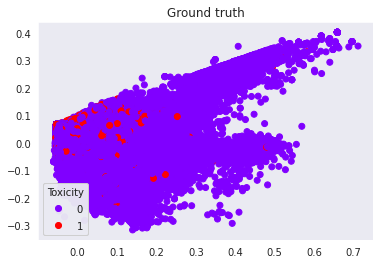
We performed TSVD using the sklearn.decomposition library with the number of components set to 2. The output of the TSVD is as shown in the figure below

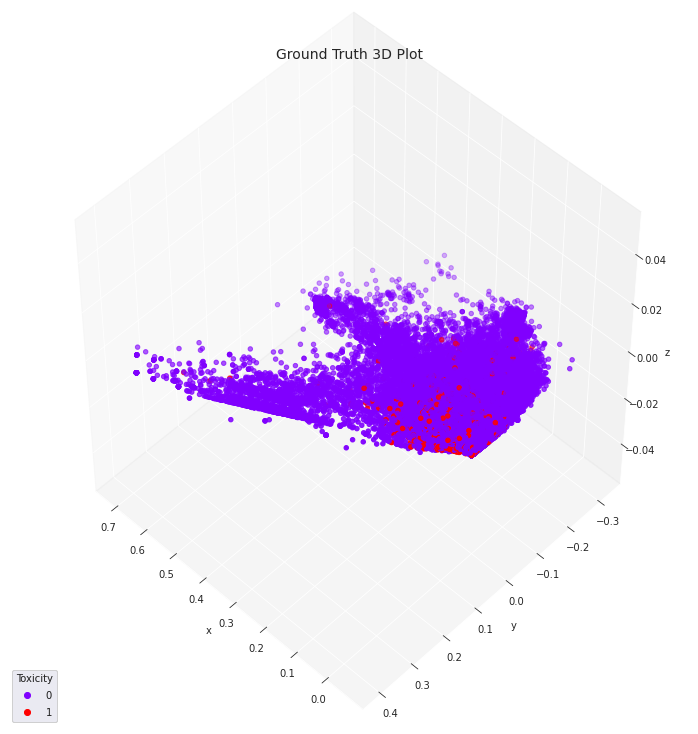
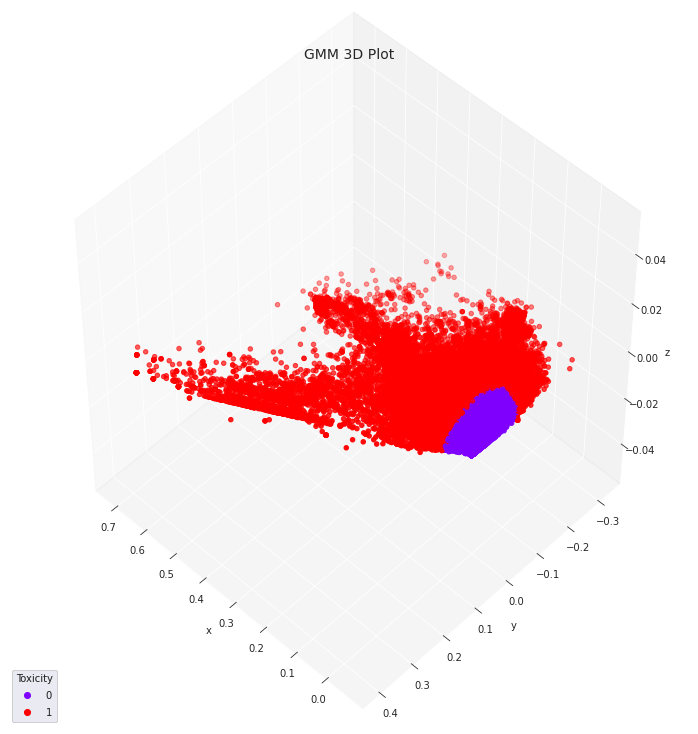


Gaussian Mixture Models:

A Gaussian mixture model is a probabilistic model in which all data points are assumed to be generated by a mixture of a finite number of Gaussian distributions with unknown parameters. Mixture models can be thought of as generalizing k-means clustering to include information about the covariance structure of the data as well as the centers of the latent Gaussian distributions.

The GMM algorithm has been applied using sklearn.mixture library and the output is as shown below



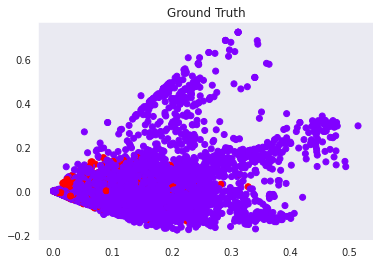
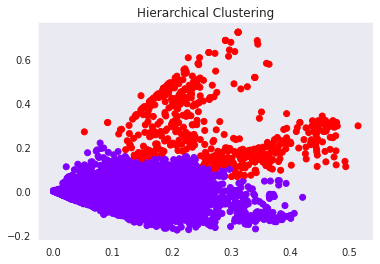
Performance Metrics:

| F1 score | 0.063 |
| --- | --- |
| Accuracy | 75% |
| FMM score | 0.756 |

Hierarchical Clustering:

Hierarchical clustering is a broad class of clustering algorithms that create nested clusters by successively merging or splitting them. This cluster hierarchy is depicted as a tree (or dendrogram). The root of the tree is the one-of-a-kind cluster that collects all of the samples, while the leaves are single-sample clusters.

