Writeup

Ruben Wolhandler
Liat Schiff

1. **Describe how you handled unknown words in the HMM.**

   We have used varied word signatures. We go over all the words that appear only one time in the training set and we check if the word belongs to one of them. If this word doesn't belong to any signature, we consider that this word belongs to a special class, the ^UNK class. We chose the different word signatures according to the English linguistic characteristics. In English, suffixes of words can help a lot to determine POS. So we use a range of the most frequent and informative suffixes such as -ed, -ing, -ance, -tion, -able, etc.
   In addition, we use some word signatures for tokens that contain numbers with or without punctuation (e.g., slash, colon, comma, etc.) for understanding implicitly if the word is a date, an hour, or a simple number, etc.
   We also checked if the word starts with an upper-case or if all the letters are upper-case.
   We have used overall fifty word signatures, you can see all of them in the python file MLETrain.py
   We count the word signatures according to the rare words (words that appeared only once in the training set). In inference time, if a word doesn't appear in the known words in the training set, we replace it with a word signature and use the emission probability of the word signature instead of the emission probability of the original word.

2. **Describe your pruning strategy in the Viterbi HMM.**

   Instead of considering all the possible tags for each word (or signature word), we only consider the tags that appear with this word in the training set. To clarify, if a word (or signature word) was never labeled with a particular tag in the training set, this tag will not be measured in the prediction process. This pruning strategy considerably reduces the run time.

3. **What is the runtime complexity of the trigram Tagger?**

   For every token of the sentence and all tag pair (of the two previous tokens) we do maximization over all tags, so the complexity is $O(N\ |T|^3)$
   N: is the number of tokens in the sentence
   T: is the set of all the tags

4. **On each dataset, report your test scores when running each tagger (hmm-greedy, hmm-viterbi, feature-based-classifier)**
5. **For the NER dataset, report token accuracy as well as span precision, recall and F1.**
   All the results are on the dev-set of POS and NER datasets

	POS	**NER**
HMM Greedy	**Accuracy: 0.941**	**Accuracy: 0.956** **Span precision: 0.745** **Span recall: 0.798** **Span F1: 0.7707**
HMM Viterbi	**Accuracy: 0.960**	**Accuracy: 0.964** **Span precision: 0.822** **Span recall: 0.818** **Span F1: 0.820**
Features based	**Accuracy: 0.965**	**Accuracy: 0.974** **Span precision: 0.881** **Span recall: 0.864** **Span F1: 0.873**

6. **Is there a difference in behavior between the greedy and viterbi taggers? Discuss.**

   Yes, the main difference is that Viterbi is an exhaustive algorithm. For each pair of tags, we calculate the highest likely sequence of tags probability ending with this pair's tags ending at position k of the sentence to help with dynamic programming. This algorithm will give us the most likely sequence of tags but the space complexity is O(N*T) and the runtime complexity is $O(n\ |T|^3)$.

   That's why we can also use the Greedy algorithm with a complexity smaller: O(n*T)

   Greedy algorithm choose the first tag: tag1 that maximizes the product of the emission probability and the transition probability: e(x1 | tag)q(tag | START, START) when x1 is the first token of the sentence.

And choose the second tag :tag2 that maximize e(x1 | tag)q(tag | START, tag1)
the k-th tag : tagk that maximize e(x1 | tag)q(tag | tagk-2, tagk-1)

The greedy algorithm can miss some tags sequence with higher probability. For example, if the greedy algorithm predicts the sub-sequence t1,t2,t3 and given this tag the next one will be t4, so greedy predict that t1,t2,t3,t4 is the “best sequence tag” but t1,t5,t6,t7 was better tag sequence with the highest probability (the sequence t1,t5 was less probable than t1,t2 so greedy choose t1,t2). So greedy can’t guarantee to predict the optimal tag sequence but only an approximation.

7. Is there a difference in behavior between the datasets? Discuss.

There are two main differences between the datasets.
First, the Ner dataset isn’t balanced, the tag ‘O’ appears above 80% of the tokens. That means that even a majority baseline algorithm that always chooses the major class
will get decent results.

By contrast, the POS dataset doesn't suffer from this problem at that level. There are tags that are more prevalent (e.g. 'IN' appears 107862 times and 'NN' appears 132935 times) than others (e.g. '#' appears 142 times), But still the most prevalent tag is about 13% of the data and there is more than one dominant tag.

Our reported results included the ‘O’ tag in the calculation and for this reason, they are higher than the POS task results. If we wouldn't refer to this tag in the evaluation the accuracy was much lower (e.g. the accuracy of the Features-based tagger was 84% instead of 97% with the ‘O’ tag). This gap may indicate the influence of an unblanched dataset.

Second, the ambiguity in the POS task is higher than in the NER task, which means that a word may belong to more labels depending on the context. The ambiguity makes it more difficult for algorithms (and even people) to learn especially when the context is limited (trigram).

8. Why are span scores lower than accuracy scores?

First, above 80% of the words in the NER task are labeled with the ‘O’ tag and thus we get bias on accuracy and we can get very high accuracy easily. Secondly, the span scores are very strict, the span metric considers a correct prediction if all the tokens of the spans are predicted correctly. So it is much harder to receive these scores.

9. **What would you change in the HMM tagger to improve accuracy on the named entities data?**

   We can use lexicons of well-known organizations, first and last names, etc, which will help our model by string match. We can also change the word signatures to be more adapted to this task.

10. **How did you improve the accuracy of the NER tagger? what did you try? what worked? what didn't work? What does your best NER model include?**

    Our best Ner model includes surprisingly only the features we have used for the POS task, such as suffixes[1:5], prefixes[1:5], two words before, two words after, and the tags of the two words before.

    We have tried adding recommended features such as containing capital letters (all-capitalized, is-capitalized), punctuation, numbers (all-digits, alphanumeric), and dates. We have tried different combinations of these features, but all of them have had a bad influence on our result (F_score range: 0.64 - 0.85).

    In addition, we have explored the pos features with some of the lexicons that were given. We examined a feature that check if the word is contained in the person lexicons (e.g. "lexicon/firstname.5k","lexicon/firstname.10", "lexicon/lastname.5000", "lexicon/people.family_name" ,etc.), but the F.score was dropped to 0.866.
    We have also tried a location feature that checks if the word is contained in the location lexicons ("lexicon/venues", "lexicon/location.country" and "lexicon/location"), but the F.score was dropped to 0.86. Adding these two features together decreased the F.score to 0.856.

    To this end, the model of part 2 has the best results for the NER task with a span F1 of 0.873.