Lecture 21

Polyfit function in Matlab (cont'd)

```
N = 4
% Generate 4 data points for training
rng(1);
x = 10*rand(1, N);

Z = 1 + 2*x + 3*x.^2;  % Target values

% Formulate the input data matrix
Fx = zeros(3,N);
for i = 1:N
   Fx(:,i) = [1, x(i), x(i)^2];
end

W = Z / Fx
W2 =  Z * pinv(Fx)
```

```
W =

   1.0000    2.0000    3.0000


W2 =

   1.0000    2.0000    3.0000
```

```
[p,s] = polyfit(x, Z, 2);   % notice the reversed order

>> p

p =

   3.0000    2.0000    1.0000
```

Structure Returned by polyfit ( )

```
[p,S] = polyfit(x,y,n) also returns a structure S
that can be used to obtain error estimates.

S is a structure containing three elements:
(1) The triangular factor from a QR decomposition
    of the Vandermonde matrix,
(2) The degrees of freedom and,
(3) The norm of the residuals.

    S.R = R;
    S.df = max(0,length(y) - (n+1));
    r = y - V*p;
    S.normr = norm(r);
```

```
>> s.R

ans =

 -55.4817   -8.5417   -1.4134
      0    2.3359    0.9953
      0         0   -1.0058

>> s.normr

ans =

  6.6613e-16
```

```
% Now with noise added
N = 4
rng(1);
x = 10*rand(1, N);

rng(1);
Z = 1 + 2*x + 3*x.^2 + randn(1, N);

% Formulat the input data matrix
Fx = zeros(3,N);
for i = 1:N
   Fx(:,i) = [1, x(i), x(i)^2];
end

W = Z / Fx
```

```
W = Z / Fx

N =

     4


W =

    0.2300    1.6518    3.0862
```

```
% Mimic the Matlab implemenation
% The Vondermonde matrix
V = fliplr(Fx');    % Transpose followed by left to right flip
[Q,R] = qr(V,0);   % Economy-size QR decomposition
% Orthornormal columns
Q'*Q

p2 = mldivide(R, Q'*Z');
p2'
```

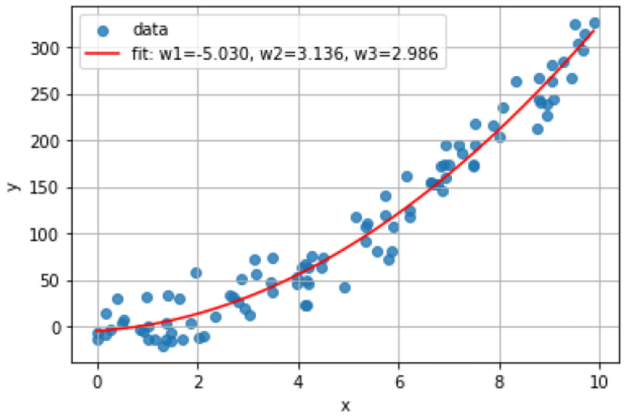
```
ans =

    3.0862    1.6518    0.2300
```

```
% Compared wit the structure returned by polyfit( )
[p,s] = polyfit(x, Z, 2);
p
s.R
s.normr
% Compare the norm of the residues

normr2 = norm(W*Fx - Z)
normr3 = norm(V*p2 - Z')
```

regression_demo.py




```
import numpy as np
infile = "regression_dataset.csv"
dataset = np.loadtxt(infile, delimiter=',')
xdata = dataset[:, 0]
ydata = dataset[:, 1]


from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2)
xdata = xdata[:, np.newaxis]

xdata_poly = poly.fit_transform(xdata)

reg = LinearRegression(fit_intercept=False).fit(xdata_poly, ydata)
# reg = LinearRegression().fit(xdata_poly, ydata)

reg.coef_
# reg.intercept_
```

Logistic Regression

- Here we adopt a generative approach, where we model the class-conditional densities, and class priors, then use these to compute posterior probabilities through Bayes' theorem (using the two-class problem as an example):

$$\begin{aligned} p(\mathcal{C}_1|\mathbf{x}) &= \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} \\ &= \frac{1}{1+\exp(-a)} = \sigma(a) \end{aligned}$$

where

$$a = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$

$\sigma(a)$ is the *logistic sigmoid* function defined by $\sigma(a) = \frac{1}{1+\exp(-a)}$

The logistic sigmoid function is sometimes called ***logistic*** function.