CS786P—Topics in Artificial Intelligence
Computational Learning

Assignment 2

Fall 2001
Department of Computer Science
University of Waterloo

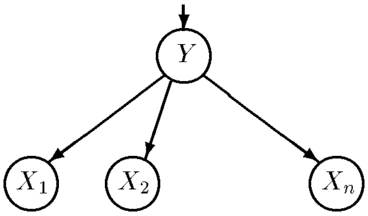
Due: in class, *Monday, November 26*
Worth: 25% of final grade (5 questions of equal weight, so each question is worth 5%)
Instructor: Dale Schuurmans, DC1310, x3005, dale@cs.uwaterloo.ca

---

Note 2 of the 5 questions require you to write small Matlab programs which are to be submitted by email. When finished, please send a *single* tar file containing all of your .m files to dale@cs.uwaterloo.ca with a subject heading "A2 solutions".

Question 1 (Probability models)

In class we discussed the "Naive Bayes" (NB) probability model, given by the generator



which specifies that the probability of a joint assignment to the variables is given by

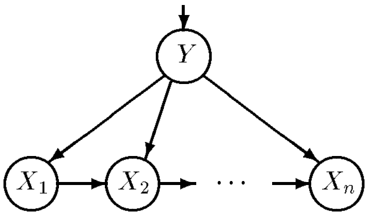
$$\mathrm{P}(y, x_1, x_2, ..., x_n) \quad = \quad \mathrm{P}(Y = y) \prod_{j=1}^{n} \mathrm{P}(X_j = x_j | Y = y).$$

Recall that for boolean attributes $X_j \in \{0, 1\}$, $Y \in \{0, 1\}$ this model is determined by $1 + 2n$ free parameters: $\theta = \mathrm{P}(Y = 1)$, $\theta_{j1} = \mathrm{P}(X_j = 1|Y = 1)$, and $\theta_{j0} = \mathrm{P}(X_j = 1|Y = 0)$. Also remember that the minimum-expected-error classification rule for this model

$$\hat{y} = 1 \quad \Leftrightarrow \quad \mathrm{P}(Y = 1|x_1, ..., x_n) \ \geq \ \mathrm{P}(Y = 0|x_1, ..., x_n)$$

is equivalent to a linear discriminant $\hat{y} = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x} - b)$ that uses weights $w_j = \log \frac{p_{j1}}{p_{j0}} \frac{1-p_{j0}}{1-p_{j1}}$ and bias $b = \log \frac{1-p}{p} + \sum_{j=1}^{n} \log \frac{1-p_{j0}}{1-p_{j1}}$. In this question we will explore a simple generalization of this model that has some interesting properties.

A "*Chain Augmented Naive Bayes*" model (CAN) is a Naive Bayes-like model where we include an additional chain of dependencies that runs through the variables as follows



This generative model specifies that we compute the probability of a joint assignment to the variables according to

$$\mathrm{P}(y, x_1, x_2, ..., x_n) \quad = \quad \mathrm{P}(Y = y)\mathrm{P}(X_1 = x_1|Y = y) \prod_{j=2}^{n} \mathrm{P}(X_j = x_j|Y = y, X_{j-1} = x_{j-1}).$$

Assume boolean attributes $X_j \in \{0, 1\}$, $Y \in \{0, 1\}$.

(a) (1%) Specify the parameters of the CAN model in a manner similar to the NB model above. How many free parameters are there (as a function of n)?

(b) (1%) Given training data

$$\begin{bmatrix} x_{11} & \cdots & x_{1n} & y_1 \\ \vdots & & \vdots & \vdots \\ x_{t1} & \cdots & x_{tn} & y_t \end{bmatrix}$$

what are the maximum likelihood estimates for the CAN parameters?

(c) (1%) Assume we are doing Bayesian learning with a Beta(1,1) prior on each of the parameters. Making the strong independence assumptions we discussed in class, what are the maximum a posteriori (MAP) estimates for these parameters?

(d) (2%) Derive a simple form for the minimum-expected-error classification rule for this model in a similar manner to the NB model above. What form does this decision rule have? (Hint: it is not a linear discriminant, but it is close.)

