Conditional Random Fields for Punctuation Prediction Learning Algorithms, Project 2

Mohsen Malmir, Erfan Sayyari

February 13, 2014

1 Abstract

2 Introduction

In this paper, we provide the results and details of implementation of a Conditional Random Field (CRF) model for predicting English language text punctuations. CRFs are a variant of undirected graphical models that are well suited for predicting structured labels. We train our model by maximizing log conditional likelihood (LCL) of the training data. During our experiments, we found that the model could predict individual tags by more than 94% accuracy. However, as we performed more and more experiments, we discovered the over fitting to some prevalent tags in the training set. Using several experiments and methods that will be described later in this paper, we could overcome the over-fitting problem to some degrees.
We choose two techniques to train the proposed model: Collin's Perceptron and Contrastive Divergence. Both of these methods are approximations to the general gradient following method. We choose these two methods because of their two characteristics: simplicity and elegance. Both Collin's Perceptron and Contrastive Divergence provides simplified updates rule for the parameters, which are less expensive compared to the general gradient following for maximizing LCL. Despite simplicity, both methods have very clear and intuitive explanation behind them: they maximize the LCL of the data by throwing spurious samples at the model. However, as we explain our experiments in more details, we do not stick to the basic algorithms and tweak them in several ways to improve their performance.
One of the most important parts of the CRF model for predicting text punctuation is the choice of feature functions as it effects both the performance and accuracy of the model. The feature functions we develop here use Part-of-Speech (POS) tags for the input sentence. The main characteristic for

the proposed feature functions is their efficiency in computation time: training of our method only takes a few minutes on a single core machine. We show that using these feature functions, the model can predict the punctuation tags with high accuracy.
For the given training and test data, danger of over-fitting is high because of the imbalanced distribution of punctuation tags. We deal with this problem with different techniques. We show how bounding the weight parameters and early stopping can improve the accuracy for the Collin's Perceptron. By introducing random sampling and guided sampling, we achieve similar improvements in Contrastive Divergence. Then we introduce the Turn-taking train procedure: for each tag we train a different predictor and then combine them using .... We show this last technique is much more effective and achieves the highest accuracy for punctuation prediction for the given data set.

3 Design and Analysis of Algorithm

3.1 The general log-linear model

Generally log-linear model is an extension of logistic regression. Conditional random Fields (CRFs) are a special case of log-linear models as well. Consider x as an example that could be drawn from set X, and if we consider y as a label which could be chosen from set of labels Y. In log-linear model we write the conditional probability $p(y|x;w)$ as:

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

where $Z(x,y)$ is a normalization factor that is equal to:

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

In above equations $F_j(x,y)$ is called a feature function. If $w_j > 0$ then observing a positive value for a feature function makes label y more probable to be label of x (other things fixed). Conversely, if $w_j < 0$, observing a positive value for a feature function makes label y less probable to be the label of x.

If we have weights w_j and feature functions $F_j(x,y)$s, in order to assign a label to a test example x, we have to solve an argmax problem. Mathematically we could write it as:

$$\hat{y} = arg \max_y p(y|x;w) = arg \max_y \sum_{j=1}^{J} w_j F_j(x,y). \tag{3}$$

Above formula is called softmax function which is a differentiable and convex function.

3.2 Feature functions

In general, a feature function can be a mapping from data space X and label space Y to real-valued numbers $\mathbb{R}$, $F_j : X \times Y \rightarrow \mathbb{R}$. In our project, F_j is a mapping from data space and label space to Boolean space, $F_j : X \times Y \rightarrow \{0, 1\}$.

Usually, we define classes of feature functions using a template. We define $F_j(x, y)$ as the sum of some local feature functions, f_j. We could write:

$$F_j(x, y) = \sum_{i=1}^{n+1} f_j(y_{i-1}, y_i, x_{i-1}, x_i). \tag{4}$$

where in our project x_i and x_{i-1} are words i and $i-1$ of the sentence and f_j is an indication function. Instead of using each word in above equation we use Part-of-Speech (POS) tags of each word, so we could rewrite above equation as:

$$F_j(x, y) = \sum_{i=1}^{n+1} f_j(y_{i-1}, y_i, POS(x_{i-1}), POS(x_i)). \tag{5}$$

POS tags are corresponding part of speech of each word in a sentence, for example adjective, noun, verb and etc. In the next part we investigate them more.

