**Data Science Project Protocol**

***Toxic Comments Classification***

***Using ML and DL Models for NLP***

*Author: Leah Awawdeh*

January, 2023

# Introduction



With the rize of social media platforms, many people turned to online communication. This was further amplified in 2020, when schools and offices across the globe closed in response to the spread of COVID-19. As a result, even kids had to drastically increase their daily screen time to at least [four hours a day](https://morningconsult.com/2020/08/20/youtube-netflix-and-gaming-a-look-at-what-kids-are-doing-with-their-increased-screen-time/) inc. studying and connecting with teachers and peers.

However, online interactive communication hides many hazards such as fake news, online harassment and toxicity. In fact, nearly half (46%) of teens age 13-17 have been bullied or harassed online. ([Pew Research](https://www.pewresearch.org/internet/2022/12/15/teens-and-cyberbullying-2022/), December 15, 2022). As described by Dr. Adam Pletter, clinical psychologist, this form of “cyberharassment is both public and permanent” which amplifies the upset. Dr. Pletter adds that cyberbullying also leaves a digital tattoo (a more fitting term than digital footprint), resulting in years of upset and often direct mental health implications.

One of the most common forms of cyberharassment is called ‘Toxic Comments’, and it is not limited to verbal violence, but also includes comments that are rude, hateful, and disrespectful. Although there are efforts to enhance the safety of online environments based on crowdsourcing voting schemes or the capacity to denounce a comment, in most cases these techniques are inefficient and fail to predict a potential toxicity. Thus, automatic toxic comment identification and prediction in real time is of paramount importance, since it would allow the prevention of several adverse effects for internet users.

And even though all major companies, including Google, Facebook, and Tweeter are already using Artificial Intelligence to automatically detect, flag, or remove such unhealthy contents from their platforms, other chat-services and applications are still in need of efficient classifiers being developed and incorporated into their services.

In this project, I tried to create and assess the efficiency of binary toxic comments classifiers that are based on different Machine Learning and Deep Learning models.

# Methodology (Project design)

## Data

For training and evaluating the classifier, I have used datasets provided on Kaggle website:

* <https://www.kaggle.com/datasets/surekharamireddy/malignant-comment-classification?select=train.csv>
* <https://www.kaggle.com/datasets/saurabhshahane/cyberbullying-dataset?select=kaggle_parsed_dataset.csv>

The datasets are comprised of the following subsets:

* train.csv
* test.csv
* aggression\_parsed\_dataset.csv
* attack\_parsed\_dataset.csv
* kaggle\_parsed\_dataset.csv
* toxicity\_parsed\_dataset.csv
* twitter\_parsed\_dataset.csv
* twitter\_racism\_parsed\_dataset.csv
* twitter\_sexism\_parsed\_dataset.csv
* youtube\_parsed\_dataset.csv

Each of the above files contains data scraped from various websites including Twitter, Wikipedia, YouTube, and more. Thus, each has a column with text, either one binary label column with 0/1 label for toxicity, or multiple labels columns to denote the toxicity subtype by 0/1. Some other metadata, such as date and userID is included for some records, but not the other. It was not used because of inconsistency. The Subsets were imported as tables and then merged to a single flat file on the basis of text column, label, unique comment count, and origin.

The label was used as target variable for binary classification.

The flat file then was imported into Jupyter Notebook (CommentsClassification\_EDA) as Pandas dataframe with a total of 379181 rows and 11 columns. Only 4 rows had missing data and were deleted. Comments were considered outliers and were deleted as well in case there were no meaningful words included (e.g. ‘Err#’ computer error comments). My data enrichment strategy was allowing a bigger vocabulary, which on the other hand was constrained by memory related issues (and lack of memory errors). This constraint was the main bottleneck which affected the performance of models. The metadata that was left for data enrichment purposes were not used in the final models as it is impossible to get the same kind of metadata in the production environment in the first place; and also DL models and pipelines, which were the most effective, do not allow such metadata.

The other possible source of bias in the data is the presence of computer bots that generate certain comments lots of times. This may impact the words and N-grams frequencies and bias models’ learning process. Thus each comment was allowed only once.

