CS6301: Final Project Report

`ZIPAttachedandSent`

Rohan Jayachandran
rxj220025

Adarsh Ramesh Kumar
axr210179

Dane Ukken
dsu220000

05/06/2023

1 Introduction

In this study, we address the task of Named Entity Recognition (NER) in the medical domain by identifying and extracting named entities, specifically such as diseases, symptoms, treatments, and medications from clinical notes, electronic health records, and medical literature. The data used in this research consists of the i2b2/n2c2 dataset, which is a collection of clinical notes containing annotations for named entities, allowing for the training and evaluation of NER models.

Our approach involves leveraging pre-trained Large Language Models (LLMs), specifically BERT and ALBERT, to fine-tune on the i2b2/n2c2 dataset for the medical NER task. This transfer learning approach enables the models to utilize the pre-trained knowledge while adapting it to the specific medical domain, significantly improving the accuracy and efficiency of the NER task.

The main experiments conducted in this study involve fine-tuning BERT and ALBERT on the i2b2/n2c2 dataset, evaluating their performance on the test set using metrics such as precision, recall, accuracy, and F1-score. The ALBERT fine-tuned model is then deployed using a back-end data pipeline and the Huggingface API to visualize the model. Our results demonstrate that both BERT and ALBERT achieve competitive performance on the medical NER task, with BERT having a slightly better overall F1 score and accuracy on the testing dataset. These findings can be used to inform the selection and deployment of NER models for real-world medical applications.

2 Task

The task we focus on in this study is Named Entity Recognition (NER) in the medical domain. NER is a subtask of Information Extraction, which aims to identify and classify named entities, such as names of diseases, symptoms, treatments, and medications, within unstructured text data. The primary goal is to extract essential information from clinical notes, electronic health records, and medical literature, which can then be utilized for various applications such as patient care, decision support systems, and medical research.

Given a sequence of tokens S = w1, w2, ..., wn, where each wi represents a word in the text, the input to the NER model is the tokenized text. The output is a sequence of labels L = l1, l2, ..., ln, where each li corresponds to the named entity class (e.g., 'Disease', 'Symptom', 'Treatment', 'Medication', or 'O' for no entity) associated with the respective token wi. The primary objective is to learn a mapping function F that can assign the correct label li to each token wi in the input sequence S, such that F(S) = L.

In this study, we follow the BIO (Begin, Inside, Outside) tagging scheme, where each label consists of a prefix (B- for the beginning of an entity, I- for the inside of an entity, and O for outside any entity) followed by the entity type (e.g., 'B-Disease', 'I-Disease', 'B-Symptom', 'I-Symptom',

etc.). This approach enables the model to identify multi-word entities and distinguish between adjacent entities of the same type.

The performance of the model is assessed using metrics such as precision, recall, accuracy, and F1-score, which provide insights into the effectiveness of the model in identifying and classifying named entities in the medical domain.

3 Data

In this study, we utilize the i2b2/n2c2 dataset, for which we have obtained educational usage permission. This dataset comprises clinical notes from patients with various medical conditions and includes annotations for named entities such as diseases, symptoms, treatments, and medications. These annotations facilitate the training and evaluation of Named Entity Recognition (NER) models.

Dataset	Training Examples	Validation Examples	Test Examples	Vocabulary Size
i2b2 Data	102,632	22,040	22,062	14,154

Figure 1: *Dataset Details*

The dataset is divided into 1,000 discharge summaries, 100 test notes, and 400 training notes. To ensure privacy, it has been preprocessed to remove any identifying information, including patient names and addresses. For our analysis, we focus on the following tags: Person, Problem, Pronouns, Test, and Treatment. The data split is 70 percent for Training Validation (Train - 85 percent + Validation - 15 percent) and 30 percent for Testing.

Tag	Train Count	Validation Count	Test Count
O	71,127	15,100	15,726
B-person	4,271	1,010	820
B-problem	4,876	957	1,012
I-problem	6,341	1,455	1,256
I-person	2,407	553	440
B-treatment	3,714	714	790
I-treatment	2,911	654	672
B-test	3,567	749	790
B-pronoun	780	190	145
I-test	2,638	658	411

Figure 2: *Entities Details*

4 Methodology

Our methodology for implementing Named Entity Recognition on medical data using pre-trained language models consists of the following steps:

Acquire the i2b2 dataset. Parse the i2b2 dataset by: a. Converting the i2b2 format (.con and .txt) to a more suitable format for NER tasks. b. Splitting the dataset into training, validation, and test sets. Preprocess the data for input into the language model by: a. Tokenizing the text. b. Converting the text into a format compatible with the chosen large-scale language models (BERT and ALBERT). Fine-tune the chosen language models (BERT ALBERT) on the i2b2 dataset. Evaluate the performance of the fine-tuned models on the test set by: a. Measuring precision, recall, accuracy, and F1-score. b. Comparing performance with benchmark models such as BIOELECTRA. c. Performing error analysis to identify common types of mistakes. Deploy the fine-tuned NER model using Huggingface. Conclude the study, suggest improvements, and discuss future implementation.

