Compositional Noisy-Logical Learning

Alan Yuille YUILLE@STAT.UCLA.EDU
Department of Statistics, University of California, Los Angeles, CA 90095 USA

Songfeng Zheng SONGFENGZHENG@MISSOURISTATE.EDU
Department of Mathematics, Missouri State University, Springfield, MO 65897 USA

Abstract

We describe a new method for learning the conditional probability distribution of a binary-valued variable from labelled training examples. Our proposed Compositional Noisy-Logical Learning (CNLL) approach learns a noisy-logical distribution in a compositional manner. CNLL is an alternative to the well-known AdaBoost algorithm which performs coordinate descent on an alternative error measure. We describe two CNLL algorithms and test their performance compared to AdaBoost on two types of problem: (i) noisy-logical data (such as noisy exclusive-or), and (ii) four standard datasets from the UCI repository. Our results show that we outperform AdaBoost while using significantly fewer weak classifiers, thereby giving a more transparent classifier suitable for knowledge extraction.

1. Introduction

AdaBoost (Freund & Schapire, 1996) is one of the most influential machine learning algorithm for binary classification. It estimates a strong classifier from training data by combining a weighted sum of weak classifiers and can be formulated as coordinate descent on an upper bound of the training error. But despite its successes in terms of classification performance, AdaBoost is less effective at detecting underlying structure, or extracting knowledge, from the data. Knowledge extraction is important both for transfer learning and for debugging and understanding machine learning systems (Alpaydin, 2004).

There is a deep connection between AdaBoost and additive logistic regression (Friedman et al., 2000) where the conditional distribution of the output is expressed in exponential form in terms of statistics of the input. The statistics and their parameters of logistic regression correspond to the weak classifiers and their weights of AdaBoost. Friedman et al. prove theorems relating these approaches in the asymptotic limit (Friedman et al., 2000).

These results suggest that the effectiveness of AdaBoost for a specific application will depend on how well the conditional distribution for the application data can be approximated by an exponential distribution. If we can approximate the distribution *sparsely* (i.e., using an exponential distribution with a small number of statistics) then we expect that AdaBoost will give high classification performance *and* will perform knowledge extraction by specifying which statistics are important. But suppose that the distribution can only be approximated by an exponential model which uses a large number of statistics. In this case, AdaBoost may be successful for classification but will result in a strong classifier which is a combination of many weak classifiers and will fail to extract useful knowledge about the data.

We can make an analogy to harmonic analysis where the goal is to express functions in terms of a linear combination of basis functions, e.g, (Meyer, 2001). There are many choice of bases – Fourier series, Haar bases, wavelets, polynomials – which are complete in the sense that any function can be represented in terms of them (by weighted linear combination). But for any specific application it is usually desirable to use bases which can represent the functions *sparsely* so that a good approximation to the function is obtained by using a small number of basis functions. This gives insight into the application and can be thought of as knowledge extraction. Mathematicians have shown that certain bases are best for representing functions in specific functional classes (Meyer, 2001) in the sense

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009.

that fewer basis functions are required to get good approximations.

These considerations suggest that we design machine learning algorithms based on different representations of conditional distributions. Recently *noisy-logical distributions* were proposed (Yuille & Lu, 2008) as an alternative to exponential models (this paper did not contain a learning algorithm). These distributions are complete, in the sense that any conditional distribution can be expressed in noisy-logical form, and are built by composing elementary components which yield a structured representation for the distribution. Certain distributions – such as noisy-or and noisy-and-not (Pearl, 1988) – can be sparsely represented in noisy-logical form but are hard to represent sparsely by exponential distributions (and conversely). In general, noisy logical distributions are sparse if there is a (noisy) logical process generating the data.

This paper presents two Compositional Noisy-Logic Learning (CNLL) algorithms which perform classification by learning noisy-logical distributions. The hope is that CNLL algorithms will have better performance and yield sparser representations than AdaBoost for certain forms of classification problems. We emphasize that sparse representations can enable knowledge extraction which can yield many benefits including transfer learning. Sparse representations can also have practical benefits because they reduce the computation time (after learning) and could be useful for applications where computation power is limited.

Our experimental results using CNLL are very promising. Firstly, we consider applications where the data is generated by a noisy-logical process – for example, a noisy version of the exclusive-or problem. In these cases, CNLL is able to learn the underlying (noisy) logical process that generates the data and hence performs knowledge extraction. Secondly, we apply CNLL to four standard datasets from the UCI repository which are used to test AdaBoost and its variants. On these datasets we obtain significantly better performance than AdaBoost and also obtain sparse noisy-logical representations of the data.

2. Background: Noisy-Logical and Exponential Distributions

The noisy-logical distribution (Yuille & Lu, 2008) was proposed as a generalization of the noisy-or and noisy-and-not distributions (Pearl, 1988). These types of distributions are of interest to cognitive scientists since human performance on causal learning tasks can be successfully modeled by assuming that humans use noisy-or and noisy-and-not distributions (Cheng, 1997; Griffiths & Tenenbaum, 2005) and more complex noisy-logical distributions (Yuille & Lu, 2008).