We used AgglomerativeClustering from sklearn.cluster to implement this and the arguments passed were n\_clusters=2, affinity=”euclidean”. The output is as shown below



Performance Metrics:

| F1 score | 0.0015 |
| --- | --- |
| Accuracy | 86% |
| FMM score | 0.873 |

**Unsupervised Learning (with BERT)**

Why Bert Encoder?

To cluster text data, it is essential to represent each text to a vector. Among many different encoding algorithms, BERT Encoder is one of the state-of-the-art technologies. However, mapping text data to vectors with a pre-trained BERT without fine-tuning would not provide useful vector representation for toxicity clustering.

In this section, our team used the BERT encoder fine-tuned via supervised learning to get vector representations of text data and applied clustering algorithms to them. For training details, refer to supervised learning with BERT section.

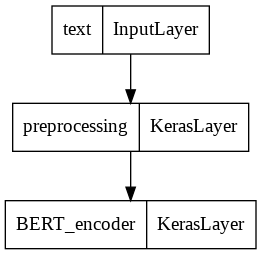


Figure: BERT Encoder fine-tuned for vector representation

Experiment Setting

Our team used vector representations of validation data to compare the results of clustering algorithms. Each vector has 512 dimensions and there is total 31915 vectors as validation data.

Vector Representation Visualization with PCA

Our team applied PCA (n\_components=2 & 3) to visualize vector representation. This is to compare the results of clustering of each algorithm.

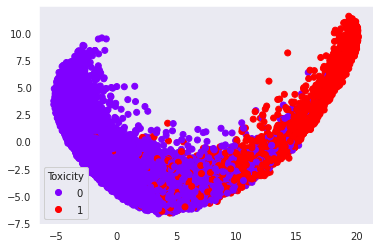


Figure: Ground Truth Label of Toxicity with PCA (2D)

K-means Clustering

Our team applied K-means (n\_clusters=2) to the vector representation. As expected, K-means algorithm divided the dataset into two parts, making a large number of wrong decisions along the decision border. One thing to note was that K-means algorithm was vulnerable to False-Positive in this experiment.

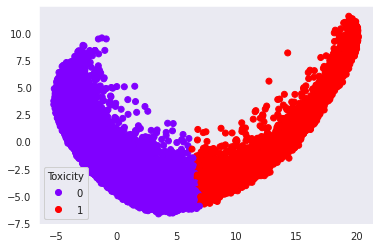


Figure: K-means Clustering of Toxicity (2D)

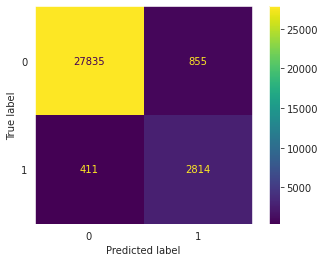


Figure: K-means Confusion Matrix

Gaussian Mixture Model (GMM)

Our team applied GMM (n\_clusters=2) to the vector representation. GMM produced an even larger number of False-Positive predictions than K-means. This was because the model predicted too many data points as toxic.

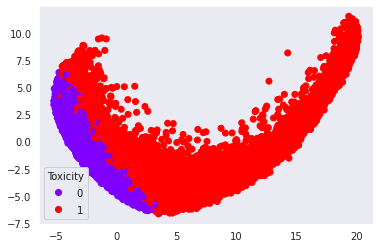


Figure: GMM Clustering of Toxicity (2D)

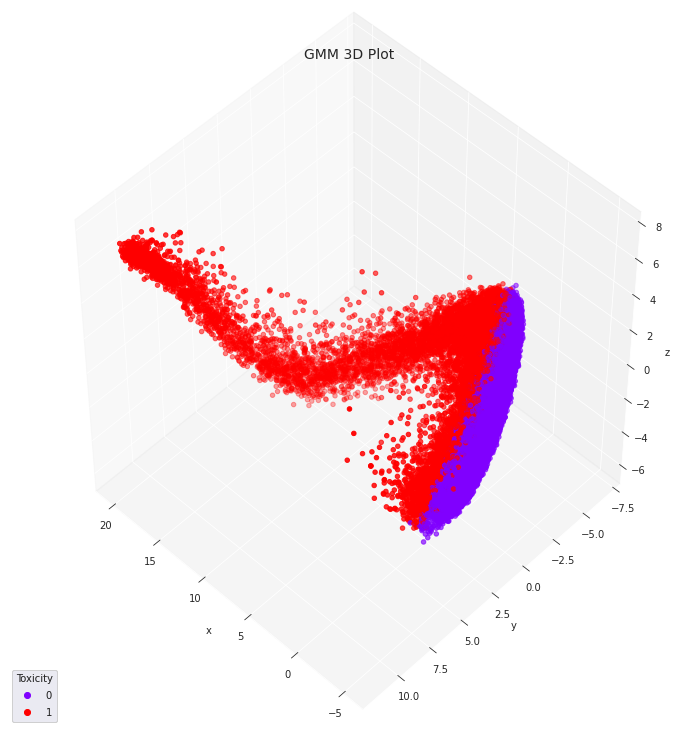


Figure: GMM Clustering of Toxicity (3D)

