Assignment 3
NLP 201: Natural Language Processing I

University of California Santa Cruz

First submission due date: December 9, 2022, 11:59pm
Second submission due date: December 13, 2022, 11:59pm

This assignment is to be done in Python.

Your final writeup for this assignment, including the problem and the report of the experimental findings of the programming part, should be no more than four pages long. You should submit it as a PDF. We *strongly* recommend typesetting your scientific writing using LaTeX. Some free tools that might help: TexStudio (Windows), MacTex (Mac), TexMaker (cross-platform), and Overleaf (online).

Problem

In this assignment, you will build a **parts-of-speech (POS) tagger** for English using a Hidden Markov Model (HMM).

Parts of speech are generally represented by placing the tag after each word, delimited by a slash. The following is an example of a POS tagged sentence:

The/**DT** Santa/**NNP** Cruz/**NNP** Sentinel/**NNP** is/**VBZ** a/**DT** daily/**JJ** newspaper/**NN** published/**VBN** in/**IN** Santa/**NNP** Cruz/**NNP**,/**,** California/**NNP**,/**,** and/**CC** owned/**VBN** by/**IN** Media/**NNP** News/**NNP** Group/**NNP**./**.**

Dataset

For this assignment, we will use the WSJ part of the Penn Treebank dataset, split into

- `train`: sections 00-18 (51, 681 sentences), data for training your HMM.
- `dev`: sections 19-21 (7, 863 sentences), data for debugging and choosing the best hyperparameters.
- `test`: section 22-24 (9, 046 sentences), data for evaluation.

The Penn Treebank POS tagset has 36 tags (45 including punctuation). The complete list of tags and examples is given in Figure 8.1, Chapter 8, of the textbook by Jurafsky, and also reproduced here in Table 1.

The dataset is downloadable from Canvas. To help you get started, we have provided starter code in starter_code.py . You will need to install the packages `nltk` and `scikit-learn` to run the starter code.

The starter code contains methods to load the dataset and automatically create the train, dev and test splits. Each split contains a list of POS tagged sentences in the same format as the example cited above. We also provide an evaluate() method to help you report your results. As an example, we call evaluate() on the test sentences tagged by the default NLTK POS tagger. For more information about reading and tagging text using nltk, please refer to chapter 5 of the NLTK book.

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	and, but , or	PDT	predeterminer	all, both	VBP	VBP verb non -3sg present	eat
CD	cardinal number	one, two	POS	possessive ending	’s	VBZ	verb 3sg pres	eats
DT	determiner	a, the	PRP	personal pronoun	I, you , he	WDT	wh-determ.	which , that
EX	existential ‘there’	there	PRP$	possess. pronoun	your, one’s	WP	wh-pronoun	what , who
FW	foreign word	mea culpa	RB	adverb	quickly	WP$	wh-possess.	whose
IN	preposition/ subordin-conj	of, in, by	RBR	comparative adverb	faster	WRB	wh-adverb	how, where
JJ	adjective	yellow	RBS	superlatv. adverb	fastest	$	dollar sign	$
JJR	comparative adj	bigger	RP	particle	up, off	#	pound sign	#
JJS	superlative adj	wildest	SYM	symbol	+,%, &	“	left quote	‘ or “
LS	list item marker	1, 2, One	TO	“to”	to	”	right quote	’ or ”
MD	modal	can, should	UH	interjection	ah, oops	(	left paren	[, (, {, ⟨
NN	sing or mass noun	llama	VB	verb base form	eat	)	right paren	],), }, ⟩
NNS	noun, plural	llamas	VBD	verb past tense	ate	,	comma	,
NNP	proper noun, sing.	IBM	VBG	verb gerund	eating	.	sent-end punc	. ! ?
NNPS	proper noun, , plu.	Carolinas	VBN	verb past part.	eaten	:	sent-mid punc	: ; ... – -

Table 1: Penn Treebank POS Tagset

1 Deriving the Viterbi Algorithm (25%)

The Viterbi algorithm for an HMM finds the sequence of tags which maximizes the probability $P(\mathbf{t}|\mathbf{w})$, or equivalently the log probability of $P(\mathbf{t}, \mathbf{w})$:

$$\begin{aligned}
\hat{\mathbf{t}} &= \operatorname*{argmax}_{\mathbf{t}} \log(P(\mathbf{t}, \mathbf{w})) \\
&= \operatorname*{argmax}_{t_1 \dots t_{n+1}} \log\left(\prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})\right) \\
&= \operatorname*{argmax}_{t_1 \dots t_{n+1}} \sum_{i=1}^{n} \log(P(w_i|t_i)) + \log(P(t_i|t_{i-1})) \\
&= \operatorname*{argmax}_{t_1 \dots t_{n+1}} \sum_{i=1}^{n} score(\boldsymbol{w}, i, t_i, t_{i-1}) \qquad (1)
\end{aligned}$$

where $score(\boldsymbol{w}, i, t_i, t_{i-1}) = \log(P(w_i|t_i)) + \log(P(t_i|t_{i-1}))$.

Here $\boldsymbol{w} = w_0 \dots w_{n+1}$ are the input words and $\boldsymbol{t} = t_0 \dots t_{n+1}$ is a tag sequence. The number of words in $\boldsymbol{w}$ is n, and we are including the start and stop symbols in $\boldsymbol{w}$ and $\boldsymbol{t}$, so $w_0 = \langle \text{START} \rangle$ and $w_{n+1} = \langle \text{STOP} \rangle$, and similarly for $\boldsymbol{t}$.

