Robust Classification using Mixtures of Dependency Networks

José A. Gámez[1] and Juan L. Mateo[1] and Thomas D. Nielsen[2] and José M. Puerta[1]
[1] Computing Systems Department – SIMD $i^3\mathcal{A}$ University of Castilla-La Mancha 02071, Albacete, Spain
[2] Department of Computer Science Aalborg University 9220 Aalborg, Denmark

Abstract

Dependency networks have previously been proposed as alternatives to e.g. Bayesian networks by supporting fast algorithms for automatic learning. Recently dependency networks have also been proposed as classification models, but as with e.g. general probabilistic inference, the reported speed-ups are often obtained at the expense of accuracy. In this paper we try to address this issue through the use of mixtures of dependency networks. To reduce learning time and improve robustness when dealing with data sparse classes, we outline methods for reusing calculations across mixture components. Finally, the proposed model is empirically compared to other state-of-the-art classifiers, both in terms of accuracy and learning time.

1 Introduction

Classification is a typical data mining task in which the class label for new instances must be inferred from the values taken by their predictive attributes. The induction of accurate classifiers from pre-labelled data is a hot area of research in machine learning and artificial intelligence. Among the wide range of paradigms used to induce classifiers, Bayesian network classifiers (BNCs) (Friedman et al., 1997) have received much attention.

In this paper we consider the use of a special class of dependency networks (Heckerman et al., 2000) for classification. Compared to BNs, dependency networks allow directed cycles, and structural learning can therefore be performed very efficiently, since the node families can be learned independently of each other. On the other hand, due to the presence of cycles, standard BN inference algorithms cannot be adapted and so approximate algorithms (Gibbs sampling being the most prevalent) are often required. Fortunately, for classification problems where only the class variable is unknown, inference boils down to simply multiplying the appropriate table entries, and it can therefore be performed in linear time w.r.t. the number of variables.

The efficient learning algorithms for DNs hide their main problem, namely inconsistency (a DN does not necessarily represent a joint probability distribution). Moreover, inherent to DNs is the problem of overfitting: the node families are generally larger than in BNs, hence the estimation of the local probability distributions is less reliable and requires more data than in a BN (to reduce parameter variance). To address this problem, Heckerman et al. (2000) proposed to use probabilistic decision trees (PDTs) for parameter specification, and recently Gámez et al. (2008a) has proposed a specification language based on combinations of probability tables (PTs).

A study on inconsistencies in DNs was conducted by Gámez et al. (2008b), who also proposed a heuristic procedure to reduce the inconsistencies. In the study, Gámez et al. (2008b) conclude that PTs outperform PDTs in terms of accuracy, and in their experimental results the inconsistencies almost disappear when using PTs. On the other hand, PDTs have the ability to represent *contextual independence* (i.e., independence relations between variables that hold for some but not necessarily all values), but by focusing on PTs we lose this property.

In order to include aspects of contextual independence when working with PTs, we consider DN-based classifiers inspired by multinets (Geiger and Heckerman, 1996). Multinets are useful for representing certain types of contextual independence, which cannot easily be represented by a single PT-based model. In classification we may, for example, represent different independence relations for the different class values.

As we will also demonstrate, multinets support a notion of *re-usability*. The underlying idea of re-usability is to exploit similarities in the dependency structures across classes. That is, if we have an unbalanced class distribution, we may in some situations be able to reuse parts of a learned probability model (for an instance-rich class) when learning the probability model for a class with few instances. Thus, re-usability may produce more robust classifiers when dealing with unbalanced class distributions.

2 Dependency Networks

Dependency networks (DNs) were proposed by Heckerman et al. (2000) as an alternative to BNs. Formally, a dependency network is a tuple $(\mathbb{G}, \mathbf{P})$ over a domain $\mathbf{X}$ where $\mathbb{G}$ is a directed graph (not necessarily acyclic) and $\mathbf{P}$ is a set of conditional probability distributions, one for each variable in $\mathbf{X}$. Every $P \in \mathbf{P}$ must be such that

$$P(X_i|\mathbf{Pa}_i) = P(X_i|\mathbf{X} \setminus X_i).$$

This means that the set of parents $Pa(X_i)$ for every variable X_i is its Markov blanket $MB(X_i)$.

