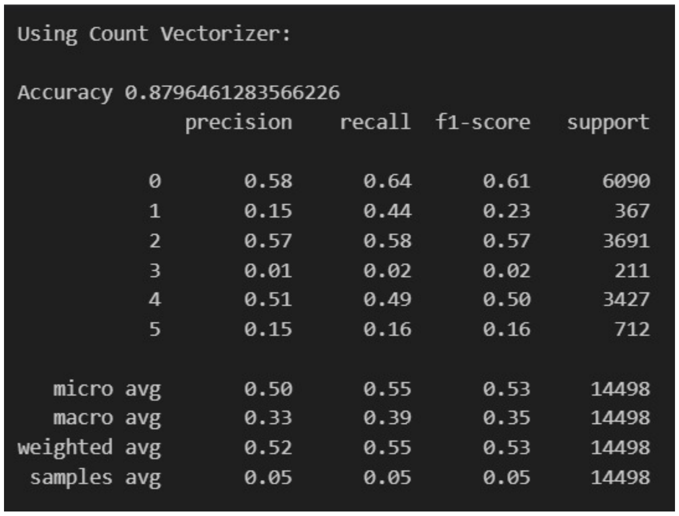
Report on Content Moderation and Toxicity Classification

Naive Bayes

- **Introduction**
    - Our recent project involved the use of a Naive Bayes model for text classification. The goal was to enable a computer to automatically categorize text into different labels.
- **Data Preparation**
    - We began by assembling our test dataset, consisting of various texts and their associated labels. It was crucial to exclude any texts that lacked labels to ensure the accuracy of our training process.
- **Data Preparation and Cleaning**
    - We start by cleaning our textual data. This included removing common words like "the" and "and"(stopwords), as well as any punctuation and numbers. This process was essential to highlight the most significant words that would aid in accurate text classification.
- **Model Setup and Training**
    - To make the inputs suitable for the model, we used two types of vectorizers, CountVectorizer and TF-IDF Vectorizer.
        - CountVectorizer tracks the frequency of each word's occurrence within the text.
        - TF-IDF Vectorizer counts how often each word shows up in the text and it figures out which words are really important. For example, common words like "is" or "of" might show up a lot, but they don't always tell us much about what the text is really about. The TF-IDF Vectorizer helps us focus on the words that are less common but more meaningful.
    - The vectorized inputs were passed through MultiOutputClassifier that encompasses a MultinomialNB classifier. The combination is particularly effective in handling cases where a single text might be associated with multiple labels.
    - The core of our model was the Naive Bayes classifier. This classifier operates on the assumption that each word in the text contributes independently to the probability of the text belonging to a particular label. Despite this being a simplification of actual language dynamics, it surprisingly leads to reliable classification results on the test data:

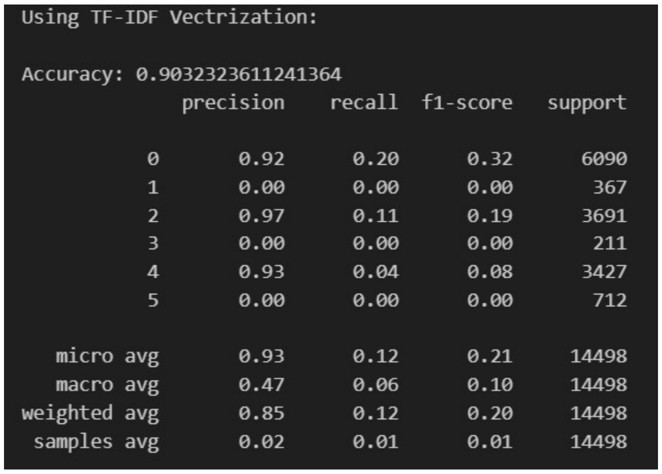
- **Using Countvectorizer:**



```
Using Count Vectorizer:

Accuracy 0.8796461283566226
              precision    recall  f1-score   support

           0       0.58      0.64      0.61      6090
           1       0.15      0.44      0.23       367
           2       0.57      0.58      0.57      3691
           3       0.01      0.02      0.02       211
           4       0.51      0.49      0.50      3427
           5       0.15      0.16      0.16       712

   micro avg       0.50      0.55      0.53     14498
   macro avg       0.33      0.39      0.35     14498
weighted avg       0.52      0.55      0.53     14498
 samples avg       0.05      0.05      0.05     14498
```

  - The model's accuracy is pretty good (around 88%), which means it often correctly decides if a label should be given to something. But this number doesn't tell us everything. When we have many different labels to predict, the overall accuracy might not show how the model does on each specific label, especially if some labels are much more common than others.
  - Labels 0, 2, and 4 show moderate precision, recall, and F1-scores, suggesting reasonable model performance for these categories. Labels 1, 3, and 5 exhibit low precision and recall, resulting in low F1-scores. This indicates the model struggles to accurately predict these labels, either missing them (false negatives) or incorrectly assigning them (false positives).

