Chapter 7

Experimental results

This section describes the experiments that were run in order to assess the capabilities and usefulness of the mixture of trees model. The first experiments examine the ability of the **MixTree** algorithm to recover the original distribution when the data are generated by a mixture of trees. For these models, only artificial data are used. The next group of experiments studies the performance of the mixture of trees as a density estimator; the data used in these experiments are *not* generated by mixtures of trees. Finally, we perform classification experiments. In these experiments, we study both the mixture of trees and a classifier consisting of a single tree trained as a density estimator. Comparisons are made with classifiers trained in both supervised and unsupervised mode. The section ends with a discussion of the single tree classifier and its feature selection properties.

In all the experiments below, unless otherwise stated, the training algorithm is initialized at random, independently of the data or of the knowledge about the desired solution. Log-likelihoods are expressed in bits/example[1] and therefore are sometimes called *compression rates*. The lower the value of the compression rate, the better the fit to the data.

In the experiments that involve small data sets, a smoothing procedure has sometimes been used. The technique I applied is called *smoothing with the marginal* and is described in [24, 53]. One computes the pairwise marginal distributions for the whole data set P_{uv}^{total} and replaces the marginals P_{uv}^k by

$$\tilde{P}_{uv}^k \;=\; (1-\alpha)P_{uv}^k + \alpha P_{uv}^{total} \tag{7.1}$$

Intuitively, the effect of this operation is to give a small probability weight to unseen instances and to make the m trees more similar to each other, thereby reducing the effective model complexity. For the method to be effective in practice, α is a function of Γ_k and P_{uv}^k. In particular, for the experiments in this thesis, a global smoothing parameter α is apportioned between mixture components and, within each mixture component, beween tree edges. Whereas α in the above equation is always between 0 and 1, the global smoothing parameter may be larger than 1. In some cases (e.g. when fitting a single tree), I used smoothing with a uniform distribution instead of smoothing with the marginal.

7.1 Recovering the structure

7.1.1 Random trees, large data set

For the first experiment, we generated a mixture of 5 trees over 30 variables with $r = 4$ for all vertices. The distribution of the choice variable λ as well as each tree's structure

[1] 1 bit is the negative base 2 logarithm of a likelihood of 0.5.

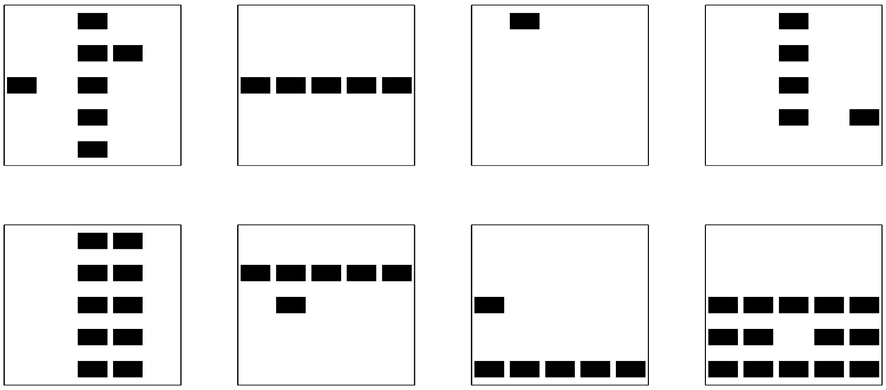


Figure 7-1: Eight training examples for the bars learning task.

and parameters were sampled at random. The mixture was used to generate 30,000 data points that were used as training set for a **MixTree** algorithm. The initial model had $m = 5$ components but otherwise was random. We compared the structure of the learned model with the generative model and computed the likelihoods of both the learned and the original model on a test dataset consisting of 1000 points.

The results on retrieving the original trees were excellent: out of 10 trials, the algorithm failed to retrieve correctly only 1 tree in 1 trial. This result can be accounted for by sampling noise; the tree that wasn't recovered had a λ of only 0.02. Instead of recovering the missing tree, the algorithm fit two identical trees to the generating tree with the highest λ. The difference between the log likelihood of the samples of the generating model and the approximating model was 0.41 bits per example. On all the correctly recovered trees, the approximating mixture had a higher log likelihood for the sample set than the generating distribution. This shows that not only the structure but the whole distribution was recovered correctly.

7.1.2 Random bars, small data set

The "bars" problem is a benchmark structure learning problem in neural network unsupervised learning algorithms. It has many variants, of which I describe the one used in the present experiments.

The domain V is the $l \times l$ square of binary variables depicted in figure 7-1. The data are generated the following manner: first, one flips a fair coin to decide whether to generate horizontal or vertical bars; this represents the hidden variable in our model. Then, each of the l bars is turned on independently (black in figure 7-1) with probability p_b. Finally, noise is added by flipping each bit of the image independently with probability p_n.

A learner is shown data generated by this process; the task of the learner is to "discover" the data generating mechanism.

What would it mean to discover the data generating mechanism with a mixture of trees? A mixture of trees model that approximates the true structure (figure 7-2) for low levels of noise is shown in figure 7-3. Note that any tree over the variables forming a bar is an equally good approximation. Thus, we will consider that structure has been discovered when the model learns a mixture with $m = 2$, each T^k having l connected components, one for each bar. Additionally, we shall test the "classification accuracy" of the learned model

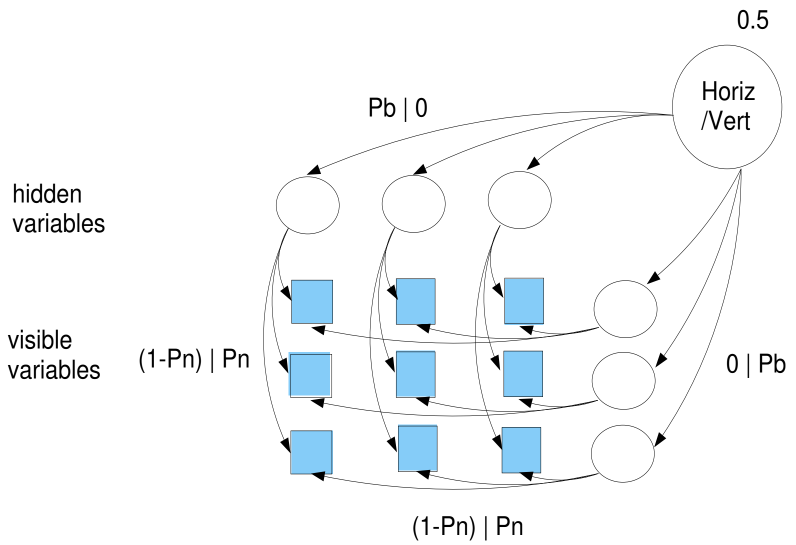


Figure 7-2: The true structure of the probabilistic generative model for the bars data.