A noisy-logical distribution represents the distribution $P(y|\vec{C})$ of a binary variable $y \in \{0,1\}$ conditioned on a binary vector $\vec{C} \in \{0,1\}^N$ in terms of composition of elementary distributions, see Fig. 1(left panel). Each elementary distribution is of form $P(H_i|\psi_i(\vec{C}),\omega_i)$ where $H_i \in \{0,1\}$ is a hidden variable, $\psi_i(\vec{C}) \in \{0,1\}$ is a *causal feature* of $\vec{C}$, and ω_i are the parameters of the distribution. The full distribution is obtained by setting the output y to be a (deterministic) logical combination of the hidden states $\{H_i\}$. This is formally expressed as:

$$P(y=1|\vec{C};\vec{\omega}) = \sum_{\vec{H}} \delta_{y,f(H_0,\cdots,H_{2^N-1})} \prod_{i=0}^{2^N-1} P(H_i=1|\psi_i(\vec{C});\omega_i), \quad (1)$$

where $f(H_0,\cdots,H_{2^N-1})$ is a logical (i.e. deterministic) function of the hidden variables $\{H_i\}$; $\delta_{y,f}=1$ if $y=f$ and 0 otherwise.

where $\psi_i(\vec{C}) \in [0,1]$ is a causal feature
$\vec{\omega}_i = (\alpha_i,\beta_i), \alpha_i \in [0,1], \beta_i \in [0,1]$ are weights
$P(H_i=1|\psi_i(\vec{C})=1)=\alpha_i, \quad P(H_i=1|\psi_i(\vec{C})=1)=\beta_i$
$f(H_0,\cdots,H_{2^N-1})$ is a logical function of $\{H_i\}$

It was shown (Yuille & Lu, 2008) that any conditional distribution on binary variables can be expressed in noisy-logical form. In particular, the standard noisy-or and noisy-and-not distributions (Pearl, 1988) can be obtained as special cases by setting $\vec{C}=(C_1,C_2)$, using causal features $\psi_1(\vec{C})=C_1, \psi_2(\vec{C})=C_2$, and letting $y=H_1 \vee H_2$ to obtain noisy-or and $y=H_1 \wedge \neg H_2$ to obtain noisy-and-not. In these cases, the noisy-logical distribution takes a simple form which gives insight into the structure of the data – in particular, by showing the logical relationship between the output y and the hidden states H_1, H_2.

More generally, we can express this relationship in Disjunctive Normal Form (DNF), for example, $y = (H_1 \wedge H_2 \wedge \cdots) \vee (H_3 \wedge H_4 \wedge \cdots) \vee \cdots$, see Fig. 1(right panel). We can think of this DNF as specifying a *logical process* underlying the data which is made noisy by the distributions $P(H_i|\psi_i,\omega_i)$. In other words, there is a deterministic rule for the data specified by the $\{H_i\}$ but we cannot observe that $\{H_i\}$ directly and instead must infer them from noisy observations $\{\psi_i(\vec{C})\}$.

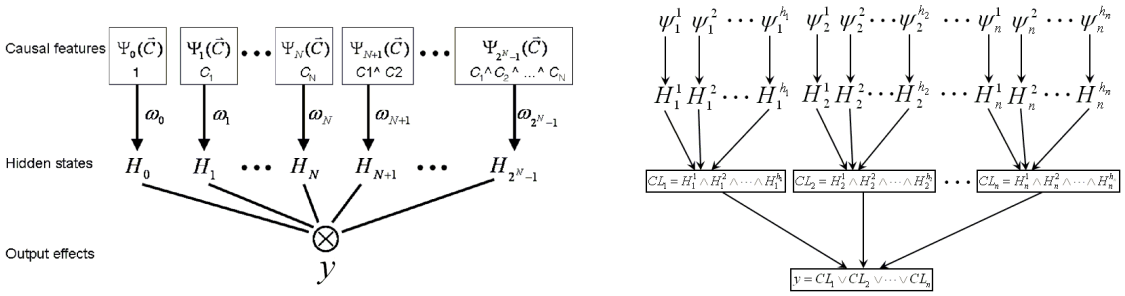


Figure 1. The basic Noisy-Logical Distribution (left panel). The DNF formulation of Noisy-Logical (right panel).

By contrast, additive linear regression (Friedman et al., 2000) is a standard way to represent distributions. We change notation so that the output variable $y \in \{-1, +1\}$ and the input is $\mathbf{x}$. This conditional distribution can be expressed in the form:

$$P(y|\mathbf{x}) = \frac{1}{Z[\mathbf{x}, \{\lambda_i\}]} \exp\{y \sum_i \lambda_i \psi_i(\mathbf{x})\}, \qquad (2)$$

where the $\{\psi_i(\mathbf{x})\}$ are *statistics*, the $\{\lambda_i\}$ are parameters, and $Z[\mathbf{x}, \{\lambda_i\}]$ is the normalization term.

The exponential distributions in Eq. (2) can be learnt from training data $\{(\mathbf{x}_\mu, y_\mu) : \mu = 1, \cdots, N\}$ ($y_\mu \in \{\pm 1\}$) by maximum likelihood (ML) to find $\lambda^* = \arg\max_\lambda \prod_{\mu=1}^N P(y_\mu|\mathbf{x}_\mu)$. This has several limitations: (i) performing ML is difficult because it requires evaluating the normalization term $Z[\mathbf{x}, \{\lambda_i\}]$; (ii) ML learning may not be optimal for learning a decision rule when only a finite number of training samples are available.