- **Using TF-IDF Vectorization:**



```
Using TF-IDF Vectrization:

Accuracy: 0.9032323611241364
              precision    recall  f1-score   support

           0       0.92      0.20      0.32      6090
           1       0.00      0.00      0.00       367
           2       0.97      0.11      0.19      3691
           3       0.00      0.00      0.00       211
           4       0.93      0.04      0.08      3427
           5       0.00      0.00      0.00       712

   micro avg       0.93      0.12      0.21     14498
   macro avg       0.47      0.06      0.10     14498
weighted avg       0.85      0.12      0.20     14498
 samples avg       0.02      0.01      0.01     14498
```

- The model is good at deciding if a label fits a sample or not, about 90.32% of the time. But, this number doesn't tell us everything, especially when the dataset is not balanced and there are many types of labels. The precision, recall, and F1-scores for individual labels are quite low. For labels 1, 3, and 5, these scores are 0, indicating the model is not correctly identifying these labels at all. For labels 0, 2, and 4, while precision is relatively high, recall and F1-scores are very low. This suggests the model is very conservative, only predicting these labels in very clear cases, leading to many false negatives.

- The two ways of sorting words in text (TF-IDF and Count Vectorization) work differently, which leads to different results. TF-IDF focuses more on rare words, which might cause it to miss out on important common words. This could be why it doesn't do as well in correctly identifying all classes. Count Vectorization, however, seems better at handling different types of classes fairly, showing more balanced results. While the TF-IDF method is generally more accurate overall, it doesn't do as well when we look at how it identifies each specific class, especially in recognizing all the examples of a class.

Sequence Models (LSTM)

- **Introduction**
  - The LSTM (Long Short-Term Memory) model is a recurrent neural network architecture designed for sequential data processing. It is chosen for toxicity classification as it can capture sequential dependencies in text data better than a regular RNN, which is crucial for understanding the context of words in sentences.
- **Tokenizing**
  - In the case of LSTM, tokenization is not explicitly performed using pre-trained tokenizers like in BERT. Instead, the text is preprocessed using standard natural language processing techniques:
    - **Lowercasing**: Text is converted to lowercase for uniformity.
    - **Removing Punctuation, Stopwords, Numbers, and URLs**: These steps remove irrelevant information and focus on the semantic content of the text.
    - **Filtering Short Sentences**: Excluding short sentences enhances the model's ability to learn from meaningful and context-rich data.
- **Training and Evaluating**
  - The LSTM model is trained on the preprocessed text data using the following configuration:
    - **Embedding Size of 100**. The embedding size determines the dimensionality of the word embeddings learned by the model. A size of 100 is chosen to strike a

        balance between capturing rich semantic information and managing computational resources.

        - **LSTM Layer**: Contains 50 units to capture sequential dependencies in the data. The choice of 50 units determines the capacity of the LSTM layer. It's a balance between capturing contextual information and managing model complexity.

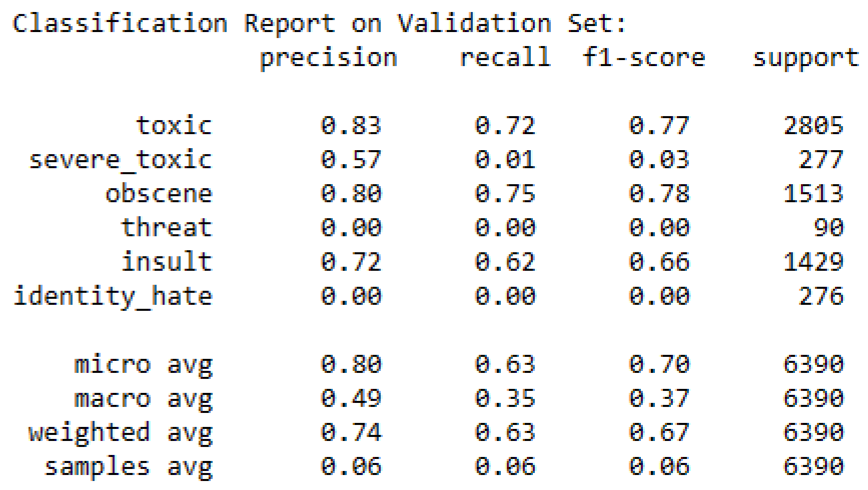
        - **Dense Layer**: Output layer with a sigmoid activation function for binary classification of toxicity types. The Dense layer with 6 units corresponds to the six toxicity types. Sigmoid activation is used for binary classification tasks, as it squashes output values to the range [0, 1], allowing the model to independently predict the presence or absence of each toxicity type.

