

# Parameter Reusing and Perturbing for Learning Latent Class Models

Gytis Karčiauskas

Institute of Fundamental Technological Research, Polish Academy of Sciences,  
Warsaw, Poland

## Abstract

A problem in learning latent class models is that local-maximum parameters are often found. The standard solution of having many random starting parameterisations for iterative numerical methods is often too expensive computationally. The approach of reusing parameters from a previously learned model often obtains better than random starting parameterisations. However, we have found that it becomes inferior to the standard approach when the number of parameterisations becomes high. We propose an approach that combines the parameter-reusing technique with random perturbations of parameters. In the experiments with real data, our new approach outperforms the standard and parameter-reusing approaches at any time for models containing many classes.

**Keywords:** latent class, naive Bayes, EM algorithm, local maximum

## 1 Introduction

*Latent class analysis* is a method for finding classes of similar instances from multivariate categorical data. The data is assumed to be generated by a *latent class* (LC) model. The standard approach for determining parameters of an LC model is to use iterative numerical methods that start from random parameterisations. However, it often finds only local-maximum solutions or is too expensive computationally. In our earlier work (Karčiauskas, 2005, 2007), we have addressed this problem by proposing the *parameter-reusing* approach where parameters from a previously learned model are used for determining parameters of a next model. In the current work, we discover the problem that the parameter-reusing approach, when given more time, is often unable to find better solutions. To be able to do this, it needs to explore more different parameterisations. So, we introduce an approach where parameter reusing is combined with random perturbations of model parameters. We test our new approach on real data and analyse the results.

## 2 Notation and Definitions

In this section we give the notation and definitions that will be used in this paper.

A categorical *variable* is denoted by an upper-case letter (e.g.,  $D_i$ ), a *state* of a variable by a lower-case letter (e.g.,  $d_i$ ). A *vector of states* for a set of variables is denoted by a bold lower-case letter (e.g.,  $\mathbf{d}$ ). *Data* (or *data set*) over variables  $D_1, \dots, D_k$  is  $\mathbf{D} = (\langle \mathbf{d}_1, n_1 \rangle, \dots, \langle \mathbf{d}_N, n_N \rangle)$ , where each *instance*  $\mathbf{d}_j$  is a  $k$ -dimensional vector, instance *weight*  $n_j \geq 0$  is a real number, and  $\mathbf{d}_j \neq \mathbf{d}_{j'}$  for  $j \neq j'$ . The *size* of data  $\mathbf{D}$  is  $|\mathbf{D}| = \sum_{j=1}^N n_j$ . The *number of states* of a variable  $X$  is denoted by  $|X|$ .  $P$  denotes a *probability distribution* when it has variables as arguments, and a single *probability* otherwise.

A *model* is denoted by an upper-case letter (e.g.,  $M$ ) and is used for representing a joint probability distribution  $P_M(D_1, \dots, D_k)$ . Model  $M$  is specified by its *structure*  $\mathbf{m}$  and *parameters*  $\theta$ . That is,  $M = (\mathbf{m}, \theta)$ . The *likelihood* of data  $\mathbf{D}$  given model  $M$  is  $P(\mathbf{D}|M) = \prod_{j=1}^N P_M(\mathbf{d}_j)^{n_j}$ , where  $P_M(\mathbf{d})$  is the probability of  $\mathbf{d}$  given  $M$ . The *log-likelihood* of  $\mathbf{D}$  given  $M$  is denoted as  $LL(\mathbf{D}|M) = \ln P(\mathbf{D}|M) = \sum_{j=1}^N n_j \ln P_M(\mathbf{d}_j)$ . For fixed  $\mathbf{D}$  and  $\mathbf{m}$ , the *maximum-likelihood* parameters are  $\theta_{ML} = \arg \max_{\theta} P(\mathbf{D}|\theta, \mathbf{m})$ . The *dimension* of a model  $M$  (i.e., the number of independent parameters) is denoted by  $\dim(M)$ . A variable from  $M$  that is never observed in  $\mathbf{D}$  is called a *hidden* variable. Otherwise, it is called an *observed* variable.

