Student Name: Parsa Mazaheri
Student ID:

Natural Language Processing I

Assignment 3

---

1 Deriving the Viterbi Algorithm

1.1

For getting $argmax(P(t|w)) = argmax(log(P(t,w)))$, first, by using Bayes Rule here, we know that: $P(t|w) = \frac{P(t,w)}{P(w)}$. So by knowing this we can say that $P(t|w)$ is related to $P(t,w)$. and we get to:

$$P(t|w)\ \alpha\ P(t,w)$$

After that, by knowing that $log(x)$ is a monotonic, non-decreasing function, we can get $log()$ of both sides and since we want to maximize the value of $P(t,w)$ get get to

$$argmax(log(P(t|w)))\ \alpha\ argmax(log(P(t,w)))$$

And finally, since we only care about maximazing the value and as mentioned, log is a non-decreasing monotonic function, we know that $log(P(x))\ \alpha\ P(x)$ is also monotonic and thus we can use $P(x)$ instead of $log(P(x))$. and finally we get:

$$argmax(P(t|w))\ =\ argmax(log(P(t,w)))$$

1.2

To prove that π_i can be derived from π_{i-1} we need to show that for a given $score(w,i,t_i,t_{i-1})$, we can split the function into π_{i-1} and $score(w,i-1,t_{i-1},t_{i-2})$. For showing this we take out the last score and rewrite the score function as below:

$$\sum_{i=1}^{j} score(w,i,t_i,t_{i-1}) = [\sum_{i=1}^{j} score(w,i-1,t_{i-1},t_{i-2})] + score(w,i,t_i,t_{i-1})$$

Since $\sum_{i=1}^{j} score(w,i-1,t_{i-1},t_{i-2})$ can be written also as π_{i-1} we can extract that from the given equation and rewrite the equation as below:

$$\sum_{i=1}^{j} score(w,i,t_i,t_{i-1}) = \pi_{i-1} + score(w,i,t_i,t_{i-1}) \xrightarrow{then}$$

$$\pi i = argmax(\sum_{i=1}^{j} score(w,i,t_i,t_{i-1}) \xrightarrow{then}$$

$$\pi i = argmax(\pi_{i-1} + \sum_{i=1}^{j} score(w,i,t_i,t_{i-1}))$$

After that we have have our answer and showing that π_i can be be computed using π_{i-1}.

1.3

Since the scoring function proposed by the problem is an back tracking algorithm and each π_i as shown can be get from π_{i-1}, then we can say that for a sequence of length N, we need to only iterate over items once and check the tags for them. For tags length of T, by looking at Viterbi algorithm pseudocode, we can see that iterates T^2 for tags and the final Order becomes

$$O(N.T^2), \quad N: \ seq\ length\ ,\ T: \ Tag\ array\ length$$

1.4

The whole idea for using semirings is that we can replace an operator by our need and get the both algorithms without any other changes. Since the semiring operators (sorry that I can't show them): *the sum operator of semiring* and *the multiply sign of semiring* have the same function as sum and multiply as even mentioned in the problem itself, we can follow the exact proof and just replace the signs to get the same results for the semirings.

2 Hidden Markov Model

For the rest of the problems, we want to create a HMM with add-α smoothing. For each problem we try to create a and build a single component of that till in the end, have the complete code.
In this problem we want to read the training data and build the transition and emission tables which we will later use for our algorithm.

For this assignment for each sentence we need to add `<start>` and `<stop>` tokens. After separating tokens and tags we would have 77 tags with 44545 tokens (not including `<start>` and `<stop>`).

2.1 Building the tables

For building the tables, we need to get the count of each tag and where they occur (observed) and store them. first we add alpha to every transition and emission from the states.

After that we count the number of each tag occurrence for the transitions and emissions table. The next step is to normalize the data by dividing by the sum of all counts for each state (tag). After that we would have a table for transitions and emissions, and an initial table for starting.

2.2 Class-Based Tagger

Since here we need to store a lot of information about the data, tokens, tags, transitions and emissions, we create a model to store these data with each other. This will later help us when we what to load test and dev data as well and help make everything organized. For training we get the word and tag count `dicts` and then create the transission and emission matrices. Hopefully the code is self explanatory but comments have also been provided to help for better understanding of the code.

3 Viterbi Algorithm

For developing the algorithm. we use dynamic programming (dp) to store all the data and to use them in the rest of the sequence if needed.

First, we initialize the the Viterbi and backtrack arrays before starting. After we iterate over the sequence for each sentence (list of tokens) and store the best prediction for each steps and also the the best previous state name.

4 Evaluation

In this problem, we're going to evaluate the mode we just developed and see how well it works on dev and test data.

$\rightarrow$

- For the test and dev set given $\alpha = 1$ we got the following results:

Set	Precision	Recall	Accuracy
validation set	0.93	0.94	0.935
test set	0.94	0.94	0.94

- I tried different values for alpha from 0.2 to 1 and from them 1 have better results than the others. Have trained the model on train data with $\alpha = 1$ and have obtained the F1 scores as in the previous section. The $\alpha = 1$ gives the best performance compared to others.

- Precision and recall was given in the previous table in section 1 of problem 4. The Confusion Matrix in the below is given for the first test case in the dev set.

-	precision	recall	f1-score	support
accuracy	-	-	0.94	171138
macro avg	0.73	0.64	0.66	171138
weighted avg	0.94	0.94	0.94	171138

```
sentence = [
    ('Influential', 'JJ'), ('members', 'NNS'), ('of', 'IN'), ('the', 'DT'),
    ('House', 'NNP'), ('Ways', 'NNP'), ('and', 'CC'), ('Means', 'NNP'),
    ('Committee', 'NNP'), ('introduced', 'VBD'), ('legislation', 'NN')
]

Confusion Matrix ...
[[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 5 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 4 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 9 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 6 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 2 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1]]
```

- **Difference between HMM POS tagger and the baseline model:**
  $\rightarrow$ The difference between the baseline mode and using a HMM POS tagger is that in the baseline mode, the algorithm only saves the `argmax` token at each step but in Viterbi algorithm, since we're

using dynamic programming, we save the data for the whole tags. (we can also note here if we use k-best instead of all the nodes, we can still get very good result and improve performance).
Based on statistics and the results given in the notebook as well, the POS tagger with the Viterbi algorithm is superior compared to the baseline model. implementation

- **Improving the performance of the model:**

  → **More data:** One thing that can be helpful is to use a larger training data and after using more training data, we can have better transition and emission metrics and more accurate data.
  → **Paralleling:** Another thing that can help with the performance, especially when training on large amount of data is to able to paralleling the computation for creating the matrices.
  → **Fine-Tune:** One other thing that can improve the performance of our model is to fine tune the model for our specific problem. In which we try to tune the model to do better for that given sub problem.

- **For making the model run faster we can:**

  → **Paralleling:** Again here as well one thing that can improve the speed of the tagger is to paralleling the computation for creating the matrices and for calculating the best path.
  → **Speed-Accuracy Trade-off:** One thing aging to the time when first deep learning was created is the trade-off between accuracy and speed. The more accurate we want our model be, the less speed it may generally have. It usually depends on our need and what is the best choice for us in how accurate should the model be and how fast should it predict. For example, if our need is satisfied with 70% of the accuracy, we can use that to increase the speed for instance.