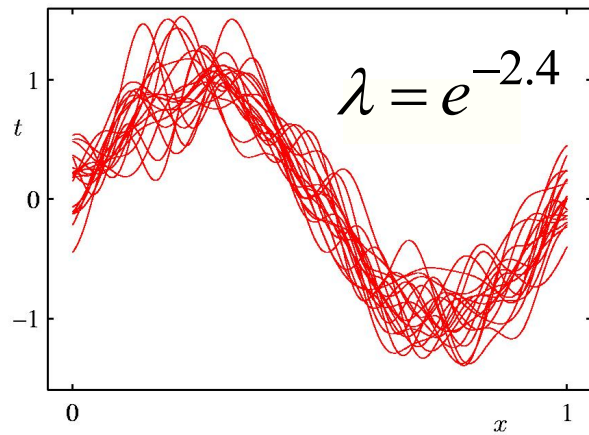
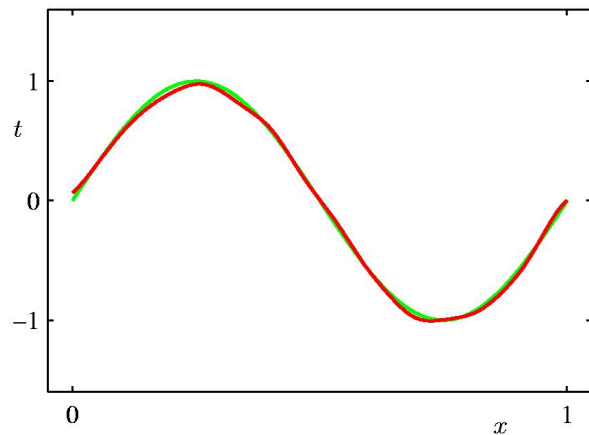
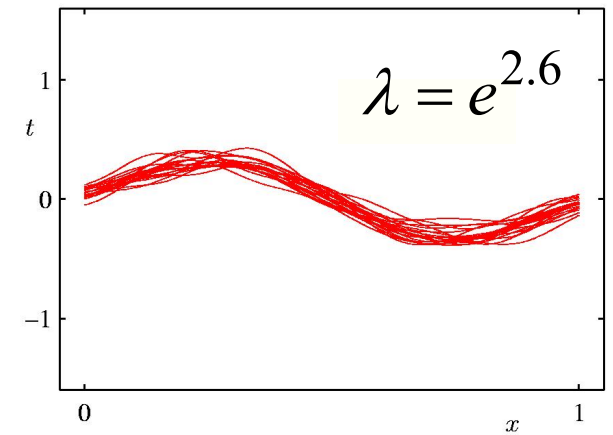
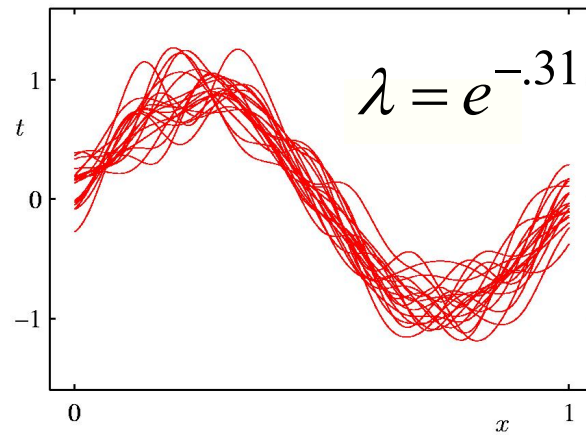


How the regularization parameter affects the bias and variance terms

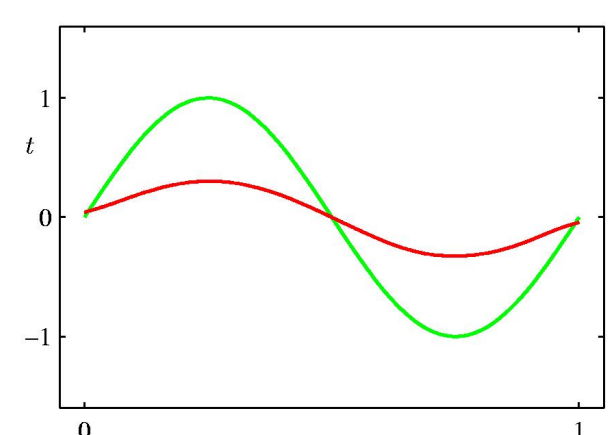
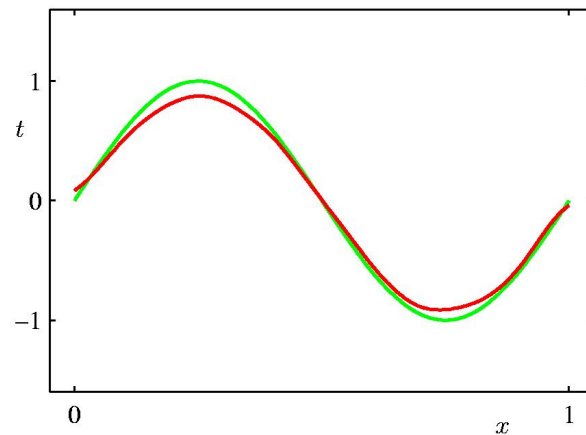
high variance