$M$  is a *mixture* of  $m$  models if  $P_M(\mathbf{d}) = \sum_{l=1}^m P_M(h_l) P_M(\mathbf{d}|h_l)$ , where  $P_M(h_l)$  is the *weight* of the  $l$ th model,  $P_M(\mathbf{d}|h_l)$  depends only on the parameters of the  $l$ th model, and  $\sum_{l=1}^m P_M(h_l) = 1$ . So, a mixture can be seen as a model containing hidden variable  $H$  with states  $h_1, \dots, h_m$ , where each state corresponds to one of  $m$  models. For a mixture, we will also use the term *component* to indicate one of  $m$  models (i.e., one of states  $h_1, \dots, h_m$ ).

$\mathbf{D}_l$  denotes the part of  $\mathbf{D}$  that probabilistically belongs to component  $h_l$  in a mixture model. Formally,  $\mathbf{D}_l = (\langle \mathbf{d}_1, n_{l1} \rangle, \dots, \langle \mathbf{d}_N, n_{lN} \rangle)$ , where  $n_{lj} = n_j P_M(h_l|\mathbf{d}_j)$  (so,  $\sum_{l=1}^m n_{lj} = n_j, \forall j = 1, \dots, N$ ). Here  $P_M(h_l|\mathbf{d}_j) = \frac{P_M(h_l)P_M(\mathbf{d}_j|h_l)}{P_M(\mathbf{d}_j)}$ .

### 3 Latent Class Models

In this section we overview latent class models and methods for learning them.

In *latent class analysis* (Lazarsfeld and Henry, 1968), data  $\mathbf{D}$  is assumed to be generated by a *latent class* (LC) model, depicted in Figure 1. An LC

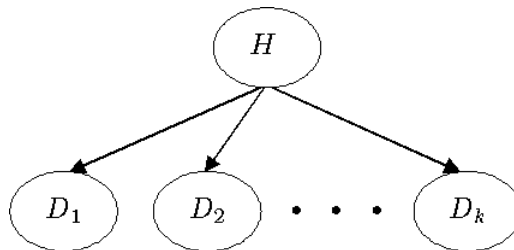


FIGURE 1: A latent class model.

model consists of a hidden *class variable* ( $H$ ) and observed *manifest variables*

$(D_1, \dots, D_k)$ . Manifest variables are assumed to be conditionally independent given the class variable. Seeing an LC model as a mixture model, each state of the class variable corresponds to a different component (class) and a sub-model for each component consists of variables  $D_1, \dots, D_k$  with no edges. An LC model is also known as a *naive Bayes* model with a hidden class variable. The *parameters* of an LC model  $L$  consist of a marginal probability distribution  $P_L(H)$  and conditional probability distributions  $P_L(D_i|H)$ ,  $i = 1, \dots, k$ . If the states of  $H$  are  $h_1, \dots, h_m$ , then the probability of  $\mathbf{d} = (d_1, \dots, d_k)$  given  $L$  is  $P_L(\mathbf{d}) = \sum_{l=1}^m P_L(h_l)P_L(\mathbf{d}|h_l) = \sum_{l=1}^m P_L(h_l) \prod_{i=1}^k P_L(D_i = d_i|h_l)$ . An LC model gives a “soft” classification of  $\mathbf{d}$ . That is,  $\mathbf{d}$  belongs to each class  $h_l$  with a probability  $P_L(h_l|\mathbf{d}) = \frac{P_L(h_l)P_L(\mathbf{d}|h_l)}{P_L(\mathbf{d})}$ .

The goal in latent class analysis is for given data  $\mathbf{D}$  to determine the optimal number of components  $|H|$  and model parameters. This is normally done by estimating the maximum-likelihood parameters for different  $|H|$  and then selecting  $|H|$  that gives the best model according to some criterion.

