STAT 161/261: Homework 4 Solutions
Due Saturday, June 4, noon

1. (20 points) See MATLAB published file `imageseg.m`.
2. (20 points) See MATLAB published file `nonpar.m`.
3. (Problem removed.)
4. The data $\{x_i\}$ are scalars, so we can picture them on a one-dimensional number line. The given form of classifier,
$$\widehat{r}_i = \operatorname{sign}(x_i + w_0)\,, \tag{1}$$
takes $-w_0$ as a separating point, assigning all inputs in $(-\infty, -w_0)$ to the -1 class and all inputs in $(-w_0, \infty)$ to the $+1$ class.

   (a) (5 points) The data are linearly separable when there exists w_0 such that $x_i < -w_0$ for all i such that $r_i = -1$ and $x_i > -w_0$ for all i such that $r_i = +1$. In simpler terms, all the -1 data are to the left of all the $+1$ data.

   (b) (8 points) First, let us understand an arbitrary single term of the loss function
$$L(w_0) = \sum_i \max\{-r_i(x_i + w_0), 0\}\,. \tag{2}$$
Suppose x_i is in the $+1$ class. If $x_i + w_0 > 0$, then $\widehat{r}_i = +1$, so $-r_i(x_i + w_0)$ is negative. This makes the result of the max operation equal zero. A similar argument applies when x_i is in the -1 class and $x_i + w_0 < 0$. Putting these together, we see that a correct classification makes zero contribution to the loss function (2).

   Now again suppose x_i is in the $+1$ class, but now with $x_i + w_0 < 0$. Then $\widehat{r}_i = -1$ and $-r_i(x_i + w_0)$ is positive. This makes the result of the max operation equal $-r_i(x_i + w_0)$ (a positive quantity). A similar argument applies when x_i is in the -1 class and $x_i + w_0 > 0$. Putting these together, we see that an incorrect classification makes a contribution to the loss function equal to the distance to the discriminant $-w_0$.

   Intuitively, it must be best to put the discriminant where it approximately separates the data. We can use the description of the loss function above to be precise. Suppose N_+ is the number of errors on samples in the $+1$ class and N_- is the number of errors on samples in the -1 class. Moving the discriminant by ϵ to the right increases $L(w_0)$ by $\epsilon(N_+ - N_-)$, so by this differential argument, the discriminant is in an optimal position only when $N_+ = N_-$, i.e., the numbers of the types of errors are equal.

To more formally use calculus, one can differentiate $L(w_0)$ with respect to w_0 to come to the same conclusion. Specifically,

$$\frac{d}{dw_0}L(w_0) = N_- - N_+$$

because the ith term contributes -1 for an error on a $+1$ example, $+1$ for an error on a -1 example, and 0 for a correctly classified example.

5. (12 points) As suggested in the problem, assume $x_1 < x_2 < \cdots < x_N$. At the bottom layer, we just want to produce N binary outputs

$$b_i = \begin{cases} 1, & x \geq x_i; \\ 0, & \text{otherwise.} \end{cases}$$

This is achieved with a block with constant 1 input weighted by $-x_i$ and input x weighted by 1 (so that the linear combination $-x_i + x$ is computed), and a nonlinearity that outputs 1 if and only if its input is nonnegative.

Now the N binary values can be weighted so that a piecewise-constant approximation is formed. For the second layer, let the weight for the constant input be $w_0 = 0$, the weight for b_1 be $w_1 = y_1$, and the weight for b_i be $w_i = y_i - y_{i-1}$ for $i \geq 2$. Now just by evaluating $z = \mathbf{w}^T\mathbf{b}$ by cancelling terms in the telescoping sum,

$$z = \begin{cases} 0, & x \in (-\infty, x_1); \\ y_i, & x \in [x_i, x_{i+1}); \\ y_N, & x \in [x_N, \infty). \end{cases}$$

The second layer has no nonlinearity. The system has N hidden units.

6. (15 points) We will derive the backpropagation equations for the regression problem, but the equations for the classification problem are very similar. The neural network with H hidden layers can be described as follows: Suppose we are given a vector of predictors $\mathbf{x}$ and we want to predict a vector $\mathbf{r}$ of response variables. The NN generates the predicted response vector $\mathbf{y}$ by the recursive equations

$$z_{0,j} = x_j, \quad v_{h,i} = \sum_j W_{h,ij} z_{h,j}, \quad z_{h+1,i} = \sigma(v_{h,i}), \quad y_i = v_{H,i},$$

where $h = 0, \ldots, H$ is the layer index, $z_{h,i}$ and $v_{h,i}$ are hidden units in the layer, $W_{h,ij}$ are weights in the layer and $\sigma(v) = 1/(1+e^{-v})$ is a sigmoidal function. Note that the input $\mathbf{x}$ is used as the input of the first layer and the predicted response $\mathbf{y}$ is taken from the output of the final layer.

