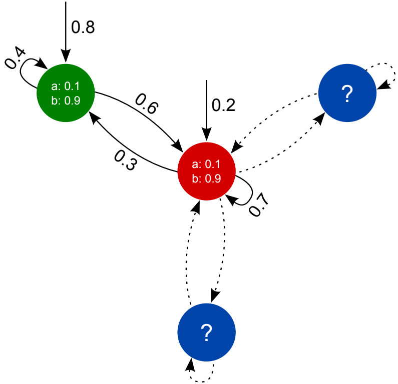
Learning Probabilistic Automata



Mathias M. Andersen
Kent M. Caspersen
Anders R. Nielsen
Juraj Kolčák
Guillaume Grippari
Theis Christensen

MODELS

$$n\ grams \approx Markov\ chains \leq DPFA \ldots \leq$$

$$\leq HMM \approx PFA \leq$$

$$\leq Multiplicity\ automata$$

PAutomaC

- 2012 online PFA learning competition.
- All results and training data published after competition finished.
- Large amount of data available artificially generated by several types of models (Markov chains, DPFA, PFA, HMM).
- The generator models vary in number of states/symbols and density of the transitions and emissions providing for a large scale of different data.

PROBLEM DEFINITION

"Based on the *PAutomaC* competition data, how can we learn not only the probabilistic parameters but the the structure of an HMM as well."

HIDDEN MARKOV MODELS

DEFINITION

Alphabet of observable symbols $\Sigma = \{\sigma_1, ..., \sigma_m\}$.
Set of hidden states $S = \{s_1, ..., s_n\}$.
Sequence of observable symbols $\mathbf{O} = (o_1, ..., o_T) = \Sigma^T$.
Sequence of hidden states $\mathbf{Q} = (q_1, ..., q_T) = S^T$.

$$\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$$

Transition matrix $\mathbf{A}$: $\forall s, r \in S : a_{sr} = P(q_{t+1} = r | q_t = s)$
Emission matrix $\mathbf{B}$: $\forall s \in S, \forall \sigma \in \Sigma : b_s(\sigma) = P(o_t = \sigma | q_t = s)$
Initial probability distribution $\boldsymbol{\pi}$: $\forall s \in S : \boldsymbol{\pi}_s = P(q_1 = s)$

EVALUATION

Evaluation signal $\mathbf{O} = (o_1, ..., o_T)$ and model λ.

$$
\begin{aligned}
P(\mathbf{O}|\lambda) &= \sum_{\mathbf{Q} \in \mathcal{Q}_\lambda^T} P(\mathbf{O}|\mathbf{Q}, \lambda) P(\mathbf{Q}|\lambda) \\
&= \sum_{\mathbf{Q} \in \mathcal{Q}_\lambda^T} \pi_{q_1} b_{q_1}(o_1) a_{q_1 q_2} b_{q_2}(o_2) \ldots a_{q_{T-1} q_T} b_{q_T}(o_T)
\end{aligned}
$$

Where $\mathcal{Q}_\lambda^T$ is a set of all walks through the hidden state space of λ of length T.

FORWARD-BACKWARD PROCEDURE

Forward variable:

$$\forall s \in S : \alpha_1(s) = \pi_s b_s(o_1)$$
$$\forall s \in S, t \in \{2, ..., T\} : \alpha_t(s) = \sum_{r \in S} (\alpha_{t-1}(r) a_{rs}) b_s(o_t)$$

Backward variable:

$$\forall s \in S : \beta_T(s) = 1$$
$$\forall s \in S, t \in \{1, ..., T-1\} : \beta_t(s) = \sum_{r \in S} (a_{sr} b_r(o_{t+1}) \beta_{t+1}(r))$$

$$\sum_{s \in S} \alpha_T(s) = P(\mathbf{O}|\lambda) = \sum_{s \in S} \beta_1(s)$$