The text was cleaned, by conversion to lowercase, and removing emails, phone numbers, emojis, stopwords, punctuations, extra spacing etc. The text was lemmatized, as stemming was found too aggressive. At the end of EDA I presented the most common offensive words as charts and Word Cloud (see the CommentsClassification\_EDA notebook).

## Models

As the problem is a classic binary classification problem, the following approaches were used:

The work was done in two directions, ML(CommentsClassification\_modelsML\_finetuning, CommentsClassification\_modelsML\_Undersampled notebooks) and Deep Learning (CommentsClassification\_modelsNN notebook). For Deep Learning I used both, custom and pretrained NN models, while for ML I used plain models and pipelines.

F1 score was mainly used as an evaluation metric for models performance. For it is measuring the relationship between recall and precision, and thus concerned with percentage of minority class classification vs misclassification.

I’ve also checked the balancing of the dataset. The labels ratio initially was 88:12% (0:1).

Tomek links didn’t give the desired effect, so I’ve tried Random Undersampling and taking most common comments (with count >2) and labels ratio of 75:25% (which is good for NLP). kNN based undersampling failed due to memory issues. I didn’t get better results for a balanced dataset, and vice versa, the results were less favorable probably because of data loss (specifically vocabulary). The entire dataset was used for NN based models and for ML pipelines. Undersampled datasets were used for ML models due to memory issues. Vocabulary selected as features for ML models was restricted to 20K also due to memory shortage errors. While 200K vocabulary was used for NN based models.

The data was divided using sklearn train\_test\_split with ratio 0.7-0.3 to train-test. In the case of NN, the split was .7 - .2 - .1 for train-val-test respectively.

No ensemble was built, But I ran prebuilt ensembles XGBoost, Random Forest, using bootstrapping and cross-validation.

## Tfidf vectorizer was used, tried also simple CountVectorizer, basic NLTK tokenizer, and model-specific tokenizers, like DistilBertTokenizer. I’ve tried Gensim tokenizer, but no difference was found in performance. Thus, sticked with tokenizers/vectorizers as described for each model.

## Model Deployment

The model is ready to be deployed on any platform that has a social/communication component, including chats, reviews, blogs, social media and more. The model would get user input and assess it to predict whether it is appropriate to be published. If not, the comment may be marked or completely removed by the platform. Final user, however, has to be given an appeal option, where he can explain why his comment was not offensive. The appeal has to be checked manually and inserted into the ‘appeal’ database for future use.

In case the model fails to predict and returns null, the prediction will be considered as non-toxic comment.

The QA of the project would consist of two parts. First, the models would be checked to misclassifications using test division of data.The model could be deployed when misclassifications are minimized. The second part of QA would be performed during the production phase using manual check of user feedback in cases of misclassifications. The corrected user data should be stored in the database and fed to the model for finetuning once every quarter.

The following table summarized the models that were used for prediction of comment toxicity (F1 score used for evaluation as explained above):

