JR Latta (jrl336), Stephanie Wang (qw79)

CS 4740 Project 3 NER tagging using HMM

Final Report

Sequence Tagging Model

A. **Implementation Details**

We chose to implement a HMM from scratch. We implemented the Viterbi Algorithm for HMM.

The basic idea is that for each word in the test set, we calculate a score for it based on our training model. To be more specific, the algorithm follows the steps below:

1. score = P(each NER being the start of the sentence) * P(word token | each NER) → Find the NER that gives the maximum out of all the probabilities and save it for step 2.
2. score = score from step 1 * P(each NER | NER selected from step 1) * P(word token | NER selected from step 1) → Find the NER that gives the maximum out of all the probabilities and save it for step 3.
3. score = score from step 2 * P(each NER | NER selected from step 2) * P(word token | NER selected from step 2) → Find the NER that gives the maximum out of all the probabilities and save it for step 4.
4. … The iteration continues until the end of the test set.

Using this method, we were able to calculate a score of each NER for each word. To ensure efficiency in lookup time for lexical and transitional probabilities, we built two hashtables with tree structures. The two major data structures are illustrated below:

WORD TOKEN	NER	PROBABILITY
	NER	PROBABILITY

e.g. CRICKET	B-LOC	0.02
	O	0.82

Figure 1 Lexical_Probability Hashtable

NER	NER	PROBABILITY
	NER	PROBABILITY
e.g. B-PER	B-LOC	0.02
	O	0.82

Figure 2 Transitional_Probability Hashtable

Using this data structure, we were able to efficiently look up probabilities when we calculated scores during testing.

B. Pre-processing

For preprocessing, we used standard python library functions to iterate through the training text file. We parsed the lines into appropriate data structures to capture all the information relevant to a word token while maintaining ordering.

We created a training set using hashtable. This hashtable has the (word token, its NER) as the key and its count as the value. By doing this, we can easily count the number of seen-once NERs and use it to insert unknown tokens into the training set.

In order to deal with unknown word tokens in the test set, we inserted ‘UNK’ tokens into the training set:

```
lexicon[("UNK", "B-PER")] = B_PER_ONCE
lexicon[("UNK", "B-ORG")] = B_ORG_ONCE
lexicon[("UNK", "B-LOC")] = B_LOC_ONCE
```

```
lexicon[("UNK", "B-MISC")] = B_MISC_ONCE
lexicon[("UNK", "I-PER")] = I_PER_ONCE
lexicon[("UNK", "I-ORG")] = I_ORG_ONCE
lexicon[("UNK", "I-LOC")] = I_LOC_ONCE
lexicon[("UNK", "I-MISC")] = I_MISC_ONCE
lexicon[("UNK", "O")] = O_ONCE
```

The number of 'UNK' tokens for each NER category is decided based on the number of seen-once NERs in that category. This allows us to avoid 0 probability as we go through test set when we encounter a word that never showed up in the training set.

C. Experiments

The first experiment we ran was for dealing with unseen words in lexical probability table. We realized that not all the NERs are the start of the sentence in the training set. In fact, only 5 out of 9 NERs (B-MISC, B-ORG, B-PER, B-LOC, and O) were ever the start of the sentence. Therefore, when we were looking for a probability of (word token | given NER), the entry may or may not exist in our lexical probability table. Therefore, we made an assumption P(word token | given NER) is more correlated to P(word token | existing NER). As such, for those given NERs that didn't have corresponding P(word token | given NER) entry, we used P(word token | existing NER). We expected this approach to handle all the cases and not give us an empty scores list.

The second experiment we ran was to alter the first experiment and assume instead that P(word token | given NER) is more correlated to P(given NER being the start of the sentence). We again expected the approach to handle all the cases and not give us an empty scores list.

The third experiment we ran was changing how to select the intervals for the output. For example, we decided that if an 'O' marked word was followed by a 'I-***' word then our system would handle this word as a 'B-***' word.

The fourth experiment we ran was with smoothing to handle unseen word-tag or tag-tag combinations. We inserted a very small value for all the unseen combinations.

D. Results

Attempt	Validation	Test
1	40 (Baseline)	40.2 (Baseline)
2	42	41
3	43	42.3
4	48	45
5	46	44.2

Table 1 Results

Our baseline system is rather rudimentary. We built a lexicon of all the named entities in the training data and during testing only those named entities that are part of the lexicon. We didn't deal with unknown words in the baseline system.

