Discrete-Time Discrete-Symbol Sequence Prediction Using Graphical Models

Abhisaar Sharma
abhisaas@uci.edu

Karthik Prasad
prasadkr@uci.edu

Zhengli Zhao
zhengliz@uci.edu

Abstract

This paper describes our graphical-model approach to the data-oriented project SPiCe (Sequence PredictIction ChallengE)[1]. Hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with latent states and can be presented as the simplest Bayesian network. We show our approach in modeling the problem as a HMM and using Baum-Welch Expectation Maximization technique and Spectral learning techniques to learn the parameters of this graphical model and make predictions. Further, model this problem as a Probabilistic Weighted Finite Automata using spectral methods. The predictions generated from our approach stands third among all the participants on the leader-board for the problems attempted.

1 Introduction

The Sequence PredictIction ChallengE (SPiCe) is a competition where the aim is to learn a model that predicts the next symbol of a given prefix as a ranked order of five most probable next symbols. Training datasets consist of variable length sequences with a fixed number of symbols. The competition uses real-world data from different fields (Natural Language Processing, Biology, Signal Processing, Software Verification, etc.) and synthetic data especially created for this challenge. The next five symbols which we submit is scored on a ranking metric based on normalized discounted cumulative gain (NDCG)

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 $P(.|y_i)$, i.e. the target probability distribution of possible next symbols given the prefix y_i. The NDCG is given by

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

where $p_1 \geq p_2 \geq p_3 \geq p_4 \geq p_5$ are the top 5 values in the distribution $P(.|y_i)$

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. Alternatively, we can also model this as a simple probabilistic weighted automata whose weights have to be learned.

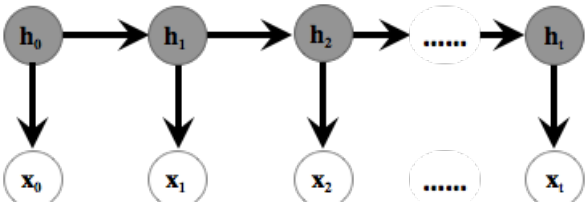

Figure 1: A hidden Markov model, h_i are the hidden variables and x_i are the observables.

2 Methodology

We will present three approaches that we explored in this paper. We modeled this problem as an HMM. First, we learned the parameters of the model from the training data sequences using BaumWelch algorithm and then using Spectral methods. We also tried to fit the data onto higher order Hidden Markov Models – with feedback edges, skip edges – but found the improvement in accuracy to be negligible, and in some cases non-existent. Finally, we fit

[1]Webpage: http://spice.lif.univ-mrs.fr/

the data into a probabilistic finite automaton using spectral techniques. We discuss each of the three approaches in the subsequent sections.

2.1 Notation

For the Hidden Markov model, we define the following notations

- $h_1, h_2, ..., h_t$ are a sequence of discrete hidden states. Let the number of possible hidden states that h_t can represent be m
- $x_1, x_2, ..., x_t$ are a sequence of discrete observations. Let the number of possible observable states that a value x_t can represent be n
- The state transitions is described by T, which is a homogeneous time independent stochastic transition matrix.

$$T = \{a_{ij}\} = P(h_t = j | h_{t-1} = i)$$

- The emission of each of the states is described by O

$$O = \{b_j(x_t)\} = P(x_t | h_t = j)$$

- The initial state distribution is given by

$$\vec{\pi}_i = P(h_1 = i)$$

3 Baum-Welch Algorithm

The BaumWelch algorithm is used to find the the maximum likelihood estimate of unknown parameters of a hidden Markov model (HMM) given a set of observed feature vectors using the Expectation-Maximization (EM) algorithm. We assume the $P(h_t|h_{t-1})$ is independent of time t.

A single observable sequence is given by

$$X = (X_1 = x_1, X_2 = x_2, X_3 = x_3, .., X_L = x_L)$$

A Hidden Markov Model is completely parameterized by $\theta = (A, B, \vec{\pi})$. The BaumWelch algorithm finds a local maximum $\theta^* = argmax_\theta P(X|\theta)$ – the HMM parameters θ that maximizes the probability of the given observation sequence.