low variance



low bias

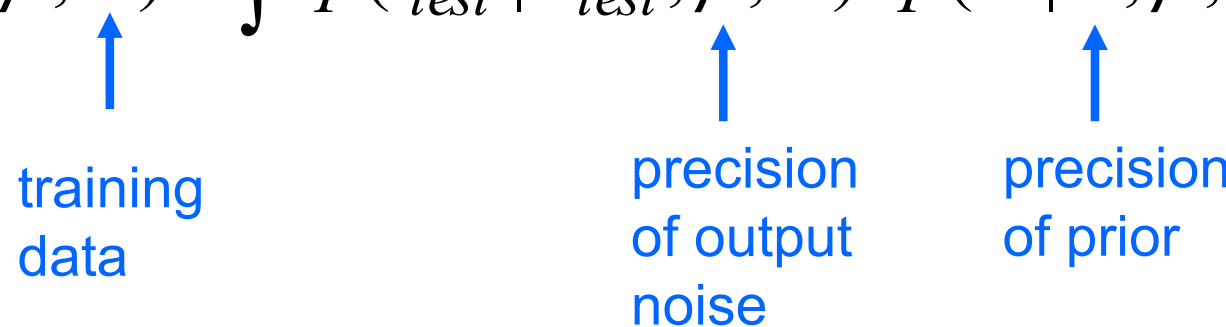


high bias

Using the posterior distribution

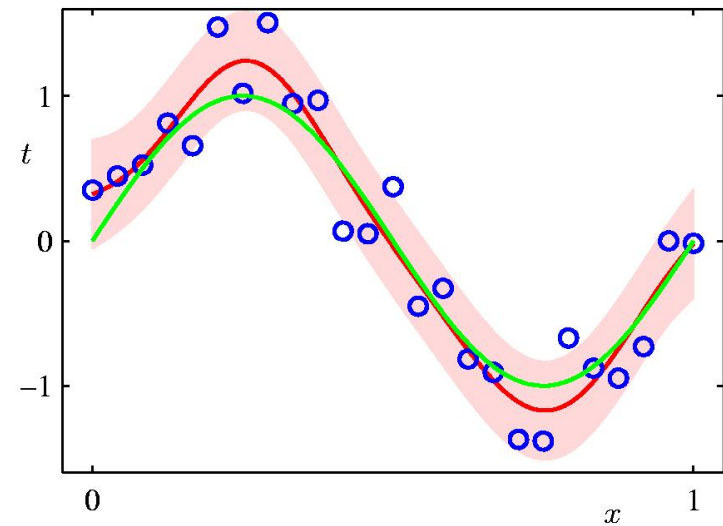
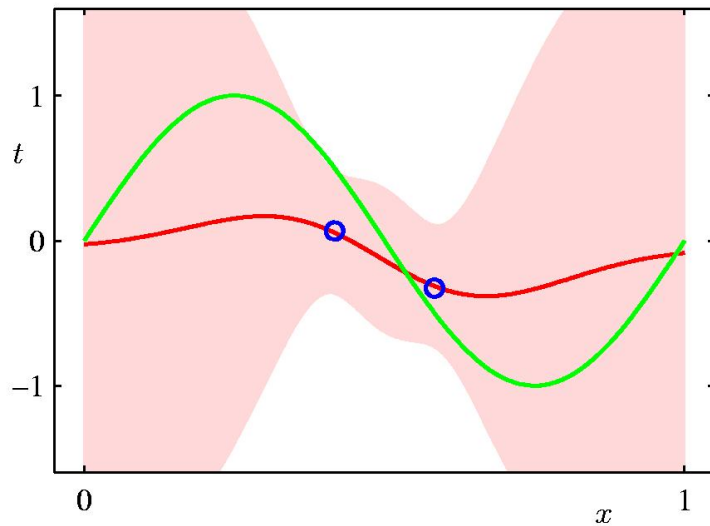
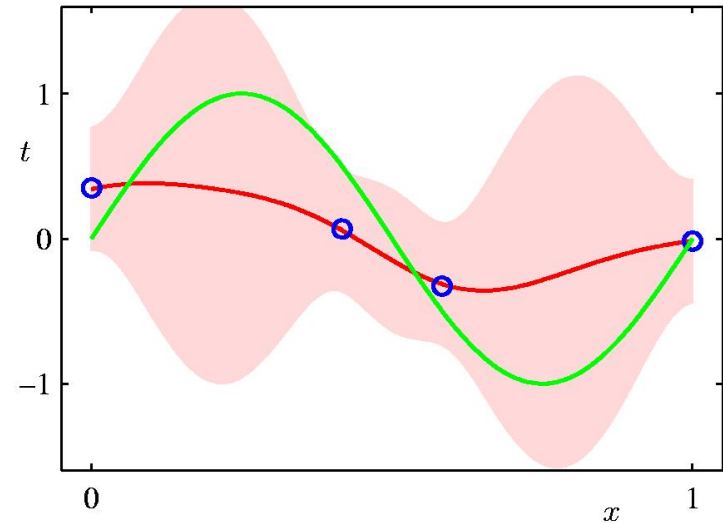
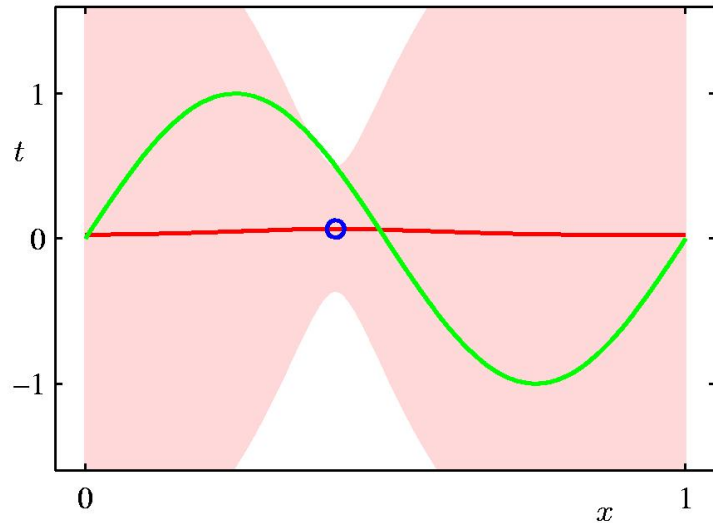
- If we can afford the computation, we ought to average the predictions of all parameter settings using the posterior distribution to weight the predictions:

$$p(t_{test} \mid x_{test}, \alpha, \beta, D) = \int p(t_{test} \mid x_{test}, \beta, \mathbf{w}) p(\mathbf{w} \mid \alpha, \beta, D) d\mathbf{w}$$



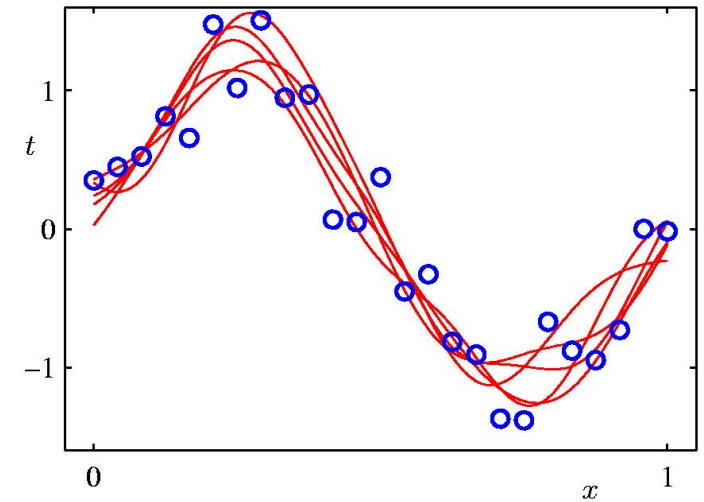
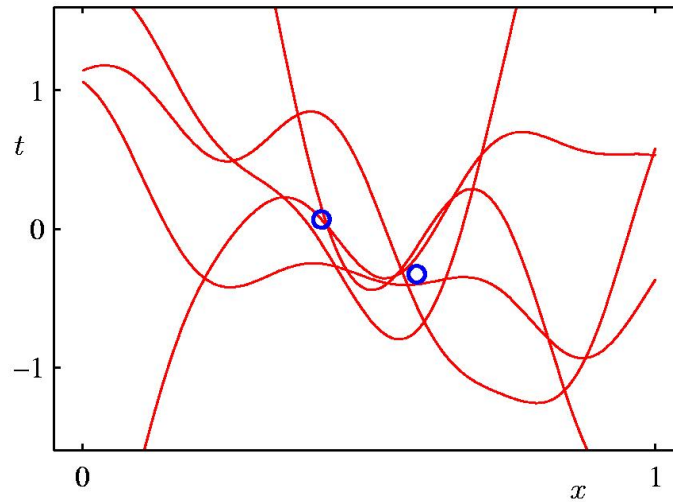
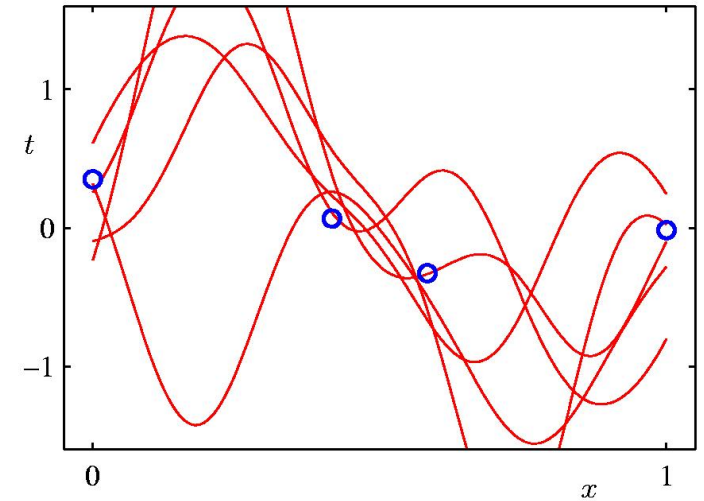
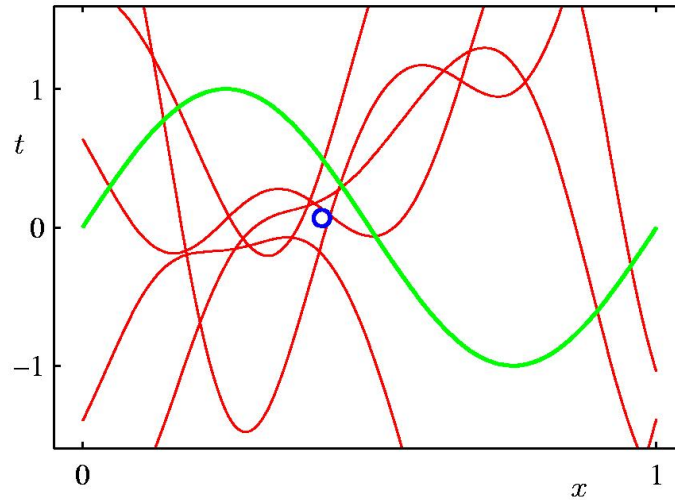
training data precision of output noise precision of prior