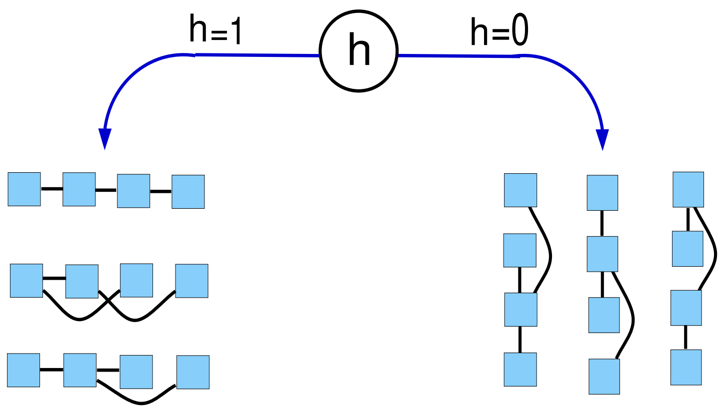


Figure 7-3: A mixture of trees approximate generative model for the bars problem. The interconnection between the variables in each "bar" are arbitrary.

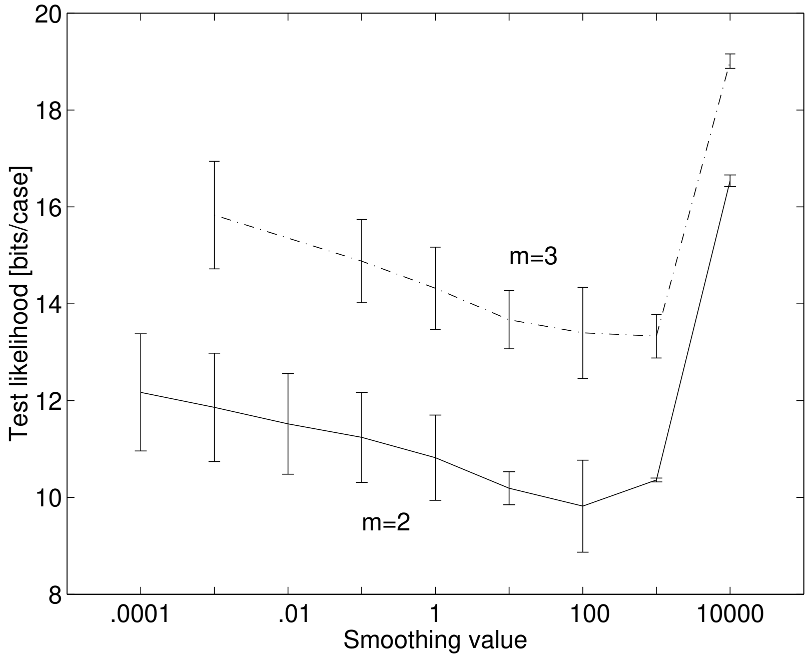


Figure 7-4: Test set log-likelihood on the bars learning task for different values of the smoothing α and different m. Averages and standard deviations over 20 trials.

by comparing the true value of the hidden variable (i.e. "horizontal" or "vertical") with the value estimated by the model for each data point in a test set.

We do not compare all of our results directly against neural network results, mainly because most of the published results are qualitative, focusing on whether the model is able to learn the structure of the data. We closely replicate the experiment described in [15] in a way that makes the present experiment slightly more difficult than theirs. The training set size is $N_{train} = 400$ (in [15] it is 500), the data set contains ambiguous examples (examples where no bars are present, which consequently are ambiguous from the point of view of classification) and the noise level is small but not zero (it is zero in [15]). Because the other authors studying this problem assume incomplete knowledge of structure (in this case, the number of hidden units of the neural net), we do something similar: we train models with $m = 2, 3, \ldots$ and choose the final model by the likelihood on a holdout set. Typical values for l in the literature are $l = 4, 5$ (the problem becomes harder with increasing l); we choose $l = 5$ following [15]. The probabilities p_b and p_n are 0.2 and 0.02 respectively. The test set size is $N_{test} = 200$. To obtain trees with several connected components we use a small edge penalty $\beta = 5$. Because the data set is small smoothing with different values for α is used. For each value of m we run the **MixTree** learning algorithm 20 times on the same training set with different random initial points and average the results.

Selection of m The average test set log-likelihoods (in bits) for $m = 2, 3$ are given in figure 7-4, together with their standard deviations. Clearnly, $m = 2$ is the best model.

Structure recovery. For $m = 2$ we examined the resulting structures: in 19 out of 20 trials, structure recovery was perfect (in the sense explained above); in the remaining trial the learned model missed the correct structure altogether, converging to a different local minimum apparently closer to its initial point. For comparison, [15] examined two training methods and the structure was recovered in 27 and respectively 69 cases out of 100. This

Table 7.1: Results on the bars learning task.

Test set	ambiguous	unambiguous
l_{test} [bits/datapt]	9.82 ± 0.95	13.67 ± 0.60
Class accuracy	0.852 ± 0.076	0.951 ± 0.006

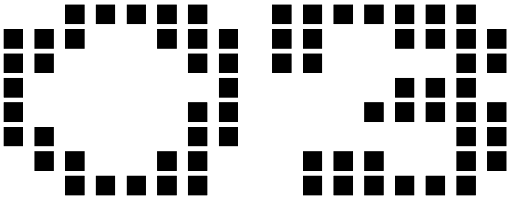


Figure 7-5: An example of a digit pair.

result held for the whole range of the smoothing parameter α.