For a fixed  $|H|$ , the parameters are usually estimated by using iterative numerical methods. Among these, a popular choice is the *EM algorithm*. Dempster *et al.* (1977) have shown that the EM algorithm converges to a local maximum in terms of the likelihood as a function of model parameters, or equivalently, in terms of the log-likelihood  $LL(\mathbf{D}|M)$ . Since LC models often have many local maxima, the simple EM algorithm often finds local- rather than global-maximum parameters. The standard way of dealing with this problem is to run the simple EM many times from random starting points (i.e., starting parameterisations). The more starting points are used, the closer to the global maximum the best final parameters should be. However, often a high number of starting points is required, thus making the algorithm computationally very expensive. A more feasible computationally is the *multiple-restart EM* algorithm (Chickering and Heckerman, 1997), where many different random starting points are used, but repeatedly, after a specified number of EM iterations, only parameterisations giving the highest likelihood are retained. Even though this algorithm is faster, often it still requires much time and it may discard the potentially best parameterisation on a way.

After estimating parameters for different  $|H|$ , the optimal  $|H|$  can be selected by the  $G^2$  criterion or by a scoring function such as Laplace, BIC, Cheeseman-Stutz scores.

## 4 Parameter-Reusing Approach

In this section we give an overview of our previous work on the parameter-reusing approach for learning LC models.

As discussed above, the standard approach for learning LC models often becomes computationally expensive. In Karčiauskas 2005, 2007 we address this problem by proposing the *parameter-reusing* approach that tries to learn better models in the same time (or the same models in less time). When estimating parameters of a model with  $m + 1$  components, already estimated parameters of a model with  $m$  components are used. Starting points for EM are no longer completely random but are obtained by *splitting* a component in an  $m$ -component model.

**Definition 1** We say that model  $L^*$  is obtained from model  $L$  by splitting a component  $h_s$  if  $L^*$  instead of  $h_s$  contains components  $h_s^1$  and  $h_s^2$  that are both similar to  $h_s$ , and all the other components are identical in both models. Formally, if  $L$  contains components  $h_1, \dots, h_m$ , then  $L^*$  contains components  $h_1, \dots, h_{s-1}, h_s^1, h_s^2, h_{s+1}, \dots, h_m$  and the following is true:

- $P_{L^*}(h_l) = P_L(h_l)$ ,  $P_{L^*}(D_i|h_l) = P_L(D_i|h_l)$ ,  $l = 1, \dots, m$ ,  $l \neq s$ ,  $i = 1, \dots, k$ ,
- $P_{L^*}(h_s^1) = P_{L^*}(h_s^2) = \frac{1}{2}P_L(h_s)$ ,
- $\|P_{L^*}(D_i|h_s^1) - P_L(D_i|h_s)\| < \epsilon$ ,  $\|P_{L^*}(D_i|h_s^2) - P_L(D_i|h_s)\| < \epsilon$ ,  $i = 1, \dots, k$ , where  $\|\cdot\|$  is an  $L^2$ -norm of a vector and  $\epsilon \in \mathbb{R}$  is chosen in advance and is close to 0.

In our implementation of component splitting,  $P_{L^*}(D_i|h_s^1)$  and  $P_{L^*}(D_i|h_s^2)$  are initialised by making small random perturbations in  $P_L(D_i|h_s)$ . More precisely,  $h_s$  is split randomly by making  $P_{L^*}(D_i|h_s^1) - P_L(D_i|h_s) = \vec{r}_i$  and  $P_{L^*}(D_i|h_s^2) - P_L(D_i|h_s) = -\vec{r}_i$ , where each element of vector  $\vec{r}_i$  is a random number from  $[-p; p]$  and  $p \in \mathbb{R}$  is a small positive parameter.

