Named Entity Recognition with Semi-supervised Learning

Yu Ke, Wenbin Xiao, Changhao Han, and Jun Yang

Faculty of Applied Sciences, Simon Fraser University

Abstract

We reproduced three semi-supervised learning algorithms for named entity recognition (NER). One is improved Yarowsky algorithm with cautious, one is Yarowsky-based algorithm called DL_CoTrain and the other one is using conditional random fields (CRFs). The basic idea of these algorithm is that we first trained models from a small set of labeled data and then improved models with unlabeled data. We use precision and recall to evaluate the performances of our methods on both validation set which is separated from labeled data and test set. We explore the performances of semi-supervised leaning algorithms with using large unlabeled data and data from different domains.

1 Introduction

Named entity recognition (NER) or tagging is an important part of information extraction task in natural language processing which find named entities such as organizations, persons, locations, etc from text data. Whether a word is named entity (NE) or not can be largely determined by the context and its neighbors. We can either use word context features or neighborhood, or combine them together to determine named entities from context.

Semi-supervised learning utilize the unlabeled data by mitigating the effect of relatively small labeled data to relatively large unlabeled data. Training a well performed tagging model with only labeled data is costly as it requires sufficient human-annotated data. However, in most cases we can access sufficient unlabeled data much easily. Thus semi-supervised learning is a common approach for this kind of problems where we usually have a small labeled data and a large unlabeled data. With semi-supervised learning we are trying to make both labeled data and unlabeled data contribute to our trained model and get a better performed model with relatively low cost.

The most important part of semi-supervised learning algorithms is how we utilize unlabeled data to train the model. For Yarowsky Algorithm, originally, it is a supervised model based on Decision List. There are n labeled inputs, by using these inputs, we can calculate a Decision List Score for each of unlabeled data. Based on the score, which is probability of the label, the classification can be made. For Yarowsky-cautious Algorithm[2], the Decision List step keeps only the top rules, while the Yarowsky Algorithm treats everything that suitable as a rule. Besides, because Yarowsky-cautious is a semi-supervised version, the initial training data, called seeds, are in small quantity and are randomly selected from the label data. This will be further discussed in 2.1 and 3.2.

In DL-Cotrain [4], the method of utilizing unlabeled data is to label the unlabeled data via the relationship between them. The relationship can be defined as the the relationship between a word and its context. We maintain two decision lists: spelling decision list (spelling DL) and contextual decision list (contextual DL). In each iteration, we derive new rules in the contextual DL from the spelling DL and vise versa. The way we derive is based on the relationship between a

word and its context. This will be further discussed in 2.2 and 3.3.

As NER is often posed as a sequence classification problem, Liao and Sriharsha [3] suggested a semi-supervised algorithm based on conditional random fields (CRFs) in which we first get a trained CRFs model based on the labeled data. The trained CRFs model takes sentence of word features as input and return a sequence of predicted tags. Then in each iteration, we apply this model on unlabeled data and add most convinced part of predicted data with their predicted labels into training data set and retrain the model. The way we select most convinced data will be discussed later in section 3.4.

2 Methodology

2.1 Yarowsky

(Yarowsky 95) algorithm was originally proposed to solve the problem about word-sense disambiguation that exploits redundancy in contextual feature. Then (Collins and Singer 99)[2] modified it and applied it to the field of NER.

The core concept of Yarowsky Algorithm is the Decision List, which is an ordered list of feature label rules. Decision List can be constructed by assigning a score to each rule and then sorting on the score. The basic Yarowsky Algorithm can be shown in this pseudo code of Algorithm 1:

Algorithm 1: The basic Yarowsky Algorithm

Result: The Final Dcision List
Traing Data X and a random seed DL $\Delta^{(0)}$
Using $\Delta^{(0)}$ for X to label $Y^{(0)}$
while *iteration i to maximum or convergence* **do**
  train a new Decision List Δ on $Y^{(i)}$
  apply Δ to X to get $Y^{(i+1)}$
end

The Decision Score can be represented by a function *S*

$$S(x, y) = \frac{Count(x, y) + \alpha}{Count(x) + k\alpha} \tag{1}$$

Count(x,y) is the number of feature *x* shown in label *y*, and *Count(x)* is the number of all using feature *x*. α is a smoothing parameter, and *k* is the number of labels. Yarowsky can be illustrated in *figure 1*.

