Qualified Probabilistic Predictions Using Graphical Models

Zhiyuan Luo and Alex Gammerman

Computer Learning Research Centre,
Royal Holloway, University of London,
Egham, Surrey TW20 0EX, UK
{zhiyuan, alex}@cs.rhul.ac.uk

Abstract. We consider probabilistic predictions using graphical models and describe a newly developed method, fully conditional Venn predictor (FCVP). FCVP can provide upper and lower bounds for the conditional probability associated with each predicted label. Empirical results confirm that FCVP can give well-calibrated predictions in online learning mode. Experimental results also show the prediction performance of FCVP is good in both the online and the offline learning setting without making any additional assumptions, apart from i.i.d.

1 Introduction

We are interested in making probabilistic predictions about a sequence of examples z_1, z_2, …, z_n. Each example z_i consists of an object x_i and its label y_i. The objects are elements of an object space X and the labels are elements of a finite label space Y. The example space Z can be defined as $Z = X \times Y$. It is assumed that the example sequence is generated according to an i.i.d. (independently and identically distributed) probability distribution P in Z^n.

Suppose that the label space Y is enumerated for all possible classification labels 1, 2, ..., $|Y|$. The learner $\varGamma$ is a function on a finite sample of n training examples $(z_1, z_2, ..., z_n) \in Z^n$ that makes a prediction for a new object $x_{n+1} \in X$

$$\varGamma : \ Z^n \times X \to [0,1]^{|Y|}. \tag{1}$$

Probability forecasting estimates the conditional probability of a possible label given an observed object. For each new object x_{n+1} (with true label y_{n+1} withheld from the learner), a set of predicted conditional probabilities for each possible labels are produced. In the online learning setting, examples are presented one by one. The learner $\varGamma$ takes object x_i, predicts $\hat{y_i}$, and then gets a feedback y_i. The new example $z_i = (x_i, y_i)$ is then included in the training set for the next trial. In the offline setting, the learner $\varGamma$ is given a training set $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ to predict on a test set $x_{n+1}, x_{n+2}, ..., x_{n+k}$.

This paper considers probabilistic predictions by using graphical models where examples are structured and more importantly, the data generating probability distribution P can be decomposed [4]. Firstly, we briefly discuss the

Bayesian belief network approach to probabilistic predictions and a technique called sequential learning to represent and update imprecision of conditional probabilities in the light of new cases [7]. Then we present a newly developed approach, called Venn probability machine, to probabilistic predictions [8] . In particular, we discuss fully conditional Venn predictor (FCVP) designed for the graphical models and its implementation. Finally, experiments are carried out on the simulated datasets to evaluate the FCVP. The empirical results confirm that predictions of FCVP are well-calibrated, in the sense that the error probability intervals produced by the FCVP bound the number of prediction errors. The experimental results demonstrate the performance of FCVP is good.

2 Bayesian Belief Networks

Bayesian belief networks are graphical knowledge representations. A Bayesian belief network can be represented as a pair (G, P). Qualitative knowledge G is a directed acyclic graph where the nodes V of the graph G are random variables. In this paper, we only consider the nodes V that take a finite set of values. The graph G is a representation of quantitative knowledge P that factorises in the form

$$P(V) = \prod_{v_i \in V} P(v_i \mid pa(v_i)), \tag{2}$$

where $pa(v_i)$ is a set of parent nodes of v_i in G.

Various algorithms exist to exploit and take full advantage of the independence relationships embodied in the network and efficient evidence propagation algorithms have been developed [1]. One of these approaches is called junction tree algorithm [3]. The junction trees are tree-like data structures whose vertices are labelled by cliques and whose edges labelled by separator sets form by intersection of two cliques on either side. Given a Bayesian belief network, a junction tree can be obtained [1]. This is done by (1) constructing an undirected graph called the moral graph from the Bayesian belief network; (2) selectively adding arcs to the moral graph to form a triangulated graph; (3) identifying the maximal cliques from the triangulated graph; (4) building the junction tree, starting with cliques as the nodes, where each link between two cliques is labelled by a separator. It has been shown that the joint probability distribution $P(V)$ in a junction tree can be represented as

$$P(V) = \frac{\prod_{c_i \in C} \Psi(c_i)}{\prod_{s_i \in S} \Psi(s_i)}, \tag{3}$$

where Ψ indicates the potential function on the cliques (C) and separators (S) which takes non-negative values. Note that $\Psi(c_i) \propto P(c_i)$, $\Psi(s_i) \propto P(s_i)$ for $c_i \in C$ and $s_i \in S$.

