2009 International Conference on Machine Learning and Applications

An Incremental Model Selection Algorithm Based on Cross-Validation for Finding the Architecture of a Hidden Markov Model on Hand Gesture Data Sets

Aydın Ulaş
Department of Computer Engineering
Boğaziçi University
Istanbul, Turkey
Email: ulasmehm@boun.edu.tr

Olcay Taner Yıldız
Department of Computer Engineering
Işık University
Istanbul, Turkey
Email: olcaytaner@isikun.edu.tr

***Abstract*—In a multi-parameter learning problem, besides choosing the architecture of the learner, there is the problem of finding the optimal parameters to get maximum performance. When the number of parameters to be tuned increases, it becomes infeasible to try all the parameter sets, hence we need an automatic mechanism to find the optimum parameter setting using computationally feasible algorithms. In this paper, we define the problem of optimizing the architecture of a Hidden Markov Model (HMM) as a state space search and propose the MSUMO (Model Selection Using Multiple Operators) framework that incrementally modifies the structure and checks for improvement using cross-validation. There are five variants that use forward/backward search, single/multiple operators, and depth-first/breadth-first search. On four hand gesture data sets, we compare the performance of MSUMO with the optimal parameter set found by exhaustive search in terms of expected error and computational complexity.**

***Keywords*-Hidden Markov model; model selection; cross-validation**

I. Introduction

Every classification algorithm has an assumption about the data and has different structures and parameters to tune for achieving best performance. Since number of parameters to be trained for optimum performance is large, we need a systematic and computationally feasible way to try and optimize different structure and parameter options.

For a hidden markov model (HMM), the model is defined by the graphical model structure such as the number of hidden states, the number of mixture components in these hidden states (if the node has a mixture model) and the connectivity between the hidden states, and once a model structure is fixed, Baum-Welch algorithm [1] learns the state transition, observation and initial state probabilities in the network. An introductory tutorial on Hidden Markov Models can be found in [1] and [2]. In this paper, we use HMM with a mixture of Gaussians at each hidden state on three different model structures namely left-right model (lr), left-right-loop model (lrl) and full model (full). In a previous work [3], we proposed the MOST algorithm (Multiple Operators using Statistical Tests) and showed its application on multilayer perceptrons (MLP).

The optimal architecture of an HMM is a graphical model which is large enough to learn the underlying function and is small enough to generalize well. A smaller model cannot learn the problem well, whereas a model larger than the optimum overlearns and does not generalize well and has a more complex architecture. The key is the trade-off between bias and variance [4]: A small and/or simple model underfits and fails to learn the data (bias is high and variance is low). A large and/or complex model overfits; it learns the data (bias is low) but also learns the noise (variance is high). The optimal architecture is the one with low bias and low variance so that the model learns the function underlying the data and not the added noise.

Increasing number of data mining applications and machine learning tools used in the industry increased the number of end users, most of who are not experts and lack the knowledge of the algorithms and the models used. They present the problem and the data at hand expecting an acceptable solution to be automatically found by the tool. Specifically for the case of HMM, it is hard to estimate the number of hidden states or function in each hidden state even for an expert user. There is therefore need for a methodology to optimize the model structure in a manner invisible to the end user, which, though may not necessarily return the optimal, returns a good enough structure in reasonable time. HMM's are widely used in Bioinformatics (as secondary structure predictors [5]) and Speech Processing (as audio modeling tools [6]) communities for a long time.

Biem et. al. [7] proposed a Bayesian based model selection criterion for HMM topology optimization. In the usual Bayesian Information Criterion (BIC), one uses multivariate normal distribution for the priors for each parameter of the HMM model. On the other hand, Biem et. al. [7] model each parameter with a different distribution. They compared proposed algorithm to BIC on an online recognition of hand-written characters task. In another work of the same author [8], Discriminative Information Criterion (DIC) was proposed for model selection in HMM. DIC selects the HMM model for each class that maximizes the difference between the loglikelihood of that model against the average

978-0-7695-3926-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ICMLA.2009.91

log likelihood of all models of other classes. Yet another criterion is Predictive Information Criterion (PIC) [9], which is used to choose the best tree structured HMM model by a top-down prior/posterior propagation algorithm.

In the next section, we present and discuss our proposed MSUMO algorithm. We present the experimental setup and results in Section III where we compare in detail five different variants of MSUMO on four hand gesture data sets. Section IV gives the conclusions and discusses possible future directions.

II. MSUMO: A Meta Learning Algorithm for Architecture Selection

A. Structure Learning as State Space Search

Optimizing the architecture of an HMM is a search in the state space of all architectures of HMM. In this work, when the model structure is fixed (left-right model, left-right-loop model and full model), the search space contains all the combinations of number of hidden states and number of mixtures at each hidden state. When the number of parameters to optimize increase, it becomes infeasible to try and evaluate all possible architectures for selecting the best one, therefore a heuristic strategy is needed to find a near-optimal solution by visiting as few as possible states in the search space. In a parameter setting where parameters are continuous, number of different architectures is infinite.

We define operators that modify the HMM structure and allow moving from one state to another. For example, the operator ADD-1 adds one hidden state and applying this operator takes us from state $HMM_{4,1}$ to $HMM_{5,1}$, that is from an HMM with four hidden states to one with five states. Therefore, we consider incremental algorithms as implementing *forward search* which starts from a simple initial state and use ADD operators. On the other hand, decremental/pruning algorithms implement *backward search* by starting from a complex state and using REMOVE operators to prune state(s)/mixture(s). *Floating search* allows using both ADD and REMOVE operators.

After applying an operator, the state evaluation function compares the performance metric of the next state with that of the current state according to the selection criteria and accepts/rejects the operator depending on whether the performance metric is improved or not. This state evaluation function trains and validates the HMM corresponding to the next state and uses a model selection criterion taking into account the generalization error and a measure of complexity, so that architectures that are accurate and simple are favored.