Figure 1: General Process of Yarowsky

When the Yarowsky algorithm is further modified to weaken its supervised feature, thereby strengthening the unsupervised feature, then Yarowsky-cautious algorithm can be obtained. Compared to Yarowsky algorithm, the main difference of the Yarowsky-cautious algorithm is about Decision List and the rules. The Decision List training step keeps only the top n rules over the threshold for each label instead of including all suitable rules. Because Yarowsky Algorithm adds a very large number of rules in the first few iterations. In this case, we can tell that Yarowsky-cautious algorithm is much "safer". Additionally, the thredshold Ω is checked against *Count(x,y) / Count(x)* and there is no more smooth parameters α. New seed DL is added to current DL every iteration after applying the cautious pruning. For the last iteration, Yarowsky-cautious algorithm applies the current labelling to

train a Decision List without cautiousness or threshold. Then this Decision List can be used to test its accuracy, which can be used for next loop of training, as well as re-training.[5]

2.2 DL-Cotrain

The idea of Co-training has been discussed by Avrim Blum and Tom Mitchell in their paper: Combining labeled and unlabeled data with co-training[4]. Here we give a brief introduction to this method.

For each example in the data set, we can describe it from two distinct views. In NER, we can describe a word via its spelling or its context. For example, in sentence "We arrived in London yesterday", to describe the word "London", we can directly use its spelling "London" or use its context like "arrived in". Both of them are sufficient to identify the entity type of "London" (a location). We call the spelling as "spelling features" and the contextual words as "contextual features"[2]. Given an unlabeled data set, we first manually label some words as our "seeds". The pair (word, label(word)) are called "decision rules". The collection of such rules are called "decision list" (DL). We now maintain our initial spelling DL from the seeds. Then we apply the spelling DL to the unlabelled data and get the training data. For each labeled word in the training data, we first find its context features. Here we use the two words behind and after the labeled word as our contextual features. We maintain a Count(f, l) where f stands for the feature and l stands for the label, Count(f,l) is the number of seeing feature f together with label l in the whole data set. When we get a contextual feature f of a word labeled with l, we let Count(f,l) += 1. After scanning the whole data set, we got a final Count(f,l). Then for each feature f, we compute a h(f,l) as follow:

$$h(f,l) = \frac{Count(f,l)}{\sum_{l_i \in L} Count(f,l_i)}$$

For each label l, we select features where h(f,l) is greater than some threshold (0.95 in our cases). Then we choose top n most common features from these features according to Count(f,l) and add them to the contextual decision list.

So far, we have discussed how to derive new contextual rules from existing spelling rules. The way of deriving spelling rules from contextual rules is almost the same. We loop through the data set and calculate Count(f,l) for each spelling features indicated by contextual DL and finally get the h(f,l) and the spelling rules to be added in the spelling DL.

In each iteration, we update contextual DL and spelling DL once respectively. As we iterates, the two DLs become larger and larger. We set n=n+5 to add more new rules in each iteration. When n hits a relatively large number (3000 in our cases), we can stop the loop and merge the spelling DL and the contextual DL as our final DL.