AdaBoost (Freund & Schapire, 1996) learns a strong classifier $H(\mathbf{x}) = \text{sign}\{\sum_{i=1}^M \lambda_i \psi_i(\mathbf{x})\}$ from training data without needing to evaluate the normalization term $Z[\mathbf{x}, \{\lambda_i\}]$, using a set of weak classifiers $\{\psi_i(\mathbf{x})\}(\psi_i(\mathbf{x}) \in \{\pm 1\})$. It classifies new data $\mathbf{x}$ as positive if $H(\mathbf{x}) \geq 0$ or as negative if $H(\mathbf{x}) < 0$. It can be formulated as coordinate descent on the function $F(\vec{\lambda}) = \sum_\mu \exp\{y_\mu \sum_i \lambda_i \psi_i(\mathbf{x}_\mu)\}$, where $\{\lambda_i\}$ are the components of $\vec{\lambda}$.

It can be shown (Friedman et al., 2000) that AdaBoost converges asymptotically to a strong classifier $H(\mathbf{x}) = \text{sign}\{\sum_i \lambda_i^* \psi_i(\mathbf{x})\}$ where the coefficients $\{\lambda_i^*\}$ are the maximum likelihood (ML) estimators $\lambda^* = \arg\max_\lambda P(\{y_\mu\}|\{\mathbf{x}_\mu\}, \lambda)$ of the linear additive regression model. Hence the strong classifier corresponds to the log-likelihood ratio test $\text{sign}\{\log \frac{P(y=1|\mathbf{x},\lambda^*)}{P(y=-1|\mathbf{x},\lambda^*)}\}$.

3. Compositional Noisy-Logical Learning (CNLL)

We now develop Compositional Noisy-Logical Learning (CNLL) as an alternative learning strategy to AdaBoost based on the noisy-logical distribution. We supply two CNLL algorithms in sections (3.1,3.3). They both proceed by minimizing a performance measure in a greedy manner (similar to AdaBoost).

We are given a set of training examples $\{(\mathbf{x}_\mu, y_\mu) : \mu \in 1, \cdots, N\}$, where $\mathbf{x}_\mu$ is the data vector and $y_\mu \in \{0, 1\}$ is the label. We specify a set of causal features $\{\psi_\rho(\mathbf{x})\}$ with $\psi_\rho(\mathbf{x}) \in \{0, 1\}$, where $\rho \in \Omega$ with Ω indexes all the possible causal features. We will specify the precise form of these causal features in the experimental section (4). Our default is to have a set of stump features $\{f_a(\mathbf{x})\}$ for which we determine a casual feature (weak classifier): $\psi_{(a,b,c)}(\mathbf{x}) = 1$ if $p_b \log \frac{P(y=1|f_a(\mathbf{x}))}{P(y=0|f_a(\mathbf{x}))} > T_c$, where $p_b \in \{\pm 1\}$ is parity and T_c is a threshold, and $\psi_{(a,b,c)}(\mathbf{x}) = 0$ otherwise. We also consider logical AND's and OR's of these causal features (e.g. $\psi_{(a,b,c)}(\mathbf{x}) \wedge \psi_{(a',b',c')}(\mathbf{x})$ and $\psi_{(a,b,c)}(\mathbf{x}) \vee \psi_{a',b',c'}(\mathbf{x})$). To simplify notation, we write all causal features as $\psi_\rho(\mathbf{x})$.

Both our CNNL algorithms construct will construct a noisy-logical distribution using the causal features as input. In this paper we use a variant of the noisy-logical form presented in (Yuille & Lu, 2008). In equation (1) we set $\omega_i = (\alpha_i, \beta_i)$ where $\alpha_i = P(H_i = 1|\psi_i(\mathbf{x}) = 1; \omega_i)$ and $\beta_i = P(H_i = 1|\psi_i(\mathbf{x}) = 0; \omega_i)$ ($\beta_i = 0 \; \forall i$ in (Yuille & Lu, 2008)).

The performance measure is the error E_R of the log-likelihood classifier $R(\mathbf{x}) = \text{sign}\{\log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}\}$. The false positive errors are weighted more than the false negative errors to allow for the different number of positive and negative examples in the datasets.

3.1. Basic CNLL: The First CNLL Algorithm

CNLL grows the distribution $P_t(y|\mathbf{x})$ by composition, see Fig. 2. This involves adding a new hidden unit H_{t+1}, a causal feature $\psi_\rho(\mathbf{x})$, a distribution $P(H_{t+1}|\psi_\rho(\mathbf{x}))$ parameterized by ω_{t+1}, and a compositional rule to obtain $f_{t+1}(\{H_1, \cdots, H_t\})$ by combining $f_t(\{H_1, \cdots, H_t\})$ with H_{t+1}. We grow the distribution $P_t(y|\mathbf{x})$ in a greedy manner by coordinate descent on the performance measure hence selecting the causal feature, and logical combination rule, that gives maximal increase in performance.

