Implementation and Evaluation of Various Approaches to Named Entity Recognition Task

Wu Tianyu

August 2020

1 Implementation

1.1 Conditional Random Field (CRF)

Summarize and keep the following statistics of training data: number of different tags (T), number of different words (V), number of different Part-Of-Speech-tags (POS-tags) after reading data. Words, POS-tags and Named Entity Recognition tags are all converted to integers at the beginning to accelerate both learning and evaluation. All the essential algorithms (forward, backward, Viterbi, loss calculation and gradient calculation) are implemented in NumPy package to better exploit the slicing and broadcasting operation of NumPy arrays. Features are categorized into five categories: transition features (y_{i-1}, y_i), emission features $(y_i, word_i)$, POS-tag emission features (y_i, pos_i), combination features $(y_{i-1}, y_i, word_i)$ and POS-tag combination features (y_{i-1}, y_i, pos_i). Each set of feature weights are stored in the format of 2D-array. The shape of each weight matrix is listed in Table 1. For transition weight matrix, last index in the first dimension represents START and last index in the second dimension represents STOP. Likewise, the last index in the first dimension for combination weight matrix and POS-tag combination weight matrix represent START as well.

transition	emission	pos-emission	combination	pos-combination
(T+1, T+1)	(T, V)	(T, POS)	(T+1, T, V)	(T+1, T, POS)

Table 1: shape of five sets of feature weights

To avoid overflow, all algorithms are implemented in a log-sum manner. Logsumexp function is imported from scipy.special module to make code concise. Within forward, backward and Viterbi algorithm, the transition between tags are abstracted into a helper function called link_weight_sum. link_weight_sum returns the score change (matrix of shape $T \times T$) resulting from tag change. This abstraction helps a lot and makes code concise when more and more feature sets are incorporated into the algorithm. In the BFGS optmization algorithm, the feature weights and gradients are flattened and concatenated to be passed into fmin_l_bfgs_b function in scipy.optimize module.

1.2 Hidden Markov Model (HMM)

Specifically, if only transition features and emission features are considered and both transition weights and emission weights are set according to the counting statistics of training dataset ($-log(p)$ for existing pairs and $-inf$ for non-existing pairs), Conditional Random Field model is reduced to Hidden Markov Model. This point of view shows that CRF is a much more advanced and general model compared to HMM.

1.3 Structured Perceptron

The given dev.out file is split into validation set and test set with a splitting ratio of 0.5. The need for a validation set results from the fact that the performance (loss) on the training set fluctuates a lot as the algorithm runs. To reduce the fluctuation, a simple learning rate decay of $1/k$ is applied to the k^{th} epoch. The final weights are the ones with highest f1 score in validation set.

1.4 LSTM-CRF

This model is implemented in PyTorch and the initial word embeddings are obtained from BERT. Validation set is also required to avoid overfitting. The framework used is written by Allan Jie [1]. The idea behind LSTM-CRF is to generate the emission score via a bi-LSTM layer and a linear layer to better incorporate contextual information.

2 Evaluation

There are in total three combinations of feature sets: transition features + emission features (Features 1), transition features + emission features + POS-tag emission features (Features 2), transition features + emission features + POS-tag emission features + combination features + POS-tag combination features (Features 3). Features 1, Features 2 and Features 3 are all implemented for CRF while only Features 1 and Features 3 are implemented for structured perceptron. There are no selection of features for HMM and LSTM-CRF. The evaluation on the test set are listed in Table 2.

Models and Features	training precision	training recall	training f1	test precision	test recall	test f1
HMM	93.01	91.65	92.32	53.75	54.66	54.20
StrPercpt Features 1	93.63	90.55	92.06	75.00	50.81	60.58
StrPercpt Features 3	98.03	97.26	97.64	66.37	60.48	63.29
CRF Features 1	95.21	89.76	92.40	82.61	56.36	67.00
CRF Features 2	95.22	91.10	93.11	75.36	67.37	71.14
CRF Features 3	98.59	98.17	98.38	74.31	68.64	71.37
LSTM-CRF	-	-	-	75.53	75.85	75.69

Table 2: evaluation for various approaches and features

A steady improvement is observed from HMM, Structured Perceptron, CRF and LSTM-CRF. Generally, more features result in better performance if comparison is made within the same model. The CRF code has been optimized to achieve a fast training speed. It takes 1h:52min:06sec, 1h:47min:49sec and 2h:32min:55sec to train for Features 1, Features 2 and Features 3 respectively. Training of Structured Perceptron and HMM is a blitz compared to the training of CRF.

3 Reference

[1]: https://github.com/allanj/pytorch_lstmcrf/tree/3312f3498ac29767c172d2e8a37016b182f8385e