Classification. The classification performance is shown in table 7.1. The result reported is obtained for a value of α chosen based on the test set log-likelihood. Note that on data generated by the previously described mechanism, no model can achieve perfect classification performance. This is due to the "ambiguous" examples, like the one shown in figure 7-1 (upper row, third from left), where no bars are on (or, less likely, where all bars are on). The probability of an ambiguous example for the current value of p_b is

$$p_{ambig} \;=\; p_b^l + (1-p_b)^l \;=\; 0.25 \tag{7.2}$$

Since ambiguous examples appear in either class with equal probability, the error rate caused by them is $0.5p_{ambig} = 0.125$. Comparing this theoretical upper bound with the value in the corresponding column of table 7.1 shows that the model has indeed a very good classification performance, even trained on ambiguous examples.

To further support this conclusion, a second test set of size 200 was generated, this time including only non-ambiguous examples. The classification performance, shown in the corresponding section of table 7.1 rose to 0.95. The table also shows the likelihood of the (test) data given the learned model. For the first, "ambiguous" test set, this is 9.82, 1.67 bits away from the true model entropy of 8.15 bits/data point. For the "non-ambiguous" test set, the compression rate is significantly worse, no surprise since the distribution of the test set is now different from the distribution the model was trained on.

7.2 Density estimation experiments

7.2.1 Digits and digit pairs images

We tested the mixture of trees as a density estimator by running it on a subset of binary vector representations of handwritten digits and measuring the average log-likelihood. The datasets consist of normalized and quantized 8x8 binary images of handwritten digits made available by the US Postal Service Office for Advanced Technology. One data set (that we call "digits") contained images of single digits in 64 dimensions, the second (called "pairs" hitherto) contained 128 dimensional vectors representing randomly paired digit images. Figure 7-5 displays an example of a digit pair. The training, validation and test

Table 7.2: Average log-likelihood (bits per digit) for the single digit (Digit) and double digit (Pairs) datasets. Boldface marks the best performance on each dataset. Results are averaged over 3 runs.

m	Digits	Pairs
16	**34.72**	79.25
32	34.48	**78.99**
64	34.84	79.70
128	34.88	81.26

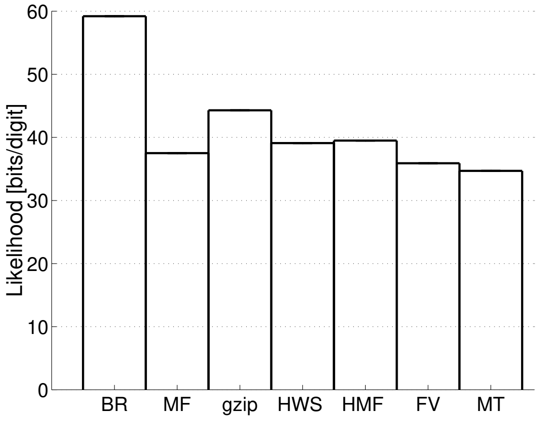
set contained 6000, 2000, and 5000 exemplars respectively. The data sets, the training conditions and the algorithms we compared with are described in [21]. Each model is trained on the same training set until the likelihood of the validation set stops increasing. We tried mixtures of 16, 32, 64 and 128 trees, fitted by the basic algorithm. For each of the digits and pairs datasets we chose the mixture model with the highest log-likelihood on the validation set and using it we calculated the average log-likelihood over the test set (in bits per example). The averages (over 3 runs) are shown in table 7.2. Notice the small difference in test likelihood between models that is due to early stopping in the training algorithm.

In figure 7-6 we compare our result (for $m = 32$) with the results published by [21].The other algorithms plotted in the figure are the mixture of factorial distributions (MF), the completely factored model (which assumes that every variable is independent of all the others) called "Base rate" (BR), the Helmholtz Machine trained by the wake-sleep algorithm [21] (HWS), the same Helmholtz Machine where a mean field approximation was used for training (HMF) and a fully visible and fully connected sigmoid belief network (FV). Table 7.2 displays the performances of all the mixture of trees models that we tested.

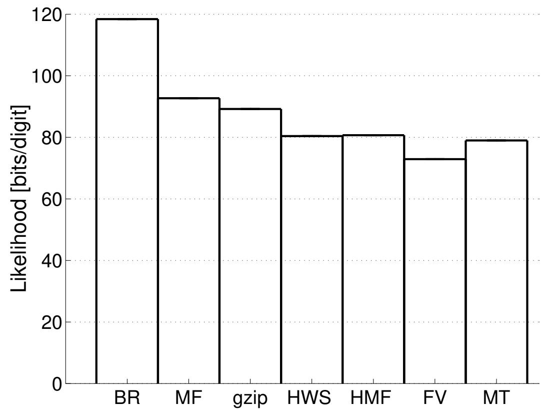
The results are very good: the mixture of trees is the absolute winner for compressing the simple digits and comes in second as a model for pairs of digits. A comparison of particular interest is the comparison in performance between the mixture of trees and the mixture of factored distribution. In spite of the structural similarities, the mixture of trees performs significantly better than the mixture of factorial distribution indicating that there exists some structure that is exploited by the mixture of spanning trees but can't be captured by a mixture of independent variable models. Comparing the values of the average likelihood in the mixture of trees model for digits and pairs we see that the second is more than twice the first. This suggests that our model (just like the mixture of factored distributions) is able to perform good compression of the digit data but is unable to discover the independence in the double digit set.

7.2.2 The ALARM network and data set

The second set of density estimation experiments features the ALARM network [33, 6]. This Bayes net model is a benchmark model for structure learning experiments. It is a medical diagnostic alarm message system for patient monitoring and it was constructed from expert knowledge. The domain has $n = 37$ discrete variables taking between 2 and 4 values, connected by 46 directed arcs. Since the ALARM net was not constructed from a complete data set I use in the experiments data generated from the model itself. This is artificial data, but it is an interesting experiment for two reasons: (1) the data generating



(a)



(b)

Figure 7-6: Average log-likelihoods (bits per digit) for the single digit (a) and double digit (b) datasets. MT is a mixture of spanning trees, MF is a mixture of factorial distributions, BR is the base rate model, HWS is a Helmholtz machine trained by the Wake-Sleep algorithm, HMF is a Helmholtz machine trained using the Mean Field approximation, FV is fully visible fully connected sigmoidal Bayes net. Notice the difference in scale between the two figures.

distribution is *not* a tree or a mixture of trees (17 nodes have more than one parent) but the topology of the DAG is sparse suggesting that the dependence can be approximated by a mixture of trees with a small number of components m; (2) the generative model captures the features of a real domain of practical importance.

