Guide for Instructors Creating 57x Homeworks

David Inman

June 27, 2018

Contents

This document is meant to be a guide for instructors who are crafting or modifying homeworks for LING 570, 571, and 572. The goal of this document is to help these homeworks to be clear to students, easily gradable by TAs, and to automatable. The content of the assignment itself is up to the instructor. This document will focus on writing assignment descriptions (§1), output formatting (§), and the creation of instructor-directed support files (§).

1 Homework Descriptions

This is a set of best practices for making homeworks easy for students to understand and to standardize grading practices across the CLMS series.

1. Homeworks descriptions should include an at-a-glance summary of turn-in files at the end of the document.

2. Homeworks should include an in-line point distribution, with points assigned at section headers or when deliverables are listed.

   The points specific to the assignment itself should add up to 75 (§2.2). Either at the end or beginning of the homework description, the standard portion of the grade (25 points) should be referenced (§2.1). Rather than reprinting the entire standard portion of the grade. For the standard portion, students can simply be pointed to the definition at `http://depts.washington.edu/uwcl/clms/course-policy.pdf`, or the point breakdown can be repeated, if desired.

3. Algorithms and outputs should be described separately.

   It is sometimes difficult to fully clarify an assignment. When writing an assignment description, the best practice for clarity is to separate different parts of the assignment by section. If possible, describe the expected function of the program (Viterbi, classification, etc) in one section, and then describe the format of the output in the following section or a separate sub-section. Try to minimize component interleaving in the description, although full separation may be impossible.

4. Points should be awarded for program *output*, not for code structure.

   This prevents the TA from needing to review the code for each student. For instance, "Program outputs a valid path (10 points)" is preferable to "Initializes lattice (10 points)."

5. Graded outputs should be strictly formatted.

   If a portion of a student's output is to be graded, that output must be uniform across all students. This means that the output must be very precisely formulated. For instance, "Epsilon arcs are followed without printing an output value (10 points)" is preferable to "Handles epsilon transitions (10 points)." Likewise, "Re-print the input string with the labeled parts of speech after each word, followed by a tab character: e.g., word1 \t POS1 \t word2 \t POS2 (...)" is better than "Print the input string with the parts of speech after each word."

   The more detailed the output description, the better.

6. Graded outputs should be strictly ordered.

   Because the grading script works by diffing outputs against gold formats, the order in which material is printed is important. If students are printing a dictionary (and only a dictionary), the dictionary can be pre-sorted alphabetically before diffing. However a complex output cannot be sorted this way. Within a line, there must be a specific order in which contents are produced. For instance, if the output is

```
<Class 1> probability <Class 2> probability <Class 3> probability
```

then the homework description must specify the order in which these items should be printed. E.g., by alphabetical order of class, or by descending probability. Any ambiguity in print orders will make assignments difficult to grade.

2 Rubric

The rubric across 570, 571, and 572 has been standardized. There are two parts: the standard component, which is graded identically across all assignments, and then the homework-specific portion, which is defined for each assignment. This section is presumed to be done by the TAs.

2.1 Standard Portion (25 points)

There is no deviation for this component, and it is graded automatically by the grader script. Points are assigned as follows:

1. Submitted Files (10 points)
    - 1.1 Tarfile submitted (2 points)
        - 1.1.1 Exists (1 point)
        - 1.1.2 Correctly named (1 point)
    - 1.2 Readme file submitted (2 points)
        - 1.2.1 Exists (1 point)
        - 1.2.2 Correctly named (1 point)
    - 1.3 Requested files exist in tarfile (6 points)
2. Program runs as expected (15 points)
    - 2.1 Code runs to completion on input (10 points)
    - 2.2 Output of running code matches turned-in files (5 points)

An additional 2 points may be taken off if output files are turned in with Windows carriage returns.

2.2 Homework-specific Portion (75 points)

This section cannot be schematized, but is based on the needs of each assignment. However, there are some best practices. Although the gross distribution of points is defined by the professor in the homework description, the TA still has to make decisions about the finer distribution.

1. Some points should be assigned for correct formatting for each output file.
2. Points distributed for the program being implemented correctly should depend on evaluating the program output on test files that are as modular as possible.

   For instance, the grader should create an input that will test for the code's ability to handle multiple labeling paths, and assign points for "Chooses the correct labeling path" based on that test. Then a separate test may look for the code handling unknown words. These tests should not be intermixed (or there should be a separate test input that mixes them).

3 submit-file-list

Most of the grading files are created by the TA, however the submit-file-list should be created by the instructor. This file lists all the expected files in the student's submitted tar. The format is a set of newline-separated list of expected files, in relative path format. For instance, the file might look like the following:

```
run.sh
table
Q2/q2.sh
Q2/decoder.sh
Q3/q3.sh
Q3/results/data.txt
```