Here is a figure further illustrates this:

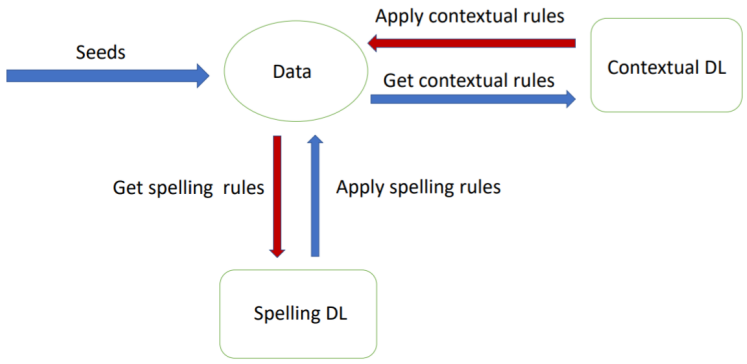


Figure 2: General Process of DL-Cotrain

2.3 Semi-supervised CRFs

The idea of applying CRFs model on NER is inspired by the paper of Liao et al. "A simple semi-supervised algorithm for named entity recognition"[3] We will give a brief introduction to its method in the following.

Our main idea is to use the CRFs model to do classification, however, CRFs are undirected graphical models trained to maximize the conditional probability of a sequence of labels ($Y = y_1...y_N$) given the corresponding input sequence ($X = x_1...x_N$). The conditional probability of Y given X is:

$$P(Y|X) = \frac{1}{Z(X)} exp(\sum_{n=1}^{N} \sum_{k} \lambda_k f_k(y_{n-1}, y_n, X, n))$$

where Z(X) is a normalization term, f_k is a feature function, and λ_k is a learned weight associated with the feature function f_k, which can be learned by maximizing the loglikelihood function which is given by

$$\sum_{i} logP(Y_i|X_i) - \sum_{k} \frac{\lambda_k^2}{2\sigma_k^2}$$

where σ_k is the smoothing (or regularization) parameter for the feature function f_k.

One big advantage of CRFs is that it can naturally represent rich domain knowledge with features. In our CRFs classifier, some most commonly used features are: the Token itself; whether Token is capitalized or not; whether Token is tile or not; whether Token is digit or not; Token's position of sentence ('POS'); and the Token's neighbors' features, etc.

Then we designed some rules to classify each token based on its features, which will take precedence over the CRFs. We assume that we have a small amount of labeled data L and a classifier C_0 that is trained on L. Then we exploit a large unlabeled corpus U from the test domain from which we automatically and gradually add new training data D to L, such that L has two properties: 1) accurately labeled, meaning that the labels assigned by automatic annotation of the selected unlabeled data are correct, and 2) nonredundant, which means that the new data is from regions in the feature space that the original training set does not adequately cover. Thus the classifier C_k is expected to get better monotonically as the training data gets updated.

Algorithm 2: The semi-supervised NER algorithm

Input:
- L - a small set of labeled training data
- U - unlabeled data

Loop for k iterations:
- Step 1: Train a classifier C_k based on L;
- Step 2: Extract new data D based on C_k;
- Step 3: Add D to L;

But how do we extract new data D? To do this, we should first compute a confidence score for each token. A confidence score can be computed using using the constrained forward-backward algorithm (Culotta and McCallum, 2004)[6], which calculates the sum of the probabilities of all the paths passing through the constrained segment.To simplify this, in our classifier, we use the final conditional probability instead.

We developed two ways to retain our CRFs classifier. One is to train on the whole unlabeled data set U; the other is to divide U into k separated segments and gradually add new unlabeled data to the training set. The ideas are similar, the following gives a brief method of the second way we applied.

Algorithm 3: Step2: Extract new data D based on Ck

1) Classify the k^{th} portion of U and compute confidence scores;
2) Find high-confidence Tokens and use them to tag other low-confidence Tokens
3) Find qualified O tokens
4) Add extracted data to D

In the first step, we partitioned U into k separated portions and compute the confidence score based on the old CRF model. Then we will find high-confidence Tokens and use them to tag other low-confidence scores. For example, we have a high-confidence ORG "**Safeway** Inc. ticked up ..." and low-confidence ORGs "1) **Safeway** shares ticked up ... 2) Wall Street expects **Safeway** to post earnings ..." etc. The high-confidence Token **Safeway** will be labeled as 'ORG' (Organization) with high probability because there is a **Inc.** after it, while the other two might be labeled as 'O' (Others) by the classifier. Here what we will do is to tell CRF to learn that since these words are the same, so the latter two '**Safeway**'s should also be labeled as 'ORG'. To balance the training data, we will also add some 'O' tagged Tokens to our training data if it has a high confidence score and does not have a label feature 'O'. At last, we will add these extracted data to D and retrain the CRF model to get a new classifier for the next iteration.