As in our previous work [3], when we define the problem of finding the optimal architecture as a state space search, we have five choices:

1) *Initial state.* If we select an HMM with a mixture containing a single Gaussian and a single hidden state as the initial state, the algorithm is mainly incremental and adds hidden states and/or mixtures to the architecture. If we start from an HMM with N hidden states ($HMM_{N,1}$), with large N, the algorithm is decremantal and reduces the number of states and Gaussians in the mixture.
2) *State transition operators.* Operators add states/mixtures or remove them. For faster convergence, we may also use operators that make longer jumps in the state space by adding/removing multiple units, e.g., ADD-5 adds five hidden states, ADD-1 adds only a single state. In this work, we do not use multiple jumps.
3) *Search beam.* We can have a single operator which gives us a single candidate architecture or we can apply multiple operators to get multiple candidates. In the case of multiple operators, breadth-first, depth-first, best-first search, or any variant thereof can be used.
4) *State evaluation function.* One can use cross-validation and an associated statistical test, or some other model selection criterion based on information theoretic metrics, such as Akaike's information criterion (AIC), minimum description length (MDL), Bayesian information criterion (BIC), or structural risk minimization (SRM) [4].
5) *Termination condition.* A trivial condition is to stop the search when no candidate improves on the current best. Another possibility is to stop when the error falls below a certain level, or when a fixed number of iterations are made.

B. MSUMO Operators

MSUMO supports the following operators (ordered by their effect on the complexity in increasing order):

1) REMOVE-1: Remove a single hidden state from the HMM.
2) ADD-1: Add a single hidden state to the HMM.
3) REMOVE-L: Add a new Gaussian to the mixture.
4) ADD-L: Remove a Gaussian from the mixture.

The above operators are selected to enable MSUMO to search widely in the search space. Operators REMOVE-1 and ADD-1 fine-tune a small neighborhood. To create large jumps in the search space, we could use REMOVE-n and ADD-n operators, but we did not include those experiments in this work. With ADD-L, a new Gaussian is to be added to the architecture.

C. The Role of Accuracy and Complexity: The MultiTest Algorithm

We define the complexity of an HMM by the product of the number of hidden states and number of Gaussians. In case of ties, we say that the architecture with more number of hidden states is more complex because that increases the

number of connections of the HMM architecture also. Other complexity measures, such as the number of connections, can also be used.

At start of MSUMO, initial architecture is selected as current BEST. To generate candidate models, C_i, operators are applied ordered by their complexities, and that is why candidate models are sorted. After generation, a candidate model is first trained and validated over cross-validation folds and its expected validation error is compared with that of the current best using a one-sided statistical test such as k-fold paired t test.

The idea is to keep the architecture simple, unless the additional complexity is justified by the significant decrease in error, and in applying the test, we take complexity into account as follows [10]: When comparing two models i and j where i is simpler than j, we test if model i has an expected error rate less than or equal to the expected error of model j:

$$H_0 : \mu_i \leq \mu_j \text{ vs. } H_1 : \mu_i > \mu_j$$

We set a prior preference of i over j because it is simpler. By assuming a prior ordering, we would like to test whether it is supported by the data, the hypothesis follows the prior and is one-sided. If the test does not reject, we favor i: Either $\mu_i < \mu_j$, that is, the simpler model indeed has less error and we choose it because it is more accurate; or, $\mu_i = \mu_j$ and we prefer the simpler model. Model j is favored only if the test rejects, i.e., when the additional complexity is justified by the significant decrease in error and the test (data) overrides our prior preference. That is, accuracy is checked first and given equal accuracies, the simpler model is favored.

Hence, in the case of MSUMO, if candidate C_i is more complex than the current best, it must have significantly less error in order to replace the best model. Similarly, if C_i is simpler than the current best, it replaces the best unless it has significantly larger error. When the best model changes, the algorithm continues by generating new candidates. If the best cannot be replaced, the algorithm terminates and returns the current best.

If at any state multiple operators are to be applied and we want to do breadth-first search, we generate *all* the next states and need to choose the best among all new candidates and the current best. Choosing the best of two models taking into account expected error and complexity by using a statistical test can be generalized to choosing the best of an arbitrary number of models using the *MultiTest algorithm* [10].

To find the best of K models, MultiTest makes $K(K-1)/2$ tests, and in order to have a confidence level of $(1-\alpha)$ for the final best model, the confidence level of each one-sided test should be corrected. The two correction methods are Bonferroni and Holm.

D. Five MSUMO Variants

Different search algorithms can be obtained by changing the choices in the MSUMO framework. One can produce an incremental or decremantal algorithm by changing the initial state, operator set, and the order of trying the operators. Changing the search variant affects the performance of the solution architecture as well as the complexity of search until a solution is found. We investigate the following five MSUMO variants (Fig. 1):

1) One-step forward (1-FW): Starts with single state, single Gaussian and uses ADD-1 until there is no improvement.
2) One-step backward (1-BW): Starts with $HMM_{10,1}$ and uses REMOVE-1 until no improvement.
3) Forward MSUMO (FW): Starts with single state single Gaussian and applies all operators above in increasing order of complexity in a depth-first manner, starting with the simplest one.
4) Backward MSUMO (BW): Starts with $HMM_{10,5}$ and applies all operators plus an operator for state removal in decreasing order of complexity in a depth-first manner, starting with the most complex.
5) Forward MSUMO with MultiTest (MFW): Starts with single state single Gaussian and instead of applying one by one, in a breadth-first manner, applies all operators at once and chooses the best among these new candidates and the current best using MultiTest. (i.e. If there are m operators, MultiTest chooses the best of $m+1$, that is, m new candidates and the current best).

