Introduction to Machine Learning - Final Project

Garuglieri Chiara, Sanson Valentina, Scarpa Claudia

16/07/2025

1 Introduction

The assignment involves a multi-label classification task using a dataset of short children's stories, each of them labeled with zero or more tags. It consists of solving six binary classification problems, and our aim is to develop and compare two different models to perform this task. In addition, we will analyze the different labels - *BadEnding*, *Conflict*, *Dialogue*, *Foreshadowing*, *MoralValue*, and *Twist* - and discuss which ones were more difficult to predict and why. The first model (Model A) is a pre-trained language model called RoBERTa, an improved version of BERT developed by Facebook AI (Meta AI) (link to the paper): it is one of the most widely used models in Natural Language Processing (NLP) for tasks such as text classification, sentiment analysis, question answering, and more. The second model (Model B) we developed is a convolutional neural network (CNN) that we built from scratch. CNNs are a class of neural networks that are able, using convolutional filters, to hierarchically learn different features, from simple patterns to more complex ones and are commonly applied in text processing.

1.1 Analysis of the dataset

The code provided divides the dataset into a test set (10,000 stories) and a training set (2,735,100 stories), in which the average length of a story is approximately 160 words (excluding punctuation) and the longest story exceeds 1,000 words. Before implementing the models, we analyzed the main characteristics of the training set in order to have an overview of all the data. We found that most of the stories are labeled with a single tag, while the rest are almost equally divided between stories with two tags and stories without tags (about 20% each). Very few stories have three or more tags, making such cases relatively rare in the dataset. We also examined the distribution of individual labels: over 50% of the stories are labeled with the tag *Dialogue*, around the 20% with *Twist* and each of the remaining tags appears in approximately 10% of the training stories. In addition, we analyzed potential co-occurrences between labels. As expected, the *Dialogue* tag frequently appeared alongside other tags, given that it is the most common label. However, no strong associations were found among the remaining labels: the maximum co-occurrence between any two non-*Dialogue* tags was only 12%. These observations were useful to understand that the tag distribution was not uniform, meaning that some classes were more frequent than others.

2 Data Preprocessing

As previously mentioned, the original dataset was very large and due to technical limitations, in particular the long training times, we were unable to use it in its entirety. Specifically, we trained Model A using 50,000 stories, while Model B, which is computationally lighter, was trained on 400,000 stories. To create these smaller datasets, we implemented a function that extracts a subset of the full dataset. Since purely random sampling could lead to some classes being heavily underrepresented - or even missing entirely - we enforced a minimum number of stories per tag. This threshold was set at 2% of the total subset of data to ensure each class had at least a basic level of representation, without significantly altering the data distribution. The remaining stories were selected randomly, to preserve the overall class distribution as closely as possible to the original dataset.

2.1 Model 1: RoBERTa

For Model A we used RobertaTokenizer from the Hugging Face Transformers library, as it is the natural choice when using the homonymous model. It breaks down text into smaller units called subwords, which helps handle rare or complex words more effectively. The tokenizer also converts these subwords into numerical representations that the model can process, while managing input length through padding and truncation to ensure consistency. We chose a maximum length of 256 tokens, as the stories had an average length of approximately 160 words with a standard deviation of about 66. Although RoBERTa can handle sequences up to 512 tokens, we observed that using the full length significantly increased training time without leading to substantial improvements in performance.

2.2 Model 2: CNN

For Model B, we implemented a simple custom tokenizer and chose to keep the punctuation as separate tokens. We considered this important for our task, as punctuation marks such as quotation marks (used to introduce dialogue) and exclamation points can carry meaningful information and help capture the structure and tone of the stories. We then created our vocabulary by collecting all tokens generated from tokenizing our small dataset. It is important to note that in the test script we also loaded the same small dataset to reconstruct the exact vocabulary used during training to guarantee consistency of the model predictions. Since punctuation is included as tokens, the length of the tokenized stories increased. To determine an appropriate maximum sequence length for padding and truncation, we calculated the value corresponding to the 95th percentile of the tokenized training dataset. This choice ensures that most stories fit within the maximum length without excessive truncation.

