# MultiClass Text Classification

Vermisoglou Konstantinos
*AEM: 2988*
*kvermisoglou@uth.gr*

Gkagkosis Nikolaos
*AEM: 3079*
*ngkagkosis@uth.gr*

Roumpos Ioannis
*AEM: 2980*
*iroumpos@uth.gr*

***Abstract*—Text classification is one of the fundamental tasks in the field of machine and deep learning. In this coding development project, we have developed a text classifier (for news classification) for categorizing relatively short texts into categories and subcategories. The objective of this project is to find a neural network architecture that achieves the best accuracy for a given dataset. We concluded on using transformers , since after testing it, it gave the best results among other architectures and provided the best generalization for both of the categories we had to classify.**

## I. Introduction

Text classification, also known as text categorization or document classification, is the process of automatically assigning predefined categories or labels to textual documents based on their content. It plays a crucial role in various natural language processing (NLP) tasks, such as sentiment analysis, spam detection, topic labeling, and content organization. The ability to effectively classify large volumes of text data has become increasingly important in today's digital age, where vast amounts of textual information are generated and shared across various online platforms.

In recent years, the advent of deep learning techniques, particularly neural networks, has revolutionized the field of text classification. Deep learning models can automatically learn hierarchical representations of text data, enabling them to effectively capture intricate semantic and syntactic features without the need for explicit feature engineering. Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and more recently, Transformer architectures, have shown remarkable performance in various text classification tasks.

In the case of this project transformer architecture was used. A transformer is a deep learning architecture that learns context and thus meaning by tracking relationships in sequential data like the words in this sentence. Transformer models apply an evolving set of mathematical techniques, called attention or self-attention, to detect subtle ways even distant data elements in a series influence and depend on each other.The objective of this project is using the aforementioned architecture achieve the best accuracy for the categories of the dataset we were given, taking into account the training time, the inference time of the network and the limitations of resources. In that context, we take advantage of pretrained models that will help our model learn faster and boost the performance. [1]

The remainder of the project is organized as follows: Section II refers to the methodology that we followed for implementing our architecture, Section III analyzes the experimentation on our model, Section IV give an overview of the results with proper visualization and Section V concludes this project.

## II. Methodology

### A. Description of the dataset

The project will utilize the MN-DS Multilabel News dataset that has over 10.000 news articles [2]. The dataset contains 11 columns that can be used as features and 2 columns that are going to be used as labels for our Neural Network training and testing. The first category level contains 17 unique categories while the second category level contains 109 unique categories.

### B. Preprocessing Steps

Before feeding the neural network with data, it is essential to distinguish what and in what form data should be processed in order to achieve the best performance. In our case, from the 11 features we chose only two of them( 'title','content') to train our model. The reason that we decided no to use all of them was because dimensional reduction can help address issues, such as reduce computational complexity and improve model performance. Our first step was to reprocess the 'title' and 'content' columns by concatenating them into a new 'text' column, and then removing any characters that are not letters, commas, or digits, as well as removing all numeric digits. The latter is very important and connects with next techniques which are tokenization and label encoding. Tokenization is the process of breaking down raw text into smaller units called tokens. These tokens can be words, subwords, or characters, depending on the granularity required for the task at hand. Label encoding is the process of converting categorical labels or classes into numerical values. Due to the fact that our classes are categorical this step in necessary for the assessment of our model.

### C. Description of the NN architecture

We implemented a text classification model using the DistilBERT architecture, a lightweight version of BERT (Bidirectional Encoder Representations from Transformers), which is pre-trained on a large corpus of text data [3]. The BertClassifier class defines our model architecture. It consists of a DistilBERT model initialized from the pre-trained 'distilbert-base-cased' checkpoint, followed by a dropout layer with a dropout probability of 0.5 to prevent overfitting. A linear layer maps the output of the BERT model to the number of unique labels in our dataset, which is obtained dynamically from

the label column of the input dataset. The rectified linear unit (ReLU) activation function is applied to introduce non-linearity. During the forward pass, input sequences are tokenized using the DistilBERT tokenizer, padding to a maximum length of 512 tokens and truncating if necessary. The encoded sequences are then fed into the DistilBERT model, and the output representation of the [CLS] token is extracted from the last hidden layer. This representation is passed through the dropout layer before being transformed by the linear layer and ReLU activation function to produce the final output logits for each class.