The unknown parameters are the weights $\mathbf{W} = \{W_{h,ij}\}$, which is the set of all weights over all layers. To train the network we are given data $(\mathbf{x}^t, \mathbf{r}^t)$, $t = 1, \ldots, N$. Given any training sample t and candidate set of weights we generate a set of outputs

$$z^t_{0,j} = x^t_j, \quad v^t_{h,i} = \sum_j W_{h,ij} z^t_{h,j}, \quad z^t_{h+1,i} = \sigma(v^t_{h,i}), \quad y^t_i = v^t_{H,i}. \tag{3}$$

For regression problems we want to select the weights $\mathbf{W}$ to minimize a loss function of the form

$$E(\mathbf{W}) := \frac{1}{2}\sum_{t=1}^{N}\sum_{i}(y_i^t - r_i^t)^2. \tag{4}$$

We need to compute the partial derivatives of this energy function. The trick to derive backpropagation is to first consider the partial derivatives,

$$\lambda_{h,j}^t = \frac{E}{\partial v_{h,j}^t},$$

where the derivative is taken holding all the weights from layer $h+1$ onwards constant. We show that we can compute the values $\lambda_{h,j}^t$. We begin with layer $h = H$. Since $y_i^t = v_{H,i}^t$, we have

$$\lambda_{H,j}^t = \frac{E}{\partial v_{H,j}^t} = \frac{E}{\partial y_j^t} = y_j^t - r_j^t.$$

The values for the other layers can be computed recursively:

$$\begin{aligned}
\lambda_{h,j}^t &\overset{(a)}{=} \frac{\partial E}{\partial v_{h,j}^t} \overset{(b)}{=} \sum_i \frac{\partial E}{\partial v_{h+1,i}^t}\frac{\partial v_{h+1,i}^t}{\partial v_{h,j}^t} \\
&\overset{(c)}{=} \sum_i \lambda_{h+1,i}^t W_{h+1,ij}\frac{\partial z_{h+1,j}^t}{\partial v_{h,j}^t} \overset{(d)}{=} \sum_i \lambda_{h+1,i}^t W_{h+1,ij}\sigma'(v_{h,i}^t) \\
&\overset{(e)}{=} \sum_i \lambda_{h+1,i}^t W_{h+1,ij} z_{h+1,i}^t(1 - z_{h+1,i}^t),
\end{aligned}$$

where (a) is the definition of $\lambda_{h,j}^t$; (b) is the chain rule; in (c) we used the definition of $\lambda_{h+1,j}^t$ and the fact that

$$v_{h+1,i}^t = \sum_j W_{h+1,ij} z_{h+1,j}^t;$$

and in (d) we used the fact that $z_{h+1,j}^t = \sigma(v_{h,j}^t)$; and in (e) we used the relation that if $z = \sigma(v)$, then

$$\sigma'(v) = \frac{e^{-v}}{(1+e^{-v})^2} = z\frac{e^{-v}}{1+e^{-v}} = z(1-z).$$

The derivatives of the energy function (4) with respect to the weights can then be computed as

$$\frac{\partial E}{\partial W_{h,ij}} = \sum_t \frac{\partial E}{\partial v_{h,j}^t}\frac{v_{h,j}^t}{\partial W_{h,ij}} \sum_t \lambda_{h,j}^t z_{h,i}^t.$$

Summarizing we get the following steps for backpropagation: Start with data $(\mathbf{x}^t, \mathbf{r}^t)$ and some estimates for the weights $\mathbf{W}$.

(a) Run the *forward* equations (3) to obtain the predicted response r_i^t for all the training samples t

(b) Initialize $\lambda_{H,j}^t = y_j^t - r_j^t$ for all samples t and outputs j.

(c) Recursively compute the *backward* equations

$$\lambda_{h,j}^t = \sum_i \lambda_{h+1,i}^t W_{h+1,ij} z_{h+1,i}^t (1 - z_{h+1,i}^t).$$

(d) Compute the gradient

$$\frac{\partial E}{\partial W_{h,ij}} = \sum_t \lambda_{h,j}^t z_{h,i}^t.$$

(e) Update the parameters

$$W_{h,ij} = W_{h,ij} - \eta \frac{\partial E}{\partial W_{h,ij}},$$

for some stepsize η.

7. (a) (5 points) For matching a binary-valued function, a single-layer perceptron cannot do anything but implement a planar discriminant (separation boundary). Here, that is not good enough because the $(x_1, x_2) = (1, 1)$ point has to be separated from $(1, 0)$ and $(1, 2)$ but lies exactly on the line connecting them.

(b) (8 points) We can draw inspiration from the XOR example in Section 11.6 (discussed in lecture on May 25) to expect to find some solution with two layers. There are many possibly solutions. A first layer can separate the space $\mathbb{R}^2$ with lines such as the thick and thin lines in Figure 1. As in Figure 11.7 of Alpaydin, the positive directions for hidden layer variables z_1 and z_2 are marked. We then want to compute the AND of z_1 and z_2 to match the function r.

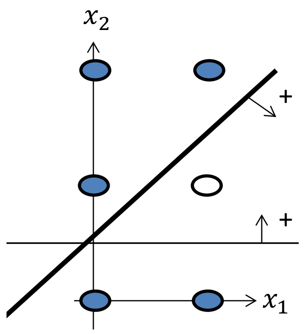


Figure 1: Separation of input space created with two-layer perceptron.