Predictive distribution for noisy sinusoidal data modeled by a linear combination of nine radial basis functions.



A way to see the covariance of predictions for different values of x

We sample models at random from the posterior and show the mean of the each model's predictions



Lecture 4: Linear Classification Methods

What is “linear” classification?

Classification is intrinsically non-linear

It puts non-identical things in the same class, so a difference in input vector sometimes causes zero change in the answer
(what does this show?)

“Linear classification” means that the part that adapts is linear

The adaptive part is followed by a fixed non-linearity.

It may be preceded by a fixed non-linearity (e.g. nonlinear basis functions).

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0,$$



adaptive linear
function

$$Decision = f(y(\mathbf{x}))$$



fixed non-
linear function

Representing the target values for classification

For two classes, we use a single valued output that has target values **1** for the “positive” class and **0** (or **-1**) for the other class

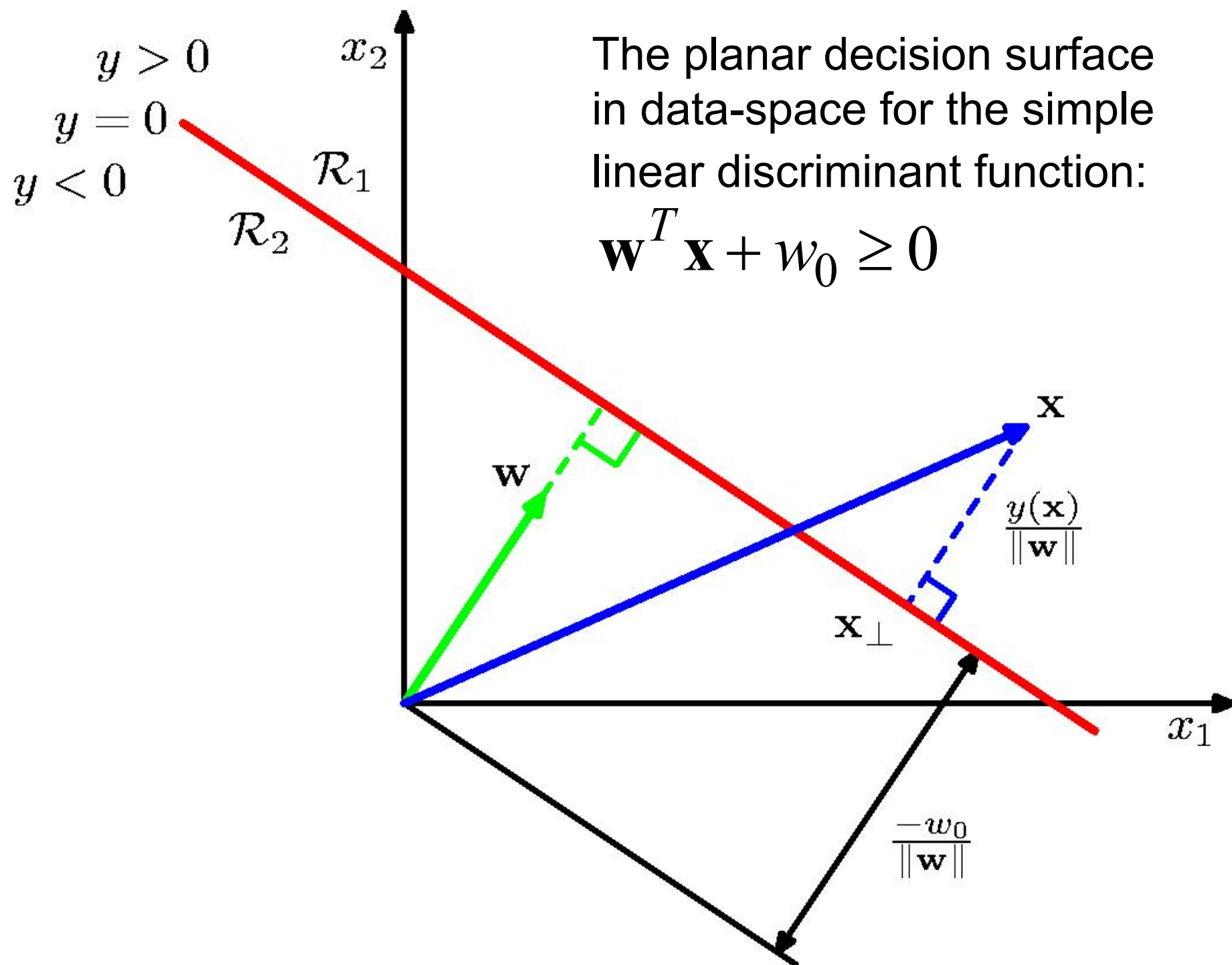
For probabilistic class labels the target value can then be the probability of the positive class and the output of the model can also represent the probability the model gives to the positive class.