Below we give an algorithm based on component splitting for learning LC models. The algorithm takes the training data  $\mathbf{D}$ , the number of starting points  $N_0$ , and the scoring function *score* as an input. Function *score* takes an LC model and a data set as arguments. Since the data set argument in our case is always  $\mathbf{D}$ , for the sake of simplicity we will always indicate only the first argument of *score*. The algorithm returns an LC model  $L$  that describes data  $\mathbf{D}$ . In step 1 the algorithm creates the one-component model, for which the maximum-likelihood parameters are computed deterministically from  $\mathbf{D}$ . Then in steps 2-13 the algorithm repeatedly increases the number of components until the model score no longer improves.

---

Algorithm *splitN*( $\mathbf{D}, N_0, \text{score}$ )

- 1: Let  $L$  be the one-component LC model for  $\mathbf{D}$ .
  - 2: **repeat**
  - 3:   **for each** component  $h_s$  of  $L$  **do**
  - 4:     Produce set  $\mathcal{L}_0$  of two-component models by performing  $N_0$  random independent splits of  $h_s$  and for each split producing a model that contains only new components  $h_s^1$  and  $h_s^2$  from  $L^*$  (see Definition 1).
  - 5:     Obtain model  $L_0$  by running the multiple-restart EM with  $\mathcal{L}_0$  as starting points and with data  $\mathbf{D}_s$ .
  - 6:     Obtain model  $L_s$  from  $L$  by substituting  $h_s$  with the two components from  $L_0$ .
  - 7:   **end for**
  - 8:    $L' \leftarrow \arg \max_{L_s} \text{score}(L_s)$ .
  - 9:   Optimise the parameters of  $L'$  by running the simple EM with data  $\mathbf{D}$ .
  - 10:   **if**  $\text{score}(L') > \text{score}(L)$  **then**
  - 11:      $L \leftarrow L'$ .
  - 12:   **end if**
  - 13: **until**  $L \neq L'$ .
  - 14: **return**  $L$ .
-

Parameter  $N_0$  regulates the trade-off between the model quality and the speed. Increasing  $N_0$  increases the number of starting points for EM, thus increasing the chances of finding higher-scoring models, but it also increases the computation time.

Algorithm *splitN* uses component splitting exactly in the same way as our algorithm from Karčiauskas 2005. There, we have compared experimentally the parameter-reusing and the standard approaches for learning LC models. In the standard approach, model parameters have been estimated by running the EM algorithm from random starting points. The number of starting points has been fixed in both approaches. In these experiments, the parameter-reusing approach was often better than the standard one, especially when  $|H|$  was high.

## 5 Parameter-Reusing-and-Perturbing Approach

In this section we report a problem with the parameter-reusing approach and describe a solution based on the idea of parameter perturbing.

The number of random splits  $N_0$  has been fixed in our experiments in the previous work because of computation time. However, a better experimental comparison could be obtained if the algorithms were run more than once, each time with a different number of starting points for EM. To be able to perform such experiments, we have re-implemented the algorithms in C++. In these experiments, algorithm *splitN* is compared to an algorithm based on the standard approach, which we give below. The structure of this algorithm is similar that of *splitN*, with the difference that starting points for EM are completely random (step 3) and in each iteration there is only one run of the multiple-restart EM (step 4).

---

Algorithm *standardN*( $\mathbf{D}$ ,  $N_0$ , *score*)

- 1: Let  $L$  be the one-component LC model for  $\mathbf{D}$ .
  - 2: **repeat**
  - 3:   Produce set  $\mathcal{L}_0$  of  $N_0$  randomly parameterised models having one component more than  $L$ .
  - 4:   Obtain model  $L'$  by running the multiple-restart EM with  $\mathcal{L}_0$  as starting points and with data  $\mathbf{D}$ .
  - 5:   **if**  $score(L') > score(L)$  **then**
  - 6:      $L \leftarrow L'$ .
  - 7:   **end if**
  - 8: **until**  $L \neq L'$ .
  - 9: **return**  $L$ .
- 

The setup of experiments is the following. Function *score* is the BIC score (Schwarz, 1978), defined as  $BIC(M) = LL(\mathbf{D}|M) - \frac{dim(M)}{2} \ln |\mathbf{D}|$ . The EM algorithm searches for the maximum-likelihood parameters. In the multiple-restart EM, half of the models giving the highest log-likelihood are retained after 1, 3, 7, 15, . . . iterations of EM, until only one best model is left. The EM algorithm is run until either the difference between successive values of log-likelihood is less than 0.001 or 1000 iterations are reached. Parameter  $p$  from Section 4 is 0.001.

