THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Final Group Project Report

Team 3: Akhil Bharadwaj, Brunda Mariswamy & Chirag Lakhanpal
The George Washington University
DATS 6312: Natural Language Processing
Professor Amir Jafari

Table of Contents

Introduction

Legal documents are very tough to interpret as they are very long and have a lot of important information. Humans find it quite challenging to quickly go through legal documents and understand key aspects without missing vital information. Most legal documents have a lot of information that is more relevant such as dates, deadlines, names of people, etc. Attorneys, judges, lawyers, and others in the justice system are constantly surrounded by large amounts of the legal text, which can be difficult to manage across many cases. They face the problem of organizing briefs, judgments, and acts.

Due to the huge amount of legal information available on the internet, as well as other sources, the research community needs to do more extensive research on the area of legal text processing, which can help us make sense of the vast amount of available data. This information growth has compelled the requirement to develop systems that can help legal professionals, as well as ordinary citizens, get relevant legal information with very little effort.

We have worked on different tasks such as Text summarization (both extractive and abstractive), Sentiment analysis to Predict the Judgment, and Text generation of legal documents.

Dataset

Text Summarization: - This dataset consists of 7,030 train examples and 100 test examples of text articles consisting of legal court proceedings in India along with their corresponding summaries.

Predicting Legal judgment -The Indian Legal Documents Corpus found in GitHub repo for Court Judgment (ILDC) family of datasets contains different types of training data. Among them, the ILDC-multi dataset consists of approx. 35k Supreme Court of India case documents each having multiple claims/appeals; a single label is assigned for each document – 'accept' if at least one appeal in that case document is accepted, 'reject' otherwise. For our current experiment, we considered the ILDC-multi dataset. The dataset has 3 columns text, label, and split. Around 33,299 number of legal case documents were used for Training and 1517 for Testing.

Document Generation: For training the GPT-2 model in legal text generation, this project utilized a carefully selected subset of the albertvillanova/legal_contracts dataset. From the original 650,833 records (33 GB), a 10% subset (65,083 records, 2.87 GB) was chosen for the main training phase, balancing data diversity with computational efficiency. Additionally, a 1% subset (6,508 records, 260 MB) was used specifically for hyperparameter tuning, optimizing model performance without excessive computational Demand.

Task	Source	Sample Size		Dataset columns
		Train	Test	
Text Summarization	Hugging Face Library	7,030	100	Text, Summary
Classification	GitHub	33,299	1517	Text, Label, Split
Text Generation	Hugging Face Library	65083		Text

Table1 Dataset description for NLP tasks

Natural Language Models

Bidirectional Encoder Representations from Transformers (BERT)

BERT is a natural language processing (NLP) model developed by Google. It was introduced in a research paper titled "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," published in 2018.

Unlike previous NLP models that processed language data in a left-to-right or right-to-left manner, BERT considers the entire context of a word by looking at the words to the left and right of it.

The bidirectional training allows BERT to capture complex relationships and dependencies within a sentence, making it highly effective for a wide range of NLP tasks, including sequence classification etc.Researchers pre-train BERT on a large corpus of text data, and the pre-trained model can then be fine-tuned for specific tasks with smaller datasets. Below Is the architecture of BERT for text classification.