After these preprocessing steps, the datasets were correctly labeled and ready for use with machine learning algorithms. To facilitate efficient training and evaluation, we then converted the datasets into DataLoaders.

3 Model 1: RoBERTa

As mentioned before, RoBERTa is a robust approach to the BERT model, a pre-trained transformer from the HuggingFace library. Since it has already been trained on a large amount of text data, it can be fine-tuned on specific classification tasks using relatively a small amount of labeled data. RoBERTa is well suited for text classification, in fact it creates embeddings that reflect the meaning of each word in its context, which is essential for understanding the overall meaning of a sentence.

3.1 Model Architecture

The most important component in transformers is the self-attention layer, which computes relationships between all tokens in a sequence. The pure self-attention mechanism, that is without any positional information, is permutation equivariant, meaning that if the input tokens are permuted, the output tokens will be permuted in the same way. However, RoBERTa is not permutation equivariant, because it includes positional embeddings that encode the order of the tokens. Additionally, a key component inside the self-attention layer is the attention mask which instructs the model to focus on the tokens that are meaningful for the task (i.e. real words in the sentence) and to ignore the padding tokens added during the padding and truncation phase.

3.2 Training

During the training and validation phase, we made several choices in order to make our model more efficient and better suited to the task. Since we are dealing with a multi-label classification problem, the most appropriate loss function is the Binary Cross-Entropy with Logits Loss (BCEWithLogitsLoss). The BCEWithLogitsLoss applies the sigmoidal function to turn logits, that are the raw outputs of the model's final linear layer, into a probability between 0 and 1, and then it computes binary loss for each label independently. As for the optimizer, we used AdamW, which is an improvement of the classic Adam optimizer with proper weight decay, applied separately after the gradient update, and it is also the recommended optimizer in the original BERT and RoBERTa papers. Regarding hyperparameters, we considered the standard ones: we trained for only 4 epochs because RoBERTa is a powerful and pre-trained model that can learn quickly from relatively small labeled datasets and we saved the updated weights only if the performance of the model improved on the evaluation set, in order to prevent overfitting. We set the learning rate to $2 * 10^{-5}$, as higher values can lead the model to diverge during fine-tuning. Then we evaluated whether the model improved using the function `f1_per_tag`, which computes the F1-score for each tag. We chose this metric instead of the loss function because the loss does not always reflect how well the model performs in making practical predictions, indeed it may overlook minority classes, especially in imbalanced datasets.

3.3 Evaluation

Before evaluating the model on the test set, we first evaluated it on the validation set to compute the precision-recall curve that shows how precision and recall change as we vary the threshold, and the best threshold is the one that maximizes the F1-score. Precision and recall are two complementary metrics that provide insight into the model's performance: precision measures how many of the predicted positives were actually correct, while recall indicates how many of the actual positives were correctly identified by the model. This curve helped to determine the optimal threshold values for each class, which were then used instead of the default threshold of 0.5. The selected thresholds, [0.84, 0.39, 0.46, 0.20, 0.47, 0.42] - respectively for *BadEnding*, *Conflict*, *Dialogue*, *Foreshadowing*, *MoralValue*, and *Twist* - reflect that some classes require higher confidence to be predicted as positive, while others can be classified with lower confidence. On the test set, we also evaluated the accuracy and the confusion matrices. Accuracy measures how often the model's predictions are correct, and is calculated as the ratio between the number of correct predictions over the total number of predictions. For all classes, we achieved an accuracy of over 90%, which is a good result. The confusion matrices show how many examples were correctly or incorrectly classified into each class. It is particularly useful because it highlights the types of errors the model makes and whether it tends to favor certain class above others.