Now to be more explicit, use $s(\cdot)$ to denote the threshold function as in (11.3) of Alpaydin. Then

$$\begin{aligned} z_1 &= s(0.5 + x_1 - x_2), \\ z_2 &= s(-0.5 + x_2), \\ y &= s(-1.5 + z_1 + z_2) \end{aligned}$$

gives the desired output.

8. (a) (5 points) We are merely being asked to justify equation (11.16) by computing the derivative of the nonlinearity. We are given

$$z_h = \frac{1}{1+\exp[-\mathbf{w}_h^T\mathbf{x}]}.$$

Thus,

$$\begin{aligned}
\frac{\partial z_h}{\partial w_{hj}} &= \frac{-1}{(1+\exp[-\mathbf{w}_h^T\mathbf{x}])^2} \cdot \frac{\partial}{\partial w_{hj}} \left(1+\exp[-\mathbf{w}_h^T\mathbf{x}]\right) \\
&= \frac{-1}{(1+\exp[-\mathbf{w}_h^T\mathbf{x}])^2} \cdot \exp[-\mathbf{w}_h^T\mathbf{x}] \cdot \frac{\partial}{\partial w_{hj}} \left(-\mathbf{w}_h^T\mathbf{x}\right) \\
&= \frac{-1}{(1+\exp[-\mathbf{w}_h^T\mathbf{x}])^2} \cdot \exp[-\mathbf{w}_h^T\mathbf{x}] \cdot (-x_j) \\
&= \frac{1}{1+\exp[-\mathbf{w}_h^T\mathbf{x}]} \cdot \frac{\exp[-\mathbf{w}_h^T\mathbf{x}]}{1+\exp[-\mathbf{w}_h^T\mathbf{x}]} \cdot x_j \\
&= z_h(1-z_h)x_j,
\end{aligned}$$

as desired.

(b) (12 points) See `train_and_test.m`, which uses `trainAutoencoder.m`, which uses `sigmoid.m`.

(c) (5 points) (This is a subset of the following part.)

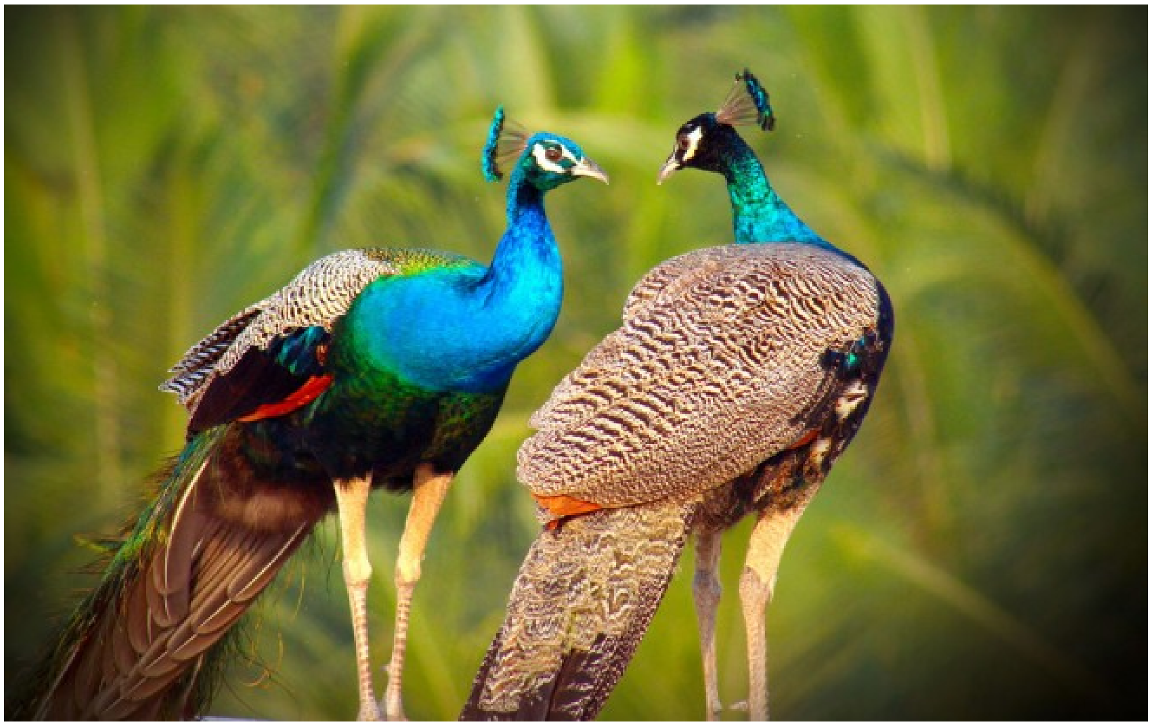
(d) (8 points) The script `experiment.m` makes the computations and produces a plot. This shows the MSE reducing as H is increased from 1 to 3 and then staying approximately constant. As H is increased, the set of approximation points produced by the autoencoder can take a more complicated form. $H = 1$ and $H = 2$ are clearly inadequate to approximate the distribution, but starting with $H = 3$ the approximation is reasonable.