Attempt 2: (first real attempt) Did not have great improvements over the baseline. We believe our model didn't score well because it had a simple rule for creating the interval for the NERs.

Attempt 3: This did not have significant improvements either. We determined that there must be some structural problems with our model.

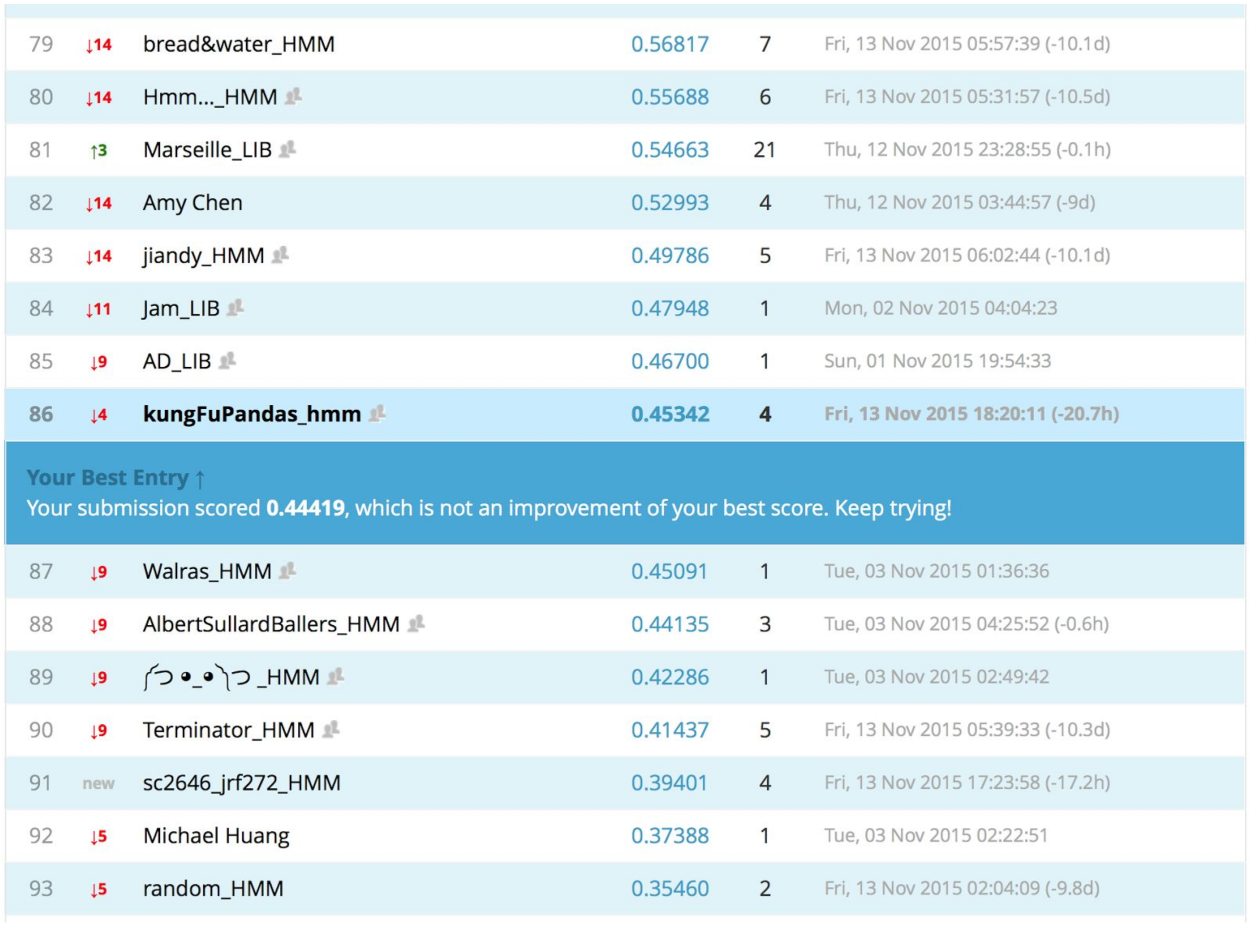
Attempt 4: Sadly, our model didn't score significant improvements again. We believe our model doesn't handle ambiguous words very well (or at all).

Attempt 5: Our new smoothing method actually hurt our model :(
It was rudimentary but we predicted it to have very little effect (+-3%) on our score and we were correct in that sense.

