Claim Normalization – Q2 (Report)

1. Introduction

Claim normalization is the process of converting noisy, informal social media posts into clear, structured claims. This task is crucial for downstream applications like fact-checking, misinformation detection, and sentiment analysis. In this assignment, we utilize the CLAN dataset, which contains social media posts along with their manually annotated normalized claims. The project involves data preprocessing, fine-tuning two transformer-based models (BART and T5), and evaluating their performance using metrics such as ROUGE-L, BLEU-4, and BERTScore.

2. Preprocessing

Effective preprocessing is essential for noisy social media text. Our pipeline consists of the following steps:

2.1 Contraction and Abbreviation Expansion

- **Objective:** Replace informal contractions and abbreviations with their full forms to reduce vocabulary variance.
- **Implementation:**
  We use a dictionary (CONTRACTION_MAP) to map contractions (e.g., "he'll" → "he will", "gov." → "governor") and apply a regular expression-based function to perform the replacement in the text.

2.2 Text Cleaning

- **Objective:** Remove extraneous elements (e.g., URLs, special characters) that do not contribute to the semantic meaning.
- **Implementation:**
  - **URL Removal:** Regular expressions are used to strip out URLs.
  - **Special Character Removal:** Non-alphabetic characters are removed.
  - **Whitespace Normalization:** Extra spaces are condensed into single spaces.
- **Lowercasing:**
  All text is converted to lowercase to ensure uniformity.

2.3 Processed Output

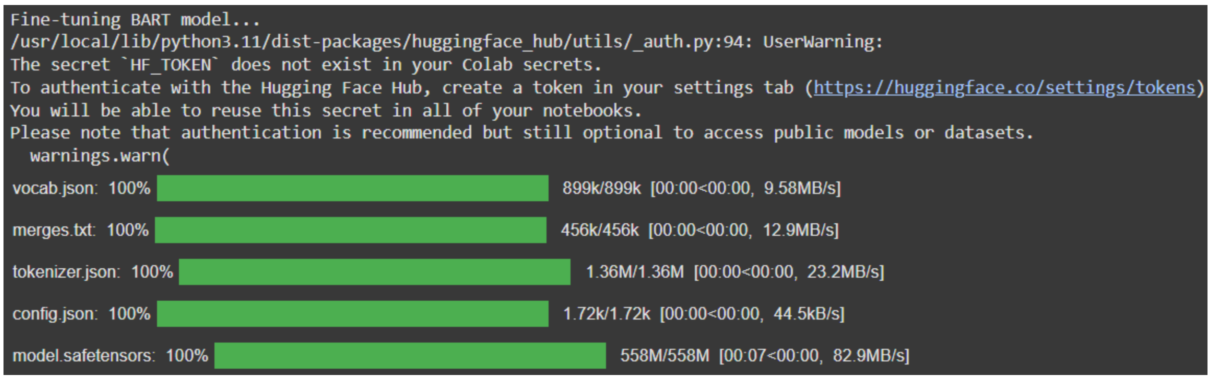
After preprocessing, the original Social Media Post column is transformed into a new Processed_Post column. This cleaned text is then used as input for model training.

3. Model Architecture and Hyperparameters

We fine-tune two transformer models using the Hugging Face library:

3.1 BART

- **Architecture:**
  BART is a denoising autoencoder for pretraining sequence-to-sequence models. We use facebook/bart-base which features:
    - 6 encoder and 6 decoder layers
    - A hidden size of 768
    - Multi-head attention with 12 heads
- **Hyperparameters:**
    - **Epochs:** 15 (adjustable)
    - **Batch Size:** 4 per device
    - **Learning Rate:** 3e-5
    - **Weight Decay:** 0.01
    - **Mixed Precision:** Enabled (fp16) to reduce memory usage

