01.112 Machine Learning

Design Project

This document includes a brief report with instructions on how to run our code. Our codes are in .ipynb format. You can run the .ipynb file locally on jupyter notebook. We documented our code in the notebooks along with texts to indicate what each block does, so you can follow through while running each block to get a detailed explanation.

Part 2 - Emission-only Predictions

First we loaded the training dataset into a pandas dataframe. We then performed smoothing on the data where words that appeared less than k(3) times were replaced with a #UNK# token.Next, we used the concept of MLE to calculate the emission values of each unique word to tag pair using pandas inbuilt functionalities.

Next, we loaded the dev.in file and used the tags that had the highest possibilities of generating a particular word to label each of the words in the dev.in dataset and saved the file in dev.p2.out.

The following is a table of results when we ran our prediction output against the gold standard output:

[Average Runtime for EN dataset: 45s]

	EN	**AL**	**SG**	**CN**
#Entity in gold data	13179	8408	4537	1478
#Entity in prediction	19406	19484	19557	9774
#Correct Entity	9152	2898	2595	762
Entity precision	0.4716	0.1487	0.1327	0.0780
Entity recall	0.6944	0.3447	0.5720	0.5156

Entity F	0.5617	0.2078	0.2154	0.1354
#Correct Sentiment	7644	2457	1147	281
Entity Type precision	0.3939	0.1261	|0.0586	0.0287
Entity Type recall	0.5800	0.2922	0.2528	0.1901
Entity Type F	0.4692	0.1762	0.0952	0.0499

Part 3 - Viterbi Implementation

Step 1 - Finding Transition values

We used pandas dataframes to import the training data and used pandas library functionalities to get the resulting transition parameters that we would use. Again, we applied MLE and got our desired values to be used alongside the previously found emission parameters, for our next step.

Step 2 - Viterbi Algorithm

First, we took both the emission and transition parameters we found earlier, and put their values into a 2d numpy matrix so that we could reference this matrix later on during our calculations. We specifically used this method because numpy allows us to run our codes a lot faster than referencing straight from pandas.

After running our Viterbi Algorithm, we stored the predicted values into dev.p3.out and compared it once again with the gold standard file, and got the following results:

[Average Runtime for EN dataset: 2min 9s]

	EN	**AL**	**SG**	**CN**

#Entity in gold data	13179	8408	4537	1478
#Entity in prediction	12738	8523	2674	675
#Correct Entity	10833	6734	1496	296
Entity precision	0.8504	0.7901	0.5595	0.4385
Entity recall	0.8220	0.8009	0.3297	0.2003
Entity F	0.8360	0.7955	0.4149	0.2750
#Correct Sentiment	10419	6079	957	206
Entity Type precision	0.8179	0.7132	0.3579	0.3052
Entity Type recall	0.7906	0.7230	0.2109	0.1394
Entity Type F	0.8040	0.7181	0.2654	0.1914

Part 4 - Using Viterbi to get the 7th best path

This time round, instead of finding the argmax for each of the nodes, we are storing the top k(7 in this case) scores from the previous layer nodes.

During the backtracking portion of the Viterbi algorithm, we look for the kth(7th in this case) best score from start to stop and from then on, we find the indexes of each of the previous layers to get the kth best path.

After outputing our results into dev.p4.out, we compared with the gold standard again to get the following results:

[Average Runtime for EN dataset: 2min 20s]

	EN	**AL**
#Entity in gold data	13179	8408

#Entity in prediction	13054	8987
#Correct Entity	10223	5974
Entity precision	0.7831	0.6647
Entity recall	0.7757	0.7105
Entity F	0.7794	0.6869
#Correct Sentiment	9712	5003
Entity Type precision	0.7440	0.5567
Entity Type recall	0.7369	0.5950
Entity Type F	0.7404	0.5752

Part 5 (Design) - 2nd Order HMM

For this part we decided to use 2nd order HMM as it is very similar to the 1st order HMM that we learned in class.

Initially we used transition parameters between just node u and v. In this algorithm however, we take into consideration the transition parameter that comes from node u, v and w that each come from different layers in our Viterbi network. The same is done for our emission values.

For 2nd order HMM we calculated 2nd order Transition parameters i.e. [count(u, v, w)/count(u,v)] and 2nd order Emission parameters i.e. [count(u, v $\rightarrow$ o)/count(u, v)].

For first stage of our 2nd order viterbi algorithm i.e. START to first layer, we still use 1st order parameters but from second layer onwards we use 2nd order parameters.

Backtracking for 2nd order HMM is also similar to 1st order HMM, just that in second order HMM we retrace by using both 2nd order transition and emission parameters, instead of just transition parameters.

Unfortunately, using this method resulted in much worse scores. Despite several attempts at improving the score, via attaching weights to the prediction as per the method described on page 4 of the paper Second-order Belief Hidden Markov Models referenced below, we didn't manage to improve the scores.

$$P(s_k^t|s_i^{t-2},s_j^{t-1}) = \lambda_1\hat{P}(s_k^t|s_i^{t-2},s_j^{t-1}) + \lambda_2\hat{P}(s_k^t|s_j^{t-1}) + \lambda_3\hat{P}(s_k^t)$$

$$\begin{aligned}\lambda_1 &= k_3\\ \lambda_2 &= (1-k_3)\cdot k_2\\ \lambda_3 &= (1-k_3)\cdot(1-k_2)\end{aligned}$$

$$\text{where } k_2 = \frac{log(C(s_j^{t-1},s_k^t)+1)+1}{log(C(s_j^{t-1},s_k^t)+1)+2},\ k_3 = \frac{log(C(s_i^{t-2},s_j^{t-1},s_k^t)+1)+1}{log(C(s_i^{t-2},s_j^{t-1},s_k^t)+1)+2},$$

and $C(s_i{}^{t-2}, s_j{}^{t-1}, s_k{}^t)$ refers to the frequency of a sequence $s_i{}^{t-2}, s_j{}^{t-1}, s_k{}^t$ in the training data.

Once again, we output these values into dev.p5.out and compared with the gold standard to get the following results"

[Average Runtime for EN dataset: 1min 50s]

	EN	**AL**
#Entity in gold data	13179	8408
#Entity in prediction	12993	12782
#Correct Entity	9813	3527
Entity precision	0.7553	0.2759

Entity recall	0.7446	0.4195
Entity F	0.7499	0.3329
#Correct Sentiment	9145	2686
Entity Type precision	0.7038	0.2101
Entity Type recall	0.6939	0.3195
Entity Type F	0.6988	0.2535

Papers We Referenced For Part 5:

A Second-Order Hidden Markov Model for Part-of-Speech Tagging:
https://www.aclweb.org/anthology/P99-1023.pdf

Log-Viterbi algorithm applied on second-order hidden Markov model for human activity recognition:
https://journals.sagepub.com/doi/pdf/10.1177/1550147718772541

Second-order Belief Hidden Markov Models:
https://arxiv.org/pdf/1501.05613