All of our attempts beat our naive baseline system but not by that much. We must have done a poor job implementing the HMM because our model should have scored significantly better than the naive baseline.

E. Competition Score

Our team name is **KungfuPandas_HMM**.
Here's the screenshot of our best performance:



79	↓14	bread&water_HMM	0.56817	7	Fri, 13 Nov 2015 05:57:39 (-10.1d)
80	↓14	Hmm..._HMM	0.55688	6	Fri, 13 Nov 2015 05:31:57 (-10.5d)
81	↑3	Marseille_LIB	0.54663	21	Thu, 12 Nov 2015 23:28:55 (-0.1h)
82	↓14	Amy Chen	0.52993	4	Thu, 12 Nov 2015 03:44:57 (-9d)
83	↓14	jiandy_HMM	0.49786	5	Fri, 13 Nov 2015 06:02:44 (-10.1d)
84	↓11	Jam_LIB	0.47948	1	Mon, 02 Nov 2015 04:04:23
85	↓9	AD_LIB	0.46700	1	Sun, 01 Nov 2015 19:54:33
86	**↓4**	**kungFuPandas_hmm**	**0.45342**	**4**	**Fri, 13 Nov 2015 18:20:11 (-20.7h)**

Your Best Entry ↑
Your submission scored **0.44419**, which is not an improvement of your best score. Keep trying!

87	↓9	Walras_HMM	0.45091	1	Tue, 03 Nov 2015 01:36:36
88	↓9	AlbertSullardBallers_HMM	0.44135	3	Tue, 03 Nov 2015 04:25:52 (-0.6h)
89	↓9	༼つ ◕_◕ ༽つ _HMM	0.42286	1	Tue, 03 Nov 2015 02:49:42
90	↓9	Terminator_HMM	0.41437	5	Fri, 13 Nov 2015 05:39:33 (-10.3d)
91	new	sc2646_jrf272_HMM	0.39401	4	Fri, 13 Nov 2015 17:23:58 (-17.2h)
92	↓5	Michael Huang	0.37388	1	Tue, 03 Nov 2015 02:22:51
93	↓5	random_HMM	0.35460	2	Fri, 13 Nov 2015 02:04:09 (-9.8d)

Extension

Our extension experiments with various smoothing methods. We used smoothing to handle unseen word-tag or tag-tag combinations.

The first way of smoothing we attempted was to pick out the highest probability value from the `lexical_prob[word]` for both the word at start of the sentence and the words in the rest of the sentence in order to avoid having an empty scores table. This smoothing method assumes that the NER of each word is highly correlated to the word's NERs in the training set. This smoothing method did not work extremely well as we obtained a F-measure score of 0.45. We suppose it's because we can't directly assume that P(word | given NER) is highly correlated to P(word | existing NER).

The second way of smoothing we attempted was to differentiate between the start of the sentence and the words in the rest of the sentence. To be more specific, instead of assuming the highest probability value from `lexical_prob[word]` for both the word at start of the sentence and the words in the rest of the sentence, we assumed the highest probability value from `NER_start_prob` for the word at start of the sentence and `lexical_prob[word]` for the words in the rest of the sentence. This smoothing method recognizes the difference between the start of the sentence and the rest of the sentence since not all NERs appeared at the start of the sentence. This smoothing method, as the result suggests, did not work very well either. We obtained a F-measure score that was marginally different from the previous one and was still 0.45. We suppose that the difference between the start of the sentence and the rest of the sentence is not as significant as we thought and we may need to resort to other smoothing methods.

The third way of smoothing we attempted was to include unseen word-tag and tag-tag probabilities into our lexical_probability and table. We chose to assign a small arbitrary probability for each unseen probability. We first added those entries to the lexical probability table as well as the transitional probability table and then we run the test using the new probability tables. We were hoping that by "filling in the blanks", the model could return a higher precision and recall value. However, as it turned out, this smoothing approach rendered our precision and recall marginally lower. On kaggle, we obtained a F-measure score of 0.44. We evaluate that our method of smoothing for unseen word-tag and tag-tag combinations was very naive and made poor mathematical sense. We added .000001 to each of these unseen combinations, which made our probabilities no longer a valid probability function. Due to this, our score actually decreased when we added this to the our system.

Individual Member Contribution

We worked on all aspects of the project together as a team.