Conditional Random Fields for Automatic Punctuation

Yufei Wang and Si Chen
Department of Electrical and Computer Engineering
University of California San Diego
{yuw176,sic046}@ucsd.edu

Abstract

We model the relationship between sentences and their punctuation labels using conditional random fields. Some feature functions are hand-designed and others are generated by templates. We train the same model by stochastic gradient ascent, Collins Perceptron and contrastive divergence respectively and compare their performance. On the provided dataset, we achieve word-level accuracy of 94.56%. At last, we propose a heuristic that can deal with cost-sensitive tasks.

1 Introduction

The problem of adding punctuation to English sentences is quite challenging. The punctuation symbol for each word depends on the whole sentence or even on specific context, which makes the model complicated. Also, for a given sentence, there may exist more than one correct punctuation. Table 1 shows some automatic punctuation results of our system and the corresponding groundtruth labels. The first example clearly shows the ambiguity of punctuation. The second example shows the unavoidable noise in the training set.

input	If you need any other information please let me know
output	If you need any other information, please let me know.
groundtruth	If you need any other information please let me know.
input	Can you further break it out by intramonth and term
output	Can you further break it out by intramonth and term?
groundtruth	Can you further break it out by intramonth and term.
input	Fuck me if I'm wrong but is your name Helga
output	Fuck me if I'm wrong, but is your name Helga.
groundtruth	Fuck me if I'm wrong but is your name Helga?

Table 1: Examples of the automatic punctuation system

In this project, we use a linear-chain conditional random fields (CRFs) to model the problem because of its ability to model complex conditional distribution. To train the parameters, we implement three different training methods, namely stochastic gradient ascent (SGA), Collins perceptron algorithm (Collins) and contrastive divergence (CD).

This report is organized as follows. Section 2 gives a brief overview of CRFs and the training methods. We describe the feature function design in Section 3. In Section 4, we describe the implementation details. Section 5 shows the experiment results with discussions. Finally, we draw some conclusions in Section 6.

2 Overview

2.1 Conditional Random Fields

If x is a training example and y is a possible corresponding label, the probability of y given x in the general log-linear model is

$$p(y|x;w) = \frac{exp\sum_{j=1}^{J} w_j F_j(x,y)}{Z(x,w)} \tag{1}$$

where $F_j(x,y)$ is a feature function and w_j is the corresponding weight demonstrating the importance of this feature function. $Z(x,w)$ is a normalizing factor, which is defined as

$$Z(x,w) = \sum_{y' \in Y} exp \sum_{j=1}^{J} w_j F_j(x,y') \tag{2}$$

CRFs are a special case of general log-linear models, which is originally proposed to model sequences of labels for sentences. In our task, we can assume that each feature function F_j is a sum along the word, from $i = 1$ to $i = n$ where n is the length of x:

$$F_j(x,y) = \sum_{i=1}^{n} f_j(y_{i-1}, y_i, x, i) \tag{3}$$

By doing so, we can have a fixed set of feature functions even though sentences are typically not of fixed length. Inference for CRFs can be solved effectively using Viterbi algorithm. Let J be the number of feature functions and let m be the cardinality of the set of tags. If the length of the sentence is n, we can compute the optimal $\hat{y}$ in no more than $O(m^2nJ + m^2n)$ time. In practice, most feature functions are usually zero. As a result, J is typically much smaller.

2.2 Training Methods

The objective function for training via SGA is the logarithm of the conditional likelihood (LCL) of the set of training examples. The partial derivative of the LCL is

$$\frac{\partial}{\partial w_j} log p(y|x;w) = F_j(x,y) - E_{y' \sim p(y'|x;w)}[F_j(x,y')] \tag{4}$$

The first term in the right hand side can be easily calculated. And the second term (conditional expectation given the current model) can be calculated in $O(Jm^2n)$ time using forward and backward vectors. There are also two variants to approximate the expectation term, which will be discussed as follows.

