Project 1 Named Entity Recognition Report

Chenkuan Liu
Natural Language Processing
UT Austin
Feb 2021

1 Introduction

This project uses Hidden Markov Model (HMM) and Conditional Random Field (CRF) to perform named entity recognition on CoNLL 2003 Shared Task dataset. Compared to mini 1, we not only need to predict whether a token is part of a person's name, but also need to predict locations, organizations and miscellaneous named entities, as well as if it is the start or continuation of a named entity.

2 Hidden Markov Model

In Hidden Markov Model, the training set is used to generate the initial counts, emission counts, and transition counts regarding different tags and tokens, and then transform them into probabilities and store in logarithms. The decoding process is to find the tokens $\mathbf{y}$ associated with a given sentence $\mathbf{x}$ so that the conditional probability $P(\mathbf{y}|\mathbf{x})$ is maximized. From properties of conditional probability and Markov structure, we have $P(\mathbf{y}|\mathbf{x}) \propto P(\mathbf{y}, \mathbf{x}) = P(y_1) \prod_{i=2}^{n} P(y_i|y_{i-1}) \prod_{i=1}^{n} P(x_i|y_i)$. In this way, we can employ dynamic programming and use Viterbi algorithm to find the largest likelihood and the associated tags.

In my implementation, since the probabilities are stored in logarithm, all the computations inside Viterbi algorithm becomes addition inside of multiplication. In particular, I created an empty dictionary to store back-pointers at the beginning of decoding process. Inside the algorithm, whenever a max score is found, its associated tag will be stored into the dictionary as a key-value pair. For instance, suppose it is found out that, when calculating the score value of the first tag at the third token, and its value comes from the fourth tag at the second token, which is the max one, then the key-value pair will become `"1,3":4`. Specifically, the dictionary keys are transformed from integers to strings, and a comma is added inside so that each key is unique and associated with only one back-pointer.

After the last column is computed, its `argmax` value will be stored as the first element inside an array. And then, I will trace back the previous tags using the dictionary and append them into the array. In this way, the elements of the array are the predicted tags in reversed order. I will reverse it back and transform the tag number sequence into the corresponding bio tag sequence, and return it as the prediction. As a result, the F1 score achieved by HMM here is 76.89, which is the same as the instructors' reference implementation.

3 Conditional Random Field

The Conditional Random Field model provides more flexibility than HMM through its computation of potentials. In particular, we use linear feature-based potentials at here so that the formula becomes $\phi_k(\mathbf{x}, \mathbf{y}) = w^t f_k(\mathbf{x}, \mathbf{y})$. Therefore, at here, we have $P(\mathbf{y}|\mathbf{x}) \propto \exp w^t[\sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, \mathbf{x})]$. At here, I hard-coded the transition features such that all illegal transitions have values `-np.inf`, all "[Beginning, Beginning]" features (such as [B-LOC, B-LOC]) have values -100. To let each row approximately sum into probability one, I set all other transition values as $log(1/5)$. The emission features are based on indicators related to each tag at each position of the sentence.

Since the transition features are hard-coded, the gradient of likelihood becomes: $\frac{\partial}{\partial w}\mathcal{L}(\mathbf{y}^*, \mathbf{x}) = \sum_{i=1}^{n} f_e(y_i^*, i, \mathbf{x}) - \mathbb{E}_{\mathbf{y}}[\sum_{i=1}^{n} f_e(y_i, i, \mathbf{x})] = \sum_{i=1}^{n} f_e(y_i^*, i, \mathbf{x}) - \sum_{i=1}^{n}\sum_{s} P(y_i = s|\mathbf{x}) f_e(s, i, \mathbf{x})$. In order to compute the marginal probability inside emission feature expectation, we use dynamic programming and forward-backward algorithm so that $P(y_i = s|\mathbf{x}) = \frac{\text{forward}_{i(s)}\text{backward}_{i(s)}}{\sum_{s'}\text{forward}_{i(s')}\text{backward}_{i(s')}}$.

In my implementation, both the hard-coded transition weights and the forward-backward algorithm are implemented inside the HmmNerModel class due to their similarity to the original HMM transition values and Viterbi decoding algorithm. In this way, at the beginning of CRF model training, I initialized an object of HmmNerModel class. However, since I am not using the values provided inside the original HMM, only tag indexers are passed into the object creation, while all other parameters are dummy initializations. After that, during training, I transformed all feature indicators into Counter objects to accelerate sparse computation. In addition, to avoid overflow, I stored all values in logarithm, so that the usual addition becomes `logaddexp` and usual multiplication becomes "+"s, and, the original zero initializations become `-np.inf` and the original one initializations become zeroes. After all terms are computed, I apply gradient descent to update the weight values.