1-FW and 1-BW are the basic incremental and decremental algorithms. FW and BW allow multiple operators and use floating search, that is, allow both additions and removals. MFW does breadth-first search evaluating all candidates at any intermediate state. FW, MFW and 1-FW can be viewed as incremental and BW, 1-BW as decremental. It is important to note that MSUMO framework is general and by using different initial states and operator sets, one can achieve a spectrum of MSUMO variants.

III. EXPERIMENTS

A. Experimental Factors and Evaluation Criteria

In the following experiments, we investigate the following factors:

- *MSUMO variant used in search.* These are 1-FW, 1-BW, FW, BW, and MFW, implementing forward vs. backward search, using single vs multiple operators, and searching depth-first vs breadth-first.
- *Statistical test used in comparison.* We use one parametric test namely k-fold paired t test. One could also use other parametric/nonparametric tests.

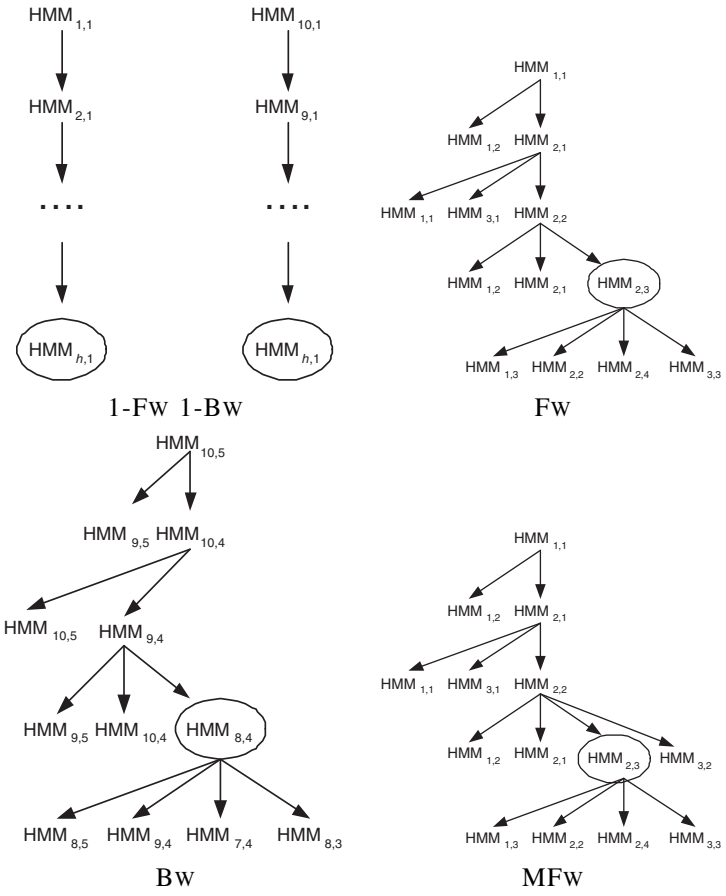

Figure 1. Example search paths of the five MSUMO variants. For Fw, MFw and Bw, at step $HMM_{i,j}$, four candidate architectures $(HMM_{(i-1),j}, HMM_{i,(j-1)}, HMM_{i,(j+1)}, HMM_{(i+1),j})$ are produced by applying operators. Note that in Fw, candidate networks are processed from simple to complex whereas the order is from complex to simple in Bw. The difference between MFw and Fw is that in MFw, instead of using an order and processing one by one, all candidate architectures are processed at once and the best is selected. The circled states are the final optimal states selected by the algorithm.

- *Confidence level* $(1-\alpha)$ *of the test.* We use 0.95, 0.99, 0.995, and 0.999 but report only the results of 0.95 in this work due to page restrictions.
- *Correction used when applying multiple tests.* We use Holm correction [11] (we omit Bonferroni correction due to lack of space) and compare with no correction and without using statistical tests.

HMMs are trained using HMM toolbox implemented by Kevin Murphy[1]. The retraining of the architecture, when an operator is applied, is done by completely re-initializing all the probabilities and retrain from scratch. No probabilities are kept or frozen.

The architectures found by MSUMO variants are compared with the optimal architecture. The optimal architecture of each data set is found by applying an exhaustive search over all HMMs with a single hidden state up to five Gaussians, and HMM with ten hidden states up to five Gaussians on each state. Hence, we have $10 \times 5 = 50$ different architectures to choose from. By using a statistical test, we compare each of these 50 architectures with the other 49 architectures and find the optimal architecture using MultiTest (MultiTest also gives an ordering of these models and we use this ordering to calculate the "distance" between an architecture found by MSUMO and the optimal architecture, as we see below).

We compare the architectures found by the five MSUMO variants in terms of three criteria:

1) The accuracy of the estimated architecture,
2) The complexity of the estimated architecture, and
3) Computational complexity of the search until an architecture is found.

To measure the goodness of an architecture found, we define the measure of *order*, which, found by MultiTest (for each data set), has the optimal architecture in position one, second best architecture in position two, and the worst architecture in the last position. Accuracy and complexity are both used to find this order by MultiTest: A low order

[1]Available at http://www.cs.ubc.ca/∼murphyk/Software/HMM/hmm.html

indicates that we either find an architecture that is close to the optimal architecture in terms of accuracy, or if the accuracies are comparable, we find an architecture that is simpler than an architecture with higher order.

Our second measure *rank* determines how fast we get to the final state, which uses the number of states visited as a measure of the complexity of search. It is important to keep this measure low because it corresponds to the number of architectures that should be trained and validated (over multiple folds). The architectures found by the algorithms are ordered first using MultiTest to determine their orders and then ranked by their proximity to the optimal architecture. The search complexity is then taken into account if two or more MSUMO variants find the same architecture. In that case, ranks replace the order and the one which visits less number of states takes a lower rank. If the number of states visited is also equal, the average of the ranks are given to each architecture. An example for calculating the ranks is given in Table I. A MSUMO variant with a low rank indicates that it finds an architecture that is close to the optimal architecture and in doing this, visits few states, when compared with another MSUMO variant with higher rank.

