Project Report

NER In Hindi Legal Text

July 28Th, 2023

Submitted To - Dr. Pawan Kumar

Submitted By -

20UCS059 - Chitransh Jaiswal

20UCC115 - Vitthal Sanadhya

20UCS163 - Romil Patni

21UCC111 - Varad Bane

Abstract

The automatic extraction and recognition of named entities (NER) in legal documents hold paramount significance for facilitating efficient information retrieval and analysis within the legal domain. The primary objective of this project is to develop and fine-tune an advanced BERT-based token classification model specifically tailored for Named Entity Recognition on legal documents composed in the Hindi language. Leveraging the pre-trained BERT model, 'google/muril-base-cased,' empowers the system to adeptly encode the contextual intricacies of the legal text and meticulously classify each token into predefined named entity categories.

The model's fine-tuning process involves meticulous training on a meticulously annotated dataset of legal documents. This dataset has been diligently labeled to encompass an array of named entities, encompassing person names, organization names, locations, legal terminologies, and other pertinent entities germane to the legal context. The incorporation of these diverse classes bolsters the model's ability to accurately discern and categorize the varied named entities prevalent in legal texts written in Hindi.

The evaluation of the model's performance is carried out utilizing standard evaluation metrics. Precision, recall, F1-score, and accuracy are carefully measured and analyzed to gauge the model's efficacy in precisely identifying and classifying named entities. The report extensively discusses the obtained results, delving into the model's strengths and limitations.

Introduction

The legal domain involves an extensive corpus of text in the form of legal documents, judgments, statutes, and more. Extracting relevant information from these documents is a challenging task, and named entity recognition plays a vital role in this process. Named entities, such as names of individuals, organizations, locations, dates, and other specialized legal terms, provide crucial contextual information in legal texts. The objective of this project is to develop a robust NER system that can accurately identify and classify these named entities in Hindi legal documents. The model will be based on the BERT architecture, a powerful language representation model, known for its contextual understanding of text.

Related Work

Prior research in Named Entity Recognition has shown significant progress, particularly with the advent of deep learning and transformer-based models like BERT. Many studies have focused on NER for English texts, but only a limited amount of work exists for NER in the Hindi language and specifically for legal documents. Researchers have experimented with various architectures and transfer learning techniques to tackle the NER task in Hindi. However, there is still a need for a specialized NER system for Hindi legal texts, considering the domain-specific terminologies and complexities involved.

Data Collection and Preprocessing

The dataset used in this project was obtained from authentic legal documents sourced from the website ***https://services.ecourts.gov.in/ecourtindia_v6/.*** These documents include court judgments, legal agreements, and legal notices in the Hindi language. The dataset was meticulously annotated by us to mark the boundaries of named entities, such as person names, organization names, locations, dates, and legal terminologies. The annotation process involved carefully identifying and labeling the relevant entities in the text samples to create a comprehensive and accurate labeled dataset. This dataset serves as the foundation for training and evaluating the BERT-based token classification model for Named Entity Recognition on Hindi legal documents. The authenticity and richness of the dataset enable the model to capture the domain-specific terminologies and complexities present in real-world legal texts, making it well-suited for addressing the NER task effectively.

Model Architecture and Training

The BERT-based token classification model, 'google/muril-base-cased,' is chosen for its ability to encode contextual information and its pre-trained multilingual capabilities. The model is fine-tuned using the annotated dataset, and training hyperparameters such as batch size, learning rate, and number of epochs are carefully selected. The model is trained on a GPU-based environment to accelerate the training process.

Specifications

1. Dataset and Training Details:

- Dataset Size: 1200 lines of annotated data.
- NER Tags:

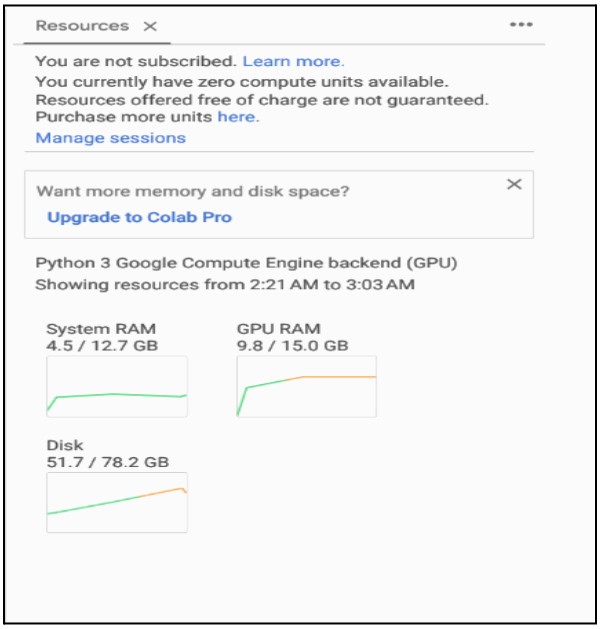
{'OTHER_PERSON', 'DATE', 'PLACE', 'WITNESS', 'PRECEDENT', 'CASE_NUMBER', 'PROVISION', 'PETITIONER', 'LAWYER', 'DISTRICT', 'JUDGE', 'COURT', 'RESPONDENT', 'ORG', 'STATE', 'YEAR', 'AGE'}