The junction tree can be used for efficient inference. When evidence arrives in a network, it is first absorbed into the junction tree. Then message passing protocol is used to propagate the evidence. The marginal distribution of a variable, conditional on some evidence can be found by local computation on the junction tree [1].

3 Parameter Learning

So far, we have assumed that the conditional probability tables in (2) for a given Bayesian belief network can be specified precisely. However, this assumption may not be realistic. The conditional probabilities derived from subjective assessments or specific dataset are subject to inevitable imprecision. The goal of parameter learning is to revise conditional probabilities for a given network topology as new cases arrive [5]. One such parameter learning method was proposed by Spiegelhalter and Lauritzen [7], namely sequential learning, and we follow here their approach.

The basic idea of sequential learning is to represent the imprecision of these conditional probabilities explicitly as parameters θ. In a Bayesian belief network setting, it is reasonable to partition the space θ into a set of small spaces θ_i concerning each node v_i and assume θ_i is independent a prior to each other. That is, $P(\theta) = \prod_{i=1}^{n} P(\theta_i)$, where n is the number of nodes in V. Each conditional probability table attached to node v_i is determined uniquely by the parameter θ_i. $P(V \mid \theta)$ can be written as follows due to the conditional independence reflected in the model $P(V \mid \theta) = \prod_{v_i \in V} P(v_i \mid pa(v_i), \theta_i)$. The joint probability distribution on V and θ is then calculated as $P(V, \theta) = \prod_{v_i \in V} P(v_i \mid pa(v_i), \theta_i)P(\theta_i)$. It is clear that the parameter θ_i may be considered as another parent node of v_i in the network. These θ_i parameters represent summary of past cases.

Given the network structure, $P(v_i \mid pa(v_i), \theta_i)$ and $P(\theta_i)$ specified for each node v_i, the task now is to calculate the posterior distribution $P(\theta \mid e)$ when an instantiation of variables e is obtained. Three basic operations are involved: dissemination of experience, propagation of evidence and retrieval of new information. The procedure can be repeated in the same manner as more instantiation of variables arrive. Different assumptions are made for different operations to simplify the computation. Firstly, independence of each parameter θ_i over node v_i is assumed. This allows the dissemination operation to be carried out locally. For each variable v_i, we apply

$$P(v_i \mid pa(v_i)) = \int P(v_i \mid pa(v_i), \theta_i)P(\theta_i)d\theta_i \tag{4}$$

to get the means of the conditional probabilities $P(v_i \mid pa(v_i), \theta_i)$ for each node v_i. Secondly, the current 'marginal probabilities' are used to initialise the standard evidence propagation methods, such as the junction tree algorithm described before. Finally, in the retrieval operation, the following calculation is performed:

$$P(\theta_i \mid e) = \sum_{v_i, pa(v_i)} P(\theta_i \mid v_i, pa(v_i), e)P(v_i, pa(v_i) \mid e). \tag{5}$$

Since θ_i is conditional independent of e given v_i and $pa(v_i)$, thus

$$P(\theta_i \mid e) = \sum_{v_i, pa(v_i)} P(\theta_i \mid v_i, pa(v_i)) P(v_i, pa(v_i) \mid e). \tag{6}$$

It is clear that there is a mixture distribution for the parameter θ_i if v_i and $pa(v_i)$ are not observed in the new case e. To simplify the retrieval operation, it is assumed that the individual parameter θ_i for node v_i can be further partitioned and is conditional on each possible configuration of its parent set $pa(v_i)$. Therefore, each conditional probability distribution under a configuration of the parent nodes can be individually updated in the light of e.

In this paper, we model θ_i as Dirichlet distributions and update these parameter θ_i with complete new cases. In particular, we use a Dirichlet prior distribution as a conjugate form and the mean of the Dirichlet distribution is used as the estimation of $P(v_i \mid pa(v_i))$. The Dirichlet has a simple interpretation in terms of pseudo counts. Both dissemination and retrieval operations are straightforward with completed data. Note that we use BDeu prior (likelihood equivalent uniform Bayesian Dirichlet) in our experiments [2].