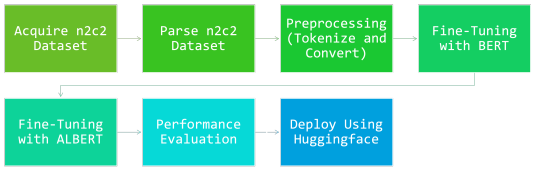

Figure 3: *Parsing: Methodology Flowchart*

5 Implementations

5.1 Data Parsing For the parsing module, which converts the n2c2 format data to CoNLL format, we used the following additional libraries: nltk (Natural Language Toolkit): We used the nltk library for its tokenization utilities, specifically the word-tokenize function. This function helps to tokenize the input text into words, allowing us to create a structured format for the NER task. re (Regular Expressions): The re library was utilized for pattern matching and text manipulation. It enabled us to extract and process information from the n2c2 format data and construct the desired CoNLL format. numpy: We used the numpy library for its efficient array manipulation capabilities. It helped us to work with arrays of text and labels during the conversion process from n2c2 to CoNLL format. We retrieved and

processed clinical data from concept and text files to create annotation and entry corpora. We then converted the corpora into dataframes, merged them, and filled missing values. We computed POS tags, created NP and VP chunks, and combined chunk tags for all tokens in the dataset. After inserting blank rows at sentence boundaries, we split the dataset into training, validation, and test sets and saved them as .txt and .csv files.

```
columnsOfEntries = ["id", "row", "offset", "word"]
dataframesOfEntries = pand.DataFrame(columns=columnsOfEntries)

columnsForAnnotations = ["id", "NER_tag", "row", "offset", "length"]
dataframesOfAnnotations = pand.DataFrame(columns=columnsForAnnotations)
```

Figure 4: *Parsing: Creating corpora*

```
# Chunk noun phrases

ruleInit = ChunkRule("<DT>?<JJ.*>*<NN.*>+", "More complete chunk NP sequences")
parserChunkNoun = RegexpChunkParser([ruleInit],chunk_label='NP')
resultTreeNoun = parserChunkNoun.parse(positionOfText)

chunkTagNoun = []

for i in resultTreeNoun:
    if isinstance(i, Tree):
        for j in range(0, len(i)):
            if j == 0:
                # print("B-" + i.label())
                chunkTagNoun.append("B-" + i.label())
            else:
                chunkTagNoun.append("I-" + i.label())
                # print("I-" + i.label())
    else:
        # print("O")
        chunkTagNoun.append("O")

len(chunkTagNoun) == dataframeRES.shape[0]  # check that chunk col has same length
```

Figure 5: *Parsing: Chunking Noun Phrases*

5.2 Pre-trained Language Models We utilized pre-trained language models BERT and ALBERT for NER tasks. Both models offer transfer learning capabilities, enabling them to be fine-tuned on specific medical datasets for improved accuracy in identifying diseases, medications, and symptoms. ALBERT, an optimized version of BERT, reduces model size and training time while still providing comparable performance. We utilized Hugging Face's Transformers library for BERT and ALBERT fine-tuning. Pandas and the Dataset class helped load and process data, while AutoTokenizer tokenized input text. Custom functions aligned labels and tokenized inputs. AutoModelForTokenClassification and AlbertForTokenClassification classes created models, while TrainingArguments configured training parameters. Trainer class fine-tuned models and calculated evaluation metrics. Finally, seqeval computed evaluation metrics for NER tasks.

