CS388 Project 1: Sequential CRF for NER

Yixuan Ni, yn2782

Abstract

In this project, I implement the Viterbi algorithm to decode HMM model and forward-backward algorithm for decoding and training CRF models. I apply these models on the NER task using data in the CoNLL 2003 Shared Task and achieve a best performance of 76.89 F_1 score using HMM and 86.99 F_1 score using CRF. I improve these implementation by using a faster decoding method, beam search, and achieve 72.76 F_1 score and 85.03 F_1 score for HMM and CRF, respectively.

1 Introduction

The goal of this project is to implement a sequence tagger for the Named Entity Recognition (NER) task. NER is the task to identify named entities in text and classify them into predefined categories such as locations, person, organizations, etc. We build sequential models like Hidden Markov Model (HMM) or a Conditional Random Field (CRF) to compute the conditional probability of a sentence having a sequence of labels and thereby predict the most possible label for each word.

For this project, I use data in the CoNLL 2003 Shared Task. The data contains four columns: the word, the POS tag, the chunk bit, the NER tag. The NER tag is either an "O" tag, indicating the word is not an named entity, or a "B" tag, meaning that the word is the start of an named entity and "I" tag, meaning that the word is the continuation of the previous named entity. There are constraints related to the tags to be built into the model for a better performance. A function is given to extract features for each word in a sentence, and I use these features along with the NER tags to train the model.

The project includes three main parts: implementing the Viterbi algorithm to decode HMM model, implementing the forward-backward algorithm for decoding and training CRF model, and extensions to improve the CRF models. In the next sections, I will explain in details the implementations, the experiments I have done and compare the results.

2 Implementation

2.1 Viterbi Algorithm

To do inference for HHM, I follow closely to the lecture notes of CS378 to implement the Viterbi algorithm. The goal of the algorithm is to compute $argmax_y P(\mathbf{y}, \mathbf{x})$ given a sentence $\mathbf{x}$ and the corresponding labels $\mathbf{y}$. I maintain a matrix v to store this probability for each label at each word. The inputs are: S, initial log probabilities; T: transition probabilities; E, emission probabilities. For more convenient calculations, all probabilities are stored in log space. The algorithm has three major steps:

1. Intialization. Calculate $v[0, y] = S[y] + E[y, x_0]$ for all possible tags y at the first word x_0

2. Recursion. For all the other words x_i, all previous tags y_{prev} and all possible current tags y, calculate $v[i, y] = E[y_{prev}, x_i] + max_{y_{prev}}(T[y_{prev}, y] + v[i-1, y_{prev}])$

3. Final state. If a "STOP" token is added at the end of each sentence, calculate $v[n, y] = v[n, y] + T[y, STOP]$. For this implementation, since we do not use a "STOP" token, this step is ignored

I use a matrix to store a pointer indicating which previous tag has lead to a particular state at a particular word, for tracing back and reconstruct the hidden states at the end of the algorithm.

2.2 Forward-Backward Algorithm

The forward-backward algorithm is a generalization of the Viterbi Algorithm. It is used to calculate the marginal probability $P(y_i = s|\mathbf{x})$, which is then used to calculate the gradients (*ignoring the transmission features*):

$$\frac{\partial}{\partial w}\mathcal{L}(\mathbf{y}^*, \mathbf{x}) = \sum_{i=1}^{n} f_e(y_i^*, i, \mathbf{x}) - E_y[\sum_{i=1}^{n} f_e(y_i^*, i, \mathbf{x})]$$

where the expected emission feature is

$$E_y[\sum_{i=1}^{n} f_e(y_i^*, i, \mathbf{x})] = \sum_{i=1}^{n}\sum_{s} P(y_i = s|\mathbf{x}) f_e(s, i, \mathbf{x})$$

In this implementation, we use binary features $f_e(s, i, \mathbf{x})$ to simplify the calculation.

The forward-backward algorithm is similar to the Viterbi algorithm with two major differences:

1. Use two matrices $forward$ and $backward$ to store the values of $\alpha_t(s_t)$ and $\beta_t(s_t)$ as defined in the lecture notes

2. Use summation instead of argmax at the recursion step of the Viterbi algorithm

The normalizing constant, $Z = \sum_s forward_i(s)backward_i(s)$, is the same regardless of the current word. Since the last column of the $backward$ matrix is all zeros, I calculate Z at the last word. For debugging, I have the option of calculate Z at multiple words and assert their equality.

