Massachusetts Institute of Technology

6.867 Machine Learning, Fall 2006

Problem Set 2: Solutions

1. (a) **(5 points)** From the lecture notes (Eqn 14, Lecture 5), the optimal parameter values for linear regression given the matrix of training examples $\mathbf{X}$ and the corresponding response variables $\mathbf{y}$ is:

$$\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

The quantity $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is also known as the pseudo-inverse of $\mathbf{X}$, and often occurs when dealing with linear systems of equations. When $\mathbf{X}$ is a square matrix and invertible, it is exactly the same as the inverse of $\mathbf{X}$.

MATLAB provides many ways of achieving our desired goal. We can directly write out the expression above or use the function `pinv`. Here are the functions `linear_regress` and `linear_pred`:

```
function theta = linear_regress(y,X)
  theta = pinv(X)*y;

function y_hat = linear_pred(theta,X)
   y_hat = X*theta;
```

In `linear_regress`, note that we are not calculating θ_0 separately. This differs from the description in the lecture notes where the training examples were explicitly padded with 1's, allowing us to introduce an offset θ_0. Instead, we will use a feature mapping to achieve the same effect.

(b) **(2 points)** Before we describe the solution, we first describe how the dataset was created. This may help you appreciate why some feature mappings may work better than others. The $\mathbf{x}$ values were created by sampling each coordinate uniformly at random from (-1,1):

`X_in = rand(1000,3)*2 - 1`

Given a particular $\mathbf{x}$, the corresponding y_{true} and y_{noisy} values were created as follows:

$$\begin{aligned} y_{true} &= -10\log x_1^2 - 15\log x_2^2 - 7.5\log x_3^2 + 2 \\ y_{noisy} &= y_{true} + \epsilon \qquad \epsilon \sim N(0,100) \end{aligned}$$

To evaluate the performance of linear regression on given training and test sets, we created the function `test_linear_reg` which combines the regression, prediction, and evaluation steps. You may, of course, do it in some other way:

```
function err = test_linear_reg(y_train, X_train, y_test, X_test)
% train linear regression using X_train and y_train
% evaluate the mean squared prediction error on X_test and y_test

  theta = linear_regress(y_train, X_train);
  y_hat = linear_pred(theta,X_test);
  yd = y_hat - y_test;
  err = sum(yd.^2)/length(yd);      %err = Mean Squared Prediction Error
```

Using this, we can now calculate the mean squared prediction error (MSPE) for the two feature mappings:

```
>> X1 = feature_mapping(X_in,1);
>> test_linear_reg(y_noisy, X1, y_true, X1)
ans =  1.5736e+003
>> X2 = feature_mapping(X_in,2);
>> test_linear_reg(y_noisy, X2, y_true, X2)
ans =    0.5226
```

The two sets of errors are 1573.6 (ϕ_1) and 0.5226 (ϕ_2), respectively. Unsurprisingly, the mapping ϕ_2 performs much better than ϕ_1– it is exactly the space in which the relationship between $\mathbf{X}$ and $\mathbf{y}$ is linear.

(c) **(10 points)** Recall from the notes (Eqn 8, lecture 6), that the desired quantity we need to maximize is

$$\frac{\mathbf{v}^T A A \mathbf{v}}{1 + \mathbf{v}^T A \mathbf{v}}$$

where $A = (\mathbf{X}^T\mathbf{X})^{-1}$. In the notes, an offset parameter is explicitly assumed so that $\mathbf{v} = [\mathbf{x}^T, 1]^T$. However, in our case this is not necessary and so $\mathbf{v} = \mathbf{x}$.

Active learning may, in general, select the same point $\mathbf{x}$ to be observed repeatedly. Each of these observations corresponds to a different y_{noisy}. However, due to practical limitations, we had supplied you with only one set of y_{noisy} values. Thus, if some $\mathbf{x}_i$ occurs repeatedly in `idx`, you will need to use the same $y_{noisy,\, i}$ for each occurrence of $\mathbf{x}_i$. Alternatively, you could change your code so as to disallow reptitions in `idx`. This is the option we have chosen here.