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

3.1 Forward Procedure

As discussed in [1], let $\alpha_i(t) = P(x_1, x_2, ..., x_t, h_t = i|\theta)$, which is the probability of seeing the output sequence $x_1, x_2, ..., x_t$ and being in state i at time t. This is found recursively by the following equations:

$$\alpha_i(1) = \vec{\pi}_i b_i(x_1)$$
$$\alpha_j(t+1) = b_j(x_{t+1}) \sum_{i=1}^{N} \alpha_i(t) a_{ij}$$

3.2 Backward Procedure

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

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

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. We addressed this problem by using a normalized version of the Forward-Backward procedure. We discuss this in detail in the implementation section.

3.3 Updates

Define $\gamma_i(t)$ as the probability of being in a state i at time t given the observed sequence X and the parameters θ. As detailed in [2], it can be shown that

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

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 X and parameters θ. This can be shown to be equal to

$$\xi_{ij}(t) = P(h_t = i, h_{t+1} = j|X, \theta)$$
$$\xi_{ij}(t) = \frac{\alpha_i(t) a_{ij} \beta_j(t+1) b_j(x_{t+1})}{\sum_{k=1}^{N} \alpha_k(T)}$$

θ is updated with the following set of equations:

$$\vec{\pi}_i^* = \gamma_i(1)$$

which is the probability of state i at time 1.

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

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.

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

$b_i^*(v_k)$ 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.

4 Spectral Learning of Hidden Markov Models

The basic idea of this method is to discover the relationship between the observed states and the hidden states by spectral/Singular Value Decomposition methods that correlate the previous observations of a sequence to the future observations. When applied to learning HMM under a certain natural separation condition (a spectral condition), the algorithm is efficient in learning the model, without explicitly recovering the transition and the emission matrices. Drawing from the idea that we can represent the probability of sequences as a product of matrix operators, we define

$$A_x = T.diag(O_{x,1}, O_{x,2}, ..., O_{x,m}) \ \ \forall x \in [1, n]$$

For any intermediate position t in a sequence, we have

$$P[x_1, x_2, ..., x_t] = \vec{1}_m^T A_{xt} ... A_{x1} \vec{\pi}$$

where $\vec{1}_m$ is a vector of all ones.

As explained in [5](a seminal work in this domain), the algorithm requires that the HMM obeys the rank condition:

$$\vec{\pi} > 0 \quad elementwise, \ and \ O \ and \ T \ are \ rank \ m$$

Under the above condition, the HMMs allow an efficiently learn-able parameterization that depends only on the observables. The representation of the HMM can be defined in terms of the following vectors and matrices:

$$\begin{aligned} [P_1]_i &= P[x_1 = i] \\ [P_{2,1}]_{i,j} &= P[x_2 = i, x_1 = j] \\ [P_{3,x,1}]_{i,j} &= P[x_3 = i, x_2 = x, x_1 = j] \quad \forall x \in [n] \end{aligned}$$

where $P_1 \in \mathbb{R}^n, P_{2,1} \in \mathbb{R}^{n \times n}, P_{3,x,1} \in \mathbb{R}^{n \times n}$ are the marginal probabilities of the observation sequence singletons, pairs, and triplets.

Performing a singular value decomposition on $P_{2,1}$, we obtain $U \in \mathbb{R}^{n \times m}$, $\Sigma \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{m \times n}$

The observable representation of the HMM can then be defined as follows:

$$\begin{aligned} \vec{b}_1 &= U^T P_1 \\ \vec{b}_\infty &= (P_{2,1}^T U)^+ P_1 \\ B_x &= (U^T P_{3,x,1})(U^T P_{2,1})^+ \quad \forall x \in [n] \end{aligned}$$

When $P_{2,1}$ is rank m,

$$B_x = G A_x G^{-1}$$

where $G \in \mathbb{R}^{m \times m}$ is invertible, we have

$$\begin{aligned} B_{t:1} &= B_{x_t} B_{x_{t-1}} ... B_{x_1} \\ &= G A_t G^{-1} \, G A_{t-1} G^{-1} \, ... \, G A_1 G^{-1} \\ &= G A_t A_{t-1} \, ... \, A_1 G^{-1} \end{aligned}$$