4 Venn Probability Machines

The Venn Probability Machine (VPM) is a simple yet powerful framework for probability forecasts [8]. Unlike many conventional probabilistic prediction approaches, VPM gives several probability distributions for the predicted label. These probability distributions are close to each other so that probabilistic prediction made by the VPM will be practically useful. Therefore, VPM is a type of multiprobability predictor.

The basic idea behind the VPM is as follows. Given the training example sequence $(x_1, y_1), ..., (x_{n-1}, y_{n-1})$ and a new test example x_n, we consider each possible completion for x_n. For each possible completion $y \in |Y|$, we have n examples $(x_1, y_1), ..., (x_n, y)$ and then divide all the examples into a number of categories. It is required that such division of examples is independent of the order of examples. Many existing supervised machine learning algorithms can be used to perform the division. For example, a simple way to divide the examples into different categories is based on the 1-nearest neighbour algorithm. Two examples are assigned to the same category if their nearest neighbours have the same label. Taking the category T containing the example (x_n, y), we can estimate the relative frequence of examples labelled j in T as

$$A_{y,j} = \frac{|\{(x', y') \in T : y' = j\}|}{|T|}. \tag{7}$$

Those relative frequencies obtained in (7) are interpreted as empirical probability distributions for the predicted labels.

Having considered all possible completion for x_n, we have a $|Y| \times |Y|$ Venn probability matrix A. The rows of the matrix A represent the frequency count of each class label in the training examples set which have the same type as the new test example. The minimum and maximum frequency counts within each row give us the lower and upper bounds for conditional probabilities of possible labels given x_n. VPM predicts the label for the new test example using the respective column which contains the largest of minimum entries.

VPM is different from Bayesian learning theory and PAC learning [8]. Unlike Bayesian learning theory, VPM requires no empirical justification for probabilities. In contrast with PAC learning which aims to estimate the unconditional probability of error, VPM tries to estimate the conditional distribution of the label given the new object. A useful property of VPM is its self-calibration nature in the online learning setting. It has been proved that the probability intervals generated by the VPM is well-calibrated in the sense that the VPM can bound the true conditional probability for each new test object in an online test [8]. Using the VPM's upper and lower intervals for conditional probabilities, we can estimate the bounds for the number of errors made.

4.1 Online Compression Models

The Venn probability machine can be generalised to online compression models which can summarise statistical information efficiently and perform lossy compression [9]. Formally, an online compression model (OCM) is defined as $M = (\Sigma, \Box, Z, (F_n), (B_n))$ where

- Σ is a measurable space called summary space containing summaries σ.
- $\Box \in \Sigma$ is special summary called the empty summary and we set $\sigma_0 = \Box$.
- Z is a measurable space containing the examples z_i
- $F_n, n = 1, 2, ...$ are measurable functions of the type $\Sigma \times Z \to \Sigma$. F_n are called forward functions that allow us to update summary σ_{n-1} to σ_n given the example z_n in an online fashion. Therefore, we have $F_n(\sigma_{n-1}, z_n) = \sigma_n$.
- $B_n, n = 1, 2, ...$ are backward kernels of the type $\Sigma \hookrightarrow \Sigma \times Z$. It is required that B_n are inverse to F_n in the sense that $B_n(F_n^{-1}(\sigma) \mid \sigma) = 1$ for each $\sigma \in F_n(\Sigma \times Z)$. B_n map $\sigma \in \Sigma$ to probability distributions in Z.

Intuitively, the summaries σ can be considered as sufficient statistics for the observed example sequence. For example, the summaries σ can be the number of ones in a binary sequence generated by Bernoulli models. We start with the empty summary $\Box$ which indicates that we do not have information about the data, i.e. $\sigma_0 = \Box$. When the first example z_1 arrives, we update our summary to σ_1 using $F_1(\sigma_0, z_1)$. We update our summary to $\sigma_2 = F_2(\sigma_1, z_2)$ given the second example z_2, so on and so forth. Basically, forward functions F_n extract all useful information from the observed example sequence and perform lossy compression. It is important that the summaries are calculated in an online fashion, i.e. F_n updates σ_{n-1} to σ_n given z_n. On the other hand, backward kernels B_n perform decompression and allow us to find the conditional distribution of a particular example sequence $(z_1, z_2, ..., z_n)$ given the summary σ_n. This is done iteratively. Given σ_n, we generate (σ_{n-1}, z_n) from the distribution $B_n(\sigma_n)$. Then we generate (σ_{n-2}, z_{n-1}) from $B_{n-1}(\sigma_{n-1})$, so on and so forth.