2.3 Decoding CRF

I also use Viterbi algorithm to do inference for CRF. The difference is, we combine the transition probabilities and the emission probabilities into scores, which are the weighted summation of features. There are also some changes to make considering the constraints on the transmission features:

1. The first tag cannot be an "I" tag. Set $v[0, y] = -\infty$ if y is a "I" tag

2. "O" tags cannot not be followed by "I" tags

3. "I" tags cannot not be followed by "I" tags of a different category

4. "B" tags cannot not be followed by "I" tags of a different category

If the previous tag y_{prev} and current tag y violates any of the constraints $2 - 4$, I set $score[x_i, y] + v[i-1, y_{prev}] = -\infty$. At the recursion step of the Viterbi algorithm, it will not choose the path with incorrect sequence when calculating argmax .

2.4 Beam Search

As an extension, I implement beam search to facilitate the argmax calculation in the Viterbi algorithm. A "Beam" class is provided to us as a priority queue data structure, where we can store top K tags with the highest probabilities/scores. Instead of a matrix v, I use a list of beams to keep track of the top K tags for each word in a sentence. At the recursion step, I will only need to consider the top K previous tags and hence reduce the computation time. Backtracking the path is simply finding the head of each beam in reverse order.

3 Experiments

3.1 Basics

I first implement the Viterbi and forward-backward algorithms and test their performance on the dataset. The F_1 scores on the development set are stored in Table 1.

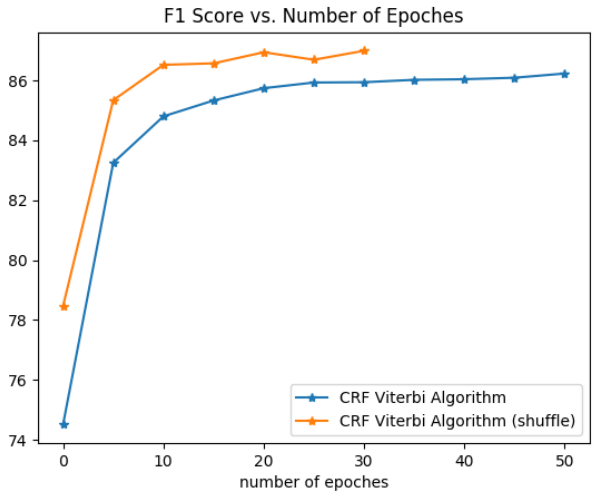

Figure 1

Figure 2: F_1 score on the development set for the CRF model using Viterbi algorithm, measured every 5 epochs. The yellow line is when data are shuffled for every epoch.

For the CRF model, I run the training step for 50 epochs, and evaluate every 5 epochs. The resulting F_1 scores are plotted in Figure 2. I use SGD optimizer with learning rate 0.1. The model achieves above 85 in 15 epochs, and increments slightly afterwards. The highest score has been reached at epoch 50. The model may perform better if we train it more; however, since each epoch requires about 1.5 minutes to run and I do not expect a huge increase after 50 epoch, I stop the experiments. Note that the model achieved over 74 F_1 score at the first epoch, and almost 84 at the 5th epoch, which means that the model can learn very fast.

To improve the performance, I shuffle the data for each epoch and train for 50 epoch. The results are also plotted in Figure 2. The model achieves 86.52 very quickly at epoch 10. I was only able to train 30 epochs in time, and the final model achieved 86.99 F_1 score. Training the model with different orders of sentences is indeed a major technique for such training tasks.

3.2 Extensions

I then implement beam search for decoding both HMM and CRF. I record the performance of using beam size from 1 to 9 (the total number of tags is 9) in Table 1. We can see that, as expected, the F_1 score increases as the beam size increases. A greedy approach (beam size of 1) gives an im-

Features	Beam Size	F_1
HMM	Viterbi	76.89
CRF	forward-backward	86.49
CRF (shuffle)	forward-backward	**86.99**
HMM	1	71.16
HMM	3	72.41
HMM	5	72.64
HMM	7	72.76
HMM	9	72.86
CRF	5	85.03

Table 1: Performance on the development set of different models. The first two results use Viterbi and forward-backward algorithm for decoding HMM and CRF, respectively. The others use beam search with different beam size.

pressive score of more than 70, considering it only keeps track of the tag with the highest score. I notice that even with beam size of 9, the performance is worse than the Viterbi algorithm. This may be due to a potential bug in the implementation; unfortunately, there has not been enough time to investigate more on this problem due to the time constraint of this project.

For CRF model, since it takes about 50 minutes to finish training 50 epochs, I only have time to test beam size of 5. It achieves 85.03 F_1 score, which is relatively close to the forward-backward decoding method.

4 Conclusion

In this project, I implement the Viterbi algorithm for decoding HMM model and forward-backward algorithm for decoding and training CRF models. I also implement Beam search for faster decoding for both models.

Some potential future works can be done involving more advanced optimization techniques, such as using AdaGrad Optimizer or using adaptive learning rate for different epochs. In addition, different combinations of features can be used for the CRF model. I can also implement structured SVM and compare the performance with CRF.