Application of Discriminative Training Method with Perceptron Algorithm for Hidden Markov and Conditional Random Field Models

Divya GROVER
divyagrover@u-psud.fr

Anh Khoa NGO HO
ngohoanhkhoa@gmail.com

Abstract

We apply the discriminative training method for Hidden Markov and Conditional Random Field(CRF) models to typos-correction problem in Natural Language Processing. The method, proposed by Michael Collins in 2002 [1], is based on perceptron algorithm, in which Viterbi algorithm is used for decoding the data sample and then the parameters are updated iteratively. We show experimental results where we compare this method with Maximum Likelihood estimation for HMM and CRF model trained with this method compared to CRF trained with Maximum Likelihood.

1. Introduction

The research of Michael Collins gives a simple alternative training method for HMM/CRF style models, which is a variant of the perceptron algorithm. For this typos-correction problem, there are two sequences: observed sequence and correct sequence. For example, the observation sequence could be "hnt" but the correct sequence could be the sequence "hit". Note that henceforth we call the correct sequence as **tag**, as this problem resembles very much to the tagging problem in general. In this method, the Viterbi algorithm predict the structured object/the tag sequence from the observation sequence. Based on perceptron algorithm, the parameters of the underlying model are updated by simple additive operations. These updates depends on the difference between the real and predicted tag sequence.

This report describes shortly the HMM,CRF models and the perceptron algorithm in section 2. The experiments including the dataset description, the code explanation and the result comparison between these two approaches are shown in section 3. The section 4 presents a critical view of these methods.

2. Models and Training Methods

There are primarily two problems in a structured model like HMM/CRF:

1. Inference of label/hidden-states given a parameter setting and observations.
2. Parameter estimation given a training set.

We shall be comparing the effects of perceptron training for parameter estimation versus the default maximum-likelihood(ML) estimation for two models HMM and CRF. For example, in case of HMM, the training algorithm needs to estimate the parameters associated with trigram tags and pair- observation tags. The main difference between perpectron learning and ML estimation is the update step for the parameters. In fact, ML estimation computes the parameters before using Viterbi algorithm for prediction, whereas perceptron algorithm updates these parameters by calling Viterbi algorithm for each sequence i.e. for each word to get the current estimate of its tag.

2.1. HMM/CRF with Maximum-Likelihood

For the HMM model, we count the tag trigram and pair observation/tag in all sequences of the dataset, then calculate the log probability that a tag depends on the previous two tags and the log probability that an observation depends on a tag. In this case, we need to compute the two normalization constants by sum of tag transition probabilities and sum of observation emission probabilities. The score of a tag sequence is the log of the joint probability of observation and tag sequence. The Viterbi algorithm uses these parameters for predicting the highest scoring tag sequence.
For the CRF model, the parameter estimation is usually done by an iterative gradient descent algorithm which involves minimizing the loss (negative log likelihood). Forward-backward message passing is needed for each example for calculating the posterior over the given label, which is then used to calculate a single gradient iteration over the parameter-space.

2.2. Perceptron Algorithm

Perceptron was one of the earliest models in machine learning, and the learning rule for that simple model is referred to as perceptron algorithm. In this context of HMM, it works as follows1 : we suppose trigrams tags and pairs

observation/tag are the features of the dataset. The parameters estimated, indicating the importance of these features or the conditional probabilities mentioned, are initially set to be zero. For each observation sequence, Viterbi algorithm uses these parameters to find the best tag sequence. If the true tag sequence is different this best tag sequence predicted, these parameters are updated by using the number of tag trigram and of pair observation/tag in these two sequences.

Input: number of iterations over training set T
Parameters: $\alpha_{trigram}$ parameters of tag trigram and α_{pair} parameters of observation/tag pair
Initial: $\alpha_{trigram}$ = 0 ; α_{pair} = 0
for *each iteration of T* **do**
  for *each observation sequence in dataset* **do**
    Predicting the best tagged sequence with Viterbi algorithm
    for *each tag trigram* **do**
      c_1 = number of tag trigram in the true tag sequence
      c_2 = number of tag trigram in the predicted tag sequence
      $\alpha_{trigram}$ = $\alpha_{trigram}$ + c_1 - c_2
    end
    for *each observation/tag pair* **do**
      c_1 = number of pair in the true tag sequence
      c_2 = number of pair in the predicted tag sequence
      α_{pair} = α_{pair} + c_1 - c_2
    end
  end
end

Algorithm 1: Perceptron algorithm for HMM training

Notice that here the trigram tags and observation/correction pairs can be regarded as features, each associated with it's own weight, we then use the perceptron algorithm to estimate these weights. This notion of discriminative training can be generalized to other models, such as CRF. The generalized algorithm is shown below2.2 : In a general setting, The task is to learn a mapping from inputs $x \in X$ to outputs $y \in Y$. For example, X might be a set of sentences, with Y being a set of possible tag sequences. We assume:

Inputs: Training examples (x_i, y_i)
Initialization: Set $\bar{\alpha} = 0$
Algorithm:
For $t = 1 \ldots T$, $i = 1 \ldots n$
Calculate $z_i = \arg\max_{z \in \mathbf{GEN}(x_i)} \Phi(x_i, z) \cdot \bar{\alpha}$
If$(z_i \neq y_i)$ then $\bar{\alpha} = \bar{\alpha} + \Phi(x_i, y_i) - \Phi(x_i, z_i)$
Output: Parameters $\bar{\alpha}$

Figure 1. A generalized variant of the Perceptron Algorithm

- Training examples (x_i, y_i) for $i = 1...n$
- A function GEN which enumerates a set of candidates $GEN(x)$ for an input x.
- A representation ϕ mapping each $(x, y) \in X \times Y$ to a feature vector $\phi(x, y) \in R^d$
- A parameter vector $\bar{\alpha} \in R^d$.