Above model has some problem, for example in practice many effective features depends on different positions of the sentence not consecutive ones. This model just considers POS tags of just two words that are consecutive. In addition, in above equation we do not consider different length of different sentence. We design feature functions in this way to avoid higher complexities that makes our problem unsolvable.

3.2.1 Part of Speech tagging

In linguistics, part-of-speech tagging is the process of assigning a part of speech to a word. There are many packages in python which do POS tagging. In this project we use *topia.termextract 1.1.0* **??** to do this preprocessing task. This package will assign different tags to different words of the sentence. The list of tags and their definition are represented in Table 1. Beside tags that are represented in Table 1, there are some other tags that if *topia* does not know what it is consider it as a separate POS tag. In our project there are some of these unknown tags, for example \$, " ", etc.

In order to extract POS tags, first we have to tokenize the sentence. Due to unknown tags and words in the dataset, we tokenize the sentence based on space first. Then we use *topia.termextract 1.1.0* to assign tags to words. It does not take much time, and this package is fairly fast.

The number of tags in Table 1, are large, which makes our algorithm so slow. In order to reduce them we use closed class words. We have to merge some of them into each other without any significant drawback on our problem. The set of simplified POS tags are represented in Table 2. We merge different types of nouns, adjectives, words start with wh (other than adverbs form because they are important for question mark), verbs, and pronouns into one group. Furthermore, we merge other unknown labels as Noise as well. In addition we merge determiner and predeterminer into adjectives too. (they related to nouns so we merge them into adjectives)

3.3 Conditional random fields

A linear conditional random eld is a way to apply a log-linear model to the task where we have different sequence of words with different length. The standard log-linear model is:

$$\hat{y} = arg \max_{y} p(y|x; w) \tag{6}$$

for each training example x. Since the number of label tag sequences are exponential this task is very complex. Restricting the problem to just two adjacent POS tags and two consecutive labels makes the problem much less easier to solve. Moreover, in this task we maximize log conditional likelihood (LCL) with regularization instead of original problem.

3.4 Inference algorithms for linear-chain CRFs

In order to solve the argmax problem, we can ignore the regularization factor since it is constant. Mathematically, we could write the problem as:

$$\hat{y} = arg \max_{\bar{y}} p(\bar{y}|\bar{x}; w) = arg \max_{\bar{y}} \sum_{j=1}^{J} w_j F_j(\bar{x}, \bar{y}). \tag{7}$$

Using $F_j = \sum_{i=1}^{n+1} f_j(y_i, y_{i-1}, POS(x_i), POS(x_{i-1})$ we could rewrite the objective function as:

$$\hat{y} = arg \max_{\bar{y}} \sum_{j=1}^{J} w_j \sum_{i=1}^{n+1} w_j f_j(y_i, y_{i-1}, POS(x_i), POS(x_{i-1}) \tag{8}$$

$$= arg \max_{\bar{y}} \sum_{i=1}^{n+1} g_i(y_{i-1}, y_i). \tag{9}$$

where we define:

$$g_i(y_{i-1}, y_i) = \sum_{j=1}^{J} w_j f_j(y_{i-1}, y_i). \tag{10}$$

	Word tag	Meaning
1	CC	Coordinating conjunction
2	CD	Cardinal number
3	DT	Determiner
4	EX	Existential there
5	FW	Foreign word
6	IN	Preposition or subordinating conjunction
7	JJ	Adjective
8	JJR	Adjective, comparative
9	JJS	Adjective, superlative
10	LS	List item marker
11	MD	Modal
12	NN	Noun, singular or mass
13	NNS	Noun, plural
14	NNP	Proper noun, singular
15	NNPS	Proper noun, plural
16	PDT	Predeterminer
17	POS	Possessive ending
18	PRP	Personal pronoun
19	PRP$	Possessive pronoun
20	RB	Adverb
21	RBR	Adverb, comparative
22	RBS	Adverb, superlative
23	RP	Particle
24	SYM	Symbol
25	TO	to
26	UH	Interjection
27	VB	Verb, base form
28	VBD	Verb, past tense
30	VBG	Verb, gerund or present participle
31	VBN	Verb, past participle
32	VBP	Verb, non-3rd person singular present
33	VBZ	Verb, 3rd person singular present
34	WDT	Wh-determiner
35	WP	Wh-pronoun
36	WP$	Possessive wh-pronoun
37	WRB	Wh-adverb

