Conditional Random Fields Applied to Punctuation Tagging

CSE250B, Winter 2014, Project 2

Prabhav Agrawal
UC San Diego
prabhav@ucsd.edu

Soham Shah
UC San Diego
srshah@ucsd.edu

March 14, 2014

---

1 Abstract

Nowadays, most smart-phones are moving towards a full touch interface, with a keypad popping up at the time of user interaction. Users want to quickly respond to email messages, create documents and send text messages, but screen real-estate limits the keypad to only alphanumeric characters, often requiring users to switch to a different keypad to add the punctuations. We aim to solve this problem by training a conditional random field (CRF) to predict the required punctuation tags for a given sentence in the English language.
We experiment with two different optimization techniques - Stochastic gradient ascent (SGA) and Collins perceptron (CP), on a set of feature function templates and analyze their results. The former provides word-level accuracy of 89.36%, and the latter 92.31%. The application requires quick prediction, thus we also focus on an efficient and further speed up through preprocessing, achieving average prediction time of 3ms.

2 Introduction

Our task is to train a conditional random field to add appropriate punctuations for a given English Sentence. We are targeting smartphone users who's keypad is too small to display both letters and punctuation marks. We want the user to just focus on entering text because punctuation marks will be inserted automatically.
We train the model on a dataset containing 57188 English sentences with correct punctuation marks. We observe that use of parts of speech (POS) tags helps training a model that captures hidden dependencies among various parts of speech and punctuations.
In a CRF, we use templates for feature functions, each of which is typically true only for a certain combination of POS tags and punctuation tags. We use a template to generate 2432 feature functions.
The Collins perceptron model gives word-level accuracy of 92.31% taking 22 minutes for training per epoch; and the SGA model gives an accuracy of 89.36% taking 4 minutes per epoch. Each epoch consisted of 45750 sentences. Collins perceptron, uses an approximation and thus achieves faster training. We also notice better accuracy by training the model using Collins perceptron. The prediction time over the test set was 1.75 minutes for 22,438 test sentences, which is about 3ms for a sentence.
Section 3 talks about CRFs and the optimization techniques used to train such a model. Section 4 then explains the design of our experiments followed by Section 5 which depicts our results, observations and lessons learnt through the course of the project.

3 Model analysis

3.1 Feature functions

In general, a feature function can be any real-valued function of the data space X to which the examples x belong and of the label space Y. Formally, feature function is any mapping $F_J : X \times Y- > R$. Let $\bar{x}$ represent a variable length input sentence and $\bar{y}$ represent the corresponding punctuation label such that y_i is the punctuation following the word x_i. A feature function $F_j(\bar{x}, \bar{y})$ represents the compatibility of a sentence $\bar{x}$ with a punctuation tag sequence $\bar{y}$. After training, each $F_j(\bar{x}, \bar{y})$ has a weight w_j associated with it which defines

the direction and magnitude of the compatibility, i.e. if $w_j > 0$ implies a higher value of the corresponding feature function $F_j(\bar{x}, \bar{y})$ takes you closer to true label and vice versa.
As the number of feature functions required are very large, instead of defining them manually, we use templates. $F_j(\bar{x}, \bar{y})$=$A_a(x).B_b(y)$ where a is a subscript that indexes a function over POS tags and b is a subscript that indexes a function over punctuation tags.

3.2 Conditional random fields

Our problem of finding punctuations is a structured prediction task, which is much harder than a binary classification task as we need to account for relations between neighbors, variable length sentences and the number of tag sequences is exponential.
The standard log-linear model is given by

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

where $Z(x, w)$ is given by

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

In order to specialize this model for the task of predicting the label $\bar{y}$ of an input sentence $\bar{x}$, assume that each feature function is actually a sum along the output label, for $i = 1$ to $i = n$ where n is the length of $\bar{y}$

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

Using this constraint, we can do the prediction task efficiently in polynomial time. Each lower level feature function f_j depends on the entire input $\bar{x}$, the position i within the input sentence $\bar{x}$, the previous punctuation tag y_{i-1} and the current punctuation tag y_i.

