Assignment: Multi Label Classification

Name: Subhadip Sardar

Roll No:MRM2024011

Dataset Understanding:

ECTHR-B Dataset (European Court of Human Rights – Binary) :

Background

- The **European Court of Human Rights (ECHR)** handles cases where individuals claim violations of rights guaranteed by the **European Convention on Human Rights**.
- In each case, the court considers **one or more Articles** (e.g., Article 3: prohibition of torture, Article 6: right to a fair trial).
- The dataset provides **facts of the case** (text paragraphs) and the corresponding **articles allegedly violated**.

Structure

Each entry contains:

- `text` → The factual description of the case (one or more paragraphs).
- `labels` → A **list of integers** corresponding to the violated articles.
- The dataset is **multi-label**: each case can involve **multiple Articles of the European Convention on Human Rights (ECHR)**.
- For **ECHR-B**, the dataset has **10 labels** corresponding to the 10 most frequent Articles.

Here are the labels (Articles):

1. **Article 2** – Right to life
2. **Article 3** – Prohibition of torture
3. **Article 5** – Right to liberty and security
4. **Article 6** – Right to a fair trial
5. **Article 8** – Right to respect for private and family life
6. **Article 9** – Freedom of thought, conscience, and religion
7. **Article 10** – Freedom of expression
8. **Article 11** – Freedom of assembly and association
9. **Article 14** – Prohibition of discrimination
10. **Article 1 of Protocol 1** – Protection of property

Dataset Size

Approximate splits:

- **Train:** 9000 cases
- **Validation:** 1000 cases
- **Test:** 1000 cases

Total: **~11,000 factual case descriptions**

Type of task :

Multi-label classification :

- A case can involve **multiple articles simultaneously**.
- Example:
  - Input text: `"The applicant was detained in inhuman conditions and denied the right to a fair trial."`
  - Labels: `[1, 3]` (Article 3: torture, Article 6: fair trial).

Approaches:

Explored the data in and understood it .
How many subsets are there , checked entries labels

1. Understanding Dataset Structure

- Checked dataset size (**train, validation, test splits**).
- Explored available columns:
  - **`text`** → case facts (long paragraphs).
  - **`labels_multihot`** → r **10 possible Articles**.
- Reason: This ensures we know what features are available for modeling.

2. **Text Analysis**

- **Approach:**
  - **Measured length of case texts (word count distribution).**
  - **Checked length to understand preprocessing needs .**
- **Reason: Legal texts can be very long; token limits in transformers may cut context. This analysis tells us what max sequence length to set.**

3. Checked if there was a missing test or missing labels.

Then preprocessed the data by doing followings :

1. **Lower casing**
2. **Html tag removall**
3. **Remove punctuation**
4. **Removed stopwords**
5. **Lemmatization**
6. **Stemming**

Choose legelbert(bert_base_uncased) LM:

🔎 Why LegalBERT is the best choice ?

I choose LegalBERT because it is pretrained on legal corpora, making it highly effective for capturing domain-specific terminology, multi-label dependencies, and long legal documents. General-purpose models like BERT or DistilBERT lack this specialization, leading to lower accuracy. Large LLMs (like GPT/Llama) are impractical for

fine-tuning in academic/industry settings due to size, cost, and lack of domain-specific training.

Challenges faced :

1.data’s internal structure understanding : The main challenge i faced the during preprocessing

The data and it’s structure as the `text` column is not of type string, but a NumPy array.
Each cell in `text` is actually a NumPy array of objects, not a single string.

Solution: Convert array to string first, then started to preprocessing

Original: **After Conversion:**

```
┌────────────────┐
┌────────────────────────────┐
│ text      │      │ text          │
├────────────────┤
├────────────────────────────┤
```

│ ['11.', 'At', │   │ '11. at the beginning ...'│

│ 'the', 'be...'│   │   │

└───────────────────┘

└─────────────────────────────────┘

2. **Challenge in Label Parsing :**

In the provided dataset, the labels column was stored in an unconventional format. Instead of being a valid Python list like `[8, 3]`, the labels were saved as `" [8 3]"` (with spaces inside brackets instead of commas).

- **Attempting to directly parse these strings using `ast.literal_eval()` resulted in errors.**
- **This prevented us from converting them into usable lists of integers / floats.**

Solution

To handle this challenge, we designed a safe label parser:

1. **Preprocessing: We replaced spaces with commas → `" [8 3]"` → `"[8,3]"`.**

2. **Safe Parsing: Used `ast.literal_eval` to reliably convert the string into a Python list.**

3. **Multi-Hot Encoding: Converted the list of labels (e.g., `[8,3]`) into a multi-hot encoded vector of size 10 (our total number of classes).**

   - **Example: `[8,3]` → `[0.,0.,0.,1.,0.,0.,0.,0.,1.,0.]`.**

Why Multi-Hot Encoding?