The components GEN,ϕ and $\bar{\alpha}$ define a mapping from an input x to an output $F(x)$ through

$$F(x) = \underset{y \in GEN(x)}{\operatorname{argmax}} \phi(x, y).\bar{\alpha} \quad (1)$$

where $\phi(x, y).\bar{\alpha}$ is the inner product $\sum_s \alpha_s \phi(x, y)$.The learning task is to set the parameter values $\bar{\alpha}$ using the training examples as evidence.

3. Experiments

In this section, we give a brief introduction to the dataset used, then for the uninitiated, we describe in detail what a CRF model is, moving onto the code used and results obtained.

3.1. Dataset

We use typos-correction dataset. In this dataset, we use two sequences of character(words), one including the correct letter that should have been typed and second the actual letter that was typed. For example "rit" instead of "rat". As a result, with 26 letters, there are 26 states and 26 observations. In the assumption of a trigram HMM, a letter depends on the previous two letters. Therefore, each sequence is a word and Viterbi algorithm predicts the sequence of the correct letter based on the actual letters.

For the CRF model, a feature-vector representation is used. A feature vector representation $\phi : H \times T \rightarrow R^d$ is a function ϕ that maps a historytag pair to a d-dimensional feature vector. Each component $\phi_s(h, t)$ for $s = 1...d$ could be an arbitrary function of (h, t). It is common (e.g. , see [2]) for each feature ϕ_s to be an indicator function. For example, one such feature might be

$$\phi_2(h, t) = \begin{cases} 1 & \text{if current letter is "o" is word is "hot"} \\ 0 & \text{if otherwise} \end{cases}$$

Similar features might be defined for every word/tag pair seen in training data. Another feature type might track trigrams of tags, for example $\phi_3(h, t) = 1$ if $(t_{-2}, t_{-1}, t) =$

(A, N, T) and 0 otherwise. Similar features would be defined for all trigrams of tags seen in training. A real advantage of these models comes from the freedom in defining these features: for example, ([2]; [3]) both describe feature sets which would be difficult to incorporate in a generative model.

3.2. Code

The code consists of 2 classes, HMM and CRF tagger, and separate training methods for ML-estimation and Perceptron learning. The training for CRF, we compare the accuracy across training methods for these two models, and also show the effect of averaged weight updates when using the perceptron rule. Finally we use a very different implementation of CRF trained using perceptron rule to show that the underlying features also affects the accuracy of the model.

3.3. Results

The results are summarized in the table below:

Error Table		
Model	Perceptron estimation	ML estimation
HMM	0.1	0.04
CRF	0.2	0.1
PerceptronTagger	0.05	NA

Test error for perceptron estimation are generally bad, except for the last model, this can be attributed to the following reasons:

1. For the HMM implementation, as it is written by us, it is not optimized to perform fast viterbi inference(also because of pure python implementation).

2. For the CRF model, although the perceptron-learnt model gives 20% even the ML-estimated model performs relatively bad (0.1 error). This can be attributed to bad feature selection for this given implementation.

3. In support of above argument, a separate CRF implementation, called PerceptronTagger, which uses only the perceptron rule for training gives 5% error, almost equivalent to an HMM trained with ML(0.04).

4. Conclusion

In conclusion, we would like to highlight the following points:

1. Perceptron learning works as good ML estimation, although, for simpler models such as HMM, it requires significantly longer training time.

2. Being a simple learning algorithm, it can be used across many models(HMM and CRF) and serves as a general training algorithm.

3. It indirectly helped bridge the understanding gap between generative(HMM) and discriminative(CRF) models by looking at generative models as a subclass of such discriminative models, where the feature representation is across both the input and labels.

4. Our contribution was to reinforce the results of this well proven paper on a new dataset(typo-correction), as well as to provide a simple, all inclusive ipython-notebook style code, which can be used for pedagogic purposes as well.

References

[1] Michael Collins. *Discriminative Traning Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms*. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Philadelphia, July 2002.

[2] Ratnaparkhi, A. *Maximum Entropy Markov Models for Information Extraction and Segmentation*. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 1996.

[3] Andrew McCallum, Dayne Freitag and Fernando Pereira. *A Maximum Entropy Model for Part-Of-Speech Tagging*. Proceedings of ICML, 2000.

[4] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly, 2009. `http://www.nltk.org/book/`

[5] Terry Peng, Mikhail Korobov. *Python-crfsuite*. O'Reilly, 2009. `http://python-crfsuite.readthedocs.io/`