Baum-Welch Algorithm

$$\forall t \in \{1, ..., T-1\}, \forall s, r \in S : \xi_t(s, r) = P(q_t = s, q_{t+1} = r | \mathbf{O}, \lambda) =$$

$$= \frac{\alpha_t(i) a_{sr} b_r(o_{t+1}) \beta_{t+1}(r)}{\sum_{u \in S} \sum_{v \in S} (\alpha_t(u) a_{uv} b_v(o_{t+1}) \beta_{t+1}(v))}$$

$$\forall t \in \{1, ..., T\}, \forall s \in S : \gamma_t(s) = P(q_t = s | \mathbf{O}, \lambda) = \sum_{r \in S} \xi_t(s, r)$$

BAUM-WELCH ALGORITHM

$$\forall s \in S : \overline{\pi_s} = \gamma_1(s)$$

$$\forall s, r \in S : \overline{a_{sr}} = \frac{\sum_{t=1}^{T-1} \xi_t(s, r)}{\sum_{t=1}^{T-1} \sum_{u \in S} \xi_t(s, u)}$$

$$\forall s \in S, \forall \sigma \in \Sigma : \overline{b_s(\sigma)} = \frac{\sum_{t \in \mathcal{T}_{\mathbf{O}}(\sigma)} \gamma_t(s)}{\sum_{t=1}^{T} \gamma_t(s)}$$

$$\mathcal{T}_{\mathbf{O}}(\sigma) = \{t \in \{1, ..., T\} | o_t = \sigma\}$$

$$P(\mathbf{O}|\overline{\lambda} = (\overline{\mathbf{A}}, \overline{\mathbf{B}}, \overline{\boldsymbol{\pi}})) >= P(\mathbf{O}|\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}))$$

HMM VS PFA

- HMMs and PFAs are mutually convertible between each other.
- PFAs often utilise stopping probabilities (also used for *PAutomaC* models).
- Distribution defined by PFA with stopping probabilities: $P(\Sigma^*)$ against the HMM: $\forall n \in \mathbb{N} : P(\Sigma^n)$.
- A possible solution: introduce a new "stopping" symbol $x \notin \Sigma$ and create an HMM over the alphabet $\overline{\Sigma} = \Sigma \cup \{x\}$. End the signal once the new symbol x is reached.

HMM DENSITY

- Empirical results show that real world entities display "sparse" behaviour.
- Current state-of-the-art methods (Baum-Welch for HMMs) require the user to fully specify the amount of states and structure of the transition graph.
- Sparse transition matrix can also provide for a computational speedup.

Overfitting

Example:

We want to learn the model that generated some training data:

Training data

2135
4313
1325
2213
4133

Learned model

2135
4313
2135
2213
4133

Learned model

$2,2135$
$1,4313$
$1,2213$
$1,4133$

LL of training data: $\sum_O log\ P(O \mid \lambda)$

What about other data generated by the model?

New sequences	**Learned model**
4135	2, 2135
1322	1, 4313
4213	1, 2213
5135	1, 4133
1345	

LL of new data?

- 0?

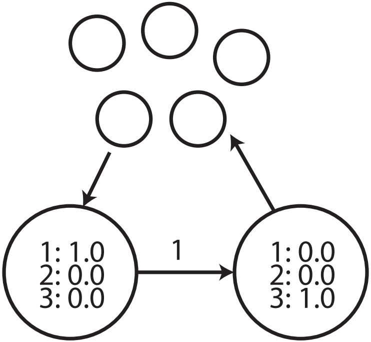
When does overfitting happen?

- Learning a too complex model
  - Models all the noise
- Training data too little
  - May not reflect model good enough

Assume a very general pattern exists:

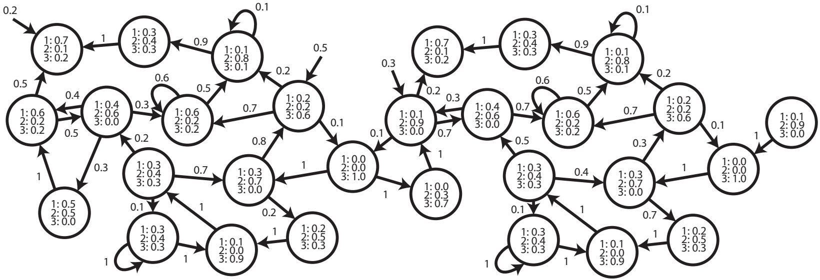
Training data
2$\underline{13}$5
43$\underline{13}$
$\underline{13}$25
22$\underline{13}$
4$\underline{13}$3