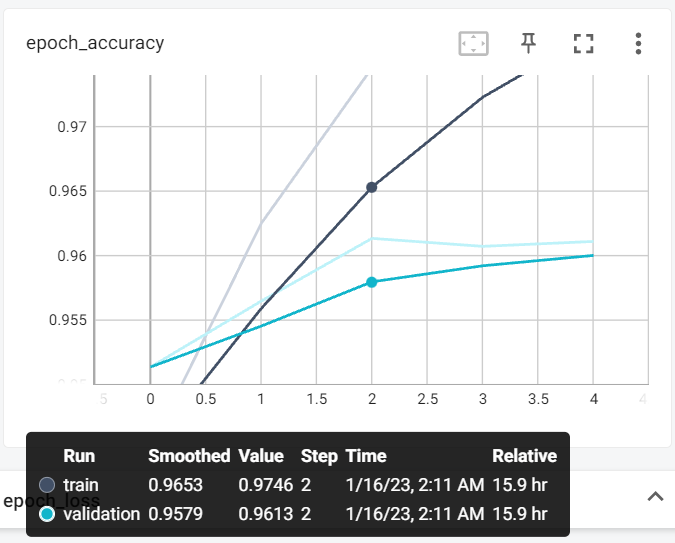
| ***Type*** | ***Model*** | ***Hyperparameters changed from defaults*** | ***Dataset*** | ***Vocabulary*** | ***F1 Score*** |
| --- | --- | --- | --- | --- | --- |
| ML - ensembler | RandomForestClassifier | n\_estimators=200 | Undersampled | 20K | 0.73 |
| ML - ensembler | LGBMClassifier | 'learning\_rate': 0.06, 'n\_estimators': 1500, 'colsample\_bytree': 0.5 | Undersampled | 20K | 0.80 |
| ML - ensembler | RandomForestClassifier | n\_estimators=200, criterion='entropy', max\_depth=5, random\_state=123, warm\_start=True) | Full | 20K | 0.74 |
| ML - ensembler | RandomForestClassifier | n\_estimators=3000 | Full | 20K | 0.80 |
| ML - ensembler | LGBMClassifier | 'learning\_rate': 0.5 'n\_estimators': 1500, 'colsample\_bytree': 0.5,  'max\_depth' : 7,  boosting\_type='goss' | Full | 20K | 0.80 |
| ML - pipeline | Pipeline with Naive Bayes Classifier | TfidfVectorizer  ComplementNB | Full | Not limited | 0.92 |
| ML - pipeline | Pipe with Logistic regression | TfidfVectorizer  LogisticRegression | Full | Not limited | 0.95 |
| ML - pipeline | Pipe with Logistic regression – grid search of best params | params\_tf = {'max\_features': 3000,  'ngram\_range': (1, 2),  'use\_idf': False}  params\_lr = {'penalty': None,  'random\_state': 123,  'verbose': 2,  'warm\_start': True} | Full | Not limited | 0.93 |
| DL - custom | Sequential with LSTM unit | max\_length = 1800  trunc\_type='post'  padding\_type='post'  oov\_tok = "<OOV>"  embedding\_dim = 32, vectorizer TextVectorization | Full | 200K | 0.79 |
| DL - custom | sequential with distilBert incorporated  (Google) | special\_tokens = True,pad\_to\_max\_length = True,truncation=True | Full | Full | 0.96  (0.83 for toxic!) |
| DL - prebuilt | fastText (Facebook) | wordNgrams = 3 | Full | Full | 0.9572 |
| DL - prebuilt | fastText (Facebook) | wordNgrams = 2 | Full | Full | 0.9569 |
| DL - prebuilt | fastText (Facebook) | autotune for 10 min | Full | Full | 0.9565 |