Given any feature space, the criterion above will aim to find points far apart in that space. However, these points may not be far apart in the feature space where classification actually occurs. This is of particular concern when the latter feature space might not be easily accessible (e.g., when using a kernel like the radial basis function kernel).

Here's the active learning code:

```
function idx = active_learn(X,k1,k2)
idx = 1:k1;
N = size(X,1);
for i=1:k2
    var_reduction = zeros(N,1);
    X1 = X(idx,:);
    A = inv(X1'*X1);
    AA = A*A;
    for j=1:N
        if ismember(j,idx)  %this is the part where we disallow repititions
            continue;
        end
        v = X(j,:);
        a = v*AA*v' / ( 1 + (v*A*v'));
        var_reduction(j) = a;
    end
    [a, aidx] = max(var_reduction);
    idx(end+1) = aidx;
end
```

Using it, we can now compute the desired quantities:

```
>> idx1 = active_learn(X1,5,10)
idx1 =    1  2  3  4  5  437  270  928  689  803  670  979  932  558  490
>> test_linear_reg(y_noisy(idx1), X2(idx1,:), y_true, X2)
ans =  2.5734e+003
>> idx2 = active_learn(X2,5,10)
idx2 =   1  2  3  4  5  955  803  270  558  628  490  283  592  761  414
>> test_linear_reg(y_noisy(idx2), X2(idx2,:), y_true, X2)
ans =   24.4643
```

Thus, the MSPE when using ϕ_1 for active learning chooses points that are not well-placed (for this particular dataset); ϕ_2 performs much better. Note that the feature mapping used to perform the regression (and evaluation) is the same for both (ϕ_2) so the performance difference is due to the points chosen by active learning.

The answers change when repititions are allowed (MSPE for ϕ_1= 117.47 ; for ϕ_2 = 25.25), but they still illustrate our concept.

For completeness' sake, the answers for the original version of the problem are: (ϕ_1: 2618.9 (repeats), 2618.6 (no repeats)) and (ϕ_2: 25.2546 (no repeats) and 24.4643 (repeats)).

(d) **(4 points)** This error will vary, depending upon the number of iterations you perform and the random points selected. In my simulations, the value of error (using mapping ϕ_2 for regression and evaluation) was 33.694. When this simulation was run for a 1000 runs, the error was close to 25.34 . Clearly, it seems to be much better to perform random sampling than perform active learning in ϕ_1's space. This may seem surprising at first, but is completely understandable: in the space where regression is performed (ϕ_2) the points chosen by performing active learning in the ϕ_1 space are not far apart at all, and are thus particularly bad points to be sampled.

For completeness' sake, the answer for the original version of the problem is: (ϕ_1: 2218.9 , ϕ_2: 33.694).

(e) **(4 points)** The figures are shown in Fig 1. For clarity's sake, we have only plotted the last 10 points of `idx` (since the first 5 are the same across all cases).

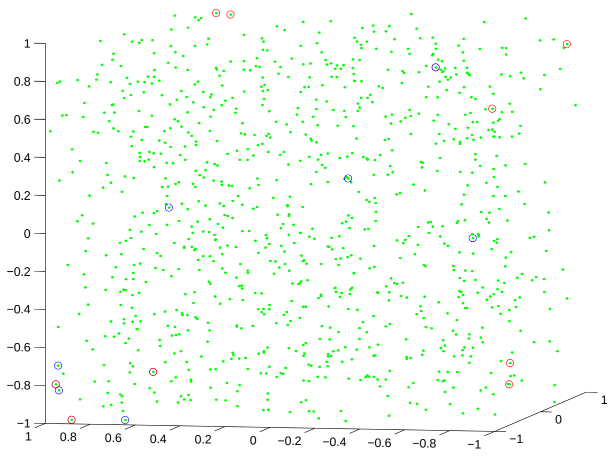
In the original feature space (fig a), the points selected by active learning in the ϕ_1 space are spread far apart, as expected. However, as part (b) showed, a better fit to data is obtained by using ϕ_2. In this space (fig b), the points selected by active learning in the ϕ_1 space are very closely bunched together. The points selected by active learning in the ϕ_2 space are, in contrast, spread far apart. This helps explain why the points learned using active learning on the ϕ_1 space did not lead to good performance in the regression step.