- Data Format: JSON file
- Model Architecture: google/muril-base-cased
- Learning Rate: 2e-5
- Evaluation Steps: 100
- Number of Epochs: 60

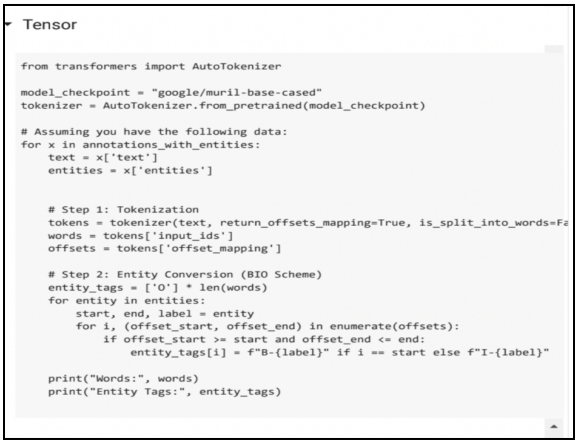

```
{
        "classes": ["COURT", "PETITONER", "RESPONDENT", "JUDGE", "LAWYER", "DATE", "ORG", "PROVISION", "PRECEDENT", "CASE_NUMBER",
"WITNESS", "OTHER_PERSON", "PLACE", "DISTRICT", "STATE", "AGE", "YEAR"],
        "annotations": [
                ["सरकार बनाम सुनील कुमार\r", {
                        "entities": [
                                [0, 5, "PETITONER"],
                                [11, 22, "RESPONDENT"]
                        ]
                }],
                ["प्रथत सूचना रिपोर्ट - 02/2017-18\r", {
                        "entities": [
                                [22, 32, "CASE_NUMBER"]
                        ]
                }],
                ["एनसीवी संख्या 20/2018\r", {
                        "entities": [
                                [14, 21, "CASE_NUMBER"]
                        ]
                }],
                ["प्रकरण संख्या - 14/2018\r", {
                        "entities": [
                                [16, 23, "CASE_NUMBER"]
                        ]
                }],
                ["पुलिस थाना आबकारी\r", {
                        "entities": [
                                [0, 10, "ORG"],
                                [11, 17, "PLACE"]
                        ]
                }],
                ["निरोधक दल शाहुपरा\r", {
                        "entities": [
                                [0, 9, "ORG"],
```

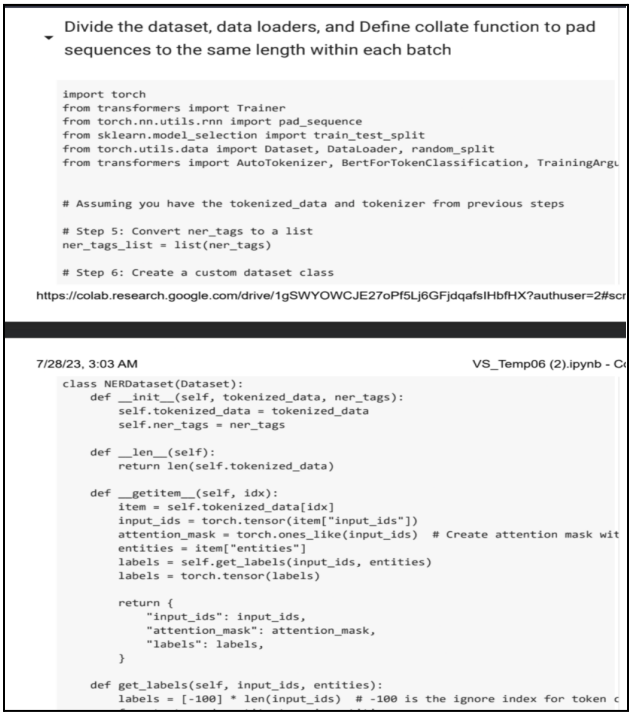
2. Hardware and Environment Details:

- System RAM: 12.7 GB
- GPU RAM: 15.0 GB (assuming a GPU is used for training)
- Disk Space: 78.2 GB (assuming the total disk space available)



3. Training and Validation Results:





Divide the dataset, data loaders, and Define collate function to pad sequences to the same length within each batch

```
import torch
from transformers import Trainer
from torch.nn.utils.rnn import pad_sequence
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader, random_split
from transformers import AutoTokenizer, BertForTokenClassification, TrainingArgu


# Assuming you have the tokenized_data and tokenizer from previous steps

# Step 5: Convert ner_tags to a list
ner_tags_list = list(ner_tags)

# Step 6: Create a custom dataset class
```

https://colab.research.google.com/drive/1gSWYOWCJE27oPf5Lj6GFjdqafsIHbfHX?authuser=2#scr

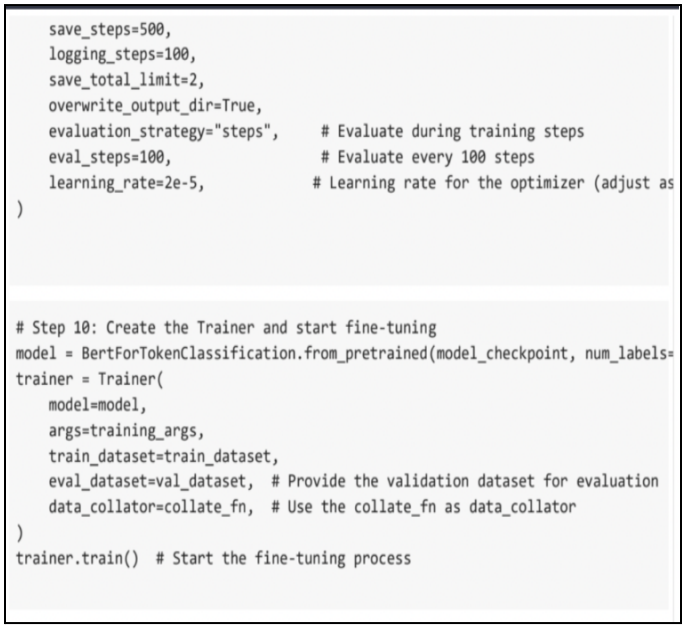
7/28/23, 3:03 AM VS_Temp06 (2).ipynb - C

```
class NERDataset(Dataset):
    def __init__(self, tokenized_data, ner_tags):
        self.tokenized_data = tokenized_data
        self.ner_tags = ner_tags

    def __len__(self):
        return len(self.tokenized_data)

    def __getitem__(self, idx):
        item = self.tokenized_data[idx]
        input_ids = torch.tensor(item["input_ids"])
        attention_mask = torch.ones_like(input_ids)  # Create attention mask wit
        entities = item["entities"]
        labels = self.get_labels(input_ids, entities)
        labels = torch.tensor(labels)

        return {
            "input_ids": input_ids,
            "attention_mask": attention_mask,
            "labels": labels,
        }

    def get_labels(self, input_ids, entities):
        labels = [-100] * len(input_ids)  # -100 is the ignore index for token c
```

```
    def get_labels(self, input_ids, entities):
        labels = [-100] * len(input_ids)  # -100 is the ignore index for token c
        for start, end, entity_type in entities:
            labels[start] = self.ner_tags.index(entity_type)
            for i in range(start + 1, end):
                labels[i] = self.ner_tags.index(entity_type)
        return labels

# Step 6: Create the dataset
dataset = NERDataset(tokenized_data, ner_tags_list)  # Pass tokenized_data and r

# Step 7: Divide the dataset into training, validation, and testing sets
train_size = int(0.8 * len(dataset))
val_size = int(0.1 * len(dataset))
test_size = len(dataset) - train_size - val_size
train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, va

# Step 8: Define collate function to pad sequences
def collate_fn(batch):
    input_ids = [item["input_ids"] for item in batch]
    attention_mask = [item["attention_mask"] for item in batch]
    labels = [item["labels"] for item in batch]

    # Pad sequences to the same length within each batch
    input_ids = pad_sequence(input_ids, batch_first=True, padding_value=tokenize
    attention_mask = pad_sequence(attention_mask, batch_first=True, padding_valu
    labels = pad_sequence(labels, batch_first=True, padding_value=-100)

    return {
        "input_ids": input_ids,
        "attention_mask": attention_mask,
        "labels": labels,
    }

# Step 9: Create data loaders for training, validation, and testing
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True, collate_fr
val_loader = DataLoader(val_dataset, batch_size=16, collate_fn=collate_fn)
test_loader = DataLoader(test_dataset, batch_size=16, collate_fn=collate_fn)
```

