Homework: PragmatiCQA with LLMs

Course: NLP with LLMs

Topic: Pragmatics, Cooperative QA

Framework: DSPy

1. Introduction and Learning Objectives

This assignment explores the area of pragmatics in Natural Language Processing. Pragmatics is the study of how context contributes to meaning and which processes speakers and hearers use to enrich literal meaning into intended meaning. Pragmatics goes beyond the literal interpretation of words (semantics) to consider speaker intent, shared knowledge, and conversational norms.

In this assignment, you will:

- Gain a deeper understanding of the process of **cooperative QA** within conversations.
- Work with a real-world academic dataset designed to test these phenomena (the *PragmatiCQA dataset* published in 2023).
- Use the DSPy framework to build and evaluate a program based on language models for a pragmatic reasoning task.
- Compare a "traditional" NLP approach (using a pre-trained QA model) with a multi-step reasoning approach using a large language model (LLM).
- Reflect on the connection between pragmatic reasoning and Theory of Mind (ToM) in AI.

2. Theoretical Background

2.1. Cooperative Question Answering

There exist multiple variants of the QA task:

- Closed QA: a question is followed by multiple options, the model must choose one of the options as an answer.
- Passage-grounded QA: a question is asked in the context of a relevant passage, the model must extract a span from the passage as an answer to the question.
- Open QA: a question is asked without context, the model must generate an answer on the basis of its parametric memory.
- Retrieval-based QA: a question is asked in the context of a document collection, the model must retrieve relevant passages and generate an answer on the basis of the relevant retrieved context.

In this assignment, we analyze a variant of retrieval-based QA which is called *cooperative QA*: in this task, the model must generate an answer based on a collection of documents, but beyond the literal answer, which provides only the information explicitly requested in the question, a cooperative model will pragmatically infer additional information from the context to answer additional questions that the user will likely ask as followup questions.

Cooperative QA is rooted in plan inference and rational speech acts (RSA) applied to textual responses. This approach is also called ”over-answering”.

The benchmark we will analyze (*PragmatiCQA*) is constructed to test the capability of models to model such pragmatic retrieval and generation. It assumes the model has access to relevant documents that the user does not know ahead of time (asymetric data access) and tests whether models can hypothesize what additional information should be retrieved beyond that explicitly requested by the literal meaning of a question, and how to present it in the answer.

It also investigates one additional question: how to predict a followup interesting question in a conversation given the beginning of the conversation.

The mechanism to infer what are useful followup questions is the same to produce pragmatically enriched cooperative answers and to predict a useful followup question, but they rely on slightly different contexts, which you will investigate.

2.2. Theory of Mind

Theory of Mind (ToM) is the ability to attribute mental states — beliefs, intents, desires, emotions, knowledge —to oneself and to others. In NLP, a model with ToM would be able to understand not just what is said, but also what the speaker believes and intends to communicate. Reasoning about pragmatic questions is a step towards this kind of sophisticated understanding.

3. The Dataset: PragmatiCQA

For this assignment, we will use the PragmatiCQA dataset. It is an open-domain conversational question answering dataset designed to test pragmatic reasoning. The task is to provide an answer to a question given a conversational history, where the desired answer often depends on understanding the unspoken needs or intent behind the question.

You can find the data and the original paper here: https://github.com/jifan-chen/PRAGMATICQA

The dataset is built in the domain of Fandom trivia (Fandom refers to a community of people who are passionate about something, whether it's a film, a band, a television show, a book, or a sports team). The dataset contains two components:

- A set of topics (like 'Batman', 'Doctor Who') and for each topic a set of documents in HTML format (a few dozens for each topic)
- A set of about 800 JSON documents (split in ~200 test, ~200 validation and ~400 training) that encode conversations about the topics.