Table 1: topia.termextract 1.1.0 POS tags and their meanings

	Simplified Word tag	Members	Meaning
1	ADJ	JJ, JJS, JJR, PDT, DT	Adjective
2	PRP	PRP, PRP$	Pronoun
3	NN	NN, NNS, NNP, NNPS, FW	Nouns
4	VB	VBD, VBG, VBP, VBN, VBZ, VB	Verbs
5	WH	WP, WP$, WDT	Wh-words
6	RB	RB, RBR, RBS	Adverb
7	Noise	#, ' ', $, (,)	Unknown
8	POS	POS	Possessive ending
9	WRB	WRB	Wh-adverb
10	CC	CC	Coordinating conjunction
11	CD	CD	Cardinal number
12	IN	IN	Preposition or subordinating conjunction
13	EX	EX	Existential there
14	MD	MD	Modal
15	SYM	SYM	Symbol
16	UH	UH	Interjection

Table 2: Closed class used as POS tags instead of original POS tags

Supposing that we have $\bar{x}$, w, and i computing g_i needs $O(m^2 J)$ time, if we assume that we have m different labels including $STOP$ and $START$ which show start and stop of the sentence. $U(k, v)$ is defined as the maximum of sum over g_is from $i = 1$ to k such that the label of k^{th} tag is v. This is equal to:

$$U(k, v) = \max_{y_1, \cdots y_{k-1}} \sum_{i=1}^{k-1} g_i(y_{i-1}, y_i) + g_k(y_{k-1}, v). \tag{11}$$

This equation could be rewrite as:

$$U(k, v) = \max_{y_{k-1}} \max_{y_1, \cdots y_{k-2}} \sum_{i=1}^{k-2} g_i(y_{i-1}, y_i) + g_k(y_{k-2}, y_{k-1}) + g_k(y_{k-1}, v) \tag{12}$$

$$= \max_{u} [U(k-1, u) + g_k(u, v)]. \tag{13}$$

where y_{k-1} is equal to u. We need to compute $\max U(k-1, u)$ for m different u, so computing $U(k, v)$ requires $O(m)$ time, if v is fixed. Therefore, we can compute $U(k, v)$ for different vs in $O(m^2)$ time. Lastly, we can find best label for the entire sentence by:

$$\hat{y}_n = arg \max_{v} U(n, v). \tag{14}$$

In the above analysis we assume that g_is are known. Considering these g_is for a sentence $\bar{x}$ of length n, we can compute optimal $\hat{y}$ in $O(m^2 nJ + m^2 n)$.

3.5 Gradients for log-linear models

In order to train a log-linear model we have to find w_js that maximize the objective function which is the conditional probability of labels given training examples. To solve maximization problem we calculate derivative of objective function with respect to parameters and set them zero. Since our problem is convex, this gives us the best answer. It is useful to note that instead of solving the main problem we solve LCL. Consequently, we can write:

$$\frac{\partial}{\partial w_j} logp(y|x;w) = F_j(x,y) - \frac{\partial}{\partial w_j} logZ(x,w) \tag{15}$$

$$= F_j(x,y) - \frac{\partial}{\partial w_j} Z(x,w). \tag{16}$$

where y is the known true label of the training example x.

$$\frac{\partial}{\partial w_j} Z(x,w) = \frac{\partial}{\partial w_j} \sum_{y'} [exp \sum_{j'} w_{j'} F_{j'}(x,y')] \tag{17}$$

where y' is different candidate labels. So we have:

$$\frac{\partial}{\partial w_j} Z(x,w) = \sum_{y'} [exp \sum_{j'} w_{j'} F_{j'}(x,y')] F_j(x,y') \tag{18}$$

And lastly we have:

$$\frac{\partial}{\partial w_j} logp(y|x;w) = F_j(x,y) - \frac{1}{Z(x,w)} \sum_{y'} F_j(x,y') [exp \sum_{j'} w_{j'} F_{j'}(x,y')] \tag{19}$$

$$= F_j(x,y) - \sum_{y'} F_j(x,y') [\frac{exp \sum_{j'} w_{j'} F_{j'}(x,y')}{Z(x,w)}]. \tag{20}$$