imageseg.m: Image segmentation using k-means

Contents

Read the image

```
%X = imread('images\119082.jpg');
%X = imread('images\43074.jpg');
X = imread('images\birds2.jpg');
imshow(X);
```



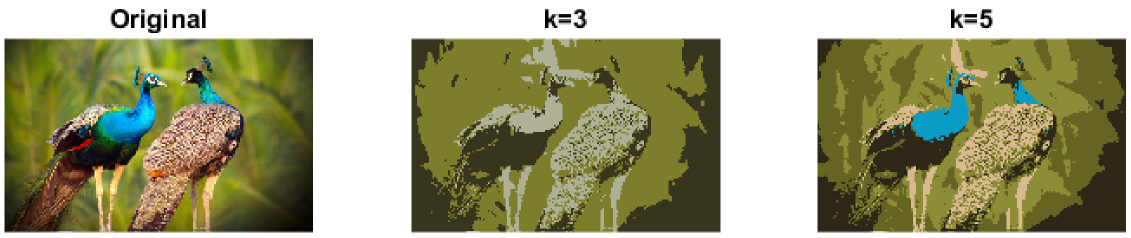
Run k-means

```
% Get dimensions
nrow = size(X,1);
ncol = size(X,2);
nrgb = size(X,3);
nctest = [3 5];      % number of clusters to test
ntest = length(nctest);
nrep = 3;    % number of repetitions to avoid local minima

% Reshape the image to a matrix and run PCA
X1 = reshape(double(X),nrow*ncol,nrgb);

% Run k-means for each different number of clusters
Ic = zeros(nrow*ncol,ntest);
mu = zeros(max(nctest),nrgb,ntest);
for it = 1:ntest
```

```
    nc = nctest(it);
    [Ic1, mu1] = kmeans(X1,nc,'distance','sqEuclidean', 'Replicates',nrep);

    % Save indices and cluster centers
    Ic(:,it) = Ic1;
    mu(1:nc,:,it) = mu1;

end
```

Plot the quantized image

```
% Plot the original image
subplot(1,ntest+1,1);
imshow(X);
title('Original');

% Plot the quantized images
for it = 1:ntest

    % Replace the color in each pixel by its cluster center
    nc = nctest(it);
    Xc = X1;
    for ic = 1:nc
        nj = sum(Ic(:,it)==ic);
        Xc(Ic(:,it)==ic,:) = repmat(mu(ic,:,it),nj,1);
    end

    % Reshape to a square and convert to uint8
    Xc = uint8(round(reshape(Xc,nrow,ncol,nrgb)));

    % Plot the image
    subplot(1,ntest+1,it+1);
    imshow(Xc);
    str = sprintf('k=%d', nc);
    title(str);

end
```