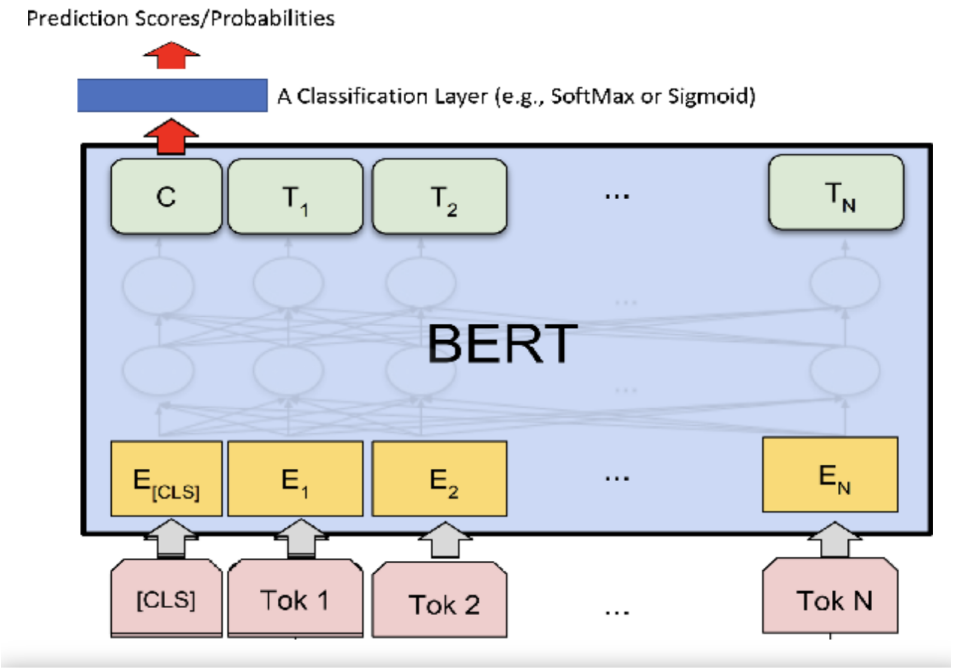

Figure 1:Bert base model architecture

The sequence classification is built on top of the general BERT architecture and is composed of:

Input Representation:

- Tokenization: The input text is tokenized into subwords or words, and special tokens such as [CLS] (classification) and [SEP] (separator) are added.

Embedding Layer:

- Word Embeddings: Represents each token in the input sequence with a 768-dimensional vector.
- Positional Embeddings: Encodes the position of each token in the sequence.

Transformer Encoder:

- Composed of multiple layers typically 12 layers for the base model.Each layer includes a self-attention mechanism, intermediate feedforward layer, and output layer

Pooler Layer:

- Applies pooling over the output of the last layer to obtain a fixed-size representation for the entire sequence.A linear layer and Tanh activation are used for this pooling.

Additional Layers:

- Linear Classifier: A linear layer that maps the pooled representation to the output classes. For sequence classification output to a range between 0 and 1. The purpose of this layer is to predict the probability of two output features. Predicting Judgment: "Accepted" or "Rejected

Generative Pre-trained Transformers (GPT2)

The GPT - 2 Model was chosen to perform the text generation task. GPT-2, which is built on a transformer model, transforms the way computers comprehend and produce human language. It functions by employing layers of transformer decoders, each of which adds to the model's capacity to comprehend context and produce
logical answers. With multiple variants spanning from 117 million to 1.5 billion parameters, GPT-2 has exceptional text creation capability.

Working and Architecture

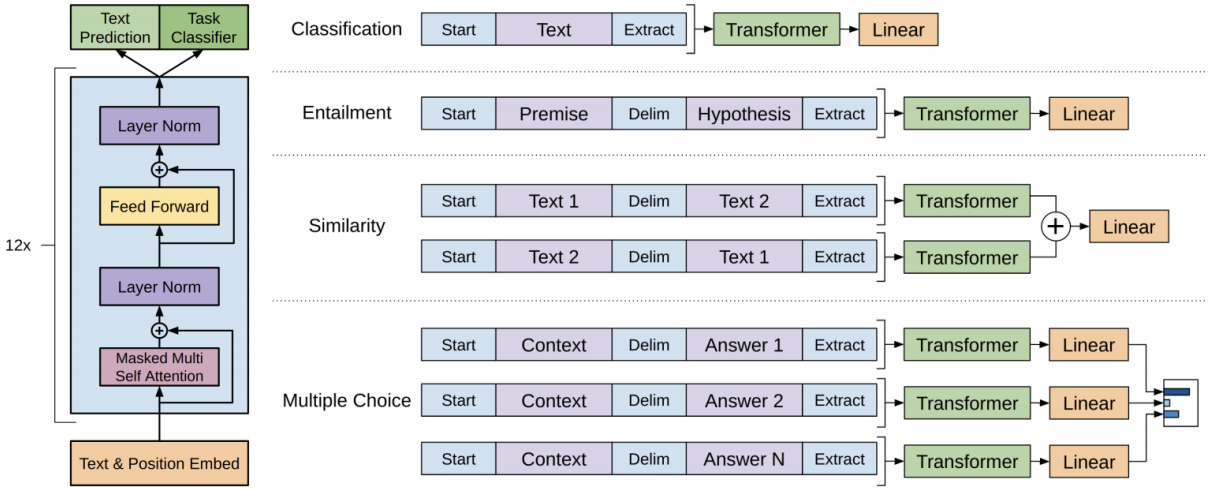

Figure 2:GPT2 model architecture

The core of GPT-2 lies in its multi-layered structure, comprising multiple decoder layers. Each layer employs a multi-head self-attention mechanism, enabling the model to weigh different parts of the input text differently, thereby understanding context more effectively. Positional encoding in GPT-2 adds the dimension of word order, further refining its language comprehension.

For tasks involving natural language processing, GPT is an architecture and training approach based on transformers. There are two steps in the training process. Initially, the unlabeled data is used to learn the basic parameters of a neural network model using a language modeling objective. These parameters are then modified using the associated supervised goal to fit a target task.

