CSE 250B - Project 2

Lund, Sigurd Stoeen
sigurlu@stud.ntnu.no

Wolff, Thomas
thomawo@stud.ntnu.no

This report presents a method to predict punctation in English sentences using Conditional Random Fields (CRF).

Introduction

In a team of two students, we conducted a project whose goal was to learn a conditional random field (CRF) model, that can add punctation (commas, exclamation marks, etc.) to English sentences in which the punctation is absent. The intended application is for typing on a smartphone, where the keyboard may be too small to show letters and punctation simultaneously. A user should only need to type in the words of a sentence, and the punctation will be automatically added by the software[1].

We learn a model and test the accuracy on two given dataset, that consist of English sentences with punctation, taken from email messages.

The model is learned by using two different training methods: Collins' perceptron (CP) and contrastive divergence (CD). Both uses feature functions to learn the model, and the trainers task is to assign weights to these functions. These learned weights will then be used to map a sentence to a sequence of punctation tags. A variation of the Viterbi algorithm are used for computing this sequence.

CP achieved an accuracy rate of 93.75%, and CD 93.83%. The algorithms accuracy rate depends on how well the feature functions are defined, and how many of them one has. In this project we have not implemented any part of speech (POS) tagger, so we are not able to take advantage of any knowledge about the grammar in the designing of feature functions. By doing that, and put more effort in the design of feature functions, it should be possible to get even higher accuracy rates.

Design and analysis of algorithms

Two different learning methods will be presented for learning Conditional Random Fields (CRF), which is a special case of Log Linear Models, CP and CD. First is the problem we apply the two algorithms to defined, followed by an explanation of each of the methods.

Definition of the problem

The learning algorithm is supposed to learn a set of weights $\mathbf{w}$, which after training are used for prediction. In this project the set of weights are used for predicting punctuation marks for a given English sentence. More precisely, given a sequence of words $\bar{x}$, predict a sequence of tags, also called a label, $\bar{y}$, where tag y_i is the punctuation mark after word x_i in sentence $\bar{x}$. Here, tag y_i is an element in the set of possible punctuation marks used in the English language, and the word x_i is a word in the English language.

The problem of predicting the correct label for a given English sentence is formulated mathematically as:

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

I.e choose the label $\hat{y}$ that has the highest probability given the sentence $\bar{x}$ and the set of weights $\mathbf{w}$. If the number of different punctuation marks are m and the length of the given sentence is n, there are m^n different possible labels for that sentence. Going through all possible labels to determine which label has the highest probability is intractable as it requires $O(m^n)$ time. Thus, we need a method that reduces the time complexity for solving the prediction problem.

Finding a tractable way to calculate $\hat{y}$

In Log Linear Models the probability of y, given x and the set of weights $\mathbf{w}$ is:

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

where

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

and J is the number of feature functions. Since the denominator is equal for every y given x and $\mathbf{w}$, to determine which label y has the highest probability, we can look only at the numerator. We can drop the exponential function as well, and only look at the sum in the numerator when evaluating the probability for different y. The expression we use to evaluate the probability of label y becomes

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

To achieve tractability when solving the prediction problem we introduce the following restriction on the feature functions:

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

This means that each feature function $F_j(\bar{x}, \bar{y})$ is a sum of a low level feature function f_j that only looks at consecutive pairs of tags, for all positions i. Note that the low level feature function f_j can look at the whole sentence. The probability of a label now becomes:

$$p(y|x;w) = \sum_{j=1}^{J} w_j \sum_{i=1}^{n} f_j(\bar{x}, y_{i-1}, y_i, i, n)$$

As the next step in computing $\hat{y}$ in reasonable time we define:

$$g_i(y_{i-1}, y_i) = \sum_{j=1}^{J} w_j f_j(\bar{x}, y_{i-1}, y_i, i, n)$$

The probability of a label y given x and $\mathbf{w}$ now becomes:

$$p(y|x;w) = \sum_{i=1}^{n} g_i(y_{i-1}, y_i)$$

Now we have to sum g_i for all positions i in the sentence. As a preprocessing step in solving the prediction problem we compute g_i for all positions i and for every combinations of punctuation marks y_{i-1} and y_i. Recall that there are m different punctuation marks. That means that there are m^2 different combinations of consecutive pairs of punctuation marks for a given position i. If there are J different feature functions, computing the g-functions for all positions and every pair of punctuation marks, takes $O(n \cdot m^2 \cdot J)$ time and requires $O(n \cdot m^2)$ space.

