Course Project - Comment Classification with RNNs

Deep Learning KU, WS 2024/25

Team Members		
Last name	First name	Matriculation Number
Nožić	Naida	12336462
Hinum-Wagner	Jakob	01430617
Haring	Dominik	12103719
Karsai	Áron	12427834

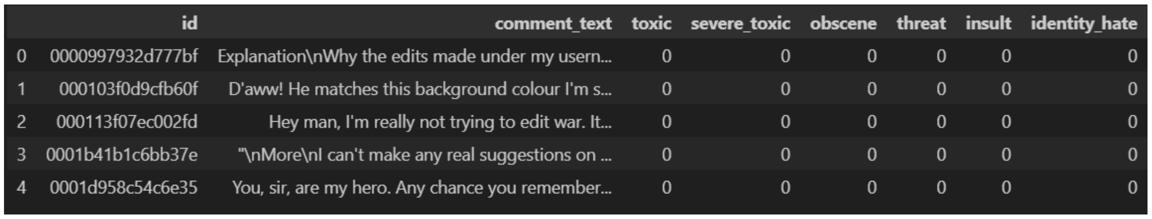
Contents

Chapter 1

Introduction

1.1 Visualization of data

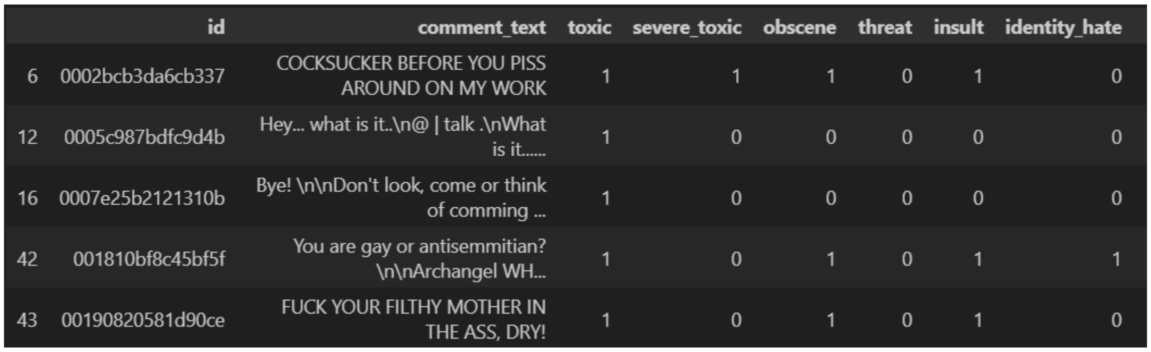
The topic of our project is toxic comment classification using Recurrent Neural Networks (RNNs). The dataset we used is from a Kaggle challenge named Toxic Comment Classification Challenge. To understand the task at hand let us first visualize the dataset. The data found in a CSV file looks the following loaded into a Pandas data frame:



	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 1.1.1: The first five rows of the dataset

As we can see from the figure the data consists of comments, each comment having a unique ID and six flags which label different kinds of toxicity. The value of a toxicity flag can be 1 or 0 (true or false) and a single comment can be flagged by multiple labels, i.e. a comment can be flagged as e.g. `obscene` and `threat` at the same time. So far we have not seen these flags in action since all comments in Figure 1.1.1 were unflagged – in the follwoing: positive. The flagging is illustrated by the following figure where we have selected some flagged – in the following: negative – comments to a data frame:



	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
6	0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
12	0005c987bdfc9d4b	Hey... what is it..\n@ | talk .\nWhat is it......	1	0	0	0	0	0
16	0007e25b2121310b	Bye! \n\nDon't look, come or think of comming ...	1	0	0	0	0	0
42	001810bf8c45bf5f	You are gay or antisemmitian? \n\nArchangel WH...	1	0	1	0	1	1
43	00190820581d90ce	FUCK YOUR FILTHY MOTHER IN THE ASS, DRY!	1	0	1	0	1	0

Figure 1.1.2: Some negative comments from the dataset

Let us now try to visualize some aspects of the dataset. To this end we first separate the comments to two set: positive comments and negative comments. Overall there are 159571 comments of which 143346 are positive and the remaining 16225 are negative. Having separated the negative comments we can visualize the distribution of the different flags (labels) over all the flags given to the negative comments. This is done via the following bar and pie charts:

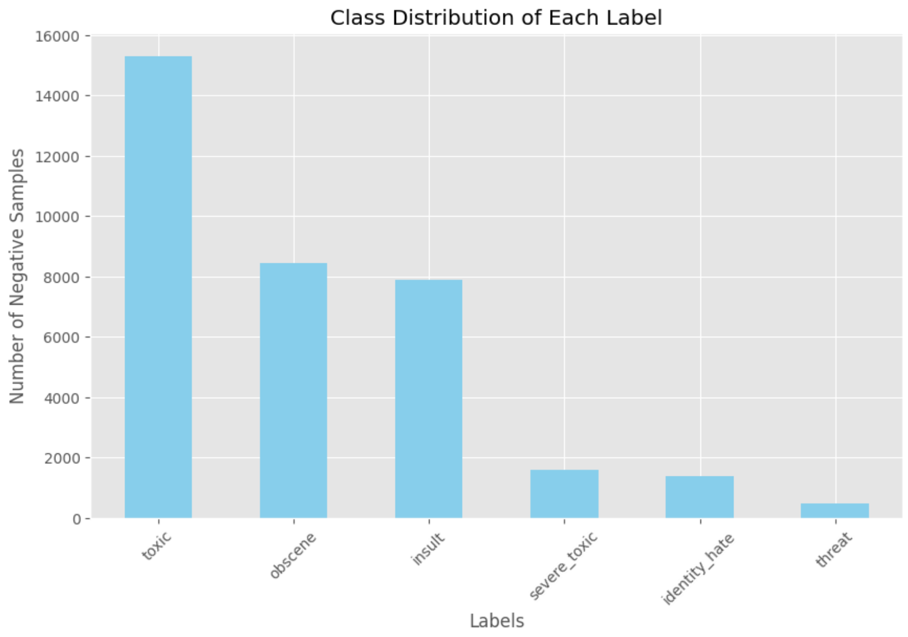


Figure 1.1.3: Bar chart of negative negative labels from all negative comments

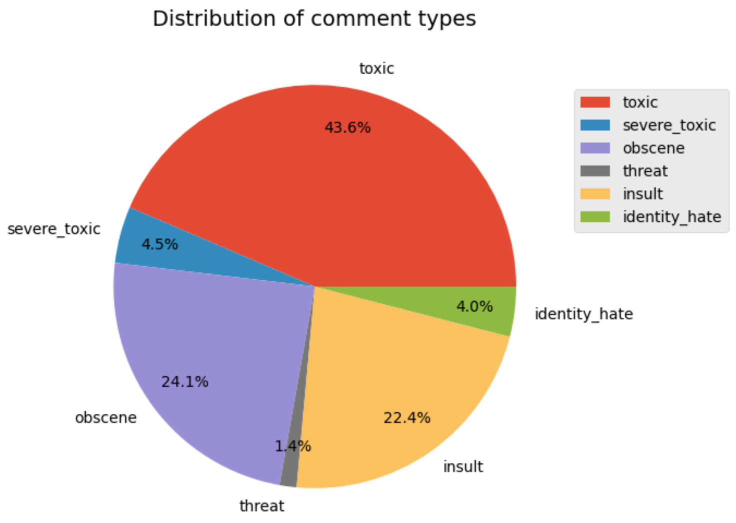
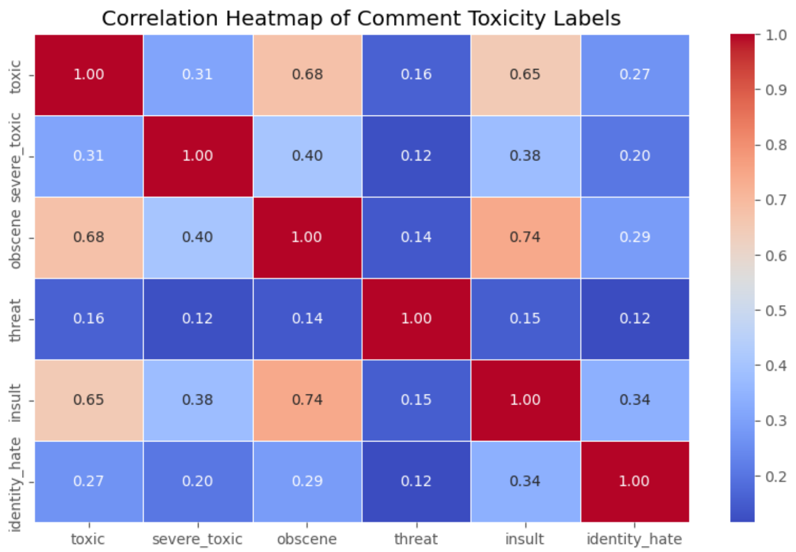


Figure 1.1.4: Pie chart of negative negative labels from all negative comments

Since one comment can have multiple labels we also show a correlation heat map that illustrates how likely each flag is to be given to the same comment as other flags:



	toxic	severe_toxic	obscene	threat	insult	identity_hate
toxic	1.00	0.31	0.68	0.16	0.65	0.27
severe_toxic	0.31	1.00	0.40	0.12	0.38	0.20
obscene	0.68	0.40	1.00	0.14	0.74	0.29
threat	0.16	0.12	0.14	1.00	0.15	0.12
insult	0.65	0.38	0.74	0.15	1.00	0.34
identity_hate	0.27	0.20	0.29	0.12	0.34	1.00

Figure 1.1.5: Correlation heat map of labels

Lastly we can also visualize the most common words in positive and negative comments:

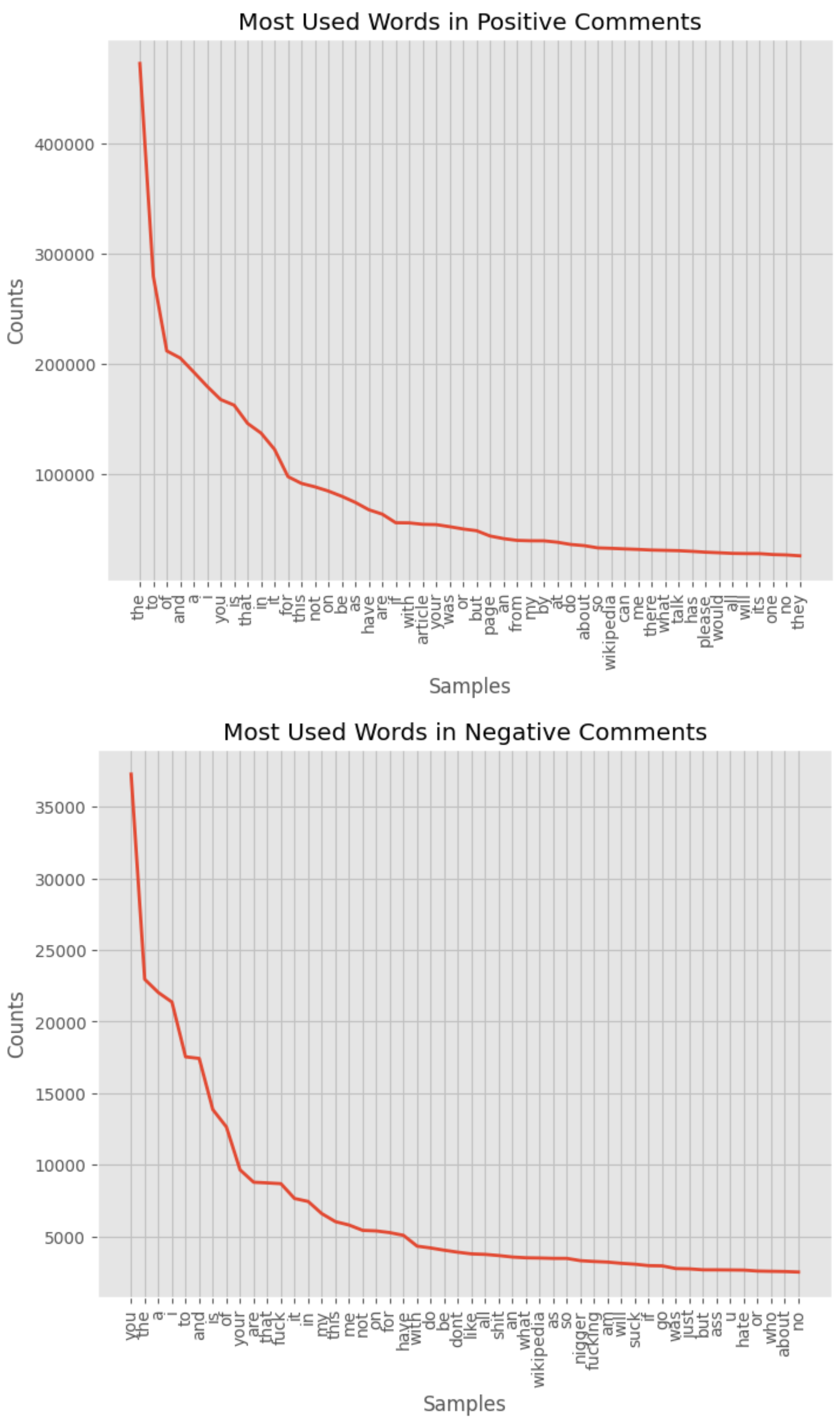


Figure 1.1.6: Most common words in each comment type

1.2 Problem formulation

Having been familiarized with the dataset now we can properly formulate the goal of this project. Our goal is to build an RNN-based deep neural network using reinforcement learning to classify a comment to the appropriate labels, i.e. give the comment the appropriate possibly multiple flags. For comparison we also implement a simple Logistic Regression classifier as a baseline model.