Pre-training with Extracted Gap-sentences for Abstractive Summarization (PEGASUS)

- Pegasus has an encoder-decoder architecture that was designed specifically for the task of text summarization.
- This model makes use of two different masks strategies:- one is masking 15% of input tokens similar to BERT, other is Gap Sentence Generation (GSG) which is creating pseudo summaries out of input text and using it as labels.
- In GSG, sentences are selected from documents and concatenated into a pseudo summary.
- These sentences are either selected randomly, lead (first m sentences from the document) or principal (top m scored sentences according to importance). For principal, importance is calculated by computing the ROUGE1-F1 score between the sentence and the rest of the document [1].

Here are some key features of PEGASUS:

- It has a self-supervised pre-training objective called GSG involving generating summary-like text from an input document.
- Through GSG, PEGASUS learns to recognize which sentences logically follow each other and how they contribute to the overall meaning of the text.
- This model is evaluated for 12 downstream summarization tasks like news, science, stories, instructions, emails, patents and legislative bills.

Architecture

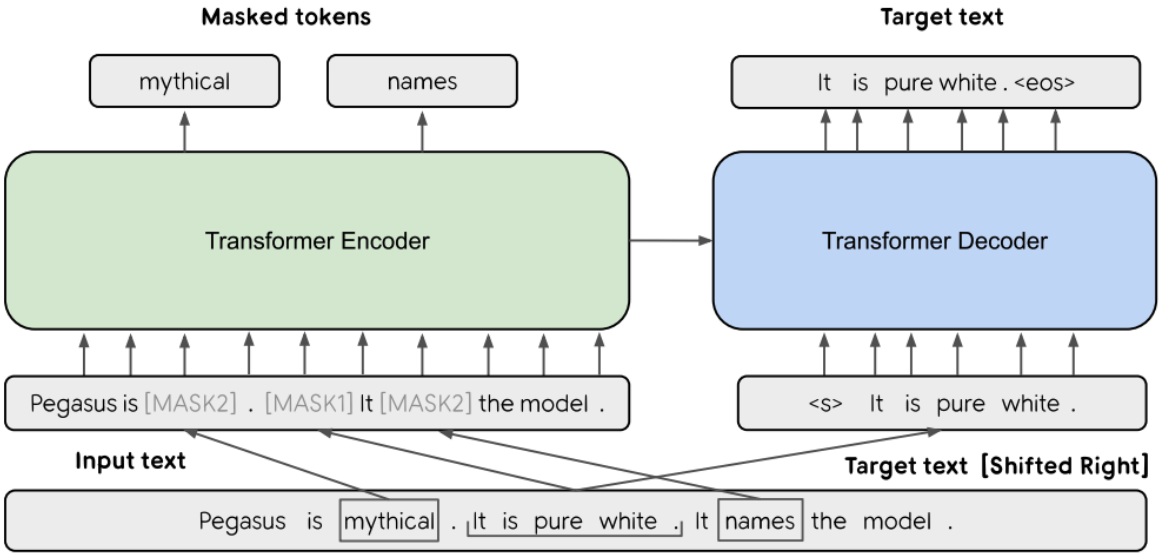

Figure 3:PEGASUS base model architecture

Data Preparation

Glimpse of Raw Data

Figure 4:Image of raw data content

Data Preprocessing

- Dataset Processing: Normalization of text from the albertvillanova/legal_contracts dataset was a vital step in the preprocessing procedure. To ensure text quality and uniformity, non-ASCII characters were removed throughout this procedure. To guarantee consistency, the text was transformed to lowercase.

- Regular Expression Usage: Regular expressions were extensively used for cleaning the data, including.
    - Replacing sequences of dashes with spaces
    - Removing extraneous newline, carriage return, and tab characters
    - Stripping excessive whitespace.
    - Spell-checking the words. Due to computation limitations, this could not work.
    - NER on names, persons, and organizations. Again, due to the sheer amount of the data, it got computationally expensive.

Model Initialization and Training

BertForSequenceClassifcation

- The model was fine-tuned using PyTorch framework with below Hyperparameter values.

- Pretrained Model Used:"law-ai/CustomInLawBERT'' from hugging face hub,the model specifically designed for the Indian Legal domain. It features a custom tokenizer with a vocabulary tailored to legal language. The model's architecture mirrors that of the popular "bert-base-uncased" model, encompassing 12 hidden layers, a hidden dimensionality of 768,12 attention heads, and approximately 110 million parameters.

- Data Split:The dataset as column which identifies the data for train,test.Hence there are 33,299 number of legal documents for training and 1,517 for testing.