3.3 Inference algorithms

The prediction problem is defined as:

$$\hat{y} = argmax_{\bar{y}} p(\bar{y}|\bar{x}; w) = argmax_{\bar{y}} \sum_{j=1}^{J} w_j F_j(\bar{x}, \bar{y}) \tag{4}$$

Using the modified definition for F_j and dynamic programming, we can find out $argmax_{\bar{y}}$ using following two equations:

$$U(k, v) = \max_u [U(k-1, u) + g_k(u, v)] \tag{5}$$

$$\hat{y}_{k-1} = argmax_u [U(k-1, u) + g_k(u, \hat{y_k})] \tag{6}$$

In total, we can compute the optimal $\hat{y}$ for any $\bar{x}$ in $O(m^2nJ + m^2n)$ time

3.4 Optimization techniques

Training a CRF involves finding the vector w that maximizes probability $p(\bar{y}|\bar{x}; w)$. We compare two gradient based optimization techniques.

3.4.1 Stochastic gradient ascent

Gradient following techniques involves calculating the gradient at each iteration and then updating the parameters based on this value until convergence is achieved. In stochastic gradient ascent, we update the parameter vector w based on a single representative training example $\langle x, y \rangle$. As in the notes, the partial derivative of the log conditional likelihood (LCL) with respect to parameter w_j is:

$$\begin{aligned} \frac{\partial}{\partial w_j} logp(y|x; w) &= F_j(x, y) - \sum_{y'} F_j(x, y') p(y'|x; w) \\ &= F_j(x, y) - E_{y' \sim p(y'|x;w)} F_j(x, y') \end{aligned} \tag{7}$$

This implies that the gradient of LCL is the value of the feature function $F_j(x, y)$ for the example $\langle x, y \rangle$ minus the weighted average value of the feature function for x and all possible labels y'.
The update rule for the parameters w_j is given by

$$w_j := w_j + \lambda(F_j(x, y) - E_{y' \sim p(y'|x;w)} F_j(x, y')) \tag{8}$$

3.4.2 Collins perceptron

Collins perceptron approximates the expected value in equation 8 to the value of feature function of the most likely tag sequence $\hat{y}$. The resulting update expression for the vector w is :

$$w_j := w_j + \lambda F_j(x, y) - \lambda F_j(x, \hat{y}) \tag{9}$$

In words, this expression update causes a increase in w_j for features F_j whose value is higher for y than for $\hat{y}$. Thus increasing the probability of y compared to the probability of $\hat{y}$.

4 Design of experiments

We conduct our experiments using stochastic gradient ascent and Collins perceptron using a set of 57188 training sentences and their corresponding punctuations tags obtained from email messages. We then test our model on 22438 sentences in the test set. Each training sentence has a punctuation tag sequence of same length associated with it.

4.1 Preprocessing

4.1.1 POS Tagging

To exploit the dependencies between POS tags and punctuation tags we perform POS tagging on the input sentences. We use the Stanford POS Tagger to convert each sequence of words in a sentence into one of 36 different parts of speech. We remove the apostrophe characters in the data before performing POS tagging, as the Stanford POS Tagger broke a word with an apostrophe it into two separate words. Finally, we parse the output of the POS tagger and convert the tags into integers between 1-36. This was done purely as a performance optimization, refer to equation 11.

4.1.2 Cleaning the data

We find anomalies in the training data, such as comma, colon, hash and other special characters. We also find mismatches in the number of the POS Labels and the punctuation tags for a given sentence. These are introduced after POS tagging due to noise in the data. We discard these sentences. Finally, our training set is reduced to 57,188 sentences which we divide as 45,750 training examples and 11,438 validation examples.