3 Experiment

3.1 Data set

We used the CoNLL-2003 as our main data set. The CoNLL-2003 data files contain four columns separated by a single space. Each word has been put on a separate line and there is an empty line after each sentence. The first item on each line is a word, the second a part-of-speech (POS) tag, the third a syntactic chunk tag and the fourth the named entity tag. The chunk tags and the named entity tags have the format I-TYPE which means that the word is inside a phrase of type TYPE. Only if two phrases of the same type immediately follow each other, the first word of the second phrase will have tag B-TYPE to show that it starts a new phrase. There is an example in figure 3:

```
U.N.          NNP  I-NP  I-ORG
official      NN   I-NP  O
Ekeus         NNP  I-NP  I-PER
heads         VBZ  I-VP  O
for           IN   I-PP  O
Baghdad       NNP  I-NP  I-LOC
.             .    O     O
```

Figure 3: The CoNLL-2003 Data Set Sample

3.2 Yarowsky

From (Collins and Singer 99)[2], it used the data that from New York Times with the parser of (Collins 96). This data has some unique properties for its context and features, but in our data, we do not have same properties, so we can not do every step in this paper. In this case, we define our own type of context and own features based on CoNLL 2003. For Yarowsky, I used its classification, spelling and some context together as the features of one entity. As for context, a NNP or NNPS word right after a labeled named entity is included. From the paper, it tells that Yarowsky-cautious can perform better than Yarowsky Alogrithm, however, after implementing the algorithm, the results of my experiment are not quite obvious. In the poster, the accuracy of Yarowsky and Yarowsky-cautious are around 0.80 and 0.85 respectively. But after that I found I have made some mistakes, for

that I did not consider some unlabel properties of Yarowsky-cautious. Then I did some revisions. At last the accuracy of Yarowsky is around 0.80 and the accuracy of Yarowsky-cautious is around 0.75-0.80, which shows that Yarowsky-cautious did some negative result. As far as I am concerned, the features differences between my data and data used for (Collins and Singer 99) are main influence. Because Yarowsky algorithm is a straightforward algorithm, then any revision on it can lead to good or bad result.

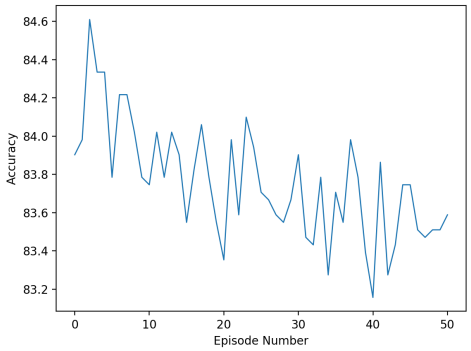
3.3 DL-Cotrain

If we follow the steps in Collins's paper [2], when testing with the labeled data, we only get the accuracy of 0.65, which is far lower than our expectation. After printing out the new rules with their Count(f,l) and h(f,l), we found that many of them are only with Count(f,l) equal to 1 or 2. This means that although the rules are of high support (greater than 0.95), the occurrences of them are pretty low. If we add such rules to our DL, there is a big chance to add some rules containing occasional relationship between words. The reason why the original method does not perform well for our case is that the way we extract the contextual features is different from Collins. Collins analyzed the whole sentence and extract the most relevant words as contextual features. This is not easy to implement. We defined the context to be a fixed-length region near the current word. This method is pretty straightforward but lacks semantic sense. To compensate the weakness of our method, we give a lower bound to the newly added rule in terms of their Count(f,l). Only those rules with Count(f,l) greater than 3 could be added to the DL. Now, we have constraints of support threshold, only selecting top n most common rules and a lower bound of Count(f,l). All these constraints avoid including occasional rules in DL. Test our model again, we get a decent accuracy of 0.84.