The results of our initial tests have been surprising: even though for low values of  $N_0$  algorithm *splitN* in most cases found higher-scoring models than *standardN* at the same time, increasing  $N_0$  for *splitN* did not generally lead to higher-scoring models while for *standardN* it did. In the end, for high values of  $N_0$ , *standardN* unexpectedly performed better than *splitN*. So, it seems that the component-splitting operation, as it has been implemented, is not powerful enough.

Since for high number of starting points, where the highest-scoring models are expected to be found, the parameter-reusing approach performed worse than the standard approach, one may wonder if the parameter-reusing approach is worth any more attention. We think that the idea of using information about previous best parameterisations is still practical, because this may allow to start the EM algorithm from the parameterisations that are near the optimal one rather than from completely random parameterisations. The fact that increasing the number of initial parameterisations for *splitN* does not give the desired effects indicates that our implementation of component splitting concentrates too much on similar guesses of near-optimal parameterisations. Increasing the number of initial parameterisations only increases the number of these similar guesses, without even trying more different ones. So, we believe that algorithm *splitN* could be improved by introducing more random guesses of near-optimal parameterisations. If these guesses were completely random, we would have the standard approach, and the idea of parameter reusing would be lost. So, we propose to perturb randomly the parameterisations obtained after component splitting. This way, we still reuse the parameters from previous models, while at the same time giving the algorithm freedom to move to more different parameterisations.

Below we give an algorithm that implements this idea of parameter perturbing. The structure of this algorithm is similar to that of *splitN*. In step 10 the parameters are perturbed by changing randomly each marginal and conditional probability in a model. More precisely, model  $L_s^*$  with perturbed parameters of model  $L_s$  is obtained by creating a completely randomly parameterised model  $L^+$  that has the same structure as models  $L_s$  and  $L_s^*$ , and setting each marginal and conditional probability  $p^*$  in model  $L_s^*$  to  $p^* = p + c(p^+ - p)$ , where  $p$  is a corresponding probability in  $L_s$ ,  $p^+$  – a corresponding probability in  $L^+$ , and  $c \in [0, 1]$  – a constant specifying the strength of perturbation. In our implementation,  $c = \frac{1}{10}$ .

In our initial tests we have found that running the algorithm many times with  $N_0 = 1$  instead of running it once with a high value of  $N_0$  allows to explore more guesses of near-optimal parameterisations. That is why in the final experiments  $N_0$  will be 1.

## 6 Experiments

In this section we present the experiments where we compare different approaches for learning LC models.

The algorithms that are tested in the experiments take training data  $\mathbf{D}$  as an input and try to find an LC model that is the best according to function *score*. The first algorithm, named *standard*, implements the standard approach for