4.2 Selecting the feature functions

To capture the hidden dependencies between POS tags and punctuation tags we define a large number of feature functions. CRF supports low level feature function as defined in equation 3. To generate these feature functions we use the following template:

$$F_j(\bar{x}, \bar{y}) = A_a(\bar{x}).B_{b1,b2}(y_{i-1}, y_i) \tag{10}$$

where $A_a(\bar{x}) = I(x_i = a)$ and $B_{b1,b2}(y_{i-1}, y_i) = I(y_{i-1} = b1).I(y_i = b2)$
We also add $START$ and $STOP$ tags in both the POS and punctuation tag set. There are 38 different POS labels and 8 different punctuation tags, this template defines $38 * 8 * 8 = 2432$ feature functions. These feature functions capture relationships between a POS tag and its previous and following punctuation tag. The definition of this feature function is such that for a position i in a given sentence only one combination of the triplet is non-zero. We implement these low level feature functions by using the following expression:

$$j = LabelNumber + (TagNumber1 - 1) * nPos + (TagNumber2 - 1) * nPos * nTag; \tag{11}$$

where j represents the feature function index, $nPos$ represent the number of possible POS tags, $nTag$ represents the number of punctuation tags, $LabelNumber$ is an index for POS tags and $TagNumber1$ and $TagNumber2$ represent the index associated with corresponding previous and next punctuation tags.
This is a constant time operation that can efficiently map a given triplet to a feature function. Additionally we optimize space using a sparse representation for feature function.

4.3 Stochastic gradient ascent

4.3.1 Computing the gradient efficiently

Equation 4 suggests that evaluating the gradient will require summing over exponential tag sequences. Instead we use a dynamic programming paradigm to evaluate forward vectors α and backward vectors β.

$$\begin{aligned}\alpha(k+1,v) &= \sum_{y_1,\ldots,y_k} exp[\sum_{i=1}^{k} g_i(y_{i-1},y_i) + g_{k+1}(y_k,v)] \\ &= \sum_u \alpha(k,u)[expg_{k+1}(u,v)]\end{aligned} \tag{12}$$

$$\beta(u,k) = \sum_v [expg_{k+1}(u,v)]\beta(v,k+1) \tag{13}$$

$$Z(\bar{x},w) = \alpha(n+1,STOP) = \beta(START,0) \tag{14}$$

If n is the length of the sentence $\bar{x}$ and m is the number of punctuation tags, then both α and β matrix each are of dimensions n x m. Computing each element of the matrix requires $O(m)$ time, requiring a total of $O(nm^2)$ time. As a sanity check, we verified if $\alpha(n+1,STOP) = \beta(START,0)$ to determine if the gradients have been computed correctly. Using these α and β vectors we can compute the expected value in equation 8 as:

$$E_{\bar{y}}[F_j(\bar{x},\bar{y})] = \sum_{i=1}^{n+1} \sum_{y_{i-1}} \sum_{y_i} f_j(y_{i-1},y_i,\bar{x},i) \frac{\alpha(i-1,y_{i-1})[expg_i(y_{i-1},y_i)]\beta(y_i,i)}{Z(\bar{x},w)} \tag{15}$$

The cost of computing $E_{\bar{y}}[F_j(\bar{x},\bar{y})]$ is $O(nm^2)$ as with each $\sum$ contributing n, m and m time respectively and the expression being evaluated in constant time.

4.3.2 Selecting the value of hyper-parameter

Stochastic gradient ascent requires a hyper-parameter λ that specifies the learning rate. We need to choose a learning rate that isn't too low, as this will increase the training time, at the same time we don't want it to be too large that the model doesn't converge and keeps bouncing around the maxima.

