CS 388 Assignment 2

Isha Joshi
imjoshi@utexas.edu
Yineng Yan
yinengy@utexas.edu

1 Model

We have implemented sequence labeling algorithms - HMM(Hidden Markov Model) and Conditional Random Fields(CRF) in this assignment. The input was in the format of a list of tokens and its corresponding labels which are one of the following - 'O', 'B-ORG', 'B-MISC', 'B-PER', 'I-PER', 'B-LOC', 'I-ORG', 'I-MISC', 'I-LOC'. These algorithms classify the entire sentence into a sequence of labels. HMM makes use of emission (probability that a given label occurs for a given token), transition(probability of the given label for a given previous label), and initial(probability that a given label is in the first position) in the form of log probabilities to avoid underflow. These probabilities are calculated using a count-based method. We have used the Viterbi algorithm for the decoding step which calculates the probability of a given sequence of labels using recursion. We have hard-coded some of the restrictions on the label sequences so that invalid sequences are not considered. For example - an I tag can only follow an I tag or B tag of the same type.

We also implemented CRF which is a generalized form of logistic regression for sequences. For feature extraction, we have used the following emission features - *WordFeatures* + *POSFeatures* + *WordShape* + *ShortShape* + *Suffix*. The total number of emission features used is 636,309. These features are thoroughly analyzed in the Analysis section. Transition probabilities have been initialized to 0 if it is a valid transition, otherwise a large negative number. Similarly, for initial probabilities, tags starting with I are initialized with a large negative number, otherwise zero. The weights have been initialized to 0. We have used an unregularized Adagrad trainer to update the weights. We implemented the forward-backward algorithm for calculating marginal probabilities in the training stage and the Viterbi algorithm for calculating the probability of a sequence of labels in the inference stage.

2 Results

The F1-score, Precision, and Recall obtained on the development set and test set are mentioned in the table1. This result was obtained after training for 5 epochs and using the *WordShape* + *ShortShape* + *Suffix* feature set. The *eta* of the optimizer is set to 0.75.

Set	F1	Precision	Recall
Development	87.98	89.69	86.34
Test	80.4	82.5	78.3

Table 1: CRF results

We run with different epoch (up to 5) to see how performance changes (*eta* is set to 1 at this time). And the performance of the experiment on the development set is in the figure 1. Where the x-axis is the epoch and the y-axis is the scores (F1, precision, and recall).

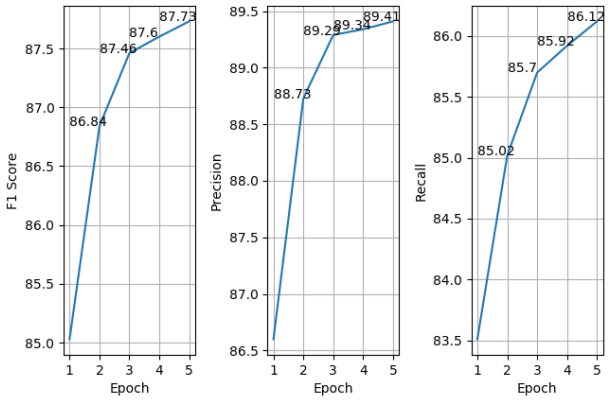

Figure 1: Performance with different Epoch

We can see that F1, precision, and recall increase when the epoch number increase. And it increases faster when the epoch is less than 3 and slower after 3.

The performance with respect to the changes of *eta* will be discussed in the last section.

3 Analysis

Remove	F1	Precision	Recall	Time
Nothing	83.89	85.2	82.62	21.68
WordShape	83.95	85.49	82.47	20.43
ShortShape	84.21	86.02	82.48	20.65
WordShape + ShortShape	83.55	85.9	81.32	18.36
Prefix	85.03	86.6	83.51	20.37
Suffix	83.38	84.70	82.10	21.75
Prefix + Suffix	84.27	85.94	82.67	18.26
All	80.98	84.61	77.64	17.88

Table 2: Ablation Study of New Features

One of the extensions we made is adding new features. We implemented 4 features templates and do an ablation study on these features using the dev set. The result is at table 2. In this table, column *Remove* refer to which part is removed in this run, and column *Time* refers to the decoding time (in seconds) to show how these features impact efficiency. This experiment is done with *epoch* 1 and unregularized adagrad trainer with *eta* equals to 1. The baseline to remove nothing, which means all features templates are used.

The first feature template is *WordShape*, which is generated by map the upper letter to X, lower letter to x, and digit to d. For example, CS388 maps to XXddd. The feature templates capture the shape of a word so an unknown word would be related to a known word with the same shape. And in this case, they typically have the same label. For example, if we seen a word ACL2020 labeled with B-ORG and maps to XXXdddd, any similar conference name like ACL2021 or HPC2021 could be discriminated easier. But it will not be able to build a relation between ACL2020 and ASPLOS2021 since they have different shapes.