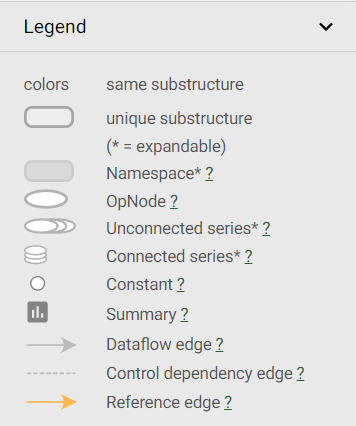
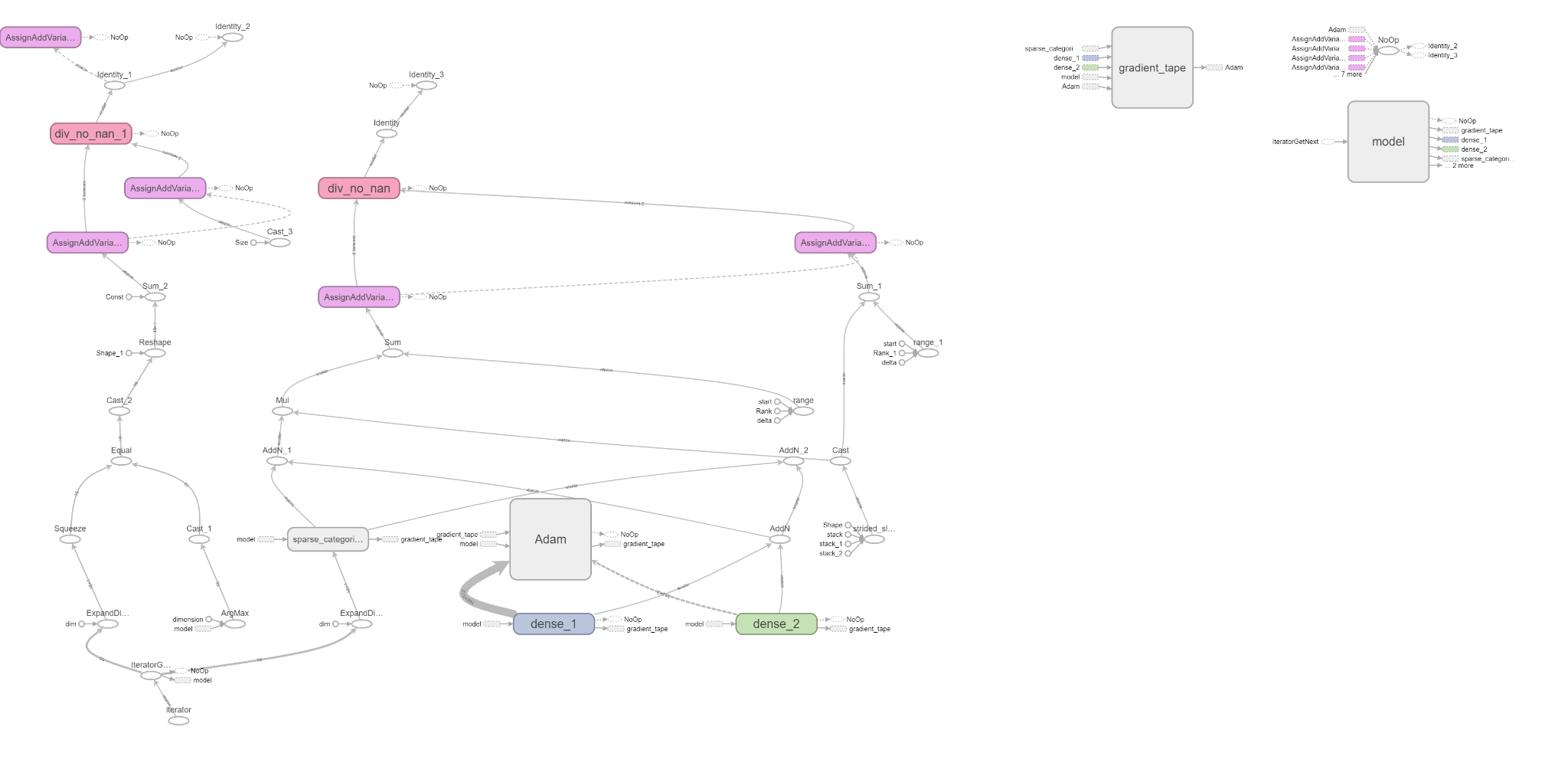
# Results

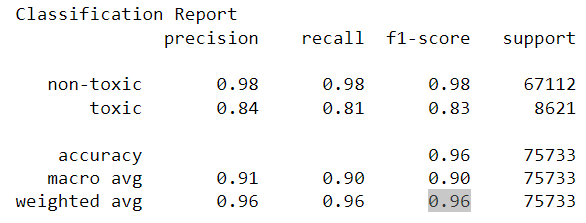
As seen from the table above, with ML approach I could get a maximum F1 score of 0.95 using pipeline approach with TfidfVectorizer and plain Logistic regression with default parameters both. As for deep learning models, the prebuilt models allow the best result of F1 score 0.96. Both, fastText and distilBert give similar results which are drastically better than plain custom models. Prebuilt models are fast and easy to run either.

Several problems, if solved, would allow even better results. First, and main problem is memory limitations. Second point is the problems with vocabulary, when people use non-English letters to encode offensive words. Or purposefully misspell offensive words. This may be solved with either increasing vocabulary size, or implementing FuzzyWuzzy logic to catch and ‘correct’ such misspelled words.