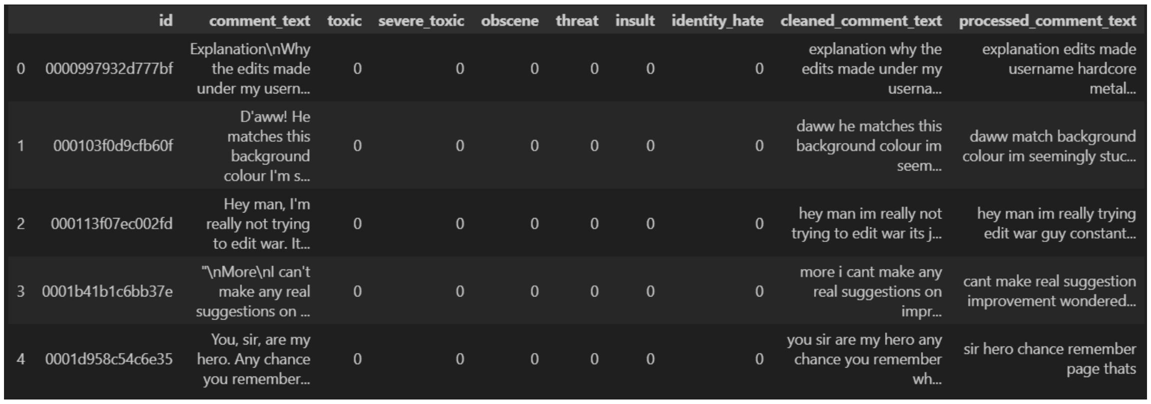
It can be clearly seen from Figure 1.1.6 that the most common words in each comment type are stop words in the english language such as ’the’, ’to’, ’of’, ’and’, etc. It would make training a neural network to identify the the meaningful words and label the comments accordingly much harder to have these words present in the dataset. Therefore, we need to preprocess the texts in the dataset.

1.3 Preprocessing the data

As it can be seen in Figures 1.1.1 and 1.1.2 some comments can include punctuation, special characters, numbers, line separators, CRLF characters and in some cases even links. We removed these from the `comment_text` fields of the dataset and added it to the CSV file as a new `cleaned_comment_text` column.

Furthermore, we also lemmatized the words of the cleaned comment texts to reduce the number of different words which carry the same meaning and removed the stop words altogether. We also added these texts to the CSV as a new `processed_comment_text` column.

These changes are illustrated with reference to Figure 1.1.1 in the following figure:



	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate	cleaned_comment_text	processed_comment_text
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0	explanation why the edits made under my userna...	explanation edits made username hardcore metal...
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0	daww he matches this background colour im seem...	daww match background colour im seemingly stuc...
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0	hey man im really not trying to edit war its j...	hey man im really trying edit war guy constant...
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0	more i cant make any real suggestions on impr...	cant make real suggestion improvement wondered...
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0	you sir are my hero any chance you remember wh...	sir hero chance remember page thats

Figure 1.3.1: The first five rows form the processed dataset

Chapter 2

Methods

2.1 Baseline model

We have begun the project with the implementation of a baseline model. It will be used as a starting point to which we compare the further implemented recurring neural networks. In the baseline model, a logistic regression classifier is applied to each class independently using TF-IDF (Term Frequency-Inverse Document Frequency) feature representation. It transforms our comments into numerical feature vectors. The model uses the definition of the default function $f(x) = \sigma(w \cdot x + b)$, where σ is the sigmoid function, trained for each label separately. The optimization objective is to minimize the binary cross-entropy loss, defined as:

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right]$$

Since we are training multiple logistic regression models on each class independently, the task is converted to a binary classification. The appropriate loss function is cross-entropy loss, since it measures the divergence between the predicted probabilities and actual labels, encouraging confident and correct predictions while penalizing incorrect ones. Furthermore, we have set the `max_iter` parameter to 1000 to ensure convergence during optimization. It is also important to note that the 80% of the data was used for training while for testing we have utilized 20%.

Building upon the baseline model, we decided to extend the baseline with an advanced approach by using a Multi-Output Logistic Regression classifier. This model uses logistic regression but addresses the multi-label classification problem, where each comment can belong to multiple categories at the same time. Besides the change in the model type, the rest of the implementation is identical. We transformed the text data into numerical representations using TF-IDF and trained the model with `max_iter` parameter set to 1000 to ensure convergence during optimization. The results with will discussed in the next chapter.

2.2 Background: RNNs and Deep Learning

2.2.1 Deep Learning Foundations

Deep learning is a subfield of machine learning that utilizes artificial neural networks with multiple layers (often referred to as "deep" architectures). These layers automatically learn data representations through a hierarchical process—from simple features in early layers to increasingly abstract representations in deeper layers. This hierarchical feature extraction is what endows deep learning with significant power in tasks like computer vision, natural language processing (NLP), and audio/speech recognition.

2.2.2 The Challenge of Sequential Data

Whereas many machine learning methods assume fixed-length and static inputs (such as a vector of features), many real-world domains involve sequence data (e.g., text or audio). This data can vary in length and often exhibits temporal or contextual dependencies. Handling such sequential information is essential for tasks including:

- Text classification
- Sentiment analysis
- Speech recognition
- Time-series forecasting

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of neural architectures specialized for sequential data. Unlike feed-forward networks, RNNs have a feedback loop connecting one time-step's output (or hidden state) to the next time-step's input, allowing them to maintain an internal memory of previous inputs.

Key Challenges:

- **Vanishing Gradients:** In long sequences, gradients can diminish, slowing or preventing learning.
- **Exploding Gradients:** In other cases, gradients can grow exponentially, destabilizing training.

2.2.3 Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU)

To combat the vanishing/exploding gradient issues, specialized architectures like LSTMs and GRUs were developed. These architectures feature gating mechanisms that help them retain important information over longer sequences, while discarding irrelevant details, making them more effective for tasks like language modeling and classification.

2.2.4 Bidirectionality and Stacked RNN Layers

Bidirectional RNN: Processes the sequence forwards and backwards, giving the network context from both past and future tokens.

Stacked/Deep RNN: By stacking multiple RNN layers, the model can learn increasingly higher-level features of the sequence, albeit with a higher risk of overfitting if not carefully regulated (e.g., via dropout).

2.3 RNN Model Implementation

In this project, we addressed a multi-label toxicity classification task using a text dataset of user-generated comments. Each comment is annotated for multiple toxicity-related categories (e.g., *toxic*, *severe_toxic*, *obscene*, *threat*, *insult*, *identity_hate*). Our objective was to identify which of these toxicity labels apply to a given comment.

Below is an overview of the methodology, including details on our code implementation, the rationale behind each design choice, and the formal definitions of the loss functions. We present two main recurrent neural network (RNN) models based on LSTM layers: the *Independent Probabilities Model* (multi-label) and the *Joint Probabilities Model* (multi-class). Although both models share similar low-level components, they differ in how the output layer and loss function are structured.

2.3.1 Data Loading and Preprocessing

1. **Text Cleaning:** We removed or normalized special characters and noise from each comment, ensuring a cleaner input for tokenization.
2. **Tokenization:** Each comment was split into tokens (words or subwords). These tokens form the basis for the numerical representation.
3. **Integer Encoding:** We constructed a vocabulary dictionary and mapped each token to an integer ID. This step transforms raw text into integer sequences.
4. **Padding:** We standardized all integer sequences to a fixed length (e.g., 100 tokens) so they can be batched efficiently.

2.3.2 Embedding Layer Construction

After integer encoding, we introduced an embedding layer:

- Implemented via `nn.Embedding` in PyTorch.
- Maps integer token IDs to dense embedding vectors (for example, 128 dimensions).
- Reduces the dimensionality for the LSTM and allows learning of task-specific semantic embeddings.

2.3.3 Recurrent Neural Network Architecture

Both models share the following core components:

- **Embedding Layer:** Converts each integer token to a trainable vector representation.
- **LSTM (2-layer):** We use a two-layer Long Short-Term Memory network (`num_layers = 2`). LSTM gating mechanisms mitigate vanishing gradients by allowing important information to persist through time steps.
- **Dropout:** Placed after the LSTM (and optionally between LSTM layers), randomly zeroing part of the hidden representation to reduce overfitting.
- **Fully-Connected (FC) Layer:** Maps the final hidden state(s) to the output dimension required by each model variant.
- **Gradient Clipping:** We apply norm-based clipping (`clip_grad_norm_`) at a threshold of 5 to prevent exploding gradients.