---

Algorithm *splitPerturbN*( $\mathbf{D}$ ,  $N_0$ , *score*)

```

1: Let  $L$  be the one-component LC model for  $\mathbf{D}$ .
2:  $N^* \leftarrow \max\{\lfloor \frac{N_0}{10} \rfloor, 1\}$ .
3: repeat
4:    $\mathcal{L}^* \leftarrow \emptyset$ .
5:   for each component  $h_s$  of  $L$  do
6:     Produce set  $\mathcal{L}_0$  of two-component models by performing  $N_0$  random independent splits of  $h_s$  and for each split producing a model that contains only new components  $h_s^1$  and  $h_s^2$  from  $L^*$  (see Definition 1).
7:     Obtain model  $L_0$  by optimising the parameters of each model in  $\mathcal{L}_0$  with the simple EM with data  $\mathbf{D}_s$  and then selecting the model with the highest likelihood.
8:     Obtain model  $L_s$  from  $L$  by substituting  $h_s$  with the two components from  $L_0$ .
9:     Optimise the parameters of  $L_s$  by running the simple EM with data  $\mathbf{D}$ .
10:    Produce set  $\mathcal{L}_1$  of  $N^*$  models by performing  $N^*$  random independent perturbations of the parameters of  $L_s$ .
11:     $\mathcal{L}^* \leftarrow \mathcal{L}^* \cup \mathcal{L}_1$ .
12:  end for
13:  Optimise the parameters of each model in  $\mathcal{L}^*$  by running the simple EM with data  $\mathbf{D}$ .
14:   $L' \leftarrow \arg \max_{L'' \in \{L_s\} \cup \mathcal{L}^*} \text{score}(L'')$ .
15:  if  $\text{score}(L') > \text{score}(L)$  then
16:     $L \leftarrow L'$ .
17:  end if
18: until  $L \neq L'$ .
19: return  $L$ .
```

---

learning LC models. Its pseudocode is given below. The algorithm repeatedly calls *standardN*, each time increasing the number of starting points  $N_0$ . The algorithm stops when the running time  $t$  is exceeded. Function *current\_time* always returns the current time.

The second algorithm, named *split*, implements the component-splitting approach from Karčiauskas 2005. Its pseudocode is the same as that of *standard*, except that *splitN* is called instead of *standardN*.

The third algorithm, named *splitPerturb*, implements the parameter-reusing-and-perturbing approach. Its pseudocode is the same as that of *standard*, except that *splitPerturbN* is called instead of *standardN* and the value of  $N_0$  is always 1.

Since in the standard approach the parameters of each model are estimated from scratch, there may be more efficient ways of determining final  $|H|$  than starting from  $|H| = 1$  and repeatedly incrementing it. That is why we also compare the parameter-reusing-and-perturbing with the standard approach when the latter has an unfair advantage of knowing the number of components from the start. The algorithm named *standardFixed* uses standard starting points for EM and estimates model parameters when the number of components is fixed. Besides the usual in-

---

Algorithm *standard*( $\mathbf{D}$ , *score*,  $t$ )

```

1:  $t_{start} \leftarrow current\_time$ .
2: Assume that  $score(L)$  is  $-\infty$ .
3:  $N_0 \leftarrow 1$ .
4: repeat
5:    $L' \leftarrow standardN(\mathbf{D}, N_0, score)$ .
6:   if  $score(L') > score(L)$  and  $current\_time - t_{start} \leq t$  then
7:      $L \leftarrow L'$ .
8:   end if
9:    $N_0 \leftarrow 2N_0$ .
10: until  $current\_time - t_{start} > t$ .
11: return  $L$ .

```

---

put, it also takes the number of components  $m$  as an input. Its pseudocode is the same as that of *standard*, except that in step 5 the algorithm produces set  $\mathcal{M}$  of  $N_0$  randomly parameterised LC models having  $m$  components and manifest variables from  $\mathbf{D}$  and then obtains model  $L'$  by running the multiple-restart EM with  $\mathcal{M}$  as starting points and with data  $\mathbf{D}$ . In the experiments, parameter  $m$  is equal to the number of components in the highest-scoring model found by *splitPerturb* for a given  $\mathbf{D}$ .

For training data, we selected 20 data sets from the UCI Machine Learning Repository (Asuncion and Newman, 2007).<sup>1</sup> For each data set, each of the reported algorithms is run once for the same time  $t$ , which depends on how fast the algorithms learn a model for a given data set. The setup of *score* and of the EM algorithm is the same as described in Section 5. The experiments are run on Intel® Pentium® M 740 1.73 GHz processor. The results of the experiments are summarised in Table 1.<sup>2</sup> For each algorithm we display the score (to one decimal point) and the number of components  $|H|$  of a final model found for a given data set. A score displayed in bold indicates that for a given data set the highest-scoring model has been found by a given algorithm. To make further discussion easier, above the middle line we have placed data sets for which  $|H|$  of a final model is higher than 17, and below it – the remaining data sets.

