CMPUT 466/551—Machine Learning

Assignment 2

Winter 2006
Department of Computing Science
University of Alberta

Due: 23:59:59, *Thursday, February 16*
Worth: 15% of final grade
Instructor: Dale Schuurmans, Ath409, x2-4806, dale@cs.ualberta.ca

---

Note These 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 and all plots, tables, and explanations **to the TA, Chonghai Wang, at chonghai@cs.ualberta.ca** with a subject heading "CMPUT 466/551 A2 solutions".

Question 1 (Learning real classifiers—support vector machines)

In this exercise you will write simple Matlab functions to learn maximum margin linear discriminants and test them on simulated data. You will need to be familiar with `quadprog`.

We will consider three ways of generating data $\mathbf{X}$, `y`. The training data will have the form

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & & \vdots \\ x_{t,1} & \cdots & x_{t,n} \end{bmatrix} \quad \mathtt{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix} \quad \text{for } x_{i,j} \in I\!R \text{ and } y_i \in \{-1, 1\}.$$

Data generation:

```
n = 2                 % dimension
t = 10                % training size
u = ones(n,1)         % target weights   (models 1 & 2)
v = 0.5*n             % target offset    (models 1 & 2)
p_pos = 0.5           % prob of positive example
mu_pos = ones(n,1)    % mean loc for pos (model 3)
mu_neg = zeros(n,1)   % mean loc for neg (model 3)
```

Generative model 1: target linear discriminant

```
X = rand(t,n)
y = sign( X * u - v )
```

Generative model 2: target quadratic discriminant

```
X = rand(t,n)
y = sign( X.^2 * u - v )
```

Generative model 3: noisy linear discriminant (Naive Bayes—Gaussian)

```
X = randn(t,n)
y = 2 * (rand(t,1) < p_pos) - 1
pos = find(y > 0)
neg = find(y < 0)
X(pos,1) = X(pos,1) + mu_pos(1); X(pos,2) = X(pos,2) + mu_pos(2)
X(neg,1) = X(neg,1) + mu_neg(1); X(neg,2) = X(neg,2) + mu_neg(2)
```

(a) (1%) Write a Matlab function `[w,b] = maxL2marg(X,y)` which takes a $t \times n$ matrix $\mathbf{X}$ and $t \times 1$ vector of target labels `y` and returns: an $n \times 1$ vector of weights `w` and a scalar offset `b`, corresponding to the maximum L_2 margin linear discriminant classifier $\hat{y} = \text{sign}(\mathtt{w} \cdot \mathbf{x} - \mathtt{b})$.

Your function must be able to handle arbitrary n and t.

(b) (1%) Write a Matlab function `[yhat] = classify(Xtest,w,b)` which takes a $te \times n$ matrix `Xtest`, an $n \times 1$ vector of weights `w`, and a scalar `b`, and returns a $te \times 1$ vector of classifications `yhat` on the test patterns.

Your function must be able to handle arbitrary n and te.

(c) (1%) Write a Matlab function `[w,b] = softL2marg(X,y,c)` which takes an additional scalar argument `c` and returns `w` and `b` corresponding to the maximum "soft" margin linear discriminant classifier.

Recall that the maximum soft margin discriminant is defined by a weight vector `w` and offset `b` that minimizes $\|\mathtt{w}\|_2^2 + \mathtt{c}\sum_{i=1}^{t} \mathtt{s}_i$ subject to the constraints $y_i(\mathtt{w}\cdot\mathbf{x}_i - \mathtt{b}) \geq 1 - \mathtt{s}_i$ and $\mathtt{s}_i \geq 0$ for $i = 1, ..., t$. Here the $\mathtt{s}_i$ are new "slack" variables that allow the minimum margin bound to be violated by an amount $\mathtt{s}_i$ on example i. These slacks allows the learning algorithm to "give up" on a few training points to acheive a better minimum margin on the remaining points.

Your function must be able to handle arbitrary n and t.