Question 2 (Learning probability models)

In this question we will investigate the consequences of learning probability models when incorrect assumptions are made about the structure of the distribution. The lesson is that even principled learning under bad modeling assumptions can lead to suboptimal results. This question builds upon the previous question.

Assume that data is generated according to a CAN model where

$$\begin{aligned} \mathrm{P}(Y=1) &= \tfrac{2}{3}, \\ \mathrm{P}(X_1=1|Y=1) &= \mathrm{P}(X_1=1|Y=0) \;=\; \tfrac{1}{2}, \\ \text{and } X_j &= X_{j-1} \oplus Y \text{ for } j=2,...,n,\ n \geq 2 \end{aligned}$$

where $\oplus$ denotes "exclusive or". (That is, X_j is a deterministic function of X_{j-1} and Y and therefore $\mathrm{P}(X_j = x_{j-1} \oplus y | X_{j-1} = x_{j-1}, Y = y) = 1$.) Assume also that we are (incorrectly) attempting to learn from this data source by using a Naive Bayes model.

(a) (1%) What are the parameters of the optimal NB model for this CAN data source? (Note: we haven't formally discussed optimal approximations for probability distributions in this course, but for this question just assume it means computing the corresponding conditional probabilities under the true model.)

(b) (1%) What is the minimum-expected-error classification rule for the optimal NB model?

(c) (1%) What is the misclassification probability of this NB classifier when test examples are drawn from the CAN model?

(d) (1%) What is the misclassification probability of the optimal CAN classifier? (Note: just compute the probability—you do not have to show a derivation of the optimal CAN classifier.)

(e) (1%) For this part, assume that $n = 2$. (That is, we are only considering the variables Y, X_1 and X_2.) Write down the coefficients w_1, w_2 and b of a linear discriminant that achieves better misclassification probability than the solution of Part (b). What is the misclassification probability of your alternative linear discriminant?

Question 3 (Bayesian learning—linear Gaussian models)

In this exercise you will write a MATLAB function to compute the Bayesian "maximum a posteriori" estimate of a conditional linear model, and then test this function on simulated data. You will compare the results of Bayesian learning to the earlier method used in Assignment 1 (Question 2), which was based on directly minimizing the empirical training error. This exercise demonstrates how Bayesian learning can control the dramatic overfitting effects of direct error minimization, and thereby leads to much better results in practice.

As in Assignment 1 it will be useful to know about MATLAB's `polyval` function.

In this question we will be interested in learning polynomial predictors. As in the first assignment, we will consider two ways of generating the data that start with a one dimensional input and then expand each of the training inputs x_i into vectors $\mathbf{x}_i = (x_i^d, x_i^{d-1}, ..., x_i, 1)$ to obtain the final training matrix `X`.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_t \end{bmatrix} \quad \mapsto \quad \mathtt{X} = \begin{bmatrix} x_1^d & x_1^{d-1} & \cdots & x_1 & 1 \\ \vdots & & & & \vdots \\ x_t^d & x_t^{d-1} & \cdots & x_t & 1 \end{bmatrix} \quad \mathtt{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix}$$

Data generation:

```
t = 10            % training size
sigma = 0.1       % noise level
x = rand(t, 1)    % original 1D training patterns
% X will be generated internally in your function (below)
```

Generative model 1: `y = x > 0.5` (step function)

Generative model 2: `y = 0.5 - 10.4*x.*(x-0.5).*(x-1) + sigma*randn(t,1)` (noisy cubic)

Recall from class that learning a polynomial predictor is equivalent to learning a simple linear predictor over the expanded representation. Thus, our goal is to learn a vector of weights $\mathbf{w} = (w_1, ..., w_{d+1})$ that can be used to predict on test inputs x according to $\hat{y} = \mathbf{w} \cdot (x^d, x^{d-1}, ..., x, 1)$. Here $\mathbf{w}$ is interpreted as specifying a polynomial $p(x) = w_1 x^d + w_2 x^{d-1} + \cdots + w_d x + w_{d+1}$.

We will treat this problem as learning a conditional linear Gaussian model $y|\mathbf{x}, \mathbf{w} \sim N(\mathbf{x} \cdot \mathbf{w}, \sigma^2)$ where we have a *prior* over the weight vector given by a multidimensional Gaussian $\mathbf{w} \sim N(\mathbf{0}, \Gamma)$. Note that the $(d+1) \times (d+1)$ covariance matrix Γ is a parameter that specifies how variable (and how correlated) we believe each of the weight parameters are *a priori*.