1 always followed by 3:



When learning a simple model:

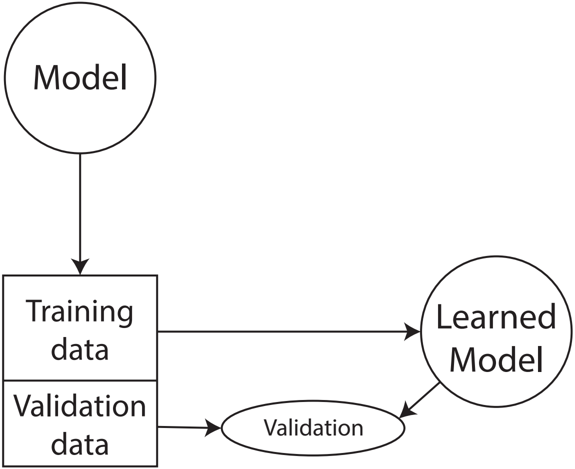
- Cannot express all noise
- May benefit more from describing general patterns



When learning a complex model:

- May be able to describe all noise
  - So why not do it?

How do we compare models / algorithms?



AVOIDING UNDERFLOW

$$P(X) = p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 \cdots$$

$$0 < p_x < 1$$

Underflow when $P(X) = 0$

Visual Studio, C#: Double precision floating point

- $0.01^{200} = 0$
- $0.02^{200} = 0$

Probability of a particular sequence can be very small:

- 23 symbols, uniformly distributed: $\frac{1}{23}^{238} = 0$
- What about rare symbols?
- What about probability of 1000 sequences?

Solution: Convert to log-space

$$log\ (a \cdot b) = log\ a + log\ b$$

Since $a < b \iff log\ a < log\ b$:

We can often stay in Log-space

$$log(0.01^{200}) = log\ 0.01 + log\ 0.01 + \cdots = -1381.55$$

$$log(0.02^{200}) = log\ 0.02 + log\ 0.02 + \cdots = -1173.60$$

Algorithms

STATIC VS. DYNAMIC

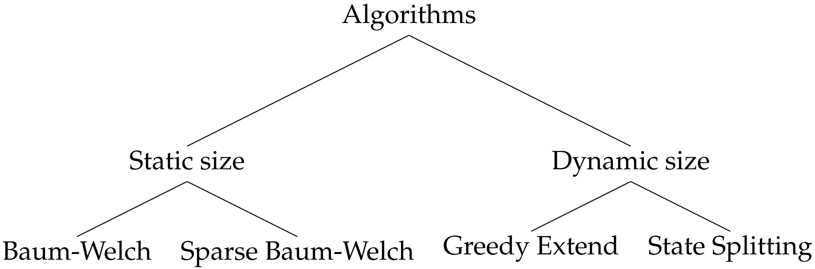


Figure: The algorithms used in our experiments

SPARSE BAUM-WELCH

- Creates HMM with n states and m symbols
- All parameters are initialized randomly
- Constraint: Each state has exactly log(n) transitions
- Other transitions set to zero
- Trained using Baum-Welch until convergence

GREEDY EXTEND: SETUP

- Works by adding states to the graph in iterations
- Starts as a single node with initial probability 1, random emission probabilities and a single transition to itself.