We initialize the distribution $P_{t=0}(y|\mathbf{x})$ by setting $f_0(\{H_i\}) = H_1 \vee H_2 \vee \cdots$ and $\alpha_i = \beta_i = 0, \forall i$, which corresponds to $P_0(y = 1|\mathbf{x}) = 0, \forall \mathbf{x}$. (An alternative initialization is $f_0(\{H_i\}) = H_1 \wedge H_2 \wedge \cdots$ with $\alpha_i = \beta_i = 1, \forall i$, which corresponds to $P_0(y = 1|\mathbf{x}) = 1, \forall \mathbf{x}$.)

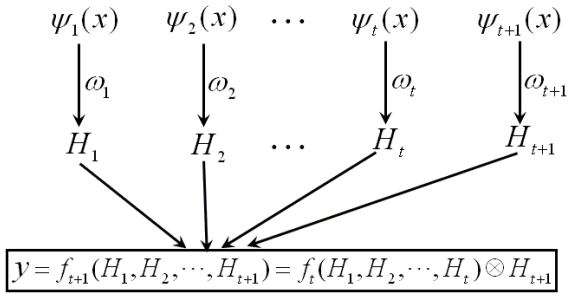


Figure 2. CNLL grows a Noisy-Logical distribution.

The first CNLL algorithm has pseudo-code given in Fig. 3. It uses the composition rule:

$$f_{t+1}(\{H_1, \cdots, H_{t+1}\} = f_t(\{H_1, \cdots, H_t\}) \otimes H_{t+1},$$

where $\otimes = \wedge$ or $\otimes = \vee$ (i.e. logical AND or OR). This composition generates a new distribution $P_{t,(\rho,\omega,\otimes)}(y|\mathbf{x})$ which defines a classifier $R^{t,(\rho,\omega,\otimes)}(\mathbf{x})$.

To select the best extension of $P_t(y|\mathbf{x})$, we solve for

$$(\rho^*, \omega^*, \otimes^*) = \arg \min_{(\rho,\omega,\otimes)} E_R(R^{t,(\rho,\omega,\otimes)}).$$

This is obtained by an exhaustive search over ρ and $\otimes$ and, for fixed $(\rho, \otimes)$, we minimize $E_R(R^{t,(\rho,\omega,\otimes)})$ with respect to $\omega = (\alpha, \beta)$. This search is simplified by the decomposition

$$E_R(R^t) = E_{R,1}(\alpha, \otimes) + E_{R,0}(\beta, \otimes),$$

where

$$E_{R,1}(\alpha, \otimes) = \sum_{\mu \in \Lambda: \psi_\rho(\mathbf{x})=1} \{1 - \delta_{R^t(\mathbf{x}_\mu), y_\mu}\}$$

and

$$E_{R,0}(\beta, \otimes) = \sum_{\mu \in \Lambda: \psi_\rho(\mathbf{x})=0} \{1 - \delta_{R^t(\mathbf{x}_\mu), y_\mu}\}.$$

It is straightforward to verify that $E_{R,1}(\alpha, \otimes)$ and $E_{R,0}(\beta, \otimes)$ are functions of α and β, respectively. Hence we can determine $\alpha_{t+1} \in [0, 1]$ and $\beta_{t+1} \in [0, 1]$ by two independent one dimensional searches. Then we set $P_{t+1}(y|\mathbf{x}) = P_{t,(\rho^*,\omega^*,\otimes^*)}(y|\mathbf{x})$.

This require more computation that AdaBoost. AdaBoost must also do exhaustive search over all classifiers ρ but can then determine the parameters λ by an analytic expression. By contrast CNLL must do one dimensional searches for α and β within the range $[0, 1]$ and check for the two possibilities (AND or OR) for $\otimes$.

If $\otimes = \wedge$, then $f_{t+1}(\{H_i\}) = f_t(\{H_i\}) \wedge H_{t+1}$ and $P_{t+1}(y = 1|\mathbf{x}) = P_t(f_t = 1|\mathbf{x}) \times(\alpha_{t+1}\psi_{t+1}(\mathbf{x}) + \beta_{t+1}(1 - \psi_{t+1}(\mathbf{x})))$.

If $\otimes = \vee$, then $f_{t+1}(\{H_i\}) = f_t(\{H_i\}) \vee H_{t+1}$ and $P_{t+1}(y = 0|\mathbf{x}) = P_t(f_t = 0|\mathbf{x}) \times\{1 - \alpha_{t+1}\psi_{t+1}(\mathbf{x}) - \beta_{t+1}(1 - \psi_{t+1}(\mathbf{x}))\}$.

3.2. Converting to DNF

The final form of the noisy-logical distribution depends on the logical function $y = f(\{H_i\})$ relating the hidden states. We can express this in Disjunctive Normal Form (DNF) as $y = CL_1 \vee CL_2 \cdots \vee CL_N$ where each clause is a conjunction of the $\{H_i\}$, see Fig 1 (right panel).

Expressing the hidden states in DNF gives insight into the structure of the data. We can think of the data as being generated by (deterministic) logical process specified by the DNF but where the $\{H_i\}$ are not observable and must be inferred from noisy measurements $\{\psi_i(\cdot)\}$.