We explored *two variants* to handle multi-label toxicity classification:

Independent Probabilities Model (Multi-Label)

Architecture.

$$\texttt{Tokens} \to \texttt{Embedding} \to \texttt{2-layer LSTM} \to \texttt{Dropout} \to \texttt{Fully-Connected} \to \sigma(\cdot) \to \textbf{Sigmoid Outputs (6)}$$

- **Output Dimensionality:** 6 (one output for each toxicity label).
- **Activation:** A sigmoid function σ maps each final logit to $(0, 1)$, producing a probability for each label independently.

Interpretation. Each of the six labels (*toxic*, *severe_ toxic*, *obscene*, *threat*, *insult*, *identity_ hate*) is an independent binary classification. A single comment can simultaneously be labeled *obscene* and *threat*, making multi-label classification natural.

Loss Function. We use `nn.BCELoss` (binary cross-entropy loss) across the six labels, defined formally in Section 2.4.

Joint Probabilities Model (Multi-Class)

Architecture.

`Tokens` → `Embedding` → `2-layer LSTM` → `Dropout` → `Fully-Connected` → `Softmax` → **64-Class Output**

- **Output Dimensionality:** 64 classes, corresponding to 2^6 possible combinations of 6 labels.
- **Activation:** A softmax layer normalizes these 64 logits into a single probability distribution.

Interpretation. Instead of modeling each label independently, each unique set of labels is treated as a single "class." This captures correlations between labels but can quickly become large or imbalanced if many labels exist.

Loss Function. A single cross-entropy loss (`nn.CrossEntropyLoss`) is applied, treating each label-combination as one class.

2.3.4 Addressing Vanishing/Exploding Gradients

- **LSTM Gating:** By design, LSTMs incorporate input, forget, and output gates that help mitigate the vanishing gradient problem over long sequences.
- **Gradient Clipping:** We clip gradients to a maximum norm of 5, preventing excessively large updates (exploding gradients) in deeper or more complex recurrent layers.

2.3.5 Training Details

- **Optimizer:** We use Adam with an initial learning rate of 0.001, chosen for its wide applicability and adaptive step sizes.
- **Regularization:** Dropout is applied in the LSTM to reduce overfitting by randomly zeroing hidden units.
- **Learning Rate Scheduler:** A linear learning-rate schedule (`lr_scheduler.LinearLR`) gradually decreases the learning rate across epochs.
- **Epochs and Batch Size:** Models are typically trained for 10 epochs with a batch size of 64, balancing computational efficiency and stable gradient estimates.

2.3.6 Architectural Choices and Model Selection

Ultimately, the Independent (multi-label) approach provided a flexible way to handle overlapping toxicity categories, while the Joint (multi-class) approach allowed us to explicitly model label dependencies. Each configuration has its merits:

1. **Independent (Multi-Label) Classification:** Sigmoid outputs + BCE handle each label as a separate Bernoulli trial.
2. **Joint Probability (Multi-Class) Classification:** Softmax outputs + CE treat each combination of labels as a single class.

The following sections detail the loss functions mathematically, explaining why each method is appropriate for its respective classification task. Numerical results for these models are presented later.

2.4 Loss Functions

We employed two different loss functions, reflecting our two approaches to multi-label toxicity detection.

2.4.1 Binary Cross-Entropy (BCE) for Multi-Label

In the **Independent Probabilities Model**, each label $y_{ij} \in \{0, 1\}$ for label j in sample i is treated separately. The output of the model for that label is

$$p_{ij} = \sigma(z_{ij}), \quad \sigma(z) = \frac{1}{1 + e^{-z}},$$

where z_{ij} is the logit from the final fully-connected layer.

Let K be the number of labels (here $K = 6$) and N be the number of training samples. The binary cross-entropy loss over the entire dataset is

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} \Big[y_{ij} \log(p_{ij}) + (1 - y_{ij}) \log\big(1 - p_{ij}\big) \Big].$$

Interpretation and Rationale.

- *Independence of Labels:* Each label j uses a sigmoid output $\sigma(z_{ij})$ and a binary (on/off) cross-entropy term. This naturally encodes the idea that any subset of labels can be active (e.g., *insult* and *obscene*).
- *Ease of Implementation:* BCE is straightforward to implement (via `nn.BCELoss`) and commonly used for multi-label tasks.

2.4.2 Cross-Entropy (CE) for Multi-Class (64 Classes)

In the **Joint Probabilities Model**, we treat each of the 2^K label combinations as one "class." For $K = 6$, we get 64 possible classes. The model outputs a logit $z_{i,c}$ for each class $c \in \{1, \dots, 64\}$, then applies a softmax:

$$p_{i,c} = \text{softmax}(z_{i,\cdot}) = \frac{\exp\big(z_{i,c}\big)}{\sum_{c'=1}^{64} \exp\big(z_{i,c'}\big)}.$$

If the ground truth for sample i is the class c_i that matches its label combination, the cross-entropy loss is

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \log\big(p_{i,c_i}\big).$$

Interpretation and Rationale.

- *Label Dependencies:* By treating each label combination as a single class, we can capture correlations among labels (e.g., "obscene + threat" might be more common than "obscene + identity_hate").
- *Single Probability Distribution:* The model outputs exactly one class per sample, which can simplify post-processing if you want one definitive combination.
- *Limitations:* As K grows, 2^K becomes prohibitively large; certain combinations may be very rare, making training and inference difficult.

2.4.3 Why These Losses Are Sensible Choices

Multi-Label (BCE). Each label is modeled as an independent Bernoulli random variable. BCE over sigmoids is the most direct way to implement multi-label classification, allowing any combination of labels to be active.

Multi-Class (CE). Standard cross-entropy is widely used for single-label classification. Here, we extend it to a multi-label context by enumerating every label combination as a distinct class. This can capture correlations but at the cost of an exponentially larger label set.

General Benefits.

- BCE and CE are well-studied and easily implemented with `nn.BCELoss` or `nn.CrossEntropyLoss` in PyTorch.
- Both are convex with respect to the logits (in a single-label or binary sense), which aids optimization stability.

- The final choice depends on whether we want independent probabilities for each label or a single probability distribution over label combinations.

In summary, the Independent Probabilities Model (BCE loss) offers the intuitive flexibility of multi-label classification, while the Joint Probabilities Model (CE loss) captures label inter-dependencies in a single multi-class framework. Both rely on a shared LSTM backbone with gradient clipping, dropout, and an embedding layer for token representations. The next sections detail the evaluation metrics and compare performance under each configuration, without presenting the numerical results at this stage.

Chapter 3

Results

3.1 Baseline model

As mentioned in the previous chapter, the baseline logistic regression model was trained using TF-IDF features extracted from the preprocessed text data. It was trained on each class separately using 80% of the data. Once trained, we have evaluated its performance on 20% of the data, reserved for testing. The utilized metrics are accuracy, F1 score, precision, and recall for each label.

The evaluation results were collected into a summary table highlighting the model's strengths and areas for improvement.

