Documentation Report: Model Architecture, Methodology, Evaluation Metrics, Challenges, Solutions

1. Model Architecture

The model architecture used in this project is based on DistilBERT, a smaller, faster, and lighter version of BERT (Bidirectional Encoder Representations from Transformers). DistilBERT retains 95% of BERT's performance while being 40% smaller and 60% faster. The specific model used is DistilBertForSequenceClassification, which is fine-tuned for text classification tasks.

- Base Model: `distilbert-base-uncased`

- Task: Sequence Classification (Text Classification)

- Output Layer: A fully connected layer with `num\_labels` outputs, where `num\_labels` corresponds to the number of unique classes in the dataset.

- Tokenizer: `DistilBertTokenizer` is used to preprocess the text data into tokenized inputs compatible with the model.

2. Methodology

The methodology for training the model involves the following steps:

1. Data Preparation:

- The dataset is loaded from a CSV file (`final\_labels.csv`).

- The dataset is preprocessed to ensure the required columns (`body` for text and `group` for labels) are present.

- Text labels are mapped to integer values for compatibility with the model.

- The dataset is split into training and validation sets (80% training, 20% validation).

2. Tokenization:

- The text data is tokenized using the `DistilBertTokenizer`. The tokenizer converts text into input IDs, attention masks, and token type IDs, which are required by the model.

- The tokenization process includes padding and truncation to ensure uniform input lengths.

3. Model Initialization:

- The `DistilBertForSequenceClassification` model is initialized with the number of labels equal to the number of unique classes in the dataset.

4. Training:

- The model is trained using the `Trainer` class from the Hugging Face `transformers` library.

- Training arguments are defined using `TrainingArguments`, including batch size, number of epochs, weight decay, and evaluation strategy.

5. Evaluation - The model is evaluated on the validation set using metrics such as accuracy, precision, recall, F1-score.

- The evaluation metrics are computed using a custom `compute\_metrics` function.

6. Saving the Model:

- After training, the model and tokenizer are saved to the directory `./subreddit\_classifier` for future use.

3. Evaluation Metrics\*\*

The following evaluation metrics are used to assess the model's performance:

1. Accuracy:

- Measures the proportion of correctly classified instances out of the total instances.

- Formula: `Accuracy = (TP + TN) / (TP + TN + FP + FN)`

2. Precision:

- Measures the proportion of true positive predictions out of all positive predictions.

- Formula: `Precision = TP / (TP + FP)`

3. Recall:

- Measures the proportion of true positive predictions out of all actual positives.

- Formula: `Recall = TP / (TP + FN)`

4. F1-Score:

- The harmonic mean of precision and recall, providing a balanced measure of the model's performance.

- Formula: `F1 = 2 \* (Precision \* Recall) / (Precision + Recall)`

5. Loss:

- The cross-entropy loss, which measures the difference between the predicted and actual labels.

These metrics are computed using the `precision\_recall\_fscore\_support` function from `sklearn.metrics` and are weighted to account for class imbalance.

4. Challenges Faced and Solutions Implemented

Challenge: Missing or Incorrect Column Names

- Issue: The dataset may have missing or incorrectly named columns (`body` for text and `group` for labels).

- Solution: The code checks for the presence of required columns and renames them to ensure consistency. If the columns are missing, an error is raised with a list of available columns.

2. Challenge: Handling NaN Values

- Issue: The dataset may contain rows with missing values, which can cause errors during training.

- Solution: Rows with NaN values are dropped using `df.dropna()`.

3. Challenge: Label Mapping

- Issue: Text labels need to be converted to integer values for the model to process them.

- Solution: A label mapping dictionary is created to map each unique text label to an integer.

4. Challenge: Tokenization and Padding

- Issue: Text inputs need to be tokenized and padded to a fixed length for the model.

- Solution: The `tokenize\_function` is used to tokenize the text and apply padding/truncation.

5. Challenge: Class Imbalance

- Issue: The dataset may have imbalanced classes, leading to biased model performance.

- Solution: Weighted metrics (precision, recall, F1-score) are used to account for class imbalance.

6. Challenge: Model Overfitting

- Issue: The model may overfit to the training data, especially with a small dataset.

- Solution: Regularization techniques such as weight decay (`weight\_decay=0.01`) are applied, and the model is evaluated on a validation set to monitor overfitting.

7. Challenge: Computational Resources

- Issue: Training large models like DistilBERT can be computationally expensive.

- Solution: The batch size is set to a manageable size (`per\_device\_train\_batch\_size=8`), and the model is trained for a limited number of epochs (`num\_train\_epochs=10`).

5. Results

After training the model for 10 epochs, the following evaluation metrics were obtained on the validation set:

- Accuracy: 59.67

- Precision: 59.92

- Recall: 59.67

- F1-Score: 59.74