This definition requires specification *consistency*, in the sense that the joint probability distribution for $\mathbf{X}$ can be recovered from $\mathbf{P}$. This is a very restrictive condition when learning from data, so Heckerman et al. (2000) define *general dependency networks* that relax the factorization requirement by letting $P(\mathbf{X}) \approx \prod_i P(X_i|Pa_i)$.

Since directed cycles are allowed, a DN can be learned from data by learning the parent set for each variable independently, thus supporting fast learning algorithms. On the other hand, by having directed cycles we cannot adapt traditional BN inference algorithms. Instead, Heckerman et al. (2000) relies on approximate inference carried out using Gibbs sampling (Geman and Geman, 1984), and propose the so-called *modified ordered Gibbs sampler*, which is more efficient than standard Gibbs sampling. Fortunately, sampling algorithms are not required when we have complete data for the predictive attributes, and we shall therefore not discuss this topic further.

3 Multinets and Re-usability: Proposed Scheme

Multinets (Geiger and Heckerman, 1996) are useful for representing natural contextual independence assertions. For example, in a classification context we may directly represent the (possibly different) independence relations over the predictive attributes for each of the class values. Consequently a multinet classifier is expected to have a higher, or at least the same, representational power than that of a single BN classifier.

A multinet classifier based on DNs (termed a *MultiDN*) is built by learning a DN model for each class value. Every DN model is built by independently learning the MB for each variable using e.g. the IAMB algorithm (Tsamardinos et al., 2003), specified in Figure 1. The algorithm relies on a method for testing independence, and in this paper we have considered traditional statistical tests, such as G^2, as well as tests measuring the score difference between candidate structures (Chickering, 2002) using e.g. the BIC (Schwarz, 1978) or the BD score (Cooper and Herskovits, 1992).

It should be noted that as an alternative to IAMB, Peña et al. (2007) proposed the PCMB algorithm, which they showed to be more data efficient than IAMB. The results, however, also indicate that for very small data sets (e.g. Alarm with 100 instances) PCMB has worse precision (although better recall) than IAMB. The combination of these two factors means that PCMB may identify larger candidate MB sets than IAMB. This behavior was also confirmed in our

```
1  { Phase I (forward) }
2  MB = ∅
3  While MB has changed
4    Y = argmax_{X∈U\(MB∪{T})} dep(X,T|MB)
5    If Y ⊥̸⊥ T|MB Then
6        MB = MB ∪ {Y}
7
8  { Phase II (backwards) }
9  For each X ∈ MB Do
10   If X ⊥⊥ T|(MB\X) Then
11     MB = MB\{X}
12
13 Return MB
```

Figure 1: The incremental association Markov blanket (IAMB) algorithm for learning the MB for a target variable T.

preliminary experiments, where we compared both algorithms based on the datasets listed in Section 4 (having similar ratio between the number of variables and instances as the example above). This limits the usability of PCMB for learning DNs, since the larger MBs make the probability estimates less reliable.

3.1 Re-usability

In a multinet classifier we basically need to learn several networks, one for each class value. Assuming that the set of independence statements for the different class values are not disjoint, one might be able to use parts of the learned probability model for one class value when learning the probability model for another class value. The potential advantages are twofold: First we may speed up learning, and, secondly, we may obtain a more robust classifier when data is scarce for some of the classes.

For MultiDNs, re-usability consists in seeding the MB learning algorithm with a candidate MB set. In the IAMB algorithm this is achieved by simply replacing line 2 with **MB** = **seedMB**. We call this new algorithm *SeededIAMB*. If the candidate seed is good, i.e., it corresponds to the true MB or a subset hereof, the algorithm can achieve substantial computational savings (the situation where this is not the case is discussed below).

