CSE P 517: Homework 2
Hidden Markov Models (HMMs)

Due: Sunday, January 31 2021 at 11:59pm

In this assignment you will design and implement Hidden Markov Models (HMMs) for part-of-speech (POS) tagging. Given a natural language sentence $\mathbf{s} = x_1 x_2 ... x_n$, the task is to find the most likely sequence of POS tags $\hat{\mathbf{y}} = y_1 y_2 ... y_n$, that is,

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}}\, p(\mathbf{y}|\mathbf{s}) = \underset{\mathbf{y}}{\operatorname{argmax}}\, p(\mathbf{y}, \mathbf{s})$$

You are provided with Twitter data to tag. Apart from the added fun factor, it will help you think about the unique challenges of the modern Internet language. You will work with the 25-tag set proposed by the recent work of Gimpel et al. (2011) [3] that is designed specifically for #nooiiissyy??!! social media. This tagset is much more compact compared to the most commonly used 45-tag Penn TreeBank tagset [4], which, in turn, is a reduced set from the original 87-tag set used for the Brown corpus [2]. (Any thoughts as to why we are reducing the tagset over time?) While compact, this new tagset includes special tags to handle all your favorite Twitter vocabularies such as #Hashtags, @UserNames, emoticons, misspellings, URLs, and so forth. See Table 1 in [3] for the full list. Here's an example of a POS tagged tweet [3]:

tag	V	D	N	P	N	,
word	Spending	the	day	withhh	mommma	!

Submission instructions Submit 2 files on Canvas:

1 **Code:** (`HW2.tgz`): You will submit your code together with a neatly written README file to instruct how to run your code with different settings. We prefer Python for compatibility and concision. Discuss with the teaching crew in advance if you must use some other programming language. We assume that you always follow good practice of coding (commenting, structuring) and we will not grade your code based on such qualities of code writing practice.

2 **Report:** (`HW2.pdf`): You will also submit a typed report in the pdf format. The report will typically be about 2-3 pages in total. No restriction on font sizes, page margins, or number of pages. Consider technical writing as part of your educational training and organize your report neatly and articulate your thoughts clearly. We prefer quality over quantity. Thus, do not flood the report with tangential information such as low-level documentation of your code that should rather belong to the comments of your code or perhaps to the README file that you submit with the code. Similarly, when discussing the experimental results, do not copy and paste the entire system output directly to the report. Instead, consider creating tables and figures to organize the experimental results. When writing about *"error analysis"*, an important practice in any NLP project, try to make concrete arguments through examples.

Dataset

For this task, we pulled a random set of tweets from around April 1st. We filtered it down to tweets that were considered English by the Twitter API [1], and POS tagged them using the state-of-the-art tagger [3]. We provide you with the following files:

- *twt.train.txt*: data for training your HMMs (50,000 tagged tweets).
- *twt.dev.txt*: development data for you to choose the best smoothing parameters (5,000 tagged tweets)
- *twt.test.txt*: test data for evaluating your HMM models (5,000 tagged tweets)
- *twt.bonus.txt*: bonus data. If you feel like training a model on more training data, you're free to use this file or a subset of it (1,671,785 tagged tweets).

Each file contains one tagged tweet per line, where each line is a JSON object containing (word, tag) pairs. For instance, the tweet above would be encoded as follows:
`[["Spending", "V"], ["the", "D"], ["day", "N"], ["withhh", "P"], ["mommma", "N"], ["!", ","]]`

Measuring performance

To measure the performance, you will compute the per-word accuracy across all words (except for the special START or STOP symbols) in the test (or dev) set:

$$acc_{word} = \frac{\text{\# correctly tagged words}}{\text{\# words total}}$$

1 Bigram HMMs (6 pts)

You will start with the *bigram* Hidden Markov Models (HMMs), where the transition probabilities are based on bigram probabilities $q(y_i|y_{i-1})$. This model is identical to the version we have seen in the lecture slides:

$$\begin{aligned} p(\mathbf{s}, \mathbf{y}) &= p(x_1 x_2 ... x_n, y_1 y_2 ... y_{n+1}) \\ &= q(y_{n+1} = \text{STOP}|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i) \cdot \end{aligned}$$

As always, assume $y_0 = \text{START}$ is given, and $y_{n+1} = \text{STOP}$.

