ממן 12 | מבוא לעיבוד שפה טבעית 22933

Sequence Labeling

Warmup

- In the dynamic programming matrix of the Viterbi algorithm (often referred to as a “trellis” which is a fancy name for this type of flow diagram), how many factors are multiplied for computing each cell?
- You are welcome to refresh yourself and follow the HMM algorithm in the book chapter, or elsewhere.
- How would the first-letter capitalization rules of the English / European writing system affect your tagger?

You do not need to submit the answers to the warm-up questions above! You are just advised to answer them for yourself before going very far into the next question.

Implementation Question: HMM

You are given again a function for fetching and reading conllu files from the English corpus already known to you, which include part of speech tags. (You do not need to read the files yourself, they are transformed for you by the accompanying code project).

1. Implement a bigram HMM* in python from scratch. Use it to perform Part Of Speech tagging over the supplied corpus. Before integrating any smoothing in your implementation (*“unknown words”* will give you a lot of pain at that point), observe the words coming up as unknown words with your test set during inference, and consider which ways you would use to overcome the challenge they pose.

2. Bonus: modify your implementation to return the top-*n* most probable tagging sequences rather than only the single best path from the trellis. Compute a top-*n* sentence level accuracy measure of your now modified model, over the same data, by modifying the function `_evalute` that is provided to you in `model_driver_12.py`, such that it computes an value (0 | 1) for each sentence, indicating whether the entire tag sequence is correctly predicted by *any* of the *n* outputs of your inference **

3. Bonus: derive a trigram HMM implementation based on your initial implementation, demonstrating a performance improvement over the bigram HMM model from question one.

---

* an HMM model assuming that the *transition* probability of a tag, is dependent only on the previous tag
** we will elaborate on the anticipated accuracy metric for this bonus question, on the course forum

Additional Bonus Questions

A. The **MEMM** is a discriminative model for sequence labeling, also decodable via the Viterbi algorithm which you have already implemented in the previous section. Implement an MEMM part-of-speech tagger while leveraging the ready-made logistic regression classifier implementation of scikit-learn (in other words, *do not* write or use your own logistic regression implementation as part of your MEMM implementation — use the sklearn implementation in your implementation).

   A practical tip: make sure to use the following arguments for sklearn's logistic regression class:

   ```
   LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
   ```

   these argument values make sure you are using multinomial logistic regression, rather than a one-vs-rest logistic regression.

B. What is the worst-case runtime complexity of inference, in terms of the number of labels *L* and the length of the input sequence *S*?

   - In a standard Bigram HMM?
   - In a standard Trigram HMM?
   - How much is the worst case complexity a relevant complexity metric, given the density of the transition matrix that you got in question one? Support your claim with a density measure of your transition matrix (you may use `np.count_nonzero` for that)

Aftermath

You do not need to submit these questions, but thinking about them helps foster another level of understanding:

A. We could use either of your previous implementations — the HMM, or the HEMM, which you have fully implemented by now — to execute *a **different** sequence labeling task*: **Named Entity Recognition**. Recall that Named Entity Recognition is typically solved by a reduction to a sequence labeling problem; as part of the reduction, we typically use the BIO, BIOSE, or other similar labeling schemes, to enable the reduction to work; this reduction would only entail minor *"adaptations" and "additions"* in your code. Suggest, why could a CRF model work better for this task?

B. One could also frame **Sentence Segmentation** (i.e. detecting the boundaries of sentences in running text) as a sequence labeling task (how?). Would a simple classifier (with the right features) potentially work as good as a CRF, for a sentence segmentation task?

Scoring Table for this Assignment

Question	Score Points	Score Criteria
1	95	Token Accuracy, and mostly correct impl.
2	12 bonus points	Top-*n* Sentence Accuracy
3	14 bonus points	Token Accuracy
A	12 bonus points	Token Accuracy
B	7 bonus points	1,2: Correct Big-O, 3: reasonable explanation

Token Accuracy is the the count of the correctly predicted tags divided by the number of tags across the test set, in our case averaged over cross-validation passes.

For the sake of this assignment we define **Top-*n* Sentence Accuracy**, as the count of all sentences for which any of the *n* predictions returned is *fully* correct, divided by the number of sentences in the test set. The motivation and rationale for Top-*n* metrics like this one, is to see how much can a model predict correctly, given the extra degree of freedom of providing its top *n* best predictions, rather than just a single one.