Assignment 1

CMSC 473/673 — Introduction to Natural Language Processing

Item	Summary
Assigned	Wednesday September 6th
Due	Wednesday September 18th
Topic	Warmup, with Pytorch and UD
Points	85

In this assignment you will step through some introductory NLP techniques. It will also give you a chance to review some concepts that are relevant to how we train and formulate NLP methods that you may have seen an arbitrary number of semesters ago.

You are to *complete* this assignment on your own: that is, the code and writeup you submit must be entirely your own. However, you may discuss the assignment at a high level with other students or on the discussion board. Note at the top of your assignment who you discussed this with or what resources you used (beyond course staff, any course materials, or public Discord discussions).

The following table gives the overall point breakdown for this assignment.

Question	1	2	3	4
Points	25	30	15	15

What To Turn In Turn in a writeup in PDF format that answer the questions; turn in all requested code, environment files (if applicable) and execution instructions necessary to replicate your results. Please do **not** turn in the raw UD data (used in Q2).

Be sure to include specific instructions on how to build (compile) your code. Answers to the following questions should be long-form. Provide any necessary analyses and discussion of your results.

How To Submit Submit the assignment on the submission site:

https://www.csee.umbc.edu/courses/undergraduate/473/f23/submit.

Be sure to select "`Assignment 1`."

Questions

1. (**25 points**) Read the syllabus, watch the video about academic integrity (`https://drive.google.com/file/d/1nx-k5kXwlxOTj2rrhBhU69uwehamlje7/view`), and then answer the following questions. You must be logged in via UMBC to watch the video.[1]

   (a) This course will require you to write written English in addition to code. So let's say you're writing a report and you want to use information from some previous source. Provide, with an example, of how:

   i. you would directly quote from that source.

   ii. you would paraphrase from that source.

   iii. you would *improperly* use information from that source in a way that would be academically dishonest. (For this particular sub-question 1((a))iii, you will obviously not get in trouble for academic integrity violations, provided your answer is your own.)

   (b) Explain, in a paragraph, how you can work together while avoiding issues of academic integrity.

   (c) When it comes to group work, is each individual responsible for the entire group's work?

   (d) Provide three potential instances of academic integrity relating to coding.

   (e) Discuss how you can utilize online resources and forums, such as Stack Overflow, effectively and with integrity in this class.

   (f) Discuss how online resources like Stack Exchange or Chegg can be misused and result in academic dishonesty.

2. (**30 points**) In this question, you'll be doing some basic counting of words—the "Hello, World" of NLP.

   You will need to load up the `en_ewt` subcorpus of the Universal Dependencies corpus. You may use the Huggingface `datasets` module to assist, or you may download and load the files yourself. Note that in its raw form UD is distributed as a collection of specialized "`conllu`" files—the Huggingface `datasets` library handles parsing those `conllu` files for you. You may also use the 9/6 colab notebook as a starting point. For this assignment, do not run your code on the test data. **If you use the test data, you will not receive full credit**!

   (a) Read through the documentation discussing the format of the dataset:

   `https://universaldependencies.org/docs/format.html`.

   Using this documentation, verify to yourself that the first sentence has 29 tokens (nothing to turn in).

   With that understanding, compute the number of sentences in both the training and development splits? (Hint: you'll want to use "`train`" and "`validation`" `splits`!) On average, how many words are there per sentence for each split?

[1] While this video was prepared for graduate student orientation, it is relevant for all 473 and 673 students.

You can use any method you want to compute the number and averages (e.g., Linux commands, or a small Python or Java program). Turn in what you wrote.

While it is a good idea to question your data, especially if it looks strange/not what you expected, for this question you can take these tokens as they are: assume that, for *some* application, they are useful. The text you see is called *tokenized text*. In particular, it is text that has been tokenized, or split into individual "words," according to a particular specification. You may be surprised that we consider punctuation as different tokens.

But let's dive into this some more. The individual instances you observe are **tokens**, where each token is drawn from a set of **types**. Using a programming analogy, we can say that word types are like classes while word tokens are like instances of that class. For example, in the following sentence there are six types and eight tokens:

```
the gray cat chased the tabby cat .
```

Notice that this computation includes punctuation.

(b) Working with the training set only: Using the "tokens" key (or if you're processing the conllu directly, the "FORM" field), how many different word types and tokens are there? Do not perform any processing that modifies the words. Turn in the code for this.

(c) Examine some of the most common words from the training set, like the twenty, thirty, and fifty most common ones. Using examples, discuss what you notice about those common words. Each word should be alphanumerically (lexically) distinct.

(d) In the above question, should all of these items actually be considered distinct? What are some ways that we could group together words? Hypothesize some effects your collapsing would have. You can argue for or against your collapsing method.

