Question Answering System for Electronic Medical Records Using ClinicalBERT

Team Members: Sriram Natarajan, Rickston Pinto

Summary

1. We identified a data source we would like to use to work with this project.
2. Begun with data cleaning loops and pre-processing
3. Developed initial code framework for model training and evaluation ahead.

Background

Our work so far has revolved around zeroing in on the dataset we want to use for our task as outlined in the proposal.
This has proven to be a more time consuming and challenging task, however we now have decided on using this dataset:

https://huggingface.co/datasets/Eladio/emrqa-msquad/blob/main/data/validation-00000-of-00001.parquet

This is so we can have data of the format in 2 files broadly, a “train” file and a “validation” file

Here’s an example of what these train.json and valid.json files could look like as we anticipate

```
{
 "context": "Patient medical record text...",
 "question": "What is the patient's current medication?",
 "answer": "Lisinopril 10mg daily",
 "answer_start": 156
}
```

This above is one block in a series of many such in the json.

The parameters for consideration:

1. Completely labeled with questions/answers that will be of use to physicians during their decision making process.
2. Able to understand medical terminology and short phrases, such as training from clinicalBERT

3. Ideally have some previous implementation, so we're not completely starting from scratch and we have some places to look to if we're unsure.

Data Source

This dataset from the Hugging Face model fit these criteria well. Other contenders, which we might revisit throughout the remainder of project

- https://paperswithcode.com/dataset/medqa-usmle This is not exactly what we are looking for, as it's more geared toward answering questions medical students would see on their USMLE, and not clinical considerations, but it is nonetheless useful and can be used to supplement our code later if need.

Progress so far has been on identifying this dataset above, and steps to obtain it. Prior to this, we looked into a Kaggle contest which had some interesting prediction challenges, and reached out to the organizers of the contest to get access to the data, but we did not hear back from them.

With this finalized dataset, we began coding the ClincalBERT-based architecture with this data.

Data Download and Processing

The given link download [10.5 MB] will lead you to a 10.5 MB download of a .paraquet file, an optimized file structure for dealing with large columnar families with spark.

1. We converted the file to a CSV using Pandas dataframes, which expanded it to ~80MB and got a preview of the data using .head() and .firstN() functions

```
import pandas as pd
import numpy.core.multiarray

# Convert to CSV
df = pd.read_parquet('validation-00000-of-00001.parquet')
df.to_csv('dataset.csv')


# Preview the first 20 rows of the CSV
df_preview = pd.read_csv('dataset.csv', nrows=20)
print(df_preview)
import pandas as pd
```

Gives

```
(base) rickston@Rickstons-MBP-2 Downloads % conda activate idlf24
(idlf24) rickston@Rickstons-MBP-2 Downloads % python3 starship.py
    Unnamed: 0  ...                                            answers
0            0  ...  {'answer_end': array([1578]), 'answer_start': ...
1            1  ...  {'answer_end': array([432]), 'answer_start': a...
2            2  ...  {'answer_end': array([1244]), 'answer_start': ...
3            3  ...  {'answer_end': array([1861]), 'answer_start': ...
4            4  ...  {'answer_end': array([412]), 'answer_start': a...
5            5  ...  {'answer_end': array([443]), 'answer_start': a...
6            6  ...  {'answer_end': array([2246]), 'answer_start': ...
7            7  ...  {'answer_end': array([476]), 'answer_start': a...
8            8  ...  {'answer_end': array([1653]), 'answer_start': ...
9            9  ...  {'answer_end': array([1806]), 'answer_start': ...
10          10  ...  {'answer_end': array([1711]), 'answer_start': ...
11          11  ...  {'answer_end': array([1159]), 'answer_start': ...
12          12  ...  {'answer_end': array([2321]), 'answer_start': ...
13          13  ...  {'answer_end': array([508]), 'answer_start': a...
14          14  ...  {'answer_end': array([1085]), 'answer_start': ...
15          15  ...  {'answer_end': array([2221]), 'answer_start': ...
16          16  ...  {'answer_end': array([455]), 'answer_start': a...
17          17  ...  {'answer_end': array([1334]), 'answer_start': ...
18          18  ...  {'answer_end': array([948]), 'answer_start': a...
19          19  ...  {'answer_end': array([778]), 'answer_start': a...

[20 rows x 4 columns]
```

Data Cleaning and Preprocessing

This is a more tedious and challenging step, we did not attempt to open the file in a conventional spreadsheet software to avoid crashing our computers, it's generally not recommended either for very large ML projects.

So, we began coding a simple loop to start processing, and cleaning each loop, the given below is our current code, however, it is NOT complete and fails for a case about Histamine.