We generated a training set having $N_{train} = 10,000$ data points and a separate test set of $N_{test} = 2,000$ data points. On these sets we train and compare the following models: mixtures of trees, mixtures of factorial distributions, the true model (without training), gzip (without training). For mixtures of trees and factorial distributions, the comparison is made in the following way: we separate a validation set of $N_{valid} = 1,000$ data points from the training set; then, models with different values of m are trained on the remaining data and evaluated (after the training process converges) on the validation set. For each value of m we run 20 trials. The averaged value of the log-likelihood on the validation set is used to choose the optimal m. Then, the models with that m are evaluated on the test set and the likelihood achieved on it is compared against other classes of models.

The results are presented in table 7.3. The result reported for gzip was obtained by writing the data in a file in binary format, with all additional information removed and presenting it to the gzip program.

One can see that, even with a 9,000 data set, the mixture of trees does not approximate the true distribution perfectly. However, note the difference in likelihood values for the true model between the training and the test set, suggesting that a data set in the thousands is still not a large data set for this domain. Among the other models used to approximate the true distribution, the mixture of trees is a clear winner. If being ahead of gzip and the base rate model was expected, it was not so with the mixture of factorials. However, the superiority of the mixture of trees w.r.t this model is also clear.

To examine the sensitivity of the algorithms to the size of the data set also ran the experiment with a training set of size 1,000. The results are presented in table 7.4.

Again, the mixture of trees is the closest to the true model, beatingthe mixture of

Table 7.3: Density estimation results for the mixtures of trees and other models on the ALARM data set. Training set size $N_{train} = 10,000$. Average and standard deviation over 20 trials.

Model	Train likelihood [bits/data point]	Test likelihood [bits/data point]
ALARM net	13.148	13.264
Mixture of trees $m = 18$	13.51 $\pm$0.04	14.55 $\pm$ 0.06
Mixture of factorials $m = 28$	17.11 $\pm$ 0.12	17.64 $\pm$ 0.09
Base rate	30.99	31.17
gzip	40.345	41.260

Table 7.4: Density estimation results for the mixtures of trees and other models on a data set of size 1000 generated from the ALARM network. Average and standard deviation over 20 trials.

Model	Train likelihood [bits/data point]	Test likelihood [bits/data point]
ALARM net	13.167	13.264
Mixture of trees $m = 2$, $\alpha = 50$	14.56 $\pm$0.16	15.51 $\pm$ 0.11
Mixture of factorials $m = 12$, $\alpha = 100$	18.20 $\pm$ 0.37	19.99 $\pm$ 0.49
Base rate	31.23 $\pm$ 0.21	31.18 $\pm$ 0.01
gzip	45.960	46.072

factorials by an even larger margin than for the large data set. Notice that the degradation in performance for the mixture of trees is relatively mild (about 1 bit), whereas the model complexity is reduced drastically (from 18 to only 2 mixture components, and this in the context of additional parameter smoothing!). This indicates the important role played by the tree structures in fitting the data and motivates the advantage of the mixture of trees over the mixture of factorials for this data set.

7.3 Classification with mixtures of trees

7.3.1 Using a mixture of trees as a classifier

Classification is a common and important task for which probabilistic models are often used. Very often density models are trained on data with the sole purpose of classification. This procedure is not asymptotically optimal, but there is empirical evidence that it can offer good performance in practice. Therefore, this section will be devoted to experimentally assessing the performance of the mixture of trees model in classification tasks.

A density estimator can be turned into a classifier in two ways, both of them being essentially *likelihood ratio* methods. Denote the class variable by c and the set of input variables by V. In the first method, adopted in our classification experiments under the name of *mixture of trees classifier*, one (mixture of trees) model Q is trained on the domain $\{c\} \bigcup V$, treating the class variable like any other variable and pooling all the training data together. In the testing phase, a new instance $x \in \Omega(V)$ is classified by picking the most likely value of the class variable given the other variables' settings.

$$c(x) \;=\; \operatorname*{argmax}_{x_c} Q(x_c, x) \tag{7.3}$$

The second method calls for partitioning the training set according to the values of the class variable and for training a density estimator on each partition. This is equivalent with training a mixture of trees with visible choice variable, the choice variable being the class c. This method was first used by [7]; if the trees are forced to have the same structure we obtain the Tree Augmented Naive Bayes (TANB) classifier of [24]. To classify a new instance x one turns to Bayes formula

$$c(x) \;=\; \operatorname*{argmax}_{k} P[c = k] T^k(x) \tag{7.4}$$

7.3.2 The AUSTRALIAN data set

We investigated the performance of mixtures of trees on three classification tasks from the UCI repository [1]. In the first experiment, the data set was the AUSTRALIAN database [1]. It has 690 examples each consisting of 14 attributes and a binary class variable. Six of the attributes (numbers 2, 3, 7, 10, 13 and 14) were real-valued and they were discretized into 4 (for attribute 10) respectively 5 (for all other attributes) roughly equally sized bins. In our experiments, in order to compare with [41] the test and training set sizes were 70 and 620 respectively (a ratio of roughly 1/9). For each value of m that was tested we ran our algorithm for a fixed number of epochs on the training set and then recorded the performance on the test set. This was repeated 20 times for each m, each time with a random start and with a random split between the test and the training set. In a first series of runs all training parameters were fixed except for m. In the second series of runs, we let β,

Table 7.5: Performance comparison between the mixture of trees model and other classification methods on the AUSTRALIAN dataset. The results for mixtures of factorial distribution are those reported in [41]. All the other results are from [50].