Hint: There is not a right or wrong answer here. You may want to examine the "lemmas" field when coming up with your answer. Some collapsing methods may be more appropriate than others, but the question is to think about these methods and what effect they may have.[2]

(e) Using the tokens/FORM field, examine some of the least common words in the training set. For this question, I recommend implementing a way of examining words that only appear m times in training. With this, you should be able to identify words that appear only once, or only appear 10 times, or 100 times.

(i) At what general point (value of m) do the words start looking like "standard" words? Note that there's no *precise* value of m that's correct or incorrect: this is about you looking at the data and thinking about possible linguistic / empirical trends.

[2]Now, there are *simpler* answers. In particular, some collapsing methods can be accomplished with simple calls to standard string processing functions. Others could be accomplished with some more advanced processing (e.g., see column 3).

(ii) Now, regardless of whether these words were *standard*, are they "reasonable?" That is, are they items that you would want to be able to talk about as distinct items? From a computational point of view, do you want to spend the computational resources to deal with them?
Argue for or against these items' reasonableness. If you find them unreasonable, propose a solution. You do not need to implement it.

(f) Now it's time to look at the development (aka validation) split.

(i) How many word tokens are there in this split?

(ii) How many word types in the validation split were not seen in the training data? We call these *out of vocabulary* (OOV) words.

(iii) This proportion of OOV words is pretty standard in NLP. Did the number of OOV words surprise you? Briefly discuss (roughly 2-3 sentences) the potential implications of having this many OOV words.

Again, do not perform any processing that modifies the words. Turn in the code for this.

3. (**15 points**) While there are a number of libraries that are popular to use for machine learning, especially when it comes to the neural/deep learning aspects of NLP, Pytorch is a very popular library. Especially if you are not familiar with Pytorch, go through the Pytorch tutorials (`https://pytorch.org/tutorials/`) listed under "Introduction to PyTorch" (on the left). The most important, *general* ones for now are the "Basics," "Tensors," "Build the Neural Network," and "Automatic Differentiation" tutorials. I recommend opening an interpreter, such as a Colab notebook, and running the code in the tutorial as you read it.

(A) A *layer* is simply a way of encapsulating some computable function. Describe, in words, what `torch.nn.Linear` computes.

(B) Pytorch is row major. However, this can get a bit annoying when having to deal with matrix-vector multiplication, since the vector would need to be a column vector represented in row major order. So, Pytorch will often try to be liberal and accommodating in how it interprets one-dimensional tensors. We can see this in the following code (assume all imports are okay):

```
f = torch.nn.Linear(3, 2, bias=False)
f.weight = torch.nn.Parameter(
     torch.Tensor([[3, 2, 1],
                   [1, 4, 1]]))
x = torch.Tensor([1, 0, 0])
y = f(x)
```

State what the value of `y` is, and describe conceptually what this computation is doing. (Hint: you could also try `f(torch.Tensor([0, 1, 0]))`, `f(torch.Tensor([0, 0, 1]))`, and / or `f(torch.Tensor([1, 1]))`. You can also try different values of `f.weight`, just make sure that it retains the same 2x3 shape.)

(C) Now, following after (B), consider

```
z = y.sum()
```

The recommended Pytorch tutorials covered that running `z.backward()` will compute "the gradient." But, what gradient is actually being computed, and how does it depend on `x` vs. how does it depend on the value of `f.weight`? To help answer this, you can examine the gradient that is actually computed (`f.weight.grad`).

4. (**15 points**) Hal Daumé III has a very nice "refresher" tutorial called "Math for Machine Learning:" `http://www.umiacs.umd.edu/~hal/courses/2013S_ML/math4ml.pdf`. Where indicated, some of the following questions are taken from that primer. I encourage everyone to read the primer, but especially if you have difficulty answering the following questions.

   (A) **[based on Exercise 1.2]** Compute the derivative of the function $f_\mu(x) = \exp(-\frac{1}{3}(x-\mu)^3)$ with respect to μ.

   (B) Based on your answer to (A), for $x = 1.5$, compute $\frac{df}{d\mu}|_{\mu=5}$, and then use Pytorch's backprop/autograd capabilities to verify it. Turn in your code.

   (C) **[Exercise 1.5]** With respect to a and b, compute the gradient of the function $f(x) = \log(ax^4 + bx^2 - 1)$. Assume natural log and assume a and b are positive numbers.

   (D) Based on your answer to (C), at $x = -2$, compute $\frac{df}{da}|_{a=1}$ and $\frac{df}{db}|_{b=1}$, and then use Pytorch's backprop/autograd capabilities to verify it. Turn in your code. You can assume $a = b = 1$.

   (E) Given two K-dimensional vectors u and v, compute the dot product $u^\intercal v$ (write out a formula for the dot product between arbitrary real-valued vectors u and v of size K).

   (F) Given the matrix $A = \begin{pmatrix} 4 & 2 & 0 \\ -1 & 0 & -1 \end{pmatrix}$, compute the values $A^\intercal A$ and $AA^\intercal$.

   (G) Using Pytorch, verify your answers to (F). Turn in your code.