Class	**Accuracy**	**F1 Score**	**Precision**	**Recall**
toxic	0.956948	0.730694	0.911046	0.609948
severe_toxic	0.990600	0.375000	0.566038	0.280374
obscene	0.976218	0.736000	0.912069	0.616910
threat	0.997681	0.177778	0.500000	0.108108
insult	0.969325	0.626478	0.815293	0.508674
identity_hate	0.991634	0.260388	0.701493	0.159864

Table 3.1: Performance metrics of the logistic regression model for each class

The model achieves high accuracy across all classes, with values ranging from 0.95 to 0.99. However, the F1 score, precision, and recall vary significantly among the classes, which points towards a potential class imbalance and varying prediction difficulties.

- The **toxic** and **obscene** classes exhibit relatively high F1 scores (0.73 and 0.73, respectively), suggesting a good balance between precision and recall. Both classes also have high precision values, meaning the model is effective at correctly identifying positive instances.

- The **severe_toxic**, **threat**, and **identity_hate** classes shave lower F1 scores (0.37, 0.17, and 0.26, respectively). This is probably because of the low recall values. The model struggles to capture all actual positive cases for these categories, leading to a higher rate of false negatives.
- The precision values for some classes, such as **identity_hate** (0.70), indicate that when the model does predict a positive instance, it is often correct. However, the low recall (0.15) suggests that many true instances are missed.

Overall, while the model performs well for more frequently occurring classes, it struggles with rare or nuanced categories, highlighting the need for further improvements such as data balancing, feature engineering, or the use of more complex models.

Regarding the Multi-Output Logistic Regression classifier, we have also used the trained model to predict the labels on the test set. For assessing the performance, we compute the same metrics, including accuracy, F1-score (both macro and micro averages), precision, and recall. These metrics have provided a detailed view of the model's ability to make correct predictions across all labels, where macro averages treat all classes equally, regardless of the class imbalance and micro metrics put more weight on classes with more samples which is useful in our case since we have an imbalanced dataset.

Overall Evaluation Metrics on Test Set pf the Multi-Output Logistic Regression:

- Accuracy: 0.9179
- Macro-Averaged F1: 0.4844
- Micro-Averaged F1: 0.6745
- Macro-Averaged Precision: 0.7343
- Micro-Averaged Precision: 0.8727
- Macro-Averaged Recall: 0.3806
- Micro-Averaged Recall: 0.5496

The evaluation metrics tell us that the model performs relatively well overall, with an accuracy of 91.79%. However, the macro-averaged F1 score of 0.4844 indicates that the model struggles to perform consistently across all classes, as the metric is sensitive to class imbalances. On the other hand, the micro-averaged F1 score of 0.6745 provides a better picture of the model's performance across all classes combined. It shows an okay performance.

Precision is relatively high for both macro (0.7343) and micro (0.8727) averages. This means that when the model predicts a positive label, it is often correct. Unfortunately, recall is low with macro recall at 0.3806 and micro recall at 0.5496. indicating that the model may miss some positive instances, especially for less frequent classes.

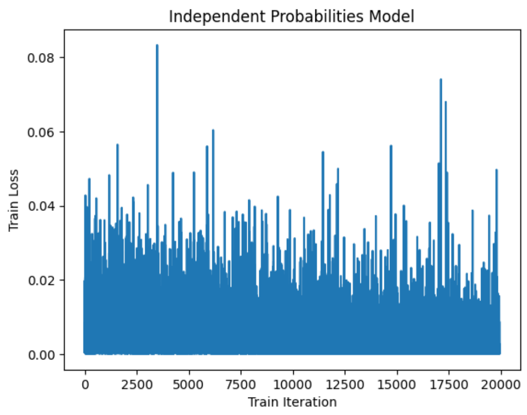
3.2 RNN models

The independent and joint models were trained separately with the appropriate loss functions such as the Binary Cross-Entropy for the independent model and Cross-Entropy for the joint model. During training we also decided to fine tune both models with hyper parameters and regularization techniques. For the learning rate we implemented a linear learning rate scheduler which after each epoch decreases the learning rate of the model. For both models we started out with an initial learning rate of 0.001. We also added dropout regularization to both models, where a smaller value for $p = 0.1$ worked out much better for the independent model, whereas for the joint model we chose to keep a higher value of $p = 0.5$. We also implemented gradient clipping to a maximum L2-norm of 5 to both models, however in practice this did not yield any form of meaningful improvement during training.

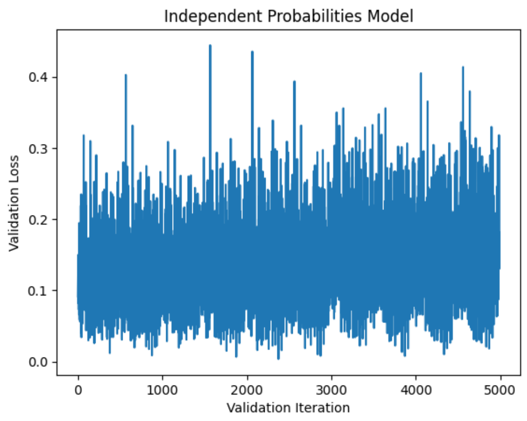
Overall the independent model ended up having a total of 30363910 trainable parameters, while the joint model was close in size but slightly different at 30378816 trainable parameters in total.

The independent model showcased a more steady decrease in the training loss over the iterations. The lowest detected train loss was of value 0.0207. On the other hand the joint model presented with slightly more oscillations and significantly struggled to decrease. The lowest train loss was 3.1912. Therefore, we can say that the independent model had learned more meaningful patterns in the data due to the steady and stable convergence. Whereas the second model that oscillates exhibited fluctuations in loss, failing to settle into a downward trend, which points towards an inability to detect accurate patterns in the data.

Once trained, we have evaluated their performance with accuracy, F1-score, precision and recall. For a better comparison, we have used the micro averages for an accurate performance evaluation with imbalanced labels. The independent model, which predicts labels independently, has a strong overall performance with an accuracy of 89.48%, an F1-score of 68.38%, precision of 70.92%, and recall of 66.02%, indicating decent prediction power. However, the joint model, which predicts a probability distribution over all 64 label configurations has showed significantly lower accuracy at only 3.37%, f1-score of 15.68%, precision of 9.58% and recall of 43%. These results suggest very bad overall prediction power, which could be due to the existing imbalance in the data and the complexity of label dependencies.

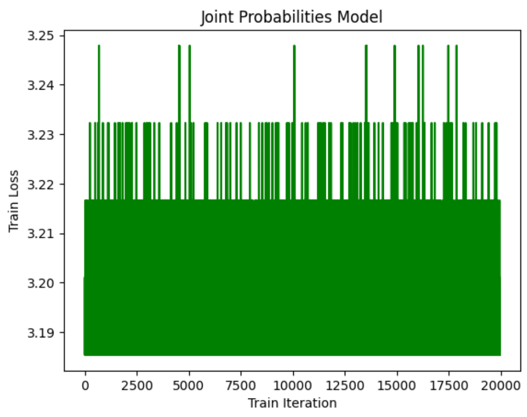


(a) Train Loss

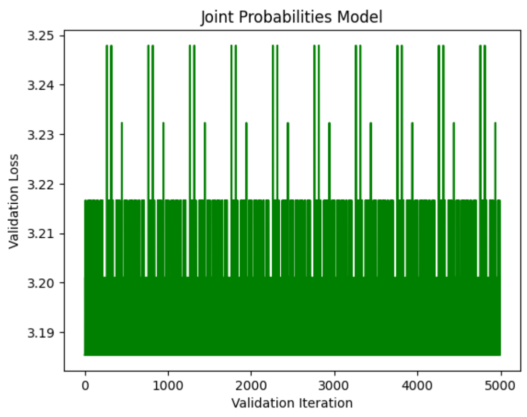


(b) Validation Loss

Figure 3.2.1: Independent Probabilities Model Losses



(a) Train Loss



(b) Validation Loss

Figure 3.2.2: Joint Probabilities Model Losses

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
Independent Probabilities Model	0.89277	0.68924	0.68653	0.69197
Joint Probabilities Model	0.03374	0.15676	0.09575	0.43200

Table 3.2: Performance metrics of the Independent and Joint Probability Models

Chapter 4

Discussion and Conclusion

4.1 Main observations

As seen in the previous chapter our results did vary. Some aspects of the classifier scores were even better in the baseline model than with the RNN models. Our model which considered joint probabilities did very poorly on both these metrics and the train and validation losses graph.

