Hyphenation with Conditional Random Field

Panqu Wang(pawang@ucsd.edu)
Phuc Xuan Nguyen(pxn002@ucsd.edu)
Saekwang Nam(s9nam@ucsd.edu)

February 21, 2012

Abstract

In this project, we approach the problem of English-word hyphenation using a linear-chain conditional random field model. We measure the effectiveness of different feature combinations and two different learning methods: Collins perceptron and stochastic gradient following. We achieve the accuracy rate of 77.95% using stochastic gradient descent.

1 Introduction

The problem of English-word hyphenation is challenging. Given a word, the task is segment it into syllables using the BIO method of labeling, where the label 'B' is for 'begin' indicates a letter is the first one in its syllable, the tag O for 'out', and 'I' for 'in'.

We approach this task by modeling the problem as a linear-chain conditional random field(CRF). To train the parameters, we implement two different training methods, namely Collins' perceptrons and stochastic gradient following.

In section 2, we will briefly the theoretical overview of the model and the training methods. Section 3 will describe the feature design process. Section 4 describes the implementation concerns. Section 5 describes two main experiments to study the problem.

2 Theoretical Overview

2.1 Conditional Random Field

Let x be an training example, and let y be a possible label for x. The log-linear model posits that the probability of any particular label y, given the example x, is

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

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

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

Conditional random fields(CRFs) are a special case of log-linear models. The linear chain CRF is popular in natural language processing predicts sequences of labels for sequences of input samples. In order to specialize equation (1) for the task of predicting label of an input word, we can assume each feature function F_j is a sum along the word, for $i = 1$ to $i = n$ where n is the length of x:

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

Training a CRF means finding the weight vector w that gives the best possible prediction

$$\hat{y} = \text{argmax}_{\bar{y}} p(\bar{y} \mid \bar{x}) \tag{4}$$

for each training example $\bar{x}$.

2.2 Collins Perceptrons

For each word, x, and its syllable label, y, this algorithm consists of two processes, finding the current best guess

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

and updating the weights according to the guess

$$w_j = w_j + \lambda(F_j(x, y) - F_j(x, y)) \tag{5}$$

where $F_j(x, y)$ is the high-level feature function.

2.3 Stochastic Gradient Following

To learn the log-linear model, we have to choose the weights w_j that maximize the log conditional likelihood(LCL) of the set of training examples. According to the lecture notes(Elkan), the partial derivative of the LCL is

$$\begin{aligned} \frac{\partial}{\partial w_j} \log p(y \mid x; w) &= F_j(x, y) - \frac{1}{Z(x, w)} \frac{\partial}{\partial w_j} Z(x, w) \\ &= F_j(x, y) - E_{y' \sim p(y'|x,w)}[F_j(x, y^{'})] \end{aligned} \tag{6}$$

where $F_j(x, y)$ is the value of high-level feature function j for x and y, and the expectation term is the weighted average value of the feature function for x and

all possible labels $y^{'}$. We will use the forward and backward vectors to compute the expectation efficiently. That is,

$$\begin{aligned} & E_{\bar{y}\sim p(\bar{y}|\bar{x},w)}[F_j(\bar{x},\bar{y})] \\ = & \sum_{i=1}^{n}\sum_{y_{i-1}}\sum_{yi} f_j(y_{i-1},y_i,\bar{x},i)\frac{\alpha^T(i-1,y_{i-1})[\exp g_i(y_{i-1},y_i)]\beta(y_i,i)}{Z(\bar{x},w)} \\ = & \sum_{i=1}^{n}\frac{\alpha_{i-1}^T Q_{ij}\beta_i}{Z(\bar{x},w)} \end{aligned} \tag{7}$$

where

$$Q_{ij}(y_{i-1},y_i) = f_j(y_{i-1},y_i,\bar{x},i)[\exp g_i(y_{i-1},y_i)], \tag{8}$$

$\alpha(i-1,y_{i-1})$ is the forward vector whose base case is $\alpha(0,y) = I(y = \text{START})$, $\beta(y_i,i)$ is the backward vector whose base case is $\beta(u,n+1) = I(u = \text{STOP})$.

The recursive definitions are

$$\alpha(k+1,v) = \sum_u \alpha(k,u)[\exp g_{k+1}(u,v)] \tag{9}$$

, and

$$\beta(u,n+1) = \sum_v [\exp g_{k+1}(u,v)]\beta(v,k+1). \tag{10}$$

Using the forward and backward vectors, we can also easily calculate $Z(\bar{x},w)$:

$$Z(\bar{x},w) = \sum_v \alpha(n,v). \tag{11}$$

After we get the expectation value, we update the weight w_j using stochastic gradient ascent, which is

$$w_j := w_j + \lambda(F_j(x,y) - E_{y^{'}\sim p(y^{'}|x,w)}[F_j(x,y^{'})]) \tag{12}$$

where λ is the learning rate.