Table I
AN EXAMPLE FOR CALCULATION OF RANKS.

	FW	BW	MFW	1-FW	1-BW
Order	3	2	1	1	2
# states	10	5	3	4	5
Rank	5	3.5	1	2	3.5

B. Data Sets

The properties of the data sets can be seen in Table II. Features show the number of features per frame. More information on the feature extraction of IDIAP data set can be found in [12]. In eNTERFACE'06 data set, when applying the left-right model, we could train HMMs with maximum 5 states, so the selection is done using $5 \times 5 = 25$ architectures. In British Sign Language data set, when applying the left-right and left-right-loop models, we could train HMMs with maximum 8 states, so the selection is done using $8 \times 5 = 40$ architectures. In this study, we used four subsets of Australian data set, au80, au60, au40 and au20, containing the first 80, 60, 40 and 20 signs respectively in lexicographic order.

Table II
PROPERTIES OF DATA SETS.

	Abbrv	size	signs	features
IDIAP [13]	twoh	490	7	20
eNTERFACE'06 [14]	eface	760	19	32
British [15]	bsl	980	91	22
Australian [16]	au	6650	95	8

C. Results

We can see the *ranks* and *orders* using no statistical tests in Table III. The first numbers show the *ranks* and the numbers in the parentheses show the *order* and number of states visited by each algorithm respectively. Entries in bold face mean that the algorithm finds the optimal model according to MultiTest (*order* $= 1$). Although there are data sets where multiple algorithms find the optimal, their ranks differ because of the number of states they visited. The optimal (BEST) is found by applying MultiTest on all architectures, therefore for BEST, *order = 1*, and number of states visited is the total number of architectures. In this setup, when we look at the algorithms, we see that the best algorithm (according to *rank*) is BW. When we do not use any statistical tests, we compare two algorithms according to their average errors. The algorithm with lowest average error is chosen to be the next candidate model. Since in Australian data set, when number of mixtures and states increases, error decreases most of the time, the best algorithms generally have high number of mixtures and states. Hence, this gives the BW algorithm an advantage over other MSUMO variants, and in this case it's the best algorithm. The *order* of BW is also the best as expected. When we compare the number of states visited, we see that the best algorithm is 1-BW, although its rank is the worst. 1-FW has less number of states visited but is not as accurate as BW. The found architectures and expected error rates of each MSUMO variant can be seen in Table IV. The first number is the expected error rate and the numbers in parentheses show the number of states and number of mixtures of the architecture found by each algorithm. We can see that in Australian data sets, BW finds either the optimal or architectures which are close to the optimal.

We also see that BW, FW and MFW has the lowest orders (closer to the optimal), which clearly supports our application of operators on both directions.

We can see in Tables V and VI the results using no correction. As we can see, using these data sets and no correction, BW is still the best according to *rank* and *order*. Since we use a statistical test for comparing two algorithms, it gets harder for a complex algorithm to beat a simpler one, which explains the decrease in *rank* of BW compared to using no statistical tests. In this case, considering number of states visited, the forward algorithms are better than backward algorithms, 1-FW being the best. BW visits the most number of states. Order of BW is still the best and has 15 algorithms difference compared to the next best. We also see that error rates of forward algorithms increase rapidly when we apply the statistical tests.

We can see the results using Holm correction in Table VII and VIII. As we get more conservative on choosing between algorithms, we can see that forward algorithms are favored. In this setup, using Holm correction, we make the statistical

Table III
Ranks (*orders*, NUMBER OF STATES VISITED) OF MSUMO VARIANTS USING NO STATISTICAL TEST. ENTRIES IN BOLD FACE SHOW THAT THE ALGORITHM FINDS THE OPTIMAL MODEL ACCORDING TO MULTITEST.

	Struc	MFW	1-FW	1-BW	FW	BW
bsl	full	**2.5(1,3)**	**1.0(1,2)**	4.0(5,2)	**2.5(1,3)**	5.0(31,12)
bsl	lr	**2.5(1,3)**	**1.0(1,2)**	4.0(5,2)	**2.5(1,3)**	5.0(25,8)
bsl	lrl	**2.5(1,3)**	**1.0(1,2)**	5.0(21,4)	**2.5(1,3)**	4.0(3,17)
eface	full	4.5(43,3)	3.0(43,2)	2.0(18,2)	4.5(43,3)	1.0(10,5)
eface	lr	3.5(16,3)	2.0(16,2)	5.0(21,3)	3.5(16,3)	1.0(3,3)
eface	lrl	2.5(20,7)	5.0(46,2)	4.0(44,4)	2.5(20,7)	1.0(9,3)
twoh	full	3.0(35,9)	4.0(46,5)	5.0(48,2)	2.0(8,13)	1.0(2,3)
twoh	lr	2.5(6,12)	4.0(41,5)	5.0(47,3)	2.5(6,12)	1.0(2,5)
twoh	lrl	3.5(13,3)	2.0(13,2)	5.0(38,2)	3.5(13,3)	1.0(4,4)
au20	full	2.0(2,28)	5.0(48,3)	4.0(44,2)	3.0(16,14)	1.0(2,4)
au20	lr	3.0(16,15)	5.0(47,8)	4.0(42,2)	2.0(16,11)	**1.0(1,4)**
au20	lrl	3.0(4,10)	5.0(49,4)	4.0(45,2)	2.0(4,6)	**1.0(1,5)**
au40	full	**1.0(1,26)**	5.0(43,7)	4.0(42,2)	3.0(2,26)	2.0(2,3)
au40	lr	3.0(7,21)	5.0(44,10)	4.0(44,2)	2.0(7,18)	**1.0(1,3)**
au40	lrl	3.0(2,12)	5.0(47,4)	4.0(45,2)	2.0(2,9)	**1.0(1,5)**
au60	full	2.0(2,26)	5.0(42,10)	4.0(42,2)	3.0(3,24)	**1.0(1,3)**
au60	lr	**2.0(1,27)**	5.0(49,2)	4.0(34,3)	3.0(4,21)	**1.0(1,4)**
au60	lrl	3.0(4,12)	4.0(46,4)	5.0(47,2)	2.0(4,9)	1.0(2,5)
au80	full	**3.0(1,27)**	5.0(49,2)	4.0(41,3)	**2.0(1,25)**	**1.0(1,3)**
au80	lr	**3.0(1,27)**	5.0(49,2)	4.0(36,3)	**2.0(1,22)**	**1.0(1,4)**
au80	lrl	3.0(4,10)	4.0(46,4)	5.0(47,2)	2.0(4,6)	1.0(3,5)
	avg	2.8(9,14)	3.9(37,4)	4.2(36,2)	2.6(8,12)	1.6(5,5)