(d) (1%) For each of the generative models 1, 2 and 3:

A: Generate a random training set $\mathbf{X}$, `y` using the model, and solve for each kind of discriminant function: `[wm, bm] = maxL2marg(X,y)`, `[ws, bs] = softL2marg(X,y,1)`,

B: Produce a 2D plot of the training data and the two hypotheses corresponding to `wm,bm` and `ws,bs`.

```
clf; axis([0 1 0 1]); hold; axis('square');
pos = find(y > 0); plot(X(pos,1)',X(pos,2)','g+');  % pos examples
neg = find(y < 0); plot(X(neg,1)',X(neg,2)','rx');  % neg examples
plot([0 1],[v v-u(1)]/u(2),'k:');                   % target discr
plot([0 1],[bm bm-wm(1)]/wm(2),'b-');               % max marg
plot([0 1],[bs bs-ws(1)]/ws(2),'m-');               % max soft mrg
print -deps experiment.2.1.<k>.ps
```

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

		misclassification error
classifier	`wm,bm`	
	`ws,bs`	

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

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

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

Question 2 (Learning support vector machines—dual formulation)

In this exercise you will write simple Matlab functions to learn maximum margin linear discriminants once again. However, this time you will implement the "dual" form of the algorithms. Here we will use the same generative models as in Question 1. As before, you will need to be familiar with `quadprog`.

(a) (1%) Write a Matlab function `[lambda,b] = dualL2marg(X,y)` which takes a $t \times n$ matrix $\mathbf{X}$ and $t \times 1$ vector of target labels `y` and returns: an $1 \times t$ vector of Lagrange multipliers `lambda` and a scalar offset `b`, corresponding to the maximum L_2 margin linear discriminant classifier $\hat{y} = \text{sign}\left((\sum_{i=1}^{t} \lambda_i y_i \mathbf{x}_i \cdot \mathbf{x}) - \mathtt{b}\right)$.

Note: For `dualL2marg` all you need to do is compute the vector of Lagrange multipliers $\boldsymbol{\lambda}$ that *maximizes* the objective

$$L(\boldsymbol{\lambda}) \;=\; \sum_{i=1}^{t} \lambda_i - \frac{1}{2} \sum_{i=1}^{t} \sum_{k=1}^{t} \lambda_i \lambda_k y_i y_k \, \mathbf{x}_i \cdot \mathbf{x}_k \tag{1}$$

subject to the constraints $\sum_{i=1}^{t} \lambda_i y_i = 0$ and $\lambda_i \geq 0$. You can do this using Matlab's `quadprog` operator to recover the vector of Lagrange multipliers $\boldsymbol{\lambda}$. To recover the offset value `b`, just solve for `b` in the equation: $\lambda_k \left(y_k \left((\sum_{i=1}^{t} \lambda_i y_i \mathbf{x}_i \cdot \mathbf{x}_k) - \mathtt{b}\right) - 1\right) = 0$ corresponding to the largest Lagrange multiplier λ_k.

Your function must be able to handle arbitrary n and t.

(b) (1%) Write a Matlab function `[yhat] = dualclassify(Xtest,lambda,b,X,y)` which takes a $te \times n$ matrix `Xtest`, a $1 \times t$ vector `lambda`, a scalar `b`, a $t \times n$ matrix `X`, and a $t \times 1$ vector `y`, and returns a $te \times 1$ vector of classifications `yhat` on the test patterns.

Your function must be able to handle arbitrary n, t, and te, and must not explicitly compute a weight vector `w` (instead you must use the Lagrange multiplier vector $\boldsymbol{\lambda}$, as shown above).

(c) (1%) Write a Matlab function `[lambda,b] = dualsoftL2(X,y,c)` which takes an additional scalar argument `c` and returns `lambda` and `b` corresponding to the maximum "soft" margin linear discriminant classifier.

Note: For `softL2marg` all you have to do is compute the vector of Lagrange multipliers $\boldsymbol{\lambda}$ that maximizes the same objective as above (1) subject to the same set of constraints, except for the slight modification that $0 \leq \lambda_i \leq \mathtt{c}$. This recovers the vector of Lagrange multipliers `lambda`. (Note that the slack variables $\mathtt{s}_i$ actually disappear in the dual formulation.) To recover the offset value `b`, just use the same procedure as in Part (a) (using any λ_k such that $0 < \lambda_k < \mathtt{c}$).

Your function must be able to handle arbitrary n and t.

(d) (1%) For each of the generative models 1, 2 and 3:

A: Generate a random training set $\mathbf{X}$, `y` using the model, and solve for each kind of discriminant function: `[lm, bm] = dualL2marg(X,y)`, `[ls, bs] = dualsoftL2(X,y,1)`,

B: Produce a 2D plot of the training data and the two hypotheses corresponding to `lm,bm` and `ls,bs`.

```
clf; axis([0 1 0 1]); hold; axis('square');
pos = find(y > 0); plot(X(pos,1)',X(pos,2)','g+');  % pos examples
neg = find(y < 0); plot(X(neg,1)',X(neg,2)','rx');  % neg examples
plot([0 1],[v v-u(1)]/u(2),'k:');                   % target discr
Z=X; for j = 1:n, Z(:,j) = X(:,j) .* y; end
wm = lm * Z;                                        % max margin:
plot([0 1],[bm bm-wm(1)]/wm(2),'b-');               %     discrim
spm = find(lm > 0); plot(X(spm,1)',X(spm,2)','co'); %     supports
ws = ls * Z;                                        % soft margin:
plot([0 1],[bs bs-ws(1)]/ws(2),'m-');               %      discrim
sps = find(ls > 0); plot(X(sps,1)',X(sps,2)','mo'); %     supports
print -deps experiment.2.2.<k>.ps
```