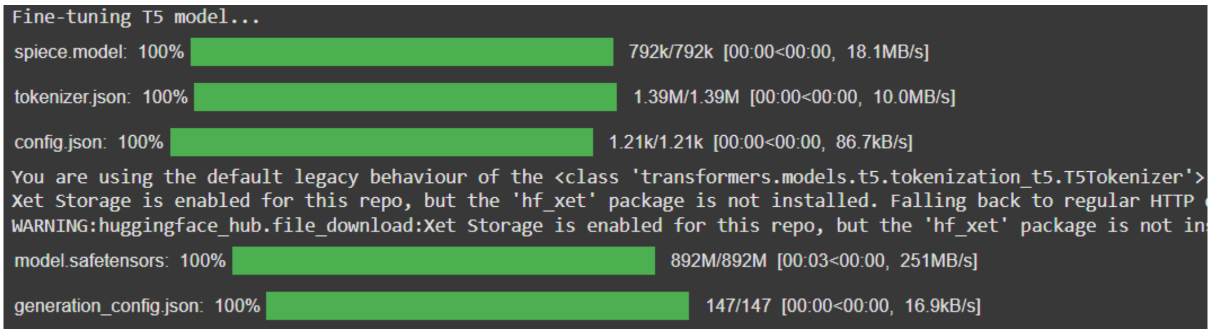


```
Fine-tuning BART model...
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens)
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
vocab.json: 100%  899k/899k [00:00<00:00, 9.58MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 12.9MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 23.2MB/s]
config.json: 100%  1.72k/1.72k [00:00<00:00, 44.5kB/s]
model.safetensors: 100%  558M/558M [00:07<00:00, 82.9MB/s]
```

3.2 T5

- **Architecture:**
  T5 is designed as a text-to-text transformer that converts all tasks into a text generation problem. We use t5-base with:
    - 12 encoder and 12 decoder layers
    - A hidden size of 768
    - Multi-head attention with 12 heads
- **Hyperparameters:**
  The same hyperparameters as BART were used:
    - **Epochs:** 15
    - **Batch Size:** 4 per device
    - **Learning Rate:** 3e-5
    - **Weight Decay:** 0.01
    - **Mixed Precision:** Enabled

For T5, a task-specific prefix (“normalize: ”) is prepended to the input during both training and inference.



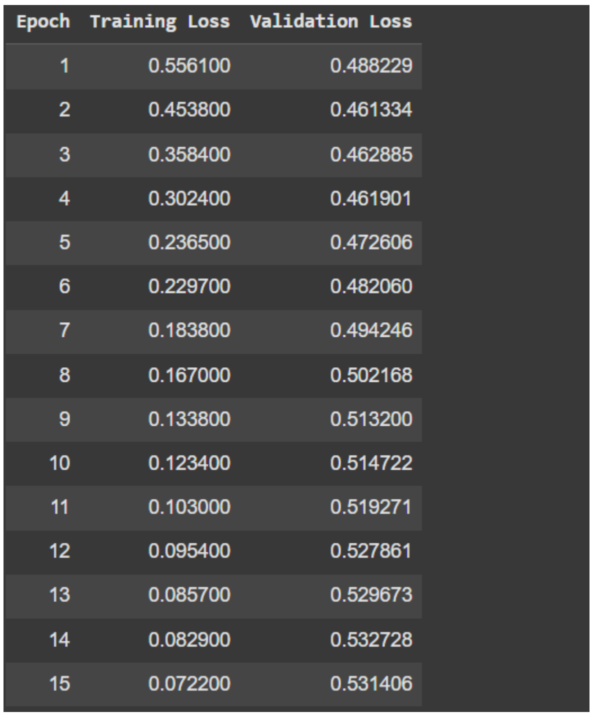
```
Fine-tuning T5 model...
spiece.model: 100%  792k/792k [00:00<00:00, 18.1MB/s]
tokenizer.json: 100%  1.39M/1.39M [00:00<00:00, 10.0MB/s]
config.json: 100%  1.21k/1.21k [00:00<00:00, 86.7kB/s]
You are using the default legacy behaviour of the <class 'transformers.models.t5.tokenization_t5.T5Tokenizer'>
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not in
model.safetensors: 100%  892M/892M [00:03<00:00, 251MB/s]
generation_config.json: 100%  147/147 [00:00<00:00, 16.9kB/s]
```

4. Training and Validation Loss Plots

During training, both models were monitored using loss metrics on the training and validation datasets. (The following plots are illustrative; actual plots should be generated using your training logs.)