Theorem 1. *Under the assumptions that the independence tests are correct and that the database D is an independent and identically distributed sample from a probability distribution P faithful to a DAG $\mathcal{G}$, SeededIAMB identifies the true MB for the target variable.*

Proof sketch: Given any initial set the algorithm will in the forward phase always introduce all variables in the MB because, by definition, there is no set of variables that can make the variables in the MB independent of the target. So at the end of this phase we will have a super-set of the MB for the target variable. The backward phase, removing all false positives, behaves in the same way as the non-seeded version. □

As indicated above, if the candidate set used for *SeededIAMB* is close to the actual MB, then we may get computational savings. On the other hand, if the candidate set does not intersect the true MB, then the seeded version of the IAMB algorithm may introduce a computational overhead. Thus, it is important to find a good ordering in which to process the classes, and to be able to determine when a previous structure should be used as seed. Below we have detailed some of our considerations.

• Ordering the class values: In our experiments we order the classes according to the number of instances associated with each class value, starting with the class value having most instances. Here we assume that more data yields more reliable MB estimates for potential reuse.

• Determine the score for a candidate seed structure: Assume that we have a collection of previous models $DN_i, i = \{1, \ldots, n\}$, and that the current model must be learned from data $\mathcal{D}_{c_j}$. The log-likelihood of the previous models DN_i given $\mathcal{D}_{c_j}$, $\log L(DN_i|\mathcal{D}_{c_j})$, can be used to select which of these models to use as seed. Furthermore, we can also use the local score for each single variable X

$$\frac{1}{N_j}\sum_{l=1}^{N_j} \log P(X|MB(X)_i, DN_i)(\mathbf{d}_l),$$

indicating how well $MB(X)_i$ predicts X in $\mathcal{D}_{c_j}$.

• Setting the threshold for when to reuse a MB: A possible threshold value is the score of the empty structure, since if IAMB is not seeded, then this is the structure that is used as prior. Here we have two strategies. One strategy (called *BESTlogL*) consists in picking the best $MB(X)_i$ for a given variable X, if its score is greater than that of the empty MB. As another strategy (called *THRESHOLDlogL*), we can pick the union of all $MB(X)_i$ for all previous models that have greater score than the empty MB.

• When to compute the score of a candidate seed structure: Although the log-likelihood is easy to compute (its complexity is linear in the number of instances), it still incurs an overhead for the algorithm. We have considered three alternatives for deciding on re-usability without having to compute a score; thus, saving computation time but making the selection less informed. The first method simply uses the first learned model as seed for all the subsequent models, and is labelled *First*. The second method uses the intersection of the MBs for all the previous learned models (*Intersection*), and the third method uses the union of the MBs for all the previous learned models (*Union*).

4 Experiments

We have performed a set of experiments in order to analyze the performance of the proposed algorithm in terms of classification accuracy, learning time, and the potential improvement obtained by re-usability.[1] For evaluating the accuracy and learning time, we have compared our classifier with a collection of other well-known classifiers, both probabilistic and non-probabilistic. For the actual learning, we have used 28 dataset from the UCI repository (Asuncion and Newman, 2007), see Table 1.

All experiments have been carried out on a PC with a 3GHz Intel Pentium IV processor and 2Gb RAM memory.

[1]Given the space limitation not all results of our experiments are included, but a complete list can be found at http://www.dsi.uclm.es/personal/JuanLuis Mateo/mixtureDN.html

Table 1: Datasets used in our experiments.

dataset	insts.	vars.	|class|
australian	690	14	2
autos	205	23	7
balance	625	5	3
breast-cancer	286	10	2
breast-w	699	10	2
car	1728	7	4
cmc	1473	10	3
diabetes	768	7	2
ecoli	336	7	8
heart	270	14	2
hepatitis	155	20	2
ionosphere	351	34	2
iris	150	5	3
kr-vs-kp	3196	37	2
labor	57	12	2
mushroom	8124	23	2
nursery	12960	9	5
page-blocks	5473	11	5
post-op	90	9	3
segment	2310	20	7
soybean	683	36	19
spambase	4601	56	2
vehicle	846	19	4
vote	435	17	2
vowel	990	14	11
waveform	5000	20	3
wine	178	14	3
zoo	101	17	7

4.1 Algorithms

The algorithms used in the comparison are J48 (Quinlan, 1993), multilayer perceptron (NN) (Bishop, 1995), k-nearest neigbours (kNN) (Aha and Kibler, 1991), support vector machine (SVM) (Platt, 1999), naive Bayes (NB) (Langley et al., 1992), k-dependence Bayesian classifier (kDB) (Sahami, 1996), tree augmented naive Bayes (TAN) (Friedman et al., 1997), multinet with Bayesian networks (MultiBN) (Friedman et al., 1997), and another DN-based classifier (ChiSqDN) (Gámez et al., 2006).