Training: You will write code to learn the model parameters using Maximum Likelihood Estimates (MLE). We will leave it to you to decide how to handle OOV and which smoothing techniques to apply.

Viterbi Decoding: You will then implement the Viterbi algorithm seen in class to tag new sentences in the test (dev) set.

1.1 Deliverables

- Describe your design decisions on smoothing and OOV handling. Compare the model performance of different design choices on the train and dev sets. Pick your best model and report its performance on the test set.
- Error analysis: provide error analysis on the dev set by discussing prominent error cases you may have noticed. Discuss with concrete examples. You can also compute the confusion matrix.

2 Trigram HMMs (9 pts)

You can extend the bigram HMMs by replacing the transition probabilities $q(y_i|y_{i-1})$ with $q(y_i|y_{i-1}, y_{i-2})$. This version of HMMs is called *trigram* HMMs.

Joint Probability Model: For this part of the assignment, you will first derive the joint probability model yourself. That is, you will need to first complete the following equation:

$$\begin{aligned} p(\mathbf{s}, \mathbf{y}) &= p(x_1 x_2 ... x_n, y_1 y_2 ... y_{n+1}) \\ &= q(...) \prod_{i=1}^{n} q(...)\ e(...) \qquad \leftarrow \textbf{Hint:} \text{ the equation will be in this form.} \end{aligned}$$

Viterbi Decoding: Next, you will derive the viterbi decoding algorithm. **Hint:** instead of $\pi(i, y_i)$, you will need to define $\pi(i, y_i, y_{i-1})$.

2.1 Deliverables

- Provide the formal equation for the joint probability model.
- Provide the viterbi decoding algorithm by providing the recursion with $\pi(i, y_i, y_{i-1})$. Make sure to handle the base case.
- Describe your design decisions on smoothing and OOV handling. Compare the model performance of different design choices on the train and dev sets. Pick your best model and report its performance on the test set.
- Compare the results with the bigram HMMs. Is the performance, better, worse, or about the same? Provide some commentary (a hypothesis and some evidence; this need not be conclusive) as to why the results are what you found.

3 Bonus: Unsupervised Training with EM (up to 2 pts)

Just for this assignment, we offer a bonus problem that you can attempt for partial credit. This bonus problem involves unsupervised training of HMMs through the forward-backward algorithm and the estimation of the HMM parameters via expected counts. Since implementing the full EM is more involving (see J&M 6.6 for the full recipe), and evaluation of unsupervised HMMs (mapping the learned tags to original tags) adds additional hassle, you may instead work with a semi-supervised version illustrated below:

Semi-supervised training with EM:

1. Partition the training set $\mathcal{D}$ into $\mathcal{D}_s$ and $\mathcal{D}_u$ of the equal size.
2. Do supervised training of HMMs using $\mathcal{D}_s$, then test these HMMs on $\mathcal{D}_u$ and record its performance on $\mathcal{D}_u$.
3. Next use the same HMMs from step #2 above to measure the expected counts on $\mathcal{D}_u$ using the forward-backward algorithm. Update the HMM parameters incorporating both the actual counts from $\mathcal{D}_s$ and the expected counts from $\mathcal{D}_u$. (This way you get to avoid the need for tag mapping.)
4. Then test (again) the newly updated HMMs to tag $\mathcal{D}_u$ using Viterbi decoding. Did the performance improve? why / why not?

Scoring scheme (to sum to a total no larger than 2 pts):

1. (pencil & paper) correct EM derivation on trigram HMMs: up to 1 pts
2. semi-supervised EM implementation on bigram HMMs: up to 1.5 pts
3. semi-supervised EM implementation on trigram HMMs: up to 2 pts
4. full EM implementation on bigram HMMs: up to 1.5 pts
5. full EM implementation on trigram HMMs: up to 2 pts

References

[1] The Streaming APIs. `https://dev.twitter.com/streaming/overview`.

[2] W Nelson Francis and Henry Kucera. Brown corpus manual. *Brown University*, 1979.

[3] Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 42–47. Association for Computational Linguistics, 2011.

[4] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993.