- Tokenization:The tokenization process involves breaking down input text into individual tokens using a BERT tokenizer. To adhere to BERT's maximum input token limit of 512, the code retains the last 510 tokens of each sequence. These tokenized sequences are then encoded into input IDs, and the function returns padded input IDs and lengths. This preparation is crucial for BERT-based sequence classification.

- Due to memory limitations, a minibatch size of 6 was chosen. Minibatch size represents the number of data samples processed together in each iteration during training, and a smaller size like 6 is selected to accommodate memory constraints.

- For optimizing the model's parameters during training, the Adam optimizer is employed with a learning rate of 2e-5. Various optimization techniques are implemented to address overfitting, including weight decay and gradient clipping. Weight decay helps prevent the model from becoming too specialized to the training data, while gradient clipping limits the size of the gradients during backpropagation, preventing potential exploding gradient issues.

- A learning rate scheduler is connected to the learning rate, dynamically adjusting it during training. In this case, the learning rate is adjusted every 3 epochs, aiding in the convergence of the model over the training iterations.

- Performance metrics: The code calculates accuracy and confusion matrix along with macro- and micro-precision, recall, and F1 score.

- Model saving: The model is saved after the training process,which can be later used for inference and can recreate the same model for later.

- During inference, a detection threshold was set at 0.5 to predict class 0 or 1.If threshold >0.5 then its class 1 else 0. The model is available in hugging face hub https://huggingface.co/brundamariswamy/Indian-Custom-Bert .

Generative Pre-trained Transformers (GPT2)

- Data Preparation and Preprocessing

    - The experiment utilizes a custom dataset, specified through the --dataset argument. This dataset is loaded from disk and kept in memory for efficient processing. The data undergoes preprocessing, which includes converting text to lowercase, removing non-ASCII characters, extra characters, whitespace, and numbers. The tokenizer from GPT-2's library is then used to tokenize this processed text.

- Model Initialization and Training

    - The GPT-2 model (GPT2LMHeadModel) is initialized with pre-trained weights from 'gpt2'. A PyTorch DataLoader is employed for handling the training and validation datasets, ensuring efficient batch processing. The model is trained on a specified device (GPU if available) using the AdamW optimizer with a learning rate of 5e-5.

- Training Process

    - For each epoch in the training phase, the model processes the input data, calculates the loss, and updates the weights using backpropagation. Gradient clipping is applied to prevent exploding gradients. The model's performance is evaluated on the validation set, and both training and validation losses are recorded.

- Performance Metrics

    - Model performance is judged based on loss, perplexity, and ROUGE scores. Perplexity measures how well the probability model predicts a sample, with lower values indicating better performance. ROUGE scores are used to assess the quality of the generated text against the reference text.

- Hyperparameter Exploration

    - Epoch Configuration: For the smaller dataset, epochs were set at 1, 10, and 50. For the larger dataset, 25 epochs were used, which took around ~45 mins per epoch, balancing depth of learning with computational efficiency.
    - Batch Size Variation: Two different batch sizes, 8 and 16, were experimented with. We tried with bigger batch sizes starting from 128 and reducing it to 32 in the power of 2, however I encountered “CUDA out of memory” error.
    - Optimizer Selection: For the training used two different optimizers- AdamW and SGD (Stochastic Gradient Descent). AdamW.
    - Learning Rate Experimentation: Multiple learning rates were tested: 5e-5, 1e-5, 5e-4, 5e-3, and 5e-2. This range allowed for observing how the model responded to both subtle and significant changes in learning rate, influencing the speed and stability of the learning process.
    - Gradient Clipping: To address the potential issue of exploding gradients, gradient clipping was incorporated. This technique involved capping the gradients during backpropagation to a predefined range, ensuring they did not exceed manageable levels.

- Overfitting and Extrapolation Detection

    - Dataset Split: The dataset is split into training and validation sets, with the validation set used to monitor the model's performance on unseen data.
    - Regularization: The AdamW optimizer inherently provides a form of regularization.
    - Validation Loss Monitoring: Regular monitoring of validation loss ensures early detection of overfitting.

PEGASUS

Fine-tuning the model

Split the dataset into train, validation and test sets. For fine-tuning the model on the dataset consisting of Indian legal documents, 1000 samples were used for training, 250 samples for validation and 250 samples for test.

Arguments

- Total number of training epochs: 5 (num_train_epochs)
- Batch size per device during training and evaluation: 1 (per_device_train_batch_size and per_device_eval_batch_size)
- Number of updates steps before checkpoint saves: 5 (save_steps)
- Strength of weight decay: 0.01 (weight_decay)
- Number of warmup steps for learning rate scheduler: 5 (warmup_steps)

The fine-tuning procedure

- Create the train, validation and test splits.
- Filter out the rows from the dataset that have null values in the labels.
- Generate the tokens from the input data text and labels using PegasusTokenizer.
- Load the base model from the hugging face hub.
- Use the model, tokenizer, train and validation set for hugging face trainer API along with training arguments mentioned above.
- Save the model locally or push it to your hugging face hub
- Evaluate the model on the test set based on the ROUGE scores.

