Data Overview

The first step is to examine the dataset to check for missing values, duplicates, and other issues that may affect model training.

1.1 Loading the Dataset

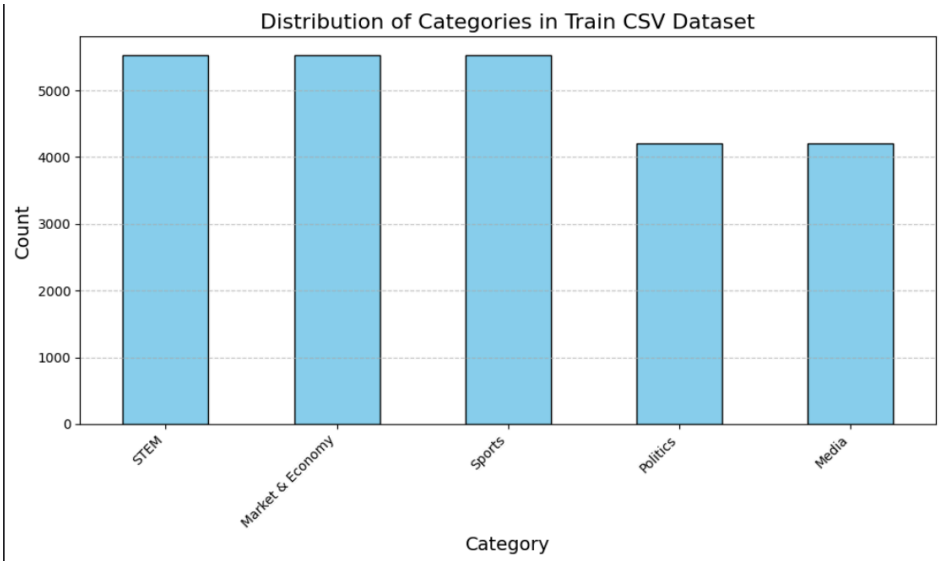
The dataset is loaded into pandas DataFrames as `train_csv` (training data) and `test_csv` (testing data). The columns of interest are:

- Discussion: The text of the discussion.
- Category: The label representing the category of the discussion.

	SampleID	Discussion	Category
0	1	Without sitting down and doing it manually, yo...	Sports
1	2	All your Search ends with this link.	STEM
2	3	No, the program you're using is made to be com...	STEM
3	4	Mike Woicik\n\nThe correct answer is: Mike Woi...	Sports
4	5	No, but not because of why you might think. Wh...	Politics

1.2 Category Distribution

We visualize the distribution of categories in the training dataset using a bar plot to make sure no category is undersampled.



1.3 Missing Values

We check for missing values in both the training and testing datasets. We observe that the Discussion column in the training data has 343 missing values, while there are no missing values in the Category or test_csv.

We fill the missing values in the Discussion column with empty strings.

```
Number of Duplicate Rows in Discussion (Before):  494
Number of Duplicate Rows in Discussion (After):  0
```

Preprocessing

2.1 Remove Duplicates

In this step, we remove any duplicate entries in the `Discussion` column from the **training dataset** only**.**

We first check for duplicates in the `Discussion` column. Initially, there were 494 duplicates.

We then drop the duplicates, leaving only unique entries in the `Discussion` column.

2.2 Drop Unnecessary Columns

We drop the `SampleID` column from the training dataset as it is not needed for model training and it might cause overfitting because it's filled with unique values that's irrelevant to training.

2.3 Text Preprocessing

Text preprocessing is crucial for cleaning and normalizing the text data before feeding it into a machine learning model. The steps involved in text preprocessing are as follows:

Contraction Expansion

We replace common contractions (e.g., "ain't" → "is not", "don't" → "do not") using a custom dictionary as there are some words in the common slang that were out of vocabulary that the model couldn't figure(yall → you all, yalldve → you all would have). This helped in making the text more uniform, and decreased the number of OOV words significantly.

Removing URLs, Email Addresses, and Mentions

We remove URLs, email addresses, and user mentions (e.g., `@username`) from the text to focus on the actual content.

Removing Extra Whitespaces

We remove any /n and extra spaces from the text.

Removing Non-Alphanumeric Characters

We remove any non-alphanumeric characters (except spaces) from the text to clean it further.

Normalizing Repeated Characters

We normalize words with repeated characters (e.g., "soooooo" becomes "so", “Heeeellllllooooo” becomes “Hello”).

Lemmatization

We use spaCy to lemmatize the text, converting words to their base form (e.g., "running" becomes "run").

Tokenization and Stopwords Removal

We remove stopwords (common words like and, the, etc.) from the text using NLTK's list of English stopwords. We also create a custom list of stopwords to remove certain words that were manually selected based on the **word cloud** (e.g., use, one) and a list to *not* remove (e.g., no, not and but) because they might correspond to the meaning.

2.4 Encoding Labels

We encode the categories (labels) into numerical values to prepare them for machine learning models.