The RNN model with the independent probabilities is definitely our choice for our final model from the RNN models. However, even this RNN model falls short to the baseline model in terms of accuracy and F1 score. Granted, it gives better results for precision and recall but it can also be seen in the training and validation data that our model hardly converged.

Still, there remains one more way to determine which model is actually usable in practice. Namely, to see *why* the models actually give the flags they give to specific comments. We explore this in the next section.

4.2 Explaining text classification with LIME

With deep and complicated classifier neural networks there is a drawback of not being able to determine *why* exactly the model gives the predictions it does. Local Interpretable Model-Agnostic Explanations – LIME for short – is a technique that aims to gives explanations for these whys. It works for *any* classifier structure by learning an interpretable model locally around a specific prediction [1]. To this end we used the `lime` library provided by the authors of [1].

In the following subsections we use these explanations first for the baseline model in 4.2.1

and then for the RNN structure in 4.2.2. In both cases we first show explanations for a comment which the model deemed positive with regards to its `toxic` flag and then for comments which the model flags for each different label. All of these comments were taken from the test set. The green and red lines on the plots indicate which word influences the model make a decision into which direction locally.

4.2.1 Explaining the baseline model

- Positive comment: Carioca RFA Thanks for your support on my request for adminship. The final outcome was (31/4/1), so I am now an administrator. If you have any comments or concerns on my actions as an administrator, please let me know. Thank you!
- Predicted flag for toxic: 0
- True flag for toxic: 0

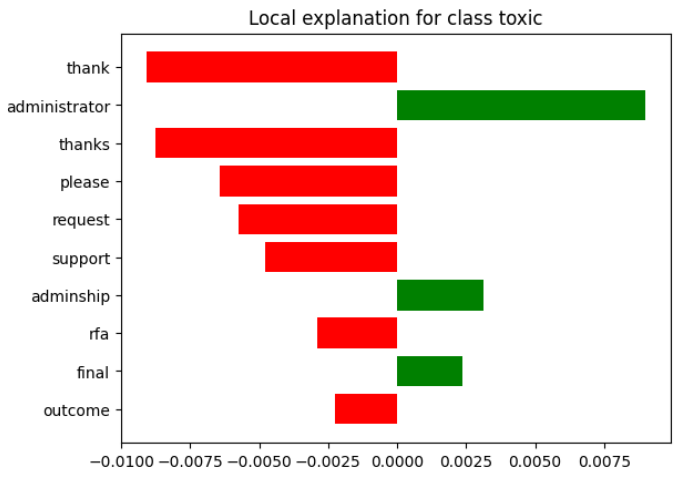


Figure 4.2.1

- Toxic comment: no worries We will use another account, and will not stop until wikipedia stops pushing the evil, sin filled gay agenda
- Predicted flag for toxic: 1
- True flag for toxic: 1

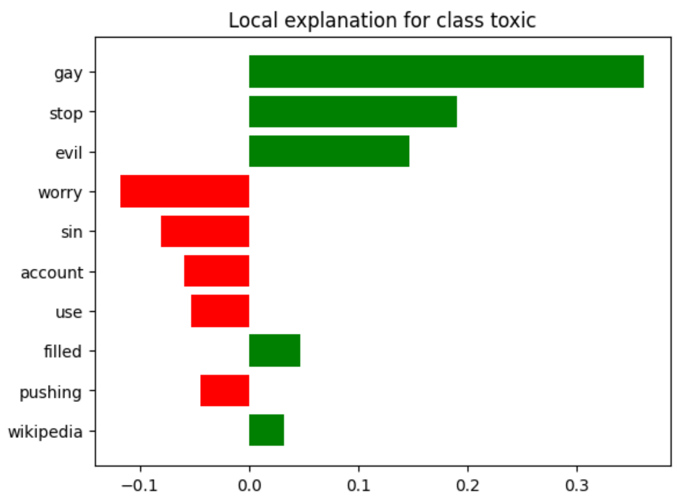


Figure 4.2.2

- Severe toxic comment: you fucking piece of shit
- Predicted flag for severe_toxic: 1
- True flag for severe_toxic: 1

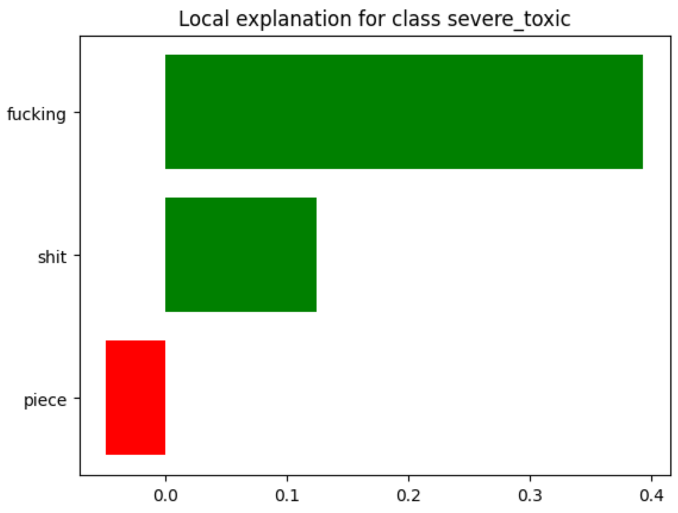


Figure 4.2.3

- Obscene comment: Renzoy16 ! Shut The Fuck Up
- Predicted flag for obscene: 1
- True flag for obscene: 1

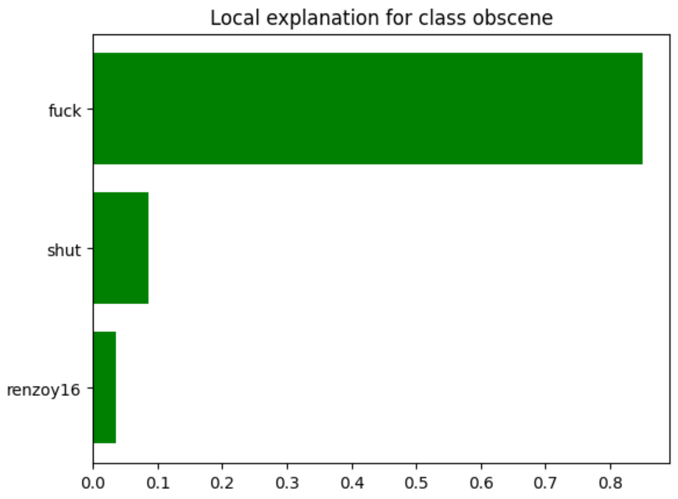


Figure 4.2.4

- Threat comment: FUCK BILLCJ AND KILL BILLCJ BECAUSE I HATE HIS BIG FAT FUCK KILL BILLCJ!! KILL BILLCJ! KILL BILLCJS ASS! KILL HIS BIG FAT FUCK!! KILL HIS ASS! KILL BILLCJ! FUCK BILLCJ KILL HIS ASS! KILL BILLCJ!!! KILL BILLCJ!! I HATE HIM! KILL BILLCJ AND AVENGE TOUGHHEAD!!! KILL HIS ASS!!!
- Predicted flag for threat: 1
- True flag for threat: 1