2. (a) **5 points** The function $f(t) = -\beta t^2/2$ monotonically decreases with t (for $\beta > 0$). The function $g(t) = e^t$ monotonically increases with t. Thus, the function $g\, o\, f(t) = h(t) = e^{-\beta t^2/2}$ monotonically decreases with t. As such, the RBF kernel $K(\mathbf{x}, \mathbf{x}') = e^{-\frac{\beta}{2}\|\mathbf{x}-\mathbf{x}'\|^2}$ defines a Gram matrix that satisfies the conditions of Michelli's theorem and is hence invertible.

(b) **5 points** Let $\mathbf{A} = (\lambda \mathbf{I} + \mathbf{K})^{-1}\mathbf{y}$. Then,

$$\lim_{\lambda \to 0} \mathbf{A} = \mathbf{K}^{-1}\mathbf{y}$$

Since $\mathbf{K}$ is always invertible, this limit is always well-defined. Now, we have $\hat{\alpha}_t = \lambda \mathbf{A}_t$ where $\mathbf{A}_t$ is the t-th element of $\mathbf{A}$. We then have:


(a) In ϕ_1 space

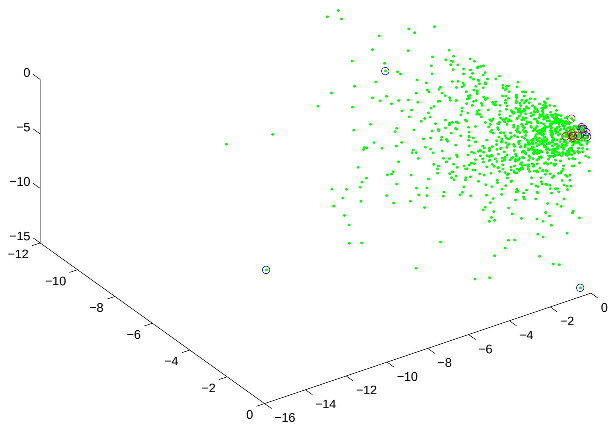

(b) In ϕ_2 space

Figure 1: The red circles correspond to points chosen by performing active learning on the ϕ_1 space and the blue ones correspond to those chosen by performing active learning on the ϕ_2 space.
Only the last 10 points of `idx` are shown for each; the first 5 points are the same.

$$\begin{aligned} y(\mathbf{x}) &= \textstyle\sum_{t=1}^{n} (\hat{\alpha}_t/\lambda) K(\mathbf{x}_t, \mathbf{x}), \quad \mathit{or} \\ y(\mathbf{x}) &= \textstyle\sum_{t=1}^{n} (\lambda \mathbf{A}_t/\lambda) K(\mathbf{x}_t, \mathbf{x}), \quad \mathit{or} \\ y(\mathbf{x}) &= \textstyle\sum_{t=1}^{n} \mathbf{A}_t K(\mathbf{x}_t, \mathbf{x}) \end{aligned}$$

We then have,

$$\begin{aligned} \lim_{\lambda\to 0} y(\mathbf{x}) &= \textstyle\sum_{t=1}^{n} (\lim_{\lambda\to 0} \mathbf{A}_t) K(\mathbf{x}_t, \mathbf{x}), \quad \mathit{or} \\ \lim_{\lambda\to 0} y(\mathbf{x}) &= \textstyle\sum_{t=1}^{n} \mathbf{B}_t K(\mathbf{x}_t, \mathbf{x}), \quad \text{where } \mathbf{B} = \mathbf{K}^{-1}\mathbf{y} \end{aligned}$$

Thus, in the required limit, the function $y(\mathbf{x}) = \sum_{i=1}^{n} \mathbf{B}_t K(\mathbf{x}_t, \mathbf{x})$