4.3.3 Training the model

The model consists of a weight associated with each feature function F_j. We initialize $w_j = \bar{0}$. For each sentence $\bar{x}$ we compute all $F_j(\bar{x},\bar{y})$ by scanning through the sentence and increasing the count of $F_j(\bar{x},\bar{y})$ corresponding to the triplet $\langle x_i, y_{i-1}, y_i\rangle$, where j is calculated using equation 11. This operation takes O(n) time where n is the length of $\bar{x}$. Next, we calculate $g_i(u,v)$ for all possible u,v $\in$ Punctuation Tags $\forall$ 1$\leq$i$\leq$n.

$$\begin{aligned}g_i(u,v) &= \sum_{j=1}^{J} w_j \cdot f_j(u,v,\bar{x},i) \\ &= w'_j \cdot f'_j(u,v,\bar{x},i)\end{aligned} \tag{16}$$

where j' can be computed from equation 11, putting $LabelNumber$ as i, $TagNumber1$ as u and $TagNumber2$ as v.
According to line 1 in equation 16, there are $O(n \cdot m^2)$ values to compute and each operation takes $O(J)$ time, totally taking $O(n \cdot m^2 \cdot J)$ time. Our representation of feature functions allows us to compute each value in constant time instead of $O(J)$ time. Next, we compute the α and β matrices and correspond $Z(\bar{x},w)$ using equation 12, 13 and 14 then calculate compute $E_{\bar{y}}[F_j(\bar{x},\bar{y})]$ using equation 15. Computing this takes $O(nm^2)$. Finally, we update the weights using equation 8. We repeat this process for each example of the training set to complete one epoch.

4.4 Collins perceptron

When all the weights are multiplied by the same nonzero constant, the label $\hat{y}$ that has the highest probability is unchanged. Collins perceptron method relies only on the identity of $\hat{y}$, and not on its probability, so the method will give the same behavior irrespective of the value of the learning rate λ. Thus we fix λ=1.

4.5 Checking for convergence and need for early stopping

At the end of every epoch, we compute the accuracy of the model on both the training set and the validation set. For the training set this accuracy goes on increasing but for the validation set this accuracy increases initially, reaches a maxima and then begins to decrease. To prevent overfitting of the training data, we stop training at the maxima and use this model for prediction on the test set. This is known as early stopping.

5 Results and discussion

5.1 Stochastic gradient ascent

To select the value of λ we consider a small subset of the 10000 training examples. We measure the norm of $\bar{w}$ at the end of every epoch and plot the difference between norm of $\bar{w}$ at the end of successive epochs. We use this for a test for convergence (difference in norm $\bar{w}$ ¡ 0.01), we want to select a λ that converges. Figure 1 depicts the behavior of training the model by performing stochastic gradient ascent using $\lambda = 10^{-3}$. As the number of epochs increases, the difference between the norm of $\bar{w}$ goes on decreasing and finally converges. We use $\lambda = 10^{-3}$ for training our model using stochastic gradient ascent.
We train the model using stochastic gradient ascent on 45,750 training examples for 15 epochs and check the accuracy of the model on both the training and validation set at the end of each epoch. Figure 2 depicts the change of accuracy with the no. of epochs. We can see the effect of overfitting after the 7^{th} epoch - the accuracy on training set increases but accuracy on validation set decreases. We use early stopping, to counter this effect and stop training after the 7th epoch, and use this model for prediction on test set.