We have used the Weka implementation of J48, NN, kNN, and SVM (Witten and Frank, 2005). For these classifiers, all parameters were set to their default values, except for kNN for which we tried both $k = 1$, and $k = 3$ with inverse distance weighting. For kDB we have experimented with $k = 1, 2, 3, 4$. For MultiBN we have considered two variants. One based on the PC learning algorithm (Spirtes et al., 2001), and the other based on local search (hillclimbing) with the BIC (Schwarz, 1978) or the BD score (Cooper and Herskovits, 1992). For MultiDN we used the IAMB algorithm to determine the MB for each variable, and the independence tests were performed using either G^2 or by measuring the difference in BIC score for the candidate structures. All algorithms (except J48, NN, kNN and SVM) have been implemented in Java with the Elvira software (Elvira Consortium, 2002). Accuracy is assessed using a 5x2 cross validation scheme.

4.2 Results

For each algorithm appearing in several versions (different parameter settings) we only report on the version giving the best accuracy results. That is, kNN with $k = 3$ and inverse distance weighting, kDB with $k = 1$, and multinets based on the BIC score.

Table 2 shows the accuracy results for some of the selected classifiers; the results reported for MultiDN relates to the BIC variant with no reuse across classes. Due to space restrictions we have only included the best classifiers, but the results for the remaining classifiers can be found at the web page specified above.

From Table 2 we see that SVM achieves the best result on average. In order to determine which of the other classifiers that (from a statistical point of view) cannot be considered weaker than SVM, we have carried out Holm's post-hoc test with the SVM classifier as control. This test shows that kDB and both multinet-based classifiers are comparable with SVM; the remaining classifiers receive worse results and the differences are statistically significant.

4.2.1 Learning Time

In Table 3 we list the learning time for the algorithms selected above. MultiDN obtains the best results in several cases and is never the worst. MultiBN, on the other hand, is never the best and several times the worst, whereas SVM is the fastest for most of the datasets, but it is sometimes the slowest too. In order to test whether there is a statistical difference, we have evaluated the results using Wilcoxon's signed rank test. With significance level 0.05 the test indicates that MultiDN is significantly faster than kDB and MultiBN, but there is no such difference between SVM and MultiDN, and they can therefore be considered equally fast. The critical values for the comparisons are 1.42e-3 for kDB, 8.20e-7 for MultiBN, and 0.52 for SVM.

Table 2: Accuracy results. For each dataset, the best results are shown in bold face, and the worst results are underlined.

	kDB	MultiBN	MultiDN	SVM
`australian`	84.64	85.42	**86.35**	84.93
`autos`	79.02	79.81	73.27	**82.14**
`balance`	74.05	73.82	74.08	74.11
`breast-cancer`	70.28	69.79	70.49	71.12
`breast-w`	96.22	96.57	97.34	96.68
`car`	93.26	91.68	91.01	92.45
`cmc`	**53.96**	52.49	53.29	53.96
`diabetes`	77.47	77.68	**79.27**	76.77
`ecoli`	84.82	85.12	82.26	83.87
`heart`	81.63	80.3	82.37	83.7
`hepatitis`	**87.88**	86.45	85.81	85.55
`ionosphere`	91.97	**92.65**	92.19	90.48
`iris`	95.07	94.53	96.27	**96.40**
`kr-vs-kp`	94.22	96.49	95.27	95.24
`labor`	89.8	92.22	**94.72**	92.29
`mushroom`	99.87	**100.00**	99.95	99.99
`nursery`	93.26	95.57	93.73	93.06
`page-block`	95.75	96.42	96.24	**96.81**
`post-op`	66.22	66.89	66.89	69.56
`segment`	94.17	94.94	91.36	95.56
`soybeam`	91.6	**94.00**	93.35	92.5
`spam-base`	92.73	93.65	92.58	**93.81**
`vehicle`	71.23	71.28	70.33	**73.14**
`vote`	93.75	93.89	94.16	**95.22**
`vowel`	73.17	69.82	64.99	79.07
`waveform`	82.25	81.79	81.26	**84.86**
`wine`	97.08	97.98	98.31	97.87
`zoo`	94.65	94.26	94.06	94.26
aver.	85.72	85.91	85.4	**86.62**