2.2.1 Collins Perceptron

For each sentence x, we can first find the current best guess of the label

$$\hat{y} = argmax_y p(y|x;w) \tag{5}$$

and replace the expectation term in Equ. 4 with $F_j(x,\hat{y})$.

This makes it a lot easier to calculate the second term. However, as is discussed previously, the time required for finding the best y (inference) is almost as much as the time required for calculating the expectation.

2.2.2 Contrastive Divergence

Instead of choosing the label with the highest probability ($\hat{y}$) given the current model, another option is to choose a single y^* that is somehow similar to the training label y, but at the same time has high probability according to $p(y|x;w)$. The idea of contrastive divergence is to first do a few rounds of Gibbs sampling which starts at the training label y. The sample y^* will then be used to take the place of $\hat{y}$ in Collins perceptron. If we only do one round of Gibbs sampling (usually denoted as CD-1), the time complexity will be $O(Jm^2n + mn)$.

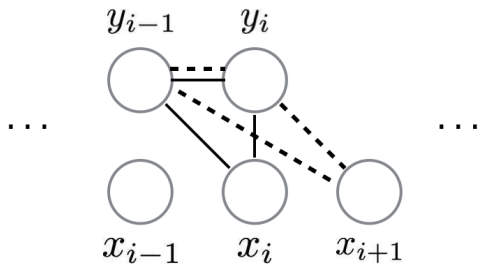

Figure 1: Template for generating feature functions. Solid line and dashed line represent A_{1a} and A_{2a} respectively.

	A	B
1	$I(lower(x_1) \in QuestionWord)$	$I(y_n =$ "$QUESTION\_MARK$")
2	$I(x_n =$ "yet")	$I(y_n =$ "$QUESTION\_MARK$")
3	$I(lower(x_n) =$ "re")	$I(y_n =$ "$COLON$")
4	$I(all\ the\ letters\ in\ x_i\ are\ capitalized)$	$I(y_i =$ "$COLON$")
5	$I(the\ first\ letter\ in\ x_2\ are\ capitalized)$	$I(y_1 =$ "$COLON$")
6	$I(x_{i+1} =$ "but" or "$please$")	$I(y_i =$ "$COMMA$")
7	$I(lower(x_1) =$ "$however$" or "$anyway$")	$I(y_1 =$ "$COMMA$")
8	$I(lower(x_1) =$ "$also$")	$I(y_1 =$ "$COMMA$")
9	$I(the\ last\ two\ letters\ in\ x_1 =$ "ly" $and\ \phi(x_1 =$ "ADV"))	$I(y_1 =$ "$COMMA$")

Table 2: The list of hand-designed features. n is the length of the sentence.

3 Feature Design

We have two kinds of feature functions. The first kind is defined via template and the second kind is hand-designed for some particular class.

For the template, we can assume that $f_j(\bar{x}, i, y_{i-1}, y_i) = A_a(\bar{x}) \cdot B_b(y_{i-1}, y_i)$ (since the sample x here is a sentence, we denote it as $\bar{x}$), where a ranges over a set $\mathscr{A}$ and b ranges over a set $\mathscr{B}$. In our project, we first map each word x_i into its part of speech (POS) tag $\phi(x_i)$. Two kinds of A_a function (See Fig. 1), denoted as A_{1a} and A_{2a} respectively, are used in our template.

We define A_{1a} function as $A_{1a}(\bar{x}) = I(\phi(x_i) = a)$ and define A_{2a} function as $A_{2a}(\bar{x}) = I(\phi(x_{i+1}) = a)$[1]. Here a ranges over the set of POS tags. B_b is defined as $I(y_{i-1} = b_1$ and $y_i = b_2)$, where $b = (b_1, b_2)$. b ranges over all punctuation pairs. The reason why we design this template is twofold:

- strong correlation between adjacent punctuation symbols
- strong correlation between the POS of a word and its surrounding punctuation symbols

In our program, i ranges from 1 to n. y_0 is defined as "START". Assume that the cardinality of the set of POS tags is p and that the cardinality of the set of labels is l, we can obtain $2 \times p \times l \times l$ feature functions via this template.