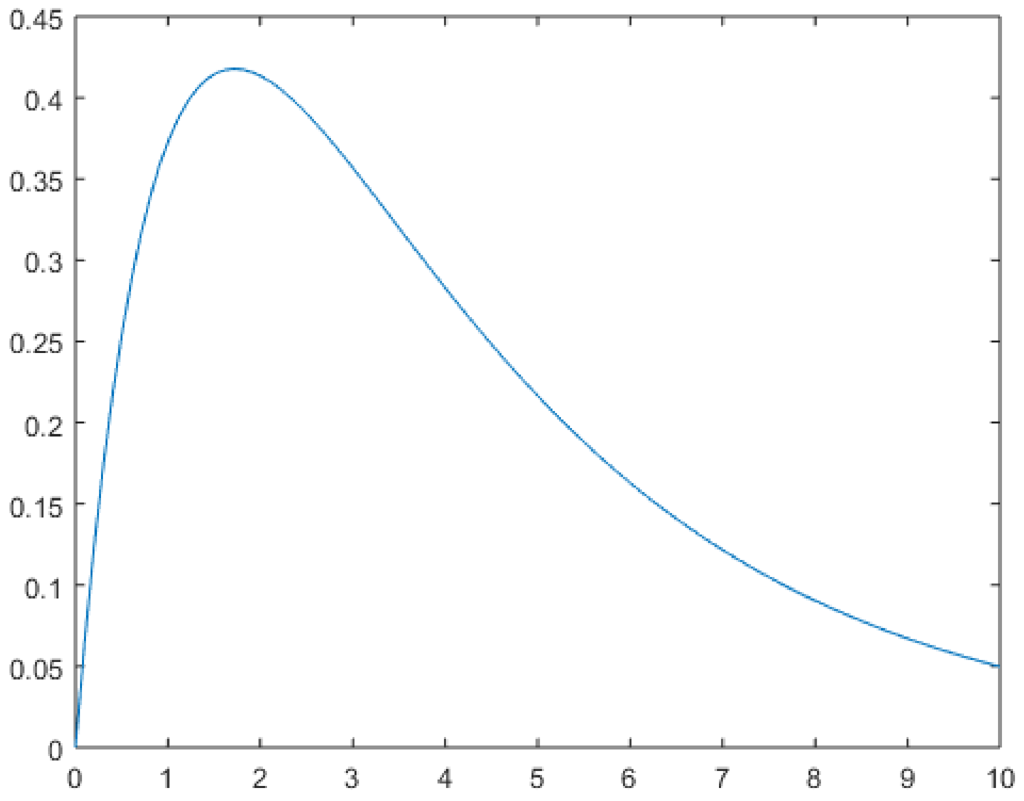

Original
k=3
k=5

nonpar.m: Non-parametric fit

Contents

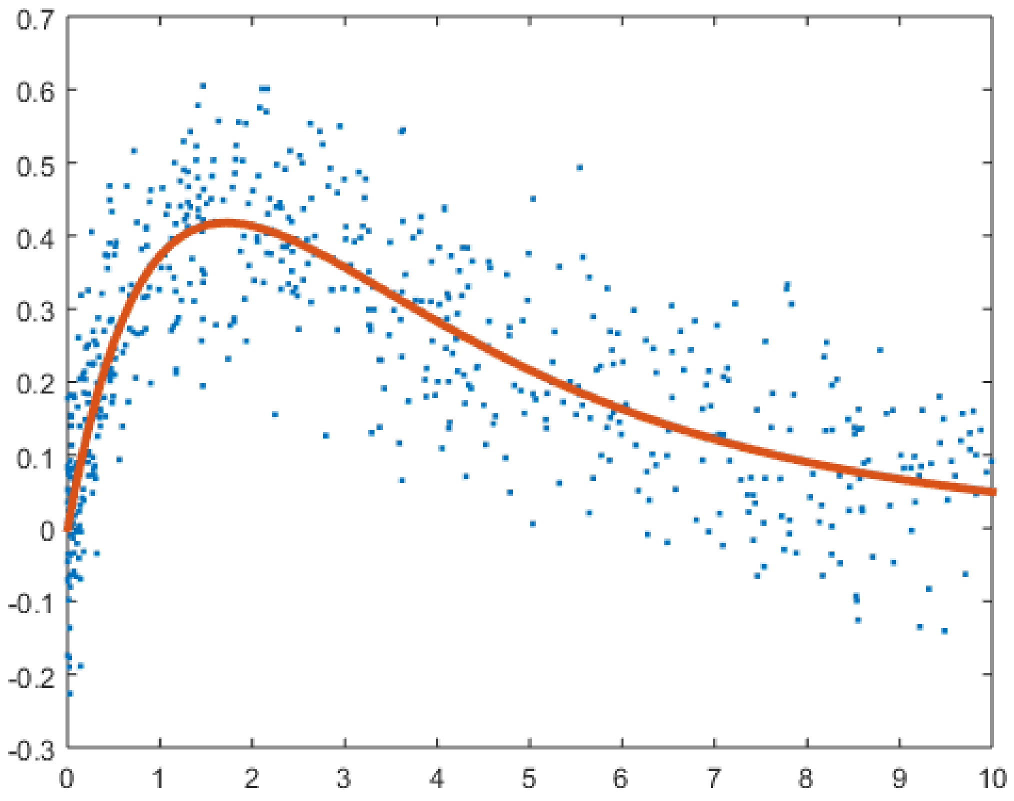
Plot the true function

```
% Parameters
xmax = 10;      % max value of x
nx = 100;       % number of points

% Define a function handle for the function
f = @(x) (1-exp(-0.7*x)).*exp(-0.3*x);

% Plot the function
x0 = linspace(0,xmax,nx);
y0 = f(x0);
plot(x0,y0);
```



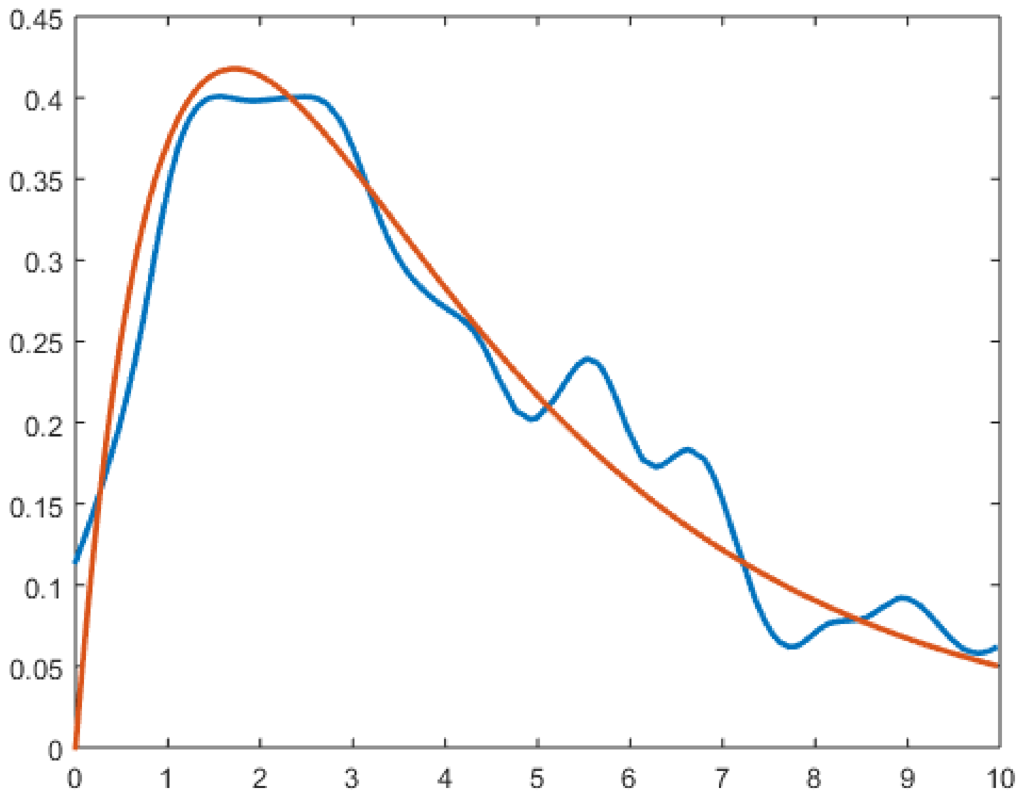
Generate data

```
% Number of training and test points
ntrain = 300;
ntest = 300;
nsamp = ntrain+ntest;
sigw = 0.1;

% Random samples and noisy measurements
x = xmax*rand(nsamp,1).^2;
r = f(x) + sigw*randn(nsamp,1);

% Scatter plot
plot(x,r,'.', x0,y0,'-','Linewidth',3);
```



