Abstract

We investigate the task of punctuation prediction in English sentences without prosodic information. In our approach, stochastic gradient ascent (SGA) is used to maximize log conditional likelihood when learning the parameters of linear-chain conditional random fields. For SGA, two different approximation techniques, namely Collins perceptron and contrastive divergence, are used to estimate the update step size. We construct 672 feature functions for our punctuation prediction model on a dataset with 70,115 training examples and 28,027 test examples. Experimental results show that SGA with Collins perceptron and SGA with contrastive divergence yield 7.11% and 13.89% word level error rate, respectively.

1 Introduction

In this paper, we investigate the task of punctuation prediction in English sentences without prosodic information. Nowadays, more users use mobile devices to compose email and text messages where the screen cannot show both letter and punctuation marks on keyboard. Users are forced to switch between letter and punctuation keyboards in order to insert punctuation. We propose an automatic punctuation prediction algorithm based on linear-chain conditional random fields (CRFs). By using this system, users can focus on typing sentences and have not to worry about adding punctuation. Take an English sentence for example, sentences without and with predicted punctuation are shown as follows

You are my friend I am yours and I am deeply honored
You are my friend, I am yours, and I am deeply honored!

Linear-chain CRFs have been widely used in various sequence labeling and segmentation tasks. The conditional probability of the model is

\[ p(\hat{y}|\bar{x}; \omega) = \frac{1}{Z(\bar{x}; \omega)} \exp \sum_{j=1}^{J} \omega_j F_j(\bar{x}, \hat{y}) \]  

where \( \hat{y} \) is a label, which is a sequence of punctuation tags in our application, and \( \bar{x} \) is an example. \( \omega \) is the parameter vector with length \( J \), and \( F_j(\bar{x}, \hat{y}) \) is the \( j \)-th feature function that returns a real value. \( Z(\bar{x}; \omega) \) is the normalizing factor

\[ Z(\bar{x}; \omega) = \sum_{\hat{y} \in Y} \exp \sum_{j=1}^{J} \omega_j F_j(\bar{x}, \hat{y}) \]  

where \( Y \) is the set of all possible labels. Therefore, the conditional probability is well defined. The prediction of the label \( \hat{y} \) given an example is computed as follows

\[ \hat{y} = \arg \max_{y} p(\hat{y}|\bar{x}; \omega) \]
Therefore, the training objective is to learn a parameter $\omega$ which gives the best prediction of $\hat{y}$. One of the most important aspects of linear-chain CRFs is the feature function $F(x, y)$. The detail of the feature functions in our approach is explained in Section 3.

The rest of this paper is organized as follows. The time complexity of the algorithm we use is analyzed in Section 2. Section 3 describes the details of our experiment, and experimental results are presented and discussed in Section 4. We conclude our findings in Section 5.

2 Analysis of Algorithms

2.1 Stochastic Gradient Ascent

In order to learn the $\omega$ parameter, we use stochastic gradient ascent (SGA) to iteratively update $\omega$ where each update is based on one example. The update rule for each $\omega_j$ is

$$
\omega_j := \omega_j + \lambda(F_j(x, y) - E_{y' | x, \omega}[F_j(x, y')])
$$

(4)

Evaluating the expectation in the above equation is computational intensive. Therefore, we use two different approximation techniques, Collins perceptron and contrastive divergence, to estimate this value. These two methods are described in the next two subsections.

2.2 Collins Perceptron

Collins perceptron simplifies the evaluation of expectation by putting all the probability mass on the highest $y$ value. That is, the update rule of SGA can be written as

$$
\omega_j := \omega_j + \lambda(F_j(x, \hat{y}) - F_j(x, \hat{y}))
$$

(5)

where $\hat{y} = \arg \max_y p(y|x; \omega)$. We use Viterbi algorithm to compute $\hat{y}$. It can be shown that the time complexity for finding $\hat{y}$ is $O(Jm^2n + mn)$, where $m$ is the cardinality of all possible punctuation tags, and $n$ is the length of $\hat{y}$. After finding $\hat{y}$, it takes only constant time to perform one update of $\omega$.

2.3 Contrastive Divergence

In contrastive divergence, $y^*$ is used instead of $\hat{y}$ to estimate the expectation, where $y^*$ is similar to true label $y$. By updating $y^*$, the probability of $y^*$ will be decreased while the probability of $y$ will be increased. To obtain $y^*$, we perform one round of Gibbs sampling starting at the true label $y$, which requires $O(Jm^2n + mn)$ time.

3 Design of Experiments

3.1 Dataset

We use English text dataset [1] which contains written sentences used in email messages. The dataset has 70,115 training examples and 28,027 test examples. For each word in an example, which is an English sentence, comes with exactly one punctuation tag. Therefore, we model the punctuation prediction problem as predicting the punctuation tag for each word.

The set of all possible punctuation tags is \{COMMA(,), PERIOD(.), QUESTION, MARK(?), EXCLAMATION, POINT(!), COLON:, SPACE( )\}

3.2 Parts of Speech Tagging

The original examples from the dataset are English sentences typed by the users on mobile devices. Since there are unlimited possibilities for what each word can be, it is difficult to construct feature functions that capture the relation between written sentences and punctuation. Therefore, we perform parts of speech (POS) tagging on the examples and train the model with finite set of POS tags as opposed to unbounded set of sentences. We use the Stanford Log-linear Part-Of-Speech Tagger [2] to transform the sentences to POS tags in preprocessing step.
3.3 Design of Feature Functions

Feature function \( F(x, y) \) is composed of two parts \( F(x, y) = A_a(x)B_b(y) \). The first part \( A_a(x) \) is used to detect POS tags. We design 16 different \( A_a \) functions by observation. For example, sentences start with "Wh-" word have the high probability to have a question mark in the end. Therefore, one example of \( A_a \) function may be \( A_a(x) = I(\text{POS}(x_1) = \text{WP}) \) where \( \text{POS}(x_i) \) denotes the POS tag for \( x_i \), \( I \) is the indicator function, and \( \text{WP} \) means Wh-pronoun in the definition of part-of-speech tags used in Penn Treebank Project [3]. Note that we only find cases for punctuation other than period and space, because in general case the last label will be period and others will be space in a sentence. Some examples for creating \( A_a \) include modal verb in the beginning, word before coordinating conjunction, and two consecutive nouns.