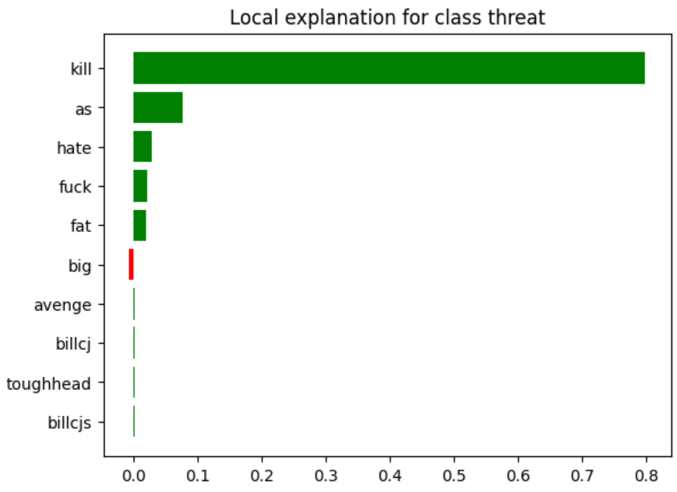


Figure 4.2.5

- Insult comment: Suck my dikkkkk == Suck my dikkkkk Gogo Gogo giving blowjobs on demand
- Predicted flag for insult: 1
- True flag for insult: 1

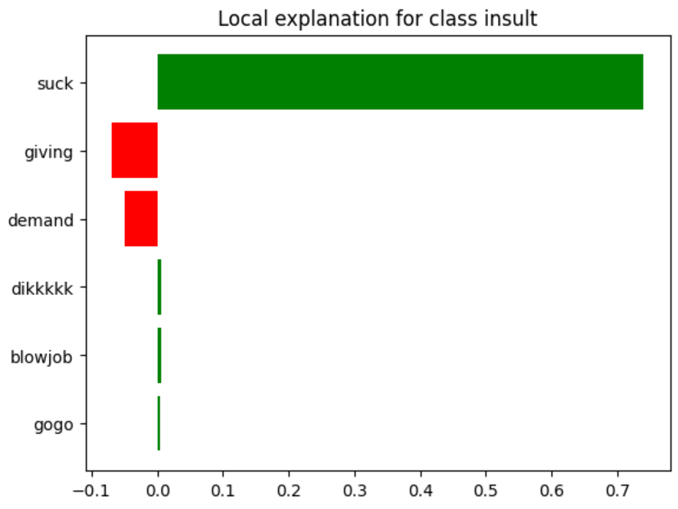


Figure 4.2.6

- Identity hate comment: your gay douche like gay porno ahaghagagagaa
- Predicted flag for identity_hate: 1
- True flag for identity_hate: 1

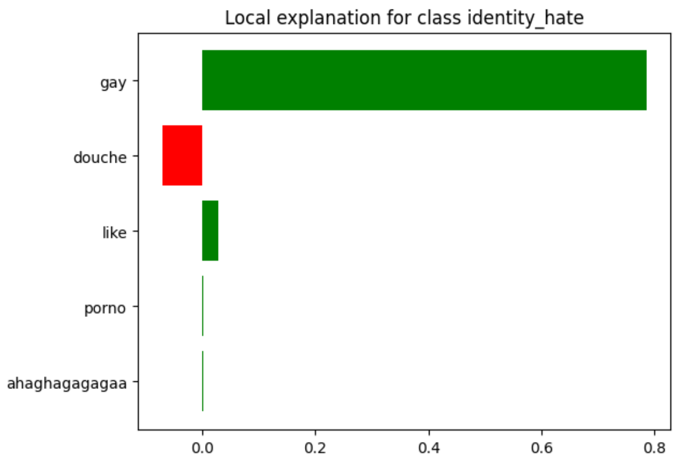


Figure 4.2.7

4.2.2 Explaining the RNN structure

- Positive comment: Why the edits made under my username Hardcore Metallica Fan were reverted? They weren't vandalisms, just closure on some GAs after I voted at New York Dolls FAC. And please don't remove the template from the talk page since I'm retired now.89.205.38.27
- Predicted flag for toxic: 0
- True flag for toxic: 0

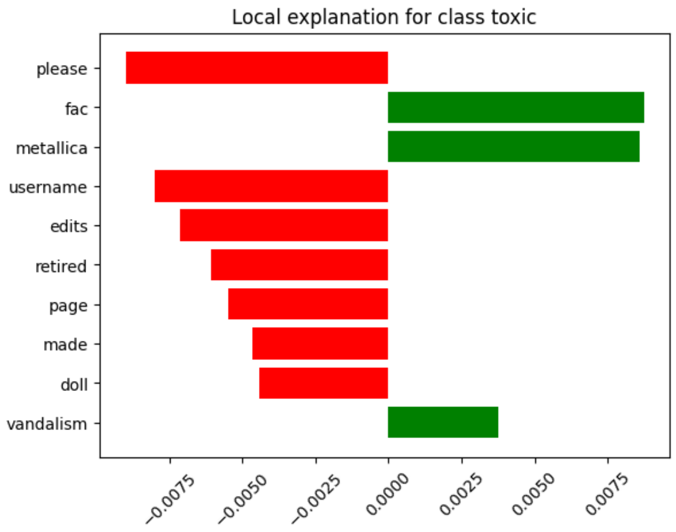


Figure 4.2.8

- Toxic comment: COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK
- Predicted flag for toxic: 1
- True flag for toxic: 1

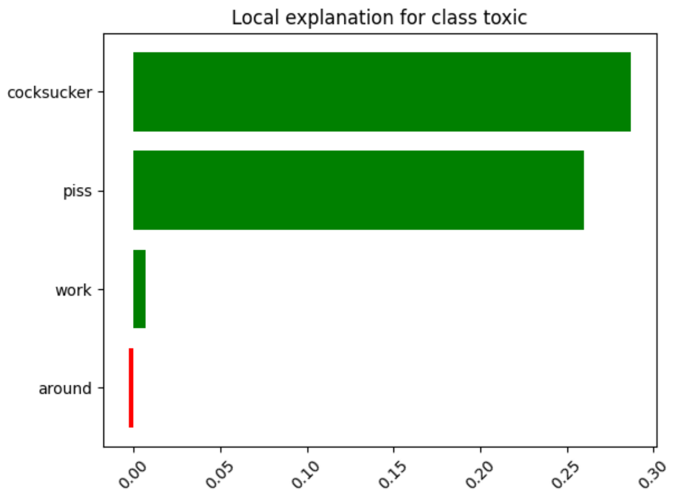


Figure 4.2.9

- Severe toxic comment: Stupid peace of shit stop deleting my stuff asshole go die and fall in a hole go to hell!
- Predicted flag for severe_toxic: 1
- True flag for severe_toxic: 1