Greedy Extend: Iterations

1. Repeat until convergence
2. $G' = (V(G) \cup \{y'\}, E(G))$
3. Randomly choose a set X of $log|V(G')|$ nodes from $V(G')$
4. $\forall x \in X$ add transitions (x, y') and (y', x) to $E(G')$ with random probabilities
5. Normalize G'
6. if $LL(BW^{\beta}(G')) > LL(G)$, let $G = BW^{\beta}(G')$

STATE SPLITTING: OVERALL APPROACH

1. Identify a set of states $\mathcal{W}$ to split
2. Split all states in $\mathcal{W}$ using a mechanic
3. Run Baum-Welch for β iterations

STATE SPLITTING: SPLITTING MECHANICS

- Clone Split
  - Makes a copy of the chosen state
  - Problem: BW unable to distinguish between clone and original
  - Alternative: Randomize clones probabilities.
- Distribution Split
  - Only splits if Transition or Emission probabilities are uniform.
  - Copies the emission probabilities from the original to the new state
  - Randomizes transition probabilities on the new state
  - Problem: Algorithm can get stuck (splits after 10 iterations)

STATE SPLITTING: IDENTIFICATION HEURISTICS

- The Heuristics compute a score ς that is used to choose which states to split
- Gamma Heuristic
  - Assign ς based on the number of times the state is visited when generating the sequence
  - $\forall i \in \{1, ..., n\} : \varsigma S_i = \sum_{O \in D} \sum_{t=1}^{T} \gamma_t(S_i)$

STATE SPLITTING: IDENTIFICATION HEURISTICS

- Viterbi Heuristic
  1. Compute $Q = \mathcal{V}_G(O)$ foreach signal $O \in D$
  2. Foreach state $s \in S$ determine its significant positions in Q
  3. $\forall s \in S, \forall O \in D$ compute $\hat{\varsigma_O}(s) = \frac{\sum_{t \in \tau_{s,\lambda}^O} b_s(o_t)}{|\tau_{s,\lambda}^O|}$
  4. Compute $\forall s \in S$: $\varsigma(s) = \sum_{O \in D} P(Q|O, \lambda)\hat{\varsigma_O}(s)$

EDGE CUTTING & STATE REMOVAL

- Edge cutting
  - Strict Edge cutting
  - Threshold Edge cutting
- State Removal

Chosen Algorithm

The algorithm we chose for the experiments had the following characteristics

- Splitting Mechanic: Distribution Split
- Identification: Gamma Heuristic
- β value: 10

TEST ENVIRONMENT

TEST ENVIRONMENT

- Benchmark
- Data
- PautomacEvaluator
- Learner
- Models

Selecting Dataset

SELECTING DATASET

- Sparsity
- Transition Density

EXPERIMENT PARAMETER

EXPERIMENT PARAMETER

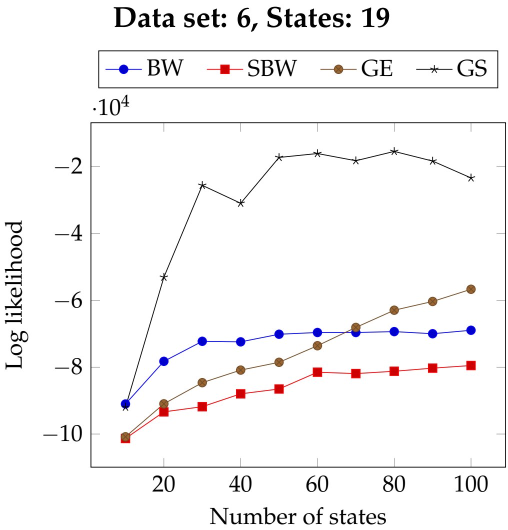
- Static Learners
- Training Sequences
- Baum-Welch Threshold
- Greedy Extend

EXPERIMENTS

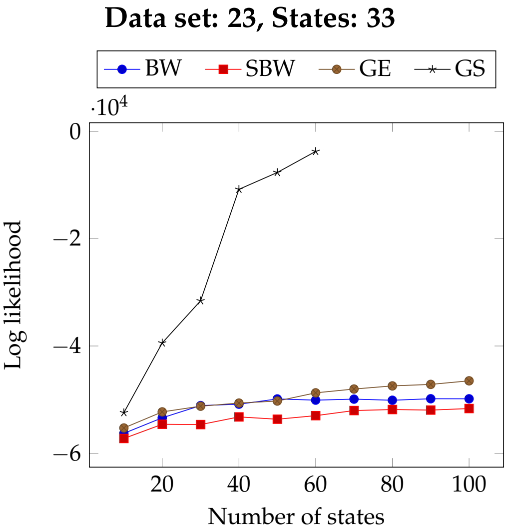
Experiments

- Training - 5000 sequences
- Validation - 5000 sequences
- States - 10 to 100
- How does problem characteristics affect prediction accuracy
- Speed comparison