Fixed h kernel regression

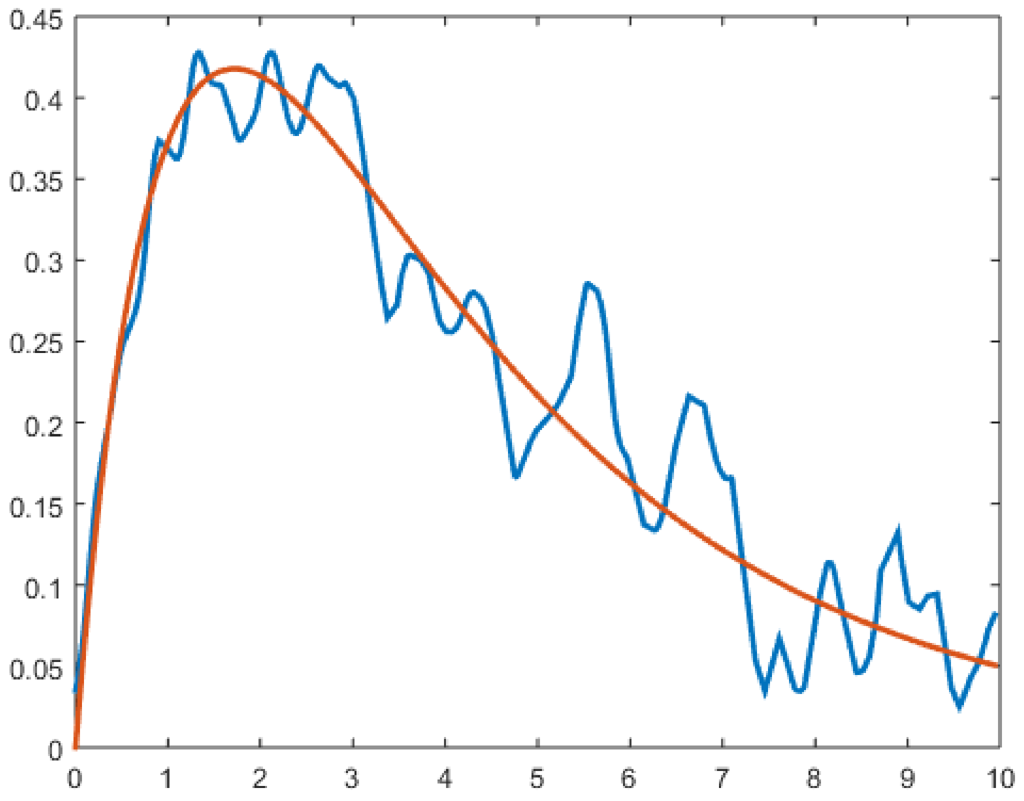
```
% Training and test data
% These are sorted to make the plotting easier
[xtr,I] = sort(x(1:ntrain));
rtr = r(I);
[xts,I] = sort(x(ntrain+1:nsamp));
rts = r(ntrain+I);

% Compute distance to each point and the reponse matrix
% D(i,j) = distance from test sample i to training j
% R(i,j) = test response j
D = abs(repmat(xts,1,ntrain)-repmat(xtr',ntest,1));
R = repmat(rtr',ntest,1);

% Kernel window
h = 0.3;
W = exp(-D.^2/(2*h^2));

% Regression output and plot
rhat = sum(W.*R,2)./sum(W,2);
plot(xts,rhat,'-', xts, f(xts),'-', 'Linewidth',2);
```



