Conditional Random Fields for Punctuation Prediction Learning Algorithms, Project 2

Mohsen Malmir, Erfan Sayyari

February 13, 2014

1 Abstract

We describe the details of implementing a Conditional Random Fields model for prediction punctuation tags for English language text . Gibbs sampling and Collin's Perceptron methods are used to train the model by maximizing the log-conditional likelihood of the data. We propose a set of feature functions that are based on part of speech tags of the input sentences. We show that using with feature functions, training of the model is very fast and memory efficient. The proposed model can predict the individual punctuation tags with 91% accuracy. We also look closely into the problem of over fitting, which occurs because of the imbalance of target tag frequency in the training data. A model averaging technique is proposed that can alleviate this problem.

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 of 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:

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

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 $\mathbb{R}$, $F_j : X \times Y \to \mathbb{R}$. In our project, F_j is a mapping from data space and label space to Boolean space, $F_j : X \times Y \to \{0, 1\}$.

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:

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

	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

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:

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

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:

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

This equation could be rewrite as:

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

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

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 $O(m^2nJ + m^2n)$.

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 0, 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 training 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 sets 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 stop the training and roll back the weights to previous values.
For the first experiment, we train Collin's perceptron with early stopping. Figure 1 shows that the training stops after 4 epochs. However, after the first epoch, the model is almost converged and the accuracy changes a small amount after that. The accuracy for predicting individual tags and sentences for validation and test sets are shown in Table 3.

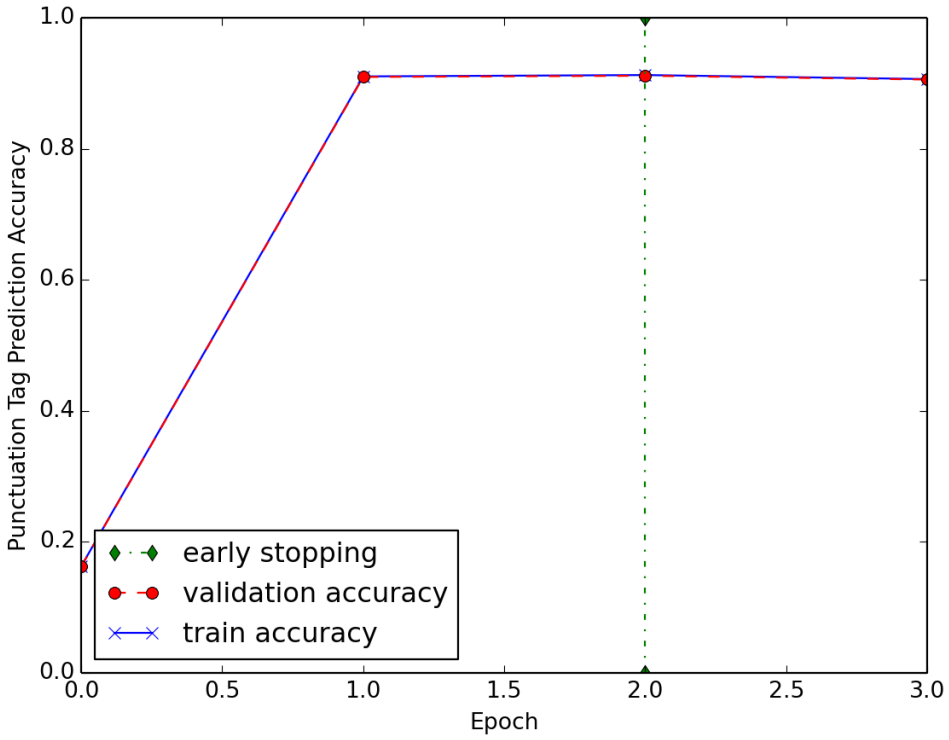

Figure 1: Early stopping for Collin's Perceptron.