4.2.2 Re-usability Analysis

To get an indication of the theoretical complexity of the re-use methods, we introduce the notion of reuse-complexity, which is defined as the number of computations[2] times the average number of variables involved in each computation; note that reuse-complexity does not take into account the overload introduced by *BEST-logL* and *TRHESHOLDlogL*. This overload is considered in learning time. Table 4 shows the average complexity and learning time among all datasets.

From the results we see that reusability based on *Intersection* achieves the best results in

[2]A computation is a call to the function that either calculates the score of a local structure with BIC or BDe, or performs a statistical test with G^2.

Table 3: Learning time in mileseconds.

	kDB	MultiBN	MultiDN	SVM
`australian`	543	1036	249	**174**
`autos`	711	3226	951	**268**
`balance`	**30**	46	37	85
`breast-cancer`	89	372	94	**72**
`breast-w`	196	228	130	**68**
`car`	173	166	**114**	352
`cmc`	396	428	**281**	653
`diabetes`	83	96	**67**	81
`ecoli`	**42**	59	63	325
`heart`	213	256	129	**40**
`hepatitis`	348	433	254	**33**
`ionosphere`	3995	5716	1849	**105**
`iris`	**14**	22	17	60
`kr-vs-kp`	47422	63283	18906	**1545**
`labor`	41	75	62	**29**
`mushroom`	25730	93243	13727	**3792**
`nursery`	2481	2087	**1285**	17293
`page-block`	1960	4760	**1443**	2763
`post-op`	**29**	58	37	61
`segment`	4875	14380	**2343**	2863
`soybeam`	8425	4374	4383	**2651**
`spam-base`	217476	79629796	39136	**8311**
`vehicle`	1557	15215	1428	**653**
`vote`	573	847	389	**49**
`vowel`	**792**	8088	994	1582
`waveform`	10810	6131	**3815**	10478
`wine`	144	148	122	**66**
`zoo`	**157**	166	217	303

terms of both complexity and learning time. Moreover, both *BESTlogL* and *TRHESHOLDlogL* have the highest learning times, which indicates that their overload for determining when and what to re-use is not justified by the learning time. As part of future work, we plan to investigate other ways to evaluate candidate seed structures.

Table 4: Complexity and learning time for the different re-usability schemes averaged over all datasets.

	Complexity	Time
NOREUSE	2045	3303
BESTlogL	1920	3671
TRHESHOLDlogL	1925	3861
First	1915	3201
Intersection	**1899**	**3136**
Union	1945	3400

The underlying idea of reusability is to use the probability estimates from data-rich classes to improve the estimates for data-poor classes. In order to evaluate this idea we have selected a subset of the datasets in Table 1, all of which have an unbalanced class distribution. The results can be found in Table 5, which shows the absolute difference in accuracy between each of the proposed re-usability methods and the plain algorithm without re-usability. On average we can see that there is always an improvement, and to compare the methods we have carried out a one-tailed Wilcoxon's signed rank test with significance level 0.05. The critical values for these five methods are 0.03 for *BESTlogL*, 0.01 for *THRESHOLDlogL*, 0.02 for *First*, 0.07 for *Intersection*, and 0.13 for *Union*. From these values the experiments indicate that *BESTlogL*, *THRESHOLDlogL*, and *First* significantly improve the original algorithm's performance. The improvement is usually small, but we have to bear in mind that this improvement is typically over class values with few instances.