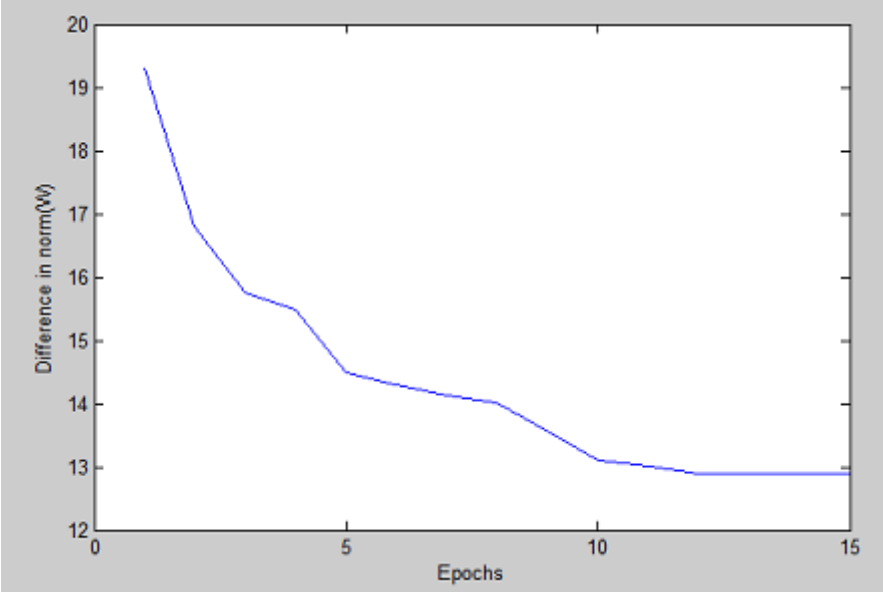

Figure 1: Difference in norm of $\bar{w}$ v/s no. of epochs for stochastic gradient ascent

Table 1: Accuracy for SGA

Dataset	Accuracy
Train	0.8993
Validation	0.8905
Test	0.8936

5.2 Collins perceptron

After setting $\lambda = 1$ and initializing weight vector $\bar{w} = \bar{0}$ we run Collins perceptron algorithm on the same training set and compute its accuracy on both the training and validation set at the end of each epoch. As with SGA, Figure 3 depicts the same overfitting phenomenon as Figure 2 and thus we stop training the model after the 8^{th} epoch, where the accuracy on the validation set is maximum.

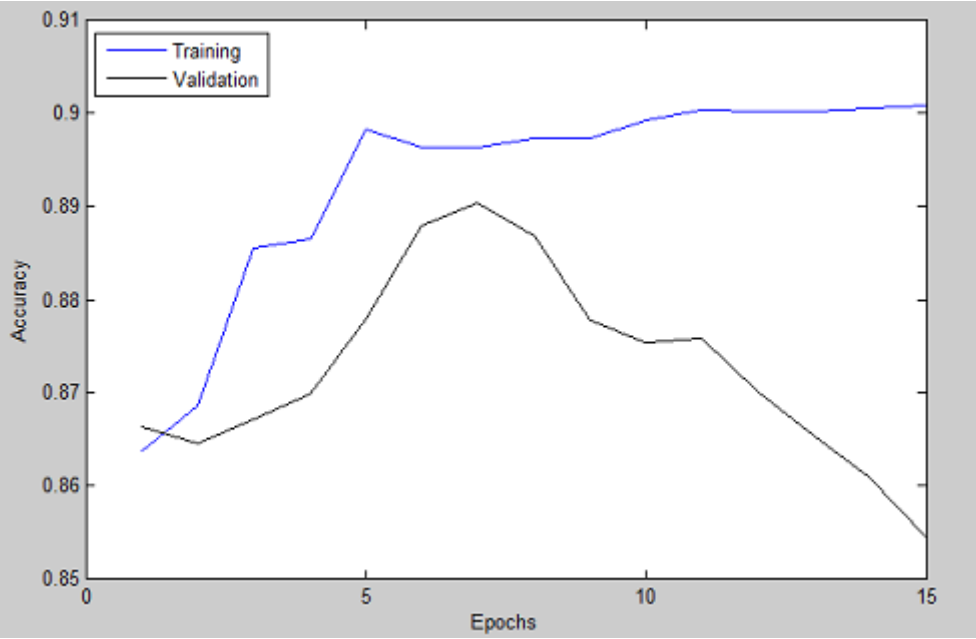

Figure 2: Accuracy v/s no. of epochs for stochastic gradient ascent with $\lambda = 10^{-3}$