We also have 9 hand-designed features, which are listed in Table. 2[2].

4 Implementation

Algorithms are realized in Python.

[1] We define $\phi(x_{n+1})$ as a new POS class.

[2] QuestionWord = ["were","are","was","is","has","have","would","will","should","shall","could","can","how","what", "when","where","why","who","which","did","do"]

4.1 Optimization

In our implementation, there are 25 different POS tags and 7 different punctuation symbols (include "START"). So the total number of feature functions is 2459 ($2 \times 25 \times 7 \times 7 + 9$). Also, there are 70115 training samples in the training set. Given the large training set and a large number of feature functions, we use some optimization technique (besides the tricks mentioned above) to speed the training and testing procedure up.

Since it is inconvenient to deal with POS tags and punctuation symbols in their original form, we first map them (all in string type) into continuous integers starting from 0. By doing this, we can then effectively find the "active" (non-zero) feature functions generated via template given $(\bar{x}, i, y_{i-1}, y_i)$ in constant time. We take A_{1a} as an example. We can encode the active feature function index I given (a, b, c) (which are the POS index of x_i, the punctuation index of y_i and the punctuation index of y_{i-1} respectively) as

$$I = a + p \times b + p \times l \times c. \tag{6}$$

However, for our hand-designed features, we have to check whether they are active one by one. Fortunately, there are only 9 hand-designed features. By such feature encoding, we can greatly reduce the factor J in the time complexity expression to a very small number (In our project, we reduce this factor from 2459 to 11). When calculating $g_i = \sum_{j=1}^{J} w_j f_j(y_{i-1}, y_i, \bar{x}, i)$, we can use this trick to avoid loop over every feature function. This also works when we compute the expectation using

$$E_{\bar{y}}[F_j(\bar{x}, \bar{y})] = \sum_{i=1}^{n} \sum_{y_{i-1}} \sum_{y_i} f_j(y_{i-1}, y_i, \bar{x}, i) \frac{\alpha(i-1, y_{i-1})[expg_i(y_{i-1}, y_i)]\beta(y_i, i)}{Z(\bar{x}, w)}. \tag{7}$$

4.2 Verification

Forward and backward vectors

we check

$$\sum_{u} \alpha(k, u)\beta(u, k) = Z(\bar{x}, w) \tag{8}$$

for all positions k. Note that we do not use "STOP" tags, so we initialize $\beta(v, n) = 1$ for each v, which is different from the lecture notes.

Gradient

For SGA, we have to check whether the derivatives are correct. We use *check_grad* function in *scipy.optimize* package. For a given $(\bar{x}, \bar{y})$, we find that the derivatives are correct for all the corresponding "active" features.

4.3 Parameter selection

We use L_2 regularization for all the three methods. Also, we have to choose the learning rate for training except Collins perceptron. So there are totally 5 hyper-parameters that need to be determined. We search the best parameters by doing experiments on a small training set with 4000 training samples for 5 epochs and test the performance on a validation set of 1000 samples. The learning rates for SGA and CD are both set to 0.1 and the regularization weight(the weight for the regularization term in the objective function) is set to 10^{-5} (we use this setting for all the experiments in this report). We find that the performance is not very sensitive to the hyper-parameters as long as they are in a rational range.

4.4 Early stopping

To avoid overfitting, we use early stopping. We first choose an initial parameter *patience*. For every several iterations (*check_frequency*), we check the word-level accuracy on the validation set. If the current accuracy is higher than the best accuracy achieved, patience is reset:

$$patience = max\{patience, iteration\_number \times 1.5\}. \tag{9}$$

The training procedure stops when *iteration_number* = *patience*. In our experiments, *patience* is initialized as 60000 and $check\_frequency$ is set to 5000. We use one training sample for each iteration.