Considering that $[\frac{exp \sum_{j'} w_{j'} F_{j'}(x,y')}{Z(x,w)}] = p(y'|x;w)$ we can simplify above equation to:

$$\frac{\partial}{\partial w_j} logp(y|x;w) = F_j(x,y) - \sum_{y'} F_j(x,y') p(y'|x;w) \tag{21}$$

$$= F_j(x,y) - E_{y' \ p(y'|x;w)}[F_j(x,y')]. \tag{22}$$

Above equation is just for an example of training set, and for the entire training set, T, we could rewrite it as:

$$\sum_{<x,y> \in T} F_j(x,y) = \sum_{<x,.> \in T} E_{y \ p(y|x;w)}[F_j(x,y)]. \tag{23}$$

3.6 Stochastic gradient ascent

Usually, we calculate the best w_j by gradient ascent method, which is equal to:

$$w_j \leftarrow w_j + \sum_{<x,y>\in T} F_j(x,y) = \sum_{<x,.>\in T} E_{y\ p(y|x;w)}[F_j(x,y)] \tag{24}$$

where λ is learning factor. Updating w_j using above equation is very time consuming so in practice we use stochastic gradient methods considering just one random sample at each time as follows:

$$w_j \leftarrow w_j + \lambda(F_j(x,y) - E_{y'\ p(y'|x;w)}[F_j(x,y')]) \tag{25}$$

In this way, the total time complexity of the updates for all j, for a single training x and its label y, is $O(Jm^2n)$.

3.7 The Collins perceptron

Suppose that $E_{y'\ p(y'|x;w)}[F_j(x,y')] = F_j(x,\hat{y})$ where $\hat{y} = arg\max_y p(y|x;w)$, so we can rewrite the stochastic gradient update rule as:

$$w_j \leftarrow w_j + \lambda F_j(x,y) - \lambda F_j(x,\hat{y}). \tag{26}$$

This method is called Collins perceptron. It is useful to mention that multiplying all w_js by the same factor does not affect on finding label $\hat{y}$ which has highest probability. So we could simply set λ to be equal to 1.

3.7.1 Gibbs sampling

Instead of assigning $E_{y'\ p(y'|x;w)}[F_j(x,y')] = F_j(x,\hat{y})$ we can compute $E_{y'\ p(y'|x;w)}[F_j(x,y')]$ by sampling y values from distribution $p(y|x;w)$. To do this we can use Gibbs sampling. if we write y as a set with its sub-tags as $y = \{y_1, ...y_n\}$, and if we suppose that we have all the conditional distributions:

$$p(Y_i = v|x, y_1, ..., y_{i-1}, y_{i+1}, ...y_n; w) \tag{27}$$

then we can draw a stream of samples as:
1) select an arbitrary initial guess $\{y1, ..., y_n\}$
2) Draw a new value for y_1 from distribution $p(Y_1|x, y_2, ..., y_n; w)$;
Draw a new value for y_1 from distribution $p(Y_1|x, y_2, ..., y_n; w)$;
Draw a new value for y_2 from distribution $p(Y_2|x, y_1, y_3, ..., y_n; w)$;

continue above procedure till finding y_n from distribution $p(Y_n|x, y_1, y_2, ..., y_{n-1}; w)$; 3)Repeat 2. It could be proved that repeating step 2 infinitely, we converge to the distribution p(y—x;w).

We can calculate distribution $p(v|x, y_{-i}; w)$ by:

$$p(v|x, y_{-i}; w) = \frac{[expg_i(y_{i-1}, v)][expg_{i+1}(v, y_{i+1})]}{\sum_{v'}[expg_i(y_{i-1}, v')][expg_{i+1}(v', y_{i+1})]} \tag{28}$$

Consider doing one round of Gibbs sampling, requires $O(mn)$ time.

3.8 Contrastive divergence

Contrastive divergence method tries to find a single y^* similar to true label y, with higher probability. We implement contrastive divergence by Gibbs sampling which is equal to finding a $y^* =< y_1^*, y_2^*, ..., y_n^* >$ by usually one round of Gibbs sampling, starting from true label as step one.

4 Design of Experiments

To test the proposed model, we perform several experiments on the given dataset, which is an English language punctuation dataset that consists of 70115 training and 28027 test sentences. We use only training data to fit the model, one third for validation and two thirds for training. After training is complete, the test data is used to assess prediction and generalization of the model. Each sentence in the training data consists of a set of words without any punctuation symbols. For multiple sentences, there was a few form of syntax complications such as use of '\$' in place of the word 'money', or words that were split to two parts by space *****. We ignored all such cases, as they were few sentences and finding a complete list of them required going through the entire dataset which was very time consuming.