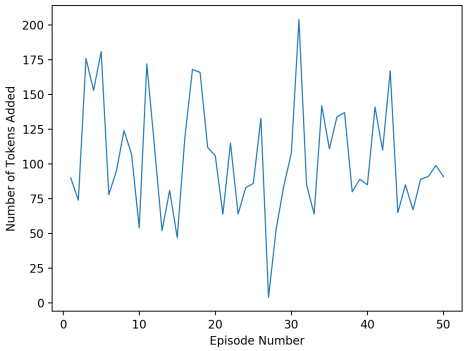
3.4 Semi-supervised CRFs

In our first version of training algorithm, in each iteration we add tokens that are not 'O' tagged with predicted confidence larger than predefined threshold and retrain the CRFs. However, this method does not help improved the accuracy of our CRFs model on validation set, but even make it worse after each iteration. By checking the detail of our added data during each iteration, we find that even with threshold been set to 0.98, we add around ten thousand tokens (extracted word features) into our data set.

Adding too many tokens each time could be the reason that our model performed worse as iteration goes (accuracy decreased nearly 4 percent within first several iterations). So we modified our algorithm to limit the number of tokens we add. To do this, we separate our unlabeled data set into batches of size 100 which means each batch contain 100 sentences and do the same operation within a single batch. Below are results with iteration been limited to 50.



(a) Accuracy over iterations



(b) Number of tokens been added over iterations

4 Conclusion

As is shown that Yarowsky performs slightly better than Yarowsky-cautious. Semi-supervised CRFs achieved better result than Yarowsky algorithm and DL-Cotrain algorithm. However, through the experiment result we can see that semi-supervised learning did not improved our model a lot but even make it worse as we adding more unlabeled data into training set. We think the main reasons for this problem could be that our labeled data and unlabeled data are not from the same domain and our data size could be too small. The difference of data domain cause that most information or model learned from unlabeled data are not relevant to our test data.
As for future work, we would like to try to get features structure with high similarity to the data used in paper (Collins and Singer 99) due to the deviation of our results and results in paper. Then try again to reproduce the results and ideas of this paper. Moreover, Collins and Singer's creative idea about cautiousness is of great significance, therefore, we believe that this idea and algorithm can be brought to other NER models and algorithms to observe whether it can play a role in improving accuracy.

References

[1] Yarowsky, David. "Unsupervised word sense disambiguation rivaling supervised methods." 33rd annual meeting of the association for computational linguistics. 1995.

[2] Collins, Michael, and Yoram Singer. "Unsupervised models for named entity classification." 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora. 1999.

[3] Liao, Wenhui, and Sriharsha Veeramachaneni. "A simple semi-supervised algorithm for named entity recognition." Proceedings of the NAACL HLT 2009 Workshop on SemiSupervised Learning for Natural Language Processing. 2009.

[4] A. Blum and T. Mitchell. "Combining labeled and unlabeled data with co-training." Proceedings of Computational Learning Theory. 1998.

[5] Max Whitney and Anoop Sarkar. "Bootstrapping via graph propagation." Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)

[6] Culotta and A. McCallum. 2004. Confidence estimation for information extraction. HLT-NAACL.

Contributions

Appendix

code repo: `https://csil-git1.cs.surrey.sfu.ca/irvingx/SSL_NER`

Contributions:

- Yarowsky algorithm - yu ke
- DL_CoTrain algorithm - Changhao Han
- Semi-supervised CRFs - Jun yang, Wenbin Xiao