In the processing, we

- Aim to create a uniform structure for all q/a
- Handle edge cases such as nested arrays and quotes appropriately
- Maintain integrity for the data during transformation

We employed regex patterns for cleaning. See screenshot below

```python
import ast
import pandas as pd
import json
import re

def convert_answers(answer):
    print("Before transformation:", answer)
    # Convert NumPy array representations to lists and remove dtype info
    answer_str = str(answer).replace('array(', '[').replace(')', ']').replace('dtype=object', '').strip()
    # Ensure all keys are quoted
    answer_str = re.sub(r'([a-zA-Z_]\w*)\s*:', r'"\1":', answer_str)
    # Correctly format the text field to ensure it's treated as a string
    answer_str = re.sub(r'"text":\s*\[([^]]*?)\]', lambda m: f'"text": [["{m.group(1).strip()}"]]', answer_str)
    # Replace single quotes with double quotes for valid JSON format
    answer_str = answer_str.replace("'", "\"")
    # Strip leading/trailing whitespace
    answer_str = answer_str.strip()
    # Remove any trailing commas in lists or objects
    answer_str = re.sub(r',\s*([\]}])', r'\1', answer_str)
    # Print the transformed string for debugging
    print("Transformed answer before JSON decoding:", answer_str)

    try:
        # Load the answer as JSON
        return json.loads(answer_str)
    except json.JSONDecodeError as e:
        print(f"JSON decoding error for answer: {answer_str}")
        raise e

def process_csv(input_file, output_file):
    df = pd.read_csv(input_file)
    df['answers'] = df['answers'].apply(lambda x: convert_answers(x) if isinstance(x, str) else x)
    df.to_csv(output_file, index=False)

if __name__ == "__main__":
    input_file = 'fed.csv'
    output_file = 'transformed_file.csv'
    process_csv(input_file, output_file)
```

The code for now is still under construction, with the latest console output (with debug print statements) as

```
Before transformation: {'answer_end': array([1578]), 'answer_start': array([1553]), 'text': array(['Glyburide 5 mg p.o. q.d.,'],
dtype=object)}
Transformed answer before JSON decoding: {"answer_end": [[1578]], "answer_start": [[1553]], "text": [["Glyburide 5 mg
p.o. q.d.,"]]}
Before transformation: {'answer_end': array([432]), 'answer_start': array([356]), 'text': array(['daily, CellCept 1500 mg
b.i.d., Protonix 20 mg daily, Pravachol 40 mg daily,'],
      dtype=object)}
Transformed answer before JSON decoding: {"answer_end": [[432]], "answer_start": [[356]], "text": [["daily, CellCept 1500
mg b.i.d., Protonix 20 mg daily, Pravachol 40 mg daily,"]]}
Before transformation: {'answer_end': array([1244]), 'answer_start': array([1186]), 'text': array(['lasix, and nebs for
wheezing, and was monitored for lytes.'],
      dtype=object)}
Transformed answer before JSON decoding: {"answer_end": [[1244]], "answer_start": [[1186]], "text": [["lasix, and nebs for
wheezing, and was monitored for lytes."]]}
Before transformation: {'answer_end': array([1861]), 'answer_start': array([1810]), 'text': array(['Hematology: He received
```

```
heparin for DVT prophylaxis'],
      dtype=object)}
Transformed answer before JSON decoding: {"answer_end": [[1861]], "answer_start": [[1810]], "text": [[""Hematology": He received heparin for DVT prophylaxis"]]}
JSON decoding error for answer: {"answer_end": [[1861]], "answer_start": [[1810]], "text": [[""Hematology": He received heparin for DVT prophylaxis"]]}
Traceback (most recent call last):
...
```

We are currently debugging this pointer, but with this, we should be able to convert the entire dataset to the JSON like file we were looking for.

Model Implementation

While we prepare the data, we have begun working on the model code as well, utilizing Emily Alsentzer's ClincalBERT which has been provided to Huggingface.
It's still under development, here's some screenshots of the current work.
The code is currently on Google Colab, however, this is only to guage the metric of "current compute units used per hour", we plan to run these with AWS credits graciously provided to complete this project.The model implementation seems to be the more straightforward path

```python
model_name = 'emilyalsentzer/Bio_ClinicalBERT'
model = BertForQuestionAnswering.from_pretrained(model_name)
tokenizer = BertTokenizer.from_pretrained(model_name)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

class EMRQADataset(Dataset):
    def __init__(self, data_file, tokenizer, max_len=512):
        self.data = json.load(open(data_file))
        self.tokenizer = tokenizer
        self.max_len = max_len

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

    def __getitem__(self, idx):
        item = self.data[idx]
        question = item['question']
        context = item['context']
        answer_text = item['answer']
        start_pos = item['answer_start']

        # Tokenizing
        encoding = self.tokenizer.encode_plus(
            question, context,
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_offsets_mapping=True,
```

```
        # Find start and end token positions
        answer_end = start_pos + len(answer_text)
        offsets = encoding['offset_mapping'].squeeze().tolist()

        start_token = end_token = None
        for i, offset in enumerate(offsets):
            if offset[0] == start_pos:
                start_token = i
            if offset[1] == answer_end:
                end_token = i

        start_positions = torch.tensor(start_token)
        end_positions = torch.tensor(end_token)

        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'start_positions': start_positions,
            'end_positions': end_positions
        }

# Load dataset
train_dataset = EMRQADataset('train.json', tokenizer)
valid_dataset = EMRQADataset('valid.json', tokenizer)
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=8)

# Set optimizer and criterion
optimizer = AdamW(model.parameters(), lr=3e-5)
```

(to be further developed)

Plan Ahead

1. Finish the last few edge cases in the data processing such as the Histamine case with some regex preprocessing.
2. Complete the model and train a different configuration, such as different learning rates to get an optimal validated trained set.
3. Optional - a tool to visualize this, however, this is a last priority, optional tool as we believe the ML training modules of the project is the biggest priority to tackle.

Conclusion

We've made some good progress so far, with resilience, despite facing challenges in data processing, and even obtaining it in the first place, we're making steady progress toward our goals. Our focus on robust data preparation will provide a strong foundation for the subsequent model training phase.