Machine Learning Experiment Report

Raghavendra S

1 Methodology

Initially, I employed a BERT-based model fine-tuned on a binary classification task to predict the better candidate between two resumes and their associated transcripts for a specific role. The model was trained on a dataset using a pre-trained BERT tokenizer and model. After training for 9 epochs on a system equipped with a Ryzen 7 6800HS CPU and a Ryzen RX 6700S GPU, the final accuracy on the test set was found to be 0.616.

In addition to this approach, I also experimented with a few-shot prompting technique, leveraging the same BERT model. For this, I utilized cosine similarity between the embeddings of candidates' transcripts and a prompt generated based on the role and examples from the training set. However, this method achieved an accuracy of only 0.55, which was lower than the previous model. So the BERT finetuned model is considered as the final model with accuracy 0.616.

Given the limitations in computing resources, I considered using more powerful large language models (LLMs) such as LLaMA, but I was unable to proceed with these due to insufficient resources. Furthermore, I acknowledge the potential for inherent biases in these models. Addressing these biases is crucial for improving the fairness and accuracy of the predictions, although I did not explore these strategies in this experiment.

2 Experiment Details

The experiment was conducted using the Hugging Face Transformers library for the BERT model. The primary steps involved data preprocessing, model initialization, training, and evaluation.

The text data from the resumes and transcripts was cleaned and tokenized using the BERT tokenizer. The model was trained for 9 epochs with a batch size of 8 and a learning rate of 2e-5. The optimizer used was AdamW.

3 Code

The following code snippet outlines the main steps of the experiment:

```
import pandas as pd
import re
import json
from transformers import BertTokenizer

# Function to clean text (Transcript)
def clean_text(text):
    # Remove unnecessary special characters
    return re.sub(r'[^A-Za-z0-9\s.,!?]', '', text)

# Function to convert Resume dictionary to text
def dict_to_text(Resume):
    if isinstance(Resume, str):
        try:
            # If Resume is a string, try converting it to a
                dictionary
            Resume = eval(Resume)  # Be cautious with eval; json.
                loads is safer for JSON input
        except:
            pass  # If conversion fails, we'll keep it as a string
    if isinstance(Resume, dict):
        Resume_text = ""
        for key, value in Resume.items():
            if isinstance(value, list):
                value = ", ".join(value)
            Resume_text += f"{key}: {value}. "
        return Resume_text
    return str(Resume)  # Return the string if it's not a dict

# Function to preprocess a single row
def preprocess_row(row):
    # Clean the Transcript text
    candidateATranscript_clean = clean_text(row['
        candidateATranscript'])
    candidateBTranscript_clean = clean_text(row['
        candidateBTranscript'])

    # Convert the Resume dictionaries to text
    candidateAResume_text = dict_to_text(row['candidateAResume'])
    candidateBResume_text = dict_to_text(row['candidateBResume'])

    # Concatenate Transcript and Resume for both candidates
    candidateA_data = candidateATranscript_clean + " " +
        candidateAResume_text
    candidateB_data = candidateBTranscript_clean + " " +
        candidateBResume_text

    # Combine into a tuple for further processing
    return candidateA_data, candidateB_data, row['role']

# Initialize BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Function to tokenize the preprocessed data
def preprocess_data(dataframe):
    inputs = []
    labels = []
```

```
for _, row in dataframe.iterrows():
    # Preprocess each row
    candidateA_data, candidateB_data, role = preprocess_row(row
        )

    # Combine both candidates' text with a separator token and
        the role as context
    combined_text = role + " [SEP] " + candidateA_data + " [SEP
        ] " + candidateB_data

    # Tokenize the combined input
    encoded_input = tokenizer.encode_plus(
        combined_text,
        add_special_tokens=True,
        max_length=512,
        truncation=True,
        padding='max_length',
        return_tensors='pt'
    )

    inputs.append(encoded_input)
    labels.append(1 if row['winnerId'] == row['candidateAId']
        else 0)

return inputs, labels
```

4 Results

The model was trained for 9 epochs, and the final accuracy on the test set was 61.6%. This accuracy was achieved after running the training loop for the specified number of epochs. The loss function used was cross-entropy loss, which is standard for binary classification tasks.

5 Conclusion

In this study, I explored two different approaches to predict the better candidate for a job role based on their resumes and interview transcripts: a fine-tuned BERT model and a few-shot prompting technique. While the few-shot prompting method did not perform as well, with an accuracy of 0.55, the fine-tuned BERT model achieved a test accuracy of 0.616. Therefore, the BERT model is the final model that i submit for this task.

Despite considering more advanced LLMs like LLaMA, I was constrained by available computational resources. Future work should include efforts to mitigate inherent biases within these models to enhance fairness and prediction accuracy.

6 Future Work

Future work could involve investigating the inherent biases present in the dataset, such as imbalances in the representation of different roles or candidate profiles. Addressing these biases might include using techniques like data augmentation, oversampling underrepresented classes, or incorporating fairness-aware learning algorithms to ensure a more equitable model performance.