I am using the unregularized Adagrad optimizer. The initial weights are all zeros and the learning rate is fixed to 1. The decoding process is the same as the HMM part, with the original emission and transition log probabilities replaced by CRF feature scores. After training for 1 epoch on all samples, the F1-score reaches above 84. After training for 3 epochs on all samples, the F1-score surpasses 87. The performance on development set is demonstrated in detail in the table below:

Table 1: CRF performance

Epochs	F1-score	Precision	Recall
1	84.12	85.14	83.12
3	**87.84**	89.00	86.71

The 3-epoch implementation with F1-score 87.84 is used to predict and write the labels into the blind test set `eng.testb.out`.

4 Extension

To extend on the original CRF model, I am experimenting with new emission features during feature extraction. Since the training time becomes much longer if we increase the training epochs, for the sake of comparison, all modifications will be tested with only 1 epoch training. In other words, the baseline performance here is 84.12 F1-score from 1-epoch training in Table 1, and we will compare whether the modified features will lead to higher or lower values than that. If one modified version surpasses all others, it will be further trained with 3-epochs and compared with the original 87.84 F1-score.

4.1 Change word shape extraction

In the original feature extraction function, when computing word shape, only upper case letters, lower case letters and digits are considered, while all other characters will have "?" as their placeholders. However, by observing the training set, we can find that some special characters are more related to named entities. For instance, "." sometimes appears at the end of a token as abbreviations of locations or organizations (such as Inc., U.S.), or as someone's middle name (such as A. Stewart); "&" sometimes appears in the middle of a named entity (such as S&P), or itself appears as an "I-ORG" token (such as Steptoe & Johnson). Therefore, at here, I added a condition during word shape computation such that whenever a "." or "&" is encountered in a token, the "." or "&" itself will be added into `new_word` instead of the original question mark. After such extension, the result of running 1 epoch on all samples becomes: F1: 84.20; precision: 85.21; recall: 83.21. We see a slight increase in all measurements.

4.2 Add word length indicator

The word length sometimes can also be indicative of the token's identity. A word that is too short or too long are less likely to become a named entity. Therefore, I added indicators with respect to the word length. Note

that, for the purpose of comparison, the additional word shape features in 4.1 are not added here. After such extension, the result of running 1 epoch on all samples becomes: F1: 83.71; precision: 84.58; recall: 82.85. It turns out that all measurements become lower than the baseline performance.

One possible explanation is that adding each length as an indicator confuses the model, since if a word has length 10, there is no way to tell if it belongs to a named entity or not without further information. However, if a word only has length 1 or 2, it is more likely not a named entity. Therefore, I reduced the number of length indicators so that all tokens with length 4 or longer belong to the same category. In this way, we only have four indicators: I[$\texttt{word\_len} = 1$], I[$\texttt{word\_len} = 2$], I[$\texttt{word\_len} = 3$], I[$\texttt{word\_len} > 4$]. Using such extension, the result of running 1 epoch on all samples becomes: F1: 84.31; precision: 85.24; recall: 83.39. We see that all measurements are improved compared to both the baseline and 4.1 performance.

4.3 Combine together

From above, we see that both 4.1 and the second approach in 4.2 increase the performance compared to the baseline. Now we combine them together, namely adding both additional word shape extractions and the four indicators on word length. After running 1 epoch on all samples, the performance becomes: F1: 84.28; precision: 85.21; recall: 83.38. We see that it is actually lower than the second approach in 4.2. One possible explanation is that, in 4.1 we increased the chance of being named entities for tokens such as S&P, A., and "&" itself, but in 4.2 we decreased their chances as the lengths of such tokens are generally less than 4, and when the two extensions combined together, such contradictory features leads to lower performance than only adding four word-length indicators. However, note that the performance here is still better than the baseline and the 4.1 extension alone.

As a result, we will pick the second extension approach in 4.2 as our candidate and train for 3 epochs, and the result is: F1: 87.65, precision: 88.88, recall: 86.45. It turns out that, despite surpassing the baseline performance with only 1 epoch, after 3 epochs training, it becomes slightly lower than the baseline model performance (87.84 F1). In fact, after further experiments, of all the extended features here, none of them have surpassed 87.84 F1 after training for 3 epochs. From here, we can conclude that, though the extended features in 4.1 and 4.2 can help better recognize the named entities in fewer epochs, they become less effective as the training epoch increases.