VPM can be generalised to an OCM. When we have seen $n-1$ examples, the OCM summaries these examples and has σ_{n-1}. Given a test example x_n, we can try all possible completion $y \in |Y|$ and have $\sigma_n = F_n(\sigma_{n-1}, (x_n, y))$. We specify a partition A_n and use it to divide the set $F_n^{-1} \subseteq \Sigma_{n-1} \times Z$ into a number categories. This is done by assigning (σ', z') and (σ'', z'') to the same category if and only if $A_n(\sigma', z') = A_n(\sigma'', z'')$ where $A_n(\sigma, z)$ represents the element of the partition A_n containing (σ, z). Consider the category $T = A_n(\sigma_{n-1}, (x_n, y))$, we estimate the probability distribution of the label y as

$$p_y = \frac{B_n(\{(\sigma^*, (x^*, y^*)) \in T : y^* = y\} | \sigma_n)}{B_n(T|\sigma_n)}. \tag{8}$$

4.2 Fully Conditional Venn Predictor

When examples z_i are generated from Bayesian belief networks, an explicit OCM can be defined and an efficient Venn predictor called fully conditional Venn predictor (FCVP) can be constructed. The junction tree constructed from the Bayesian belief network can serve as a basis for efficient summaries of observed data sequence. As discussed earlier, a junction tree is a graphical data structure consisting of cliques and separators. For convenience, we refer to both the cliques and the separators of a junction tree as clusters. We can associate a table with each cluster where the index of the table is determined by the configurations on the cluster and each entry of the table is a non-negative integer. Obviously, the number of entries in the table on a cluster depends on the number of possible configurations of the variables in the cluster. The table size is defined as the sum of all entries. All the tables on the clusters form a table set for a junction tree. We are only interested in table sets all of whose tables have the same size.

We define an example z is consistent with a configuration of a cluster E if the configuration coincides with the restriction $z|_E$ of z to E. If we assign the number of past examples which are consistent with each configuration of the clusters to appropriate entries of the tables on the clusters, we have a table set σ generated by the example sequence. The length of each example sequence generating σ will be equal to the table size of σ. The number of example sequences generating a table set σ is specified as $\#\sigma$. One of possible operations on the table set σ is to query the number assigned to a configuration of a cluster u, which is defined as the σ-count of the configuration. For example, $\sigma_u((x_i, y_i))$ will return the count assigned by the table set σ to a configuration of a cluster u which is consistent with the example (x_i, y_i).

An OCM $M = (\Sigma, \Box, Z, (F_n), (B_n))$ can be defined for the junction tree model as follows:

- Summary space Σ consists of summaries defined by the consistent table sets σ.
- Empty summary $\Box$ is a table set with size 0.
- Z consists of the set of all examples. An example z_i is simply a particular configuration on V.

– Given an example z_n, the forward function F_n will update the table set by adding 1 to the entries of the table set which are consistent with z_n.
– An example z is consistent with a summary σ if the σ-count of each configuration that is consistent with z is positive. For the size of $\sigma = n$, backward kernels B_n can be defined as

$$B_n(\{(\sigma \downarrow z, z)\} \mid \sigma) = \frac{\#(\sigma \downarrow z)}{\#\sigma} \tag{9}$$

where $\sigma \downarrow z$ means subtracting 1 from the σ-count of any configuration that is consistent with z.