(c) **10 points** To prove that the training error is zero, we need to prove that $y(\mathbf{x}_t) = y_t$ for $t = 1, \ldots, n$. From part (b), we have

$$\begin{aligned} y(\mathbf{x}_t) &= \textstyle\sum_{i=1}^{n} \mathbf{B}_i K(\mathbf{x}_i, \mathbf{x}_t), \quad \mathit{or} \\ y(\mathbf{x}_t) &= \textstyle\sum_{i=1}^{n} (\sum_{j=1}^{n} y_j \mathbf{K}^{-1}(i,j)) K(\mathbf{x}_i, \mathbf{x}_t) \quad \mathit{or} \\ y(\mathbf{x}_t) &= \textstyle\sum_{j=1}^{n} y_j \sum_{i=1}^{n} \mathbf{K}^{-1}(i,j) K(\mathbf{x}_i, \mathbf{x}_t), \quad \mathit{or} \\ y(\mathbf{x}_t) &= \textstyle\sum_{j=1}^{n} y_j \delta(j,t), \quad \mathit{or} \\ y(\mathbf{x}_t) &= y_t \end{aligned}$$

where $\mathbf{K}^{-1}(i,j) = (i,j)$-th entry of $\mathbf{K}^{-1}$ and

$$\delta(i,j) = \begin{cases} 0 & \text{for } i \neq j \\ 1 & \text{for } i = j \end{cases}$$

Here, we made use of the fact that $K(\mathbf{x}_i, \mathbf{x}_t) = K(\mathbf{x}_t, \mathbf{x}_i)$ and that if $A = B^{-1}$ then $\sum_i A(t,i)B(i,j) = \delta(t,j)$.

(d) **5 points** Sample code for this problem is shown below:

```
Ntrain  = size(Xtrain,1);
Ntest = size(Xtest,1);
```

```
for i=1:length(lambda),
   lmb = lambda(i);

   alpha = lmb * ((lmb*eye(Ntrain) + K)^-1) * Ytrain;
   Atrain = (1/lmb) * repmat( alpha', Ntrain, 1);
   yhat_train = sum(Atrain.*K,2);

   Atest = (1/lmb) * repmat( alpha', Ntest, 1);
   yhat_test  = sum(Atest.*(Ktrain_test'), 2);

   E(i,:) = [mean((yhat_train-Ytrain).^2),mean((yhat_test-Ytest).^2)];
end;
```

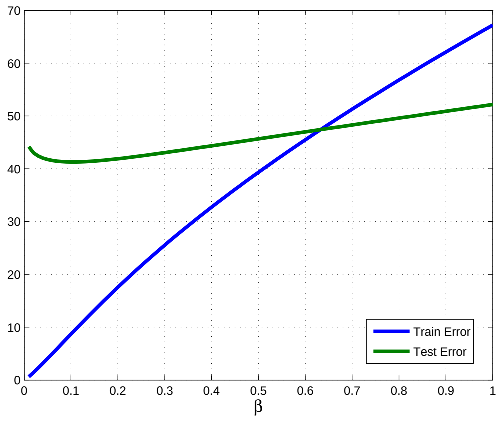


Figure 2: Training and test error for Prob # 2(e)

The resulting plot is shown in Fig 2. As can be seen the training error is zero at $\lambda = 0$ and increases as λ increases. The test error initially decreases, reaches a minimum around 0.1, and then increases again. This is exactly as we would expect. $\lambda \approx 0$ results in over-fitting (the model is too powerful). Our regression function has a low bias but high variance. By increasing λ we constrain the model, thus increasing the training error. While the regularization increases bias, the variance decreases faster, and we generalize better. High values of λ result in under-fitting (high bias, low variance) and both training error and test errors are high.

3. (a) Observe that θ is only a sum of $y_t\phi(X_t)$'s, so we can just store the coefficients:

$$\theta = \sum_{t=1}^{n} w_t y_t \phi(X_t).$$

We can update w_t's by incrementing w_t by one when a mistake is made on example t. Classifying new examples means evaluating:

$$\theta^T\phi(X) = \sum_{t=1}^{n} w_t y_t K(X_t, X),$$

which only involves kernel operations.

(b) The most straightforward proof: use the regression argument to show that you can fit the points exactly and not only achieve the correct sign, but the value of the discriminant function can be made ± 1 for every training example.