Results and Summary

Text Generation with Fine tuned GPT2 Model

Results on Small dataset (Records - 6508, Size: 260 MB)

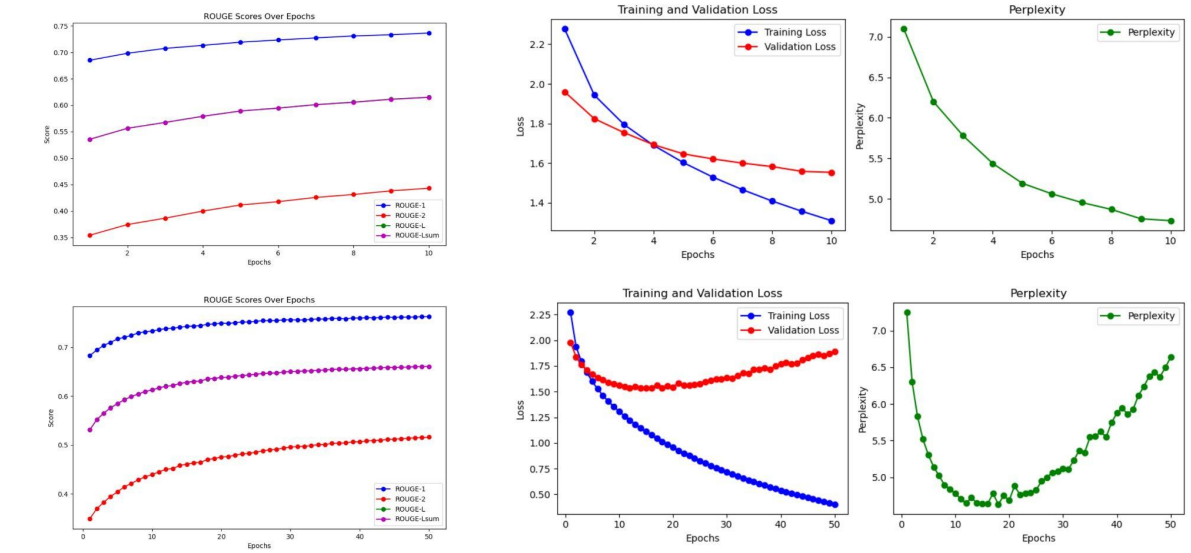

Figure 5:Visualization of of ROUGE,Training and Validation loss,Perplexity on small datasets

Results on Large dataset (Records - 65083, Size: 2.87 GB)

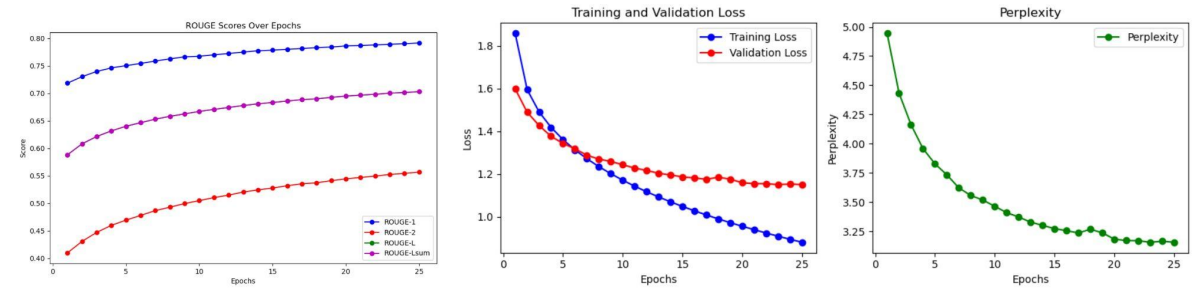

Figure 6:Visualization of of ROUGE,Training and Validation loss,Perplexity on large datasets

The figures displayed represent the results of training a text generation model across different epochs, as evidenced by ROUGE scores, training and validation losses, and perplexity metrics.

ROUGE Scores

- ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scores are a set of metrics for evaluating automatic summarization and machine translation software. They compare the overlap of n-grams, word sequences, and word pairs between the computer-generated output and a set of reference texts. Higher scores indicate better performance, with ROUGE-1, ROUGE-2, and ROUGE-L representing the overlap of unigrams, bigrams, and the longest common subsequence, respectively.
- Top Graphs: The top left graph illustrates an improvement in ROUGE scores across all three metrics over 10 epochs. The consistent upward trend across ROUGE-1, ROUGE-2, and ROUGE-L sum indicates the model's increasing proficiency in generating text closely matching the reference texts in terms of content and structure.

