POS Tagging with LLMs: The Hard Parts

NLP with LLMs - BGU CS - Michael Elhadad - Spring 2025 - HW1

We will consider a subset of the dataset of English Universal Dependency provided in https://github.com/UniversalDependencies/UD_English-EWT The dataset is encoded in text files with a standard format called 'CoNLL-U' (https://universaldependencies.org/format.html). The dataset can be loaded into Python in a convenient manner using the pyconll library https://pyconll.readthedocs.io/en/stable/starting.html.

As discussed in class, the Universal Dependency tagset includes 17 tags, as documented in https://universaldependencies.org/u/pos/index.html.

In this assignment, we will focus on understanding what are the 'hard' parts of the POS tagging task and investigate how LLMs can help us deal with complex cases. The following steps will be covered:

- Explore the dataset by computing simple statistics and training a very simple statistics-based tagger (as seen in class).
- Train a classical tagger by using features engineering and a standard machine learning algorithm (Logistic Regression). We will use the standard scikit-learn library to implement this classical POS tagger. It achieves about 95% accuracy on the task as measured on the UD benchmark.
- Analyze errors: we will investigate the errors made by this tagger and make hypotheses about the root cause of these errors.
- Test a simple LLM-based tagger using zero-shot prompting and measure its performance on the 'hard' part of the test dataset.
- Extend the LLM-based tagger by employing 'structured output' mode (force the LLM to produce a JSON format according to the constraints we define).
- Investigate word segmentation issues and drive the LLM to segment according to the UD segmentation guidelines using few-shot prompting.

The expected learning outcomes of this assignmen are:

1. Learn about word segmentation, word classification.
2. Classical ML techniques for sequence tagging using feature extraction and logistic regression.
3. Evaluation methods for sequence classification: metrics (acuracy, precision, recall, F1, sequence-level metrics) and error analysis.
4. API interaction with LLM providers.
5. Using structured output mode for LLM completion.
6. Few-shot prompting techniques.

1. Classical Machine Learning Approach to Sequential Tagging

1.1. Execute the Logistic Regression Notebook

Install a local Python environment and run the notebook ud_pos_tagger_sklearn.ipynb.

This notebook implements two baseline taggers for the UD POS Tagging task: one based on simple statistics (as discussed in class) and one based on classical machine learning using the Logistic Regression

algorithm and feature extraction (LR Tagger).

I recommend using "uv" instead of pip for working on the project (it is much faster - see https://docs.astral.sh/uv/getting-started/). I use the notebook environment embedded in Visual Studio Code and select the virtual env created by uv as a kernel to run the notebook.

1.2. Investigate Tagging Errors of the LR Tagger

Consider the Logistic Regression tagger created in the notebook. It obtains an accuracy of about 95% on the test part of the UD data.

1. Add code to the notebook to collect all the sentences containing at least one tagging error together with the correct tags and predicted tags from the test data of the UD_English-EWT dataset. This file will be used as the "hard sentences" dataset in subsequent questions.
2. Create a histogram showing the number of sentences containing each level of errors in the hard_sentences list.
3. Identify tokens that are misclassified more than 10 times in the test set. Print the sentences where the errors are predicted (about 100 sentences).
4. Provide a possible reason why these errors are made by the tagger based on your understanding of the knowledge needed to correctly tag these tokens.
5. Based on this error analysis, invent five sentences that are badly tagged. Explain what is your method to create these hard examples.

The answers should be added to your version of the notebook file with the explanations in markdown cells.

2. Create an LLM Baseline for POS Tagging

2.1 Zero-Shot Prompting

Create a simple zero-shot prompt to instruct the LLM to tag a list of sentences with the UD tagset.

First experiment in the online chat interfaces for different LLM API providers:

- https://gemini.google.com/app (Select Gemini 2.0 Flash for example)
- https://grok.com/

You can use another provider if you already have an API key to use it and it is compatible with the de-facto standard OpenAI API.