        - **Compilation**: Adam optimizer, binary cross-entropy loss, and accuracy metric. The Adam optimizer is chosen for its efficiency in adjusting learning rates during training. Binary cross-entropy loss is appropriate for binary classification tasks. Accuracy is a suitable metric for evaluating model performance in this binary classification context.

    - The sequential model is suitable for a straightforward stack of layers, where each layer has exactly one input tensor and one output tensor. This simplicity is well-suited for the task of toxicity classification.

    - The model is trained for 4 epochs with a batch size of 64.

- **Validation Results**



```
Classification Report on Validation Set:
                precision    recall  f1-score   support

        toxic       0.83      0.72      0.77      2805
 severe_toxic       0.57      0.01      0.03       277
      obscene       0.80      0.75      0.78      1513
       threat       0.00      0.00      0.00        90
       insult       0.72      0.62      0.66      1429
identity_hate       0.00      0.00      0.00       276

    micro avg       0.80      0.63      0.70      6390
    macro avg       0.49      0.35      0.37      6390
 weighted avg       0.74      0.63      0.67      6390
  samples avg       0.06      0.06      0.06      6390
```

- **Evaluation**

  - The model achieves a validation loss of 0.05 and a validation accuracy of 99%. On the test dataset, it achieves an accuracy of 95%.
  - Results were similar to BERT.

BERT

- **Introduction**
  - BERT (Bidirectional Encoder Representations from Transformers) is a decoder-only transformer-based model designed for natural language understanding. Since we need a model that understands the text/comment we provide it and assigns it a toxicity label, BERT would be a good choice as it can understand the context of words (in both left and right direction) in a sentence by pre-training a deep bidirectional representation of text.

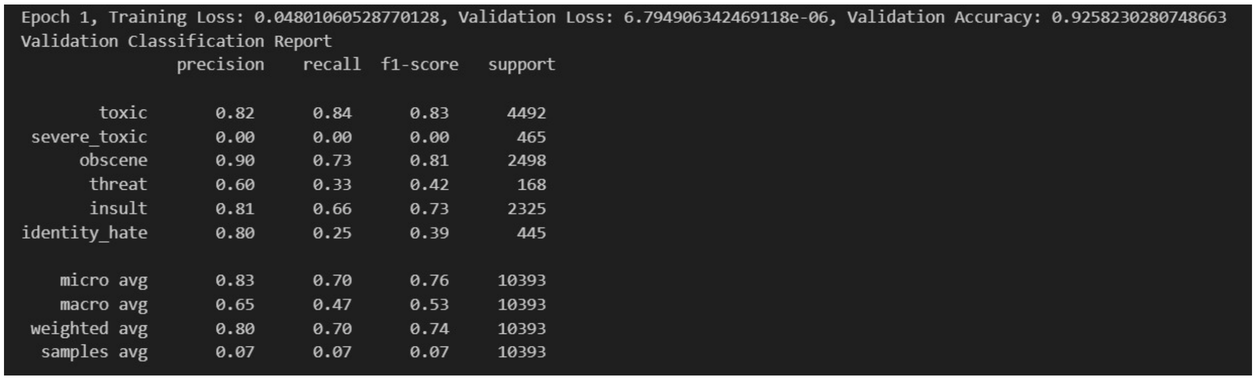
- **Tokenizing**
  - The inputs are the comments (text based and variable sized); hence we use the BertTokenizer from the transformers modules provided by HuggingFace to convert the data into something our model can learn from.
  - Specifically, we use the BertTokenizer from the bert-base-uncased model, which means it tokenizes the sequence specifically for a pre-trained BERT model with lowercase tokens. We encode all the comments by converting them into token IDs. Token IDs are unique numerical representations assigned to each token in a text sequence, enabling a model like BERT to understand and process the input by mapping tokens to corresponding embeddings and capturing positional information. They encode the text into a format the model comprehends, ensuring accurate interpretation and learning from the sequence during training. It uses special tokens like [CLS], [SEP], [PAD], and [MASK] tokens that enable the BERT model to train using some specific methods.

- **Training and Evaluating**
  - BERT utilizes two major training methods during its pre-training phase: Masked Language Model (MLM) and Next Sentence Prediction (NSP).
    1. Masked Language Model (MLM):
       - As mentioned above, BERT masks some of the words in the input sentence with a [MASK] token. The model tries to predict some randomly masked tokens by using the surrounding context (both on left and right side) as clues. This training method allows BERT to learn the (English) language and proper word usage in a sentence.
    2. Next Sentence Prediction (NSP):

    - NSP is used to help BERT learn about relationships between sentences by predicting if a given sentence follows the previous sentence or not. BERT is trained on a task where it predicts whether a pair of sentences are in the original order or if they are randomly swapped. The model uses [CLS] and [SEP] to pick out random sentences/segments for this method of training. This training method allows BERT to comprehend how sentences relate to each other in a larger context.