For N classes we often use a vector of N target values containing a single **1** for the correct class and zeros elsewhere.

For probabilistic labels we can then use a vector of class probabilities as the target vector.

Three approaches to classification

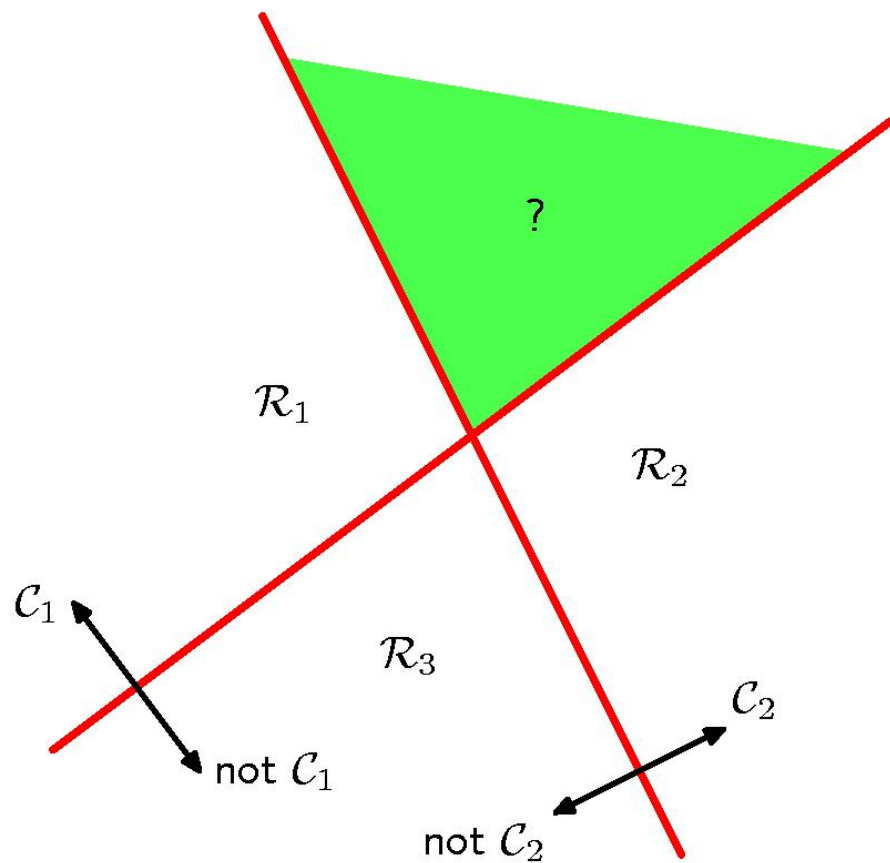
- Use discriminant functions directly without probabilities:
 - Convert the input vector into one or more real values so that a simple operation (like thresholding) can be applied to get the class.
 - The real values should be chosen to maximize the useable information about the class label that is in the real value.
- Infer conditional class probabilities: $p(class = C_k \mid \mathbf{x})$
 - Compute the conditional probability of each class.
 - Then make a decision that minimizes some loss function
- Compare the probability of the input under separate, class-specific, generative models.
 - E.g. fit a multivariate **Gaussian** to the input vectors of each class and see which Gaussian makes a test data vector most probable. (Is this the best bet?)