(a) (2%) Write a MATLAB function `mappoly(x,y,sigma,d,gamma)` which takes a $t \times 1$ vector of training inputs $\mathbf{x}$, and a $t \times 1$ vector of target values `y`, scalars `sigma` and `d`, and a $(d+1) \times (d+1)$ covariance matrix `gamma`, and returns: a $(d+1) \times 1$ vector of weights `mu` which corresponds to the maximum *a posteriori* (MAP) polynomial of degree `d`.

Note: To compute this weight vector you need to use the fact that the assumptions $\mathbf{w} \sim N(\mathbf{0}, \Gamma)$ and $y|\mathbf{w}, \mathbf{x} \sim N(\mathbf{x} \cdot \mathbf{w}, \sigma^2)$ imply that $\mathbf{w}|\mathtt{y}, X \sim N(B\mathbf{b}, B)$ for $B^{-1} = X^\top X/\sigma^2 + \Gamma^{-1}$ and $\mathbf{b} = X^\top \mathtt{y}/\sigma^2$. Thus, the MAP weight vector $\boldsymbol{\mu} = \mathrm{E}(\mathbf{w}|\mathtt{y}, X)$ is simply the solution to the system of equations $B^{-1}\boldsymbol{\mu} = \mathbf{b}$. (Notice that this is, in fact, equivalent to a "regularized" minimum squared error hypothesis, as shown in class.)

Implementing this function requires you to first expand each training input x_i into a vector $(x_i^d, x_i^{d-1} ... x_i, 1)$ to obtain X, and then solving a system of linear equations to obtain `mu`. You can do this using MATLAB's `slash` and `inv` operators.

Your function must be able to handle arbitrary t, `d` and `gamma`.[1]

[1]Note that `gamma` is actually assumed to be a symmetric, positive definite matrix, and hence invertible.

(b) (2%) For each of the generative models 1 and 2:

A: Generate a random training set **x**, **y** using the model, and solve for the best fit polynomials of degree 1, 3, 5, and 9 using your `minL2poly` solution from Assignment 1. Then re-solve for the MAP polynomials of degree 1, 3, 5, and 9 using your mappoly function with the prior covariance matrix `gamma = eye(d+1)`:

```
c1 = minL2poly(x,y,1)      mu1 = mappoly(x,y,0.1,1,eye(1+1))
c3 = minL2poly(x,y,3)      mu3 = mappoly(x,y,0.1,3,eye(3+1))
c5 = minL2poly(x,y,5)      mu5 = mappoly(x,y,0.1,5,eye(5+1))
c9 = minL2poly(x,y,9)      mu9 = mappoly(x,y,0.1,9,eye(9+1))
```

B: For each degree $d = 1, 3, 5, 9$ produce a 2D plot of the training data and the two hypotheses corresponding to `c` and `mu`.

```
clf; axis([0 1 -0.5 1.5]); hold;
plot(x',y','w*')
xx = (0:1000)/1000;
yy = "<mean value of generative model at xx>"
plot(xx,yy,'w:')
plot(xx,polyval(mu<d>,xx),'w-')
plot(xx,polyval(c<d>,xx),'w-.')
print -deps experiment.2.4.<d,k>.ps
```

C: Report the *mean* sum of squares error (*i.e.*, the sum of squares error divided by t) that each of the eight hypotheses obtained on the training data:

		L_2 error
min err poly	c1	
	c3	
	c5	
	c9	

		L_2 error
map poly	mu1	
	mu3	
	mu5	
	mu9	

D: Generate `te = 1000` test examples from the same generative model and report the mean sum of squares error on the *test* data (*i.e.*, the sum of squares error divided by `te`) in the same form as above.

Hand in a plot and two tables for each generative model.

(c) (1%) For each generative model: Repeat (b) parts A, C and D 100 times and accumulate the sum of mean squared errors for each degree in two matrices: one for the training errors and one for the testing errors. Report the *averages* for each kind of mean error at each degree in two tables (one training error and the other testing error).

