Claude 4 Opus chat on: Stage 1.1 - Prepare gold standard sheet

User:

NEReus /

User:

metadata: BB

Hi Claude. Today we're going to be working on some research around named entity recognition in Latin inscriptions. To begin testing, we're going to want to evaluate how various part of speech taggers do with inscriptions.

We now have a json for our step 1 to target our part of speech tagging tools against. However, we're going to need to validate it on a gold standard.

What I'd like is for us to have a script to take the text_interpretive_word from each of these json, split it by space, then have them be one inscription per sheet until we have a spreadsheet with 20 sheets. Optimally, we'd have a constrained vocabulary for part of speech to ease data entry.

Functionally decompose the task at hand, and then ask us one question at a time until you think you know enough to make this script.

Assistant:

Looking at this task, I need to understand your requirements better to create the most useful script for your Latin inscription POS tagging validation. Let me break this down functionally:

Functional Decomposition:

1. **Data extraction**: Parse JSON and extract `text_interpretive_word` fields
2. **Text processing**: Split texts into individual words/tokens
3. **Spreadsheet creation**: Generate multi-sheet workbook with one inscription per sheet
4. **Data validation setup**: Implement constrained POS vocabulary
5. **Output formatting**: Save in appropriate format for annotation

Now, let me ask you some questions to ensure I build exactly what you need:

Question 1: POS Tagset Selection

What part-of-speech tagset would you like to use for the constrained vocabulary? The main options are:

- **Universal Dependencies (UD)** tagset - widely used, has 17 universal tags (NOUN, VERB, ADJ, etc.)
- **Latin-specific tagset** - like the one used by the Latin Dependency Treebank or PROIEL
- **Custom simplified tagset** - if you have a specific set in mind for this project

Which would work best for your validation purposes?

User:

metadata: BB

let's go with UD:

home issue tracker This page pertains to UD version 2. Universal POS tags These tags mark the core part-of-speech categories. To distinguish additional lexical and grammatical properties of words, use the universal features. Open class wordsClosed class wordsOtherADJADPPUNCTADVAUXSYMINTJCCONJXNOUNDET PROPNNUM VERBPART PRON SCONJ Alphabetical listing * ADJ: adjective * ADP: adposition * ADV: adverb * AUX: auxiliary * CCONJ: coordinating conjunction * DET: determiner * INTJ: interjection * NOUN: noun * NUM: numeral * PART: particle * PRON: pronoun * PROPN: proper noun * PUNCT: punctuation * SCONJ: subordinating conjunction * SYM: symbol * VERB: verb * X: other © 2014–2024 Universal Dependencies contributors. Site powered by Annodoc and brat .

Assistant:

Perfect! We'll use the UD tagset with those 17 universal POS tags. That's a great choice for standardization and compatibility with other tools.

Question 2: Spreadsheet Format & Functionality

For the spreadsheet structure, I'm thinking of creating sheets where each word gets its own row with columns for:

- **Word**: The token from the inscription
- **POS**: Dropdown/data validation with the 17 UD tags

Would you also like additional columns such as:

- **Index**: Word position in the inscription
- **Notes**: Free text field for annotator comments
- **Confidence**: (High/Medium/Low) for uncertain cases
- **Lemma**: Base form of the word (if needed)

Or would you prefer to keep it minimal with just Word and POS columns?

User:

metadata: BB

word, POS, notes.

Assistant:

Excellent - keeping it clean with Word, POS, and Notes columns.

Question 3: Output File Format

What spreadsheet format would you prefer for the output?

- **Excel (.xlsx)** - Best for dropdown data validation, works well with Excel/Google Sheets
- **CSV files** - Simple but would lose the dropdown functionality (though could document the valid tags separately)
- **Google Sheets API** - Direct creation in Google Sheets with data validation already set up

Which would be most convenient for your annotation workflow?

User:

metadata: BB

xlsx

2 / 2

Assistant:

Perfect! I have everything I need. Let me create a Python script that will:

1. Load your JSON file
2. Take the first 20 inscriptions
3. Create an Excel workbook with 20 sheets
4. Each sheet will have the words from one inscription with dropdown validation for POS tags
5. Include Word, POS, and Notes columns

I've created a comprehensive Python script that will generate your POS tagging validation spreadsheet. Here's what it does:

Key Features:

1. **Creates an Excel workbook** with 20 sheets (one per inscription)
2. **Each sheet contains**:
   - Headers: Word, POS, Notes
   - All words from the `text_interpretive_word` field split by spaces
   - Dropdown validation for POS column with all 17 UD tags
   - LIST-ID metadata for reference
   - Instructions and POS tag reference at the bottom
3. **Summary sheet** at the beginning with:
   - Overall instructions
   - List of all included inscriptions with their LIST-IDs
4. **Data validation** ensures only valid UD POS tags can be selected from a dropdown menu