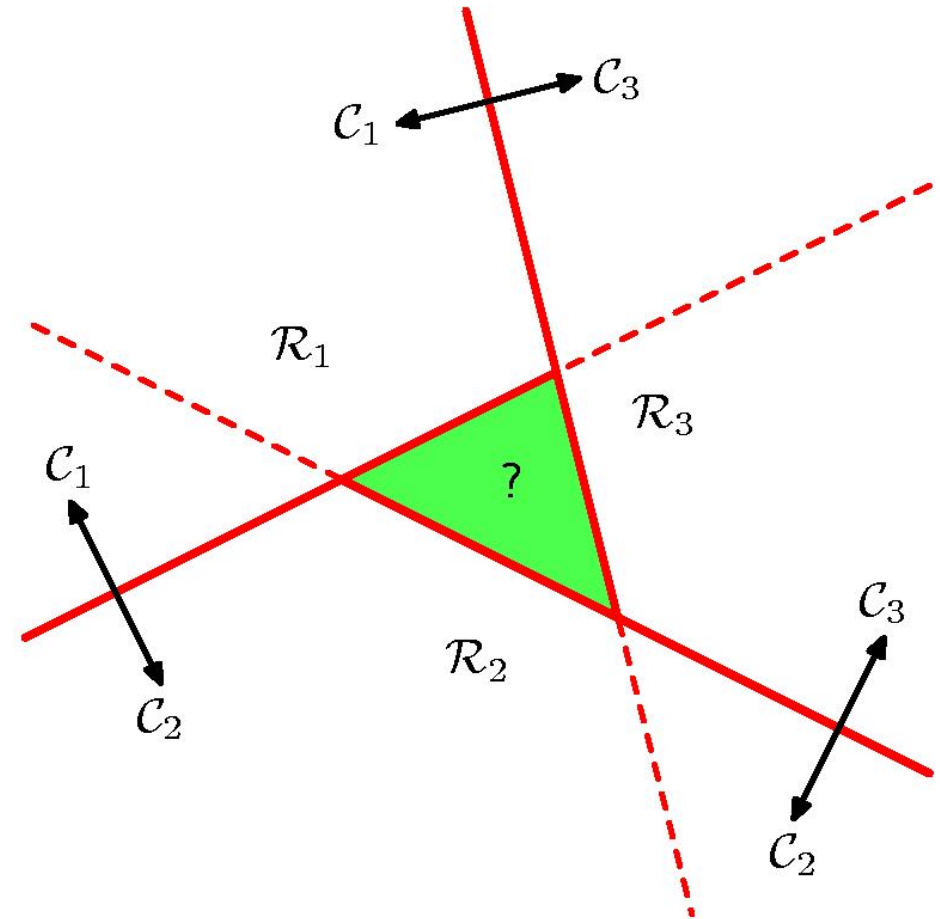
Discriminant functions for $N > 2$ classes

- One possibility is to use N two-way discriminant functions.
 - Each function discriminates one class from the rest.
- Another possibility is to use $N(N-1)/2$ two-way discriminant functions
 - Each function discriminates between two particular classes.
- Both these methods have problems

Problems with multi-class discriminant functions



More than one
good answer

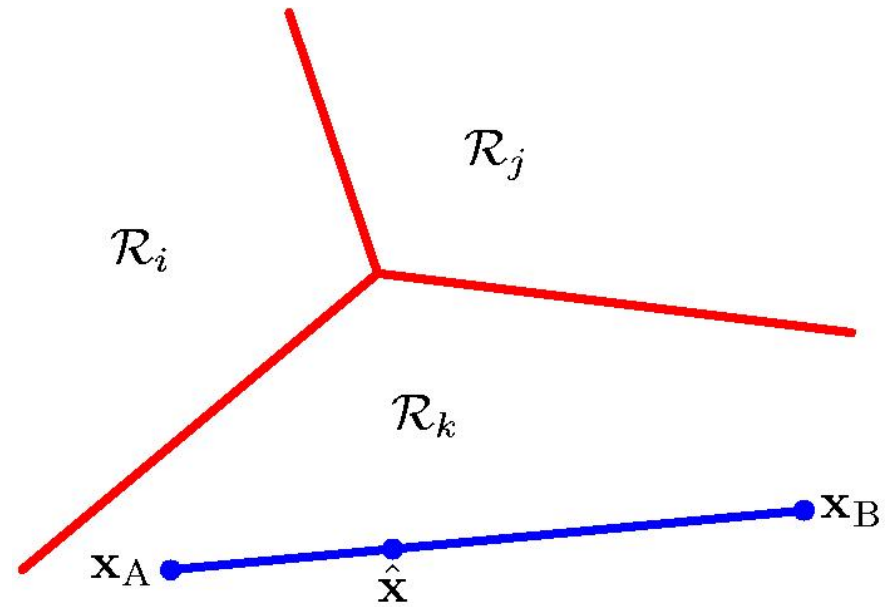


Two-way preferences
need not be transitive!

A simple solution

Use N discriminant functions,
and pick the max. $y_i, y_j, y_k \dots$

- This is guaranteed to give consistent and convex decision regions if y is linear.



$$y_k(\mathbf{x}_A) > y_j(\mathbf{x}_A) \text{ and } y_k(\mathbf{x}_B) > y_j(\mathbf{x}_B)$$