Method	**% correct**	**Method**	**% correct**
Mixture of trees $m = 20$, $\beta = 4$	**87.8**	Backprop	84.6
Mixture of factorial distributions (D-SIDE in [41])	87.2	C4.5	84.6
Cal5	86.9	SMART	84.2
ITrule	86.3	Bayes Trees	82.9
Logistic discrimination	85.9	K-nearest neighbor	81.9
Linear discrimination	85.9	AC2	81.9
DIPOL92	85.9	NewID	81.9
Radial basis functions	85.5	LVQ	80.3
CART	85.5	ALLOC80	79.9
CASTLE	85.2	CN2	79.6
Naive Bayes	84.9	Quadratic discrimination	79.3
IndCART	84.8	Flexible Bayes	78.3

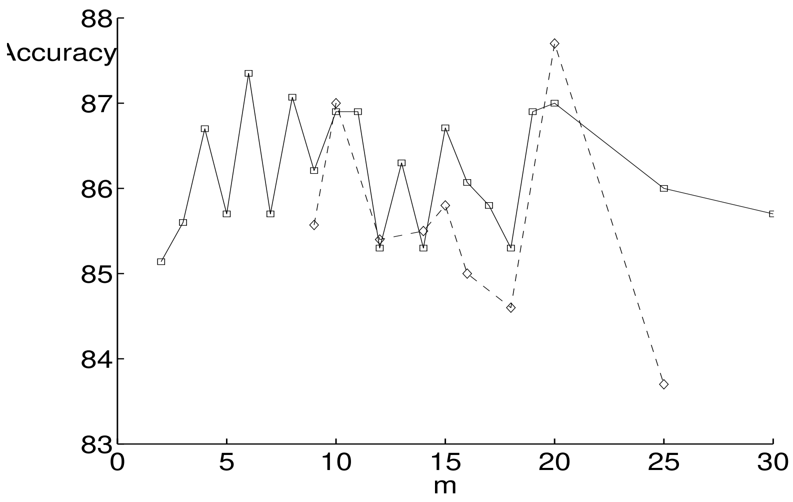


Figure 7-7: Classification performance of different mixture of trees models on the Australian Credit dataset. Results represent the percentage of correct classifications, averaged over 20 runs of the algorithm.

Table 7.6: Performance of mixture of trees models on the MUSHROOM dataset. m=10 for all models.

Algorithm	**Correctly classified**	**Soft class** $\sum \gamma_{\text{true}}/N$	**Test likelih. (bits/datapoint)**	**Train likelih. (bits/data point)**
No smoothing	.998	.997	27.63	27.09
Smooth w/ marginal $\alpha_M = 0.3,\ \alpha_P = 20$	1	.9999	27.03	26.97
Smooth w/ uniform $\alpha_M = 0.3,\ \alpha_P = 200$	1	.9997	27.39	27.02

the edge pruning parameter, change in such a way as to keep $m\beta$ approximatively constant. This results in roughly the same fraction of pruned edges independently of the number of components. The results, which are slightly better than in the previous experiment, are presented in figure 7-7. What is common to both series of experiments is the relatively large set of values of m for which the performance stays at the top of its range. We hypothesize that this is caused by the multiple ways the models are smoothed: edge pruning, smoothing with the marginals and early stopping.

The best performance of the mixtures of trees in the second case is compared to other published results for the same dataset in table 7.5. For comparison, correct classification rates obtained and cited in [41] on training/test sets of the same size are: 87.2% for mixtures of factorial distributions and 86.9% for the next best model (a decision tree called Cal5). The full description of the other methods can be found in [41, 50]. It can be seen that on this dataset the mixture of trees achieves the best performance, followed by the mixture of factorial distributions from [41].

7.3.3 The MUSHROOM data set

The second data set used was the MUSHROOM database [58]. This data set has 8124 instances of 23 discrete attributes (including the class variable, which is treated like any other attribute for the purpose of model learning). The training set comprised 6000 randomly chosen examples, and the test set was formed by the remaining 2124. The smoothing methods used were a) a penalty α_P on the entropy of the mixture variable and b) smoothing with the marginal according to (7.1) or with a uniform distribution. The smoothing coefficient α_M was divided between the mixture components proportionally to $1/\Gamma_k$. For this dataset, smoothing was effective both in reducing overfitting and in improving classification performance. The results are shown in table 7.6. The soft classification colums expresses an integrated measure of the confidence of the classifier. Note that besides the classification being correct, the classifier also has achieved high confidence.

7.3.4 The SPLICE data set. Classification and structure discovery

The last task was the classification of DNA SPLICE-junctions. The domain consists of 60 variables, representing a sequence of DNA bases, and an additional class variable. The