Table IV
EXPECTED ERROR (NUMBER OF STATES, NUMBER OF MIXTURES) OF MSUMO VARIANTS USING NO STATISTICAL TEST.

	Struc	MFW	1-FW	1-BW	FW	BW	BEST
bsl	full	7.0(1,1)	7.0(1,1)	11.7(10,1)	7.0(1,1)	13.6(6,4)	7.0(1,1)
bsl	lr	7.0(1,1)	7.0(1,1)	10.2(8,1)	7.0(1,1)	12.9(4,5)	7.0(1,1)
bsl	lrl	9.0(1,1)	9.0(1,1)	9.5(6,1)	9.0(1,1)	9.5(6,1)	9.0(1,1)
eface	full	8.0(1,1)	8.0(1,1)	4.3(10,1)	8.0(1,1)	5.8(10,4)	4.3(10,1)
eface	lr	8.0(1,1)	8.0(1,1)	7.8(4,1)	8.0(1,1)	6.7(5,5)	6.5(4,4)
eface	lrl	6.6(2,2)	7.9(1,1)	5.4(8,1)	6.6(2,2)	6.2(10,5)	5.1(6,4)
twoh	full	2.7(2,3)	2.9(4,1)	2.0(10,1)	1.8(4,4)	1.4(10,5)	1.2(9,4)
twoh	lr	1.9(3,3)	2.2(4,1)	2.7(9,1)	1.8(3,3)	1.2(8,5)	1.0(6,5)
twoh	lrl	1.9(1,1)	1.8(1,1)	2.2(10,1)	1.8(1,1)	1.6(9,5)	1.4(1,4)
au20	full	34.1(9,5)	55.5(2,1)	42.0(10,1)	39.2(5,5)	34.1(9,5)	34.0(10,4)
au20	lr	39.5(3,5)	46.2(7,1)	43.9(10,1)	39.5(3,5)	33.1(9,5)	33.1(9,5)
au20	lrl	33.7(1,5)	46.4(3,1)	46.3(10,1)	33.7(1,5)	32.8(8,5)	32.8(8,5)
au40	full	44.0(8,5)	56.5(6,1)	53.5(10,1)	44.3(10,5)	44.3(10,5)	44.0(8,5)
au40	lr	45.1(7,5)	55.7(10,1)	55.7(10,1)	45.1(7,5)	42.8(10,5)	42.8(10,5)
au40	lrl	43.4(2,5)	55.4(3,1)	55.5(10,1)	43.4(2,5)	43.1(8,5)	43.1(8,5)
au60	full	47.6(8,5)	56.4(10,1)	56.3(10,1)	47.8(9,4)	47.1(10,5)	47.1(10,5)
au60	lr	46.4(9,5)	70.5(1,1)	59.0(9,1)	48.3(8,4)	46.4(9,5)	46.4(9,5)
au60	lrl	47.2(2,5)	60.0(3,1)	59.7(10,1)	47.2(2,5)	46.8(8,5)	46.7(4,5)
au80	full	52.9(10,5)	74.8(1,1)	61.7(9,1)	52.9(10,5)	52.9(10,5)	52.9(10,5)
au80	lr	52.6(9,5)	74.8(1,1)	64.8(9,1)	52.6(9,5)	52.6(9,5)	52.6(9,5)
au80	lrl	53.1(1,5)	65.0(3,1)	65.6(10,1)	53.1(1,5)	53.0(8,5)	52.5(4,5)

Table V
Ranks (*orders*, NUMBER OF STATES VISITED) OF MSUMO VARIANTS USING k-FOLD PAIRED t TEST WITH NO CORRECTION. ENTRIES IN BOLD FACE SHOW THAT THE ALGORITHM FINDS THE OPTIMAL MODEL ACCORDING TO MULTITEST.