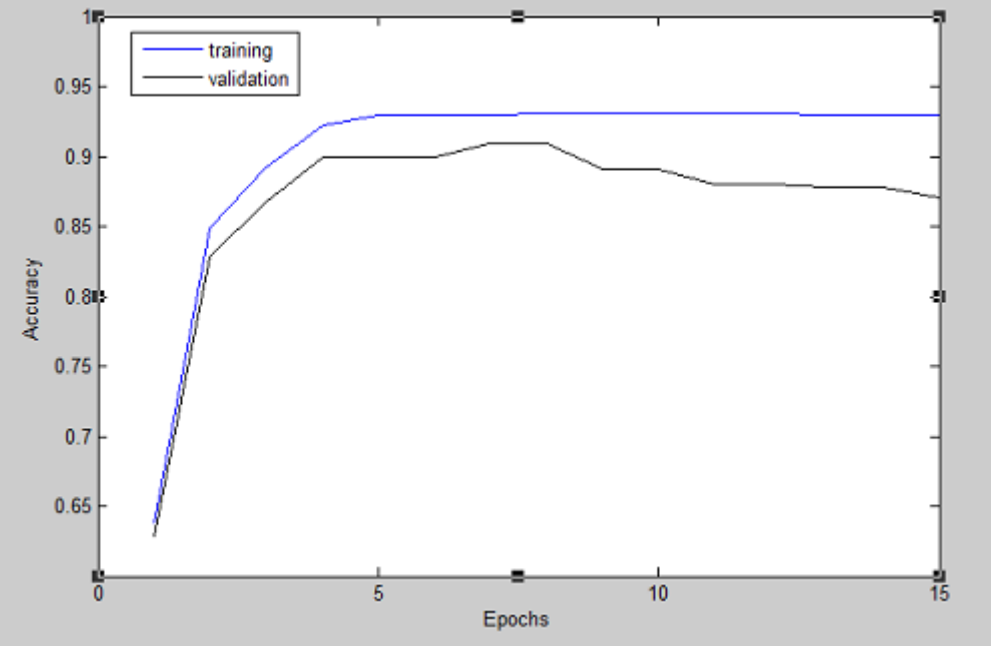

Figure 3: Accuracy v/s no. of epochs for Collins perceptron

5.3 Comparisons and lessons learnt

The Collins perceptron model gives word-level accuracy of 92.31% taking 4.5 minutes for training per epoch; and the SGA model gives an accuracy of 89.36% taking 22 minutes per epoch. Each epoch consisted of 45750 sentences. Collins perceptron, uses an approximation and thus achieves faster training. We also notice better accuracy by training the model using Collins perceptron. The prediction time over the test set was 1.75 minutes for 22,438 test sentences, which is about 3ms for a sentence.

We have learnt the importance of preprocessing. Parsing the input data and converting it into number gives us a significant performance boost.

The feature function template selection also allows us to exploit the sparsity in the feature function. Each triplet uniquely represents a feature function and thus we do not need to store it explicitly. Apart from space benefit, this gives us performance boost as in equation 16.

The results show that both models can make reasonable predictions on POS tags, which is due to the dependancies between English language parts of speech and punctuation tags.

Figure 1 and 2 both show us the benefits of early stopping or the ill effects of overfitting the training data. While training the data on a smaller subset of examples, we notice that the weight vector is changing at every epoch but the prediction on the validation set remains unchanged. This is because we have 2432 feature function and thus leading to overfitting. We need to train on more than 30000 sentences to see good results.

Table 2: Accuracy for Collins perceptron

Dataset	Accuracy
Train	0.9312
Validation	0.9088
Test	0.9231

References

[1] John D. Lafferty, Andrew McCallum, Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data, In Proceedings of the 18th International Conference on Machine Learning (ICML), 2001, pp. 282-289.

[2] Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing, pp. 1-8, 2002.

[3] Andrew McCallum. Efficiently inducing features of conditional random fields. In Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI-2003), 2003.