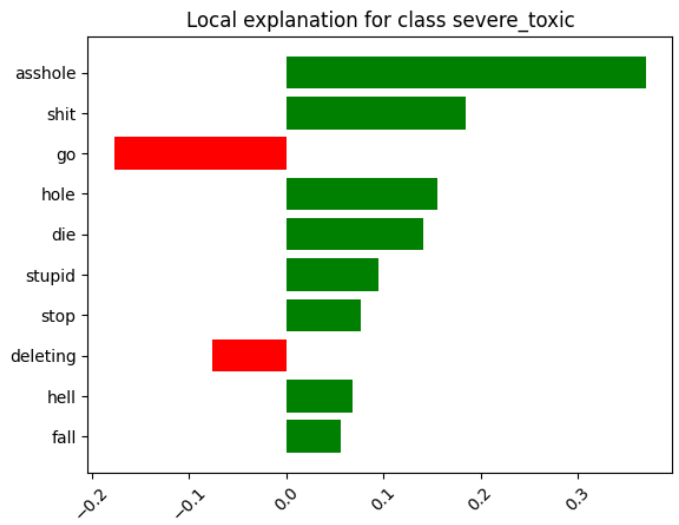


Figure 4.2.10

- Obscene comment: GET FUCKED UP. GET FUCKEEED UP. GOT A DRINK THAT YOU CANT PUT DOWN???/ GET FUCK UP GET FUCKED UP. I'M FUCKED UP RIGHT NOW!
- Predicted flag for obscene: 1
- True flag for obscene: 1

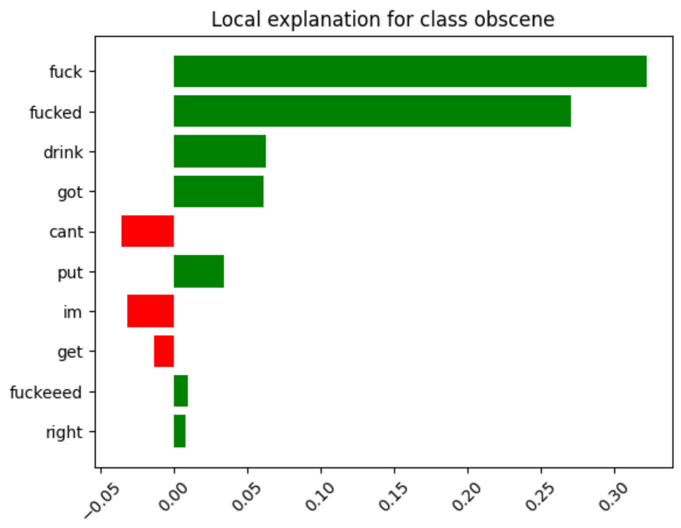


Figure 4.2.11

- Threat comment: Hi! I am back again! Last warning! Stop undoing my edits or die!
- Predicted flag for threat: 1
- True flag for threat: 1

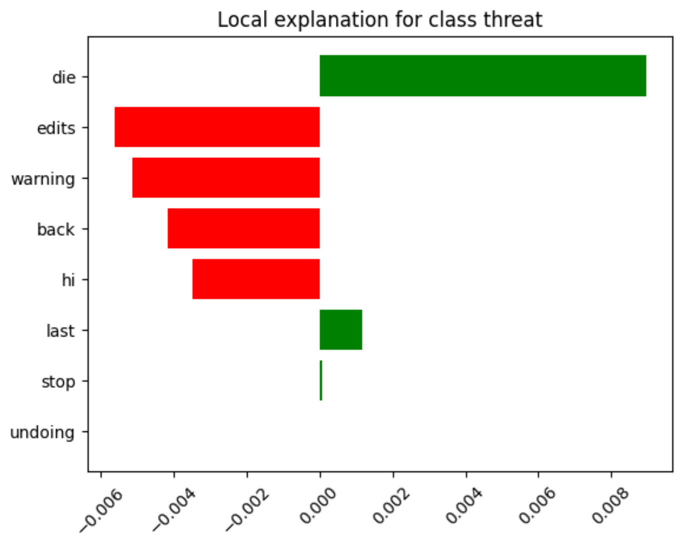


Figure 4.2.12

- Insult comment: COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK
- Predicted flag for insult: 1
- True flag for insult: 1

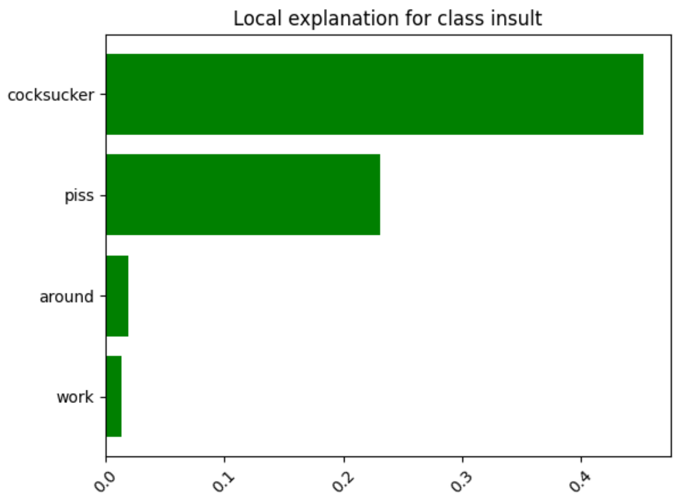


Figure 4.2.13

- Identity hate comment: u r a tw@ fuck off u gay boy.U r smelly.Fuck ur mum poopie
- Predicted flag for identity_hate: 1
- True flag for identity_hate: 1

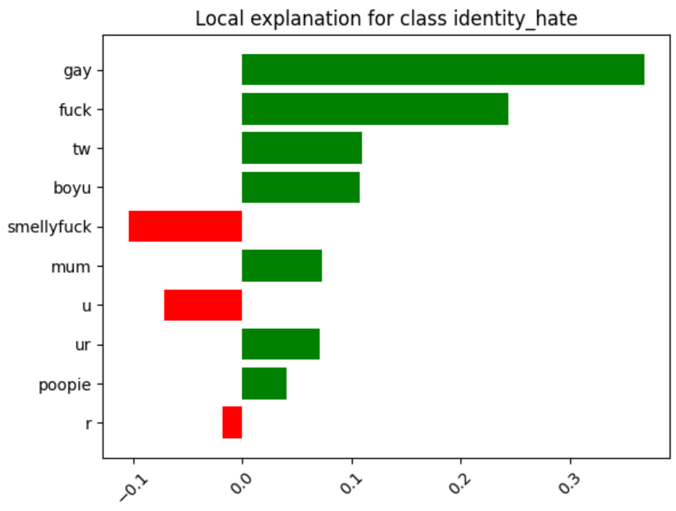


Figure 4.2.14

4.3 Limitations and conclusions

It can be seen from the LIME evaluations that both the baseline and the RNN architectures can grasp which words from the comments lead to the comments being flagged – when the true positive case stands of course.

However generally the RNN model despite being much more complicated is comparable in performance if not worse than our baseline model. It can not even generalize better than the baseline. Therefore we would recommend using the baseline out of these models for this relatively simple calssification task.

Bibliography

[1] M. T. Ribeiro, S. Singh, and C. Guestrin, “"why should I trust you?": Explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 2016, pp. 1135–1144.