	Struc	MFW	1-FW	1-BW	FW	BW
bsl	full	**2.5(1,3)**	**1.0(1,2)**	**4.0(1,10)**	**2.5(1,3)**	**5.0(1,22)**
bsl	lr	**2.5(1,3)**	**1.0(1,2)**	**4.0(1,8)**	**2.5(1,3)**	**5.0(1,20)**
bsl	lrl	**2.5(1,3)**	**1.0(1,2)**	5.0(10,6)	**2.5(1,3)**	**4.0(1,20)**
eface	full	3.5(40,3)	2.0(40,2)	5.0(40,10)	3.5(40,3)	1.0(9,20)
eface	lr	**2.5(1,3)**	**1.0(1,2)**	**4.0(1,5)**	**2.5(1,3)**	**5.0(1,15)**
eface	lrl	4.5(41,3)	3.0(41,2)	1.0(22,4)	4.5(41,3)	2.0(25,16)
twoh	full	3.5(50,3)	2.0(50,2)	5.0(50,10)	3.5(50,3)	1.0(15,18)
twoh	lr	4.5(48,3)	3.0(48,2)	2.0(6,8)	4.5(48,3)	1.0(4,7)
twoh	lrl	**2.5(1,3)**	**1.0(1,2)**	**4.0(1,10)**	**2.5(1,3)**	**5.0(1,24)**
au20	full	3.0(39,10)	5.0(48,2)	4.0(43,6)	2.0(21,12)	1.0(16,14)
au20	lr	3.0(42,7)	4.0(47,2)	5.0(50,5)	2.0(42,5)	1.0(28,14)
au20	lrl	2.0(24,7)	5.0(45,2)	4.0(41,7)	1.0(24,5)	3.0(24,21)
au40	full	3.0(18,15)	5.0(50,2)	4.0(43,6)	2.0(18,10)	1.0(4,5)
au40	lr	4.5(46,3)	3.0(46,2)	2.0(32,3)	4.5(46,3)	1.0(22,14)
au40	lrl	**2.0(1,9)**	4.0(41,2)	5.0(41,10)	**1.0(1,6)**	**3.0(1,18)**
au60	full	4.5(47,3)	3.0(47,2)	2.0(44,4)	4.5(47,3)	1.0(7,9)
au60	lr	4.5(45,3)	3.0(45,2)	2.0(32,3)	4.5(45,3)	1.0(17,10)
au60	lrl	**2.0(1,9)**	4.0(41,2)	5.0(41,10)	**1.0(1,6)**	**3.0(1,19)**
au80	full	4.5(47,3)	3.0(47,2)	2.0(38,3)	4.5(47,3)	1.0(12,9)
au80	lr	4.5(45,3)	3.0(45,2)	2.0(35,3)	4.5(45,3)	1.0(2,4)
au80	lrl	**2.0(1,10)**	4.0(41,2)	5.0(41,10)	**1.0(1,6)**	3.0(4,11)
	avg	3.2(26,5)	2.9(35,2)	3.6(29,7)	2.9(25,4)	2.3(9,15)

Table VI
EXPECTED ERROR (NUMBER OF STATES, NUMBER OF MIXTURES) OF MSUMO VARIANTS USING k-FOLD PAIRED t TEST WITH NO CORRECTION.

	Struc	MFW	1-FW	1-BW	FW	BW	BEST
bsl	full	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)
bsl	lr	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)
bsl	lrl	9.0(1,1)	9.0(1,1)	9.0(4,1)	9.0(1,1)	9.0(1,1)	9.0(1,1)
eface	full	8.0(1,1)	8.0(1,1)	8.0(1,1)	8.0(1,1)	6.3(2,3)	5.0(9,1)
eface	lr	8.0(1,1)	8.0(1,1)	8.0(1,1)	8.0(1,1)	8.0(1,1)	8.0(1,1)
eface	lrl	7.9(1,1)	7.9(1,1)	5.4(8,1)	7.9(1,1)	6.1(3,4)	6.6(3,1)
twoh	full	4.9(1,1)	4.9(1,1)	4.9(1,1)	4.9(1,1)	1.8(4,4)	2.8(3,1)
twoh	lr	4.9(1,1)	4.9(1,1)	2.2(4,1)	4.9(1,1)	1.0(6,5)	1.8(3,3)
twoh	lrl	1.8(1,1)	1.8(1,1)	1.8(1,1)	1.8(1,1)	1.8(1,1)	1.8(1,1)
au20	full	48.2(1,5)	57.3(1,1)	46.2(6,1)	41.7(3,4)	39.1(6,3)	35.6(8,4)
au20	lr	51.4(1,3)	57.3(1,1)	46.2(7,1)	51.4(1,3)	42.2(4,3)	36.0(6,5)
au20	lrl	36.5(1,3)	49.8(1,1)	46.5(5,1)	36.5(1,3)	36.5(1,3)	34.2(1,4)
au40	full	50.0(3,5)	67.2(1,1)	56.5(6,1)	50.0(3,5)	44.0(8,5)	46.0(7,4)
au40	lr	67.2(1,1)	67.2(1,1)	55.9(9,1)	67.2(1,1)	55.0(4,3)	44.3(8,4)
au40	lrl	43.9(1,4)	57.5(1,1)	57.5(1,1)	43.9(1,4)	43.9(1,4)	43.9(1,4)
au60	full	70.5(1,1)	70.5(1,1)	58.5(8,1)	70.5(1,1)	49.8(7,4)	47.8(9,4)
au60	lr	70.5(1,1)	70.5(1,1)	59.0(9,1)	70.5(1,1)	53.8(4,5)	48.3(8,4)
au60	lrl	47.5(1,4)	61.2(1,1)	61.2(1,1)	47.5(1,4)	47.5(1,4)	47.5(1,4)
au80	full	74.8(1,1)	74.8(1,1)	61.7(9,1)	74.8(1,1)	56.1(7,4)	54.1(9,4)
au80	lr	74.8(1,1)	74.8(1,1)	64.8(9,1)	74.8(1,1)	52.6(9,5)	53.9(10,4)
au80	lrl	53.1(1,5)	66.4(1,1)	66.4(1,1)	53.1(1,5)	52.5(4,5)	53.1(1,5)

test difficult to reject that the two algorithms have the same expected error, therefore forward algorithms which start from the simple $HMM_{1,1}$ state are the best according to *ranks* and number of states visited. In the data sets other than Australian data set, $HMM_{1,1}$ gives pretty accurate models so most of the time there is no statistically difference between $HMM_{1,1}$ and the other architectures. The same rule applies for the no correction case, but in that case since we do not use any correction on α, the algorithm may find statistical differences and find different architectures (though in these data sets there are only two examples of this case which are au20-full and au20-lr). When we use Holm correction, the tests become more conservative and do not find any statistical difference. So in this case, best MSUMO variant is 1-FW which achieves this using the least number of states visited. Although BW is the worst algorithm according to rank, we see that its *order* is still the best, so it again finds the algorithms closest to the optimal. We see that the found architectures and their error rates are almost identical when we use no correction or holm correction. The difference is the optimum algorithm, which in turn changes the *ranks* and *orders* as we see in Tables V and VII.