The following iterative procedure can be used to transform the the output of the CNLL algorithm – $y = f(\{H_i\}) = (\cdots(H_1 \otimes H_2) \otimes H_3)\cdots)$ – into DNF. We initialize at $t = 1$ by setting $y = H_1$, which is clearly in DNF. Now we suppose we have successfully converted the expression $y = (\cdots((H_1 \otimes H_2) \otimes H_3) \otimes H_4 \cdots) \otimes H_t$ to DNF $y = CL_1 \vee CL_2 \vee \cdots \vee CL_N$, where the CL_i's are conjunctions of H_i's. For the $(t+1)^{th}$ iteration, we have $y = ((\cdots((H_1 \otimes H_2) \otimes H_3) \otimes H_4 \cdots) \otimes H_t) \otimes H_{t+1} = (CL_1 \vee CL_2 \vee \cdots \vee CL_N) \otimes H_{t+1}$. If $\otimes = \vee$, then $y = CL_1 \vee CL_2 \vee \cdots \vee CL_N \vee H_{t+1}$ is of DNF form with $CL_{N+1} = H_{t+1}$, i.e. we just add a new clause. If $\otimes = \wedge$, then $y = (CL_1 \vee CL_2 \vee \cdots \vee CL_N) \wedge H_{t+1} = (CL_1 \wedge H_{t+1}) \vee (CL_2 \wedge H_{t+1}) \vee \cdots \vee (CL_N \wedge H_{t+1})$ is of DNF form with $CL_i = CL_i \wedge H_{t+1}$, i.e. we modify

Input: Training set $\{(\mathbf{x}_\mu, y_\mu) : \mu = 1, \cdots, N\}$, a set of causal features $\{\psi_\rho(\mathbf{x}) : \rho \in \Omega\}$, where $\mathbf{x}$ is the data vector and $y \in \{0, 1\}$ is the label, and Ω indexes all the possible features. Initialize $\alpha_i = 0$ and $\beta_i = 0$ for all i.

For $t = 1, \cdots, T$:

- Set $MinError = 1$;
- For each $\otimes \in \{\vee, \wedge\}$
  - For each $\rho \in \Omega$
    1. find $\alpha^* = \arg\min_\alpha E_{R,1}(\alpha, \otimes)$ by a 1-D search over $[0, 1]$
    2. find $\beta^* = \arg\min_\beta E_{R,0}(\beta, \otimes)$ by a 1-D search over $[0, 1]$
    3. $Error = [E_{R,1}(\alpha^*, \otimes) + E_{R,0}(\beta^*, \otimes)]/N$
    4. if $Error < MinError$, then set $(\rho^*, \alpha^*, \beta^*, \otimes^*) \leftarrow (\rho, \alpha^*, \beta^*, \otimes)$, and set $MinError = Error$
  - End for ρ
- End for $\otimes$
- Update $f_{t+1}(\{H_i\}) = f_t(\{H_i\}) \otimes^* H_{t+1}$ and the distribution $P_t(y = 1|\mathbf{x})$.

End algorithm

Output: Logical expression $f_T(\{H_i\})$ and parameterized distribution $P(y = 1|\mathbf{x}; \vec{\omega}_T, f_T)$.

Figure 3. Pseudo-code for the first CNLL algorithm

each clause by adding a conjunctive term.

3.3. The Second CNLL Algorithm

The second CNLL algorithm represents the logical form as DNF. At each iteration it combines $f_t(\{H_1, \cdots, H_t\})$ with H_{t+1} by allowing H_{t+1} to be AND-ed with any of the clauses $CL_1, \cdots, CL_N$ in $f_t(\{H_1, \cdots, H_t\})$, or AND-ed with all of the clauses, or alternatively to create a new clause. The pseudo-code for this second CNLL algorithm is given in Fig. 4. This algorithm allows a greater number of ways to construct the logical form $f_{t+1}(\{H_1, \cdots, H_{t+1}\})$ from $f_t(\{H_1, \cdots, H_t\})$ and hence is a more powerful algorithm, but it is slightly slower since it has to evaluate more possibilities.

4. Experimental Results

We evaluate CNLL in two ways. Firstly, by using data generated from noisy-logical processes – e.g. by logical rules such as exclusive-or (XOR) with some additional noise. This is to verify that CNLL gives the correct results for the class of problems that it is well suited to. Secondly, we compare it to AdaBoost on standard datasets. In both cases, we evaluate the algorithm in terms of classification performance and knowledge extraction.

4.1. Results on Noisy-Logical Data

Noisy-logical data means that the process of generating the data is basically logical but with noise added. This can either be obtained by starting with a logical rule and adding noise (our first example) or by sampling from the discriminative model. For noisy-logical data we expect that CNLL will give better performance than AdaBoost since the data will be described by a sparse noisy-logical distribution but not by a sparse exponential model. We anticipate that CNLL will learn the correct underlying logical process of the data – i.e., determine the correct DNF.