How to use:

User:

bash

```
# Basic usage (assuming your JSON is named POS-LIST-test1.json)
python latin_pos_tagger.py POS-LIST-test1.json

# Specify custom output file
python latin_pos_tagger.py POS-LIST-test1.json -o my_validation.xlsx

# Create fewer sheets if needed
python latin_pos_tagger.py POS-LIST-test1.json -n 10
```

Requirements:

You'll need to install openpyxl:

User:

bash

```
pip install openpyxl
```

The script will create a user-friendly Excel file where annotators can:

- Click on the POS column to get a dropdown with all valid UD tags
- Add notes for uncertain cases
- Work through one inscription at a time on separate sheets

This should make your validation process much more efficient and reduce data entry errors with the constrained vocabulary!

Claude can make mistakes.
Please double-check responses.

Choose Files No file chosen

metadata:

```
#!/usr/bin/env python3
"""
Latin Inscription POS Tagging Validation Spreadsheet Generator

This script creates an Excel workbook with 20 sheets, each containing words from
a Latin inscription ready for POS tagging with UD (Universal Dependencies) tags.
"""

import json
import openpyxl
from openpyxl import Workbook
from openpyxl.worksheet.datavalidation import DataValidation
from openpyxl.styles import Font, PatternFill, Alignment
from openpyxl.utils import get_column_letter
import argparse
from pathlib import Path

# Universal Dependencies POS tags
UD_POS_TAGS = [
    "ADJ",    # adjective
    "ADP",    # adposition
    "ADV",    # adverb
    "AUX",    # auxiliary
    "CCONJ",  # coordinating conjunction
    "DET",    # determiner
    "INTJ",   # interjection
    "NOUN",   # noun
    "NUM",    # numeral
    "PART",   # particle
    "PRON",   # pronoun
    "PROPN",  # proper noun
    "PUNCT",  # punctuation
    "SCONJ",  # subordinating conjunction
    "SYM",    # symbol
    "VERB",   # verb
    "X"       # other
]
```

```
def load_inscriptions(json_path):
    """Load inscriptions from JSON file."""
    with open(json_path, 'r', encoding='utf-8') as f:
        data = json.load(f)
    return data

def create_sheet_for_inscription(ws, inscription_data, sheet_index):
    """
    Create a sheet for a single inscription with POS tagging setup.

    Args:
        ws: openpyxl worksheet object
        inscription_data: dict containing inscription data
        sheet_index: index of the sheet (for naming)
    """
    # Set up headers with formatting
    headers = ["Word", "POS", "Notes"]
    header_font = Font(bold=True)
    header_fill = PatternFill(start_color="DDDDDD", end_color="DDDDDD", fill_type="solid")

    for col, header in enumerate(headers, 1):
        cell = ws.cell(row=1, column=col, value=header)
        cell.font = header_font
        cell.fill = header_fill
        cell.alignment = Alignment(horizontal="center")

    # Extract and process text
    text = inscription_data.get("text_interpretive_word", "")
    list_id = inscription_data.get("LIST-ID", "Unknown")

    # Add metadata at the top of the sheet
    ws.cell(row=1, column=5, value=f"LIST-ID: {list_id}")
    ws.cell(row=1, column=5).font = Font(italic=True)

    # Split text into words and populate rows
    if text and text.strip():
        words = text.split()

        for row_idx, word in enumerate(words, 2):  # Start from row 2 (after headers)
            # Word column
            ws.cell(row=row_idx, column=1, value=word)

            # POS column - will have dropdown
            ws.cell(row=row_idx, column=2, value="")
        # Notes column - empty for annotation
        ws.cell(row=row_idx, column=3, value="")

    # Create data validation for POS column
    # Formula for dropdown list
    pos_list = ",".join(UD_POS_TAGS)
    dv = DataValidation(
        type="list",
        formula1=f'"{pos_list}"',
        allow_blank=True,
        showDropDown=True,
        showErrorMessage=True,
        errorTitle="Invalid POS Tag",
        error="Please select a valid UD POS tag from the dropdown."
    )

    # Apply to all POS cells (column B, starting from row 2)
    if words:
        dv.add(f"B2:B{len(words)+1}")
    ws.add_data_validation(dv)

    # Adjust column widths
    ws.column_dimensions['A'].width = 20  # Word column
    ws.column_dimensions['B'].width = 12  # POS column
    ws.column_dimensions['C'].width = 30  # Notes column
    ws.column_dimensions['E'].width = 15  # Metadata column

    # Add instructions at the bottom
    instruction_row = len(words) + 4 if words else 4
    ws.cell(row=instruction_row, column=1, value="Instructions:")