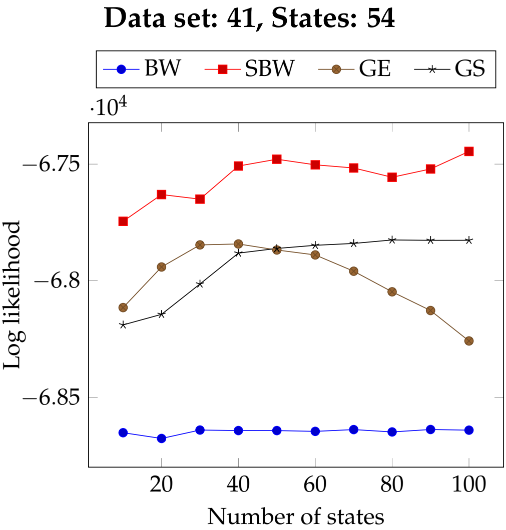
STATE SPACE


Data set: 6, States: 19
BW
SBW
GE
GS
$\cdot 10^4$
−2
−4
−6
−8
−10
Log likelihood
20
40
60
80
100
Number of states

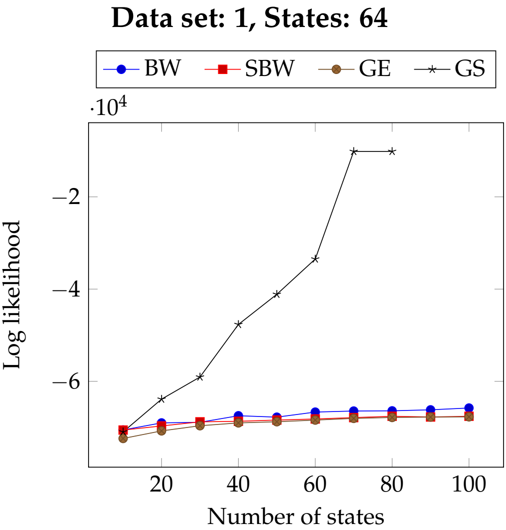
State space


Data set: 23, States: 33
BW
SBW
GE
GS
$\cdot 10^4$
0
−2
−4
−6
Log likelihood
20
40
60
80
100
Number of states

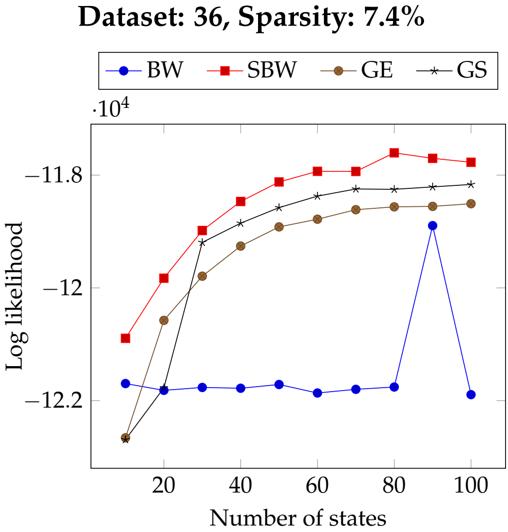
State space


Data set: 41, States: 54
BW
SBW
GE
GS
$\cdot 10^4$
−6.75
−6.8
−6.85
Log likelihood
20
40
60
80
100
Number of states