Table 3: Collin's Perceptron performance on the given data set

Measure	**Validation set accuracy**	**Test set accuracy**
Individual tag prediction	0.911	0.901
Sentence level prediction	0.389	0.371

In order to further assess the correctness of the model, we look into the accuracy for predicting individual tags, that is, how much the model is accurate in predicting different punctuation tags. Table 4 shows the accuracy for predicting individual tags. Figure 2 shows the confusion matrix for the same prediction problem. A quick look at table 4 and figure 2 reveals some properties of the Collin's Perceptron model. Here, the model has learned only the most frequent punctuation tags in sentences, that is SPACE and PERIOD. For each test sentence the model predicts a simple punctuation tag sequence which has SPACE or PERIOD in most positions (look at SPACE and PERIOD columns in

the test confusion matrix in figure 2). This behavior is somehow natural, because in training phase, the model sees sequences of tags with SPACE or PERIOD more often, and thus the corresponding weights to these weights are more frequently updated. Table 5 shows 20 largest entries in the weight vector and their corresponding feature function.

In order to investigate whether these weights are meaningful and sensible or not, we could go through the whole training set and training labels and look over the frequency of their pattern $(y_i, y_{i-1}, POS(x_i), POS(x_{i-1})$ where i is from 1 to n) that happens in the context. If the frequency of that pattern is large among the number of observance from $POS(x_i)$ and $POS(x_{i-1})$, we could conclude that w_j should be large. It is true since if we observe that pattern of POSs, we could say that with high probability their labels are from those labels in pattern. But, we could look over our training labels and the sentence associated with them and check above condition intuitively. For example, for the largest weigh, $\{NN, VB, COMMA, SPACE\}$ we search over the text and find that there are many examples associated with this, for example: "On another note do you have any idea how Patti is holding up" we have the combination of "note" and "do" at first part of sentence that their POS tags are noun (NN) and verb (VB) truly. Or for example for the combination of $\{NN, PRP, COMMA, SPACE\}$, we have another sentence: "If we have one year of data we can tell which will be cheaper" and at middle part we have the combination of "data" and "we", which are noun (NN) and pronoun (PRP) that its weight is big again. Another example could be for the big weight for the pattern $\{ADJ, PRP, COMMA, SPACE\}$. We have a sentence: "Sorry I didn't attach the form", and at first words we have "Sorry" and "I" which are adjective (ADJ) and pronoun (PRP) consecutively. Another meaningful weigh is the pattern $\{NN, IN, COMMA, SPACE\}$. We have an example: "Thank you for the offer but I am not doing the ride this year". In the middle we have "offer" and "but" which are noun and conjunction the weight according to this pattern is large, and based on grammatical point of view, we usually have "COMMA" before conjunctions.

Table 4: Accuracy of predicting different tags by Collin's Perceptron

Method	**Validation set**		**Test set**	
	Accuracy	Frequency	Accuracy	Frequency
EXCLAMATION_POINT	0.0	2567	0.0	1066
SPACE	0.947	580077	0.947	235536
QUESTION_MARK	0.145	10904	0.1444	4792
PERIOD	0.849	57259	0.847	22492
COLON	0.009	1009	0.003	1174
COMMA	0.248	28555	0.262	10665