Traning Arg & Start fine-tuning and Evaluate the model

```
# Step 9: Configure training arguments
output_dir = "./fine_tuned_model"
training_args = TrainingArguments(
    output_dir=output_dir,
    num_train_epochs=40,
    per_device_train_batch_size=4,   # Adjust this based on GPU memory (start wi
```



```
    save_steps=500,
    logging_steps=100,
    save_total_limit=2,
    overwrite_output_dir=True,
    evaluation_strategy="steps",     # Evaluate during training steps
    eval_steps=100,                  # Evaluate every 100 steps
    learning_rate=2e-5,             # Learning rate for the optimizer (adjust as
)


# Step 10: Create the Trainer and start fine-tuning
model = BertForTokenClassification.from_pretrained(model_checkpoint, num_labels=
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,  # Provide the validation dataset for evaluation
    data_collator=collate_fn,  # Use the collate_fn as data_collator
)
trainer.train()  # Start the fine-tuning process
```

Training Loss: (0.489200 in 7200 steps)

The training loss measures how well the model learns during training. It shows how much the Wmodel's predictions deviate from the ground-truth labels on the training data. Lower training loss indicates better learning.

Step	Training Loss	Validation Loss
100	2.811300	2.745647
200	2.704500	2.638340
300	2.614200	2.553470
400	2.492000	2.431439
500	2.388100	2.316432
600	2.217600	2.173510
700	2.095500	2.042587
800	2.043900	1.906650
900	1.861300	1.800923
1000	1.774200	1.660209
1100	1.595800	1.559789
1200	1.459800	1.415816
1300	1.357700	1.259709
1400	1.220600	1.195144
1500	1.018500	1.070387
1600	1.051500	1.001329
1700	0.909300	1.029336
1800	0.828900	0.800223

Evaluation Metrics

The evaluation of the NER model is based on standard metrics such as accuracy, precision, recall, and F1-score. These metrics provide insights into the model's ability to correctly classify named entities and avoid false positives or false negatives.

4. Test Accuracy:

The model's accuracy on the test dataset is not used during training or validation. It measures how well the model performs on new, unseen data. Higher test accuracy indicates better performance in real-world scenarios. Test Accuracy came out to be **0.8195**

5. F1 Score:

The F1 score is the harmonic mean of Precision and Recall. It provides a balanced measure of a model's performance, especially when there is an uneven class distribution. The F1 score combines

Precision and Recall, and it is used to assess the model's accuracy in correctly identifying positive instances while minimizing false positives and false negatives. **(0.8299)**

6. Precision:

The F1 score is the harmonic mean of Precision and Recall. It provides a balanced measure of a model's performance, especially when there is an uneven class distribution. The F1 score combines Precision and Recall, and it is used to assess the model's accuracy in correctly identifying positive instances while minimizing false positives and false negatives. **(0.8880)**

7. Recall:

The F1 score is the harmonic mean of Precision and Recall. It provides a balanced measure of a model's performance, especially when there is an uneven class distribution. The F1 score combines Precision and Recall, and it is used to assess the model's accuracy in correctly identifying positive instances while minimizing false positives and false negatives. **(0.8195)**

```
# Use the evaluate_model function to get accuracy, precision, recall, and f1-score on the test datas
test_accuracy, precision, recall, f1_score = evaluate_model(model, test_loader, device)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1_score:.4f}")
```

```
Test Accuracy: 0.8195
Precision: 0.8880
Recall: 0.8195
F1-score: 0.8299
```

8. Model Strengths:

The use of the google/muril-base-cased model is beneficial for NER tasks as it is a multilingual model and can handle Hindi text effectively.

Training for 60 epochs allows the model to learn from the data for an extended period, which could lead to improved performance.

9. Model Limitations:

The dataset size of 1200 lines may be relatively small for complex NER tasks, and the model's performance might benefit from a larger annotated dataset.

The model's performance heavily depends on the quality and representativeness of the annotated data.

Future Work

Future directions for the project include exploring advanced language representation models, domain-specific embeddings, and leveraging domain-specific lexicons to further enhance the NER system's performance. Additionally, expanding the dataset and exploring transfer learning from other related languages may improve the model's generalization to diverse legal texts.

References

[1] Dataset - https://services.ecourts.gov.in/ecourtindia_v6/

[2] Model - https://huggingface.co/google/muril-base-cased

[3] Colab - https://colab.research.google.com/

[4] Hindi NER - https://github.com/AbhiDhariwal/Hindi-NER