Since each case can belong to multiple legal articles simultaneously (multi-label classification problem), we needed a representation that:

- **Captures all labels at once.**
- **Works well with machine learning models that expect numerical input.**

Outcome

- **Successfully converted all label strings into usable vectors.**
- **Ensured robust error handling by logging warnings for out-of-range or invalid labels.**
- **Allowed downstream analysis like label frequency distribution and imbalance detection.**

Challenge 3: Float vs Int Conflicts During Training

When we started training the model, we faced errors in the dataloader / loss function because of mismatched label types.

🔴 The Problem

- **Our labels were stored as numpy float arrays (`dtype=float`) after preprocessing (e.g., `[0.0, 1.0, 0.0, ...]`).**
- **The model's loss function (`BCEWithLogitsLoss` for multi-label classification) expects torch.float32 tensors, not numpy arrays or int64.**

At one point, we also tried keeping labels as integers (`0/1 int`), which caused:

```
RuntimeError: Expected object of scalar type Float but got scalar type Long
```

- **This mismatch broke training repeatedly.**

🛠️ Solution

1. **Standardize label type:**
    - **Always convert multi-hot labels to float tensors before training.**
    - **Example:**

      ```
      labels = torch.tensor(labels_multihot, dtype=torch.float32)
      ```

Result: **Training ran without dtype conflicts.**

Fine-Tuning Approach

1. Approach

- **Pretrained Model Used: A transformer-based model (bert_base_uncased).**

- **Task: Multi-label classification on the ECHR-B dataset.**
- **Input: Preprocessed case text (lowercased, cleaned, punctuation removed).**
- **Output: Multi-hot vector representing Articles violated.**

Reasoning:

- **Transformers capture contextual semantics in legal text.**
- **Fine-tuning allows the model to adapt to domain-specific language in ECHR cases.**
- **Multi-label setup requires sigmoid + BCEWithLogitsLoss instead of softmax + cross-entropy.**

---

2. Training Setup

- **Dataset: Cleaned copies (`train_df_copy`, `valid_df_copy`, `test_df_copy`).**
- **Batch Size: 8 (train and validation).**
- **Epochs: 5 (observed validation loss to decide).**
- **Learning Rate: 2e-5.**
- **Metrics: Accuracy, Precision (Micro), Recall (Micro), F1 (Micro & Macro).**

---

3. Challenges Faced During Fine-Tuning

Challenge 1: Label Dtype Conflict

- **Problem: BCEWithLogitsLoss expects float tensors, but our labels were `int64`.**
- **Solution: Converted labels to `torch.float32` in the dataset.**

Challenge 2: Multi-Label Metric Calculation

- **Problem: Transformers Trainer expects single-label metrics by default.**
- **Solution: Wrote a custom `compute_metrics` function for multi-label metrics (F1, Precision, Recall).**
  -

Evaluation :

1. Approach

- **Objective**: Assess how well the fine-tuned model predicts multiple Articles per case.
- **Evaluation Dataset**: Preprocessed `test_df_copy` (texts + multi-hot labels).

- **Baseline Comparison**: Original pretrained model (without fine-tuning).

Procedure:

1. Load the test dataset into a Hugging Face `Dataset` object.
2. Tokenize texts using the same tokenizer as during fine-tuning.
3. Set dataset format to `torch` tensors: `input_ids`, `attention_mask`, `labels`.
4. Run the `Trainer.predict()` method on both **baseline** and **fine-tuned** models.
5. Compute multi-label metrics using a custom `compute_metrics()` function.

---

2. Metrics Used

- **Accuracy**: Fraction of correct predictions per sample (after thresholding logits > 0.5).

- **Precision Micro**: Total true positives divided by total predicted positives across all classes.
- **Recall Micro**: Total true positives divided by total actual positives across all classes.
- **F1 Micro**: Harmonic mean of Micro Precision & Recall → good for imbalanced classes.
- **F1 Macro**: Mean F1 across classes → treats all classes equally, highlights rare classes.

Reasoning for Choice:

- Multi-label classification requires micro metrics to account for multiple labels per sample.
- Macro F1 is important because some Articles appear rarely, and we want the model to learn them too.

4. Evaluation Results

Model	Accuracy	F1 Micro	F1 Macro	Precision Micro	Recall Micro
Fine-Tuned	**0.521**	0.708	0.635	0.786	0.644
Baseline	**0.003**	0.33	0.124	0.244	0.505