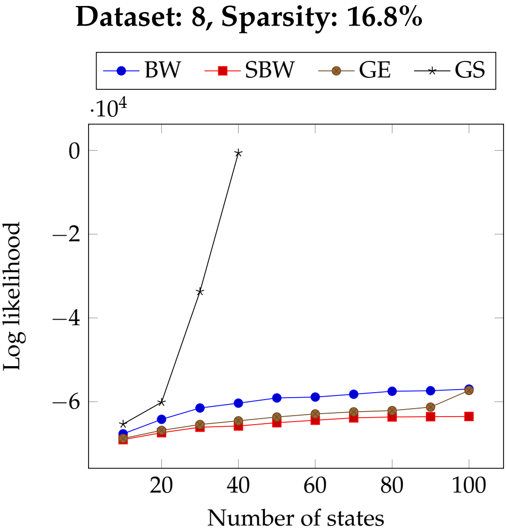
STATE SPACE


Data set: 1, States: 64
BW
SBW
GE
GS
$\cdot 10^4$
−2
−4
−6
Log likelihood
20
40
60
80
100
Number of states

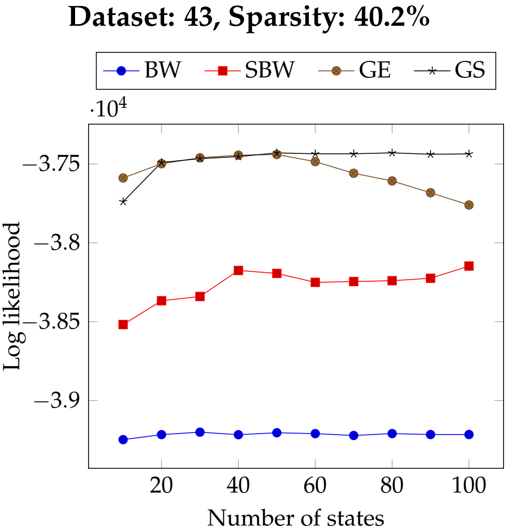
TRANSITION SPARSITY



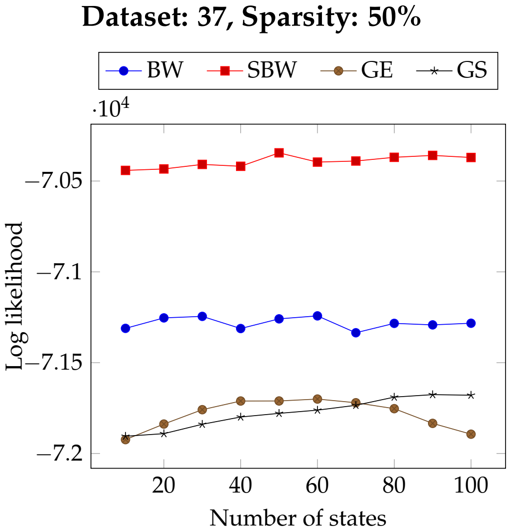
TRANSITION SPARSITY



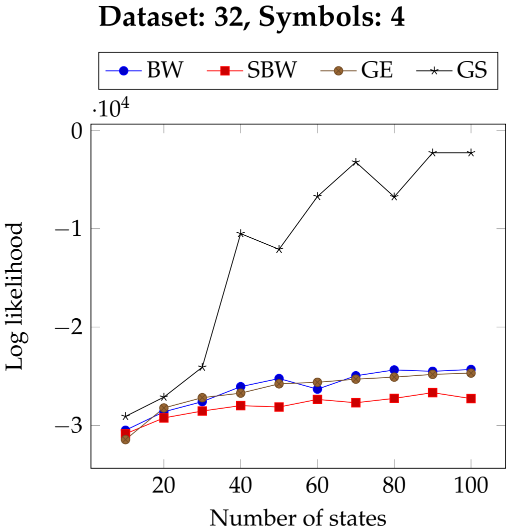
TRANSITION SPARSITY