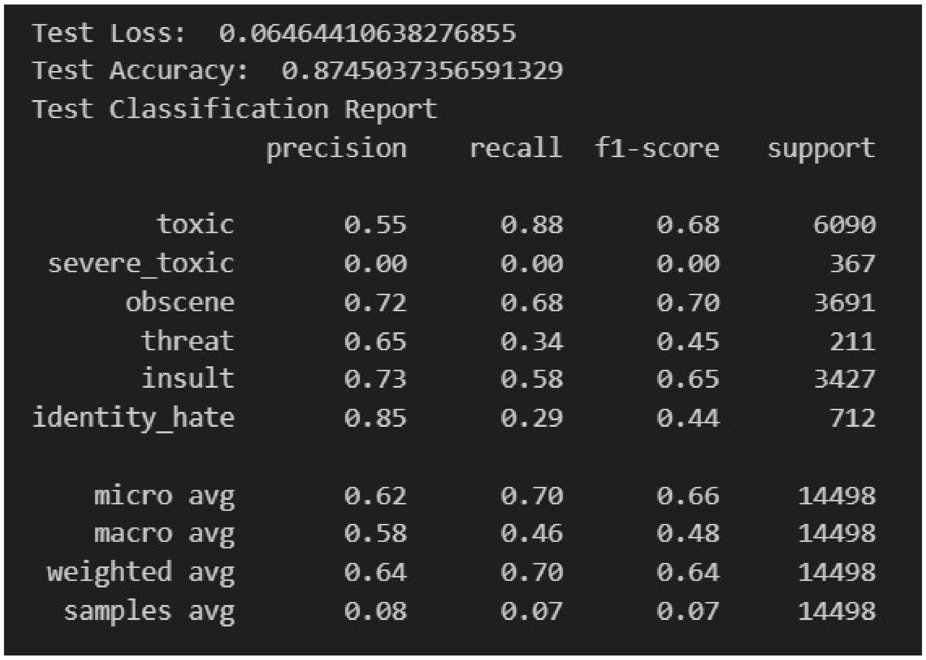
- We are using a pre-trained BERT model from a transformers module that has been trained on a large corpus of text using the MLM and NSP objectives. We then use our dataset of comments and their toxicity labels to fine-tune the model and make it more better/specialized at that task.

Validation Results



```
Epoch 1, Training Loss: 0.04801060528770128, Validation Loss: 6.794906342469118e-06, Validation Accuracy: 0.9258230280748663
Validation Classification Report
                precision    recall  f1-score   support

        toxic       0.82      0.84      0.83      4492
 severe_toxic       0.00      0.00      0.00       465
      obscene       0.90      0.73      0.81      2498
       threat       0.60      0.33      0.42       168
       insult       0.81      0.66      0.73      2325
identity_hate       0.80      0.25      0.39       445

    micro avg       0.83      0.70      0.76     10393
    macro avg       0.65      0.47      0.53     10393
 weighted avg       0.80      0.70      0.74     10393
  samples avg       0.07      0.07      0.07     10393
```

Test Results



```
Test Loss:  0.06464410638276855
Test Accuracy:  0.8745037356591329
Test Classification Report
               precision    recall  f1-score   support

        toxic       0.55      0.88      0.68      6090
 severe_toxic       0.00      0.00      0.00       367
      obscene       0.72      0.68      0.70      3691
       threat       0.65      0.34      0.45       211
       insult       0.73      0.58      0.65      3427
identity_hate       0.85      0.29      0.44       712

    micro avg       0.62      0.70      0.66     14498
    macro avg       0.58      0.46      0.48     14498
 weighted avg       0.64      0.70      0.64     14498
  samples avg       0.08      0.07      0.07     14498
```

- **Evaluation**
  - The model has a loss of 0.048 on train dataset and a loss of 6.78*10^-6 on validation dataset. It achieves an accuracy of 92.5% on validation dataset and 87.5% accuracy on the test dataset. This means it classifies a comment correctly and correctly decides the majority of the times. However, the value of accuracy can sometimes be misleading. When we have many different labels to predict, the overall accuracy might not show how the model does on each specific label, especially if some labels are much more common than others.
  - Toxic, Obscene, and Insult classes show relatively higher precision, recall, and F1-scores, indicating better model performance in predicting these categories. Severe_toxic and Threat classes have lower precision, recall, and F1-scores, suggesting challenges in accurately predicting these categories. For example, severe_toxic has zeros across precision, recall, and F1-score in this evaluation, indicating no correct predictions for this class. Identity_hate has high precision value but low recall and F1-score, indicating that the model has difficulty predicting a identity_hate comment, however when it does classifies a comment as identity_hate, it does so correctly most often.