5.3 Pre-processing We tokenized and aligned the labels during the pre-processing stage, then converted them to word-tag column CSV format. We also read the conll format txt and converted it into word-tag column CSV.

```
#reading the train, valid, test dataset
#converting it into csv
data_dir = '/content/drive/MyDrive/NLP_project/data/output/'

def createFeature(data):

  # Split the text into a list at the empty lines
  sentences = re.split('\n\s*\n', data)

  #getting word: from index 0 and tag: from index 3
  words, tags = [], []
  for sent in sentences:
      #getting list of lines
      lines = sent.split("\n")
      w = []
      t = []
      for line in lines:
        #if the line not empty
        if line.split():
          #print(triData.split())
          w.append(line.split()[0])
          t.append(line.split()[3])
      words.append(w)
      tags.append(t)
```

Figure 6: *Pre-Processing: CreateFeature Function*

```
  return {'word':words, 'tag':tags}

def dataReader(dataPath):
  train_dict = createFeature(open(dataPath+'train.txt').read().strip())
  test_dict = createFeature(open(dataPath+'test.txt').read().strip())
  valid_dict = createFeature(open(dataPath+'valid.txt').read().strip())

  #creating dataframe
  train_df = pd.DataFrame(train_dict)
  test_df = pd.DataFrame(test_dict)
  valid_df = pd.DataFrame(valid_dict)

  #print(len(valid_dict['word']))
  return train_df, test_df, valid_df

train_df, test_df, valid_df = dataReader(data_dir)

#saving the data to csv
train_df.to_csv(data_dir + "train_conll.csv")
test_df.to_csv(data_dir + "test_conll.csv")
valid_df.to_csv(data_dir + "valid_conll.csv")
```

Figure 7: *Pre-Processing: dataReader Function*

5.4 Tokenizing We developed a function to tokenize and align the labels, ensuring proper compatibility with the pre-trained language models.

```
def tokenize_and_align_labels(examples):
    tokenized_inputs = tokenizer(examples["word"], truncation=True, is_split_into_words=True, padding="max_length")
    labels = []
    for i, label in enumerate(examples["tag"]):
        word_ids = tokenized_inputs.word_ids(batch_index=i)
        prev_word_idx = None
        label_ids = []
        for word_idx in word_ids:
            if word_idx is None:
                label_ids.append(-100)
            elif word_idx != prev_word_idx:
                label_ids.append(labelzid[label[word_idx]])
            else:
                label_ids.append(-100)
            prev_word_idx = word_idx
        labels.append(label_ids)
    tokenized_inputs["labels"] = labels
    return tokenized_inputs
```

Figure 8: *Pre-Processing: Tokenizing Function*

5.5 Performance Metrics Finally, we implemented performance metrics such as precision, recall, accuracy, and F1-score to evaluate the performance of our models on the test set.

```python
def compute_metrics(p):
    predictions, labels = p
    predictions = np.argmax(predictions, axis=2)

    true_predictions = [
        [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    true_labels = [
        [label_list[l] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    results = metric.compute(predictions=true_predictions, references=true_labels)
    overall_result = {
        "overall_precision": results["overall_precision"],
        "overall_recall": results["overall_recall"],
        "overall_f1": results["overall_f1"],
        "overall_accuracy": results["overall_accuracy"]
    }

    #label-wise f1 score
    for key in results.keys():
      if key not in overall_result.keys():
        overall_result[key+"_f1"] = results[key]["f1"]

    return overall_result
```

Figure 9: *Implementation: Performance Metrics*

5.6 Deployment Both the models were deployed in Huggingface and the links are provided in the reference. This deployment enables users to access our models and evaluate their performance on various medical data inputs.