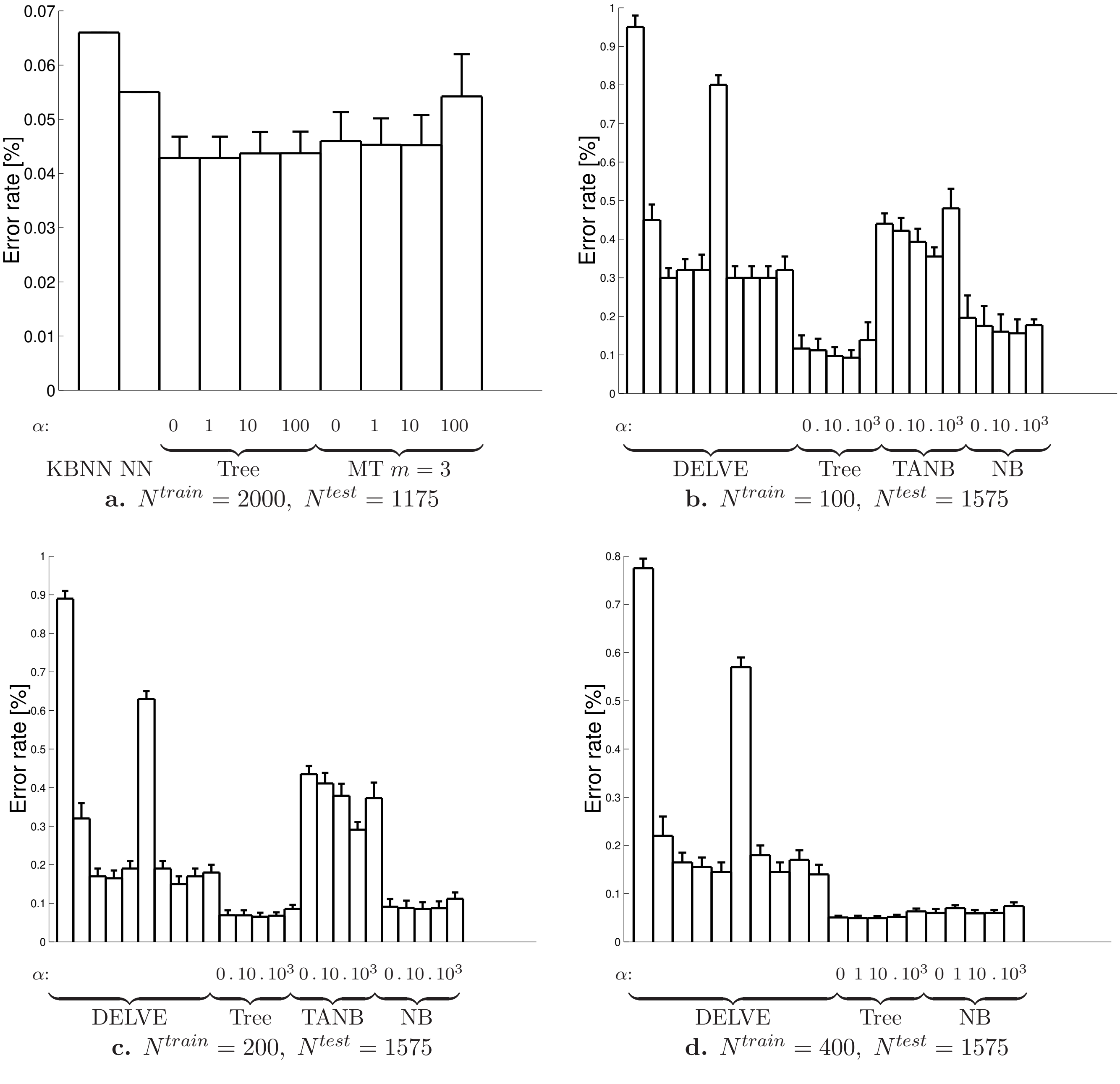


Figure 7-8: Comparison of classification performance of the mixture of trees and other models on the SPLICE data set. The models tested by DELVE are, from left to right: 1-nearest neighbor, CART, HME (hierarchical mixture of experts)-ensemble learning, HME-early stopping, HME-grown, K-nearest neighbors, Linear least squares, Linear least squares ensemble learning, ME (mixture of experts)-ensemble learning, ME-early stopping. TANB is the Tree Augmented Naive Bayes classifier of [24], NB is the Naive Bayes classifier, Tree represents a mixture of trees with $m = 1$, MT is a mixture of trees with $m = 3$. KBNN is the Knowledge based neural net, NN is a neural net.

task is to determine if the middle of the sequence is a splice junction and what is its type[2]. Hence, the class variable can take 3 values (EI, IE or no junction) and the other variables take 4 values corresponding to the 4 possible DNA bases coded here by the symbols (C, A, G, T). The data set consists of 31175 labeled examples. We compared the Mixture of trees model with two categories of classifiers and thus we performed two series of experiments. A third experiment involving the SPLICE data set will be described later.

For the first series, we compared our model's performance against the reported results of [67] and [54] who used multilayer neural networks and knowledge-based neural networks for the same task. The sizes of the training set and of the test set are reproduced from the above cited papers; they are 2000 and 1175 examples respectively. We constructed trees ($m = 1$) and mixtures of $m = 3$ trees with different smoothing values α. Fitting the single tree can be done in 1 step. To fit the mixture, we separated N_{valid}=300 examples out of the training set and learned the model using the EM algorithm on the remaining 1700. The training was stopped when the likelihood of the validation set stopped decreasing. This can be regarded as an a additional smoothing for the $m = 3$ model. The results, averaged over 20 trials, are presented in figure 7-8,a. It can be seen that the tree and the mixture of trees model perform very similarly, with the single tree showing an insignificantly better classification accuracy. Since a single tree has about three times fewer parameters than a mixture with $m = 3$ we strongly prefer the former model for this task. Notice also that in this situation smoothing does not improve performance; this is not unexpected since the data set is relatively large. With the exception of the "oversmoothed" mixture of trees model ($\alpha = 100$) all the trees/mixture of trees models significantly outperform the other models tested on this problem. Note that whereas the mixture of trees contains no prior knowledge about the domain, the other two models do: the neural network model is trained in supervised mode, optimizing for class accuracy, and the KBNN includes detailed domain knowledge before the training begins.

The second set of experiments pursued a comparison with benchmark experiments on the SPLICE data set that are part of the DELVE repository [56]. The DELVE benchmark uses subsets of the SPLICE database with 100, 200 and 400 examples for training. Testing is done on 1500 examples in all cases. The algorithms tested by DELVE and their performances are shown in figure 7-8,b,c, and d. We fitted single trees ($m = 1$) with different degrees of smoothing. We also learned naive Bayes (NB) and Tree Augmented Naive Bayes (TANB) models [24]. A NB or a TANB model can be fit in one step, just like a tree. According to the DELVE protocol, we ran our algorithms 20 times with different random initializations on the same training and testing sets.

The results are presented in figures 7-8b,c and d. Most striking in these plots is the dramatic difference between the methods in DELVE and the classification obtained by a simple tree: in all cases the error rates of the tree models are *less than half* of the performance of the best model tested in DELVE. Moreover, the average error of a single tree trained on 400 examples is 5.5%, which is only 1.2% away from the average error of trees trained on the 2000 examples dataset. The explanation for this remarkable accuracy preservation with the decrease of the number of examples is discussed later in this section. The Naive Bayes model exhibits a behavior that is very similar to the tree model and only slightly less accurate. However, augmenting the Naive Bayes model to a TANN significantly hurts

[2] The DNA is composed of sections that are useful in coding proteins, called *exons*, and of inserted sections of non-coding material called *introns*. Splice junctions are junctions between an exon and an intron and they are of two types: exon-intron (EI) represents the end of an exon and the beginning of an intron whereas intron-exon (IE) is the places where the intron ends and the next exon, or coding section, begins.