As we know, sometimes some w_j will go to infinity if we follow the gradient and the derivative is always positive, and it will in result the problem of overfitting. The update rule with regularization is

$$w_j := w_j + \lambda[(F_j(x,y) - E_{y^{'}\sim p(y^{'}|x,w)}[F_j(x,y^{'})]) - 2\mu w_j] \tag{13}$$

where λ is the learning rate.

In order to check the gradient reaches the global maximum, we have

$$\sum_{<x,y>\in T} F_j(x,y) = \sum_{<x,\cdot>\in T} E_{y\sim p(y|x,w)}[F_j(x,y)] \tag{14}$$

where T is the training set. The equality is true only for the whole training set.

3 Features design

All feature functions follows one common high-level specification: "all consecutive sequence of a number of specific letters corresponding with a pattern of the previous label and the current label". We generate the feature functions by varying the length of the letters(2,3,4), pattern of the previous and current label('BB','BI'...), and the position of label matching(only applied to the length of 3 and 4.)

One example of our low-level features could be "all consecutive sequence of 3 specific letters 'ing' with pattern 'BI' with the second position of label matching" or mathematically,

$$f_{'ing',BI',2}(\bar{x}, i, y_{i-1}, y_i) \quad = \quad I(x_{i-2}x_{i-1}x_i =' ing') \times I(y_{i-1}y_i =' BI')$$

Consider a similar feature function to the previous feature function, "all consecutive sequence of 3 specific letters 'ing' with pattern 'BI' with the first position of label matching", or mathematically,

$$f_{'ing',BI',1}(\bar{x}, i, y_{i-1}, y_i) = I(x_{i-1}x_ix_{i+1} =' ing') \times I(y_{i-1}y_i =' BI').$$

Notice the indexing of x, we effectively shift the matching letters by one index. Also, notice that all low-level feature function is binary as this will help facilitate our implementation.

The total number of possible features from this high-level specifications is, $9\times26^2+9\times26^3\times2+9\times26^4\times3 = 12,660,804$. We observe that there are patterns of consecutive letters that are not possible or relatively rare in the English languages, such as, 'gggg','lll'. Instead of enumerating all possible consecutive letters, we run through the data set and collect the sets of all possible 2,3,4-letter patterns. We organize these into sorted lists with sorting key as frequency of appearances. Table 1 shows the top ten entries from each list.

Performing this step reduce the number of possible feature functions to 823,365. In experiment 1, we want to test whether this reduction hurts the accuracy performance.

4 Implementation Concerns

4.1 Speed

Since we are dealing with a large feature space(823,365 feature functions), we need to take care of code arrangement. Our first implementation gives correct results; however, the running time of 1 epoch of Collins Perceptron with 50000 training examples and 4905 feature functions takes more than 20 hours to complete. The rest of this sections discuss the optimization techniques that we use to speed this process up. Our final implementation runs 1 epoch of Collins Perceptron with 50000 training examples and 823,365 feature functions in 10 minutes.

2-letters	3-letters	4-letters
in	ing	tion
es	ion	atio
er	ati	ting
ti	ate	ness
ng	tio	ions
te	ers	ling
ed	ess	ring
at	ter	ment
re	ent	ally
on	ies	ates

Table 1: Most appear lists
The top-10 most appeared consecutive sequence gathered from training set. The total size of the 2-letters, 3-letters, 4-letters sets are 545, 5756, and 26476 correspondingly.

Feature encoding

For each training example, only a small subset of the feature weights is modified. Instead of looping over every feature functions, we arrange them in such a way so that querying which feature functions are active in constant time.

We arrange the feature functions by letter count, matching position, and patterns of label, in this order. By using this ordering constraint, we effectively create a bijective mapping between feature index space and space of all the combinations of letter count, matching position, and pattern of labels. Furthermore, this mapping can be computed in constant time.

Decomposing feature functions We design the low-level feature functions as a product of two parts, i.e. $f_j(\bar{x}, i, y_{i-1}, y_i) = I_j(\bar{x}, i) \times J_j(y_{i-1}, y_i)$. This decomposition implies that if either part of the product is 0, computation of the other function is unnecessary. This property enables us to further preprocess the training data.

In many cases, we are only interested in letter patterns that is sub-string of the training data, x_i. We proceed to build a sparse matrix A, which each entry a_{iw} is a binary variable representing whether the w^{th}letter pattern is a sub-string of the training data x_i.