The junction tree has the property that for each pair U, V of cliques with intersection S, all cliques on the path between U and V contain S. For a table set σ defined on a junction tree, it is consistent if and only if: (1) each table in σ has the same size, and (2) if clusters E_1 and E_2 intersect, the marginalisations of their tables to $E_1 \cap E_2$ coincide. Given a summary σ of size n in the junction tree model, the number of example sequences of length n that are consistent with the table set σ is

$$\#\sigma = \frac{n! \prod_{s \in S} \mathrm{fp}_\sigma(s)}{\prod_{c \in C} \mathrm{fp}_\sigma(c)} \tag{10}$$

where $\mathrm{fp}_\sigma(E)$ is the factorial-product of a cluster E in a summary σ and $\mathrm{fp}_\sigma(E) = \prod_{a \in configurations\ of\ E} \sigma_E(a)!$. It has been proved in [9] that given the summary σ_n of the first n examples, the conditional probability that $z_n = (x_n, y)$ based on maximum likelihood estimation is

$$\frac{\prod_{c \in C} \sigma_c((x_n, y))}{n \prod_{s \in S} \sigma_s((x_n, y))}. \tag{11}$$

Note that the ratio defined in (11) is set to 0 if any of the factors in the numerator or denominator is 0; in this case $z_n = (x_n, y)$ is not consistent with the summary σ.

Having specified the OCM for junction tree model, we are now ready to describe the Venn predictor. When a junction tree OCM has one or more variables as labels, a Venn predictor called fully conditional Venn predictor can be defined by determining partition A_n in which $A_n(\sigma, z)$ consists of all (σ, z') for which z and z' match on all non-label variables. Once the partition A_n is established, the VPM can make predictions and provide upper and lower bounds for the conditional probability associated with each predicted label. The FCVP algorithm in the online learning mode is presented below.

5 Experiments

5.1 Dataset

The well-known 'Visit to Asia' example is used for our experiments [4]. There are 8 binary variables in this example, see Figure 1. For the online learning experiments, three datasets with 1000, 2000 and 5000 examples were randomly

Algorithm 1. Fully Conditional Venn Predictor

```
Require: a list of variables and the values each variable can take
Require: junction tree with its cliques $C$ and separators $S$
Require: object space $X$, label space $Y$ and target label space $Y^t \subseteq Y$
Require: $N$ examples $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$
  $\sigma_0 = \Box$
  for $n = 1$ to $N$ do
    get $x_n \in X$ of example $(x_n, y_n)$
    for $y = 1$ to $\mid Y \mid$ do
      $\sigma = F_n(\sigma_{n-1}, (x_n, y))$
      for $y' = 1$ to $\mid Y \mid$ do
        $A_{y,y'} = \frac{\prod_{c \in C} \sigma_c((x_n, y'))}{\prod_{s \in S} \sigma_s((x_n, y'))}$ {$A_{y,y'}$ is set to 0 if any of the factors in the numerator
        or denominator is 0}
      end for
    end for
    $A_{y,y'} = \frac{A_{y,y'}}{\sum_{y'} A_{y,y'}}$
    for $y^t = 1$ to $\mid Y^t \mid$ do
      $A_{y,y^t} = \sum_{y' \setminus y^t} A_{y,y'}$
    end for
    predict $\hat{y}^t = \arg\max_{y^t \in Y^t}(\min_{y \in Y} A_{y,y^t})$
    output predicted probability interval for $\hat{y}^t$ as $[\min_y A_{y,\hat{y}^t}, \max_y A_{y,\hat{y}^t}]$
    get $y_n \in Y$ of example $(x_n, y_n)$
    $\sigma_n = F_n(\sigma_{n-1}, (x_n, y_n))$
  end for
```

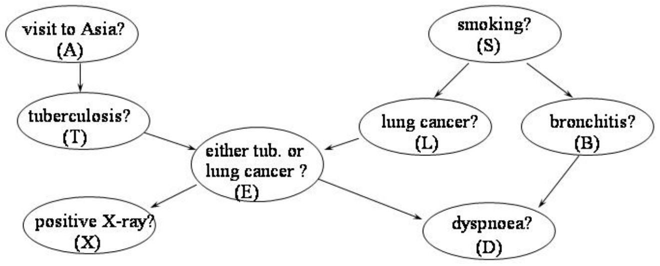

Fig. 1. 'Visit to Asia' example

generated using the network structure and the associated conditional probabilities. For the offline learning experiments, another three datasets were randomly generated: (training size=3000, test size=1000), (training size=2000, test size=2000) and (training size=1000, test size=3000).

5.2 Methods

For experiment purpose, we assume that we have evidence on the variables A, S, X and D (patient history and diagnostic tests) and would like to predict the conditional probabilities of the variables B, T, E and L (medical diagnosis), respectively, given these observations.