4.1 Feature extraction

Our features are of the form $F_j(x, y) = \sum_{i=1}^{n+1} f_j(y_{i-1}, y_i, POS(x_{i-1}), POS(x_i))$, where y_i and y_{i-1} denote labels in positions i and $i-1$ of the sentence and $POS(x_i)$ and $POS(x_{i-1})$ denotes POS tag of sentence in positions i and $i-1$. So each small feature function f_j depends on two different POS tag and two different label. In other words we have J different feature functions that $J = m^2k^2$ where m is equal to number of labels and k is equal to number of POS tags. Since our feature functions are indication functions, they are just non-zero for especial POS tags. It is useful to mention that actual number of feature functions are far smaller than J since many of combinations of POS tags do not happen in our training set. In order to implement it more efficiently, we assign each combination of POS tags an index, and we assign each label an index too. In overall we assign each combination of $\{y_{i-1}, y_i, POS(x_{i-1}), POS(x_i)\}$ an index which is equal to the index for the weight w_j associated with them too. Consequently, if we want to compute $F_j(x, y)$ we count number of observations of the combination $\{y_1, y_2, POS(x_1), POS(x_2)\}$ along the sentence and put it inside the memory associated with their index. To update w_j we use this F_j and update it.

In order to calculate a unique index for each combination we use the following formula:

$$idx = idx_{POS_2} * (numberoflabels)^2(numberofPOStags) \tag{29}$$
$$+idx_{POS_1} * (numberoflabels)^2 + idx_{label_2} * (numberoflabels) + idx_{label_1}. \tag{30}$$

The other important thing about our feature functions is that, we increase the length of sentences to become the same, equal to d. In shorter sentence after the $STOP$ we have -1, for $STOP$ we assign zero and to $START$ we assign $d + 1$ (d is the maximum length of sentence in training data).

4.2 Initialization

For Collin's Perceptron, as pointed out in the lecture, the learning rate can be fixed to $\lambda = 1$. This is because scaling scaling the weight vector w does not affect the predicted label by the model. Using different learning rates leads to different weight vectors that are scaled versions of each other. However, one should note that this is correct only if the model converges to the global optimum ******.
For both experiments on Collin's Perceptron and Contrastive Divergence, we fix $\lambda \leftarrow 1$.
For initializing the weight vector w, we use small random Normal numbers from $\mathcal{N}(0., 1e-5)$. This is because as we describe later, limiting the weight vector entries w_j to be in a small range $[-a, a]$ helps to prevent over-fitting. Because both a and $-a$ represents the extreme learned situation, a good option for initial weights is a random vector that is not biased for specific feature functions, that is w_j better be close to 0 for all j.

4.3 Preprocessing

4.4 Performance Measure

We report the performance of the models in two different ways. First, we report the prediction of individual target tags, That is, how accurate the model can predict the individual punctuations tags in corpus sentences. We report the accuracy of prediction for the entire tags as wells as individual target tags. The benefit of looking into individual punctuation tag prediction is that it reveals if over-fitting has happened due to imbalanced distribution of tags in the training data. Next, we report the accuracy of model for predicting sentence punctuation. The accuracy for this case should be lower because each sentence punctuation is composed of multiple tags.

5 Results of Experiments

5.1 Collins perceptron

For training Collin's Perceptron, we divide the training data into two thirds for train and one third for validation. The initial value for the weight vector entries w_j is chosen to be small random numbers from $\mathcal{N}(0, 1e-5)$. We use early stopping as a criteria for terminating the training procedure. Because Collin's Perceptron has no hyper parameters, there is a chance of over fitting to the training data. In the train procedure, we divide the data into train and validation and start training only with train data. After each epoch, we measure the accuracy for predicting individual punctuation tags on the validation data. If the accuracy has decreased, we stoop the training and roll back the weights to previous values.

5.2 Gibbs sampling

6 Findings and Lessons Learned

6.1 Numerical Issues and Preprocessing

6.2 Overfitting

6.3 Model Selection

6.4 Future works

7 References

url = "https://pypi.python.org/pypi/topia.termextract/"