Then implement Python API-based code to execute the calls to the LLM API in an automated manner. Simple starting points are provided in ud_pos_tagger_gemini.py and ud_pos_tagger_grok.py.

For Gemini, you should register a free API Key (that comes with strong rate limitations but is sufficient to perform this assignment on the free tier). Follow the instructions in https://ai.google.dev/gemini-api/docs/api-key.

Pay attention to rate limitations on the accounts:

- RPM: Requests per minute
- TPM: Tokens per minute
- RPD: Requests per day

With gemini-2.0-flash on the free tier, the limits are:

- RPM 15
- TPM 1,000,000
- RPD 1,500

You should batch multiple sentences into a single prompt to avoid rate limitations (RPM and RPD). Experiment on the interactive (free) console to identify the maximum size of the batch you can use. (In my tests with Gemini, I used a batch of 5 sentences for each request). With Grok I experimented with 10 to 20 sentences per request.

With Grok-3-mini, I have the following limits:

- Limit: 5 requests per second
- Context: 131,072 tokens
- Text input: $0.30 per million
- Text output: $0.50 per million
- model = 'grok-3-mini'

I will discuss how to obtain pre-paid API keys for Grok in class.

Prompting with Structured Output (JSON Mode)

In our calls to the LLM API, we would like to obtain results that follow a well-defined structure (as opposed to free text). This will allow us to constrain the output of the LLM and ease parsing of the results according to our expectations.

Most LLM APIs support a form of "structured output" call. This mode lets you define schemas using tools like Pydantic to enforce data types, constraints, and structure.

- For Gemini, see the API https://ai.google.dev/gemini-api/docs/structured-output?lang=python
- For Grok, see the API https://docs.x.ai/docs/guides/structured-outputs#structured-outputs (This approach is supported by most providers who support the OpenAI API).

2.2 Error Analysis

Compare the errors for the two taggers: Logistic Regression and the LLM-based with structured output on the hard sentences list of sentences that have 1 to 3 errors each (we ignore sentences that have many more errors as these cases are probably caused by a complete break-down of the expectations of the tagger).

1. Compute the same statistics reported for the Logistic Regression Tagger in the notebook for the LLM Tagger (token-level metric).
2. How many of the Logistic Regression Tagger errors are fixed by the LLM Tagger
3. How many errors are made by the LLM Tagger that were not made by the Logistic Regression Tagger
4. Report sentence-level error metrics: show the histogram of hard sentences with 1,2, and more errors using the LLM Tagger. Compare it to the histogram with the Logistic Regression Tagger.
5. Make hypotheses about what cases are difficult for the LLM Tagger and create new sentences that make it fail.

2.3 Error Explanation and Synthetic Data

Given a set of errors that are made by the taggers (both the LR and the LLM one), and a definition of the POS tagging task, we want to exploit knowledge in the LLM to explain what can be the cause of observed tagging errors.

Here is an example of error analysis we may obtain with the proper prompt on the following error case:

```
WE HAVE A DATE FOR THE RELEASE OF RAGNAROK ONLINE 2 ( beta anyway )
September 16 – 18 , this was announced by Gravity CEO Kim Jung – Ryool on
either 16th or 17th of july and as i do n't want to take someone 's
credit's i got it here ^^ GameSpot
Number of errors: 7
WE                      PRON
HAVE                    VERB
A                       DET
DATE                    NOUN
FOR                     ADP
THE                     DET
RELEASE                 NOUN
OF                      ADP
RAGNAROK                PROPN
ONLINE                C: ADJ          P: NOUN         **** Error
2                       NUM
(                       PUNCT
beta                    NOUN
anyway                  ADV
)                       PUNCT
September               PROPN
16                      NUM
–                     C: SYM          P: PUNCT        **** Error
18                      NUM
,                       PUNCT
this                    PRON
was                     AUX
announced               VERB
by                      ADP
Gravity               C: PROPN        P: NOUN         **** Error
CEO                     NOUN
Kim                     PROPN
Jung                    PROPN
–                       PUNCT
Ryool                   PROPN
on                      ADP
either                  CCONJ
16th                  C: NOUN         P: ADJ          **** Error
or                      CCONJ
17th                  C: NOUN         P: ADJ          **** Error
of                      ADP
july                  C: PROPN        P: ADV          **** Error
and                     CCONJ
as                      SCONJ
i                       PRON
do                      AUX
```

```
n't                     PART
want                    VERB
to                      PART
take                    VERB
someone                 PRON
's                      PART
credit's                NOUN
i                       PRON
got                     VERB
it                      PRON
here                    ADV
^^                  C: SYM          P: PUNCT        **** Error
GameSpot                PROPN
```