As the last step in achieving a reasonable time complexity for solving the prediction problem, we define:

$$U(k, v) = \max_{y_{k-1}} U(k-1, y_{k-1}) + g_k(y_{k-1}, v)$$

U(k, v) gives the probability of most likely label of length k where y_k is locked to be v. Given $U(k-1, y_{k-1})$ and that the g-functions are calculated prior to this step, calculating U(k, v) requires $O(m)$ time because we take the argmax over all m possible tags. U makes up a matrix where the element in row i and column v is the probability for the best label of length i ending with the tag v. One can finally compute the most likely label $\hat{y}$ for the given sentence $\bar{x}$ and set of weights w in the following way:

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

and every $\hat{y}_i$ where $i < n$ as:

$$\arg\max_{v} U(i-1, v) + g_i(v, y_i)$$

Given the g-functions, the time complexity for solving the prediction problem is $O(m^2 \cdot n)$. In total, the time complexity for solving the prediction problem is $O(m^2 \cdot J \cdot n + m^2 \cdot n)$. Where the first term is for calculating the g-functions, and the latter term is for filling the U-matrix.

The Collins' Perceptron algorithm

CP is an alternative training method to Stochastic Gradient Following (SGF). Like SGF, CP goes through all the examples in the training data once in what is referred to as an epoch. In each iteration of an epoch one example is drawn from the training examples, and each weight w_j in the set of weights $\mathbf{w}$ are updated according to the update rule:

$$w_j = w_j + (F_j(\bar{x}, \bar{y}) - F_j(\bar{x}, \hat{y}))$$

This update rule is similar to the one used in SGF:

$$w_j = w_j + \alpha(F_j(\bar{x}, \bar{y}) - E[F_j(\bar{x}, y\prime)])$$

One of the ideas behind CP is to estimate the expectancy of y' instead of calculating it exactly. The expectancy is:

$$E[F_j(\bar{x}, y\prime)] = \sum_{y\prime \in Y} p(y\prime|x; w) \cdot F_j(x, y\prime)$$

The sum of probabilities over all $y\prime$ is equal to one, and the expectancy can thus be approximated to be:

$$E[F_j(\bar{x}, y\prime)] \approx F_j(x, \hat{y})$$

where $\hat{y}$ is the label with the highest probability, given the current model and sentence.

Understanding of the update rule

What happens to w_j in CP? When looking at the update rule we see that w_j increases if $F_j(x, y) > F_j(x, \hat{y})$, and decreases if $F_j(x, y) < F_j(x, \hat{y})$. It stays the same if $F_j(x, y) = F_j(x, \hat{y})$. Recall that the expression we use to evaluate the probability of label y given sentence x and weights w is:

$$p(y|x; w) = \sum_{i=1}^{n} g_i(y_{i-1}, y_i)$$

From this we can see that in order to make the true label more probable we have to make its sum higher. To make its sum higher we have to increase the weights corresponding to the feature functions for which the true label has a higher value than the currently most probable label. Conversely, we have to decrease the weights corresponding to the feature functions for which the true label has a lower value than the currently most probable label. In this manner we make the true label more probable. Note that as a result of the sum over all probabilities for every label y being be equal to one, we also lower the probability for the other labels.

Another difference between CP and SGF, is that in CP you use a learning rate α is equal to one. The reason for this is that we are only interested in determining the identity of the most probable label $\hat{y}$, not its probability. The learning rate will not affect which label is most probable, and thus we can set the learning rate to be one in CP.

Contrastive Divergence

CD is a special case of Gibbs Sampling (GS), which will be presented in the following section. After introducing GS, the motivation behind CD and how it is a special case of GS is explained.

Gibbs Sampling

If we could draw random samples from the distribution $p(y\prime|x;w)$, we could have calculated the expectancy term in the SGF update rule using:

$$E[F_j(\bar{x},y\prime)] = \frac{1}{K}\sum_{k=1}^{K} F_j(x,\bar{y}_k)$$

But how can we draw a random sample from this distribution? If we were to evaluate the probability of every possible label, and then draw a random one amongst them based on their probability, we would use $O(m^n)$ time, which is intractable.