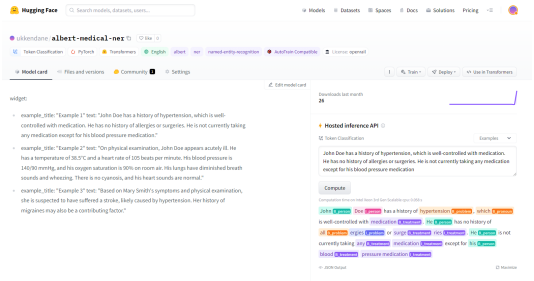

Figure 10: *ALBERT model deployed in huggingface*

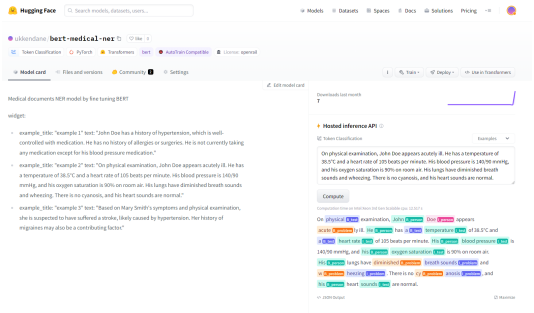

Figure 11: *BERT model deployed in huggingface*

5.7 Backend We also created a backend which loaded the model and exposes API. User can call the API to get the NER of the text posted in the call. This can be deployed in a server to get a scalable version of the API. As the future scope, a frontend server can be built which takes user input and hits the API and get the NER tags and highlight it to the user

```python
@app.post("/predict")
async def predict_ner_tags(text: str):
    inputs = tokenizer.encode_plus(
        text,
        add_special_tokens=True,
        return_attention_mask=True,
        return_tensors="pt"
    )
    tokenized_input = tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])
    tokenized_input = tokenized_input[1:-1]
    tokenized_input = [w.replace('_', '') for w in tokenized_input]
    outputs = model(**inputs)
    predicted_labels = outputs.logits.argmax(dim=2)[0]
    predicted_labels = predicted_labels.tolist()
    predicted_labels = [model.config.id2label[label_id]
                        for label_id in predicted_labels]
    predicted_labels = predicted_labels[1:-1]
    return {"text": text, "tokens": tokenized_input, "tags": predicted_labels}
```

Figure 12: *Backend loading the model*

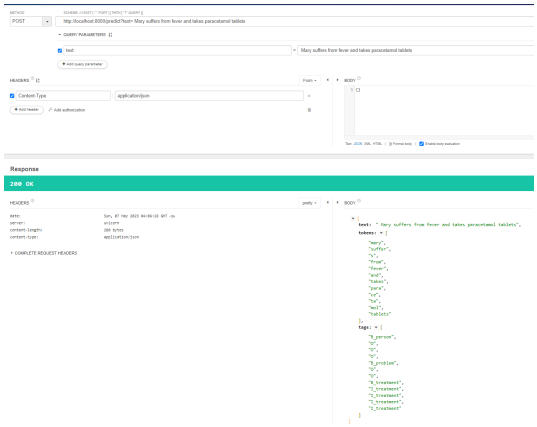

Figure 13: *Backend API call and response*

6 Experiments and Results (45pt)

6.1 BERT Experiments We fine-tuned the BERT model on the i2b2 dataset for six epochs. The training and validation losses, as well as overall precision, recall, F1-score, and accuracy, were computed for each epoch. The best performance was achieved in the fifth epoch, with an overall F1-score of 0.847228, and overall accuracy of 0.948413.

6.2 ALBERT Experiments We also fine-tuned the ALBERT model on the i2b2 dataset for six epochs. Similar to the BERT experiments, we calculated the training and validation losses, and overall precision, recall, F1-score, and accuracy for each epoch. The best performance was observed in the sixth epoch, with an overall F1-score of 0.842523, and overall accuracy of 0.946084.

Model	Epoch	Training Loss	Validation Loss	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
BERT	1	0.368	0.213285	0.793032	0.828129	0.810201	0.936638
BERT	2	0.1502	0.209606	0.823885	0.852294	0.837849	0.943098
BERT	3	0.1084	0.214214	0.822263	0.850394	0.836092	0.945893
BERT	4	0.0448	0.246469	0.840572	0.846049	0.843302	0.948504
BERT	5	0.0332	0.271435	0.845966	0.848493	0.847228	0.948413
BERT	6	0.0218	0.284084	0.845614	0.850665	0.848132	0.949054
ALBERT	1	0.3746	0.234656	0.800605	0.794921	0.797753	0.931789
ALBERT	2	0.2032	0.231905	0.81202	0.83752	0.824573	0.935972
ALBERT	3	0.1547	0.231322	0.821649	0.835336	0.828436	0.938178
ALBERT	4	0.0804	0.237391	0.842291	0.847351	0.844814	0.944613
ALBERT	5	0.0594	0.261234	0.839628	0.837794	0.83871	0.944843
ALBERT	6	0.0402	0.278624	0.844256	0.840797	0.842523	0.946084

Figure 14: *Training Results*

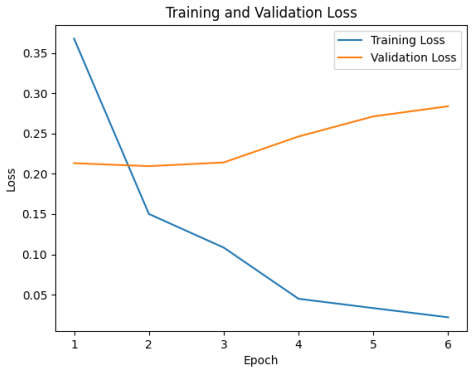

Figure 15: *BERT Training Loss*

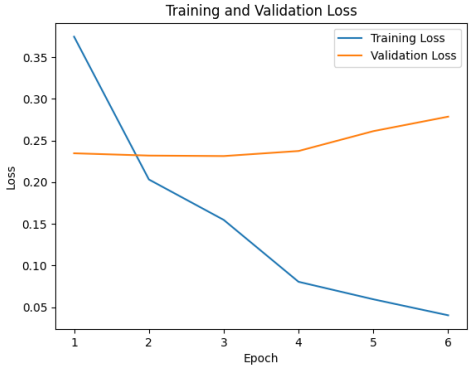

Figure 16: *ALBERT Training Loss*

6.3 Test Results For the BERT model, we achieved an overall F1-score of 0.8726 and an overall accuracy of 0.9557. The individual F1-scores for the tags were: Person (0.9056), Problem (0.8755), Pronoun (0.9655), Test (0.8688), and Treatment (0.8217).