Removing function calls This optimization is only relevant Matlab. In many case, Matlab profiler reports close to 50% of the running time is from overhead of function calls. We take an effort not to place function calls in the most inner loop of the code.

4.2 Overflow issue

When we calculate the expectation of feature functions, we encounter the overflow problem. Recall

$$\begin{aligned} \alpha(k+1,v) &= \sum_u \alpha(k,u)[\exp g_{k+1}(u,v)] \\ E_{\bar{y}\sim p(\bar{y}|\bar{x};w)}[F_j(\bar{x},\bar{y})] &= \sum_{i=1}^{n} \frac{\alpha_{i-1}^T Q_{ij} \beta_i}{Z(\bar{x},w)}. \end{aligned}$$

For the recursive equation of α, the value of exponential of g table is multiplied every time. If the word is long(e.g. *"counterintelligence"*, $k = 19$) and the w_j is a big number(e.g., 100), the value of $\exp g_{k+1}(u,v)$ will be large. If we multiply the exponential for 19 times, some corresponding value in $\alpha(k+1,v)$ will exceed the maximum value ($\exp(+307)$) allowed in Matlab and will go to infinity. Since $Z(\bar{x},w) = \sum_v \alpha(n,v)$, $Z(\bar{x},w)$ will go to infinity as well. As a result, the value of expectation will be $\frac{\text{Inf}}{\text{Inf}}$. In Matlab, it well return to NAN, which is unexpected and should be avoided.

The intuitive way to solve the overflow would apply logarithm. Unfortunately, it is hard to apply logarithm into the computing because the addition at each recursive equation bothers the effective computing of logarithm. Instead of using logarithm, we decided to divide α, β, Q, and Z by a large number. Here, α, β, and Z are computed by recursive equation. Therefore, once each value is divided by the large number, the number of division will be as many as the recurrence time. Since the recurrence time of $Z(\bar{x},w)$ is $n-1$ and that of α is k, so the recurrence time of β would be $(n-k-2)$. It means that we need to divide into Q one time so that the number of division is the same at numerator and denominator. Here we define the divider m as the exponential of the mean value of the total g table:

$$m = \exp[\frac{1}{9 \times n} \sum_{i=1}^{n} \sum_{y_{i-1}} \sum_{y_i} g_i(y_{i-1}, y_i)] \tag{15}$$

We choose m as the exponential of mean value because we hope the maximum value in the exponential of the g table could decrease. At the same time, we do not want the other values become too small, which will cause the underflow problem. After the division, the expectation is

$$\begin{aligned} E'_{\bar{y}\sim p(\bar{y}|\bar{x};w)}[F_j(\bar{x},\bar{y})] &= \sum_{i=1}^{n} \frac{\frac{\alpha_{i-1}^T}{m^k} \frac{Q_{ij}}{m^1} \frac{\beta_i}{m^{n-k-2}}}{\frac{Z(\bar{x},w)}{m^{n-1}}} \\ &= \sum_{i=1}^{n} \frac{\alpha_{i-1}^T Q_{ij} \beta_i}{Z(\bar{x},w)} \\ &= E_{\bar{y}\sim p(\bar{y}|\bar{x};w)}[F_j(\bar{x},\bar{y})] \end{aligned} \tag{16}$$

which will not change the value of expectation.

5 Experiments

5.1 Variables Settings

We wish to learn the regularization strength and initial learning rate as settings for the stochastic gradient ascent algorithm in gradient following. We use Nelder-Mead method(Nelder, 1965) combined with a 5-fold cross-validation on the training data to find the optimal settings.

The Nelder-Mead method is a simplex-based nonlinear optimization technique for minimizing an objective function in a many-dimensional space. It begins with a set of points that are considered as the vertices of a working simplex S . At each iteration, we form a transformation of S which is determined by computing one or more test points and their function values, and by comparing these values with those at the vertices. The Nelder-Mead method typically requires only one or two function evaluations at each iteration. The process is terminated when the working simplex S becomes sufficiently small.

We use the error rate (word-level) as the objective function for the Nelder-Mead algorithm. This is consistent with using the conditional log-likelihood as the objective function. In both cases, we are searching over the settings space to optimize the probability distribution of the label sequences. Figure 1 shows the result of a Nelder-Mead run using 2-letters features. We limit the number iterations of the Nelder-Mead algorithm to 18 as Figure 1 shows that the objective function value doesn't change significantly after 8 iterations.