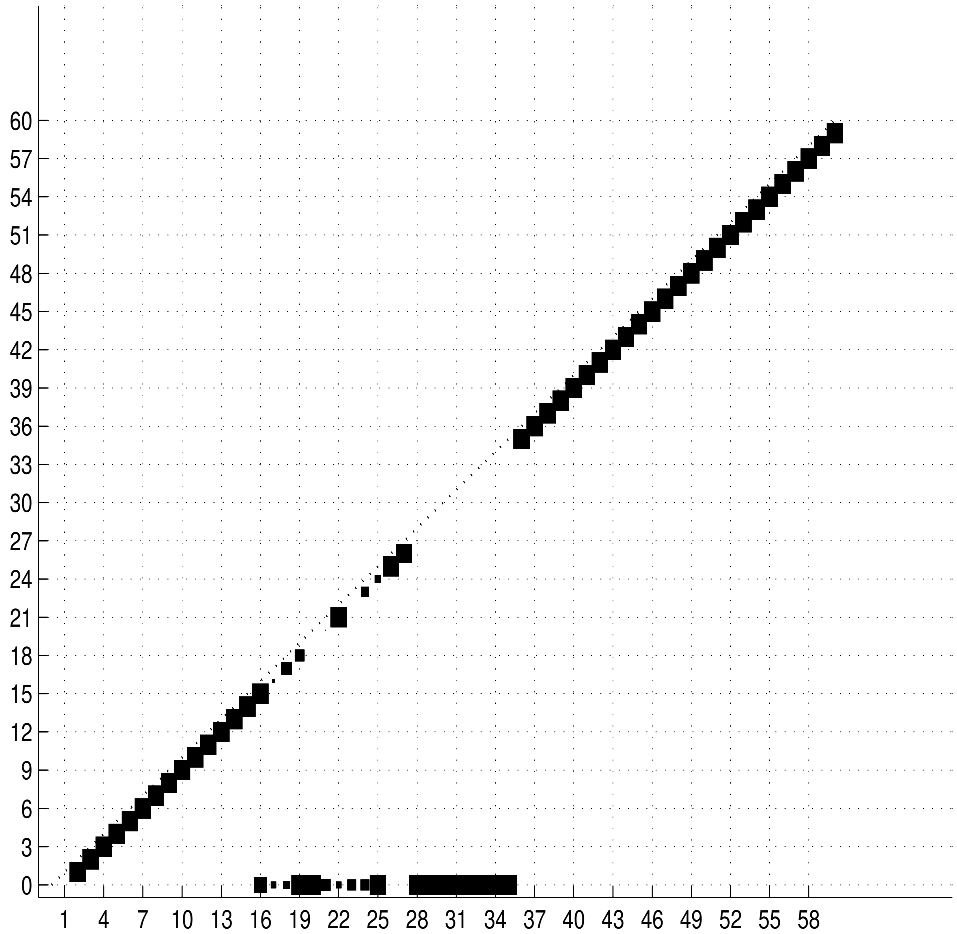


Figure 7-9: Cumulative adjacency matrix of 20 trees fit to 2000 examples of the SPLICE data set with no smoothing. The size of the square at coordinates ij represents the number of trees (out of 20) that have an edge between variables i and j. No square means that this number is 0. Only the lower half of the matrix is shown. The class is variable 0. The group of squares at the bottom of the figure shows the variables that are connected directly to the class. Only these variable are relevant for classification. Not surprisingly, they are all located in the vicinity of the splice junction (which is between 30 and 31). The subdiagonal "chain" shows that the rest of the variables are connected to their immediate neighbors. Its lower-left end is edge 2–1 and its upper-right is edge 60-59.

EI junction

	Exon			Intron					
	28	29	30	31	32	33	34	35	36
Tree	CA	**A**	**G**	**G**	**T**	AG	A	G	
True	**CA**	**A**	**G**	**G**	**T**	AG	A	G	T

IE junction

	Intron									Exon
	15	16	...	25	26	27	28	29	30	31
Tree	–	CT	CT	CT	–	–	CT	**A**	**G**	**G**
True	CT	CT	CT	CT	–	–	CT	**A**	**G**	**G**

Figure 7-10: The encoding of the IE and EI splice junctions as discovered by the tree learning algorithm compared to the ones given in J.D. Watson & al., "Molecular Biology of the Gene"[69]. Positions in the sequence are consistent with our variable numbering: thus the splice junction is situated between positions 30 and 31. Symbols in boldface indicate bases that are present with probability almost 1, other A,C,G,T symbols indicate bases or groups of bases that have high probability (>0.8), and a – indicates that the position can be occupied by any base with a non-negligible probability.

the classification performance. The plots also allow us to observe the effect of the degree of smoothing when it varies from none to very large. In contrast to the previous experiment on SPLICE data, here smoothing has a beneficial effect on the classification accuracy for values under a certain threshold; for larger values the accuracy is strongly degraded by smoothing. These accuracy profiles can be observed in tree models as well as in the TANN and Naive Bayes models and they are similar to the bias-variance tradeoff curves commonly encountered in machine learning applications. Also not surprisingly, the effect diminishes with the increasing size of the data set (and is almost invisible for a training set size of 400).

Discovering structure. Figure 7-9 presents a summary of the tree structures learned from the N=2000 example set in the form of a cumulated adjacency matrix. The adjacency matrices of the 20 graph structures obtained in the experiment have been summed[3]. The size of the black square at coordinates i, j in the figure is proportional to the value of the i, j-th element of the cumulated adjacency matrix. No square means that the respective element is 0. Since the adjacency matrix is symmetric, only half the matrix is shown. From figure 7-9 we can see first that the tree structure is very stable over the 20 trials. Variable 0 represents the class variable; the hypothetic splice junction is situated between variables 30 and 31. The figure shows that the splice junction (variable 0) depends only on DNA sites that are in its vicinity. The sites that are remote from the splice junction are dependent on their immediate neighbors. Moreover, examining the tree parameters, for the edges adjacent to the class variable, we observe that these variables build certain patterns when the splice junction is present, but are random and almost uniformly distributed in the absence of a splice junction. The patterns extracted from the learned trees are shown in figure 7-10. The same figure displays the "true" encodings of the IE and EI junctions

[3]The reader is reminded that the adjacency matrix of a graph has a 1 in position ij if the graph has an edge connecting vertices i and j and a 0 otherwise.