Politics → 0

Sports → 1

Media → 2

Market & Economy → 3

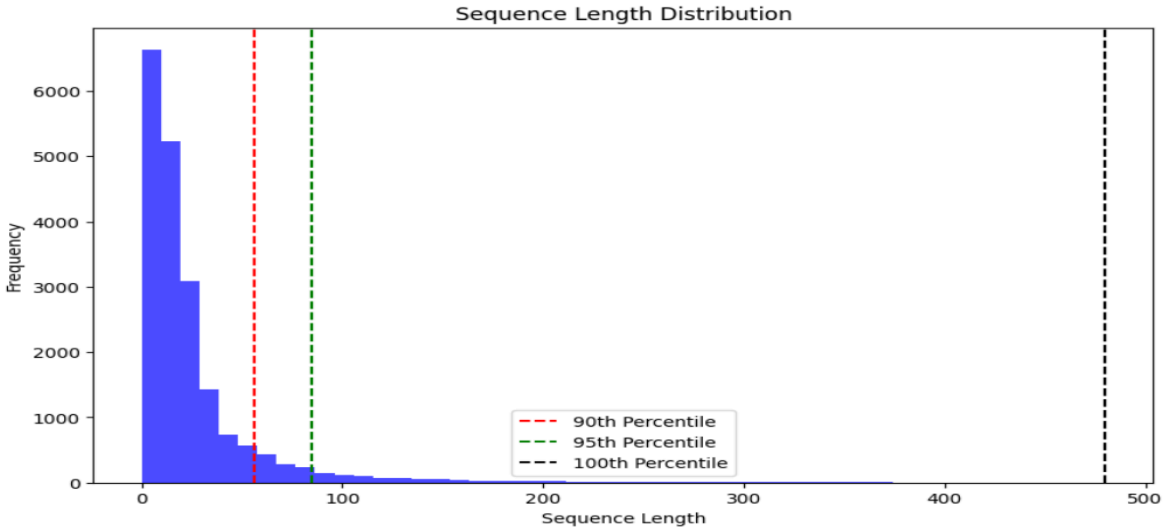
STEM → 4

Train-Test Split

Finally, we split the data into training and testing sets using an 80% - 20% split.

Tokenizing and Padding:

For the hyperparameters `MAX_SIZE` (max_words) and `MAX_SEQUENCE_LENGTH`, we employed various techniques to determine the optimal values, including grid search, visualization, and manual selection. The values `MAX_SIZE = 50000` and `MAX_SEQUENCE_LENGTH = 480` performed well across all models.




```
Chosen MAX_SEQUENCE_LENGTH: 480
```

The tokenizer will only consider the 50,000 most frequent words in the dataset (as specified by `MAX_SIZE`). After that, the text samples will be converted into sequences of integers. Next, padding will be applied to each sequence to ensure all sequences have the same length, defined by `MAX_SEQUENCE_LENGTH`. This padding step is important because it defines the input length, which is essential for feeding the data into the model.

Embeddings:

For the embedding, we used pretrained models and experimented with **GloVe**, **Word2Vec**, and **FastText**. **GloVe** performed the best among them. We set the `EMBEDDING_DIM` to 300 and used the embedding matrix in the models. Additionally, we set the `trainable` parameter to `True` so that the models could learn during training.

Models:

Simple RNN

Model Definition:

- We defined a Sequential model using Keras to build a simple RNN architecture for sequence classification.

Embedding Layer:

- We added an **Embedding** layer to transform word indices into dense vectors of a fixed size (128).
- The input dimension is set to `vocab_len` (the number of unique tokens), and the output dimension is set to 128 (embedding size).
- The input length is set to `MAX_SEQUENCE_LENGTH` to define the length of each input sequence.

Simple RNN Layer:

- We added a **SimpleRNN** layer with 64 units and ReLU activation.
- `return_sequences=False` ensures the output is a single vector representing the entire sequence.

Dropout Layer:

- We included a **Dropout** layer with a rate of 0.5 to prevent overfitting by randomly setting 50% of the input units to zero during training.

Dense Layers:

- A **Dense** layer with 32 units and ReLU activation is added to extract features from the RNN output.
- Another **Dense** layer with 5 units and **softmax activation** is added as the output layer for multi-class classification (5 classes).

Model Compilation:

- We compiled the model with the **Adam** optimizer and a learning rate of 0.001.
- The loss function used is **sparse categorical cross-entropy** for multi-class classification, and **accuracy** is used as the evaluation metric.

Early Stopping:

- **EarlyStopping** is used to prevent overfitting by halting training if validation accuracy doesn't improve for 3 consecutive epochs.
- `restore_best_weights=True` ensures that the model weights from the best epoch are restored after early stopping.

Model Training:

- The model is trained using train data with 40 epochs and a batch size of 32.
- Validation is performed using the test data, which is first processed through the `preprocess_test_data` function to transform it.

RNN using LSTM

Model Definition:

- A **Sequential model** is defined. This RNN architecture uses **LSTM** to handle sequence data and perform classification tasks.