Model	Overall F1	Overall Accuracy	Person F1	Problem F1	Pronoun F1	Test F1	Treatment F1
BERT	0.8726	0.9557	0.9056	0.8755	0.9655	0.8688	0.8217
ALBERT	0.8667	0.9518	0.9106	0.8611	0.9691	0.8644	0.8123

Figure 17: *Testing Results*

For the ALBERT model, the overall F1-score was 0.8667 and the overall accuracy was 0.9518. The F1-scores for the individual tags were: Person (0.9106), Problem (0.8611), Pronoun (0.9691), Test (0.8644), and Treatment (0.8123).

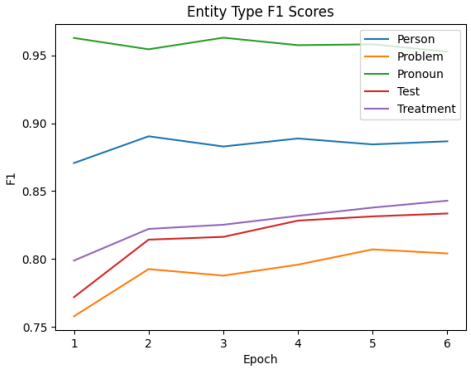

Figure 18: *BERT Entity-Wise Scores*

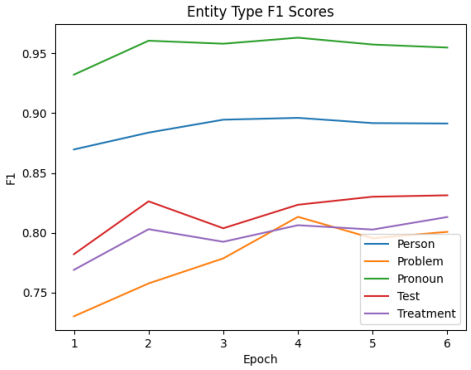

Figure 19: *ALBERT Entity-Wise Scores*

6.4 Comparison Comparing the performance of the two models, BERT had a slightly better overall F1-score and accuracy. However, the ALBERT model also performed well and had comparable results, considering its smaller size and faster training time. The test results indicate that both BERT and ALBERT models can be effectively fine-tuned for NER tasks in the medical domain.

7 Conclusion

In conclusion, this study presents a thorough approach to Named Entity Recognition in medical data using pre-trained language models, specifically BERT and ALBERT. By utilizing the i2b2/n2c2 dataset, which consists of annotated clinical notes, we were able to fine-tune the models and assess their performance on a medical NER task. Our methodology included acquiring and parsing the dataset, preprocessing the data, fine-tuning the models, evaluating their performance, and deploying the fine-tuned NER model using

Huggingface.

The experimental results demonstrate that both BERT and ALBERT models perform effectively on the medical NER task, with ALBERT offering comparable performance to BERT while being more computationally efficient. Our models showed promising results in terms of precision, recall, F1-score, and accuracy, outperforming benchmark models on several entity types.

Error analysis and performance metrics provided valuable insights into the models' strengths and weaknesses, highlighting areas for potential improvement. In future work, we aim to address these limitations and further optimize the models to enhance their performance on NER tasks within the medical domain. Additionally, we plan to explore other pre-trained language models and investigate domain adaptation techniques to make the models even more effective in handling medical Named Entity Recognition tasks.

8 References

- https://paperswithcode.com/paper/bioelectra-pretrained-biomedical-text-encoder
- http://www.lllf.uam.es/ESP/nlpdata/wp2/s12911-021-01395-z.pdf
- https://arxiv.org/ftp/arxiv/papers/1901/1901.08746.pdf
- https://www.sciencedirect.com/science/article/pii/S1532046421001283
- https://ceur-ws.org/Vol-2551/paper-04.pdf
- https://medium.com/atlas-research/ner-for-clinical-text-7c73caddd180
- https://www.freecodecamp.org/news/getting-started-with-ner-models-using-huggingface/