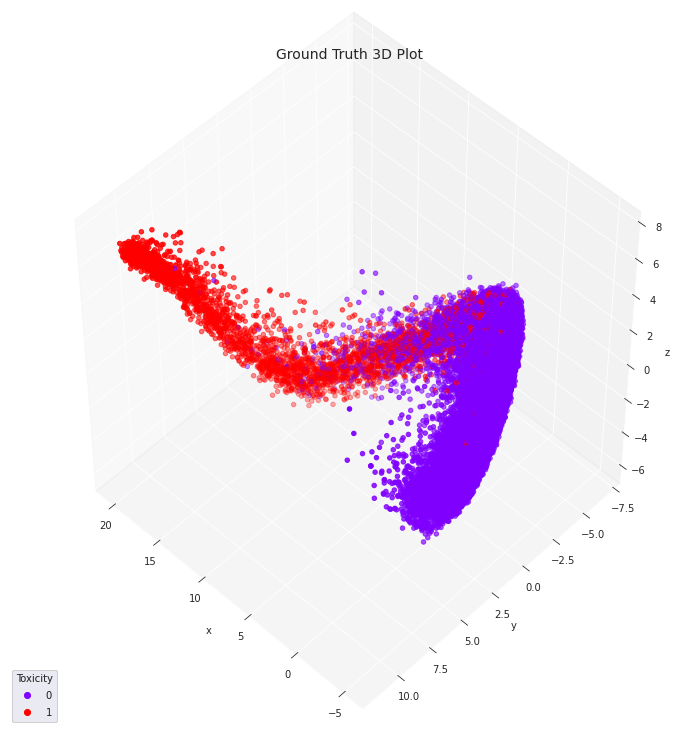


Figure: Ground Truth Label of Toxicity with PCA (3D)

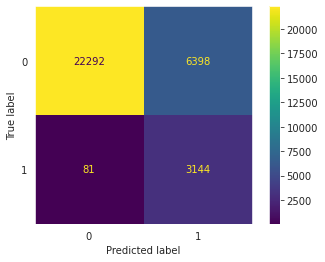


Figure: GMM Confusion Matrix

DBSCAN

Our team applied DBSCAN to the vector representation. We tuned the hyper-parameters (eps, min-samples) to get exactly two clusters except for noise cluster (-1). One big disadvantage of DBSCAN is that a certain portion of data is classified as noise. About 27.95% of the data were actually considered as noise in the model. However, when only considering clustered data points, DBSCAN showed the best results.

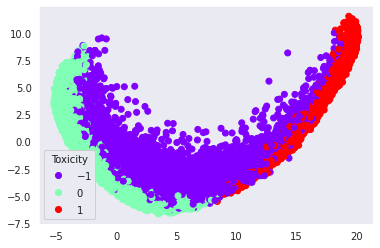


Figure: DBSCAN Clustering of Toxicity (2D) with noise

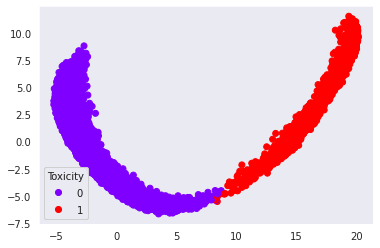


Figure: DBSCAN Clustering of Toxicity (2D) except noise

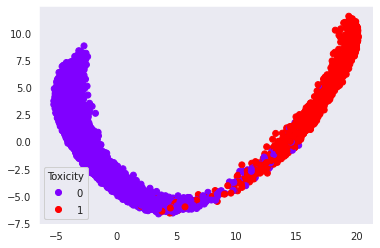


Figure: Ground Truth of Toxicity (2D) except noise

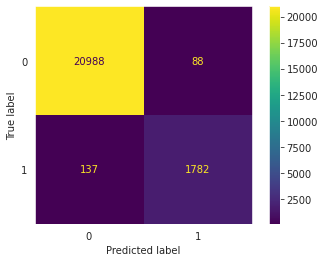


Figure: DBSCAN Confusion Matrix

Clustering Evaluation

Our team compared clustering results with popular clustering evaluation metrics such as F1-score, Normalized Mutual Information, Fowlkes-Mallows Measure, Silhouette Score, and Davies Boulding Score. DBSCAN without noise data points showed the best performance followed by K-means and GMM.

테이블이(가) 표시된 사진

자동 생성된 설명