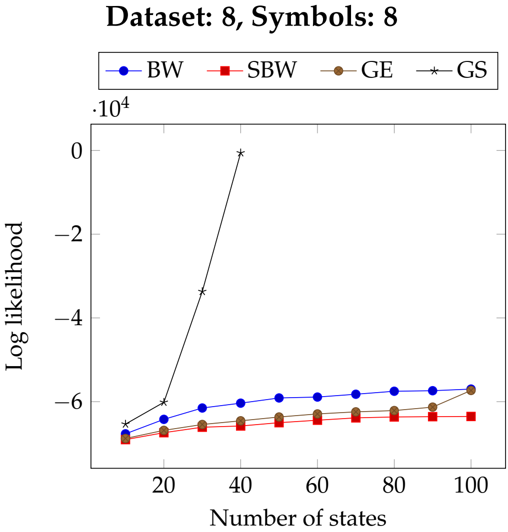
TRANSITION SPARSITY



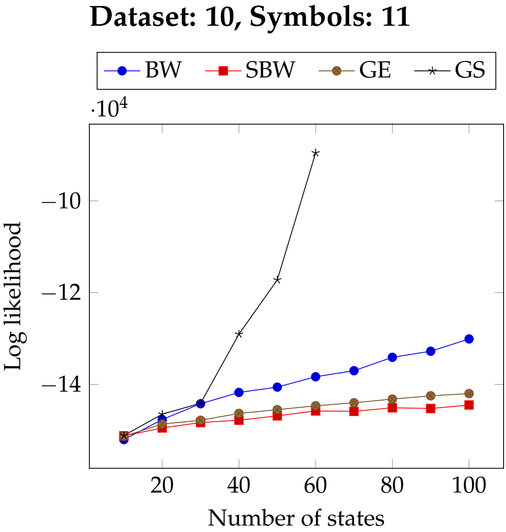
ALPHABET SIZE



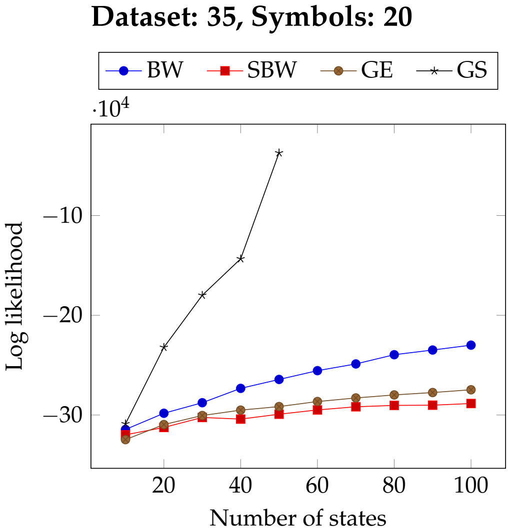
ALPHABET SIZE



Alphabet size



ALPHABET SIZE



SPEED COMPARISON

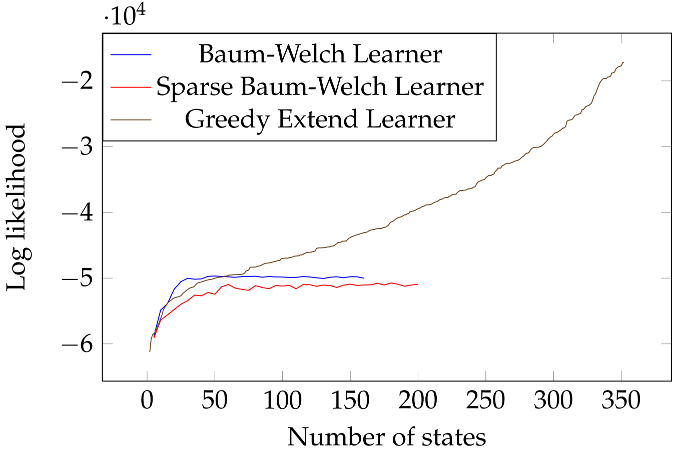


Figure: Results achieved in a time scope of eight hours.