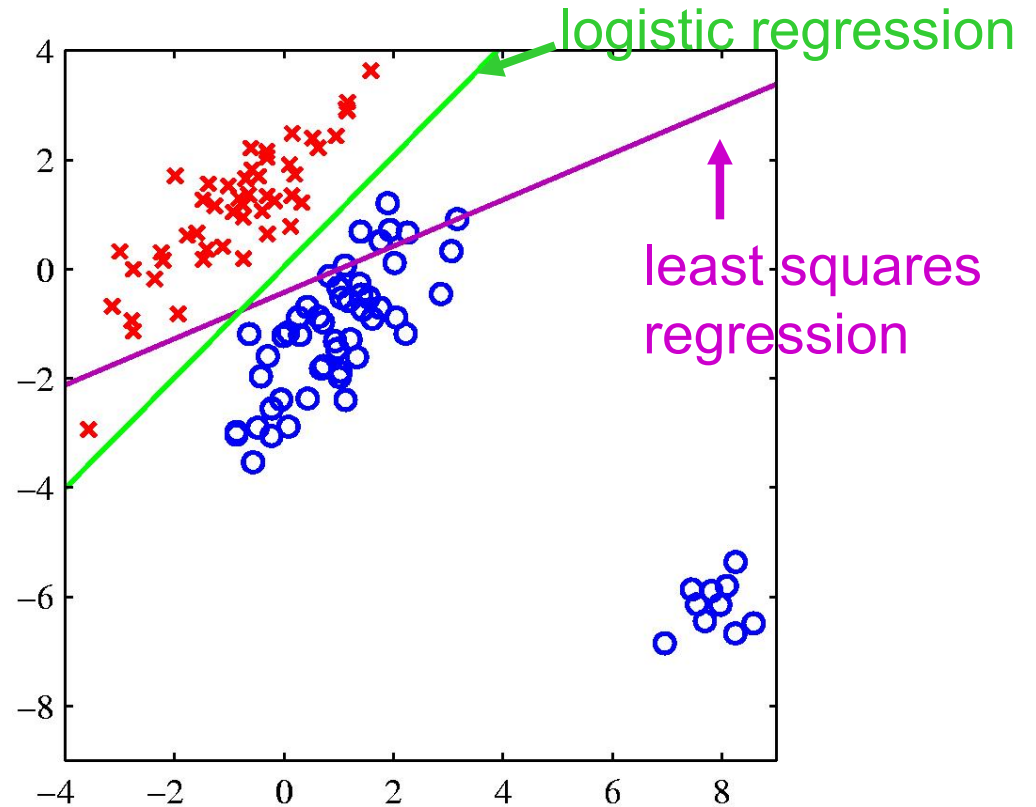
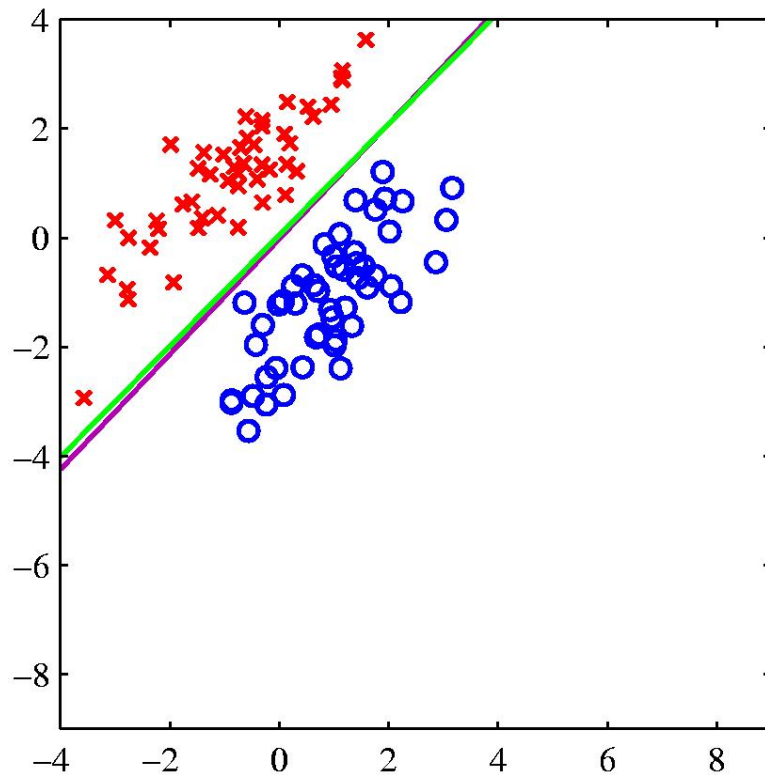
implies (for positive α) that

$$y_k(\alpha \mathbf{x}_A + (1 - \alpha) \mathbf{x}_B) > y_j(\alpha \mathbf{x}_A + (1 - \alpha) \mathbf{x}_B)$$

