Lecture 23

Logistic Regression

- In a two-class classification problem, the posterior probability of class C_1 can be written as a logistic sigmoid acting on a linear function of the feature vector ϕ so that
  $p(C_1|\phi) = y(\phi) = \sigma(\mathbf{w}^{\mathrm{T}}\phi)$
- Note that this is a model for classification rather than regression.
- For an M-dimensional feature space Φ, this model has M adjustable parameters.
- By contrast, when we previously fitted Gaussian class conditional densities using maximum likelihood, we would have used $2M$ parameters for the means, and $\frac{M(M+1)}{2}$ parameters for the (shared) covariance matrix. Together with the class prior $p(C_1)$, this gives a total of $\frac{M(M+5)}{2} + 1$ parameters, which grows quadratically with M.
- For large values of M, there is a clear advantage in working with the logistic regression model **directly**.
- To determine the parameters of the logistic regression model, we can use
  - Maximum likelihood
  - Iterative reweighted least squares

Symmetric matrix

$$\begin{array}{rc} & \overbrace{\hphantom{x \quad x \quad \cdots \quad x}}^{M} \\ \begin{array}{r} M \rightarrow \\ M-1 \rightarrow \\ \vdots \\ 1 \rightarrow \end{array} & \left.\left[\begin{array}{cccc} x & x & \cdots & x \\ & x & & \vdots \\ & & \ddots & \vdots \\ & & & x \end{array}\right]\right\} M \end{array}$$

$$1 + 2 + 3 + \cdots + M-1 + M = \frac{M(M+1)}{2}$$

```python
"""
'logistic_demo.py'

Logistic Regression

"""

import numpy as np
infile = "logistic_regress.csv"
dataset = np.loadtxt(infile, delimiter=',')
X = dataset[:, 0:2]

y = dataset[:,2]  # labels

from sklearn.linear_model import LogisticRegression as LG

clf = LG().fit(X, y)
clf.intercept_
clf.coef_

clf.score(X,y)

y_pred = clf.predict(X)
num_errors = np.sum(y != y_pred)
num_errors/np.size(y)
```

```
clf.intercept_
Out[8]: array([-15.29402959])

clf.coef_
Out[9]: array([[1.05654227, 1.39895119]])

clf.score(X,y)
Out[10]: 0.985

num_errors/np.size(y)
Out[13]: 0.015
```

```
logistic_regression_demo.m
>> B

B =

   19.0582
   -1.2745
   -1.7747
```

Perceptron Training Algorithm

- Let $\alpha > 0$ denote a correction increment (also called the learning increment or the **learning rate**)
- Let the initial weight vector $\boldsymbol{w}(1)$ take arbitrary values. Then, repeat the following steps for $k = 2, 3, \ldots,$:

For an augmented pattern vector, $\mathbf{x}(k)$, at step k,

If $\mathbf{x}(k) \in c_1$ and $\boldsymbol{w}^T(k)\mathbf{x}(k) \leq 0$, let

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + \alpha\mathbf{x}(k)$$

If $\mathbf{x}(k) \in c_2$ and $\boldsymbol{w}^T(k)\mathbf{x}(k) \geq 0$, let

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) - \alpha\mathbf{x}(k)$$

Otherwise, let

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k)$$

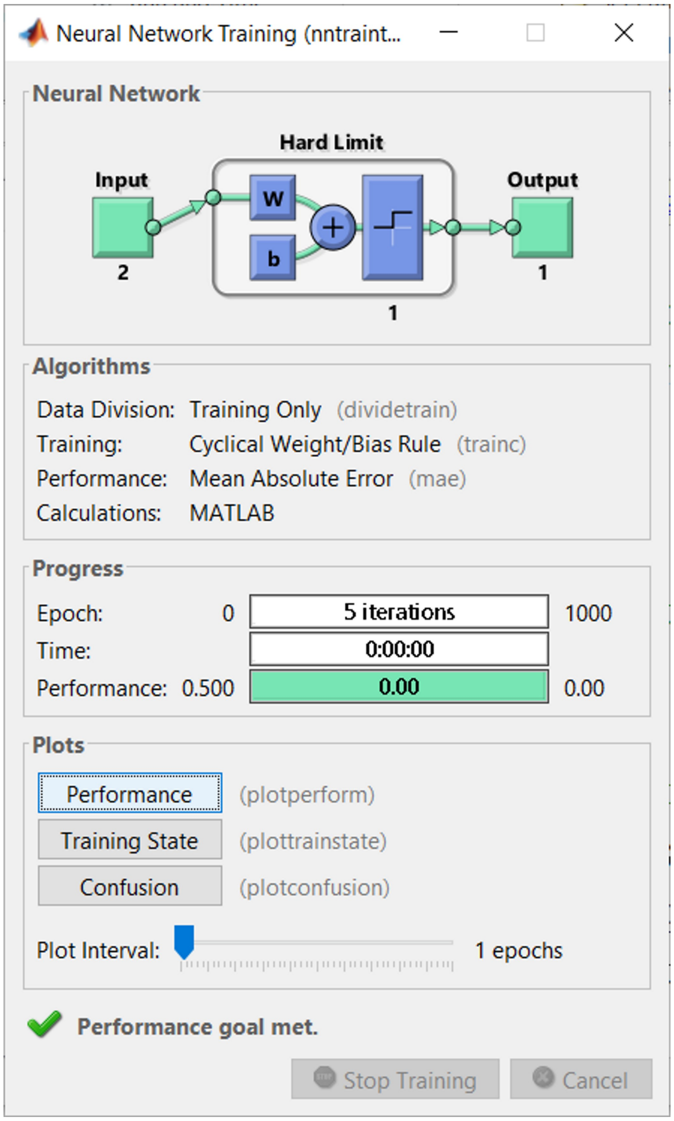
$$\boxed{\boldsymbol{w}^T\mathbf{x} = \begin{cases} > 0 & \text{if } \mathbf{x} \in c_1 \\ < 0 & \text{if } \mathbf{x} \in c_2 \end{cases}}$$

```
>> % Class 1: [3 3 1], Class 2: [1 1 1]

a = 1;  % learning rate

% Augmented input vector
x1 = [3 3 1];
x2 = [1 1 1];

w = [0 0 0];    % initial weight vector

for epoch = 1: 20
   w_prev = w;
   x = x1;
   y = dot(w, x);

   if (y <= 0)
      w = w + a*x;
   end

   x = x2;
   y = dot(w, x);

   if (y >= 0)
      w = w - a*x;
```

```
epoch

w

dot(w, x1)
dot(w, x2)

epoch =

    6


w =

    1     1    -3


ans =

    3


ans =
```

```
    if (y >= 0)
        w = w - a*x;
    end

    if (w == w_prev)
        break;
    end
end
```

```
ans =

    -1
```




```
% Use Matlab built-in function
% Do not use the augmented input vector
x1 = [3 3];
x2 = [1 1];

% The input matrix: (vert: features, horz: samples)
x = [x1' x2'];

% Target has to be 0/1 values for binary classification
target = [0 1];
method = perceptron;
net = train(method, x, target);

>> % View the weights for the connection from the first input to
the first layer
net.iw{1,1}
% View the bias values for the first layer
net.b{1}

ans =

   -1   -1


ans =

    3
```