EE 556

HW #4 Solutions

#2)
(6.6, Haykin)

$K = [K(\underline{x}_i, \underline{x}_j)]$ is a square matrix.

Therefore, it can be written as:

$K = Q \Lambda Q^T$, where Λ is a diagonal matrix w/ the eigenvalues of K and Q is an orthogonal matrix whose columns are the associated eigenvectors (this is often called a "similarity transform").

Because $K(\underline{x}_i, \underline{x}_j)$ is positive definite, all the eigenvalues are non-negative.

We can write:

$$K(\underline{x}_i, \underline{x}_j) = (Q \Lambda Q^T)_{ij} = \sum_{l=1}^{m} (Q)_{il} (\Lambda)_{ll} (Q^T)_{lj}$$

$$= \sum_{l=1}^{m} Q_{il} \Lambda_{ll} Q_{jl}$$ (since for an orthogonal matrix, $Q = Q^T$).

Let $\underline{u}_i$ denote the i-th <u>row</u> of the matrix Q. (Note that $\underline{u}_i$ is <u>NOT</u> an eigenvector of K...). (The columns are the eigenvectors...).

Then, $K(\underline{x}_i, \underline{x}_j) = \underline{u}_i^T \Lambda \underline{u}_j$

$$= (\Lambda^{1/2} \underline{u}_i)^T (\Lambda^{1/2} \underline{u}_j)$$

By definition, $K(\underline{x}_i, \underline{x}_j) = \underline{\Phi}^T(\underline{x}_i)\,\underline{\Phi}(\underline{x}_j)$.

Therefore, we have: $\underline{\Phi}(\underline{x}_i) = \Lambda^{1/2}\underline{u}_i$, i.e.

the mapping from the input space to the feature space for a kernel SVM is given by:

$\Phi : \underline{x}_i \longrightarrow \Lambda^{1/2}\underline{u}_i$ (the mapping for the training vectors ...)

#3)

We employ proof by contradiction. Suppose there were two *distinct* minimum length solution vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ with $\mathbf{a}_1^t\mathbf{y} > 0$ and $\mathbf{a}_2^t\mathbf{y} > 0$. Then necessarily we would have $\|\mathbf{a}_1\| = \|\mathbf{a}_2\|$ (otherwise the longer of the two vectors would not be a *minimum* length solution). Next consider the average vector $\mathbf{a}_o = \frac{1}{2}(\mathbf{a}_1 + \mathbf{a}_2)$. We note that

$$\mathbf{a}_o^t\mathbf{y}_i = \frac{1}{2}(\mathbf{a}_1 + \mathbf{a}_2)^t\mathbf{y}_i = \frac{1}{2}\mathbf{a}_1^t\mathbf{y}_i + \frac{1}{2}\mathbf{a}_2^t\mathbf{y}_i \geq 0,$$

and thus $\mathbf{a}_o$ is indeed a solution vector. Its length is

$$\|\mathbf{a}_o\| = \|1/2(\mathbf{a}_1 + \mathbf{a}_2)\| = 1/2\|\mathbf{a}_1 + \mathbf{a}_2\| \leq 1/2(\|\mathbf{a}_1\| + \|\mathbf{a}_2\|) = \|\mathbf{a}_1\| = \|\mathbf{a}_2\|,$$

where we used the triangle inequality for the Euclidean metric. Thus $\mathbf{a}_o$ is a solution vector such that $\|\mathbf{a}_o\| \leq \|\mathbf{a}_1\| = \|\mathbf{a}_2\|$. But by our hypothesis, $\mathbf{a}_1$ and $\mathbf{a}_2$ are minimum length solution vectors. Thus we must have $\|\mathbf{a}_o\| = \|\mathbf{a}_1\| = \|\mathbf{a}_2\|$, and thus

$$\frac{1}{2}\|\mathbf{a}_1 + \mathbf{a}_2\| = \|\mathbf{a}_1\| = \|\mathbf{a}_2\|.$$

We square both sides of this equation and find

$$\frac{1}{4}\|\mathbf{a}_1 + \mathbf{a}_2\|^2 = \|\mathbf{a}_1\|^2$$

or

$$\frac{1}{4}(\|\mathbf{a}_1\|^2 + \|\mathbf{a}_2\|^2 + 2\mathbf{a}_1^t\mathbf{a}_2) = \|\mathbf{a}_1\|^2.$$

We regroup and find

$$\begin{aligned} 0 &= \|\mathbf{a}_1\|^2 + \|\mathbf{a}_2\|^2 - 2\mathbf{a}_1^t\mathbf{a}_2 \\ &= \|\mathbf{a}_1 - \mathbf{a}_2\|^2, \end{aligned}$$

and thus $\mathbf{a}_1 = \mathbf{a}_2$, contradicting our hypothesis. Therefore, the minimum-length solution vector is unique.

