Assignment 2

In the second assingment instead we built our own tagger by using Hidden Markov Models (HMM).

This was done in 2 different ways:

1) We built an HMM tagger in a totally supervised way by using the language modeling. We used trigrams and smaller models for the predictions
2) We built an HMM tagger this time by using the Baum-Welch algorithm.

Training:

We are going now to describe the approaches described for both methods.

First of all, we divided the data into 3 subsets: T, H, S.

T = Training data, H = Heldout data, S = Test data

1) **Supervised HMM**:

   Here, the whole supervised traning data T were used.

   A. The computation of the trigrams, bigrams, unigrams and uniform distribution was performed by means of the counts obtained directly from the training data.
   We carried this out efficiently by doing one pass through the data for computing the trigram counts and then, from those, we derived the bigram and unigram counts without going through the data all over again.

   Since we used the trigram model, each state of the HMM was having a pair of tags (tag_1, tag_2).

   B. For what concerns the output probabilities instead we associated the outputs to each state (not to each arc).

   C. The smoothing of both distributions is performed over of the heldout data H.
   After the smoothing we obtained the following distributions:

   Transition distribution:

$$p\big((t_1,t_2)\big|(t_0,t_1)\big) = \lambda_3 p\big((t_1,t_2)\big|(t_0,t_1)\big) + \lambda_2 p\big((t_1,t_2)\big) + \lambda_1 p\big((t_2)\big) + \lambda_0 \cdot \frac{1}{|unique\ tags|}$$

   Output distribution:

$$p\big(w\big|(t_1,t_2)\big) = \lambda_2 p\big(w\big|(t_1,t_2)\big) + \lambda_1 p(w) + \lambda_0 \cdot \frac{1}{|V|}$$

2) **Unsupervised HMM**:

   We divided the training data T into 3 parts: ST, UT, SH

   ST + SH correspond to a small fraction of T (we used 10000 words in total for both sets in the experiment), while UT is the remaining part of T

   A. We learned the initial parameters of the HMM tagger over ST in the same way described in point 1.
   This way, we were umbalancing the HMM tagger in some direction (better than the uniform distribution which would've lead to slower unsupervised training).

   B. We smoothed the model by using the EM algorithm over the set SH.

   This step is crucial: without smoothing the initial distribution the learning with Baum Welch is not possible.

Think at the following example: the word XX wasn't seen on the supervised training data but was seen in the unsupervised training data.
Without smoothing $p(XX|s) = 0 \; \forall s \in states$, hence the model wouldn't be able to learn anything with the word XX.

C. We use the Baum-Welch algorithm over the training data without tags in order to fine tune the parameters of the HMM model.
In the experiment we set a threshold for convergence $\epsilon = 0.4$. The algorithm converges when less than t updates result in a $\Delta p > \epsilon$. We chose $t = 50$ in order to get to a faster convergence.

The formulae for this step that I have mainly used can be found at https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm

D. We smooth the obtained train and output probabilities as usual with the EM algorithm in order to avoid 0 probabilities in the evaluation phase. In this step we used the heldout data H.

Evaluation

We evaluated the performance of our model by means of the Viterbi algorithm.

A few notes on the implementation:

- When considering very long sentences, it can happen that the multiplication of many alphas in a row lead to very small alphas in the later layers.
It could happen that the representation of the alphas is so small that they cannot be represented as floating point numbers anymore, python will just approximate them to 0.
Of course, this is a problem in the execution of the viterbi algorithm.
For this reason, we took the logarithm of the formula for the computation of the alphas.
Since the logarithm is a monotonic function, this doesn't change the outcome of the Viterbi pass.

$$\alpha(t) = \log\big(\alpha(t-1) * p(s|s') * p(w|s)\big) = \log\big(\alpha(t-1)\big) + \log\big(p(s|s')\big) + \log\big(p(w|s)\big)$$

- Another important thing to point out is the importance of pruning. Pruning is very important during the computation of Viterbi algorithm in order to limit the computation time needed for the execution of the algorithm.
The approach chosen was to keep only the 20 higher α's at every step of the computation.

Results:

We report here the results obtained in this assignment:

TEXTEN2.ptg

HMM Tagger:

Quantity	Value
Split 0	0.8479910
Split 1	0.8428858
Split 2	0.8571316
Split 3	0.8242201
Split 4	0.8242201
Mean	0.8392897
Variance	17.2232e-5

Baum–Welch tagger:

Quantity	Value
Split 0	0.49572
Split 1	0.47341
Mean	0.48456
Variance	12.4434e-5

TEXTCZ2.ptg

HMM Tagger:

Quantity	Value
Split 0	0.5521015
Split 1	0.6014666
Split 2	0.5902291
Split 3	0.5574901
Split 4	0.5766924
Mean	0.5755959
Variance	35.28865e-5

Baum–Welch tagger:

Quantity	Value
After 8 hours no convergence	
Mean	-------
Variance	-------

Observations:

The performance we obtained here can be improved by using a higher number of nodes when pruning for what concerns the HMM tagger.
Decreasing the threshold for convergence and increasing the number of input words for the unsupervised learning could improve accuracy for the Baum-Welch training.

This can be done when more performing computational power is available.

For what concerns the English text, 8 hours were enough for the convergence of 2 splits. We didn’t set the parameters in the best way possible because of hardware and time limitations (I could only run the script at night).
However, for the Czech script, 8 hours weren’t enough for the convergence of 1 split, maybe this is due to the higher complexity of the language and the tags.

In any case, we can see that the performance for the English Text is very close to the one of the Brill’s Tagger. However, the training time for the HMM tagger is much lower compared to the Brill’s one. The decision of which should be chosen must be done accordingly to the kind of task (therefore, whether we prefer faster training time or higher accuracy).

For the Czech text the difference is more relevant and in this case, the Brill’s tagger should be preferred to the HMM supervised one.

The Baum-Welch tagger has very high convergence time, training only few splits was possible.
The results obtained are not very accurate but in general the time needed is very high and therefore should be chosen only when we don’ t have labeled data.