Homework 3: Syntactic Complexity and Model Robustness

CS 490: Natural Language Processing · **Spring 2026**

Due on: 03/13/2026 @ 11:59 PM (AoE)

Overview

In this assignment, you will investigate how the syntactic complexity of input text impacts the performance of various NLP models. You will define metrics to measure syntactic complexity of text, apply meaning-preserving perturbations to a standard dataset to increase its syntactic complexity, and evaluate how different model architectures handle these changes.

This is an open-ended assignment where you have the freedom to design your own metrics and select your models. Treat it as a mini-project that mirrors the format of the final project. You must submit a 3–4 page report with references using the provided template. All figures must be created **digitally**. You must also include all code needed to reproduce your results. Failure to use the provided template will result in point deduction.

Part 1: Measuring Syntactic Complexity

Your first task is to programmatically measure the syntactic complexity of a given text.

Examples of Complexity Metrics

To get you started, here are two examples of structural metrics you can use:

- **CFG Tree Depth:** The maximum depth of the context-free grammar (CFG) constituency parse tree for a given sentence. Deeper trees generally indicate more nested and complex sentence structures.
- **Subject-Verb Distance:** The number of words (or edges in a dependency parse) separating the main verb (V) of a clause and its subject noun (NP). For example, in the sentence "The person in the observatory looked at the moons of Jupiter," the main verb is "looked" and its subject noun is "person," and we can define the subject-verb distance as the distance between "looked" and "person" in the parse tree of the sentence. Longer dependencies often make sentences harder to process.

Your Task

Use a parsing library of your choice (e.g. SpaCy, Stanza) to implement at least three (3) complexity metrics. You may implement the examples provided above or design your own. In your final report, justify why your chosen metrics are valid indicators of syntactic complexity.

Part 2: The Dataset

We will use the **Multi-Genre Natural Language Inference (MultiNLI)** dataset for this assignment. MultiNLI is a standard benchmark for evaluating natural language understanding. Recall that, in the NLI task, each example consists of a *premise* and a *hypothesis*, and the task is to classify the example into one of three cases: (1) the hypothesis is logically entailed by the premise, (2) the hypothesis contradicts the premise, and (3) neither. For example, if the premise is "Alex is a cat" and the hypothesis is "Alex is a mammal", the label is `entailment`. If instead the hypothesis is "Alex is a dog", the label is `contradiction`. If the hypothesis is "Alex is sleeping", the label is `neutral`.

- You may use a subset of the data (e.g., ≥15,000 premise-hypothesis pairs from the **dev** split) to reduce compute time, though you are permitted to use the entire dev set if you prefer. When constructing your dataset, you must sample an equal number of examples from both the `matched` and `mismatched` dev sets. Ensure that the label distribution of your subset closely matches that of the original dataset (e.g., by sampling uniformly at random). Additionally, you must keep track of the source split (*matched* vs. *mismatched*) for each pair throughout your evaluation.

- Select at least four (4) distinct model architectures (e.g., MLP, RNN, encoder-only, or decoder-only Transformers). You must use **pre-trained models** for all selected architectures. For simpler architectures like an MLP, ensure you find a model that is specifically trained or fine-tuned on the MultiNLI dataset. Establish a baseline accuracy for each model on your unperturbed subset before proceeding to Part 3. In your report, you must discuss the pros and cons of each chosen architecture.

Part 3: Meaning-Preserving Perturbations

To test model robustness, you will systematically perturb the dataset's text to increase its syntactic complexity *without* altering the underlying meaning or the ground-truth NLI label (`entailment`, `contradiction`, `neutral`).

Examples of Perturbations

You can introduce complexity by injecting new syntactic structures. Here are a few ways to do this:

- **Adding Relative Clauses:** Identify adjectival modifiers attached to nouns, extract the adjective, and automatically rewrite the phrase as a relative clause.

  - *Original:* "The wealthy investor..."
  - *Perturbed:* "The investor, who is wealthy, ..."

  Another approach is to add new relative clauses that contains irrelevant information, such as:

  - *Original:* "The wealthy investor..."
  - *Perturbed:* "The wealthy investor, who is related to Sam's older sister, ..."

  Where "Sam" is not mentioned elsewhere in the premise or hypothesis.

- **Entity-Linked Appositive Insertion:** Inject complexity by using a Named Entity Recognition (NER) tagger combined with an external knowledge base (e.g., the Wikidata API).

- *Original:* "Purdue University announced a new contract with DoD."
- *Perturbed:* "Purdue University, a public research university in West Lafayette, announced a new contract with DoD."

Your Task

Develop an automated pipeline (using rule-based generation, parse tree manipulation, etc.) to apply at least three (3) distinct types of meaning-preserving perturbations to the MultiNLI dataset. Use your metrics from Part 1 to verify that the perturbed sentences exhibit increased complexity.

Part 4: Model Validation

You will now evaluate how the added complexity affects model performance.

- Use the same models you established as baselines in Part 2.
- Evaluate the models on your original MultiNLI subset and your new, perturbed subset.

Part 5: Findings and Report

Write a 3–4 page report detailing your methodology and findings. Your report must include:

1. A high-level introduction summarizing your approach and findings.
2. Definitions and justifications for your chosen complexity metrics. You must provide at least one concrete example for each metric you implemented.
3. A description of your perturbation methodology. This should include your approach, a comparison of complexity metrics before and after perturbation, and examples of original versus perturbed text.
4. A comparative analysis of model performance. Discuss whether certain architectures handled the added complexity better than others. You must also conduct an error analysis.

   Error analysis goes beyond just reporting a drop in accuracy; it requires you to isolate specific examples where a model correctly classified the original text but failed on the perturbed version. Examine the linguistic differences in these sentence pairs and hypothesize *why* the added syntactic complexity broke the model's understanding (e.g., did an added relative clause create a long-distance dependency that the model could not resolve?). Furthermore, leverage your tracking of the data splits: you should compare these failures across the `matched` (in-domain) and `mismatched` (out-of-domain) sets to discuss whether models are more vulnerable to syntactic perturbations when dealing with out-of-domain text.
5. Scatter plots showing the relationship between your complexity metrics (X-axis) and the model error rate or accuracy (Y-axis).

Evaluation and Submission

- **Code:** Submit your runnable code (must be named `hw3.py`) and the data (must be named `data.jsonl`) subset you used to Gradescope under `Homework 3 - Programming`. **It is your responsibility to ensure the reproducibility of your code; failure to do**

so will result in point deductions. For reference, see: PyTorch Reproducibility. For example, if your code relies on a random number generator, make sure to set the random seed to a fixed value at the beginning of your program, so that all subsequent executions of your code will produce the same output. The final output of your code should save all results as two CSV files with the following columns (the numbers must exactly match those shown in your report):

  - `perf.csv`: `model, perturbation_method, performance`
  - `complex.csv`: `perturbation_method, metric_type, value`

- **Report:** Submit a typed PDF report to Gradescope under `Homework 3 - Conceptual`.
- **Runtime:** Ensure your code finishes **within 24 hours**. Submissions exceeding this limit will receive zero points.