We first consider a simple variant of the classic XOR problem. We generate 3,000 points $(x_1, x_2) \in [0, 2] \times [0, 2]$ in the following manner: (i) sample 1,000 points uniformly in $[0, 1] \times [0, 1]$, (ii) sample 500 points uniformly in $[1, 2] \times [1, 2]$, (iii) sample 400 points uniformly in $[1, 2] \times [0, 1]$, and (iv) sample 1,100 points uniformly in $[0, 1] \times [1, 2]$. We classify a data-point (x_1, x_2) as a positive example if $x_1 \in [0, 1] \wedge x_2 \in [0, 1]$ OR $x_1 \in [1, 2] \wedge x_2 \in [1, 2]$ and as a negative example otherwise. The weak classifiers are defined by thresholding the x_1 and x_2 coordinates – i.e., the set of rules of form $x_i > T_1$ or $x_i < T_2$ for $i = 1, 2$ where T_1, T_2 are thresholds.

We use the second CNLL algorithm to learn the underlying logical expression. Table 1 shows the selected features, the clause structure and the error rates after a

Input: Training set $\{(\mathbf{x}_\mu, y_\mu) : \mu = 1, \cdots, N\}$, a set of causal features $\{\psi_\rho(\mathbf{x}) : \rho \in \Omega\}$, where $\mathbf{x}$ is the data vector and $y \in \{0, 1\}$ is the label, and Ω indexes all the possible features.
Goal: Learn the DNF expression $f(\{H_i\}) = CL_1 \vee \cdots \vee CL_n$, with each CL_i is a conjunction of a subset of $\{H_i\}$.
Initialize: $n = 1$, and $CL_1 = 1$, $\alpha_i = 0$ and $\beta_i = 0$ for all i.

For $t = 1, \cdots, T$:
//Trying to add H_t to an existing clause

- Set $MinErrC = 1$; //the smallest error when adding to an existing clause
- For $i = 1, \cdots, n$
  - Let $f' = CL_1 \vee \cdots \vee CL'_i \vee \cdots \vee CL_n$, with $CL'_i = CL_i \wedge H_{t+1}$
  - Search $(\rho^*, \alpha^*, \beta^*, Error)$ as in Fig. 3;
  - if $Error < MinErrC$, then set $(\rho_C, \alpha_C, \beta_C, C) \leftarrow (\rho^*, \alpha^*, \beta^*, i)$, and set $MinErrC = Error$
- Endfor // Now the parameters are stored in $(\rho_C, \alpha_C, \beta_C, C)$
  // Trying to add H_t to every clause
- Let $f' = (CL_1 \wedge H_t) \vee \cdots \vee (CL_n \wedge H_t)$.
- Search (ρ, α, β) to minimize the training error; denote the optimal parameter as $(\rho_A, \alpha_A, \beta_A)$ and the minimal error as $MinErrAll$.
  //Trying to add a new clause H_t
- Let $f' = CL_1 \vee \cdots \vee CL_n \vee H_t$ ($CL_1 = 0$ if $t = 1$).
- Search (ρ, α, β) to minimize the training error; denote the optimal parameter as $(\rho_N, \alpha_N, \beta_N)$ and the minimal error as $MinErrN$.
  // updating the Clauses, and the DNF expression
- Pick the smallest value from $MinErrC$, $MinErrAll$, and $MinErrN$, update each clause CL_i accordingly, and update the clause number n if necessary; denote the new DNF expression as $f_t(\{H_i\})$.
- Update the distribution $P_t(y = 1|\mathbf{x})$ according to the form of $f_t(\{H_i\})$.

End algorithm

Output: DNF expression $f_T(\{H_i\})$ and parameterized distribution $P(y = 1|\mathbf{x}; \vec{\omega}_T, f_T)$

Figure 4. The second CNLL algorithm, "//" stands for comments.

new feature is added. CNLL learns the logical expression $E = (x_2 < 1 \wedge x_1 < 1) \vee (x_1 > 1 \wedge x_2 > 1)$, which is the correct solution to the XOR problem. (Note: the first CNLL algorithm gets stuck at a saddle point of the error measure and lacks the correct logical rule to descend further – it learns $(H_1 \wedge H_2) \vee H_3$, but is unable to move to $(H_1 \wedge H_2) \vee (H_3 \wedge H_4)$).

Table 1. The features selected for XOR problem.

Feature	$x_2 < 1$	$x_1 > 1$	$x_2 > 1$	$x_1 < 1$
Clause	1	2	2	1
Error	0.300	0.134	0.134	0.000

We now test CNLL on more noisy-logical data obtained by sampling from the discriminative distribution $P(y|\{\psi_i\})$. We specify a underlying logical process $y = (H_1 \wedge H_2) \vee (H_3 \wedge H_4)$ for the data with added noise (described below). We compare the classification performance of CNLL and AdaBoost on this data and check to see whether CNLL discovers the correct noisy-logical structure.