    ws.cell(row=instruction_row, column=1).font = Font(bold=True, italic=True)
    ws.cell(row=instruction_row + 1, column=1,
        value="Select POS tags from the dropdown in column B. Add any notes or uncertainties in column C.")
    ws.merge_cells(f"A{instruction_row + 1}:E{instruction_row + 1}")

    # Add POS tag reference
    ref_row = instruction_row + 3
    ws.cell(row=ref_row, column=1, value="UD POS Tags Reference:")
    ws.cell(row=ref_row, column=1).font = Font(bold=True)

    tag_descriptions = {
        "ADJ": "adjective", "ADP": "adposition", "ADV": "adverb", "AUX": "auxiliary",
        "CCONJ": "coordinating conjunction", "DET": "determiner", "INTJ": "interjection",
        "NOUN": "noun", "NUM": "numeral", "PART": "particle", "PRON": "pronoun",
        "PROPN": "proper noun", "PUNCT": "punctuation", "SCONJ": "subordinating conjunction",
        "SYM": "symbol", "VERB": "verb", "X": "other"
    }

    for i, (tag, desc) in enumerate(tag_descriptions.items()):
        ws.cell(row=ref_row + 1 + i, column=1, value=f"{tag}: {desc}")

def create_pos_tagging_workbook(json_path, output_path, num_sheets=20):
    """
    Create an Excel workbook for POS tagging validation.

    Args:
        json_path: Path to the input JSON file
        output_path: Path for the output Excel file
        num_sheets: Number of inscription sheets to create (default 20)
    """
    # Load data
    inscriptions = load_inscriptions(json_path)

    # Create workbook
    wb = Workbook()

    # Remove default sheet
    wb.remove(wb.active)

    # Process inscriptions (up to num_sheets)
    for i in range(min(num_sheets, len(inscriptions))):
        inscription = inscriptions[i]
        list_id = inscription.get("LIST-ID", f"Unknown_{i+1}")

        # Create sheet with truncated name if necessary
        sheet_name = f"Insc_{i+1:02d}_ID{list_id}"
        if len(sheet_name) > 31:  # Excel sheet name limit
            sheet_name = f"Insc_{i+1:02d}"

        ws = wb.create_sheet(title=sheet_name)
        create_sheet_for_inscription(ws, inscription, i)

    # Add a summary sheet at the beginning
    summary = wb.create_sheet(title="Summary", index=0)
    summary.cell(row=1, column=1, value="Latin Inscription POS Tagging Validation")
    summary.cell(row=1, column=1).font = Font(bold=True, size=14)
    summary.cell(row=3, column=1, value="Instructions:")
    summary.cell(row=3, column=1).font = Font(bold=True)
    summary.cell(row=4, column=1, value="1. Each sheet contains one inscription with words split into rows")
    summary.cell(row=5, column=1, value="2. Select the appropriate UD POS tag from the dropdown in the POS colum
    summary.cell(row=6, column=1, value="3. Add any notes, uncertainties, or comments in the Notes column")
    summary.cell(row=7, column=1, value="4. Save the file regularly to preserve your work")

    summary.cell(row=9, column=1, value="Inscriptions included:")
    summary.cell(row=9, column=1).font = Font(bold=True)

    for i in range(min(num_sheets, len(inscriptions))):
        inscription = inscriptions[i]
        list_id = inscription.get("LIST-ID", "Unknown")
        text_preview = inscription.get("text_interpretive_word", "")[:50] + "..."
        summary.cell(row=10+i, column=1, value=f"Sheet {i+1:02d}: LIST-ID {list_id}")
        summary.cell(row=10+i, column=2, value=text_preview)

    summary.column_dimensions['A'].width = 25
    summary.column_dimensions['B'].width = 60

    # Save workbook
    wb.save(output_path)
    print(f"Created Excel workbook: {output_path}")
    print(f"Generated {min(num_sheets, len(inscriptions))} inscription sheets")

def main():
    parser = argparse.ArgumentParser(description='Generate POS tagging validation spreadsheet for Latin inscriptions')
    parser.add_argument('input_json', help='Path to the input JSON file')
    parser.add_argument('-o', '--output', default='pos_tagging_validation.xlsx',
                        help='Output Excel file path (default: pos_tagging_validation.xlsx)')
    parser.add_argument('-n', '--num-sheets', type=int, default=20,
                        help='Number of inscription sheets to create (default: 20)')

    args = parser.parse_args()

    # Check if input file exists
    if not Path(args.input_json).exists():
        print(f"Error: Input file '{args.input_json}' not found")
        return 1

    try:
        create_pos_tagging_workbook(args.input_json, args.output, args.num_sheets)
        return 0
    except Exception as e:
        print(f"Error creating workbook: {e}")
        return 1

if __name__ == "__main__":
    exit(main())
```