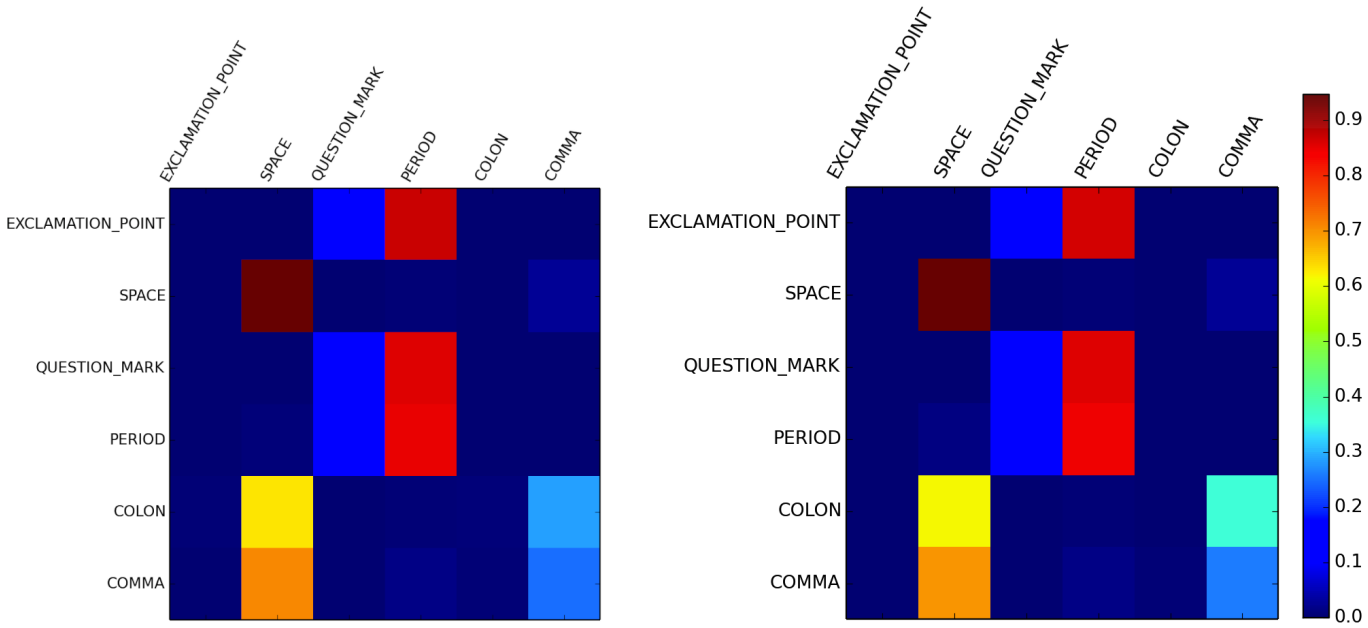

Figure 2: Confusion matrix for predicting individual tags using Collin's Perceptron on Left: validation, and Right: test sets.

Table 5: 20 Largest entries of the weight vector for Collin's Perceptron after learning. For each weight entry, its magnitude and corresponding feature function is listed.

Number	**Magnitue**	X_{i-1}	X_i	Y_{i-1}	Y_i
1	3964.99	NN	VB	COMMA	SPACE
2	3927.99	NN	VB	SPACE	SPACE
3	3308.99	ADJ	NN	SPACE	COMMA
4	2875.99	IN	NN	SPACE	COMMA
5	2460.99	PRP	NN	SPACE	COMMA
6	2099.00	NN	IN	COMMA	SPACE
7	2067.00	NN	IN	SPACE	SPACE
8	1800.99	NN	NN	COMMA	SPACE
9	1771.99	PRP	VB	SPACE	COMMA
10	1759.00	NN	NN	SPACE	SPACE
11	1508.99	NN	WRB	COMMA	SPACE
12	1463.00	NN	WRB	SPACE	SPACE
13	1451.00	NN	PRP	COMMA	SPACE
15	1409.00	NN	PRP	SPACE	SPACE
16	1209.99	ADJ	PRP	COMMA	SPACE
17	1198.99	VB	ADJ	SPACE	COMMA
18	1077.00	ADJ	PRP	SPACE	SPACE
19	1056.00	NN	ADJ	COMMA	SPACE
20	1018.99	NN	ADJ	SPACE	SPACE
21	997.000	IN	PRP	SPACE	COMMA

5.2 Contrastive Divergence

For Contrastive Divergence, the training setup is the same as mentioned in previous section. The training data is divided into training and validation sets, and early stopping is applied to terminate the training. The initial value for the weight vector entries are chosen randomly from $\mathcal{N}(0, 1e-5)$.

5.2.1 Training with Bounded Weights

As we saw in Collin's Perceptron, the weight vector entries grow unbounded for tags that are more frequent in the training data. This unbounded growth can cause problems during prediction, because the large weights tend to bias the decision in favor of more frequent tags (e.g. SPACE or PERIOD). One way to deal with this situation is to use a separate learning rate for each entry in the weight vector. Then if a w_j has been updated frequently, we can decrease its learning rate. A similar approach is to force the w_js to be in a specified range $[-a, a]$. For each update, one then should check that

w_j doesn't exceed this bound. We adapt this technique in training of Contrastive Divergence model, with hope that this reduces over-fitting to specific punctuation tags.
Figure 3 shows that early stopping terminates the training for Contrastive Divergence after two epochs. Tables 6 and 7 show the performance for this model. These results are very interesting as we compare it row by row to table 4. We see that the tag EXCLAMATION_POINT, QUESTION_MARK and COLON are learned more and accuracy of their prediction has increased in Contrastive Divergence compared to Collin's Perceptron. Specifically this is interesting because these are the lowest frequency symbols in the training dataset. However, this learning occurred at the expense of *unlearning* COMMA and PERIOD, which lead to decreased overall prediction accuracy both on sentence level and individual tag level prediction. Table 8 displays some of the weights learned by Contrastive Divergence.