Select optimal value by cross validation

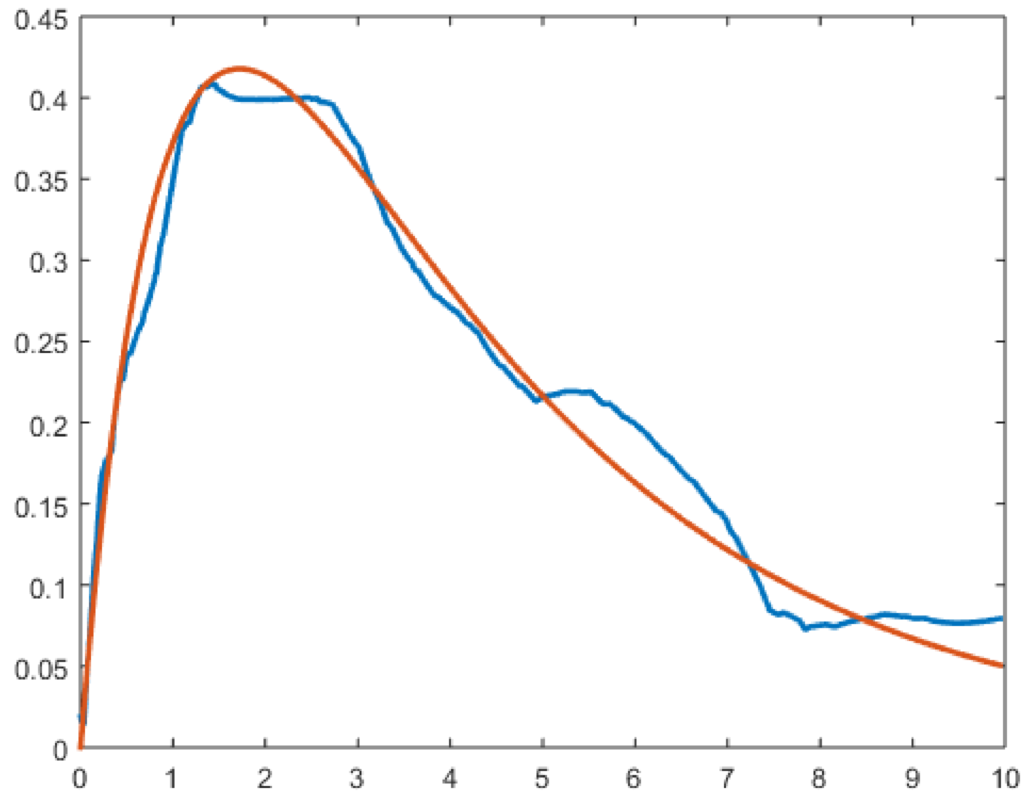
```
% Values of h to test
htest = linspace(0.1,2,100)';
ntest = length(htest);

% Compute RSS for each h
rss = zeros(ntest,1);
for i=1:ntest
    h=htest(i);
    W = exp(-D.^2/(2*h^2));
    rhat = sum(W.*R,2)./sum(W,2);
    rss(i) = mean((rhat-rts).^2);
end

% Find minimum h
[rssmin,im] = min(rss);
h = htest(im);

% Plot the corresponding fit
W = exp(-D.^2/(2*h^2));
rhat = sum(W.*R,2)./sum(W,2);
plot(xts,rhat,'-', xts, f(xts),'-', 'Linewidth',2);
fprintf(1,'Fixed h:  RSS=%12.4e hopt=%12.4e\n', rssmin,h);
```

```
Fixed h:  RSS=  1.0465e-02 hopt=  1.0000e-01
```



K-NN search

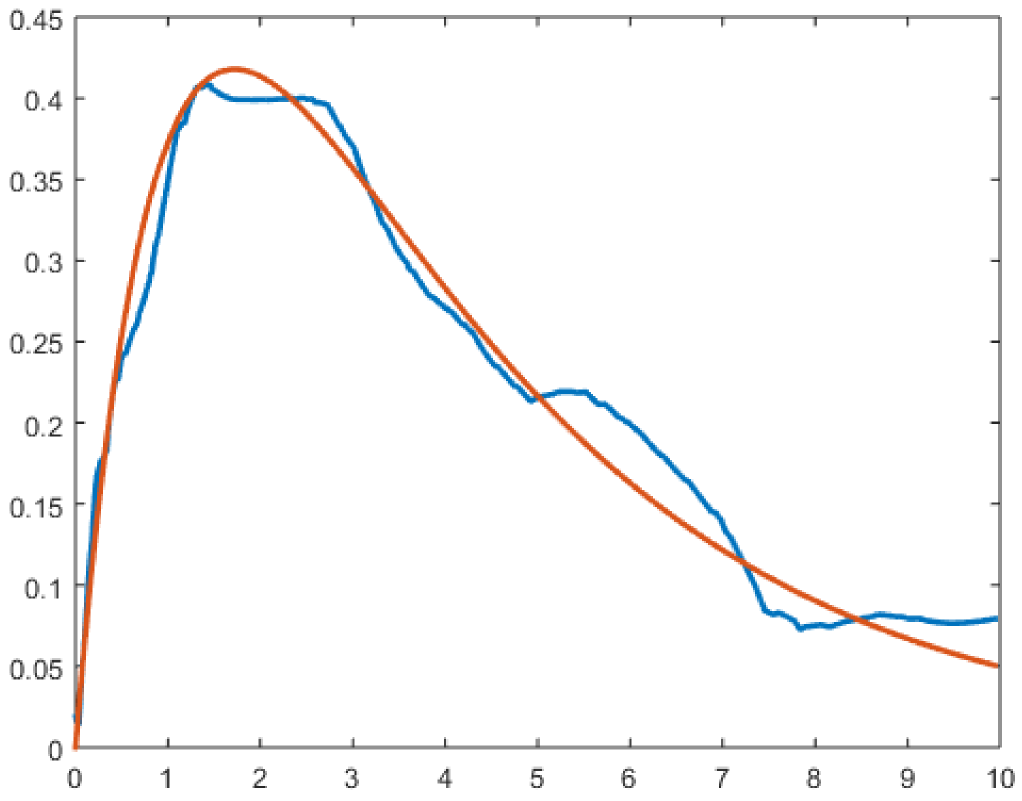
```
% For each test point, find nearest neighbor and distance
k = 20;
[idx, dist] = knnsearch(xtr,xts,'K',k);

% Compute estimate with adaptive step size
h = repmat(dist(:,k),1,ntrain);
W = exp(-D.^2./(2*h.^2));
rhat = sum(W.*R,2)./sum(W,2);
plot(xts,rhat,'-', xts, f(xts),'-', 'Linewidth',2);
```