Table VII
Ranks (*orders*, NUMBER OF STATES VISITED) OF MSUMO VARIANTS USING k-FOLD PAIRED t TEST WITH HOLM CORRECTION. ENTRIES IN BOLD FACE SHOW THAT THE ALGORITHM FINDS THE OPTIMAL MODEL ACCORDING TO MULTITEST.

	Struc	MFW	1-FW	1-BW	FW	BW
bsl	full	**2.5(1,3)**	**1.0(1,2)**	**4.0(1,10)**	**2.5(1,3)**	**5.0(1,22)**
bsl	lr	**2.5(1,3)**	**1.0(1,2)**	**4.0(1,8)**	**2.5(1,3)**	**5.0(1,20)**
bsl	lrl	**2.5(1,3)**	**1.0(1,2)**	5.0(4,6)	**2.5(1,3)**	**4.0(1,20)**
eface	full	**2.5(1,3)**	**1.0(1,2)**	**4.0(1,10)**	**2.5(1,3)**	5.0(9,20)
eface	lr	**2.5(1,3)**	**1.0(1,2)**	**4.0(1,5)**	**2.5(1,3)**	**5.0(1,15)**
eface	lrl	**2.5(1,3)**	**1.0(1,2)**	4.0(8,4)	**2.5(1,3)**	5.0(23,16)
twoh	full	**2.5(1,3)**	**1.0(1,2)**	**4.0(1,10)**	**2.5(1,3)**	5.0(29,18)
twoh	lr	**2.5(1,3)**	**1.0(1,2)**	4.0(4,8)	**2.5(1,3)**	5.0(42,7)
twoh	lrl	**2.5(1,3)**	**1.0(1,2)**	**4.0(1,10)**	**2.5(1,3)**	**5.0(1,24)**
au20	full	5.0(48,5)	4.0(47,2)	3.0(44,6)	2.0(24,12)	1.0(3,14)
au20	lr	4.0(44,3)	3.0(44,2)	5.0(47,5)	2.0(39,5)	1.0(15,14)
au20	lrl	**2.0(1,7)**	4.0(38,2)	5.0(39,7)	**1.0(1,5)**	**3.0(1,21)**
au40	full	3.0(28,15)	5.0(50,2)	4.0(45,6)	2.0(28,10)	1.0(12,5)
au40	lr	4.5(50,3)	3.0(50,2)	2.0(36,3)	4.5(50,3)	1.0(19,14)
au40	lrl	2.0(2,9)	4.0(41,2)	5.0(41,10)	1.0(2,6)	3.0(2,18)
au60	full	4.5(49,3)	3.0(49,2)	2.0(47,4)	4.5(49,3)	1.0(5,9)
au60	lr	4.5(48,3)	3.0(48,2)	2.0(38,3)	4.5(48,3)	1.0(23,10)
au60	lrl	**2.0(1,9)**	4.0(37,2)	5.0(37,10)	**1.0(1,6)**	**3.0(1,19)**
au80	full	4.5(50,3)	3.0(50,2)	2.0(40,3)	4.5(50,3)	1.0(4,9)
au80	lr	4.5(49,3)	3.0(49,2)	2.0(32,3)	4.5(49,3)	1.0(11,4)
au80	lrl	**1.0(1,9)**	4.0(37,2)	5.0(37,10)	2.0(2,6)	3.0(11,11)
	avg	3.0(18,5)	2.5(26,2)	3.8(24,7)	2.7(17,4)	3.0(10,15)

Table VIII
EXPECTED ERROR (NUMBER OF STATES, NUMBER OF MIXTURES) OF MSUMO VARIANTS USING k-FOLD PAIRED t TEST WITH HOLM CORRECTION.