As described for Collins perceptron, we could investigate meaningfulness of weights by searching for different patterns of labels and POS tags and their frequency and compare them intuitively with our weights. Intuitively, we could say that the more frequent a pattern is, the more probable it is. But we look over the meaningfulness of large w_js by searching for different patterns and finding repetitive examples. For example for the pattern $\{PRP, EX(Existentialthere), SPACE, PERIOD\}$ we have the example: "See you there", "We will see you there", or "We look forward to seeing you there".

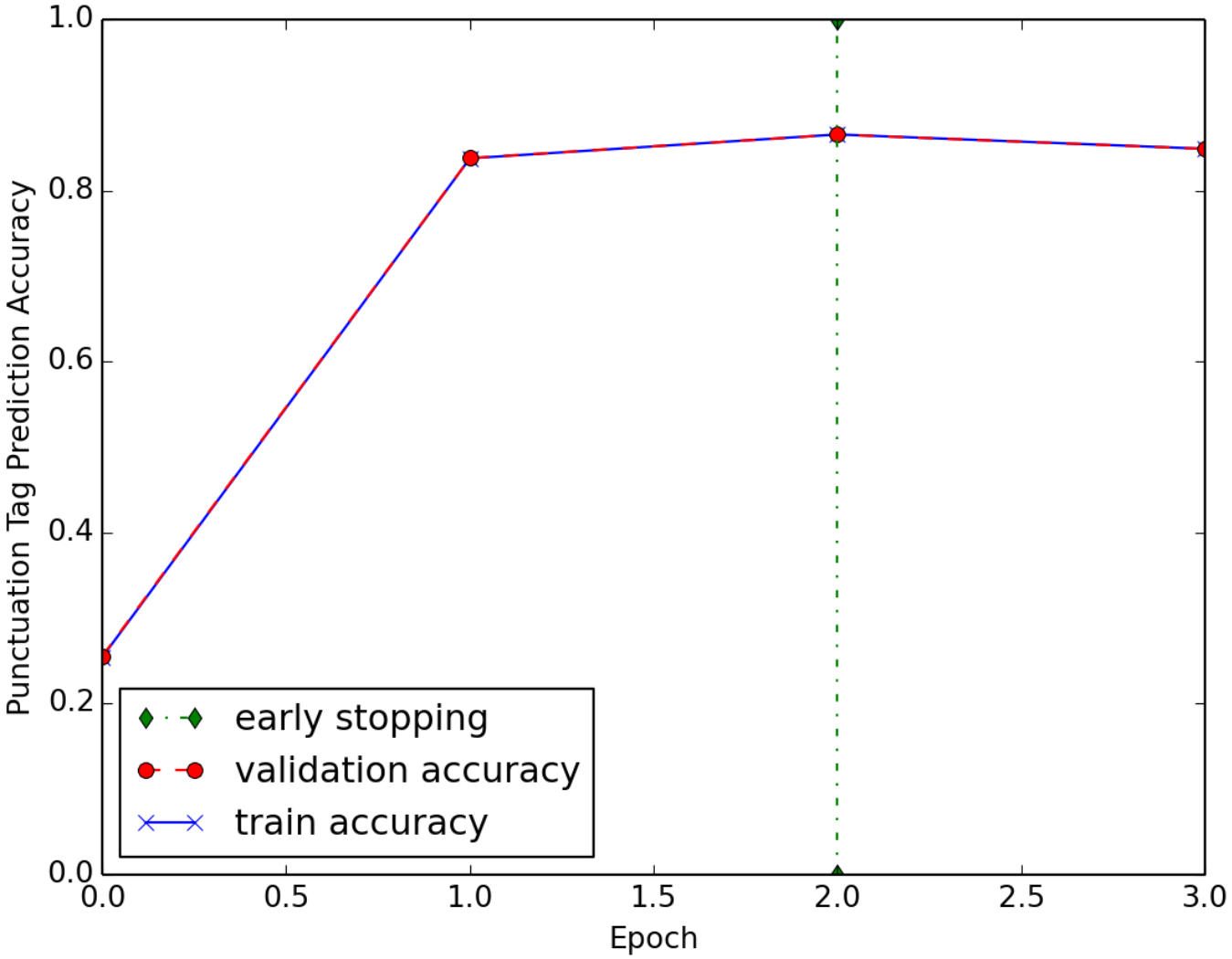

Figure 3: Early stopping for Contrastive Divergence.

Table 6: Contrastive Divergence performance on individual tag and sentence punctuation prediction.

Measure	**Validation set accuracy**	**Test set accuracy**
Individual tag prediction	0.866	0.852
Sentence level prediction	0.113	0.115

Table 7: Accuracy of predicting different tags by Contrastive Divergence