Question 4 (EM—application to Naive Bayes)

In this question you will derive an EM training algorithm for the Naive Bayes probability model introduced in Question 1. The outcome will be a learning method that can produce a good predictor $h : X^n \to Y$ based on exploiting both labeled $\langle \mathbf{x}_i, y_i \rangle$ and unlabeled $\langle \mathbf{x}_i, - \rangle$ training examples. Recall that the Naive Bayes model assumes there is a single random variable $Y \in \{0, 1\}$ whose value determines the conditional distribution over the remaining variables $X_1, ..., X_n$ ($X_j \in \{0,1\}^n$), where, in particular, the conditional distributions of $X_1, ..., X_n$ are independent given $Y = y$. This model is defined by $2n + 1$ parameters $\boldsymbol{\theta} = \langle \theta, \{\theta_{j0}\}_{j=1}^n, \{\theta_{j1}\}_{j=1}^n \rangle$.

Consider a scenario where all of the variables X_j are always observed but the Y labels are sometimes missing. That is, consider training data of the form

$$\begin{bmatrix} x_{11} & \dots & x_{1n} & z_1 \\ \vdots & & \vdots & \vdots \\ x_{t1} & \dots & x_{tn} & z_t \end{bmatrix}$$

where the z_i labels are chosen from the set $\{0, 1, -1\}$ such that $z_i = -1$ indicates that the original y_i value was *unobserved*. Recall that EM works as follows:

EM

- Start with an initial parameter vector $\boldsymbol{\theta}^{(0)}$.
- Repeatedly update the parameter vector $\boldsymbol{\theta}^{(k)} \to \boldsymbol{\theta}^{(k+1)}$ in two sub-stages:
  - **E step:** For each training example $x_{i1}, ..., x_{in}, z_i$, compute the conditional probability of each possible completion $y_i = 0$ or $y_i = 1$ given the observed x-values $x_{i1}, ..., x_{in}$:

$$Q_i^{(k)}(y) = \begin{cases} 0 & \text{if } z_i \in \{0,1\} \text{ and } y \neq z_i \\ 1 & \text{if } z_i \in \{0,1\} \text{ and } y = z_i \\ \mathrm{P}(Y = y | X_1 = x_{i1}, ..., X_n = x_{in}, \boldsymbol{\theta}^{(k)}) & \text{if } z_i = -1 \end{cases} \tag{1}$$

  - **M step:** Compute:

$$\boldsymbol{\theta}^{(k+1)} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{t} \sum_{y=0}^{1} Q_i^{(k)}(y) \;\; \log \mathrm{P}(X_1 = x_{i1}, ..., X_n = x_{in}, Y = y | \boldsymbol{\theta}) \tag{2}$$

In this question you will derive a simplified form of the EM update for the Naive Bayes model, which will allow you to implement it efficiently in Question 5 below.

(a) (1%) Express the summation in (2) as a function of $\theta, \theta_{j0}, \theta_{j1}$, x_{ij}, y and q_{yi}, where $q_{yi} = Q_i^{(k)}(y)$. (Hint: you can express $\mathrm{P}(Y = y)$ as $\theta^y(1-\theta)^{1-y}$.)

(b) (1%) Compute the partial derivatives of the summation in (2) with respect to $\theta, \theta_{j0}, \theta_{j1}$.

(c) (1%) Solve for the values of $\theta, \theta_{j0}, \theta_{j1}$ that maximize (2). (Hint: The final solutions will have a reasonably simple form. To simplify the expressions you will need to make use of the identity $\sum_{y=0}^{1} q_{yi} = \sum_{y=0}^{1} Q_i^{(k)}(y) = 1$.)

(d) (1%) Express $Q_i^{(k)}(y)$ as a function of $\theta, \theta_{j0}, \theta_{j1}$, x_{ij} and y.

(e) (1%) Combine the E and the M steps into one step by writing the updated parameter values $\theta^{(k+1)}, \theta_{j0}^{(k+1)}, \theta_{j1}^{(k+1)}$ as a function of the previous parameter values $\theta^{(k)}, \theta_{j0}^{(k)}, \theta_{j1}^{(k)}$ and x_{ij} and y.