#4) Recall Wolfe-Dual problem:

$$\max_{\underline{\lambda}} \left\{ -\frac{1}{2} \sum_{x,x'} \lambda_x \lambda_{x'} t_x t_{x'} \underline{x}'^T \underline{x} + \sum_x \lambda_x \right\}$$

$$\text{s.t.} \quad \sum_x \lambda_x t_x = 0, \quad \underline{\lambda} \geq 0$$

If the data vectors are $\underbrace{\text{orthogonal, + unit norm}}_{\text{orthonormal}}$, $\underline{x}'^T \underline{x} = \delta_{x'x} \Rightarrow$

the function to be maximized reduces to:

$$-\frac{1}{2} \sum_x \lambda_x^2 + \sum_x \lambda_x$$

For now, ignore $\underline{\lambda} \geq 0$ + take the equality constraint into account via a Lagrange multiplier, μ.

I.e., form

$$L = -\frac{1}{2} \sum_x \lambda_x^2 + \sum_x \lambda_x - \mu \left(\sum_x \lambda_x t_x \right)$$

$$\frac{\partial L}{\partial \lambda_z} = -\lambda_z + 1 - \mu t_z = 0 \Rightarrow$$

$$\lambda_z = 1 - \mu t_z$$

Plugging into $\sum_x \lambda_x t_x = 0$ gives

$$\underbrace{\sum_x (1 - \mu t_x) t_x}_{=0} \Rightarrow \mu = \frac{\sum_x t_x}{N}, \quad N$$

the # of data points.

Since $t_x \in \{-1, +1\}$, $-1 < \mu < 1 \Rightarrow \lambda_x > 0, \forall x$, i.e. inequality constraint is automatically satisfied.

This also means that all training points are support vectors, in this case.

The solution thus has the form

$$\underline{w} = \sum_x \lambda_x t_x \underline{x} = \sum_x \left(1 - t_x \left(\frac{\sum_{x'} t_{x'}}{N} \right)\right) t_x \underline{x}$$

+

$$w_0 = t_x - \underline{w}^T \underline{x}, \quad \text{any } x.$$

$$= t_x - \lambda_x t_x = t_x - (1 - \mu t_x) t_x$$

$$= \mu = \boxed{\frac{\sum_x t_x}{N}}$$

Also, we can rewrite $\underline{w}$ as:

$$\underline{w} = \sum_x (1 - \mu t_x) t_x \underline{x}$$

$$\underbrace{\boxed{= \sum_x (t_x - \mu) \underline{x}}}$$

This is related to Hebbian learning...

#5) <u>Theorem</u>: Given M linearly independent vectors $\underline{x}_1, \ldots, \underline{x}_M$, $\underline{x}_i \in \mathbb{R}^N$, $M < N$, $\exists$ a vector $\underline{w}$ s.t. $\underline{w}^T \underline{x}_i > 0$, $i = 1, \ldots M$

<u>Proof</u>: The theorem statement is equivalent to the statement that

$$X^T \underline{w} = \underline{b} > \underline{0}$$

, that is, the right-hand side is a vector with strictly positive entries.

Here, $\underbrace{X}_{N \times M} = [\underline{x}_1, \ldots, \underline{x}_M]$.