Setting

$$\begin{aligned} \vec{b}_\infty &= \vec{1}^T G^{-1} \\ \vec{b}_1 &= G \vec{\pi} \end{aligned}$$

we have,

$$P[x_1, x_2, ..., x_t] = \vec{b}_\infty B_{x_{t:1}} \vec{b}_1$$

We can obtain empirical estimates of $P_1, P_{2,1}, and \ P_{3,x,1}$ from the training data as follows:

$$\begin{aligned} [\hat{P_{2,1}}]_{i,j} &= \frac{count(x_2 = i, x_1 = j)}{N} \\ [\hat{P_{3,x,1}}]_{i,j} &= \frac{count(x_3 = i, x_2 = x, x_1 = j)}{N} \end{aligned}$$

and uses these in place of the exact matrices which would have been obtained from the true model. This results in a small error ϵ that is bounded.

Having represented the model as above, given a sequence prefix $x_1, x_2, ..., x_{t-1}$, we can calculate the conditional observation probability of the next element in the sequence as

$$P[x_t | x_{1:t-1}] = \frac{\vec{b}_\infty B_{x_t} \vec{b}_t}{\sum_x \vec{b}_\infty B_x \vec{b}_t}$$

where $\vec{b}_t$ can be recursively calculated as

$$\vec{b}_t = \frac{B_{x_t} \vec{b}_{t-1}}{\vec{b}_\infty B_{x_{t-1}} \vec{b}_{t-1}}$$

5 Spectral Learning of Probabilistic Weighted Finite Automata

While spectral algorithm for learning HMM significantly improved the learning time of the model, it did little to

improve the accuracy. We now discuss spectral learning of a probabilistic weighted finite automata as discussed in [10]– the techniques of which were derived from the seminal work [5] whose algorithm was discussed in the previous section.

Let Σ be a finite alphabet and σ to denote an arbitrary symbol in Σ. The set of all finite strings over Σ is denoted by Σ^*, where we write λ for the empty string. Let $f : \Sigma^* \rightarrow \mathbb{R}$ be a function over strings. The Hankel matrix of f is a bi-infinite matrix $H_f \in \mathbb{R}^{\Sigma^* \mathrm{x} \Sigma^*}$ whose entries are defined as $H_f(u, v) = f(uv)$ for any $u, v \in \Sigma^*$.

Define finite sub-blocks of the bi-infinite Hankel Matrix using a basis $B = (P, S)$, where $P \subseteq \Sigma^*$ is a set of prefixes and $S \subseteq \Sigma^*$ a set of suffixes. Let $p = |P|$ and $s = |S|$, then the sub-block of H_f defined by B is the $p\mathrm{x}s$ matrix $H_B \in \mathbb{R}^{\mathrm{PxS}}$ with $H_B(u, v) = H_f(u, v) = f(uv)$ for any $u \in P$ and $v \in S$. Let λ denote the empty string. Let $\Sigma' = \Sigma \cup \lambda$. The prefix-closure of basis B is the basis $B' = (P', S)$, where $P' = P\Sigma'$. A basis $B = (P, S)$ is said to be p-closed if $P = P'\Sigma'$ for some P' called the root of P. A Hankel matrix over a p-closed basis can be partitioned into $|\Sigma| + 1$ blocks of the same size. For any $\sigma \in \Sigma'$, H_σ is used to denote the sub-block of H_f over the basis $(P\sigma, S)$. The sub-block $H_\sigma \in \mathbb{R}^{\mathrm{P}\sigma\mathrm{xS}}$ of H_f is the $p\ \ s$ matrix defined by $H_\sigma(u, v) = H_f(u\sigma, v)$. Thus, if B' is the prefix-closure of B, then for a particular ordering of the strings in P'

$$H_{B'}^T = \left[H_\lambda^T | H_{\sigma 1}^T | ... | H_{\sigma_{|\Sigma|}}^T \right]$$

The rank of a function $f : \Sigma^* \rightarrow \mathbb{R}$ is defined as the rank of its Hankel matrix, $rank(f) = rank(H_f)$. A basis $B = (P, S)$ is complete for f if the sub-block H_B has full rank and we say that H_B is a complete sub-block of H_f. The rank of f is related to the number of states needed to compute f with a weighted automaton, and the prefix-closure of a complete sub-block of H_f contains enough information to compute this automaton.