Fully conditional Venn predictor (FCVP) was implemented using Bayes net toolbox (BNT) for Matlab [6]. In order to evaluate prediction performance of FCVP, we also implemented the junction tree algorithm and the sequential learning algorithm using BNT. The junction tree algorithm is specified with precise conditional probabilities, i.e. it has the same conditional probabilities as those used to generate the datasets in the previous section. On the other hand, both the FCVP and sequential learning algorithm will have to learn these conditional probabilities from the past examples. The implemented systems, namely FCVP, junction tree algorithm (JT) and sequential learning (SL) are evaluated on the datasets generated in the previous section. The conditional probabilities were calculated on each of the label variables {B, T, L, E} and predictions made.

The junction tree algorithm and sequential learning produce a single probability distribution on a label and predict the class label with the largest associated conditional probability $\hat{y}_i = arg\,\max_{y \in Y} \hat{p}_{i,y}$ given the test example x_i. On the other hand, FCVP outputs an interval for the probability that the predicted label is correct. If the interval is $[a_i, b_i]$ at the trial i, the complementary interval $[1 - b_i, 1 - a_i]$ is the error probability interval.

If more than one label has the largest associated conditional probability, we have multiple predictions. A prediction is correct if the true label for the example matches the predicted label. Otherwise it is an error.

5.3 Results

The predictions made by FCVP on the variables B, T, E and L were obtained in the experiments. Figure 2 shows the performance of FCVP on the variable B in the online learning setting on the datasets of 1000 examples. In this figure, the cumulative lower and upper probability error bounds, the prediction errors and multiple predictions are presented. These plots confirm that the error probability intervals generated by FCVP are well-calibrated. FCVP can produce a multiple prediction in the sense that the predicted probability interval for each label is [0, 1]. Note that the total number of multiple predictions is small and the multiple predictions occur at the beginning of the trials when some combination is observed for the first time. For example, 7 multiple predictions were observed on a dataset of 1000 examples. Similar prediction behaviour was observed for the variables T, L and E.

In our experiments, the three algorithms were tested and evaluated on the same datasets. Figure 3 displays the comparative performance results on B on 1000 examples in terms of the number of prediction errors. It is clear that the prediction performance of FCVP is very similar to the one produced by the junction tree algorithm with precise conditional probabilities and much superior to that of the sequential learning. Table 1 presents the summaries of results on different datasets in terms of the cumulative number of prediction errors and the number of multiple predictions. For example, the junction tree algorithm made 31 prediction errors on the variable B over 1000 examples, see Table 1. On the

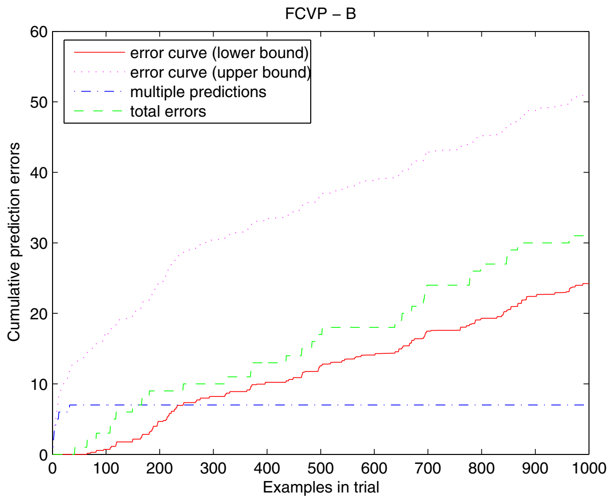

Fig. 2. FCVP results (1000 examples) - online learning mode

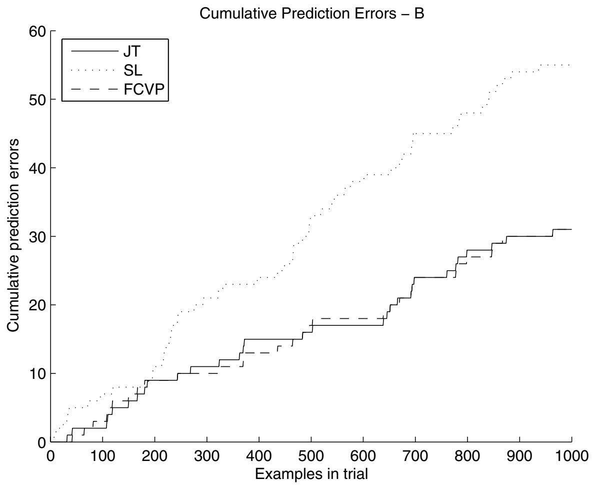

Fig. 3. Comparative performance (1000 examples) - online learning mode

other hand, the prediction errors made by the sequential learning method and FCVP were 55 and 31, respectively.