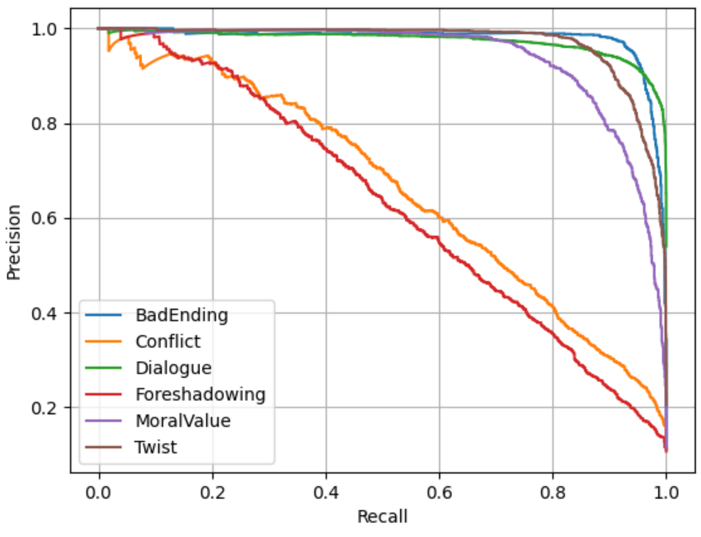


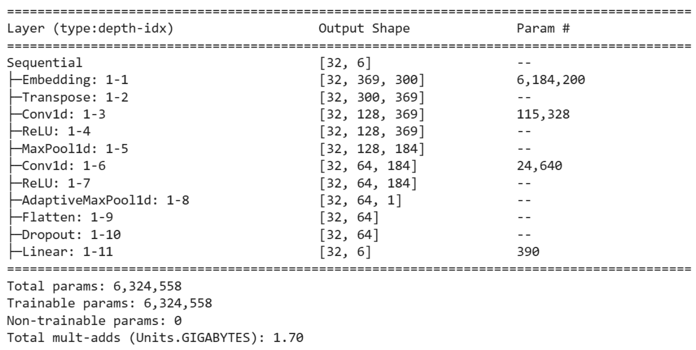
Figure 1: Precision-recall curve for RoBERTa.

4 Model 2: CNN

For Model B we implemented a network trained completely and only on the TinyStories dataset. We chose a convolutional neural network, an architecture that usually has a lower computational cost than transformers, indeed this efficiency allowed us to train the model on a significantly larger dataset. The key advantage of CNN is its capacity of learning hierarchical features: its convolutional filters can recognize simple patterns and then combine them to learn more complex aspects. Moreover, this kind of neural network has another fundamental property for our task, that is translation invariance. This means, in fact, that the model is able to identify a specific pattern no matter where it is in the text.

4.1 Model Architecture

The architecture of our CNN starts with an embedding layer, which maps token indices to real-valued vectors. This allows us to represent tokens with numerical values that capture semantic meaning and highlight different features of the words, making it easier for the convolutional layers to detect relevant patterns in the text. After this step, a transpose operation swaps the second and third dimensions of the output tensor from the embedding layer, as required by the following one-dimensional convolutional layer, which expects input in the format (batch size, channels, sequence length). The core of the model consists of two sequential convolutional blocks, where we set `padding="same"` to ensure that the output has the same shape as the input. The first convolutional layer captures local patterns, such as n-grams from the embedding map, and is followed by a ReLU activation function to introduce non-linearity. This is then followed by a max pooling layer, which reduces the dimensionality of the feature maps. The second convolutional block extracts higher-level or more abstract features from the output of the first block, and it is followed by another ReLU activation. Next, an adaptive max pooling layer reduces each feature map to its maximum value. The resulting tensor is then flattened into a one-dimensional vector, preparing it for the fully connected layer. A dropout layer is applied to randomly deactivate a portion of the neurons during training, helping to prevent overfitting. Finally, a linear layer processes the flattened vector and produces six output values (logits), each corresponding to one of the six target labels.