To learn an automata realizing an approximation of a function $f : \Sigma^* \rightarrow \mathbb{R}$ using a spectral algorithm, we will need to compute an estimate of a sub-block of the Hankel matrix H_f. In general such sub-blocks may be hard to obtain. However, in the case when f computes a probability distribution over Σ^* and we have access to a sample of i.i.d. examples from this distribution, estimates of subblocks of H_f can be obtained efficiently. It can be shown that if $H = PS$ is a rank factorization, then the weighted finite automata $A = (\alpha_1, \alpha_\infty, (A_\sigma))$ is minimal for f for

$$\alpha 1^T = h_{\lambda,S}^T S^T$$
$$\alpha_\infty = P^T h_{P,\lambda}$$
$$A_\sigma = P^T H_\sigma S^T$$

The spectral method is an efficient algorithm that implements the ideas of this result to find a rank factorization of a complete sub-block H of H_f and obtain from it a minimal weighted finite automata for f. Suppose $f : \Sigma^* \rightarrow \mathbb{R}$ is an unknown function of finite rank n and we want to compute a minimal weighted automata for it. Assume that $B = (P, S)$ is a complete basis for f. The algorithm receives as input: the basis B and the values of f on a set of strings W. The algorithm only needs a rank factorization of H to be able to apply the formulas given in above result. The compact SVD of a $p\mathrm{x}s$ matrix H_λ of rank n is given by the expression $H_\lambda = U\Lambda V^T$, where $U \in \mathbb{R}^{\mathrm{p}\times\mathrm{n}}, \mathrm{V} \in \mathbb{R}^{\mathrm{s}\times\mathrm{n}}$ are orthogonal matrices, and $\Lambda \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ is a diagonal matrix containing the singular values of H_λ. The factorization is equivalent to $H_\lambda = (H_\lambda V)V^T$.

With this factorization, equations from from above results are written as:

$$\alpha_1^T = h_{\lambda,S}^T$$
$$\alpha_\infty = (H_\lambda V)^T h_{P,\lambda}$$
$$A_\sigma = (H_\lambda V)^T H_\sigma V$$

These equations define the spectral learning algorithm. Forward and backward (empirical) probabilities for a probabilistic weighted finite automata can be recovered by computing an SVD on (empirical) string probabilities. Though state probabilities are non-observable, they can be recovered from observable quantities.

6 Implementation

The Baum-Welch algorithm was written in Java language. Python was initially used but dropped since it had extremely slow performance[2]. The following points describe the sequence of optimizations we performed.

- We discovered our implementation of forward-backward algorithm to very slow, especially when the number of hidden states was high. We then tried using HMMLearn[3] – a python based library which consisted algorithms for unsupervised learning and inference of Hidden Markov Models – in the hopes of making use of optimized code to gain efficiency. 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.

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

[3]HMMLearn source: https://github.com/hmmlearn/hmmlearn

- 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.

Spectral Learning algorithm for HMM requires the rank of P_{21} to be m. Since the training data samples have an indicator state for end of sequence, the last row of the matrix was all zeroes as there is no transition from that state to any other state. We addressed this by treating the rank as a hyper-parameter and tweaking this to obtain better results. We used the rank as a hyperparameter while learning the WFA using spectral technique.

7 Results

Table 1 describes the problem instances and their sizes in the competition.

Table 1: Problem sizes

Problem	#sequences	#symbols	Test-Size
0	20000	4	1000
1	20000	20	5000
2	20000	10	5000
3	20000	10	5000
4	5987	33	748

7.1 Baum-Welch algorithm

Table 2 summarizes the results obtained by Baum-Welch Algorithm on problem instances. The number of hidden states used were one plus the number of problem symbols(one symbol for the end character). They were run on mini-batches of size 15 and 25 which produced similar accuracy. For smaller problems like problem 0 having only 4 symbols, we tried with hidden states upto hundred, in general the accuracy improved with the increase in number of hidden states. However we could not repeat the experiment for problems having more symbols since it took a long time to train on a problem. This was due to the probability matrices taking a long time to converge.

Table 2: Baum-Welch performance

Problem	Score	#hidden-states	Training Time(s)
0	0.69	5	43
0	0.92	100	189
1	0.33	21	11665
1	0.61	100	187020
2	0.51	11	33403
3	0.49	11	39884
4	0.19	34	255546

This algorithm takes a very long time to train on problems when the number of symbols is large (more than 4 days) and performs poorly on those instances.