As seen from Table 1, the highest-scoring model has been found only by *splitPerturb* when  $|H|$  was higher than 17. When examining how the score of model  $L$  has been changing during time for each algorithm on each data set (here we do not include such graphs because of a limited space), it turned out for Adult, Letter,

---

<sup>1</sup>Class information and identifier variable, if present, were discarded. Continuous variables were converted into binary ones by performing the equal-frequency binning. When separate training and test data were available, they were joined into a single training data. Instances containing missing data were not included.

<sup>2</sup>In the table, “Credit” stands for “Credit Card Application Approval”, “Heart” – for the processed “Cleveland” data set from “Heart Disease”, “Image” – for “Image segmentation”, “Letter” – for “Letter Recognition”, “Pen” – for “Pen-Based Recognition of Handwritten Digits”, “Thyroid” – for the data set from “Thyroid Disease” suited for training artificial neural networks, “Wisconsin” – for the original data set from “Wisconsin Breast Cancer”. “Satimage”, “Shuttle”, and “Vehicle” are from the Statlog Project. For “Adult”, “Letter”, “Pen”, “Satimage”, “Spambase”, “Mushroom”, and “Pima”, time  $t$  was 20 hours, for the rest – 2 hours.

TABLE 1: Summary of results.

Data set name	<i>standard</i>		<i>split</i>		<i>splitPerturb</i>		<i>standardFixed</i>
	score	$ H $	score	$ H $	score	$ H $	score
Adult	-485344.8	22	-485787.3	21	<b>-485330.5</b>	22	-485330.8
Letter	-168353.3	41	-167438.5	64	<b>-167264.3</b>	70	-167383.2
Pen	-81311.0	42	-81388.5	40	<b>-81253.6</b>	47	-81284.3
Satimage	-63926.7	33	-64070.4	35	<b>-63879.2</b>	34	-63911.9
Shuttle	-255966.3	18	-256019.0	22	<b>-255965.6</b>	18	-255974.1
Spambase	-82733.4	21	-82667.5	20	<b>-82596.3</b>	22	-82699.9
Abalone	<b>-12389.4</b>	5	<b>-12389.4</b>	5	<b>-12389.4</b>	5	<b>-12389.4</b>
Audiology	<b>-47.8</b>	2	<b>-47.8</b>	2	<b>-47.8</b>	2	<b>-47.8</b>
Credit	<b>-7215.0</b>	4	-7227.8	4	-7216.3	4	<b>-7215.0</b>
Heart	<b>-2346.3</b>	2	<b>-2346.3</b>	2	<b>-2346.3</b>	2	<b>-2346.3</b>
Housing	<b>-3174.4</b>	9	-3201.0	6	-3175.0	8	-3175.0
Image	-15482.2	15	-15518.4	14	-15482.7	15	<b>-15482.1</b>
Mushroom	<b>-54468.7</b>	13	-55408.1	13	<b>-54468.7</b>	13	<b>-54468.7</b>
Page	<b>-22720.3</b>	16	-22778.1	15	<b>-22720.3</b>	16	<b>-22720.3</b>
Pima	<b>-3995.5</b>	4	<b>-3995.5</b>	4	<b>-3995.5</b>	4	<b>-3995.5</b>
Thyroid	<b>-44512.1</b>	7	-44543.9	7	-44540.1	8	-44537.2
Vehicle	<b>-6466.2</b>	10	-6478.9	10	<b>-6466.2</b>	10	<b>-6466.2</b>
Voting	<b>-1789.4</b>	3	<b>-1789.4</b>	3	<b>-1789.4</b>	3	<b>-1789.4</b>
Wisconsin	<b>-2510.2</b>	3	<b>-2510.2</b>	3	<b>-2510.2</b>	3	<b>-2510.2</b>
Yeast	<b>-6213.5</b>	2	<b>-6213.5</b>	2	<b>-6213.5</b>	2	<b>-6213.5</b>