Training and Validation Loss

- Loss is a measure of how far the model's predictions are from the actual outcomes. Lower loss values correspond to better model performance.
- Top Middle Graphs: The training loss (blue) and validation loss (red) both show a downward trend over 10 epochs, suggesting that the model is learning effectively and generalizing well to unseen data.

Perplexity

- Perplexity is a measurement of how well a probability distribution or probability model predicts a sample. In the context of language models, lower perplexity indicates a better predictive performance.
- Top Right Graph: A decreasing perplexity trend over 10 epochs is observed, implying the model's improving ability to predict the next word in the sequence.

Observations from Extended Epoch Training

- When extending the training to 50 epochs, the graphs show distinct trends. Bottom Graphs: The ROUGE scores continue to improve, although they begin to plateau, indicating that the model might be approaching its optimal performance. The training and validation loss graphs depict a notable divergence after around 20 epochs, potentially suggesting overfitting where the model learns the training data too well and may not perform as effectively on new, unseen data. The perplexity initially decreases, reflecting improved model predictions, but then increases dramatically, further implying that the model's performance on the validation set is worsening, possibly due to overfitting.

Best model Parameters

Parameter	Best Value	Experimental Values
Learning rate	5e-5	5e-5, 1e-5, 5e-4, 5e-3, and 5e-2
Optimizer	AdamW	AdamW and SGD
Batch Size	8	8, and 16 (Working)
Momentum (For SGD)	0.9	0.9
Epochs	10	1,10,25,50
Number of Workers (Data Loading)	8	4, 8, and 16
Dataset Size	-	10% (65083 records) and 1% (6508 records)

Inference on custom fine-tuned Model

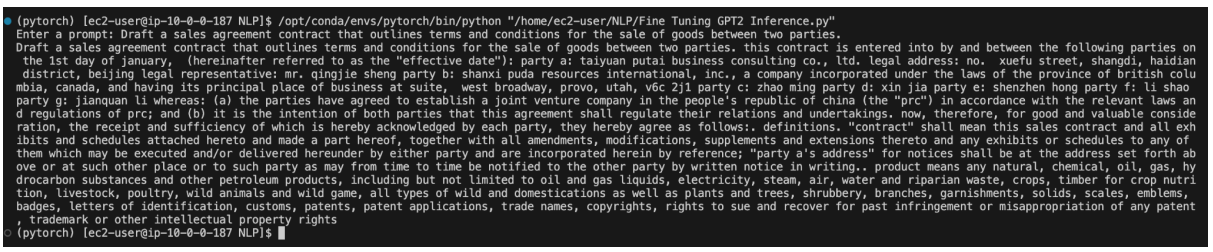

Figure 7:Inferencing results of Text Generation

Bert Sequence Classification with law-ai/CustomInLawBERT fine-tuned model

Accuracy

- Training loss – 0.38 Accuracy – 0.72

This value (0.38) represents the average training loss over the entire training dataset for a particular epoch. Training loss is a measure of how well the model is learning from the training data.he accuracy value (0.72) represents the proportion of correctly classified instances in the validation dataset. Specifically, it indicates that 72% of the instances in the validation set were classified correctly by the model.This is for the final epoch which had highest accuracy with minimal training loss.

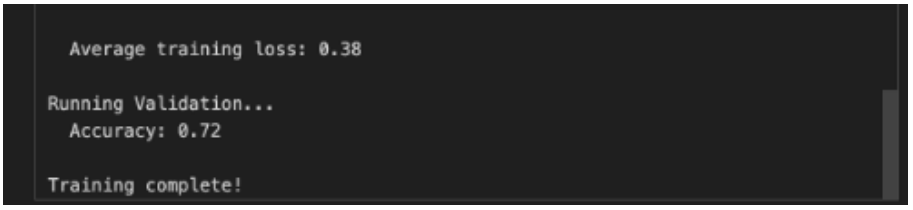

Figure 8 :Accuracy Score for Train dataset

Performance evaluation test data.

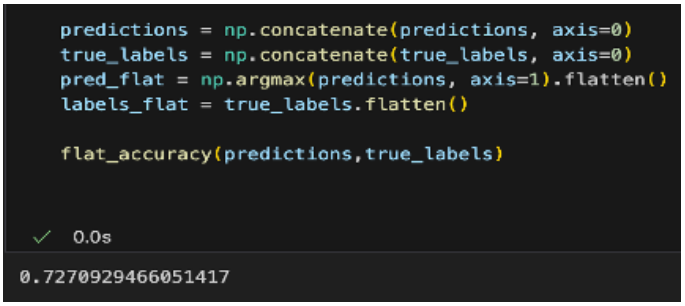

Figure 9:Accuracy on test set

The test set consisted of 1,517 legal documents, the model achieved an accuracy of approximately 72.70%. This implies that the model made correct predictions for the class labels in nearly 72.7% of instances.