Suppose $\hat{y} = \{y_1, y_2, y_3 \ldots y_k\}$ and that we can compute:

$$p(Y_i = v|x, \{y_1, y_2 \ldots y_{i-1}, y_{i+1}, \ldots y_k\}\,; w)$$

exactly numerically. Then we can draw a random label from the distribution $p(y\prime|x;w)$ using the algorithm described below.

```
Initialize $\{y_1, y_2, y_3 \ldots y_k\}$ ;
**while** *Not enough iterations* **do**
    Draw a new value for $y_1 \sim p(Y_1|x, \{y_2, y_3 \ldots y_k\}\,; w)$ ;
    Draw a new value for $y_2 \sim p(Y_2|x, \{y_1, y_3 \ldots y_k\}\,; w)$ ;
    $\vdots$
    Draw a new value for $y_n \sim p(Y_2|x, \{y_1, y_2, y_3 \ldots y_{n-1}\}\,; w)$ ;
**end**
```

We can evaluate $p(Y_i = v|x, \{y_1, y_2 \ldots y_{i-1}, y_{i+1}, \ldots y_k\}\,; w)$ for a given v using the following expression:

$$\frac{\exp g_i(y_{i-1}, v) \exp g_{i+1}(v, y_{i+1})}{\sum_{v\prime} \exp g_i(y_{i-1}, v\prime) \exp g_{i+1}(v\prime, y_{i+1})}$$

Calculating the whole distribution of $p(Y_i = v|x, \{y_1, y_2 \ldots y_{i-1}, y_{i+1}, \ldots y_k\}\,; w)$ requires $O(m)$ time because we evaluate the expression above for each v. It can be proven that if you do the while loop in the algorithm an infinite number of times, the random sample given follows the distribution $p(y\prime|x;w)$.

Idea behind Contrastive Divergence

The idea behind CD is that rather than using the most probable label $\hat{y}$ in the update rule, use a label y^*, where y^* is a label that is similar to the true label, but has higher probability. Why use y^* instead of $\hat{y}$? Finding $\hat{y}$ requires that we solve the prediction problem, which we showed earlier that can be computed in $O(n \cdot m^2 \cdot J + m^2 \cdot n)$ time. We now show that finding y^* requires less computation than solving the prediction problem, and thus improves our time complexity for training.

How do we find y^*? We use GS where we initialize $\{y_1, y_2, y_3 \ldots y_k\}$ to be the true label, and do only one iteration of the algorithm described above. The total time to calculate y^* is $O(m \cdot n)$ since there are n tags for a sentence of length n, and that there are m possible tags in each position. However, this assumes that the g-functions are already computed. The total time to find y^*, including calculating the g-functions, is then $O(m^2 \cdot J \cdot n + n \cdot m)$. We see that this requires less computation than solving the prediction problem.

Design of Feature Functions

Once the algorithms are implemented correctly, it is the design of feature functions that will affect how well the trainer can learn a good model.

The implementation of feature functions described in this paper is defined as templates, so when you define a template you get many different feature functions. How many depends on the type of the template, since it is defined in one or more dimensions.

A typical one-dimensional template is if you are looking at the last position over all different tags. In the punctuation predicting case, this template will generate one feature function that have 'period' at this tag, and this function will probably get a high weight. The other functions generated will have lower weights.

A two dimensional template takes an additional set to and makes combinations with the tag set. This could be a set of a closed word class e.g. conjunction words. All the combinations with these two sets will be generated, and some functions might be useful for predicting a 'comma'.

The templates should be added one by one to see if they actually turns out to be positive for learning the model. If not, the template can just be removed.

When to stop training

Both CP and CD loops through all the examples in the training set, before the model is tested on a validation set. The accuracy rate will typically go up in the first epochs, and then go down since the model is overfitting the training data. Our implementation stops when the accuracy rate drops from previous iteration. The previous model, which is the best model seen so far, is the one used for prediction.

Experiments

Design of Experiments

The data set used for learning the model consisted of 70 115 English sentences. Both CP and CD split those sentences into a training set and a validation set. The validation test is used after each epoch for determining the accuracy rate. The ratio of the size of the training set and the size of the validation set is 50-50 in the experiment results presented here. Different ratios have been tested during development, and with such a big set of training data, the ratio does not seem to have much effect.

The learning part will also be tested on a small subset of the training set to see if that have any affect on the accuracy achieved. The size of both set will be of 1000 sentences.

After the model is trained it is tested on a different test set containing 28 027 English sentences. The accuracy rate is measured on the test set tag by tag. That means that the accuracy rate is the number of correct predicted tags in all sentences divided by the total number of tags. The accuracy rate is also measured for each tag to see what kind of tags the model is able to predict.

Results of Experiments

CP achieved an accuracy rate of 93.75%, and CD 93.83%. Since the accuracy is measured on a tag by tag level, it is also interesting to see how good the accuracy is if you just assign space to all tags except the last one in a sentence where you assign period. This actually achieve an accuracy on 93.46% on the test set, only 0.4% worse than the CRF implementation with CD.

All the tests results is an average score from 10 runs of the program.