4.5 Overflow issue

In order to avoid overflow issue, we use log probabilities instead of probabilities. What's more, in this project, we often need to compute quantities of the form:

$$a = log \sum_t e^{b_t}, \tag{10}$$

where b_t is extremely large so that the above computation easily overflows. We deal with this problem by the "log-sum-exp" trick, which is simply:

$$log \sum_t e^{b_t} = log \sum_t e^{b_t} e^{A-A} = A + log \sum_t e^{b_t - A}, \tag{11}$$

where $A = max(b_t)$.

5 Experiments

5.1 Comparison between different training methods

We run experiments on a small training set with 4000 samples to see how fast these three methods converge (All the training sets we use for experiments are first shuffled randomly). For each method, we train the model for 8000 iterations (2 epochs) and check the accuracy on the training set after (1,2,3,4,5,...,14,15,250,500,1000,2000,4000,8000) iterations. The result is shown in Fig. 2. Although these three methods all have the stochastic nature, SGA converges much faster than the

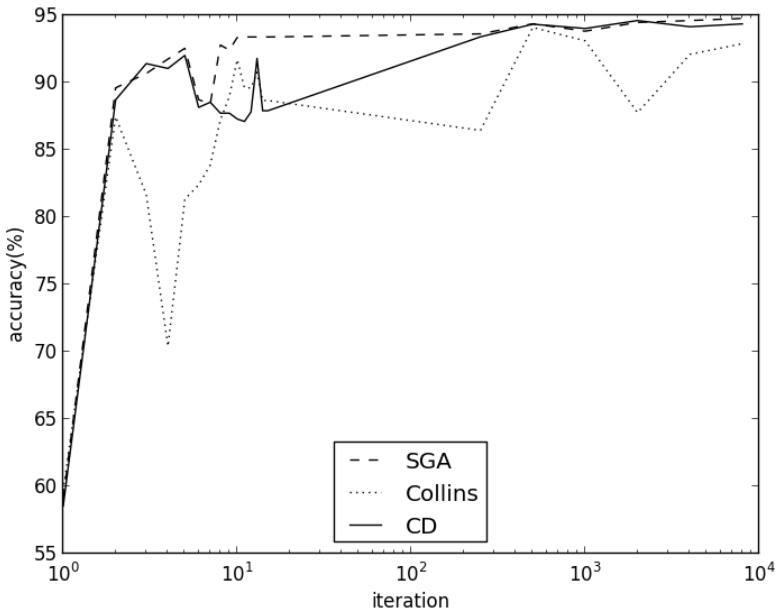

Figure 2: Convergence performance using the three different training methods

other two, because Collins and CD both use approximation when calculating the gradient for each training sample. Training time for this experiment is shown in Table 3.

We also train three models using the whole training set. We randomly sample 80% of the whole training set for training and use the remaining 20% for validation. Early stopping is used to avoid

SGA	Collins	CD
133.2	69.0	68.8

Table 3: Training time (seconds) of different methods (on a small dataset)

	SGA	COLLINS	CD	BASELINE
word-level accuracy(%)	94.40	93.62	94.56	93.46
sentence-level accuracy(%)	57.93	52.74	58.91	46.71

Table 4: Test performance for different methods

over-fitting. SGA takes 28 minutes for 60000 iterations. Collins takes 27 minutes for 75000 iterations. CD takes 21 minutes for 60000 iterations. (Note that the time here contains many validation operations for early stopping.) All the three models outperform the baseline (which is to simply predict "SPACE" for every word before the end and "PERIOD" at the end, of every sentence), as is shown in Fig. 3[3]. The model trained by Collins performs worse than the models trained by SGA, CD. More precise comparison is shown in Table 4. We think the reason is that Collins method only cares about the best guess of the model but ignores the specific distribution of the labels given the current model. If we get a better set of parameters of CRFs after an iteration, which means the LCL gets higher, the best guess may still stay the same. Therefore, unlike SGA and CD, which depend on the distribution, Collins method will not notice it. It is the "carelessness" of Collins method that makes the model less powerful compared to the others.