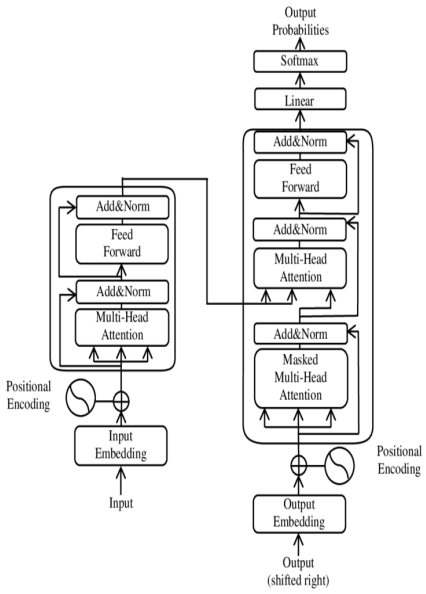


Fig. 1. Transformer Architecture

The DistilBERT pre-trained model is utilized as the backbone, which comprises tokenization functionality provided by the DistilBERT tokenizer. The DistilBERT model itself consists of a single embedding layer, responsible for converting tokens and their positions into vector representations. Transformer layers, comprising self-attention and feedforward layers, are then applied to these vector representations to extract increasingly complex linguistic information. Notably, since DistilBERT is an encoder-only model, it does not include the un-embedding layer, which converts final vector representations back into a probability distribution over tokens. Overall, this implementation adheres to the standard architecture of transformer-based models, leveraging the power of pre-trained DistilBERT embeddings for efficient text classification.

| Model | Parameters |
|---|---|
| DistilBert | 65190912 |
| Classifier | 13073 |

### *D. Hyperparameters and Optimization techniques*

**Optimizer**

The model uses Adam optimizer. The table that includes its components and their mathematical notation are shown down below.

| Components | Description |
|---|---|
| $v_{dW}$ | the exponentially weighted average of past gradients |
| $s_{dW}$ | the exponentially weighted average of past squares of gradients |
| $\beta 1$ | hyperparameter to be tuned |
| $\beta 2$ | hyperparameter to be tuned |
| $\frac{\partial \mathcal{J}}{\partial W}$ | cost gradient with respect to current layer |
| W | the weight matrix (parameter to be updated) |
| a | the learning rate |
| $\epsilon$ | very small value to avoid dividing by zero |

$$v_{dW} = \beta_1 v_{dW} + (1-\beta_1)\,\frac{\partial \mathcal{J}}{\partial W}$$

$$s_{dW} = \beta_2 s_{dW} + (1-\beta_2)\left(\frac{\partial \mathcal{J}}{\partial W}\right)^2$$

$$v_{dW}^{\text{corrected}} = \frac{v_{dW}}{1-(\beta_1)^t}$$

$$s_{dW}^{\text{corrected}} = \frac{s_{dW}}{1-(\beta_1)^t}$$

$$W = W - \alpha \frac{v_{dW}^{\text{corrected}}}{\sqrt{s_{dW}^{\text{corrected}}}+\varepsilon}$$

**Dropout**

Dropout is a regularization technique that randomly sets a fraction of input units to zero during training. This helps prevent overfitting by reducing the reliance on specific neurons and encourages the model to learn more robust features.

**Learning Rate**

We tested our model for different learning rates and the two that worked the best had values 1e-6 and 4e-6 (the one we chose for our final results). For smaller values of learning rate the model's performance was noticeably worse.

**Batch Size**

We tested our models for batch sizes equal to 2 and 8 and no major changes detected.

### *E. Splitting the dataset*

The dataset has been split into training, validation, and testing sets according to the 80/10/10 rule, where 80% of the data is allocated for training, 10% for validation, and 10% for testing. This approach ensures that the model is

trained on a sufficiently large portion of the data while still reserving separate subsets for evaluating performance and tuning hyperparameters. The training set is used to update the model parameters during training, while the validation set is employed to monitor model performance and prevent overfitting by adjusting hyperparameters. Finally, the testing set, which remains unseen during training and validation, serves as an unbiased evaluation of the model's generalization ability on new, unseen data.

## III. Experimentation

### A. Training Process

A batch of text input from the training set after being tokenized feed the Distilbert network . After getting through the 6 transformers encoder block it outputs contextualized embeddings for each token in the input sequence. For Distilbert model, these embeddings have a dimension of 768. The first token in the input sequence is a [CLS] token that represents the aggregate understanding of the entire sequence after passing through the transformer layers. Next, the [CLS] token is passed to the classifier which is a dropout layer for regularization and a linear layer with number of neurons the number of classes ,having as activation function ReLU. The predicted label is compared to the actual label using Cross-entropy as loss function and then starts backpropagation to update the weights of the network having Adam optimizer.

### B. Evaluation of metrics

**Loss Function**

Cross-entropy loss, also known as log loss or softmax loss, is a commonly used loss function in PyTorch for training classification models. It measures the difference between the predicted class probabilities and the true class labels.