Pen, Satimage, Shuttle, and Spambase data sets a model found by *splitPerturb* at any time had a higher score than models found by the other algorithms. This result, that for  $|H| > 17$ , *splitPerturb* outperforms even *standardFixed* at any time seems very strong to us. For the remaining data sets, *splitPerturb* has found the same or lower-scoring models than the standard algorithms. Generally, it seems that the chances of *splitPerturb* outperforming the standard algorithms increase as  $|H|$  increases. We think that this is because the problem of local maxima becomes more serious as  $|H|$  increases. According to Uebersax (2000), the main factor contributing to local-maximum solutions seems to be the number of components.

## 7 Summary and Future Work

Our idea of trying to escape local maximum by perturbing parameters has some similarities to the idea of simulated annealing (Kirkpatrick *et al.*, 1983). Other approaches to avoid local maximum include deterministic annealing (Ueda and Nakano, 1998), taking random subsamples (Fayyad *et al.*, 1998), perturbing training data (Elidan *et al.*, 2002).

Concerning theoretical properties, the parameter-reusing-and-perturbing algorithm is not guaranteed to find the global-maximum parameters even for a sufficiently high number of starting points. A way to obtain such a guarantee is to

make the value of  $c$  from Section 5 gradually increase and finally become 1. Since for  $c = 1$  the new parameterisation is completely random, the global-maximum parameters are guaranteed to be found after a sufficiently high number of tries. In this modification, the algorithm would be conservative in the beginning by concentrating on component splitting and would gradually put more emphasis on parameter perturbing. Another research direction could be extending our algorithm to learning Bayesian networks with hidden variables.

Our new approach, which for large models surpassed the best standard approach, gives no theoretical guarantees, but is an example of how the performance of an algorithm can be substantially improved by making it more stochastic.

## References

- A. ASUNCION and D.J. NEWMAN (2007), UCI Machine Learning Repository, URL <http://www.ics.uci.edu/~mllearn/MLRepository.html>.
- D. M. CHICKERING and D. HECKERMAN (1997), Efficient Approximations for the Marginal Likelihood of Bayesian Networks with Hidden Variables, *Machine Learning*, 29(2-3):181–212.
- A. DEMPSTER, N. LAIRD, and D. RUBIN (1977), Maximum likelihood from incomplete data via the EM algorithm, *Journal of the Royal Statistical Society*, 39(B):1–38.
- G. ELIDAN, M. NINIO, N. FRIEDMAN, and D. SCHUURMANS (2002), Data Perturbation for Escaping Local Maxima in Learning, in *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 132–139.
- U. M. FAYYAD, C. A. REINA, and P. S. BRADLEY (1998), Initialization of Iterative Refinement Clustering Algorithms, in *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pp. 194–198.
- G. KARČIAUSKAS (2005), *Learning with Hidden Variables: A Parameter Reusing Approach for Tree-Structured Bayesian Networks*, Ph.D. thesis, Aalborg University.
- G. KARČIAUSKAS (2007), Learning of Latent Class Models by Splitting and Merging Components, in *Advances in Probabilistic Graphical Models*, pp. 235–251, Springer-Verlag.
- S. KIRKPATRICK, C. D. GELATT, and M. P. VECCHI (1983), Optimization by Simulated Annealing, *Science*, 220:671–680.
- P. F. LAZARSFELD and N. W. HENRY (1968), *Latent Structure Analysis*, Boston: Houghton Mifflin.
- G. SCHWARZ (1978), Estimating the dimension of a model, *Annals of Statistics*, 6(2):461–464.
- J. UEBERSAX (2000), A Brief Study of Local Maxima Solutions in Latent Class Analysis, URL <http://ourworld.compuserve.com/homepages/jsuebersax/local.htm>.
- N. UEDA and R. NAKANO (1998), Deterministic annealing EM algorithm, *Neural Networks*, 11(2):271–282.