7.2 Spectral learning of HMM

The training time for this method was under 1 second for all the problems, which is a significant boost in training time over the previous approach. The results obtained are comparable, and in some cases better than, the ones obtained by inference using Baum-Welch algorithm. Table 3 gives a summary of the performance.

Table 3: Spectral Learning of HMM

Problem	Score
0	0.87
1	0.59
2	0.49
3	0.46
4	0.36

7.3 Spectral Learning of Weighted Finite Automata

This method performed really well and the training time is under 100 seconds for all instances. Table 4 gives a summary of the results obtained.

Table 4: Spectral Learning of WFA

Problem	Score	Hankel Rank
0	0.98	24
1	0.87	21
2	0.87	21
3	0.82	20
4	0.53	21

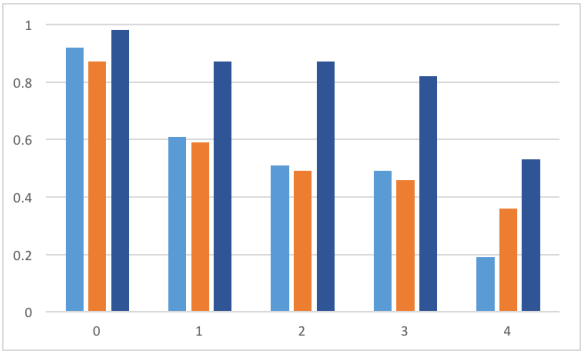

Figure 2: Comparison of the above approaches (NDCG v/s Problem Number)

7.4 Current standing in competition

We registered in the competition with the name "codeBlue". At the time of writing this paper, our rank on the competition is 3 out of over 70 registered participants for the above subset of problems. The following are the top 6 teams from the leader-board.

Table 5: Leaderboard

Rank	Team	Score
1	ushitora	5.81
2	vha	5.16
3	codeBlue	4.75
4	ToBeWhatYouWhatToBe	4.62
5	JGR	3.20
6	uwtacoma	1.93

8 Conclusion

In this paper, we presented a graphical models approach to predicting a discrete-time discrete-symbol sequence by modeling the problem as a Hidden Markov Model as well as a Probabilistic Weighted Automata. We used Baum-Welch and Spectral algorithm to learn HMM, and spectral techniques to construct the automata. Ad detailed in the results section, we found the training time to be significantly less with spectral techniques. Modeling the problem as a weighted automata yielded better results than HMM.

References

[1] Rabiner, L.R.; Juang, B.H. "An introduction to hidden markov models." *In IEEE ASSP Magazine, 1986. pp. 416.*

[2] Rabiner, L. R.. "A tutorial on hidden Markov models and selected applications in speech recognition". *In the Proceedings of IEEE, 1989. Vol. 77 Iss. 2, pp. 257-286.*

[3] Bishop, C. M. "Pattern Recognition and Machine Learning". *New York: Springer, 2006. Print.*

[4] Murphy, K. P. "Machine Learning: a Probabilistic Perspective". *MIT Press, 2012. Print.*

[5] Hsu, D.; Kakade, S.M.; Zhang, T. "A Spectra Algorithm for Learning Hidden Markov Models" *In COLT, 2009*

[6] Hsu, D.; Kakade, S.M. "Learning mixtures of spherical gaussians: moment methods and spectral decompositions" *In the Proceedings of ITCS, 2013. pp. 11-20*

[7] Achlioptas, D.; McSherry, F. "On spectral learning of mixtures of distributions." *In COLT, 2005*

[8] Kannan, R.; Salmasian, H.; Vempala, S. "The spectral method for general mixture models." *In COLT, 2005.*

[9] Lindsay, B.G. "Moment matrices: applications in mixtures." *Annals of Statistics, 1989. Vol. 17, Iss. 2, pp. 722740*

[10] Balle, B.; Carreras, X.; Luque, F.M.; Quattoni, A. "Spectral Learning of Weighted Automata A Forward-Backward Perspective" *In Springer Journal on Machine Learning, 2014. Vol. 96, Iss. 1, pp. 33-63*