Table 5: Absolute difference in accuracy for each of the re-usability methods with respect to the plain learning algorithm without re-usability. 1=*BESTlogL*, 2=*THRESHOLDlogL*, 3=*First*, 4=*Intersection*, 5=*Union*

	1	2	3	4	5
`autos`	1.17	1.07	1.07	-0.20	0.87
`balance`	0.00	0.00	-0.03	-0.03	0.00
`breast-cancer`	0.28	0.28	0.28	0.28	0.28
`breast-w`	0.00	0.00	0.00	0.00	0.00
`car`	0.00	0.00	0.00	0.00	0.00
`cmc`	0.19	0.20	0.20	0.11	0.29
`diabetes`	0.03	0.03	0.08	0.08	0.08
`ecoli`	0.00	0.00	0.00	0.00	0.00
`hepatitis`	0.26	0.26	0.65	0.65	0.65
`ionosphere`	-0.11	-0.11	-0.17	-0.17	-0.17
`nursery`	0.00	0.00	0.00	0.00	-0.58
`page-block`	0.04	0.04	0.05	0.04	0.05
`post-op`	0.00	0.00	0.00	0.00	0.00
`soybeam`	-0.09	0.03	-0.03	0.06	-0.70
`spam-base`	0.36	0.36	0.13	0.13	0.13
`vote`	0.09	0.09	0.23	0.23	0.23
`zoo`	0.00	0.00	0.00	0.00	0.20
average	0.13	0.13	0.14	0.07	0.08

In order to investigate this aspect further we can consider the confusion matrices for the two datasets `cmc` and `hepatitis`. For each dataset we perform learning with re-usability (using the *First* method) and without re-usability (see Tables 6 and 7); for *BESTlogL* and *THRESHOLDlogL* we obtain the same behavior. In these matrices we see an improvement for the class value with fewer instances, which corresponds to the second column in `cmc` and the first column in `hepatitis`. These states represent 23% and 21% of the instances, respectively, and in both cases the improvement obtained with re-usability is 3% for that state.

The impact of these results can be illustrated by considering e.g. medical diagnosis, where the number of cases with people being sick is typically much smaller than the number of cases with healthy people. A false negative for a person being sick means that she will not be given a treatment for her illness (possibly with disastrous consequences). With re-usability we can reduce this mis-classification rate, which, in situations like the medical example above, can have significant consequence.

Table 6: Confusion matrices without re-usability (a) and using with re-usability using the *First* method (b) on the cmc dataset.

	0	1	2
0	363.8	72.0	127.2
1	82.8	150.2	112.8
2	182.4	110.8	271.0

(a)

	0	1	2
0	360.4	66.4	118.8
1	87.6	159.4	124.0
2	181.0	107.2	268.2

(b)

Table 7: Confusion matrices without re-usability (a) and with re-usability using the *First* method (b) on the hepatitis dataset.

	0	1
0	21.8	11.8
1	10.2	111.2

(a)

	0	1
0	22.6	11.6
1	9.4	111.4

(b)

In comparison with the SVM classifier (see Table 8) we see a significant difference: the SVM classifier has the best average accuracy, but, in these cases, it has difficulties with class values having few instances. This behaviour can be expected by taking into account how this classifier is built. Nonetheless, if we instead look at the MultiBN or kDB classifiers, we also observe (results not included) that the proposed classifier obtains better results for states with poor representation of instances.

5 Conclusions and Future Work

In this paper we have presented a probabilistic classifier based on a class mixture of dependency networks. In addition to supporting fast learning algorithms, the proposed classifier allows for intermediate results to be reused across classes, thereby obtaining potential computational savings as well as improving the robustness for data scarce classes. We have proposed strategies for deciding on what and when to reuse. However, while the preliminary results indicate that the proposed strategies can improve classification accuracy, they also indicate that a simple uninformed strategy achieves better learning time results than more elaborate strategies. Measured in terms of the re-usability percentage, preliminary results indicate that the heuristic *BESTlogL* obtains the best results; for this strategy, 35% of the seeded variables also appear in the final Markov blankets (averaged over all the datasets). Designing other heuristic functions for guiding re-usability is a topic for future research. Here the aim is to find strategies that give a more balanced trade-off between accuracy improvements and learning time overhead.

Table 8: Confusion matrices for SVM with cmc (a) and hepatitis (b) datasets.