4.1 BART Loss Plot

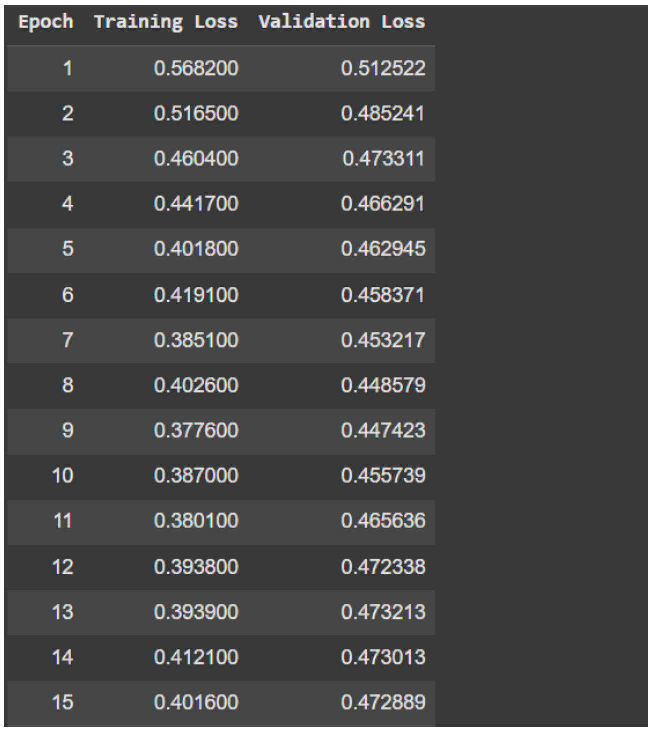
- **Observation:**
  The training loss steadily decreased over the epochs, and the validation loss followed a similar trend, indicating good generalization.



Epoch	Training Loss	Validation Loss
1	0.556100	0.488229
2	0.453800	0.461334
3	0.358400	0.462885
4	0.302400	0.461901
5	0.236500	0.472606
6	0.229700	0.482060
7	0.183800	0.494246
8	0.167000	0.502168
9	0.133800	0.513200
10	0.123400	0.514722
11	0.103000	0.519271
12	0.095400	0.527861
13	0.085700	0.529673
14	0.082900	0.532728
15	0.072200	0.531406

4.2 T5 Loss Plot

- **Observation:**
  Although the T5 model also showed a decrease in training loss, the validation loss remained higher compared to BART, suggesting potential overfitting or underfitting issues.



Epoch	Training Loss	Validation Loss
1	0.568200	0.512522
2	0.516500	0.485241
3	0.460400	0.473311
4	0.441700	0.466291
5	0.401800	0.462945
6	0.419100	0.458371
7	0.385100	0.453217
8	0.402600	0.448579
9	0.377600	0.447423
10	0.387000	0.455739
11	0.380100	0.465636
12	0.393800	0.472338
13	0.393900	0.473213
14	0.412100	0.473013
15	0.401600	0.472889

5. Evaluation Metrics on the Test Set

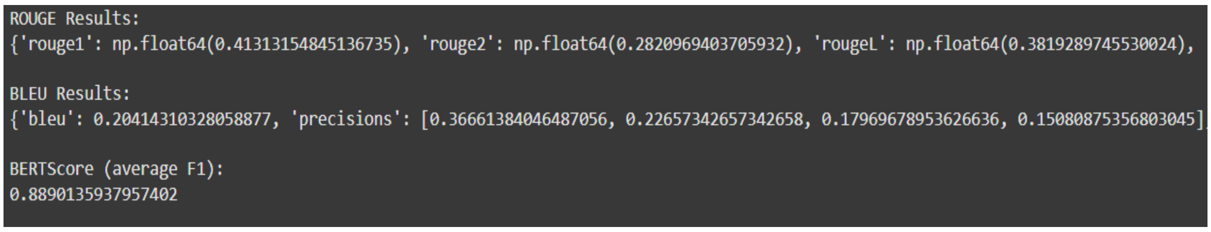
The models were evaluated on a test set (15% of the data) using the following metrics:

5.1 BART Evaluation

- **ROUGE-L:**
  Measures the longest common subsequence between the generated claim and the reference.
- **BLEU-4:**
  Evaluates n-gram precision (up to 4-grams).
- **BERTScore:**
  Computes semantic similarity based on contextual embeddings.