Although the three trained models all improve the word-level accuracy slightly, they greatly improve the sentence-level accuracy. As is shown in Fig. 3, the improvement mainly occurs in punctuation symbols which have relatively low frequency. Such improvement is of little importance for word-level accuracy but is crucial for sentence-level accuracy.

5.2 The importance of hand-designed features

The feature functions generated by templates alone cannot work well on this task. The main reason is that they are not discriminative enough for classification task. If we make the template to capture a relatively large scale dependence, we will have too much feature functions, which would lead to severe over-fitting (we try to involve dependence over 2 or 3 successive POS tags and see little gain). Therefore, feature functions designed using our own knowledge may help improve the performance.

We train three different CRFs with only hand-designed feature functions(HAND), only template-generated feature functions(TEMPLATE) and both(BOTH) respectively. All the three models are trained using SGA. The result is shown in Fig. 4, which clearly shows that both kinds of feature functions are important for this task. More specifically, the hand-designed features mainly contribute to the recognition of "QUESTION_MARK" and "COMMA".

5.3 Train the model by duplicating

Fig. 3 shows that the test accuracy of our model is quite class-dependent. The model works terribly on some classes like "COLON" and "EXCLAMATION_MARK". If we particularly care about the performance on a certain class (e.g. "COLON"), the algorithm will fail. This is a classic problem of cost-sensitive learning. The reason why the model fails on such class is that they rarely appear in the training set. Since the goal of the training algorithm is to maximize the LCL of the training set, it is rational that the model tends to ignore the accuracy for such class. As a result, we can create a new training set by duplicating the samples that contain the label we care about so that it will appear more frequently.

We try to use this heuristic to improve the accuracy of "COLON". We first find the sentences that contain "COLON" and duplicate them 30 times. The new training set contains all the copies and the original data. Then we train the model with the new training set using SGA. The accuracy improves

[3]The accuracy for each punctuation symbol α is the harmonic average of P and R, where $P = \frac{\#\,of\ correctly\ detected\ \alpha}{\#\,of\ \alpha(groundtruth)}$ and $R = \frac{\#\,of\ correctly\ detected\ \alpha}{\#\,of\ \alpha\ detected}$.

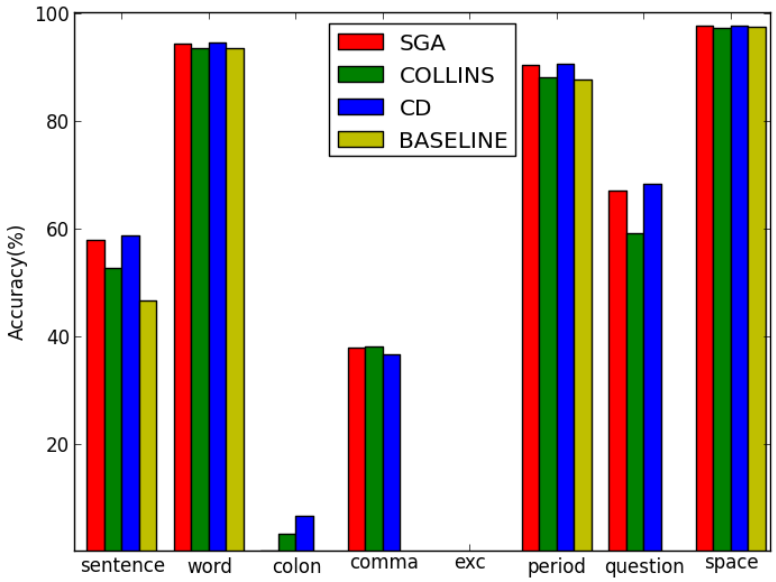

Figure 3: This plot shows the performance based on different training methods. We compare the sentence-level accuracy(sentence), word-level accuracy(word), as well as the accuracy for each class. We also include a strong baseline in the comparison.