SPEED COMPARISON

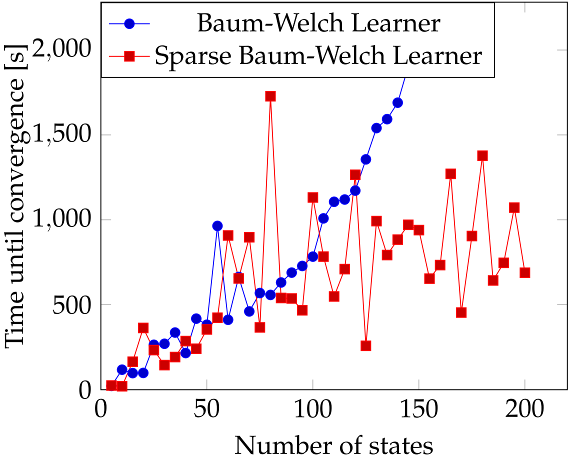


Figure: Running time comparison between Baum-Welch and Sparse Baum-Welch Learners

RESULTS

- 3 data sets: 6, 23, 35
- Parameters chosen based on the experiments presented earlier
- 5000 sequences
- 10 initial states, step size 10 for static algorithms
- 10 initial states, step size 5 for dynamic algorithms

RESULTS

Data set	**GE**	**BW**	**SBW**	**GS**	**Goal**
6	115.53	122.60	129.64	114.77	66.98
23	26.30	26.16	26.08	26.19	18.41
35	49.48	43.36	50.79	44.44	33.78

Table: The best scores of each algorithm on the three data sets.

RESULTS

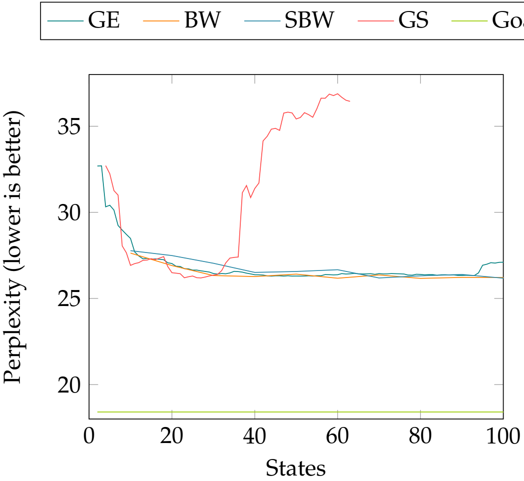


Figure: The different algorithm's score on *PAutomaC*'s 23rd data set, according to the perplexity measure used in the competition.

CONCLUSION

CONCLUSION

- Analysis of different approaches using a sparse transition matrix in the BW algorithm
- Dynamic algorithms: abous as good results as BW when not better
- Sparse matrix ensured by adding states iteratively

CONCLUSION: PROS AND CONS

Algorithm	Advantages	Drawbacks
GE	- Rather fast - Good results	- Usually needs more states to start to compare
GS	- Splitting heuristic - Best results, but...	- Runs on a dense matrix - ...unusual behaviour
SBW	- Computational speed-up - Room for improvement	- Results slightly worse than BW - No speed-up on some data sets

CONCLUSION: EXPERIMENTS CONDUCTED

- 3 types of experiment
- 4 algorithms

CONCLUSION: FIRST EXPERIMENTS

- Extremely varied behaviour on different data sets
- Three available parameters for results analysis
- Hypotheses, but no definite clear pattern
- In general, results on par with BW (sometimes outperforming it)

Conclusion: Second experiments

- Speed comparison between: BW, SBW, GE
- 8-hour run
- GE dominating both in speed and in score
- SBW unstable: random matrix causing non-deterministic behaviour (suggests buiding a data-derived one instead)

CONCLUSION: THIRD EXPERIMENTS

- Comparison using PAutomaC scores
- Unsatisfactory results, but...
- ...due to a different paradigm used: no stopping probabilities have been used

CONCLUSION: FUTURE WORK

- Eliminating possible noise
- Explain the multitude of behaviours, validating or disproving our hypotheses
- Running on more states
- Comparing with the PAutomaC scores, using the same paradigm
- Optimising the sparse transition matrices used in the algorithms
- Making sure underflow and overfitting are avoided
- Looking more in-depth into the other methods we have put aside in this project