Question 5 (Maximum likelihood learning—Naive Bayes)

In this exercise you will write simple MATLAB functions for learning maximum likelihood Naive Bayes models from training data, and classifying text vectors with these models. In particular, you will implement the EM algorithm derived in Question 4. For this question you will consider training and test data generated according to the following generative model.

Data generation:

```
n = 5                  % dimension
t = 30                 % training size
p = 0.5                % prob y=1
p0 = 0.33*ones(1,n)    % prob x_j=1 given y=0
p1 = 0.67*ones(1,n)    % prob x_j=1 given y=1
pm = 0.67              % prob y missing
y = rand(t,1) > p      % true labels
pos = find(y == 1)
neg = find(y == 0)
X = rand(t,n)
X(neg,:) = X(neg,:) <= repmat(p0,length(neg),1)  % positive egs
X(pos,:) = X(pos,:) <= repmat(p1,length(pos),1)  % negative egs
b = rand(t,1) <= pm
z = (~b&y)-b           % blocked labels (-1 means unobserved)
```

(a) (1%) Write a MATLAB function `classify(Xtest,p,p0,p1)` which takes a $te \times n$ matrix of test vectors `Xtest` and the $2n+1$ parameters of a Naive Bayes model—one scalar parameter `p` and two $1 \times n$ vectors of parameters `p0` and `p1`—and returns a $te \times 1$ vector of classifications `ytest` on the test vectors. Classifications are to be determined according to the optimal decision rule

$$\hat{y} = 1 \quad \Leftrightarrow \quad \mathrm{P}(Y = 1|x_1, ..., x_n) \; \geq \; \mathrm{P}(Y = 0|x_1, ..., x_n)$$

where `p` specifies θ in the Naive Bayes model, `p0` specifies $\{\theta_{j0}\}_{j=1}^{n}$ and `p1` specifies $\{\theta_{j1}\}_{j=1}^{n}$ (see Question 1). Your function must handle `Xtest`, `p`, `p0` and `p1` with arbitrary dimensions te and n.

(b) (1%) Write a MATLAB function `[p,p0,p1] = maxlike(X,z)` which takes a $t \times n$ matrix of training vectors `X` and a $t \times 1$ vector of partially blocked training labels `z`, and returns the parameters of a Naive Bayes model that maximizes the likelihood *only* of the training vectors with *observed* labels (i.e. ignoring any training vectors that have a label $z_i = -1$). The parameters returned are one scalar `p` and two $1 \times n$ vectors `p0` and `p1`. (Hint: You do not use EM for this part—only simple plug-in formulas.) Your function must handle `X` and `z` with arbitrary dimensions t and n.

(c) (2%) Write a MATLAB function `[p,p0,p1] = em(X,z,tol)` which takes a $t \times n$ matrix of training vectors `X`, a $t \times 1$ vector of partially blocked training labels `z` and scalar tolerance `tol`, and returns the parameters of a Naive Bayes model that maximizes the likelihood of *all* the training vectors; i.e. including those vectors with *unobserved* labels ($z_i = -1$). This function is to be implemented using the EM algorithm derived in Question 4. EM should be run until the maximum change in any parameter is less than `tol`. (You can start EM with a random parameter vector, although it might be interesting to start it with the parameter vector returned by `maxlike`.) Your function must handle `X` and `z` with arbitrary dimensions t and n.

(d) (1%) Compare the quality of these two maximum likelihood training algorithms by testing the accuracy of the classifiers they produce as follows.

A: Generate a random training set `X`, `z` using the generative model outlined above, and solve for optimal Naive Bayes parameters using the two procedures:

```
[q,q0,q1] = maxlike(X,z)
[r,r0,r1] = em(X,z,0.01)
```

B: Use the `classify` function to label the training data using each learned model `[q,q0,q1]` and `[r,r0,r1]`. Report the *mean* misclassification error that each of these models obtain on the training data *with respect to the original (hidden) y-labels.*

C: Generate `te = 1000` test examples from the same generative model and report the mean misclassification error of both models on *test* data.

Repeat parts A, B and C 100 times and accumulate the mean misclassification errors for both models in two matrices: one for training error and the other for test error. Report the *averages* for each kind of mean error at each degree in two tables (one training error and the other testing error).