1.1 Questions

1. Why does $\text{argmax}_{\mathbf{t}}\, P(\mathbf{t}|\mathbf{w}) = \text{argmax}_{\mathbf{t}} \log(P(\mathbf{t}, \mathbf{w}))$?
2. For every value of t_j, let $\pi_j(t_j)$ be the log probability of the highest scoring tag sequence of length j ending in tag t_j:
$$\pi_j(t_j) = \max_{t_1,\dots,t_{j-1}} \sum_{i=1}^{j} score(\boldsymbol{w}, i, t_i, t_{i-1})$$
Show that $\pi_j(t_j)$ can be computed using π_{j-1}.
3. Give an algorithm for computing $\hat{\boldsymbol{t}}$ defined in the last line of Eq. (1). What is the time complexity of the algorithm? Be sure to carefully argue your answer.
4. In class, we stated that the Forward algorithm also works if you replace addition and multiplication with generalized addition and multiplication from any semiring $S = \langle A, \otimes, \oplus, 0_s, 1_s \rangle$, and becomes the Viterbi algorithm if you use the Viterbi semiring. Because the semiring Forward and Viterbi algorithms are equivalent, it is typical to refer to them both as simply the Viterbi algorithm. Thus, the semiring version of the Viterbi algorithm finds:
$$\hat{\mathbf{t}} = \bigoplus_{t_0 \dots t_{n+1}} \bigotimes_{i=1}^{n} score(\boldsymbol{w}, i, t_i, t_{i-1}) \tag{2}$$
where $score(\boldsymbol{w}, i, t_i, t_{i-1})$ is an element of the semiring. (The notation $\bigoplus$ is summation $\sum$ using $\oplus$ addition, and $\bigotimes$ is $\prod$ using $\otimes$ product.) For every value of t_j, let $\pi_j(t_j)$ be:
$$\pi_j(t_j) = \bigoplus_{t_1,\dots,t_{j-1}} \bigotimes_{i=1}^{j} score(\boldsymbol{w}, i, t_i, t_{i-1})$$
Show that $\pi_j(t_j)$ can again be computed using π_{j-1}. Which semiring properties did you use in your proof? Describe the changes needed to your algorithm in question 2 to solve Eq. (2).

2 Programming: Hidden Markov Model (25%)

We are working in a supervised setting, where you are asked to implement a fully observed HMM by estimating parameters from a training set using maximum a posteriori (MAP) estimation. Implement a bigram HMM with add-α smoothing. Remember to include $\langle \text{START} \rangle$ and $\langle \text{STOP} \rangle$ tokens at the beginning and ending of each sentence.

You will need to do the following:

- Read the training data, and construct the transition and emission tables.

3 Programming: Viterbi Decoding (25%)

To evaluate your HMM tagger, implement Viterbi decoding to find the best tag sequence on test data and report the accuracy of the tagger. Remember to perform all the computations in the log space to avoid underflows.

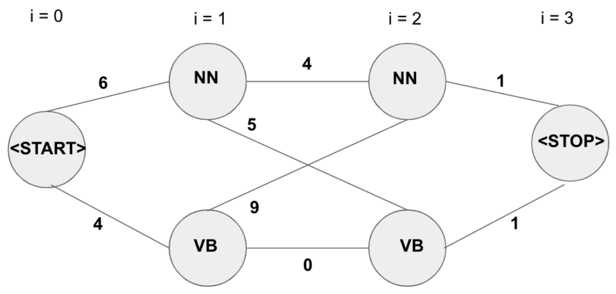

Figure 1: Decoding

You will need to do the following:

- Implement the Viterbi algorithm to find the best POS tag sequence. (Refer to Chapter 8, Fig 8.5 of the textbook by Jurafsky.)
- Implement add-α smoothing, with α= 1.
- Implement a **baseline tagger**, which assigns the most frequent tag for each word, as observed in the training set.

Debugging To help debug your code, try training it on a single example. If you train and test on single example, it should be able to learn to label it correctly. You can print out the score of the gold label sequence, and the score of the sequence your decoder found, to check that your decoder is finding the highest scoring sequence.

Fig 1 shows an example of best path computation. The best tag sequence is $\langle\text{START}\rangle$ VB NN $\langle\text{STOP}\rangle$ with a score of 4+9+1 = 14.

4 Evaluation (25%)

The accuracy of the POS tagger is calculated as the percentage of correctly labeled tags. To evaluate your POS tagger, do the following:

- Report the accuracy of your POS tagger on the dev and test set.
- Fine-tune the value of the hyperparameter α on the dev set and use it to evaluate the test set. Does this give better results than setting $\alpha = 1$?
- Calculate the **precision (P)**, **recall (R)** and **F1** on the test set for each tag, and report the macro-avg P, R and F1. Compute the **confusion matrix** for the tagset. Which tags are most often confused with which tags? What does this say about ambiguity in POS tagging? Briefly discuss your observations, and cite examples from the dataset.
- How does the HMM POS tagger compare to the baseline model?
- What can you do to improve the performance of your tagger?
- What improvements can be made to make the tagger run faster?

Submission Instructions

Submit a single zip file (`A3.zip`) on Canvas. The zip file should contain the following:

- **Code**: Submit your code together with a neatly written README file to instruct how to run your code with different settings. We assume that you always follow good practice of coding (commenting, structuring), and these factors are not central to your grade. However, please provide well commented code if you want partial credit. If you have multiple files, please provide a short description in the preamble of each file.
- **Report**: In the writeup, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.As noted above, your writeup should be four pages long, or less, in PDF (one-inch margins, reasonable font sizes).
  Part of the training we aim to give you in this class includes practice with technical writing. Organize your report as neatly as possible, and articulate your thoughts as clearly as possible. We prefer quality over quantity. Do not flood the report with tangential information such as low-level documentation of your code that belongs in code comments or the README. Similarly, when discussing the experimental results, do not copy and paste the entire system output directly to the report. Instead, create tables and figures to organize the experimental results.