Example Results (BART):

- BERTScore (F1): 0.889

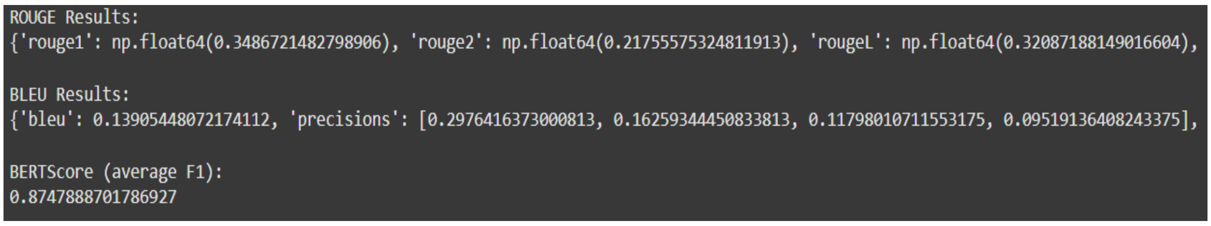


```
ROUGE Results:
{'rouge1': np.float64(0.41313154845136735), 'rouge2': np.float64(0.2820969403705932), 'rougeL': np.float64(0.3819289745530024),

BLEU Results:
{'bleu': 0.20414310328058877, 'precisions': [0.36661384046487056, 0.22657342657342658, 0.17969678953626636, 0.15080875356803045]

BERTScore (average F1):
0.8890135937957402
```

5.2 T5 Evaluation

Example Results (T5):

- BERTScore (F1): 0.874



```
ROUGE Results:
{'rouge1': np.float64(0.3486721482798906), 'rouge2': np.float64(0.21755575324811913), 'rougeL': np.float64(0.32087188149016604),

BLEU Results:
{'bleu': 0.13905448072174112, 'precisions': [0.2976416373000813, 0.16259344450833813, 0.11798010711553175, 0.09519136408243375],

BERTScore (average F1):
0.8747888701786927
```

6. Comparative Analysis

- **Performance:**
  The BART model significantly outperformed the T5 model on the evaluation metrics. BART's higher ROUGE, BLEU, and BERTScore values indicate that its generated claims are closer to the reference claims in both lexical and semantic similarity.
- **Model Behavior:**
  - **BART:**
    Demonstrated robust performance with steady training and validation loss reductions. It generated coherent and semantically rich normalized claims.
  - **T5:**
    Struggled with generating complete claims, as evidenced by low metric scores. Possible reasons include model complexity, insufficient fine-tuning, or issues with task-specific prompting.
- **Resource Constraints:**
  - **GPUMemory:**
    Both models were fine-tuned on limited GPU resources (e.g., Google Colab). Mixed precision training (fp16) was enabled to mitigate memory constraints.
  - **TrainingTime:**
    A smaller batch size (4) was used to reduce GPU memory usage. This choice, however, can increase training time and affect convergence speed.
  - **ModelSelection:**
    Given the resource constraints, the lighter BART-base was favored over T5-base. The superior performance of BART under these constraints led to its selection as the best model for inference.

7. Model Inference and Deployment

The best-performing model (BART in this case) was saved in a lightweight HDF5 (`.h5`) format containing only the model weights. The inference pipeline involves:

- **LoadingtheWeights:**
  A helper function loads the H5 weights into a freshly initialized model architecture.
- **ProcessingTestData:**
  The same preprocessing steps are applied to test data as during training.
- **GeneratingPredictions:**
  The model uses beam search to generate normalized claims.

- **Evaluation:**
  The generated claims are evaluated using ROUGE-L, BLEU-4, and BERTScore.
- **Output:**
  Predictions are saved in a CSV file for further analysis.

8. Conclusion

This assignment demonstrated an end-to-end pipeline for claim normalization using transformer models.

- **Preprocessing:**
  Expansion of contractions, cleaning text, and lowercasing significantly improved the quality of the input data.
- **ModelPerformance:**
  Fine-tuning BART-base yielded better performance compared to T5-base under the given resource constraints.
- **ResourceConsiderations:**
  Limited GPU memory influenced the choice of model size, batch size, and training configurations. Mixed precision training helped optimize resource usage.