JSON


```
[
  {
    "word": "ONLINE",
    "correct_tag": "ADJ",
    "predicted_tag": "NOUN",
    "explanation": "The word 'ONLINE' is often used as an adjective
modifying a noun like 'RAGNAROK'. However, it can also function as a noun
in other contexts (e.g., 'I'll be online'). Without strong surrounding
grammatical cues in this specific phrase ('RAGNAROK ONLINE 2'), a tagger
might default to the more common noun form, especially if 'RAGNAROK' is
treated as a proper noun, leaving 'ONLINE' as the head of a noun phrase.",
    "category": "Ambiguity (ADJ/NOUN)"
  },
  {
    "word": "–",
    "correct_tag": "SYM",
    "predicted_tag": "PUNCT",
    "explanation": "The hyphen in 'September 16 – 18' functions as a
symbol indicating a range. While hyphens often serve as punctuation to
join words or separate clauses, their use in numerical ranges or as part
of compound terms can lead to ambiguity for a tagger, potentially
misclassifying it as general punctuation rather than a specific symbol.",
    "category": "Punctuation vs Symbol"
  },
  {
    "word": "Gravity",
    "correct_tag": "PROPN",
    "predicted_tag": "NOUN",
    "explanation": "'Gravity' is the name of a specific company (Gravity
Corp.), making it a proper noun in this context. However, 'gravity' is
also a common noun referring to the physical force. Unless the tagger has
specific knowledge or strong contextual clues (like surrounding proper
nouns or capitalization patterns) to identify it as part of a named entity
('Gravity CEO Kim Jung-Ryool'), it might default to the more frequent
common noun tag.",
```

```
    "category": "Proper Noun vs Common Noun"
  },
  {
    "word": "16th",
    "correct_tag": "NOUN",
    "predicted_tag": "ADJ",
    "explanation": "Ordinal numbers like '16th' can function as adjectives
(e.g., 'the 16th day') or as nouns when referring to a specific day or
item in a sequence (e.g., 'on the 16th'). In the phrase 'on either 16th or
17th of july', '16th' and '17th' are acting as nouns specifying the dates.
The tagger likely saw the ordinal form and defaulted to the more common
adjective tag.",
    "category": "Numeral/Ordinal Ambiguity"
  },
  {
    "word": "17th",
    "correct_tag": "NOUN",
    "predicted_tag": "ADJ",
    "explanation": "Similar to '16th', '17th' is an ordinal number that
can be an adjective or a noun. In 'on either 16th or 17th of july', it
functions as a noun referring to a specific date. The tagger probably
assigned the adjective tag based on the ordinal form, overlooking its noun
function in this context.",
    "category": "Numeral/Ordinal Ambiguity"
  },
  {
    "word": "july",
    "correct_tag": "PROPN",
    "predicted_tag": "ADV",
    "explanation": "'July' is a proper noun as it's the name of a specific
month and is capitalized (though the input shows lowercase 'july', which
can confuse taggers reliant on capitalization). It functions here within a
prepositional phrase indicating time ('of july'). The misclassification as
an adverb is unusual and could stem from the tagger incorrectly
interpreting the phrase 'of july' as an adverbial modifier, or it might be
a less common error pattern related to temporal expressions or lowercase
proper nouns.",
    "category": "Proper Noun Misclassification"
  },
  {
    "word": "^^",
    "correct_tag": "SYM",
    "predicted_tag": "PUNCT",
    "explanation": "The emoticon '^^' is a symbol used for conveying
emotion. While it's not traditional punctuation, in informal text, it
functions similarly by adding expressive meaning outside the core
grammatical structure. Taggers are often trained on formal text and may
not have specific rules for emoticons, leading them to categorize them
under a general 'punctuation' tag rather than a more specific 'symbol'
tag.",
    "category": "Punctuation vs Symbol"
  }
]
```

