Discrete Time Discrete Symbol Sequence Prediction Using HMM

Zhengli Zhao, Karthik Prasad, Abhisaar Sharma

Abstract

This paper describes our graphical-model approach to the data-oriented project SPiCe (Sequence PredictIction ChallengE). Hidden Markov model (HMM) is a statistical Markov model in which the system being modelled is assumed to be a Markov process with latent states and can be presented as the simplest Bayesian network. We show our approach in modelling the problem as a HMM and using learning techniques like Baum-Welch and Spectral learning to learn the parameters of this graphical model and make predictions. The predictions generated from our approach currently stands as rank 3 amongst all the participants on the leader-board.

1 Introduction

The Sequence PredictIction ChallengE (SPiCe) is a competition where the aim is to learn a model that allows the ranking of potential next symbols for a given prefix and submitting a ranking of the 5 most probable next symbols. Training datasets consist of variable length sequences with a fixed number of symbols. The 5 next symbols which we submit is scored on a ranking metric based on normalized discounted cumulative gain.

Let the test set be made of prefixes $y_1, y_2, ..., y_M$ and the next symbols ranking submitted for i^{th} prefix y_i be $(\hat{a}_1^i, ..., \hat{a}_5^i)$ sorted from more likely to less likely. The program evaluating the submissions has access to to $p(.|y_i)$, i.e. the target probability distribution of possible next symbols given the prefix y_i. The NDCG is then

$$NDCG_5(\hat{a}_1^i, ..., \hat{a}_5^i) = \frac{\sum_{k=1}^5 p(\hat{a}_k^i|y_i)/log_2(k+1)}{\sum_{k=1}^5 p_k/log_2(k+1)}$$

The competition uses real-world data from different fields like Natural Language Processing, Biology, Signal Processing, Software Verification.

We can use a Hidden Markov Model to model this problem, where the training sequences can be treated as discrete time observables (emission variables) and the unobserved latent states can be used to capture the intrinsic sequence structure. A hidden Markov model can be considered a generalization of a mixture model where the hidden variables which control the mixture component to be selected for each observation, are related through a Markov process rather than independent of each other.

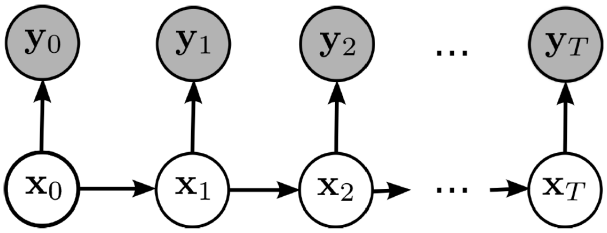

Figure 1: A hidden Markov model, x_i are the hidden variables and y_i are the observables.

2 Methodology

While trying to model the training sequence as an HMM, we would like to learn the parameters of this graphical model. We will present two approaches that we explored, the first one is the BaumWelch algorithm which uses the EM algorithm to find the maximum likelihood estimate of the parameters of a hidden Markov model given a set of observed feature vectors and the second one is using spectral methods.

2.1 Baum-Welch Algorithm

The BaumWelch algorithm is used to find the unknown parameters of a hidden Markov model (HMM). It makes

use of the forward-backward algorithm. Let X_t be a discrete hidden random variable with N possible values. We assume the $P(X_t|X_{t-1})$ is independent of time t. Let the state transitions be described by A which is a homogeneous time independent stochastic transition matrix.

$$A = \{a_{ij}\} = P(X_t = j|X_{t-1} = i)$$

The initial state distribution is given by

$$\pi_i = P(X_1 = i)$$

Let the observation variables be Y_t which can take one of K possible values. Let B describe the probability of a certain observation at time t for a state j. The emission matrix is then defined as

$$B = \{b_j(y_t)\} = P(Y_t = y_t|X_t = j)$$

A single observable sequence is given by

$$Y = (Y_1 = y_1, Y_2 = y_2, Y_3 = y_3, .., Y_T = y_t)$$

A Hidden Markov Model is completely parametrized by $\theta = (A, B, \pi)$. The BaumWelch algorithm finds a local maximum for $\theta^* = argmax_\theta P(Y|\theta)$, the HMM parameters θ that maximise the probability of the given observation sequence.

For finding the locally optimum θ, it is randomly initialized as (A, B, π). Since it has EM type updates, the Expectation step involves finding the forward and backward probabilities, which uses a dynamic programming algorithm to find the most likely states. The Maximization step then uses these probability values to estimate the parameters θ of the HMM.

2.1.1 Forward Procedure

Let $\alpha_i(t)$ =$P(Y_1 = y_1, Y_2 = y_2, ..., Y_t = y_t, X_t = i|\theta)$, which is the probability of seeing the output sequence $y_1, y_2, ..., y_t$ and being in state i at time t. This is found recursively by the following equations:

$$\alpha_i(1) = \pi_i b_i(y_1)$$
$$\alpha_j(t+1) = b_j(y_{t+1}) \sum_{i=1}^{N} \alpha_i(t) a_{ij}$$

2.1.2 Backward Procedure

Let $\beta_i(t) = P(Y_{t+1} = y_{t+1}, ..., Y_T = y_T|X_t = i, \theta)$, which is the probability of the ending partial sequence $y_{t+1}, ..., y_T$ given the starting state i at time t. $\beta_i(t)$ is found recursively by:

$$\beta_i(T) = 1$$
$$\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) a_{ij} b_j(y_{t+1})$$

One problem is that in case of long sequences as required by the Sequence prediction challenge, the forward probability values can go to zero exponentially. The solution to this problem is provided in the implementation section by using a normalized version of the Forward Backward procedure.