We generate the noisy-logical data by sampling from the distribution $P(y|\{\psi_i\})$ by exploiting the structure shown in Fig. 1. We first sample to get the $\{CL_i\}$ from $P(\{CL_i\}|y) = \frac{P(y|\{CL_i\})P(\{CL_i\})}{P(y)}$, next we sample the $\{H_i\}$ from $P(\{H_i\}|\{CL_i\})$, and finally we sample the $\{\psi_i\}$ from $P(\{\psi_i\}|\{H_i\}) = \frac{P(\{H_i\}|\{\psi_i\})P(\{\psi_i\})}{P(\{H_i\})}$. More precisely, we set $P(CL_1 = 1, CL_2 = 0|y = 1) = P(CL_1 = 0, CL_2 = 1|y = 1) = 0.3$, and $P(CL_1 = 1, CL_2 = 1|y = 1) = 0.4$; when $y = 0$, we must have $CL_1 = 0$ and $CL_2 = 0$. We also specify $P(H_i = 1, H_j = 0|CL_k = 0) = P(H_i = 0, H_j = 1|CL_k = 0) = 0.4$, and $P(H_i = 0, H_j = 0|CL_k = 0) = 0.2$, where CL_k is one of the two clauses, H_i and H_j are two components of the corresponding clause; obviously, when $CL_k = 1$, both of the two component must be 1. Finally, we specify $P(\psi_i = 1|H_i = 1)$ and $P(\psi_i = 0|H_i = 0)$ for $i = 1, \cdots, 4$ (observe that when these two probabilities are set to be 1, it will be a deterministic logical expression). The components $\psi_5, \cdots, \psi_{50}$ are unrelated to the logical expression and they are sampled uniformly at random from $\{0, 1\}$. We generated 1000 examples for each case of $y = 0$ and $y = 1$.

We used the obtained data as input to the second CNLL and AdaBoost algorithm, to compare their performance on this noisy-XOR data. Again we expect CNLL to outperform AdaBoost since the data can be sparsely described by a noisy-logical distribution but not by an exponential model.

We consider two cases: (I) $P(\psi_i = 1|H_i = 1) = 0.95$ and $P(\psi_i = 0|H_i = 0) = 0.95$. (II) $P(\psi_i = 1|H_i = 1) = 1$ and $P(\psi_i = 0|H_i = 0) = 1$.

CNLL performs well for both cases and obtains the correct DNF logical expression. It selects the correct causal cues $\psi_1, ..., \psi_4$ and ignores the rest. For case (I), CNLL has training and testing error rates of 5% and 7%, respectively. For case (II), CNLL has 0% error rate in both training and testing stages.

AdaBoost performs comparatively poorly. For case (I) it has training and testing errors of 21% and 24% respectively. For case (II) it has training and testing errors are 16% and 20% respectively. In both cases, AdaBoost algorithm selects 20 features. In the first few iterations, AdaBoost selects the correct causal cues – $\psi_1, \cdots, \psi_4$ – but at later iterations AdaBoost also selects some of the irrelevant features $\psi_5, \cdots, \psi_{50}$. Overall the performance of AdaBoost is poor at both classification and knowledge extraction.

4.2. Results on Standard Datasets and Comparison to AdaBoost

We now compare CNLL with AdaBoost on four standard datasets (breast cancer, ionosphere, splice, and ocr49) which are available from the UCI repository and for which AdaBoost results are reported. The generating processes are unknown for these datasets, so it is unclear whether exponential or noisy-logical models would give better fits.

Table 2 shows the size and the number of attributes of the datasets. We compare our results against those of AdaBoost by using stump features reported in (Reyzin & Schapire, 2006). We use similar stump features for CNLL. Table 3 shows the results of CNLL and AdaBoost, and Fig. 5 shows the average training and testing error curves of CNLL.

Our results show that we obtain significantly better results than AdaBoost on all four datasets. Moreover, CNLL outputs simpler representations using a far fewer causal features. We typically obtain good results with around 10 features, while AdaBoost typically uses $O(100)$. CNLL shows a greater tendency to over-generalize than AdaBoost, see Fig. 5. While obtaining comparable results to AdaBoost on the test set we obtain significantly better results on the training set (sometimes by a factor of three). This phenomena may be partly explained because CNLL gives a richer class of decision boundaries than AdaBoost (this can be seen by examining the form of $\log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}$ for noisy-logical distributions). Another reason might be that the CNNL algorithm is greedy and might get trapped in a local minimum. We hope that better understanding of CNLL may help us take advantage of the low training errors and obtain even better results on the test set.

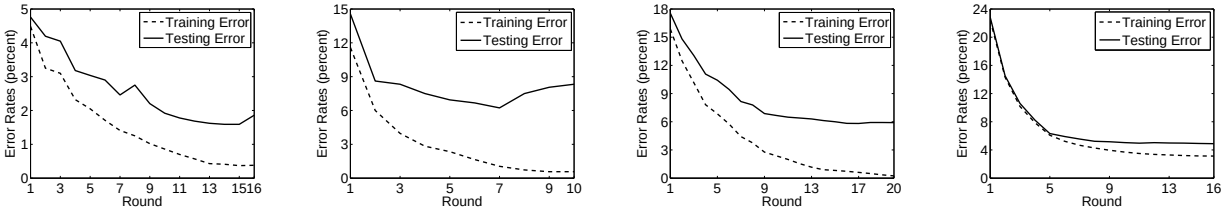
Table 2. The sizes of the training and testing datasets for each of ten trials and the number of attributes. These trial data samples are randomly selected from the datasets in the UCI repository.

	cancer	iono	ocr49	splice
Training	630	315	1000	1000
Testing	69	36	5000	2175
Attributes	10	34	64	60