1. Collect explanations generated by the LLM on a subset of errors.
2. Collect the list of error categories provided by the explanation of errors.
3. [OPTIONAL] Use the LLM to generate examples of hard sentences that include 2 to 3 instances of the error categories. Collect about 200 synthetic sentences. Ask the LLM to provide correct tagging for each of the generated sentence.
4. [OPTIONAL] Add the synthetic data to the training dataset of the LR tagger and re-run training.
5. [OPTIONAL] Perform contrastive error analysis between the two versions of the LR tagger.

3. Dealing with Segmentation

In the evaluation we performed with the LLM Tagger, we fed the input as a pre-segmented string. For example, if you consider a sentence from the UD English EWT dataset, we use the "untag" version of the sentence as in:

```
untag(test_sentences[1])
['What',
 'if',
 'Google',
 'expanded',
 'on',
 'its',
 'search',
 '-',
 'engine',
 '(',
 'and',
 'now',
 'e-mail',
 ')',
 'wares',
 'into',
 'a',
 'full',
 '-',
 'fledged',
 'operating',
 'system',
 '?']

 " ".join(untag(test_sentences[1]))
'What if Google expanded on its search - engine ( and now e-mail ) wares
into a full - fledged operating system ?'

test_original[1]
What if Google expanded on its search-engine (and now e-mail) wares into a
full-fledged operating system?
```

The difference consists in passing these strings to the LLM as input:

- CoNLL Tokenized:
  - *What if Google expanded on its search - engine (and now e-mail) wares into a full - fledged operating system ?*
- Original sentence:
  - *What if Google expanded on its search-engine (and now e-mail) wares into a full-fledged operating system?*

1. Test the performance of the LLM tagger on the original sentences. Do you need to update the evaluation metrics for this new setting? Collect sentences that fail with original sentences and succeeded with the tokenized version.
2. Prepare an LLM segmenter model: given an original sentence, it must return a tokenized list of tokens according to the CoNLL segmentation guidelines in a JSON format. The tokenization and word segmentation guidelines used in English for the Universal Dependency annotations are given in https://universaldependencies.org/en/index.html. Define evaluation metrics for this task and report the performance on the set of sentences where tagging failed in the evaluation above.
    1. Use a few-shots strategy for this task: give a few examples (up to 3) of input/output pairs (original sentence, segmented sentence) and then ask the model to segment based on these examples. In this strategy, you add the instruction in the "system" part of the invocation, and the examples in the "user" part.
    2. [OPTIONAL] It helps to select examples that are as similar as possible to the sentence to be segmented. The way to implement this "example selection" means that you prepare a list of "good examples" to illustrate how to deal with complex cases; given the sentence to be segmented, you select the top-3 most similar sentences based on a "similarity" function which identifies "relevant" examples. Implement this strategy and explain your assumptions about "relevance" in this context.
3. Prepare an improved LLM Tagger that performs segmentation and POS tagging according to the improved segmentation.
    1. [OPTIONAL] You can compare two strategies: pipeline and joint tagging. In the pipeline method, you first use the segmentatiojn model and then pass the segmented sentence to the tagger. In the joint tagging method you perform both steps at once. Compare the two strategies on a set of "hard sentences". Explain your observations.