University of Arizona	Homework#3 POS tagging
CSC 585 Algorithms for NLP	Due date: October 3, 2023
Fall 2023	**Name:**
Instructor: Eduardo Blanco	

Instructions

- Bring a hard copy to class with your answers to Questions 1 and the two mandatory questions from Question 2. If you choose to do the extra credit part, your report should also include a report of what you did and the results you get.
- Submit your implementation for Questions 2 on GitHub Classroom. You will find the invite link for the repository in d2l. Follow these steps:

```
# Accept the invite link for the assignment available at d2l.
# It will create a repo named 03-pos_tagger-GITHUB_USER
#   Please use a github user name that is your name (or something close to it).
# Then set up the environment
> git clone YOUR_REPO
> cd 03-pos_tagger-GITHUB_USER
> python3 -m venv .env
> source .env/bin/activate
> pip install -r requirements.txt
```

  Grading is automatic. You have access to all the public test cases. You can run the test cases with the following command: `> pytest`. You can also run individual test cases (e.g., `pytest test_pos_tagger_baselines.py`). While grading is automatic, we will run your implementation with other train and test files, so do not hard code anything.

  A good implementation runs the test cases in `test_pos_tagger_baselines.py` in one minute and a few seconds. Your implementation must run in less than two minutes to receive credit.

  To run the three part-of-speech taggers, run

```
> python implementation/pos\_tagger\_main.py data/train data/test.ref}
```

Question 1 [20pt]
Go to `https://explosion.ai/demos/displacy`, uncheck *Merge Phrases* and run spaCy with the following sentences (do not change anything, copy and paste the sentences below (one at a time) without changing anything. Do not change punctuation, upper and lower case, etc.):

1. `Move left to the wall on the right side of the sink where the towel holder is.`
2. `Move left.`
3. `Move right.`
4. `The LAS & LDWS does not operate until the system detects white (yellow) lane lines on either the left or right.`
5. `Place it on the middle shelf.`
6. `Place it on the middle grey shelf.`

7. `Sentences with tipos are harder.`
8. `state or describe exactly the nature, scope, or meaning of.`
9. `U.S. school district to allow students back in classrooms`
10. `Fourth-Largest U.S. School District to Allow Students Back in Classrooms`
11. `I want comprar un dog.`
12. `Encantado de conocerte.`
13. `Voici votre carte d'embarquement.`
14. `I want comprar un dog.`

First, copy the exact tokenization and part-of-speech tags you get from spaCy for the sentences above. Second, comment on the POS tagging errors you see. The following questions may help you, but I expect you to dig a bit deeper. You cannot prove anything, but you can certainly make hypotheses about what may be the causes of errors:

- Do you see any tokenization issues?
- Do you think that *left* is usually a verb, an adjective or an adverb in the training corpus?
- Do you think the training corpus includes the tokens *LAS* and *LDWS*?
- What tags does spaCy tend to predict for tokens that (most likely) it has not seen during training?
- Do you think that letter case affects the system?
- Does the system include a language detector?

Note: You may run spaCy from Python if you prefer. Spell out the version of spaCy and the specific model you use.

Question 2 [80pt]
Implement a part-of-speech tagger. You are given a tagger that always predicts `NN`. You have to implement the "most likely tag for this token type" baseline and an HMM tagger with Viterbi decoding. For the HMM, during training, you have to calculate two kinds of probabilities:

- $P(t_i|t_{i-1})$. This is similar to the bigram language model you have already implemented. Use Laplace smoothing. These are the prior probabilities.
- $P(w_i|t_i)$, the probability of observing word w_i given POS tag t_i. You don't need to smooth this probability distribution. These are the likelihood probabilities.

A few hints:

- Add `<s>` and `</s>` at the beginning and end of each sentence.
- Unknown tokens. You will find words w_i in the test set that you have never seen before, so you don't know $P(w_i|t_i)$. You can estimate $P(t_i|w_i)$ using several methods:
  - Uniform distribution (guess randomly).
  - Prior probabilities $P(t_i)$ from training. If you have to guess, you are more likely to guess right if you choose the most likely tag. Even better: choose the most likely tag for unseen tokens.
  - Use surface patterns and morphology to assign more probability to more likely tags for a specific unknown word. This is simple to implement and effective. For example, unseen tokens that are a bunch of digits should be tagged with `CD`.

  Once you have $P(t_i|w_i)$, you can get $P(w_i|t_i)$ using Bayes:

$$P(w_i|t_i) = \frac{P(t_i|w_i) \times P(w_i)}{P(t_i)}$$

One option to estimate $P(w_i)$ is to consider all unknown tokens the same (think of a dummy token type: `UNKNOWN_TOKEN`), and assign a very low probability to them. You could also use Good-Turing discounting.
I recommend that you start thinking about morphology only after you have Viterbi decoding working.

In addition to the implementation, write a brief report with answers to the following questions:

- What accuracy do you get for seen and unseen tokens? Discuss why the results are different. (The current implementation calculates accuracy for all tokens). What POS tag gets the highest and lowest accuracy? Your rules to deal with unknown tokens should focus on the tokens with the lowest accuracy.
- Why do you get better results training with `train` and testing with `test.ref` than training with `train.on` and testing with `test.on`?

You will be graded as follows (the numbers between parenthesis are the accuracies you should get training with `train` and testing with `test.ref`, but we will rerun your code with other files so do not hard code anything):

- Majority baseline: 10 points (92.231)
- HMM (most likely) filling the Viterbi matrix correctly but not reconstructing the solution (most likely sequence of tags) correctly: 20 points (>94.142)
- HMM working: 30 points. (>95.192)
- HMM with some morphology and surface rules: 10 points (>95.5)
- Something that brings more gains in accuracy: up to 10 points (extra credit). This could be having more serious rules, using trigrams instead of bigrams for prior probabilities, using a different smoothing technique, or anything else you come up with. (>97.000). If you choose to get extra credit, explain in your report what you did and the results you get.
- Written report: 10 points.