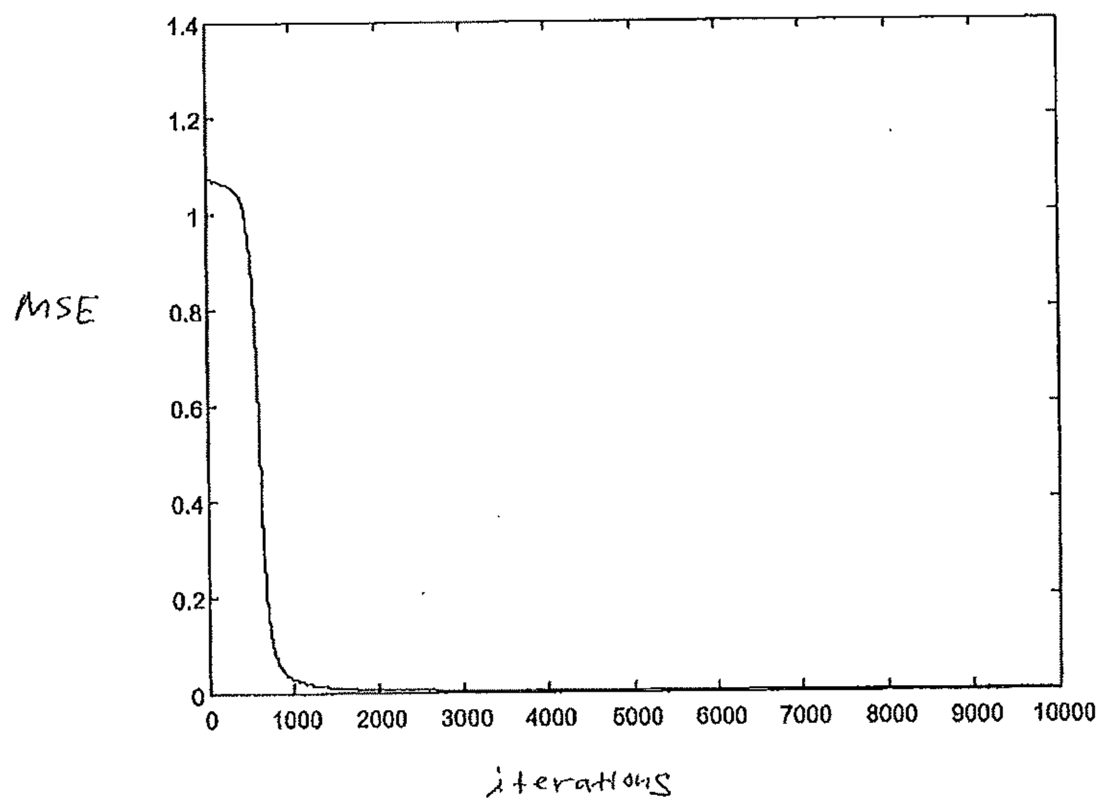
The matrix X has full column rank, M. Moreover, its <u>row</u> rank is also M. (see, e.g. $\boxed{\text{Strang}}$). This means that X^T has an M-dimensional column basis, However, note that the columns of X^T are M-dimensional vectors. This implies that the columns of X^T span $\mathbb{R}^M \Rightarrow$ <u>any</u> vector $\underline{b} \in \mathbb{R}^M$ is in the column space of X^T, including $\underline{b}$ with all positive entries.