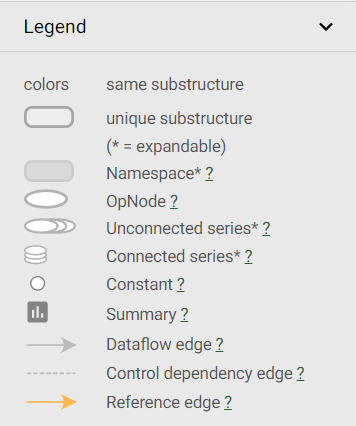
The best model in terms of performance is customized distilBert. The following are visualizations of its performance, including embeddings and loss/accuracy graphs:



Classification report:

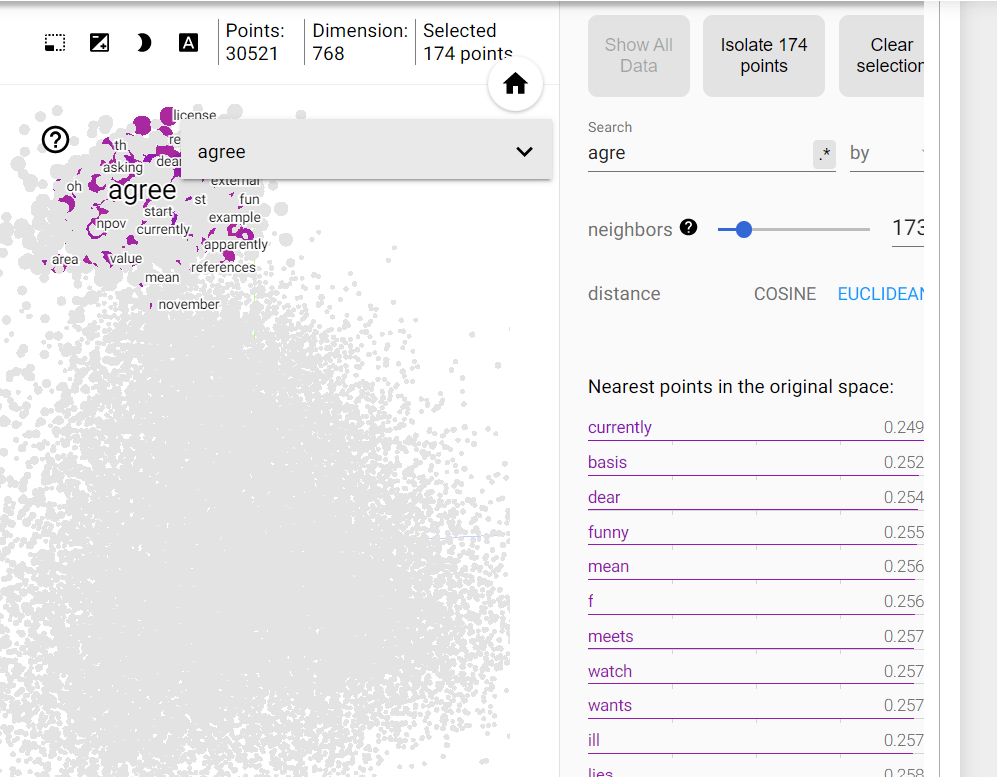
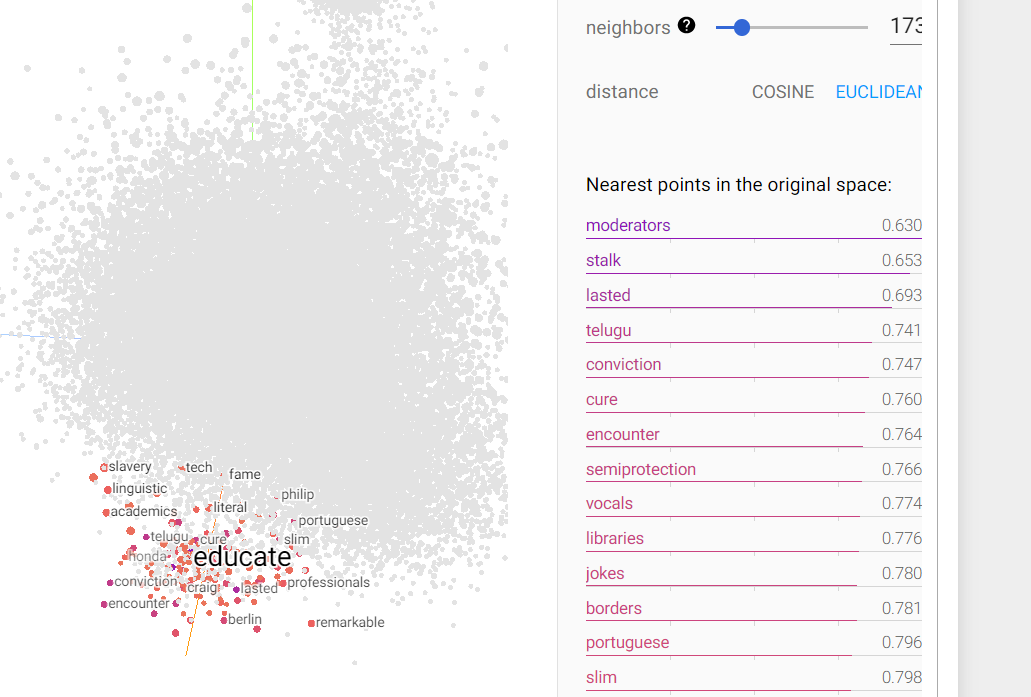