2.1.3 Updates

Define $\gamma_i(t)$ which will be the probability of being in a state i at time t given the observed sequence Y and the parameters θ. It can be shown that

$$\gamma_i(t) = P(X_t = i|Y, \theta) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}$$

Lets define $\xi_{ij}(t)$ as the probability of being in state i and j at times t and $t+1$ respectively given the observed sequence Y and parameters θ which can be shown to be equal to

$$\xi_{ij}(t) = P(X_t = i, X_{t+1} = j|Y, \theta)$$
$$\xi_{ij}(t) = \frac{\alpha_i(t) a_{ij} \beta_j(t+1) b_j(y_{t+1})}{\sum_{k=1}^{N} \alpha_k(T)}$$

θ is updated with the following set of equations:

$$\pi_i^* = \gamma_i(1)$$

which is the probability of state i at time 1.

$$a_{ij}^* = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$$

which is the expected number of transitions from state i to state j compared to the expected total number of transitions away from state i.

$$b_i^*(v_k) = \frac{\sum_{t=1, o_t = v_k}^{T} \gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)}$$

$b_i^*(v_k)$ is the expected number of times the output observations have been equal to v_k while in state i over the expected total number of times in state i. The updates are done until a desired level of convergence.

2.2 Spectral learning

3 Implementation

3.1 Baum-Welch Algorithm

The Baum-Welch algorithm was written in Java language. Python was initially used but dropped since it

had extremely slow performance[1]. The following points describe the sequence of optimizations we did.

- Hmm learn[2] - a python based library which consisted algorithms for unsupervised learning and inference of Hidden Markov Models was first used to evaluate the accuracy of this method on the data sets. Hmm learn is also included in scikit, however this library took a lot of time learning the model using Baum-Welch. The number of input sequences were at least 20,000 for each problem and the number of hidden states was at least 4. For the smallest problem it took more than one day to train the model. For larger problems, the code took about 4 days to run and exited with out of memory errors - this library was not very useful for our problem.

- Due to the inefficient library, we implemented the Baum-Welch algorithm in Java. Since the number of observed sequences is large, we implemented a mini-batch type learning of Baum-Welch algorithm. The input sequences were broken into batches (about 10-50 per batch) and the HMM parameters were learned for a given batch. For the next mini-batch the HMM was initialized with θ obtained from the previous batch.

- To get more stable and accurate solutions, we decreased contribution of the weight learned by later mini-batches by using weight smoothing.

$$\Theta_n = (1 - \alpha_n)\Theta_{n-1} + \alpha_n \theta_n$$

  Where α_n is $1/n$, n is the number of the mini-batch. Θ_n is the weight of the HMM after learning n batches, θ_n is the locally optimal set of weights learnt on the n^{th} batch by Baum-Welch algorithm initialized at Θ_{n-1}

- For longer sequences, Baum-Welch algorithm is hard to use because the forward probability values quickly became very small and go out of range of float and double data-types. This leads to underflow problems in forward-backward algorithm. We tried solving the problem by using BigDecimal class of java which does exact arithmetic, but it was too slow hence could not be used. Another approach we tried was transforming the probabilities into corresponding logs, but there are terms in Baum-Welch which would then need computation of sum inside logs, hence this approach was not very useful.

- Baum-Welch was then modified to prevent underflow, the idea is to normalize $\alpha_t(i)$ so that $\hat{\alpha}_t(i)$ - the normalized $\hat{\alpha}_t(i)$, would be proportional to $\alpha_t(i)$ and sum to 1 over all possible states. We can calculate the normalizers using the following equations.

$$\sum_{i=1}^{N} \hat{\alpha}_i(t) = 1, \hat{\alpha}_t(i) = \prod_{k=1}^{t} \eta_k \alpha_t(i)$$
$$\prod_{k=1}^{t} \eta_k \alpha_t(i) = 1/\sum_{i=1}^{N} \alpha_t(i)$$

  In the normalized Baum-Welch algorithm, we do a normalization at each step using the constants η_t for both the forward and backward steps for all the values. The updates are done in using the regular formulae as described earlier.

3.2 Spectral learning

4 Results

We participated wither the team name "codeBlue" and our current competition rank is number 3!. Thanks to Zhengli's untiring efforts and unparalleled intelligence$\longrightarrow \infty$:D.

Table 1: Leaderboard

Rank	Team	Score
1	vha	5.16
2	ToBeWhatYouWhatToBe	5.02
3	codeBlue	4.75
4	ushitora	3.81
5	JGR	3.20
6	uwtacoma	1.93

[1]Baum-Welch speed comparison: http://www.math.univ-toulouse.fr/ agarivie/Telecom/code/index.php

[2]https://github.com/hmmlearn/hmmlearn

References

References follow the acknowledgements. Use unnumbered third level heading for the references title. Any choice of citation style is acceptable as long as you are consistent.

J. Alspector, B. Gupta, and R. B. Allen (1989). Performance of a stochastic learning microchip. In D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 1*, 748-760. San Mateo, Calif.: Morgan Kaufmann.

F. Rosenblatt (1962). *Principles of Neurodynamics.* Washington, D.C.: Spartan Books.

G. Tesauro (1989). Neurogammon wins computer Olympiad. *Neural Computation* **1**(3):321-323.