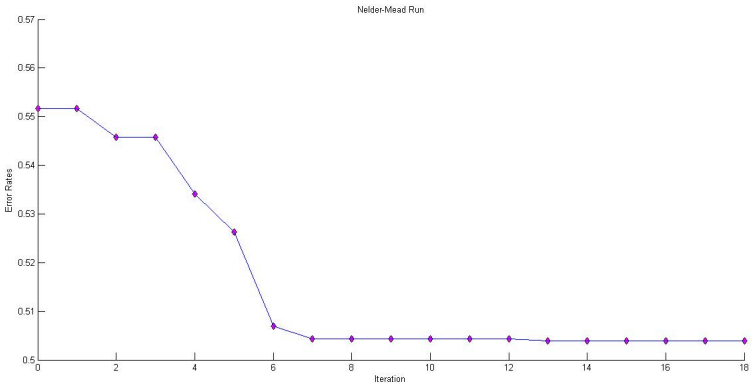

Figure 1: Nelder-Mead run for 2-letter features. The error rate does not change significantly after 8 iterations.

5.2 Stopping condition and parameter initialization

We use the mean change in the parameter values as a stopping condition. After each epoch, we check whether the L1 error distance between $\omega_e$and ω_{e+1},

$$error = \frac{1}{J}\sum_{j=1}^{J} |\omega_{e+1}^{j} - \omega_{e}^{j}| \tag{17}$$

is lower than the preset threshold,ϵ. We set the threshold, $\epsilon = .001$, as we want the average change in the parameter values to be within 3-digit precision.

We initialize the weights to be random numbers in the range of $(0, 1)$.

5.3 Experiment 1: Effects of using most appear lists

Design

In section 3, we suggest a reduction in the feature space as we use only letter pattern from the most appear lists instead of enumerating all possible letters patterns. In this experiment, we test whether this reduction hurts the word-level accuracy. We perform 10 different runs of Collins' perceptron on 50,000 training examples and test on 16001 testing examples to measure the word-level accuracy.

This experiment consists of 3 sub-experiments which aim to measure the effectiveness of the list sizes in 3 settings: 2+3-letters, 3-letters only, 4-letters only. For the 2+3-letters combination, we only vary the size of the 3-letters list as the size of the popular lists of 2-letter pattern is close to the entire enumeration.

Results

Figure 2 shows the performance of three sub-experiments with different feature type combinations: 2-letters with 3-letters, 3-letters only, and 4-letters. We can see log-like curves performance for every features combination, as the accuracy performance flats out near the end.

Discussion

The size of the popular lists does correlate positively with the word-level accuracy. There exists, however, a diminishing return as the size increases. This justifies using only the most appear lists to construct the feature set instead of all possible combinations of consecutive letters. Further, by limiting the possible set of consecutive letters, we significantly reduce the feature space dimension and, in turn, the possibility of overfitting.

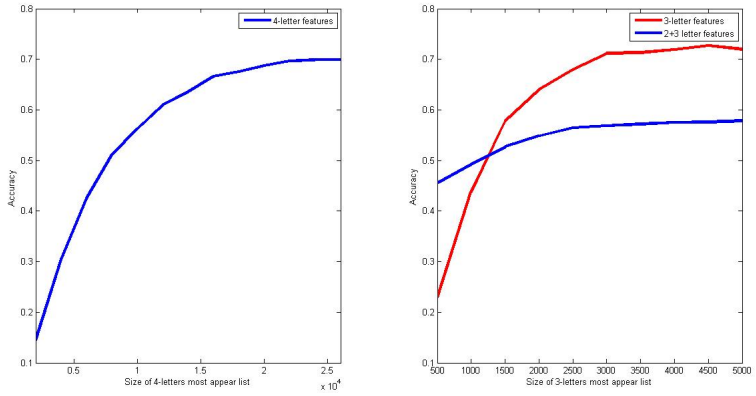

Figure 2: Performance of using different size of features
Left figure shows using different size of 4-letter features. Right figure shows using different size of 3-letter and 2+3 letter features. The accuracy does not increase significantly after we use certain amount of features.

5.4 Main experiment

Design

In this experiment, we want to test the effectiveness of the data set size, feature types, and different training methods. We create ten separate sub-experiments by crossing five different feature combinations (2-letters, 3-letters, 4-letters, 2+3-letters, 2+3+4-letters) with two training methods (Collins' perceptron and stochastic gradient following.) In each sub-experiments, we vary training set size from 5000 to 50000 examples with the increment of 5000 examples.

Results

We evaluate the classifiers on the same test set of 16001 examples. We measure word-level accuracy for each classifiers. Figure 3 and 4 shows the accuracy performance using different training set sizes, feature combinations, and learning methods.