Each JSON document has the following structure (in this dataset, the side asking the question is called "student", and the side answering "teacher", the model must implement the teacher's behavior):

```
{
    "topic": "[This is the page title in Fandom that this conversation started with, typically the main entity of that community]",
    "genre": "[This is Fandom's classification of genre for the community, usually TV, Music, Lifestyle, etc.]",
    "community": "[Community name in Fandom]"
    "qas": [ // This is the entire conversation, one question-answer pair per item
        {
            "q": "[question from the student]",
            "a_meta": {
                "literal_obj": [ // These are the spans from the corpus that
answers the question when it is interpreted literally
                    {
                        "text": "[span from the corpus]",
                        "startKey": "[element UUID from the corpus]",
                        "endKey": "[element UUID from the corpus]"
                    }
                ],
                "pragmatic_obj": [ // These are the spans from the corpus that go
beyond the literal answer to address potential follow-up questions or provide
relevant information from the corpus to keep the conversation engaging
                    {
                        "text": "[span from the corpus]",
                        "startKey": "[element UUID from the corpus]",
                        "endKey": "[element UUID from the corpus]"
                    }
                ]
            },
            "a": "[final answer combining information from both literal and
pragmatic spans, paraphrased into a consistent, natural sounding answer]",
            "human_eval": [ // Ratings from the Student of the entire Teacher
response.
            ]
        },
        ...
    ]
}
```

Each qa element within the `qas` list is one turn of conversation (one question `q` from the user and one answer `a`, together with gold retrieved content from the source documents (`literal_obj` and `pragmatic_obj`).

You will download the data and load it into your Python environment as follows:

- Create a folder 'hw3'
- In this folder:
  - Clone this repository (`git clone https://github.com/melhadad/nlp-with-llms-2025-hw3.git`)
  - Clone the PragmatiCQA repository (`git clone https://github.com/qipeng/PragmatiCQA.git`)
  - Retrieve the sources of the topics (html files) from https://drive.google.com/file/d/17vbeArdufh8rfhkg2I4Mwm0C9Og5klFR/view?usp=drive_link and unzip the collection into a subfolder `PragmatiCQA-sources`

The overall folder structure should be:

- `hw3`
  - `nlp-with-llms-2025-hw3` (this folder will contain your answers)

  - `PragmatiCQA`
    - `data` (this contains 3 jsonl files - test, train and val containing together about 800 conversations).
  - `PragmatiCQA-sources` (this contains 73 subfolders, one for each topic, each containing a few dozen html files) - this folder contains about 800MB of html files.

4. The Assignment: Step-by-Step

4.1. Setup

First, make sure you have DSPy and other necessary libraries installed:

```
# In folder nlp-with-llms-2025-hw3
uv sync
```

You will also need to configure DSPy with an LLM (a local model like ollama or the xAI model we've used in hw2 with the same API key).

You can start exploring the basis of the solution in the following notebooks:

- `pragmaticqa.ipynb` contains code demonstrating how to load the conversations from the jsonl files
- `embedder.ipynb` shows how to encode HTML files with a vector embedding and store them into a FAISS-based index which can be queried (https://github.com/facebookresearch/faiss).
- `rag.ipynb` shows how to perform basic RAG (Retrieval Augmented Generation) using DSPy over the sources in a given topic to answer questions.
- `semanticF1.ipynb` demonstrates how to evaluate the content of an answer with respect to a gold answer in terms of Precision, Recall and F1 of "conveyed information".

4.2. Part 0: Dataset Analysis

Before you begin coding, read the original paper that introduces the PRAGMATICQA dataset. In a few paragraphs:

- Summarize the key motivations and contributions of the paper.
- Explain in a qualitative manner what makes this dataset challenging for NLP models. What specific pragmatic phenomena does it target?
- Select a few (about 5) sample conversations from the dataset (from different topics) and explain how the pragmatic answer enriches the literal answer that would be produced by a non-cooperative teacher.

4.3. Part 1: The "Traditional" NLP Approach

In this part, you will use a pre-trained Question Answering model as a "traditional" baseline.

Use the retriever module from the `rag.ipynb` notebook to retrieve relevant passage given a question. Concatenate all the elements into a single context.

Write a program that uses a pre-trained QA model from Hugging Face's transformers library to generate an answer to the question given the context retrieved by the retriever module. A good model to use for this is 'distilbert-base-cased-distilled-squad' (see https://huggingface.co/distilbert/distilbert-base-cased-distilled-squad). This module will extract an answer from the context without explicit multi-step reasoning.

Evaluate on PRAGMATICQA: Run your model on the PRAGMATICQA test set. Since the answers are free-form, you need to use a metric like F1 score or ROUGE to evaluate performance. In this assignment, we will use the SemanticF1 metric provided in DSPy which uses an LLM as a Judge method.