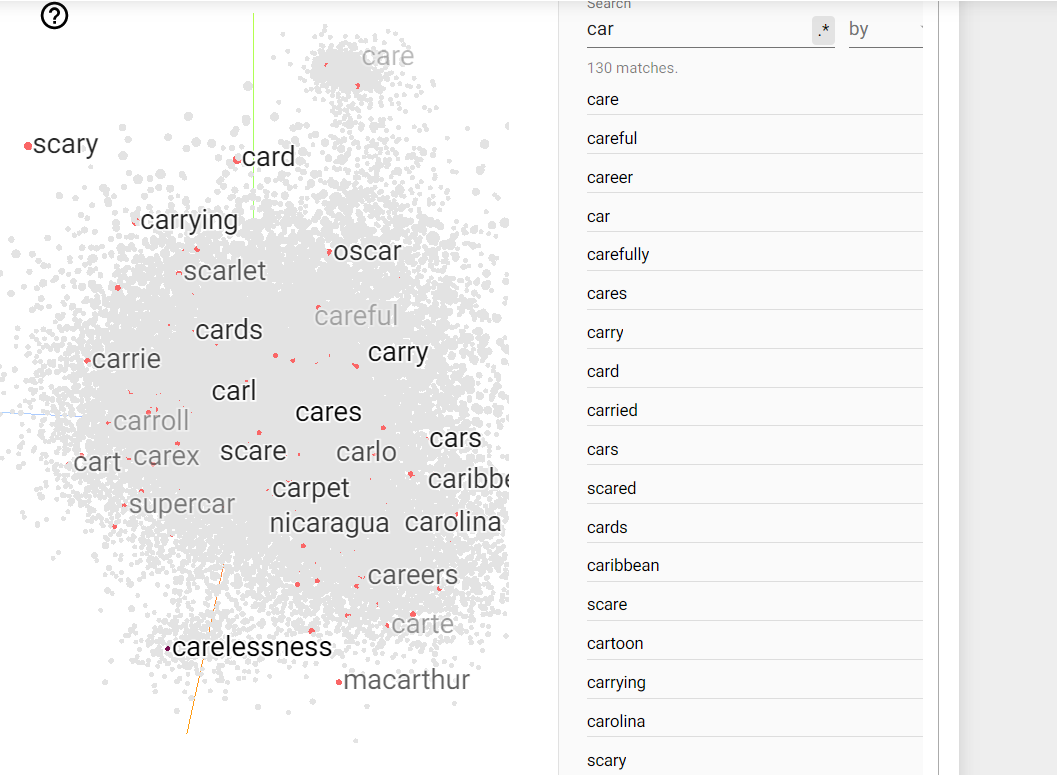
The following is model algorithmic map representation:

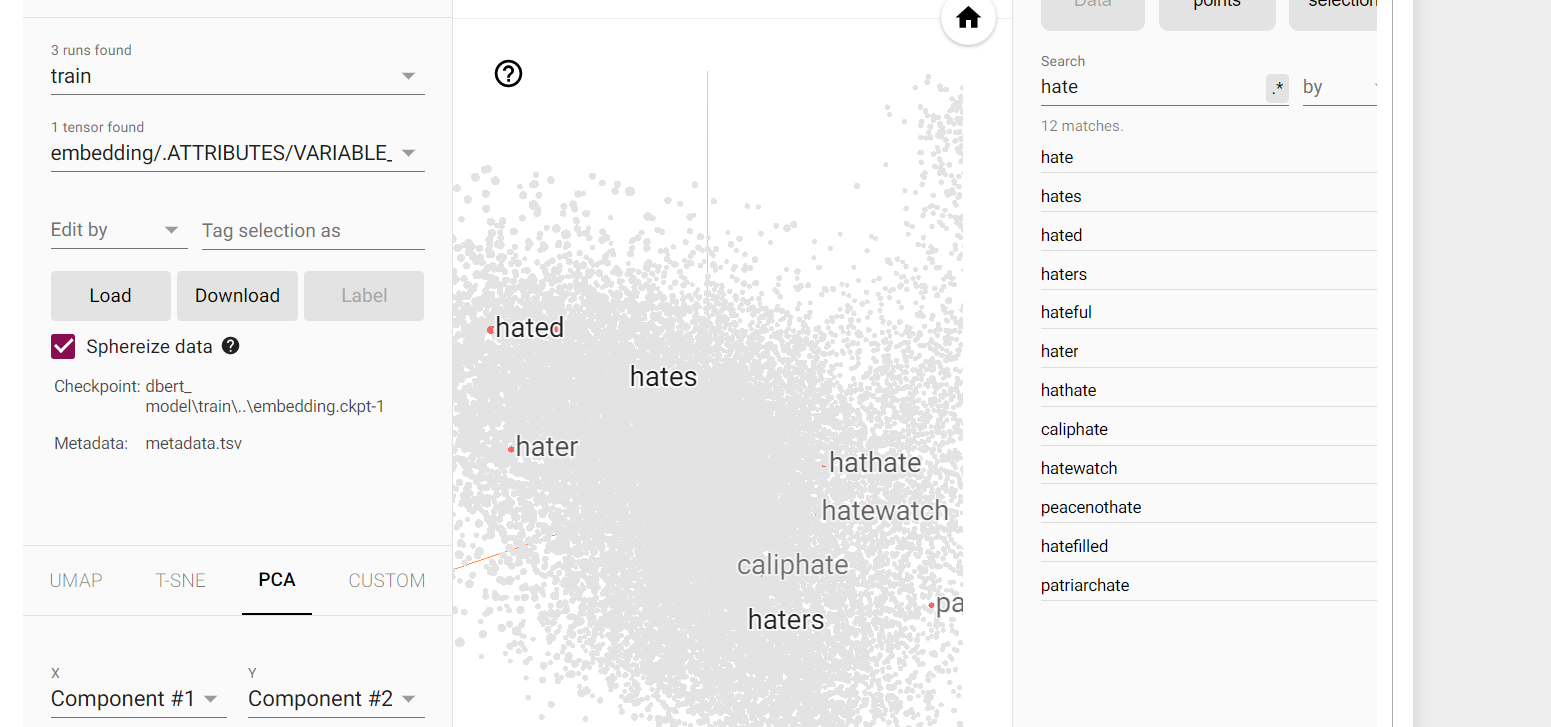


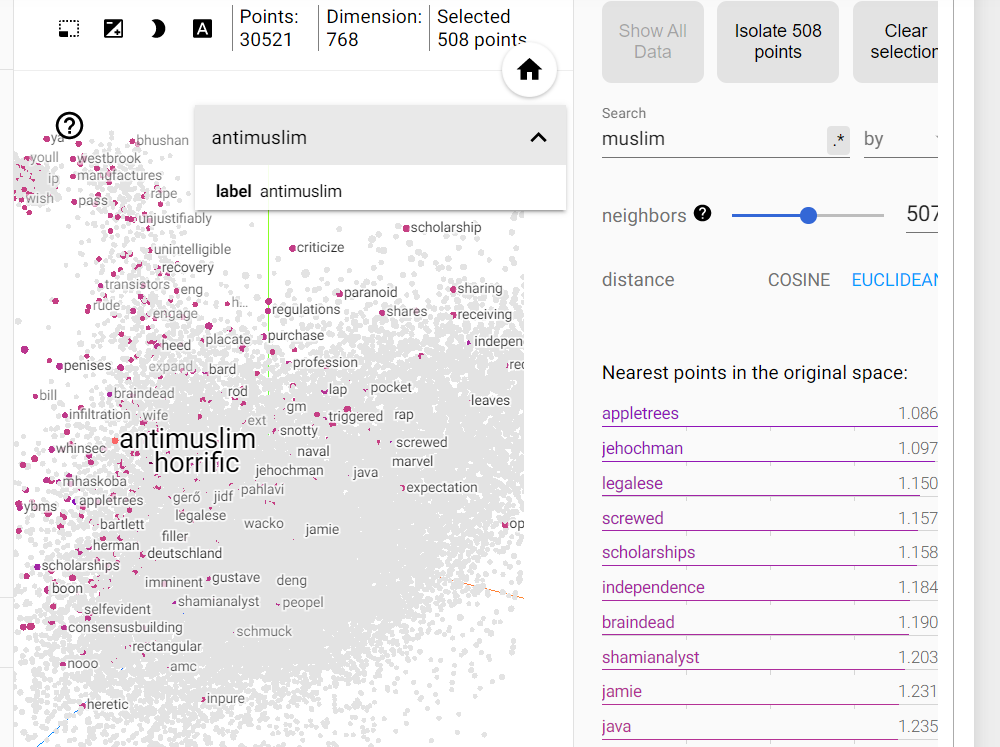
Finally, embedding representation:

* Positive:



* Negative:
* This specific snapshot shows fastText embeddings. The words are vectorized according to its component characters rather than meaning. Thus, such words as care, scare, career and even carpet are in the same hemisphere, very close to each other. Vice versa, glove embeddings are by meaning, as well as Gensim and NLTK.

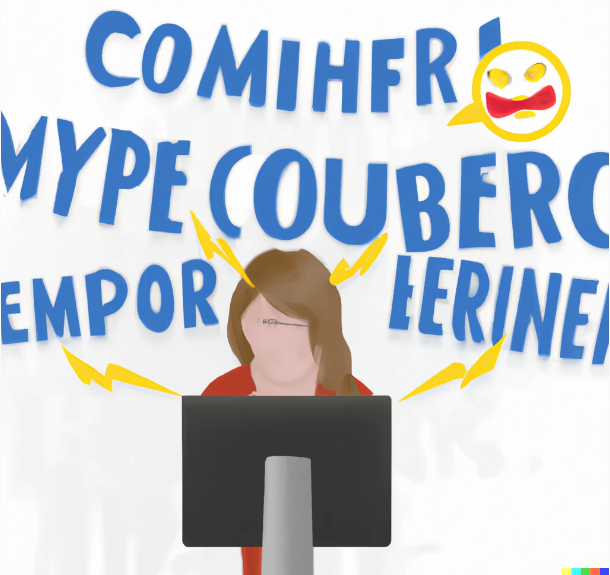




Also worth mentioning is that even though the prebuilt models gave similar results, the models are very different in terms of architecture, embeddings, and overall algorithm. To summarize, FastText is a more efficient and simple approach that is well-suited for large-scale text classification tasks, while BERT (Bidirectional Encoder Representations from Transformers) is a more complex model that is well-suited for natural language processing tasks such as question answering and sentiment analysis. FastText is based on the bag of words model while BERT is based on transformer architecture (stack of encoders) and learns the contextual relationship between words.

# Conclusion

The proliferation of toxic comments on online platforms has become a major concern in recent years, as they can lead to harm and negative consequences for individuals and communities.The report proposed and evaluated machine learning and deep learning based approaches for toxic comments classification, with the goal of improving the efficiency and accuracy of the moderation process. Machine learning and deep learning techniques have the potential to be a powerful tool for automating the process of identifying and moderating toxic comments on online platforms. However, the models still have room for improvement, and manual moderation is needed in cases people want to submit appeals.



DALL-E 2 generated