The second part of feature function is constructed as \( B_b(y) = I(y_{i-1} = c)I(y_i = d) \), where \( c \) ranges over 6 possible punctuation tags plus \( \text{START} \) tag, and \( d \) range 6 possible punctuation tags. Thus, there are total 42 combinations of \( B_b \) functions for consecutive two labels \( y_{i-1} \) and \( y_i \). This definition of \( B_b(y) \) allows the training algorithm to learn \( \omega \) for every combination of adjacent tags.

There is a trade-off between the number of feature functions and the computational complexity. Therefore, we pick the most recognizable feature functions \( A_a \) for prediction punctuation in common English sentences. However, there is an ambiguity between exclamation mark and period. Most of the time, sentences end with exclamation mark can also be ended with period. It causes the difficulty when predicting the exclamation mark.

3.4 Validation Set and Early Stopping

10,000 random examples from the training set is used as the validation set. After each epoch, we compute the error rate on the validation set. As the training go through epochs, the error rate on the validation set will first decrease, then increase caused by overfitting the training dataset. Therefore, we can stop the training process as soon as the model does not yield a less error rate on the validation dataset.

4 Experimental Results

4.1 Early Stopping

As shown in Figure 1, the error rate of training dataset will decrease and converge. However, in validation dataset, the error rate will decrease and increase back and forth. We decided to do early stopping when the second time validation error rate increase.

![Figure 1: Word level error rate after each epoch. The error rate increases after the fifth epoch, therefore, we stop the training process.](image)
4.2 Test Results

The error rate is computed based on word level. As shown in Table 1 for word level error rate, we compute the fraction of wrong tags in total number of tags as the error rate; for non-space tag error rate, we calculate the error rate on tags except spaces, since space is the most frequent tag in sentences. We also investigate the distribution of predicted tags for different tag categories, i.e. percentage of tags predicted as commas whereas the ground-truth tag is space. The result is shown in Table 2.

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Space</th>
<th>Comma</th>
<th>Period</th>
<th>Question</th>
<th>Exclamation</th>
<th>Colon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>97.53%</td>
<td>2.47%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Comma</td>
<td>77.43%</td>
<td>22.57%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Period</td>
<td>1.37%</td>
<td>0.04%</td>
<td>95.46%</td>
<td>3.13%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Question mark</td>
<td>0.00%</td>
<td>0.00%</td>
<td>47.75%</td>
<td>52.25%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Exclamation point</td>
<td>0.00%</td>
<td>0.00%</td>
<td>97.41%</td>
<td>2.59%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Colon</td>
<td>57.78%</td>
<td>42.22%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Table 1: The error rate on word level and on non-space tags. Two methods are SGA with Collins Perceptron (CP) and SGA with Contrastive Divergence (CD).

Table 2: The prediction rate on six different labels. Rows represent the ground truth label, and columns represent our prediction. The diagonal entries are the accuracy of each label category. The result is trained by stochastic gradient ascent with Collins perceptron method.

As shown in Table 2, the accuracy on space and period are very high, which is not surprising since those are the majority of the labels. The question mark has 52.25% accuracy while comma has 22.57% accuracy. Since we only choose 16 different POS tag combinations for the first part of feature function $A_v$, the result is acceptable. As for exclamation point, it is hard to discriminate from the sentences that end with period. The only feature function we design for predicting exclamation point is the imperative sentence, where the sentence starts with verb at base form. However, this may also be the case of an interrogative sentence. That explains why exclamation points are predicted as periods or question marks. Likewise, since our inputs for feature function are only POS tags, it is hard to tell the differences between the cases of colon and comma without information of boundary of sentences. Moreover, the feature function we use for finding colons is similar to the comma one. Therefore, colons are sometimes predicted as commas.

5 Findings and Lessons Learned

The first implementation issue we come across is the performance of the program. We use MATLAB for the implementation, and the bottleneck of our application is the for loop of in computing $g$ function and $F$ function. At first, it consumes 1 minute for each $\omega$ update. For a dataset with 70,115 training and 28,027 test examples, it takes 49 days to train and 19 days to test. We vectorize our low-level codes that drastically improves the efficiency up to 0.05 second per update, which takes an hour for training and 23 minutes for testing.

In our implementation, we first preprocess the sentences to POS tags, then do the training process. This method will lose some information compared to the original sentences. It makes the cases of exclamation point and colon hard to predict. On the other hand, this method uses less feature function, which can reduce the workload of string compare during the training process.

1Experiments are run on 2012-mid Macbook Air with 1.8Ghz Intel Core i5 processor.
Therefore, the computational complexity of our method is relatively low. Furthermore, to have a better accuracy, we believe that combining the POS tagging with the original sentence information into feature function is a better way. However, such strategy will incur more computational cost.

References

[1] Dataset available at https://d1b10bmlvqabco.cloudfront.net/attach/hpa8b6pdgyml/gx6tmtbtrr474/hr09hqbvcy2o/punctuationDataset.zip