Since the performance of the model depends on the accuracy of the retriever module, we will compute three different configurations:

1. Literal answer: the answer generated by the distilbert model from the literal spans included in the dataset.
2. Pragmatic answer: the answer generated by the distilbert model from the pragmatic spans included in the dataset.
3. Retrieved answer: the answer generated by the distilbert model from the context computed by the retriever.

For each of these three configurations, report precision, recall and F1 as computed by SemanticF1 on the validation dataset for the first question of each conversation only (there are 179 such cases in val.jsonl). For the first question in each conversation, there is no "conversational context", hence the input to the model only includes the question and the retrieved passages.

Note: To improve performance, consider using the `SemanticF1.batch` method to perform the dataset evaluation (https://dspy.ai/api/evaluation/SemanticF1/) or the general `dspy.Evaluate` evaluation method which runs evaluation in parallel and outputs a tabular report.

Analyze the results: where does the model succeed, and where does it fail? Does it tend to give literal answers when a more pragmatic one is needed?

4.4. Part 2: The LLM Multi-Step Prompting Approach

Now, you will build a more sophisticated model using an LLM with multi-step prompting.

We will now evaluate all the questions in the conversations, not only the first question of each conversation as in 4.3.

In each turn, the model will have access as input to the following:

- The previous turns as pairs (question, answer)
- The current question.
- The context retrieved by the retriever model given the current question.

Your model can enrich this by computing additional intermediary fields, for example:

- A summary of the student's goal or interests based on the conversation history
- A pragmatic or cooperative need underlying the student's current question (based on the past conversation and retrieved spans)
- A generated "cooperative" question which can be used to re-query the source documents and extract additional context
- A reasoning field computed by a Chain-of-Thought module for any of these intermediary steps

Implement the DSPy Module: Create a DSPy module that uses the strategy you devise to generate a cooperative answer.

For reference, you can start from the DSPy tutorials demonstrating variations around RAG:

- https://dspy.ai/tutorials/rag/ (using a retriever based on FAISS and passage embeddings)
- https://dspy.ai/tutorials/multihop_search/ (using a retriever based on BM25s)
- https://dspy.ai/tutorials/agents/ (using a ReACT module with a ColbertV2 retriever)

4.4.1 First Questions

Perform the same evaluation as in 4.3 on the first questions in each conversation and compare the results of your model with the one in 4.3 based on the traditional text-to-text transformer.

4.4.2 Conversational Context

Now consider all questions in the conversations and take into account the conversational context.

Compile and Evaluate: Compile your DSPy program (you can use a small training set from the PRAGMATICQA data for this) and evaluate it on the validation set using the same metrics as in 4.3. Explain which metric you use to drive the optimization.

5. Discussion Questions

Please include your answers to these questions in your submission.

- **Comparison of Models**: How did the performance of the "traditional" QA model compare to the LLM-based model? What are the strengths and weaknesses of each approach for this pragmatic QA task? Do you observe a difference between the first question in a conversation and later questions?

- **Theory of Mind**: To what extent do you think your LLM-based model exhibits a "*Theory of Mind*"? Does it truly "understand" the speaker's intent, or is it performing sophisticated pattern matching? Justify your answer with examples from your experiments.

- **Optional**: The end-to-end performance of the model depends on both the quality of the retriever model (which extracts relevant passages given a question) and the generator (which generates an answer given a context returned by the retriever). The following are ways to investigate the impact of the retriever performance:

    - Experiment with the `ColbertV2` model available in DSPy (https://dspy.ai/api/tools/ColBERTv2/) as an improved retriever.
    - Experiment with the influence of the parameter `top-k` which determines the number of relevant passages returned by the retriever
    - Experiment with different chunking methods (that is, how do you split the source documents into separate chunks which can be retrieved separately). Determine the size of chunks and ways to exploit the HTML structure to segment source documents into chunks.

6. Submission

Please submit a single Jupyter notebook containing:

- Your written answers for Part 0.
- Your Python code for all subsequent parts of the assignment.
- The output of your evaluations.
- Your answer to Part 5.

Your written answers to the discussion questions.

Good luck!