Table 3. The testing errors of CNLL and AdaBoost on four datasets (averaged over 10 trials for each dataset). Observe that CNLL gives better results than AdaBoost in all cases. AdaBoost was run for 500 rounds of all trials hence using roughly 500 weak classifiers. CNLL required much fewer weak classifiers in general – on average 9 for cancer, 5 for ions, 15 for OCR49, and 15 for splice.

	cancer	iono	ocr49	splice
AdaBoost	4.29%	9.58%	6.28%	6.79%
CNLL	1.74%	6.11%	5.83%	4.70%

We now attempt to use the hidden variables H and clauses CL to extract knowledge about the structure of the data. We picked one of the trial results on the cancer dataset where CNLL output the logical expression $f(\{H_1, \cdots, H_9\}) = CL1 \vee CL_2 \vee CL_3 \vee CL_4$, with $CL_1 = H_1 \wedge H_2 \wedge H_3 \wedge H_5 \wedge H_8 \wedge H_9$, $CL_2 = H_4 \wedge H_5 \wedge H_8 \wedge H_9$, $CL_3 = H_6 \wedge H_8 \wedge H_9$, and $CL_3 = H_7 \wedge H_8 \wedge H_9$. Table 4 shows the performance of the clauses: the first and second rows show the true positive and error rates for each clause. The third and fourth rows show the accumulated true positive and error rates (combining the clauses). For this example, we see that the first clause is clearly dominant and explains most of the error. We observe similar results on other trials for these datasets – dominance by a single AND clause. This suggests that either these datasets have limited OR structure, or that CNLL is unable to extract it.



(a) Breast cancer (b) ionosphere (c) OCR49 (d) splice

Figure 5. Average training and testing error curves on UCI data sets. CNLL gives good results for five causal features.

Table 4. Clause analysis for one of the trials on cancer dataset, see text for details.

Clause	CL_1	CL_2	CL_3	CL_4
Individual True Positive	0.9737	0.0310	0.2792	0.8878
Individual Error Rate	0.0206	0.6444	0.4794	0.0778
Cumulative True Positive	0.9737	0.9785	0.9809	0.9833
Cumulative Error Rate	0.0206	0.0175	0.0159	0.0143

5. Conclusion

This paper conjectured that the effectiveness of AdaBoost in terms of performance, and particularly knowledge extraction, depends on whether the conditional distribution for the data can be approximated by a sparse exponential distribution. We described how conditional distributions could also be represented as noisy-logical distributions (Yuille & Lu, 2008) and conjectured that for some applications this would lead to sparser representations than those obtained by exponential distributions.

We developed Compositional Noisy-Logical Learning (CNLL) as a way to learn two-classe classifier by composing elementary noisy-logical distributions. We specified two CNLL algorithms both of which proceeded by coordinate descent on a training error measure. This is slightly more complex than learning for AdaBoost: (i) it requires performing two searches for values in range $[0, 1]$, (ii) the training error is not convex (unlike the measure used in AdaBoost). Nevertheless, the training time increases with respect to AdaBoost only by a constant factor (due to the searches in $[0, 1]$). The resulting distribution is expressed using hidden units with DNF and has a simple transparent form.

We tested CNLL on two types of data: (i) data generated to be in noisy-logical form, and (ii) four standard datasets from the UCI repository. Both sets of results showed the effectiveness of CNLL both in terms of performance and for knowledge extraction. CNLL was much more successful than AdaBoost for the noisy-logical data and was able to identify the underlying logical process. CNLL also outperformed AdaBoost on the four datasets from the UCI repository both in terms of classification performance and in terms of the sparseness of the resulting representation (by an order of magnitude) enabling knowledge extraction. Note that CNLL takes longer to train than AdaBoost by a constant factor, but the number of weak classifiers required by CNLL is smaller (by an order of magnitude), so the total amount of training time is roughly compatible. Moreover, the resulting CNLL classifier is faster in testing because of its reliance on fewer weak classifiers. This may be an advantage for practical systems (e.g., implementing a face detector on a cell phone).

Acknowledgments

We acknowledge the support from the Air Force FA9550-08-1-0489. We appreciate conversations with YingNian Wu and HongJing Lu.

References

Alpaydin, E. (2004), *Introduction to Machine Learning*, MIT Press.

Cheng, P. W. (1997) From covariation to causation: A causal power theory. *Psychological Review, 104*, 367–405.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. *Proceedings of 13th International Conference on Machine Learning*, 148-156.

Friedman, J., Hastie, T., & Tibshirani, R. (2000).

Additive Logistic Regression: a Statistical View of Boosting. *Annal of Statistics 28*, 337–407.

Griffiths, T. L., & Tenenbaum, J. B.(2005). Structure and strength in causal induction. *Cognitive Psychology, 51*, 334–384.

Meyer, Y. (2001). *Oscillating Pattern in Image Processing and Nonlinear Evolution Equations.* University Lecture Series Vol 22. American Mathematical Society.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann.

Reyzin, L., & Schapire, R. (2006). How Boosting the Margin can also boost classifier complexity. *23rd International Conference on Machine Learning*, 753-760.

Yuille, A. L., & Lu H. J. (2008). The Noisy-Logical Distribution. *Advances in Neural Information Processing Systems.*