	0	1	2
0	387.0	81.8	139.8
1	54.8	95.6	59.0
2	187.2	155.6	312.2

(a)

	0	1
0	19.0	9.4
1	13.0	113.6

(b)

As part of future work we also plan to conduct more extensive re-usability experiments on larger datasets. Medical datasets, in particular, can be of interest, since they typically have hundreds of variables and relatively few instances. Another issue for future work is how to establish a good class ordering. One may, for example, be able to exploit the natural class ordering found in ordinal variables, i.e., we may expect that class values close to each other induce similar dependency structures over the attributes.

Acknowledgments

This work has been partially supported by Spanish Ministerio de Educación y Ciencia (project TIN2007-67418-C03-01); Junta de Comunidades de Castilla-La Mancha (project PBI-

08-048) and FEDER funds.

We would like to thank José Manuel Peña for useful comments and suggestions for earlier versions of this paper. We would also like to thank the anonymous reviewers for their constructive comments.

References

D. Aha and D. Kibler. 1991. Instance-based learning algorithms. *Machine Learning*, 6:37–66.

A. Asuncion and D.J. Newman. 2007. UCI machine learning repository.

C.M. Bishop. 1995. *Neural Networks for Pattern Recognition.* Oxford University Press.

D.M. Chickering. 2002. Optimal Structure Identification With Greedy Search. *Journal of Machine Learning Research*, 3:507–554.

G.F. Cooper and E. Herskovits. 1992. A Bayesian Method for the Induction of Probabilistic Networks from data. *Machine Learning*, 9(4):309–347.

Elvira Consortium. 2002. Elvira: An Environment for Creating and Using Probabilistic Graphical Models. In *Proceedings of the Fist European Workshop on Probabilistic Graphical Models*, pages 222–230.

N. Friedman, D. Geiger, and M. Goldszmidt. 1997. Bayesian Network Classifiers. *Machine Learning*, 29(2-3):131–163.

J.A. Gámez, J.L. Mateo, and J.M. Puerta. 2006. Dependency networks based classifiers: learning models by using independence test. In *Third European Workshop on Probabilistica Graphical Models (PGM06)*, pages 115–122.

J.A. Gámez, J.L. Mateo, and J.M. Puerta. 2008a. Improved EDNA (Estimation of Dependency Networks Algorithm) Using Combining Function with Bivariate Probability Distributions. In *Genetic and Evolutionary Computation Conference (Gecco08).*

J.A. Gámez, J.L. Mateo, and J.M. Puerta. 2008b. Towards consistency in general dependency networks. Technical Report DIAB-08-04-1, Computing Systems Deparment, University of Castilla-La Mancha.

D. Geiger and D. Heckerman. 1996. Knowledge representation and inference in similarity networks and bayesian multinets. *Artificial Intelligence*, 82:45–74.

S. Geman and D. Geman. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:147–156.

D. Heckerman, D.M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. 2000. Dependency networks for inference, collaborative filtering and data visualization. *Journal of Machine Learning Research*, 1:49–75.

P. Langley, W. Iba, and K. Thompson. 1992. An Analisys of bayesian Classifiers. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 223–228.

J.M. Peña, R. Nilsson, J. Björkegren, and J. Tegnér. 2007. Towards scalable and data efficient learning of markov boundaries. *International Journal of Approximate Reasoning*, 45(2):211–232.

J. Platt, 1999. *Advances in Kernel Methods - Support Vector Learning*, chapter Fast Training of Support Vector Machines using Sequential Minimal Optimization, pages 185–208. MIT Press.

R. Quinlan. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann.

M. Sahami. 1996. Learning Limited Dependence Bayesian Classifiers. In *Second International Conference on Knowledge Discovery in Databases*, pages 335–338.

G. Schwarz. 1978. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464.

P. Spirtes, C.N. Glymour, and R. Scheines. 2001. *Causation, Prediction, and Search.* MIT Press.

I. Tsamardinos, C.F. Aliferis, and A. Statnikov. 2003. Algorithms for large scale markov blanket discovery. In *Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference FLAIRS 2003.*

I.H. Witten and E. Frank. 2005. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2nd edition edition.