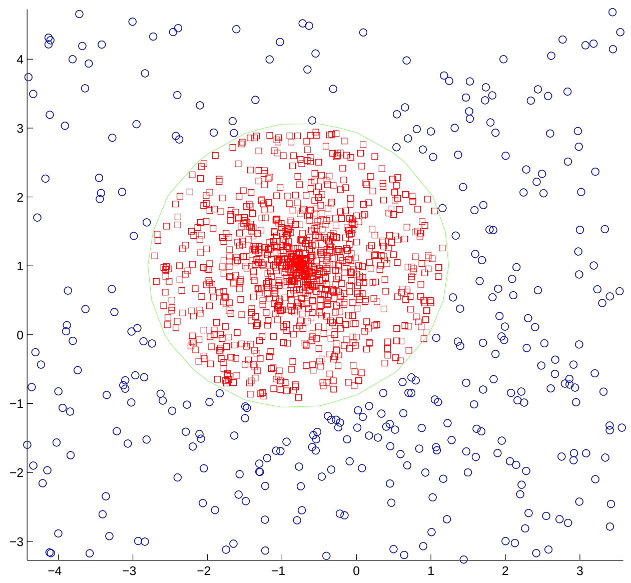
(c) Here is my solution.

```
% kernel=‘$(1 + \mathbf{transpose}(x_i)x_j)^d$’; $d = 5$; % polynomial kernel
% kernel=‘$\mathbf{exp}(-\mathbf{transpose}(x_i - x_j)(x_i - x_j)/(2s^2))$’; $s = 3$; % radial basis function kernel
function $\alpha$=TRAIN_KERNEL_PERCEPTRON($X$, $y$, kernel)
    $n, d = \mathbf{size}(X)$;
    $K = []$;
    for $i = 1 : n$
        $x_i = X(i, :)'$;
        for $j = 1 : n$
            $x_j = X(j, :)'$;
            $K(i, j) = \mathbf{eval}(\text{kernel})$;
        end
    end

    $\alpha = \mathbf{zeros}(n, 1)$;
    mistakes $= 1$;
    while mistakes $> 0$
        mistakes $= 0$;
        for $i = 1 : n$
            if $\alpha' K(:, i)y(i) \leq 0$
                $\alpha(i) = \alpha(i) + y(i)$;
                mistakes $=$ mistakes $+ 1$;
            end
        end
    end
function $f$ =DISCRIMINANT_FUNCTION($\alpha$, $X$, kernel, $X_{\text{test}}$)
    $n, d = \mathbf{size}(X)$;
    $K = []$;
    for $i = 1 : n$
        $x_i = X(i, :)'$;
        $x_j = X_{\text{test}}$;
        $K(i) = \mathbf{eval}(\text{kernel})$;
    end
    $f = \alpha' K$;
```

(d) The original dataset requires $d = 4$; the new dataset requires $d = 2$. An RBF will easily separate either dataset.

```
≫ load_p3_a
“X” and “y” loaded.
≫ kernel = ‘(1 + transpose(x_i) * x_j)^2’;
≫ α = TRAIN_KERNEL_PERCEPTRON(X, y, kernel);
≫ figure
≫ hold on
≫ plot(X(1 : 1000, 1), X(1 : 1000, 2), ‘rs’)
≫ plot(X(1001 : 2000, 1), X(1001 : 2000, 2), ‘bo’)
≫ plot_dec_boundary(α, X, kernel, [−4, −2], [0.5, 0.5], [2, 4])
```



$\gg$ kernel = '$\mathbf{exp}(-\mathbf{transpose}(x_i - x_j) * (x_i - x_j)/18)$';
$\gg$ α = TRAIN_KERNEL_PERCEPTRON(X, y, kernel);
$\gg$ **figure**
$\gg$ **hold on**
$\gg$ $\mathbf{plot}(X(1:1000,1), X(1:1000,2), \text{'rs'})$
$\gg$ $\mathbf{plot}(X(1001:2000,1), X(1001:2000,2), \text{'bo'})$
$\gg$ $\mathbf{plot\_dec\_boundary}(\alpha, X, \text{kernel}, [-4,-2], [0.5,0.5], [2,4])$