Embedding Layer:

- The **Embedding layer** converts input word indices into dense vectors of fixed size. This layer uses pre-trained embeddings provided by `embedding_matrix`, and it is set as **trainable** (`trainable=True`).This is useful when using pre-trained embeddings, like GloVe or Word2Vec.
- `vocab_len` refers to the total number of unique words in the dataset (the vocabulary size), while `emb_dim` defines the size of the dense vector for each word.

LSTM Layer:

- The **LSTM (Long Short-Term Memory)** layer with 128 units is added to the model.
- `return_sequences=False` so that the LSTM layer will output a single vector, summarizing the entire sequence of words, instead of returning a sequence of vectors for each time step.

Dropout Layer:

- A **Dropout** layer with a dropout rate of 0.5 is added to regularize the model and prevent overfitting.

Dense Layer:

- The model includes a **Dense** layer with 5 units (one for each class in a multi-class classification task) and **softmax activation**.

Model Compilation:

- The model is compiled using the **Adam optimizer** with a specified learning rate.
- The **sparse categorical cross-entropy** loss function is used, which is appropriate for multi-class classification problems.
- **Accuracy** is selected as the evaluation metric, which measures the percentage of correct predictions.

Early Stopping:

- **EarlyStopping** is used to monitor the validation accuracy during training (`patience=3`).

- The `restore_best_weights=True` argument ensures that the model will return to the best weights.

Model Training:

- The model is trained using train data with 40 epochs and a batch size of 32.
- Validation is performed using the test data, which is first processed through the `preprocess_test_data` function to transform it.

CNN Architecture

Input Representation:

- Input text is tokenized into integer sequences, padded to a uniform length, and passed through an embedding layer, which converts integers into dense vectors capturing word semantics for CNN processing.

Embedding Layer:

- We added an **Embedding** layer to transform word indices into dense vectors of a fixed size (128).
- The input dimension is set to `vocab_len` (the number of unique tokens), and the output dimension is set to 128 (embedding size).
- The input length is set to `MAX_SEQUENCE_LENGTH` to define the length of each input sequence.
- Leverages pre-trained embedding (`GloVe`) initialized in the layer.
- The Weights is set to pretrained embedding weights

Convolutional Layer:

- We added a convolution layer with filter size equal 5, number of filters equal 128 and `RELU` activation function (for non-linear feature learning).
- Produces feature maps that highlight patterns across the sequence.

Global Max Pooling Layer:

- Reduces dimensionality by selecting the most significant feature from each filter.

Dropout Layer:

- We included a **Dropout** layer with a rate of `0.5` to prevent overfitting by randomly setting 50% of the input units to zero during training.

Dense Layers:

- A **Dense** layer with 64 units and ReLU activation is added to extract complex features from the CNN output.
- Another **Dense** layer with 5 units and **softmax activation** is added as the output layer for multi-class classification (5 classes).

Early Stopping:

- Stops training when validation loss stops improving for `5` consecutive epochs.
- `restore_best_weights=True` ensures that the model weights from the best epoch are restored after early stopping.

Model Compilation:

- Configures the model with **Adam** optimizer, **sparse categorical cross-entropy loss**,accuracy metric and learning rate equal (0.001).

Model Training:

- The model is trained using train data with 20 epochs and a batch size of 32.
- Validation is performed using the test data, which is first processed through the `preprocess_test_data` function to transform it.

Transformers

Positional Encoding:

- We created a positional encoding function to add position encoding values to the embedding layer.
- We used the embedding model to create the embedding layer and then added the positional encoding to the embedding values.

Transformer Block Loop:

- We created a loop to repeat the transformer block `n` times.

Multi-Head Attention:

- In each loop, the **MultiHeadAttention** layer takes the embedding values as keys, values, and queries.
- The output of the multi-head attention (`out1`) is then normalized after applying a dropout rate of 0.4.
- We set `num_heads` to `x` and looped through `n` blocks.

Add & Normalize:

- After attention, we normalized the output (`out1`) by adding it back to the previous embedding layer and applying layer normalization.

Feed Forward Network:

- We applied a feed-forward layer after normalization.
- The output (`ffn_output`) is added to the attention output (`out1`) and then normalized again.

Global Average Pooling & Output:

- We applied **GlobalAveragePooling1D** to the final output of the last transformer block.
- We added a **Dense** layer with **softmax activation** for classification, predicting the final output.

Model Compilation & Training:

- We compiled the model with the **Adam** optimizer, **sparse categorical cross entropy** loss, and accuracy metrics.
- We set up early stopping to monitor validation loss and restore the best weights if needed.

Accuracies:

Model	Training Accuracy	Validation Accuracy	Learning Rate	Batch Size	Epochs	Early Stopping Patience
Simple RNN	0.7044	0.7006	0.001	32	20	3
RNN using LSTM	0.7225	0.7234	0.001	32	20	3
CNN	0.7191	0.7233	0.001	32	20	5
Transformer	0.7126	0.7139	0.0001	16	30	3