Method	**Validation set**		**Test set**	
	Accuracy	Frequency	Accuracy	Frequency
EXCLAMATION_POINT	0.486	2567	0.518	1066
SPACE	0.945	580077	0.947	235536
QUESTION_MARK	0.249	10904	0.247	4792
PERIOD	0.849	57259	0.847	22492
COLON	0.3	1009	0.5	1174
COMMA	0.086	28555	0.076	10665

Table 8: 20 Largest entries of the weight vector for Contrastive Divergence after learning.

Number	**Magnitue**	X_{i-1}	X_i	Y_{i-1}	Y_i
1	0.1	WRB	IN	SPACE	SPACE
2	0.1	NN	UH	SPACE	COLON
3	0.1	NN	UH	SPACE	PERIOD
4	0.1	ADJ	WH	SPACE	SPACE
5	0.1	PRP	ADJ	COMMA	SPACE
6	0.1	PRP	ADJ	SPACE	QUESTION_MARK
7	0.1	PRP	ADJ	SPACE	SPACE
8	0.1	IN	VB	SPACE	SPACE
9	0.1	VB	IN	SPACE	SPACE
10	0.1	VB	IN	SPACE	PERIOD
11	0.1	VB	IN	SPACE	COMMA
12	0.1	IN	VB	SPACE	QUESTION_MARK
13	0.1	IN	VB	SPACE	PERIOD
14	0.1	NN	SYM	COMMA	SPACE
15	0.1	CC	CD	SPACE	PERIOD
16	0.1	CC	CD	SPACE	SPACE
17	0.1	PRP	CD	SPACE	PERIOD
18	0.1	PRP	CD	SPACE	QUESTION_MARK
19	0.1	PRP	EX	SPACE	QUESTION_MARK
20	0.1	PRP	EX	SPACE	PERIOD