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\left[y_k^{(i)}\log\hat{y}_k^{(i)} + \left(1-y_k^{(i)}\right)\log\left(1-\hat{y}_k^{(i)}\right)\right]$$

Ground-truth class labels $y_i$ and model predicted class labels $\hat{y}_i$.In PyTorch, the cross-entropy loss is implemented as the nn.CrossEntropyLoss class. This class combines the nn.LogSoftmax and nn.NLLLoss functions to compute the loss in a numerically stable way. The nn.LogSoftmax and nn.NLLLoss functions are building blocks used to implement the cross-entropy loss function in PyTorch.

**Dropout**

Dropout is a regularization technique that randomly sets a fraction of input units to zero during training. This helps prevent overfitting by reducing the reliance on specific neurons and encourages the model to learn more robust features.

## IV. Results

In the figures below we have the results after running our neural network architecture. The metrics we are comparing, as mentioned in the previous section is train accuracy train loss , validation accuracy and validation loss. We can observe that during training (number of epochs is increasing) the accuracy is increasing until we a threshold where it seems to stabilize. The difference between training accuracy and validation accuracy is not significant meaning we are handling the overfitting quite good. The test dataset produces the final accuracy result.

TABLE I
Accuracy Results

| Accuracy Level 1 | Accuracy Level 2 |
|---|---|
| 83.8 | 72.34 |

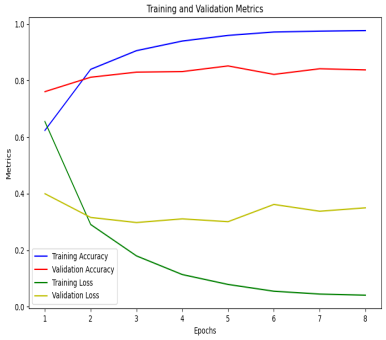


Fig. 2. Category level 1 results

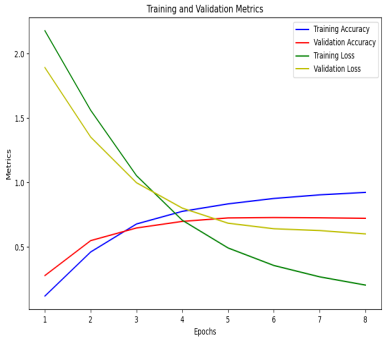


Fig. 3. Category level 2 results

The table below shows the training and inference time in respect the different batch size and category level. It is obvious that since the accuracy stays in the same level for different size we will choose the combination of hyperparameters that minize the time metrics. In deep learning, inference time is the amount of time it takes for a machine learning model to process new data and make a prediction.

TABLE II
TIME METRICS

| | Category 1 B2 | Category 2 B2 | Category 1 B8 | Category 2 B8 |
|---|---|---|---|---|
| Train Time | 64.4 min | 64.58 min | 48.23 min | 48.42 min |
| Inf Time* | 0.001 sec | 0.001 sec | 0.001 sec | 0.001 sec |

*Average Inference Time per Sample.

## V. CONCLUSION

In this project, we set out to explore text classification using neural networks, with a specific focus on employing transformer-based models, such as DistilBERT. Through extensive experimentation and analysis, several key findings have emerged, shedding light on the effectiveness of these models in handling text classification tasks.

We observed that DistilBERT, despite its smaller size compared to its counterparts, demonstrates remarkable performance in classifying news articles into distinct categories. By leveraging the transformer architecture, the model learns intricate patterns and semantic representations from the input text, enabling accurate predictions.

The results that our project produced are satisfying taking into account the resource limitations, primarily in terms of computational power and dataset size, influenced various aspects of our project. For instance, due to computational constraints, we were unable to explore larger transformer architectures or train the model for an extended number of epochs. Additionally, the size of our dataset posed challenges in achieving a comprehensive understanding of the model's performance across diverse categories and topics.

In conclusion, this project highlights the transformative impact of transformer-based models in text classification. As the field of deep learning continues to evolve, it is imperative to stay abreast of advancements in architecture and methodology. With continued innovation and experimentation, we can unlock new possibilities and propel the field of natural language processing towards greater heights.

## REFERENCES


[1] A. Vaswani, N. Shazeer , N. Parmar , J. Uszkoreit , L. Jones, A. N. Gomez , L. Kaiser , I. Polosukhin, "Attention Is All You Need", 2017

[2] Petukhova, A., Fachada N., "MN-DS: A Multilabeled News Dataset for News Articles Hierarchical Classification", Data 2023, 8, 74

[3] Sanh, Victor, et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." arXiv preprint arXiv:1910.01108 (2019).