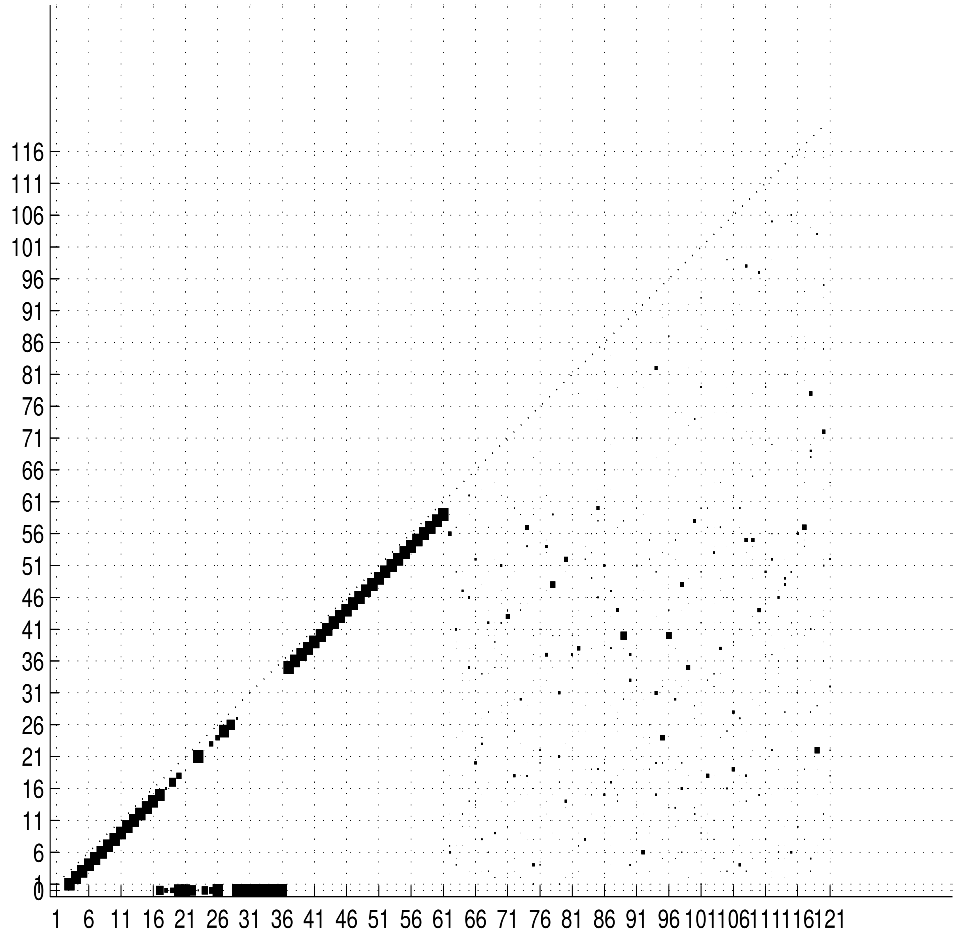


Figure 7-11: The cumulated adjacency matrix for 20 trees over the original set of variables (0-60) augmented with 60 "noisy" variables (61-120) that are independent of the original ones. The matrix shows that the tree structure over the original variables is preserved.

as given in [69]. The match between the two encodings is almost perfect. Thus, we can conclude that for this domain, the tree model not only provides a good classifier but also discovers the true underlying model of the physical reality that generates the data. Remark also that the algorithm arrives at this result *without any prior knowledge*: it does not know which variable is the class and it doesn't even know that 60 variables out of 61 are in a sequence.

7.3.5 The single tree classifier as an automatic feature selector

.

Let us examine the single tree classifier that was used for the SPLICE data set more closely. According to the separation properties of the tree distribution, the probability of the class variables depends only on its neighbors, that is, on the variables to which the class variable is connected by tree edges. Hence, a tree acts as an implicit variable selector for classification: only the variables adjacent to the queried variable (this set of variables is called the *Markov blanket* [55]) will be relevant for determining its probability distribution. This property also explains the observed preservation of the accuracy of the tree classifier when the size of the training set decreases: out of the 60 variables, only 18 are relevant to the class; moreover, the dependence is parametrized as 18 independent pairwise probability tables $T_{class,v}$. Such parameters can be accurately fit from relatively few examples. Hence, as long as the training set contains enough data to establish the correct dependency structure, the classification accuracy will degrade slowly with the decrease in the size of the data set.

Now we can explain the superiority of the tree classifier over the models in DELVE in the previous example: out of the 60 variables, only 17 are relevant to the class. The tree finds them correctly. A classifier that is not able to perform a reasonable feature selection will be hindered by the remaining irrelevant variables, especially if the training set is small. This is exactly what figure 7-8 shows: the absolute difference in classification error between the models in DELVE and the tree is largest for the smallest training set.

For a given Markov blanket, the tree classifies in the same way as a naive Bayes model with the Markov blanket variables as inputs. Remark also that the naive Bayes model itself has a built-in feature selector: if one of the input variables v is not relevant to the class, the distributions $P_{v|c}$ will be roughly the same for all values of c. Consequently, in the posterior $P_{c|v}$ that serves for classification, the factors corresponding to v will simplify and thus v will have (almost) no influence on the classification. This explains why the naive Bayes model also performs well on the SPLICE classification task. Notice however that the variable selection mechanisms implemented by the tree classifier and the naive Bayes classifier are not the same.

Sensitivity to irrelevant features. To verify that indeed the single tree classifier acts like a feature selector, we performed the following experiment, using again the SPLICE data. We augmented the variable set with another 60 variables, each taking 4 values with randomly and independently assigned probabilities. The rest of the experimental conditions (training set, test set and number of random restarts) were identical to the first SPLICE experiment. We fitted a set of models with $m = 1$, a small $\beta = 0.1$ and no smoothing and compared its structure and performance to the corresponding model set in the first experiment series. The structure of the new models, in the form of a cumulative adjacency matrix, is shown in figure 7-11. We see that the structure over the original 61 variables is unchanged and stable; the 60 noise variables connect in a random uniform patterns to the original variables and among each other. As expected after examining the structure, the classification performance of the new trees is not affected by the newly introduced variables: in fact the average accuracy of the trees over 121 variables is 95.8%, 0.1% higher than the accuracy of the original trees[4].

[4]The standard deviation of the accuracy is 3.5% making this difference insignificant.