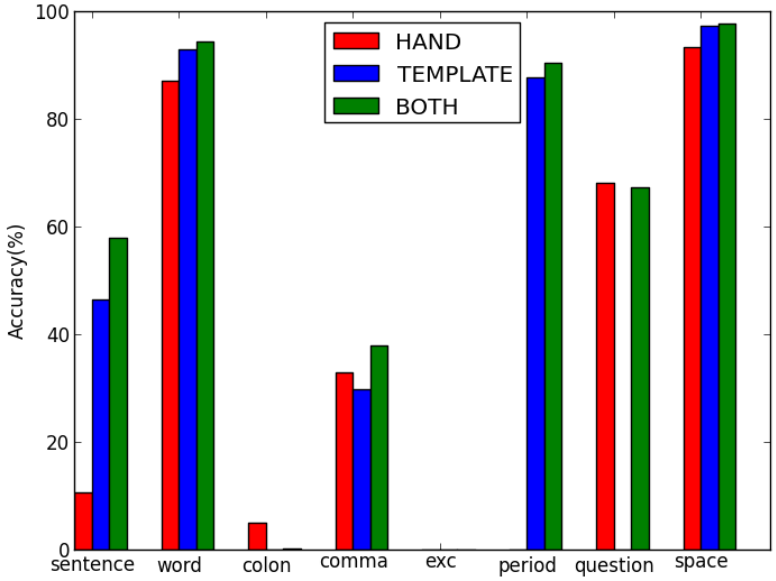

Figure 4: Test performance using different feature functions

greatly after duplicating, as is shown in Fig. 5. One disadvantage of duplicating is that the training set will be much different from the test set, which may hurt the test performance. However, at least in this experiment, the word-level accuracy is 94.33%, which is just slightly worse than the one using the original training set(94.40%).

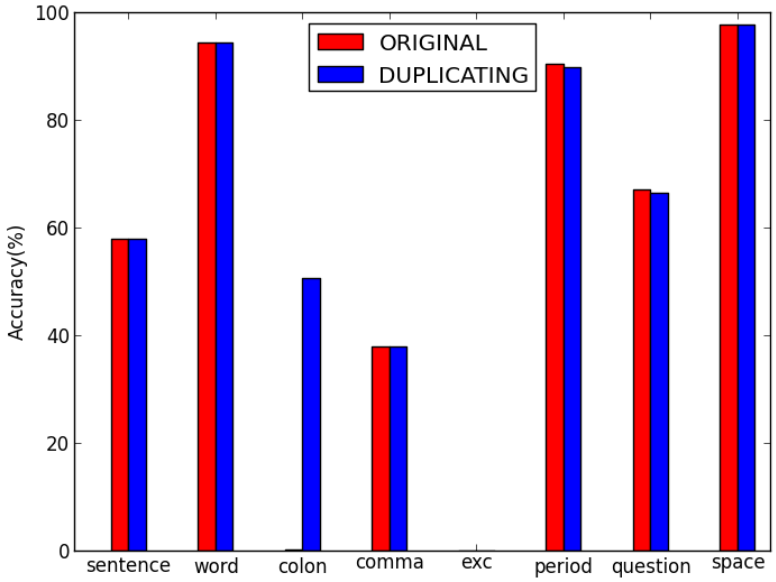

Figure 5: Test performance using the original training data and the new training data

6 Conclusions

In this project, we realize a fast implementation of CRFs and test it on the provided dataset. We train the model by three different methods. From the experiment result, we find that contrastive divergence is slightly better than stochastic gradient ascent because it is faster and can achieve equivalent performance on the test set. The model trained by Collins perceptron is the worst for its "carelessness". Feature design is crucial for the system performance. Human-engineered feature functions can make great improvement since they are more discriminative. Duplicating can help improve class-specific accuracy. Also, overflow can be effectively solved by the logsumexp trick.

References

[1] Elkan, C.:Log-linear models and conditional random fields. (February 8, 2013)