Confusion Metrics

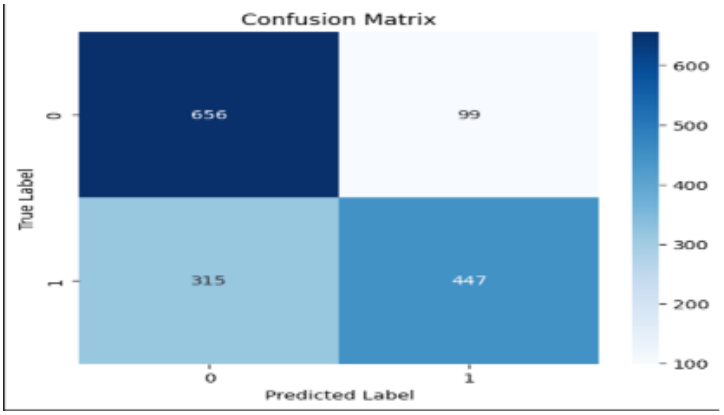

Figure 10 :Confusion matrix for test set

- True Positives (TP): 447: This is the number of instances where the model correctly predicted positive (class 1) when the actual label was positive.
- True Negatives (TN): 656:This is the number of instances where the model correctly predicted negative (class 0) when the actual label was negative.
- False Positives (FP): 99: This is the number of instances where the model incorrectly predicted positive when the actual label was negative.
- False Negatives (FN): 315: This is the number of instances where the model incorrectly predicted negative when the actual label was positive.

```
macro_precision, macro_recall, macro_f1, micro_precision, micro_recall, micro_f1 = metrics_calculator(pred_flat, labels_flat)
print(macro_precision, macro_recall, macro_f1, micro_precision, micro_recall, micro_f1)

✓ 0.0s

0.7471367458494131 0.7277441727068885 0.7373129669876355 0.7270929466051417 0.7270929466051417 0.7270929466051417
```

Figure 11:Precision,Recall and F1 scores for Micro and Macro averages

A macro perspective considers the average performance per class, ensuring a balanced evaluation. On the other hand, micro metrics, which account for the global count of true positives, false positives, and false negatives, provide an overview of the model's overall precision, recall, and F1-Score. The consistently high values for both macro and micro metrics (precision: 0.748, recall: 0.727, F1-Score: 0.737) suggest that the model achieves a well-rounded performance across individual classes and the entire dataset, demonstrating its effectiveness in classification tasks.

ROC Curve (Receiver Operating Characteristic Curve)

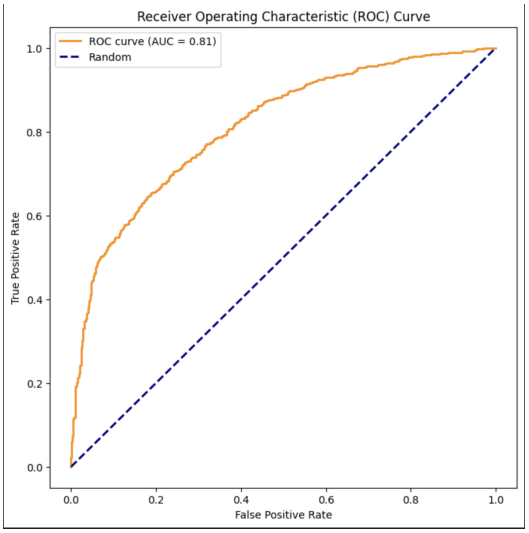

Figure 12:ROC-AUC curve

The ROC curve displays how well a model differentiates between true positives and false positives, capturing the trade-off between sensitivity and 1-specificity; a higher AUC-ROC signifies better discrimination power.An AUC of 81 curve indicates that the model has good discrimination power in distinguishing between class 0 and 1 for prediction of legal judgment.