	Struc	MFW	1-FW	1-BW	FW	BW	BEST
bsl	full	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)
bsl	lr	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)	7.0(1,1)
bsl	lrl	9.0(1,1)	9.0(1,1)	9.0(4,1)	9.0(1,1)	9.0(1,1)	9.0(1,1)
eface	full	8.0(1,1)	8.0(1,1)	8.0(1,1)	8.0(1,1)	6.3(2,3)	8.0(1,1)
eface	lr	8.0(1,1)	8.0(1,1)	8.0(1,1)	8.0(1,1)	8.0(1,1)	8.0(1,1)
eface	lrl	7.9(1,1)	7.9(1,1)	5.4(8,1)	7.9(1,1)	6.1(3,4)	7.9(1,1)
twoh	full	4.9(1,1)	4.9(1,1)	4.9(1,1)	4.9(1,1)	1.8(4,4)	4.9(1,1)
twoh	lr	4.9(1,1)	4.9(1,1)	2.2(4,1)	4.9(1,1)	1.0(6,5)	4.9(1,1)
twoh	lrl	1.8(1,1)	1.8(1,1)	1.8(1,1)	1.8(1,1)	1.8(1,1)	1.8(1,1)
au20	full	55.6(1,2)	57.3(1,1)	46.2(6,1)	41.7(3,4)	39.1(6,3)	42.0(10,1)
au20	lr	57.3(1,1)	57.3(1,1)	46.2(7,1)	51.4(1,3)	42.2(4,3)	43.8(10,1)
au20	lrl	36.5(1,3)	49.8(1,1)	46.5(5,1)	36.5(1,3)	36.5(1,3)	36.5(1,3)
au40	full	50.0(3,5)	67.2(1,1)	56.5(6,1)	50.0(3,5)	44.0(8,5)	49.8(8,2)
au40	lr	67.2(1,1)	67.2(1,1)	55.9(9,1)	67.2(1,1)	55.0(4,3)	50.1(10,2)
au40	lrl	43.9(1,4)	57.5(1,1)	57.5(1,1)	43.9(1,4)	43.9(1,4)	46.8(1,3)
au60	full	70.5(1,1)	70.5(1,1)	58.5(8,1)	70.5(1,1)	49.8(7,4)	51.9(6,3)
au60	lr	70.5(1,1)	70.5(1,1)	59.0(9,1)	70.5(1,1)	53.8(4,5)	53.4(10,2)
au60	lrl	48.0(1,4)	61.2(1,1)	61.2(1,1)	48.0(1,4)	48.0(1,4)	48.0(1,4)
au80	full	74.8(1,1)	74.8(1,1)	61.7(9,1)	74.8(1,1)	56.1(7,4)	57.9(6,3)
au80	lr	74.8(1,1)	74.8(1,1)	64.8(9,1)	74.8(1,1)	52.6(9,5)	58.0(6,4)
au80	lrl	54.3(1,4)	66.4(1,1)	66.4(1,1)	53.1(1,5)	52.5(4,5)	54.3(1,4)

IV. DISCUSSION

We propose the MSUMO algorithm for model selection in HMM's. The algorithm has five variants which implement forward/backward search with single and multiple operands. We have seen in our experiments that MSUMO finds architectures close to the optimal architecture without comparing all algorithms and extracting all architectures exhaustively. MSUMO is a general framework and can be used with other statistical tests, corrections, search strategies. It searches the state space and tries to find a good algorithm in reasonable time.

We compared different HMM architectures using their expected error, complexity and distance to the optimal algorithm defined by the measures *rank* and *order*. We investigated the effects of using statistical tests and effect of correction as well as different search strategies, different HMM architectures and four data sets using five MSUMO variants.

We see that when the conservativity of the selection criteria increases (no test, no correction, holm correction), the algorithm tends to find simpler models and moving to other states in the search space becomes difficult. These are all design choices and it depends on the problem at hand which MSUMO variant to choose. If we need simpler models, we should use MSUMO with statistical tests and corrections. If only error is important, we should use MSUMO with no statistical tests. We see that in these data sets, BW algorithm finds the most accurate architectures, but with an increased number of visited states compared to other architectures.

As future work, we plan to use other corrections and change α to see its effect on the final architecture found. In this work, our jumps were only one step, we shall try jumps with multiple steps. In that case, we should decide on the number of states to jump at each state.

REFERENCES

[1] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," in *Proceedings of the IEEE*, ser. no. 2, vol. 77, February 1989, pp. 257–285.

[2] Z. Ghahramani, "An introduction to hidden markov models and bayesian networks," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 15, no. 1, pp. 9–42, 2001.

[3] O. Aran, O. T. Yıldız, and E. Alpaydın, "An incremental framework based on cross-validation for estimating the architecture of a multilayer perceptron," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 3, no. 2, pp. 1–38, 2009.

[4] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning*. Springer-Verlag, 2001.

[5] J. Martin, J.-F. Gibrat, and F. Rodolphe, "Choosing the optimal hidden markov model for secondary-structure prediction," *Intelligent Systems, IEEE*, vol. 20, no. 6, pp. 19–25, Nov.-Dec. 2005.

[6] M. Reyes-Gomez and D. Ellis, "Selection, parameter estimation, and discriminative training of hidden markov models for general audio modeling," in *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*, vol. 1, July 2003, pp. I–73–6 vol.1.

[7] A. Biem, J.-Y. Ha, and J. Subrahmonia, "A bayesian model selection criterion for hmm topology optimization," in *Acoustics, Speech, and Signal Processing, 2002. Proceedings. (ICASSP '02). IEEE International Conference on*, vol. 1, 2002, pp. I–989–I–992 vol.1.

[8] A. Biem, "A model selection criterion for classification: application to hmm topology optimization," in *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, Aug. 2003, pp. 104–108 vol.1.

[9] J.-T. Chien and S. Furui, "Predictive hidden markov model selection for speech recognition," *Speech and Audio Processing, IEEE Transactions on*, vol. 13, no. 3, pp. 377–387, May 2005.

[10] O. T. Yıldız and E. Alpaydın, "Ordering and finding the best of $K > 2$ supervised learning algorithms," *IEEE T. Pattern. Anal.*, vol. 28, no. 3, pp. 392–402, 2006.

[11] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, pp. 65–70, 1979.

[12] O. Aran and L. Akarun, "Recognizing two handed gestures with generative, discriminative and ensemble methods via Fisher kernels," in *Multimedia Content Representation, Classification and Security International Workshop, MRCS 2006, Istanbul, Turkey, Proceedings*, vol. LNCS 4015, 2006, pp. 159–166.

[13] S. Marcel and A. Just, "IDIAP Two handed gesture dataset," Available at http://www.idiap.ch/∼marcel/.

[14] O. Aran, I. Ari, A. Benoit, P. Campr, A. H. Carrillo, F.-X. Fanard, L. Akarun, A. Caplier, and B. Sankur, "Signtutor: An interactive system for sign language tutoring," *IEEE Multimedia*, vol. 16, no. 1, 2009.

[15] R. Bowden, "British sign language dataset." [Online]. Available: http://personal.ee.surrey.ac.uk/Personal/R.Bowden/sign/index.html

[16] A. Asuncion and D. J. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html