The second feature template is *ShortShape*, which is generated by remove consecutive shapes in *WordShape*. For example, CS388 will maps to Xd this time. Compares to *WordShape*, it is simply the shape so more words could be maps to the same shape. For example, if we saw ACL2020 in the training set, not only AC2021, HPC2021 but also ASPLOS2021 could be discriminated easier. But it could not help to find the relation between ACL2020 and acl2020 since the first one maps to Xd while the second maps to xd.

In the experiment, we can see that removing *WordShape* or *ShortShape* would lead to a higher F1 score but removing all of them would give a lower F1 score and recall. This means having the two templates at the same time is not necessary. And removing *ShortShape* leads to a higher F1, precision and recall compares to removing *WordShape*. These results make sense since *ShortShape* is less precise because it maps more words to the same feature even if they are not related. For example, it will map X86 to Xd and ACL2020 to Xd too. But X86 is not related to ACL2020. In that case, the performance is worse compared to *WordShape* since the latter will map X86 and ACL2020 to different shape.

And the time is increase by around 1 second when we remove one of them and 3 seconds when both are removed, which means add these features would only slightly decrease the efficiency, which is still acceptable.

The third feature template is *prefix*, which uses a word's first two characters to form a feature. For example, UTexas maps to UT. This may works since some related entities may share the same suffix. UTCS and UTMath are both maps to UT, so they would have the same label when we use prefix as a feature. But it doesn't work to find out the relation between UMichCS and UTCS since the first two characters are different.

The last feature template is *suffix*, which uses a word's last two characters to form a feature. For example, USA maps to SA. This is similar to *prefix*. One example is UTCS and UMichCS are both maps to CS, so they would have the same label when we use prefix as a feature, which is what we want since they are CS departments at different universities.

Removing *prefix* leads to an increase in F1, precision, and recall. This means *prefix* doesn't work well at this dataset. And removing *suffix* decrease the F1, precision, and recall. This means not like *prefix*, *suffix* works better at this dataset. The decoding time is differed by around 1 second, which means add these features doesn't have a significant impact on efficiency. When we remove both of them, the F1 scores and Precision increase by a little bit. It means applying these two features templates at the same time is not that necessary. To explain why *prefix* is worse than *suffix*, we need

to look like the dataset. We can notice that there are many words with the same suffix like ”an” (e.g. Indian and Japan). But sharing the same prefix is not that common. Even worse, sometimes the word sharing the same first two characters are unrelated (e.g. Sakakibara and Salad). This will leads to better performance on *suffix* than *prefix*.

In the end, we run an experiment with removing all of these additional features. This time we get the lowest F1, precision, and recall. This means adding these features could improve accuracy by a lot.

And based on the result, we choose to use *WordShape* + *ShortShape* + *Suffix* in our final model. It achieves the highest performance among these runs. The time increase by 3 seconds compared to add nothing, which is acceptable.

4 Implementation Details

We have trained the final model for 5 epochs. The weights are updated using an unregularized Adagrad trainer with *eta* equals to 0.75.

The choice of the number of epoch is made by observing result in figure 1, which shows that epoch 5 achieves the highest performance. Epoch number higher than 5 may get better performance but the training time will be over 1 hour, which is not acceptable. **So we choose 5 as the final value of number of epoch**.

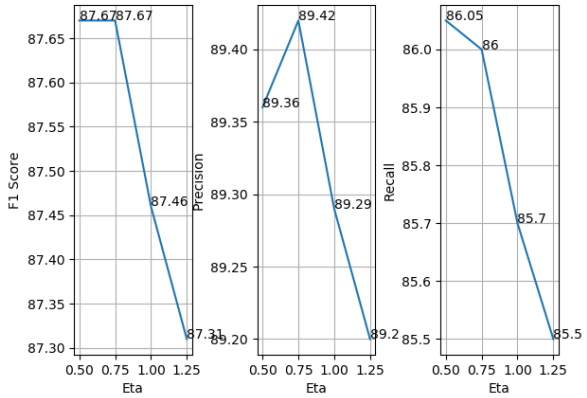

Figure 2: Performance with different Eta

To decide the value of *eta*, we do an experiment that fixing epochs at 3 and changes eta to see how performance changes. The result on development set is reported in figure 2.

We can find that when *eta* increase and is larger than 0.75, F1, precision and recall decrease. *eta* 0.5 and 0.75 has the same F1 and recall (differ by 0.05), but precision is higher when *eta* equals 0.75. **So we choose to use 0.75 as the value of *eta***.