Find optimal k via cross-validation

```
% K values to test
ktest = (5:30);
ntest = length(ktest);

% Loop over k values and find RSS for each
rss = zeros(ntest,1);
for i=1:ntest
    k = ktest(i);
    [idx, dist] = knnsearch(xtr,xts,'K',k);
    h = repmat(dist(:,k),1,ntrain);
    W = exp(-D.^2./(2*h.^2));
    rhat = sum(W.*R,2)./sum(W,2);
    rss(i) = mean((rhat-rts).^2);
end

% Select lowest RSS
[rssmin,im] = min(rss);
k = ktest(im);

% Plot the resulting fit
[idx, dist] = knnsearch(xtr,xts,'K',k);
h = repmat(dist(:,k),1,ntrain);
W = exp(-D.^2./(2*h.^2));
rhat = sum(W.*R,2)./sum(W,2);
plot(xts,rhat,'-', xts, f(xts),'-', 'Linewidth',2);
fprintf(1,'kNN:      RSS=%12.4e kopt=%d\n', rssmin, k);
```

```
kNN:       RSS=  9.3347e-03 kopt=20
```



Published with MATLAB® R2015a

```
function d = train_and_test( H, eta, Ntrain, Ntest, W, V )
%d = train_and_test( H, Ntrain, Ntest, W, V )
%
%   Train on Ntrain samples, then test on Ntest samples.
%   Return mean-squared error on the test set.
%   May provide learning rate eta and initial values for W and V.

d = 3;
if ~exist('H','var'),
    H = 7;
end
if ~exist('eta','var'),
    eta = 0.005;
end
if ~exist('Ntrain','var'),
    Ntrain = 10000;
end
if ~exist('Ntest','var'),
    Ntest  = 1000;
end
if ~exist('W','var'),
    W = randn( H, d+1 );
end
if ~exist('V','var'),
    V = randn( d, H+1 );
end

% Train
for i = 1:Ntrain,
    x = mlp_test_data;
    [W,V,y] = trainAutoencoder( W, V, x, eta, H );
end

% Test
figure;
hold on;
for i = 1:Ntest,
    x = mlp_test_data;
    [~,~,y] = trainAutoencoder( W, V, x, eta, H );
    plot3( x(1), x(2), x(3), 'k.' );
    plot3( y(1), y(2), y(3), 'r.' );
    d(i) = sum( (x-y).^2 );
end
d = mean(d);
```

```
function [Wtplus1,Vtplus1,y] = trainAutoencoder( Wt, Vt, x, eta, H )
%[Wtplus1,Vtplus1,y] = trainAutoencoder( Wt, Vt, x, eta, H )
%
%  Implementation of a multilayer perceptron with one H-unit hidden layer.

d = length(x);

% Compute first forward step
X = [1; x];
for h = 1:H,
    z(h,1) = sigmoid( Wt(h,:) * X );
end
Z = [1; z];

% Compute second forward step
for j = 1:d,
    y(j,1) = Vt(j,:) * Z;
end

% Second-layer updates
e = x - y;
for j = 1:d,
    for h = 0:H,
        Delta_v(j,h+1) = eta * e(j) * Z(h+1);
    end
end
Vtplus1 = Vt + Delta_v;

% First-layer updates
for h = 1:H,
    for j = 0:d,
        Delta_w(h,j+1) = eta * e' * Vt(:,h) * z(h)*(1-z(h)) * X(j+1);
    end
end
Wtplus1 = Wt + Delta_w;
```

```
function z = sigmoid( x )

z = 1 / (1 + exp(-x));
```

```
% experiment.m

Hvalues = [1:7, 1:7, 1:7]; % three independent trials at each H value
for i = 1:length(Hvalues),
    d(i) = train_and_test( Hvalues(i) );
end
figure;
plot( Hvalues, d, 'o' );
xlabel('H');
ylabel('Mean squared error');
```