Tag	CP small	CP large	CD small	CD large
Space	99.79%	99.58%	99.85%	99.73%
Period	97.61%	97.42%	97.69%	97.59%
Comma	6.86%	8.84%	7.15%	8.49%
Question mark	18.80%	21.60%	16.53%	19.57%
Exclamation point	0%	0%	0%	0%
Colon	0.01%	5.50%	0.03%	0.01%
Total	93.50%	93.75%	93.83%	93.83%

Table 1: Accuracy on test set

The difference between the CP and CD seems to be small as we can see in Table , but CD achieves slightly better performance. It is not surprising that the difference is small since the only change is that CP calculates $\hat{y}$ where CD calculates an evil twin to the true label, y^*. Both algorithms is making the true label more probable than other labels.

One could think that CD should be much more efficient since it does not need to calculate $\hat{y}$, but it turns out that it actually takes some time to find an evil twin which has higher probability than the true label. The run time for one epoch, which is going through all training examples once, is measured for both algorithms in Table 2, and CP turns out to use 6% less time for one epoch. However CD also uses more epochs before the accuracy rate drops, so the runtime in total is 25% better for CP compared to CD if you take the average number of epochs into account.

The difference between a small and a large training set is small, and for the CD method, it is actually the same accuracy.

	CP	CD
Runtime	289s	308s
Epochs until finished training	2.4	3.0

Table 2: Runtime for one epoch and number of epochs

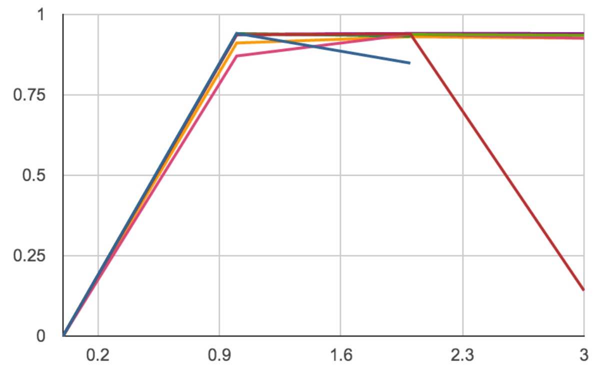

Figure 1: Accuracy on validation data vs. number of epochs using CP for 10 runs using all training data

In Figure 1 and Figure 2, each line is representing one complete run of training the model with CP and CD. Most runs are in the same range, but CP varies a little more than CP.

Findings and lessons learned

The model learned is only able to predict $0.3 - 0.4\%$ better than a naive guess of spaces and periods. That means that the learner find it hard to predict other tags, and even if the accuracy on spaces is very high, it misses some of them because of the other rules, so the percentage you win on the colon tags, does not necessary make the overall accuracy much higher. It

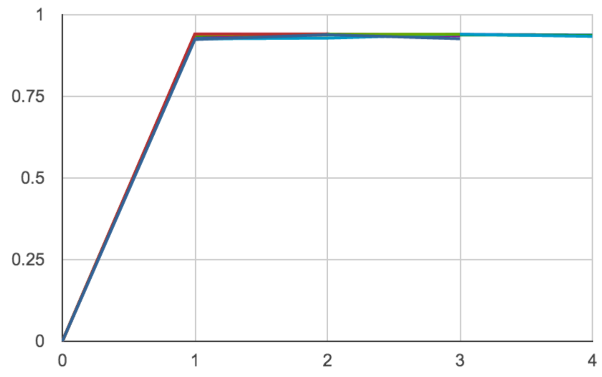

Figure 2: Accuracy on validation data vs. number of epochs using CD for 10 runs using all training data

makes some correct predictions on question marks, commas and colons, but this is only in the cases of question words for question marks, conjunction words for commas, and if the second word is starting with upper case for colon. This is not happening often enough to improve the accuracy further. Exclamation points and colons share some of the same property, but the feature functions used here can not distinguish them, and turns out to predict colon in those cases. It should be possible to write more feature functions that will be able to learn more with the grammar knowledge from POS tagging.

If a large or a small training and validation set is used does not make much difference, but this can be because of the simple feature functions. Some feature functions may need more data to learn. The difference with this implementation is almost nothing on the ”easy” tags, spaces and periods, but it need more training data to learn question marks, commas and colons. The importance of a big training set would therefore be more important if the feature functions are more sophisticated.

In this implementation CP was more efficient, but CD achieved slightly better performance. CD also achieved the same accuracy with a small subset of the training data, so that makes the CD algorithm even more efficient again.

References

1. Project description - http://cseweb.ucsd.edu/ elkan/250B/project2.pdf

2. Lecture notes - http://cseweb.ucsd.edu/ elkan/250B/CRFs.pdf