Using Collins' perceptron, we get the accuracy of 54.22% for 2-letters features, 76.53% for 3-letters features, 69.54% for 4-letters features, 57.98% for 2+3-letters features, and 61.85% for 2+3+4-letters features. Using stochastic gradient following method, we get the accuracy of 48.92% for 2-letters features, 56.38% for 3-letters features, 63.84% for 4-letters features, 72.11% for 2+3-letters features, and 77.95% for 2+3+4-letters features

Collins' perceptron performs well with single features type(best at 76.5% with 3-letters only), while stochastic gradient following performs better when features are combined(best at 77.95% using all possible features.)

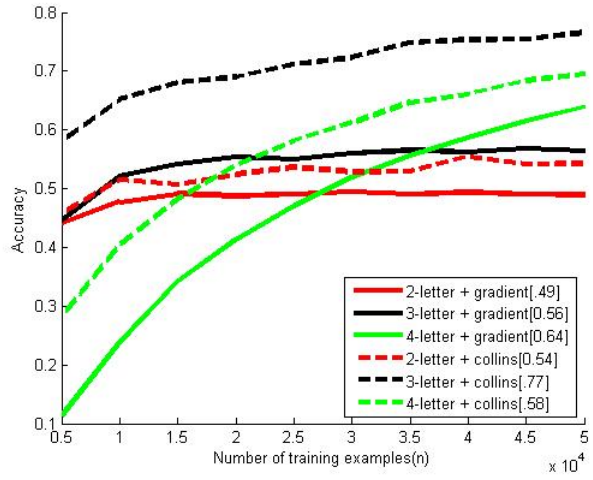

Figure 3: Classifiers performance on single feature types
Collins' perceptron(dotted line) performs better than gradient following method(solid line) with the combinations of different feature types.

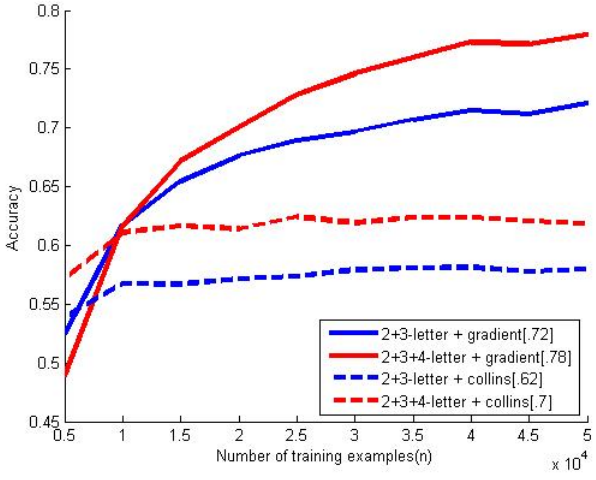

Figure 4: Performance of combined features
Gradient following method(solid line) perform better than Collins' perceptron(dotted line) with the combinations of different feature types. More data lead to significant improvement when using gradient following method, while there are not much improvement for Collins' perceptron.

Discussion

In most cases, more data lead to better accuracy, especially for stochastic gradient following method with combined features types. Figure 4 also shows the two accuracy curves for gradient training method are yet to converge. Thus, further improvement in accuracy is possible if more data is available.

Contextual information also helps improving accuracy. In the case of single features type, Figure 3 shows that 3-letter and 4-letter outperform 2-letter features as more textual information are captured in 3-letters and 4-letters than in the 2-letters case. Figure 4 further supports this point as 2+3+4-letters features outperform 2+3-letters features for any training methods.

Collins' perceptron performs better when only one feature type is used, while stochastic gradient following is better when features types are combined. We attribute this to the fact that in Collins' perceptron method, the average value of the feature function for a training example and all possible labels is approximated by the feature function value of the most possible label sequence. When different feature types are mixed, there are more contentions, not only between the possible labels within a feature type, but also across features types. In this case, the approximation is not sufficient.

6 Conclusion

In this paper, we have presented a linear-chain condition random field to the problem of English-word hyphenation. We implement Collins' perceptron and stochastic gradient following as training methods. Feature functions are generated from the most popular consecutive sequence, instead of the list of all possible sequence. We show that this reduction does not hurt the accuracy. We also describe speed improvement techniques and a solution to overflow problem. Finally, we discuss the effects of data set sizes, different feature combinations and different learning methods on the accuracy performance.

References

[1] Charles Elkan(2012), "Log-linear models and conditional random fields". February 7, 2012.

[2] Nelder, John A.; R. Mead (1965). "A simplex method for function minimization". Computer Journal 7: 308–313. doi:10.1093/comjnl/7.4.308.