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

		misclassification error
classifier	`lm,bm`	
	`ls,bs`	

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

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

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

Question 3 (Dual support vector machines—Kernel classifiers)

In this exercise you will write simple Matlab functions to learn maximum margin *kernel* classifiers. You will need to be familiar `feval`. Once you have written these programs, you will then apply them to the real world classification problem of recognizing images of handwritten digits.

(a) (1%) Write a Matlab function [K] = `polykernel`($\mathtt{X}_1, \mathtt{X}_2, \mathtt{d}$) which takes a $t_1 \times n$ matrix $\mathtt{X}_1$, a $t_2 \times n$ matrix $\mathtt{X}_2$, and a degree parameter `d`, and returns a $t_1 \times t_2$ matrix `K` of kernel values for the polynomial kernel. $\mathtt{K}_{ij}$ is the polynomial kernel value obtained by comparing row vector $\mathbf{x}_{i1}$ from $\mathtt{X}_1$ with row vector $\mathbf{x}_{j2}$ from $\mathtt{X}_2$; that is $\mathtt{K}_{ij} = (\mathbf{x}_{i1} \cdot \mathbf{x}_{j2} + 1)^{\mathtt{d}}$. Your function must be able to handle arbitrary t_1, t_2, n and `d`.

(b) (1%) Write a Matlab function `[lambda,b] = kernelL2marg(X,y,c,kernfun,par)` which takes as input $t \times n$ matrix of observations `X`, a $t \times 1$ vector of target labels `y`, a slack parameter `c`, the name of a kernel function `kernfun`, and a parameter value for the kernel `par`. The outputs are a $t \times 1$ vector of Lagrange multipliers `lambda` and a scalar offset `b` corresponding to the maximum margin classifier in the feature space.

Note: This function is the same as `dualsoftL2` (Question 2(c)) except that instead of taking simple inner products between row vectors, $\mathbf{x}_i \cdot \mathbf{x}_j$, you use the value calculated by the `kernfun`, $k(\mathbf{x}_i, \mathbf{x}_j)$. An example call would be `[lambda,b] = kernelL2marg(X,y, 100,'polykernel',2)` using the `polykernel` function written in Part(a) above.

Your function must be able to handle arbitrary n, t, te and `d`.

(c) (1%) Write a Matlab function `[yhat] = kernelclassify(Xtest,lambda,b,X,y, kernfun,par)` which takes as input a $te \times n$ matrix of test observations `Xtest`, a $t \times 1$ vector of Lagrange multipliers `lambda`, a scalar offset `b`, a $t \times n$ matrix of training observations `X`, a $t \times 1$ vector of training labels `y`, the name of a kernel function `kernfun`, and a parameter for the kernel function `par`. The output is a $te \times 1$ vector of classifications `yhat` on the test patterns.

Your function must be able to handle arbitrary n, t, te and `d`.

(d) (2%) On the course webpage, download the file data2.mat. Then type "`load data2.mat`" in Matlab. This will load the training data into a matrix `X` and a vector `y` and the test data into a matrix `Xtest` and a vector `ytest`. Each row of the matrices corresponds to a 256 dimensional vector representing a 16×16 grayscale image of a handwritten digit. The images are of handwritten '2's and '3's. The corresponding entry in the associated y-vector gives a label indicating which digit the image represents, where -1 corresponds to '2' and +1 corresponds to '3'.

Note: that you can easily view the training images in Matlab by first typing "`colormap gray`" to set up the colormap, and then viewing image `i` in matrix `X` (or `Xtest`) by typing "`imagesc(reshape(X(i,:),16,16)')`".

The goal of this question is to learn a function which can accurately distinguish images of '2's from images of '3's. Here, functions will be learned on the training data and then tested on the separate test data.

Use the following parameters to learn different classification functions:

```
[ll bl] = kernelL2marg(X,y,100,'polykernel',1)
[lq bq] = kernelL2marg(X,y,100,'polykernel',2)
[lc bc] = kernelL2marg(X,y,100,'polykernel',3)
```

Report the number of misclassification errors that each of these three functions make on both the training and the test data (in a 3×2 table).