Pegasus

Rouge1 Recall	Rouge2 Recall	RougeL Recall	Rouge1 Precision	Rouge2 Precision	RougeL Precision	Rouge 1F1-score	Rouge2 F1-score	RougeL F1-score
16.57	6.49	14.94	52.99	26.46	48.18	24.48	10.01	22.16

Table 2 Metrics for final fine-tuned model

Model	Dataset	Rouge-1 (Precision)	Rouge-2 (Precision)	Rouge-L (Precision)
Pegasus (google/pegasus-large · Hugging Face)	CNN Daily Mail (cnn_dailymail · Datasets at Hugging Face)	45.68	14.56	20.07
Legal Pegasus (nsi319/legal-pegasus · Hugging Face)	US-Litigation releases (Litigation Releases | U.S. Securities and Exchange Commission)	62.97	28.42	33.22
Pegasus Indian Legal (Fine-tuned) (akhilm97/pegasus_indian_legal · Hugging Face)	Indian-Legal documents (ninadn/indian-legal - Hugging Face)	52.99	26.4	48.1

Table 3 ROUGE (Precision) scores of all the Summarization models

Based on the memory constraints and the size of the dataset used for text summarization of the base model, the base model pegasus could not be directly used for fine-tuning as it was giving a CUDA out of memory error. Each checkpoint file in the pegasus model was around 2.2GB and it has 568M parameters [7].Therefore, the fine-tuned version of pegasus (legal-pegasus) was used which was trained on the US litigation releases website as the base model for the checkpoint.

Conclusion

In conclusion, this project explored several state-of-the-art natural language processing models for legal text processing, including BERT, GPT-2, and PEGASUS. We successfully implemented these models for tasks like text summarization, sentiment analysis for judgment prediction, and text generation.

The fine-tuned BERT model achieved decent performance for judgment prediction, with an accuracy of 72.7% on the test set. However, there is room for improvement by using a larger dataset and more compute resources. The GPT-2 model showed promising text generation capabilities when trained on legal documents, evident through improving ROUGE scores and decreasing training loss over epochs. We observed the best results with a learning rate of 5e-5, AdamW optimizer, batch size of 8, and 10 epochs. Though, the generated text lacked coherence in several instances. Fine-tuning PEGASUS also yielded fair summarization with a ROUGE-1 (Precision) score of 52.99% on Indian legal documents.

Nonetheless, some limitations were faced during the project duration. The dataset sizes were restricted due to hardware constraints, which restricted extensive hyperparameter tuning and model experimentation. Additionally, spelling corrections and named entity recognition could not be incorporated in the data preprocessing pipeline owing to computational expenses over large corpora. For text summarization, the summary that was generated from the model for the test set was giving incomplete sentences at the end of the summary.

Going forward, utilizing larger datasets with more legal corpus variety would aid in enhanced model training. Exploring different neural network architectures tailored to legal text could further boost performance. Employing enhanced compute infrastructure would enable accelerated experiments with various hyperparameters and model designs. There is also scope for building customized metrics focused on legal language syntax and semantics beyond commonly used ROUGE and perplexity. Overall, the project results showcase

promising initial steps, but more robust models could be developed through larger datasets and models designed specifically for legal domains.

References

1. Alammar, J. (n.d.). The Illustrated GPT-2 (Visualizing Transformer Language Models). Retrieved from https://jalammar.github.io/illustrated-gpt2/
2. Streamlit Docs. (n.d.). Build a ChatGPT-like app. Retrieved from https://docs.streamlit.io/knowledge-base/tutorials/build-conversational-apps#build-a-chatgpt-like-app
3. OpenAI. (n.d.). ChatGPT.
4. Papers with Code. (n.d.). GPT-2. Retrieved from https://paperswithcode.com/method/gpt-2
5. Pre-trained Language Models for the Legal Domain: A Case Study on Indian Law Shounak Paul, Arpan Mandal, Pawan Goyal, Saptarshi Ghosh. https://arxiv.org/pdf/2211.03442v1.pdf
6. ILDC for CPJE: Indian Legal Documents Corpus for Court Judgment Prediction and Explanation Vijit Malik, Rishabh Sanjay, Shubham Kumar Nigam, Kripabandhu Ghosh, Shouvik Kumar Guha, Arnab Bhattacharya, Ashutosh Modi. https://arxiv.org/pdf/2209.06049.pdf
7. https://huggingface.co/transformers/v3.1.0/model_doc/pegasus.html

Image References

1. BERT architecture https://www.semanticscholar.org/paper/Effectively-Leveraging-BERT-for-Legal-Document-Limsopatham/5bdfbded52da393eb3f7a0270a4f135bb564ee4a
2. PEGASUS Architecture - https://paperswithcode.com/method/pegasus
3. ResearchGate. (n.d.). Structure of the applied GPT-2 medium architecture. Retrieved from https://www.researchgate.net/figure/Structure-of-the-applied-GPT-2-medium-architecture_fig2_365625866
4. Alammar, J. (n.d.). The Illustrated GPT-2 (Visualizing Transformer Language Models). Retrieved from https://jalammar.github.io/illustrated-gpt2/