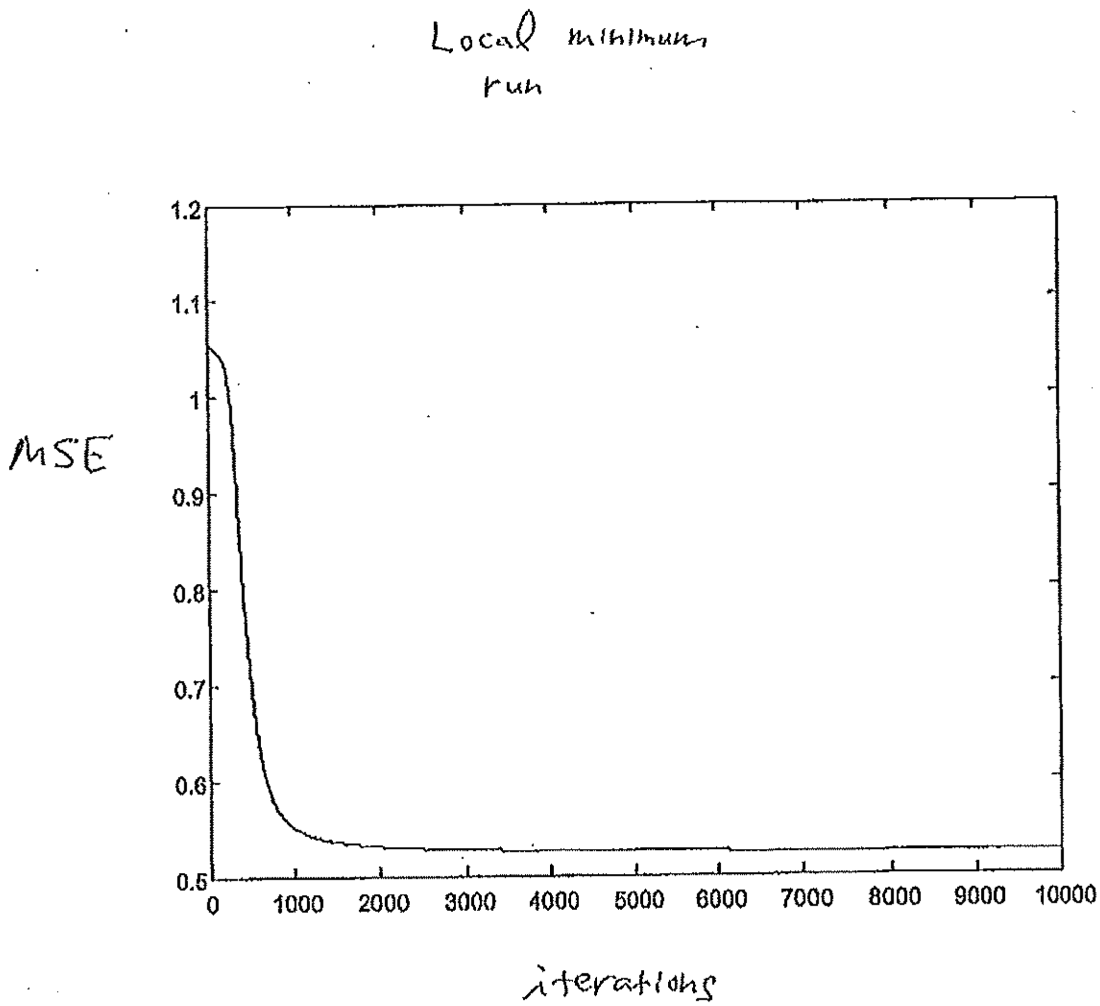
Thus, $\exists$ $\underline{w}$ s.t. $X^T \underline{w} = \underline{b} > \underline{0}$.