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Sequential                               [32, 6]                   --
├─Embedding: 1-1                         [32, 369, 300]            6,184,200
├─Transpose: 1-2                         [32, 300, 369]            --
├─Conv1d: 1-3                            [32, 128, 369]            115,328
├─ReLU: 1-4                              [32, 128, 369]            --
├─MaxPool1d: 1-5                         [32, 128, 184]            --
├─Conv1d: 1-6                            [32, 64, 184]             24,640
├─ReLU: 1-7                              [32, 64, 184]             --
├─AdaptiveMaxPool1d: 1-8                 [32, 64, 1]               --
├─Flatten: 1-9                           [32, 64]                  --
├─Dropout: 1-10                          [32, 64]                  --
├─Linear: 1-11                           [32, 6]                   390
==========================================================================================
Total params: 6,324,558
Trainable params: 6,324,558
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 1.70
```

Figure 2: CNN architecture.

4.2 Training

As specified in the RoBERTa Training paragraph, we chose BCEWithLogitsLoss as the loss function, since it is the most appropriate for our multi-label classification task. To overcome class imbalance we introduced positive class weights (`pos_weight`) as an input to the loss function. This encourages the model to give more importance to the under-represented classes and to learn better from fewer examples. According to PyTorch documentation these weights are defined as the number of positive samples over the negative ones. For optimization we used Adam, which is widely adopted for training neural networks. In the final part of the training phase, we built a grid search to find the best hyperparameters, including the learning rate and the architectural parameters of the CNN. Given the computational cost, we ran the grid search on a subset of 50,000 stories to keep training times manageable. It is commonly assumed that these hyperparameters should remain optimal even if we increase the size of the dataset. Therefore, we manually saved the best configuration from the grid search and re-trained only the final model on the initial dataset of 400,000 stories.

4.3 Evaluation

In order to select the best model configuration using the grid search, we used the macro F1-score to compare the different models. We chose this metric because it computes the F1-score for each tag separately and then takes their unweighted average, ensuring that all tags are given the same weight and importance regardless of their frequency. To provide a good evaluation of the model's performance, we computed several metrics for each of the six tags, as we did for model A. As we can see from the results, the model performed well on the test set: four out of six tags achieved an F1-score above 0.8, while the remaining two (*Conflict* and *Foreshadowing*) above 0.4. In terms of accuracy all the six tags are above 80%. From the precision and recall scores, we can observe that for tags like *Dialogue* or *BadEnding*, the model shows both high precision and high recall, indicating a robust predictive performance. For tags like *Conflict* and *Foreshadowing* the model achieved high recall (above 63%), which means that it is good at finding most of the stories that actually contain these tags, but the precision is low (around 30%), meaning the model often predicts *Conflict* or *Foreshadowing* even when those tags are not actually present. Finally, we computed the confusion matrices to gain a deeper understanding of the model's behavior.

5 Comparison

During the implementation, we observed that both models faced similar challenges that we tried to overcome in two different ways: by using the positive class weights for the CNN model and by optimizing the thresholds for the RoBERTa model. As shown by the table below, for both models the most difficult labels to predict were *Conflict* and *Foreshadowing*, both in terms of F1-score and accuracy.