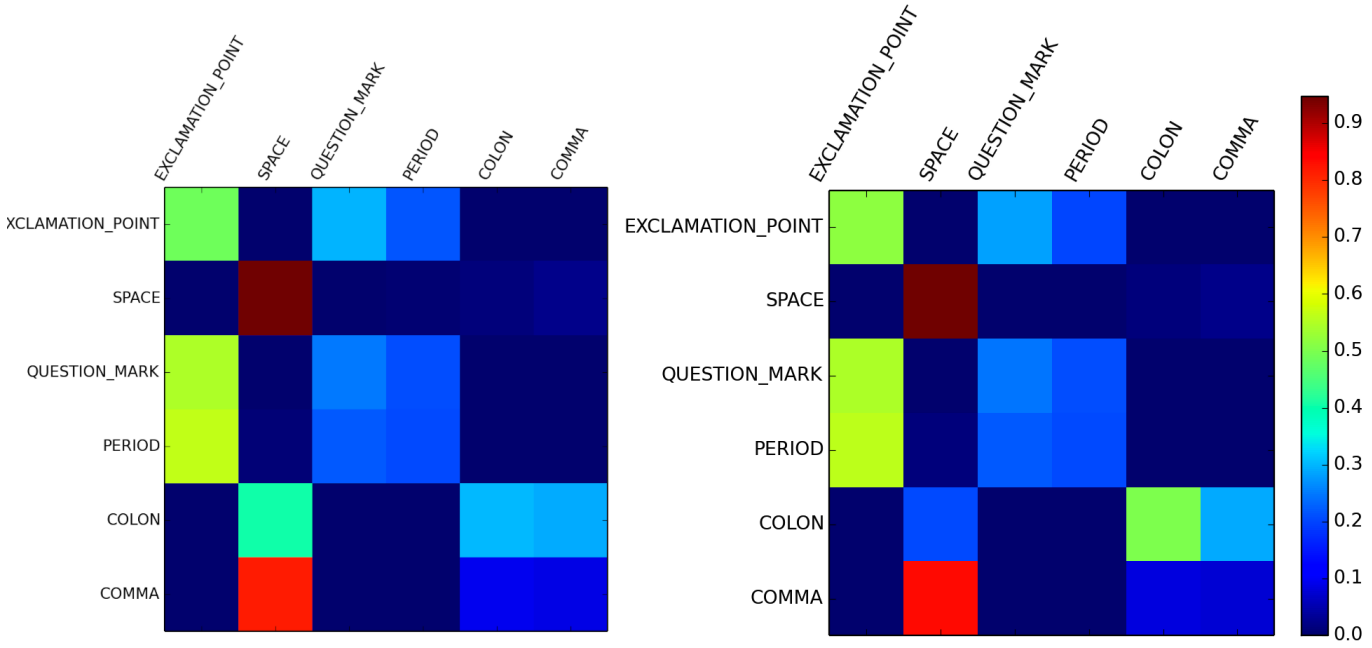

Figure 4: Confusion matrix for individual tag prediction for Contrastive Divergence for Left: validation and Right: Test data.

5.3 Model Averaging

In previous section, we saw that both models (Collin's Perceptron and Contrastive Divergence) cannot distinguish between different punctuation tags at the end of sentence. One reason for this is that the feature functions we picked look only locally to POS and punctuation tags. When this happens, the model tends to favor one specific tag at the end of sentences during training and prediction. One way to overcome the limitations of the feature functions is to train different models on the data and then combine their results. The basic idea is as follows: train different models that predict only a specific punctuation tag. Then combine these models using model averaging. By using different models for different punctuation tags, the effect of imbalanced frequency of target tags is alleviated. However, combining different models is not trivial as they are trained on different aspects of the training data. One easy way to combine different models is to average their weights and use that for prediction. This makes sense only in the case of bounded weights, where in training, weight components w_j are limited to a specified range $[-a, a]$. We pick this range to be the same for different models, and train a model for predicting each of the punctuation tags separately. We use a variant of Contrastive Divergence

method mainly because of simplicity of implementation. We believe that the choice of base model is not too important for this case. The training procedure for target tag z is as follows: pick the next training sample, change its punctuation tags only in places where the tag is z. The change is random and can be any target tag, including z itself. Then calculate the corresponding feature functions and update the weights. Early stopping is used in training each model. Then we average the weight vector of these models and use that for prediction.
We train 6 models for 6 target tags we have in the data. on 4 models, the prediction accuracy on validation set is more than 99%. On COMMA and COLON, however the prediction accuracy is much lower, around 51% for COMMA and 69% for COLON. Table 9 shows the prediction accuracy of the model with averaged weights on validation and test sets. For individual tag prediction, the averaged model achieved 83% for both validation and test sets.

Table 9: Accuracy of predicting target tags by averaged model

Method	**Validation set**		**Test set**	
	Accuracy	Frequency	Accuracy	Frequency
EXCLAMATION_POINT	0.161	2567	0.138	1066
SPACE	0.865	580077	0.866	235536
QUESTION_MARK	0.149	10904	0.151	4792
PERIOD	0.765	57259	0.762	22492
COLON	0.667	1009	0.859	1174
COMMA	0.182	28555	0.205	10665

5.4 Comparison Between Different Models

6 Conclusion