Matlab code for XOR problem -- updates weights with each sample presentation

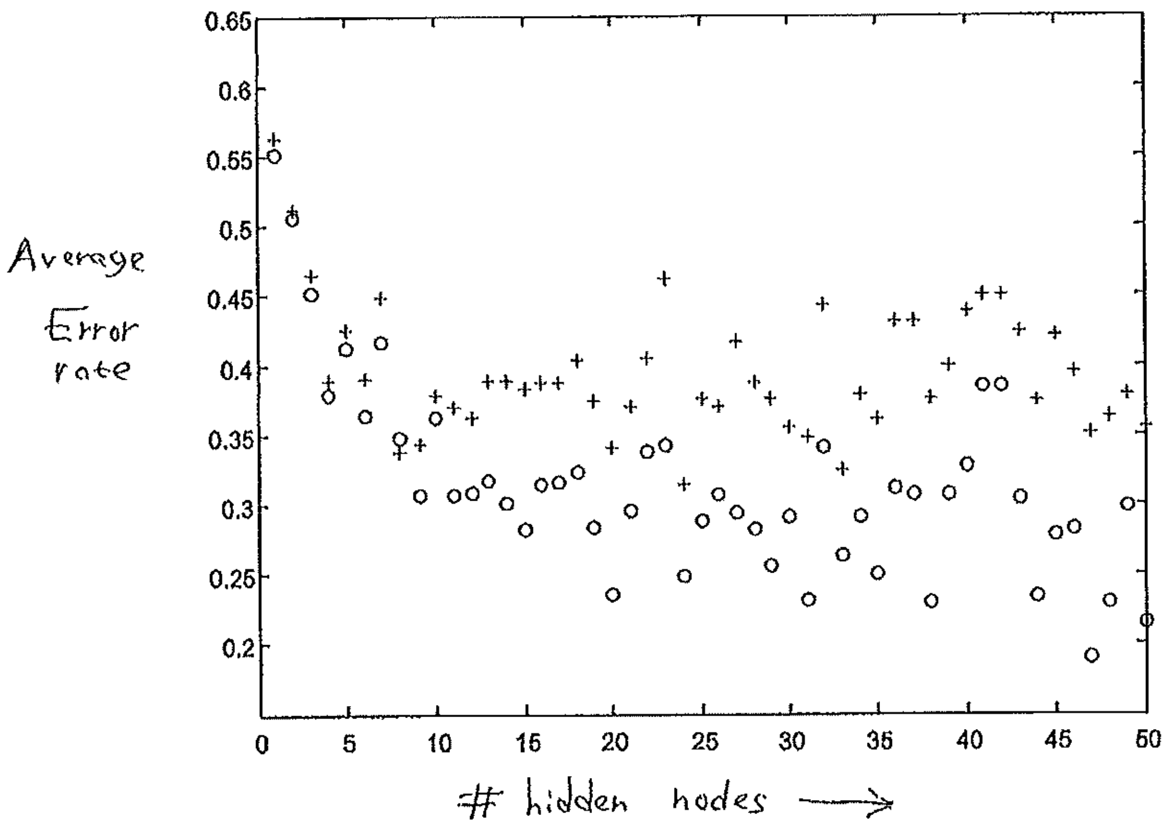
```
% XOR input for x1 and x2
input = [0 0; 0 1; 1 0; 1 1];
% Desired output of XOR
output = [0;1;1;0];
% Initialize the bias
bias = [-1 -1 -1];
% Learning coefficient
coeff = 0.7;
% Number of learning iterations
iterations = 10000;
% Calculate weights randomly using seed.
rand('state',sum(100*clock));
weights = -1 +2.*rand(3,3);

for i = 1:iterations
  out = zeros(4,1);
  numIn = length (input(:,1));
  for j = 1:numIn
    % Hidden layer
    H1 = bias(1,1)*weights(1,1) + input(j,1)*weights(1,2) + input(j,2)*weights(1,3);

    % Send data through sigmoid function 1/1+e^-x
    % Note that sigma is a different m file
    % that I created to run this operation
    x2(1) = 1/(1 + exp(-H1));
    H2 = bias(1,2)*weights(2,1) + input(j,1)*weights(2,2) + input(j,2)*weights(2,3);
    x2(2) = 1/(1+exp(-H2));

    % Output layer
    x3_1 = bias(1,3)*weights(3,1) + + x2(1)*weights(3,2) + x2(2)*weights(3,3);
    out(j) = 1/(1+exp(-x3_1));

    % Adjust delta values of weights
    % For output layer:
    % delta(wi) = xi*delta,
    % delta = (1-actual output)*(desired output - actual output)
    delta3_1 = out(j)*(1-out(j))*(output(j)-out(j));
     % Propagate the delta backwards into hidden layers
    delta2_1 = x2(1)*(1-x2(1))*weights(3,2)*delta3_1;
    delta2_2 = x2(2)*(1-x2(2))*weights(3,3)*delta3_1;

    % Add weight changes to original weights
    % And use the new weights to repeat process.
    % delta weight = coeff*x*delta
    for k = 1:3
      if k == 1 % Bias cases
        weights(1,k) = weights(1,k) + coeff*bias(1,1)*delta2_1;
        weights(2,k) = weights(2,k) + coeff*bias(1,2)*delta2_2;
        weights(3,k) = weights(3,k) + coeff*bias(1,3)*delta3_1;
      else % When k=2 or 3 input cases to neurons
        weights(1,k) = weights(1,k) + coeff*input(j,k-1)*delta2_1;
        weights(2,k) = weights(2,k) + coeff*input(j,k-1)*delta2_2;
        weights(3,k) = weights(3,k) + coeff*x2(k-1)*delta3_1;
      end
    end
  end
 mse(i) = (out - output)'*(out - output);
 end
 plot(mse)
```

Global minimum
run




Local minimum
run
MSE
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
iterations

Glass + = ~~training~~ test error rate
o = training error rate



Note: 1) training error rate lower than test error, generally.

2) Variance in performance grows with the number of hidden nodes

3) Best test error occurs with ~ 24 hidden units.

4) For this 6-class problem, test error rate is certainly much better than random guessing.