These two labels are harder to detect for several reasons. First, they are more abstract then other tags, such as *Dialogue* or *Twist*, which often have explicit markers (e.g. quotation marks). Second, these labels appear less frequently in the training data (each one in about 10% of the stories), giving the model fewer examples to learn from. This lower representation contributes to reduce their performance, particularly in terms of F1-score. In contrast, *Dialogue* and *Twist* are both more represented and, as said before, have clear markers. Although *BadEnding* and *MoralValue* are present only in 10% of the stories each, they are easier to detect. This is likely because they often involve explicit, concrete narrative elements or keywords that highlight a negative resolution or a moral lesson. Together, those factors contribute to reach an higher F1-score for these tags.

Table 1: F1-score per tag for RoBERTa and CNN

Tag	RoBERTa	CNN
BadEnding	0.9391	0.8742
Conflict	0.5753	0.4271
Dialogue	0.9396	0.9160
Foreshadowing	0.5687	0.4395
MoralValue	0.8780	0.8150
Twist	0.9154	0.8654

Table 2: Accuracy per tag for RoBERTa and CNN

Tag	RoBERTa	CNN
BadEnding	98.94%	97.53%
Conflict	91.26%	82.40%
Dialogue	93.25%	90.67%
Foreshadowing	90.64%	84.52%
MoralValue	97.43%	95.50%
Twist	96.63%	94.67%

6 Conclusion

Overall RoBERTa performed better on this task. As we expected, it required a smaller amount of data and - despite the few epochs used for training- it achieved a better performance across all six tags. The most significant difference can be seen in the precision-recall metrics for the most challenging tags: *Conflict* and *Foreshadowing*. For these tags, RoBERTa reached a precision score of approximately 0.5, while the CNN only reached about 0.3. The recall, however, was high and approximately the same for both models, with a slight advantage for the CNN. This suggest that both models are good at identifying most of the stories that truly have these tags, but RoBERTa is significantly more precise than the CNN, making fewer false positive predictions. This difference in performance is probably due to the models' different architectures. RoBERTa, being a transformer-based model pre-trained on a very large dataset is more able to capture complex structure, and thanks to the self-attention mechanism is able to detect semantic links between distant words in the text. In contrast, the CNN tends to work on local groups of words, which is less effective for tags like *Conflict* and *Foreshadowing* that are often defined by subtle cues distributed throughout the story. This can be seen, for example, in the two confusion matrices for the *Foreshadowing* tag. While both models achieved a very similar recall, even slightly higher for the CNN model with 607 correct predictions compared to the 575 of RoBERTa, the CNN model produced 1,203 false positives, whereas RoBERTa reduced them by more than half, highlighting its superior precision.

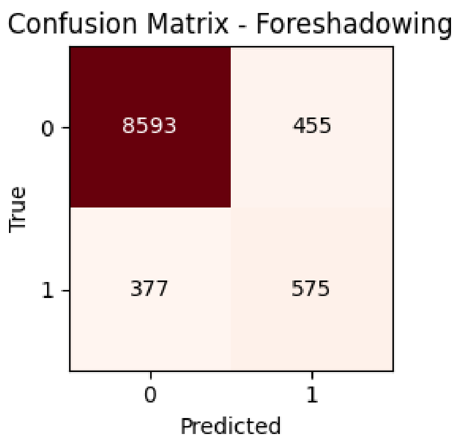


Figure 3: Confusion matrix of Foreshadowing tag for RoBERTa in the test set (10,000 stories).

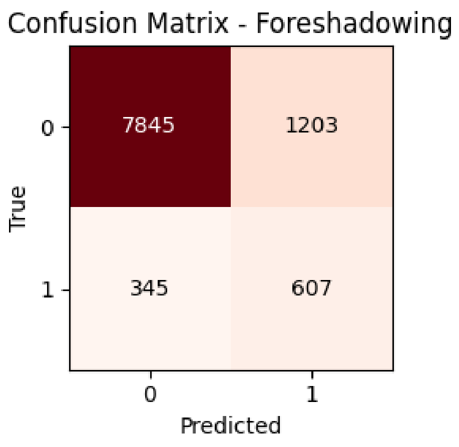


Figure 4: Confusion matrix of Foreshadowing tag for the CNN in the test set (10,000 stories).

Individual Contributions

This project was carried out in close collaboration among the three team members. We worked together for the majority of the time to discuss ideas, model choices and details. Each of us initially drafted one of the three scripts: Claudia prepared the training procedure for RoBERTa, Chiara drafted the training procedure for the CNN model, and Valentina wrote the initial version of the testing script. However, all scripts were reviewed, modified, and improved by the entire group.