Three experiments were carried out to compare the performance of FCVP with the junction tree algorithm with precise conditional probabilities and the sequential learning in offline learning setting. The results are shown in Table 2. These results demonstrate that FCVP achieves similar performance with the junction tree algorithm and outperforms the sequential learning method in almost all the experiments.

Table 1. Comparative performance - online learning mode

No. of Examples		1000		2000		5000	
Method	Label	#errs	#multi. preds	#errs	#multi. preds	#errs	#multi. preds
JT	B	31	0	73	0	144	0
	T	6	0	25	0	47	0
	L	222	0	427	0	1068	0
	E	30	0	76	0	155	0
SL	B	55	2	103	3	270	2
	T	7	1	25	1	45	1
	L	368	1	705	2	1794	1
	E	39	1	84	1	162	1
FCVP	B	31	7	69	7	143	6
	T	8	7	25	7	45	6
	L	221	7	434	7	1071	6
	E	33	7	77	7	163	6

Table 2. Comparative performance – offline learning mode

Dataset		training set=3000, test set=1000		training set=2000, test set=2000		training set=1000, test set=3000	
Method	Label	#errs	#multi. preds	#errs	#multi. preds	#errs	#multi. preds
JT	B	43	0	68	0	99	0
	T	12	0	19	0	37	0
	L	217	0	437	0	637	0
	E	40	0	77	0	104	0
SL	B	50	0	102	0	182	0
	T	12	0	19	0	37	0
	L	348	0	676	0	1069	0
	E	43	0	71	0	115	0
FCVP	B	38	0	65	0	95	0
	T	12	0	19	0	37	0
	L	221	0	437	0	648	0
	E	40	0	76	0	104	0

6 Conclusions

We present a newly developed probabilistic prediction method using graphical models, fully conditional Venn predictor (FCVP). FCVP can provide well-calibrated probabilistic predictions in the online learning setting. Unlike the sequential learning method, FCVP makes no additional independence assumptions about probability distributions associated with the graphical structure. Empirical results have shown FCVP can achieve good prediction performance over the sequential learning method in both the online and offline learning setting.

Acknowledgements. We thank Volodya Vovk and Tony Bellotti for their discussions and comments. Financial support has been received from the following bodies: MRC through grant S505/65 and Royal Society through grant "Efficient randomness testing of random and pseudorandom number generators".

References

[1] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter: Probabilistic Networks and Expert Systems. Statistics for Engineering and Information Science. Springer-Verlag (1999).

[2] D. Heckerman and D. Geiger: Likelihoods and parameter priors for bayesian networks. Technical Report MSR-TR-95-54, Microsoft Research (1995).

[3] Finn V. Jensen: An introduction to Bayesian Networks. Taylor and Francis, London, UK (1996).

[4] S. L. Lauritzen and D. J. Spiegelhalter: Local computations with probabilities on graphical structures and their application to expert systems (with discussion). J. Royal Statist. Soc. **series B**, (50):157–224 (1988).

[5] Z. Luo and A. Gammerman: Parameter learning in Bayesian belief networks. Proceeding of IPMU'92, 25–28 (1992).

[6] K. Murphy: The Bayes Net Toolbox for Matlab. Computing Science and Statistics. **33** (2001).

[7] D. J. Spiegelhalter and S. L. Lauritzen: Sequential updating of conditional probabilities on directed graphical structures. Networks, **20**(5):579–605 (1990).

[8] V. Vovk, G. Shafer, and I. Nouretdinov: Self-calibrating probability forecasting. In S. Thrun, L. Saul, and B. Schölkopf (ed.), Advances in Neural Information Processing Systems 16. MIT Press, Cambridge, MA (2004).

[9] V. Vovk, A. Gammerman, and G. Shafer: Algorithmic learning in a random world. Springer-Verlag, (*To appear*) (2005).