Using “least squares” for classification

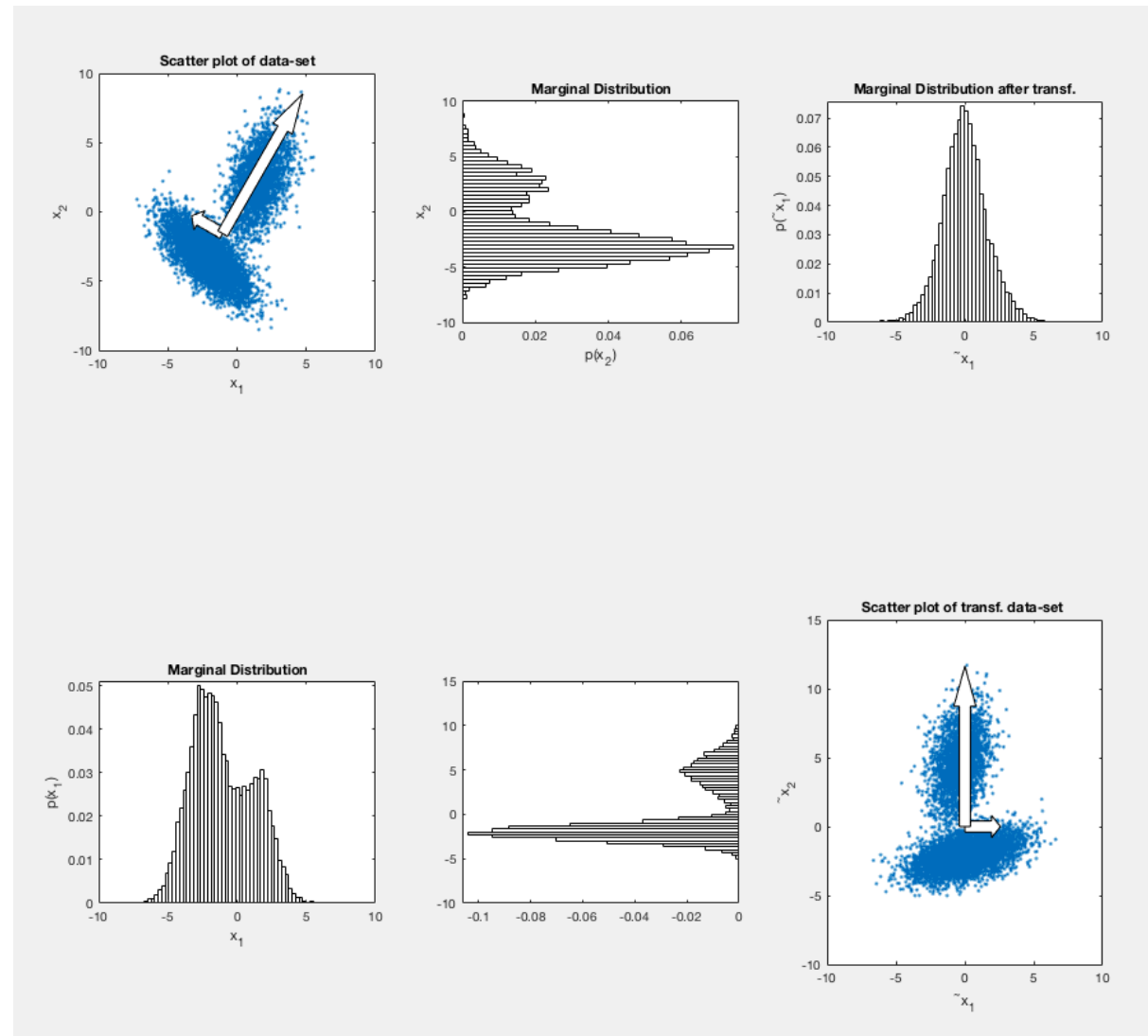
- This is not the right thing to do and it doesn't work as well as better methods, but it is easy:
 - It reduces classification to least squares regression.
 - We already know how to do regression. We can just solve for the optimal weights with some matrix algebra (see last lecture).
- We use targets that are equal to the conditional probability of the class given the input.
 - When there are more than two classes, we treat each class as a separate problem (we cannot get away with this if we use the “max” decision function).

Problems with using least squares for classification

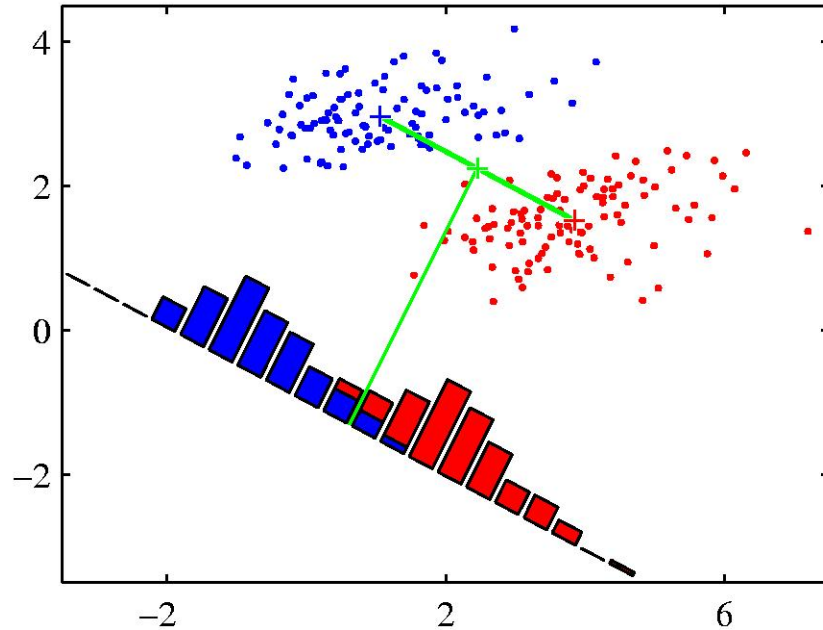


If the right answer is 1 and the model says 1.5, it loses, so it changes the boundary to avoid being “too correct”

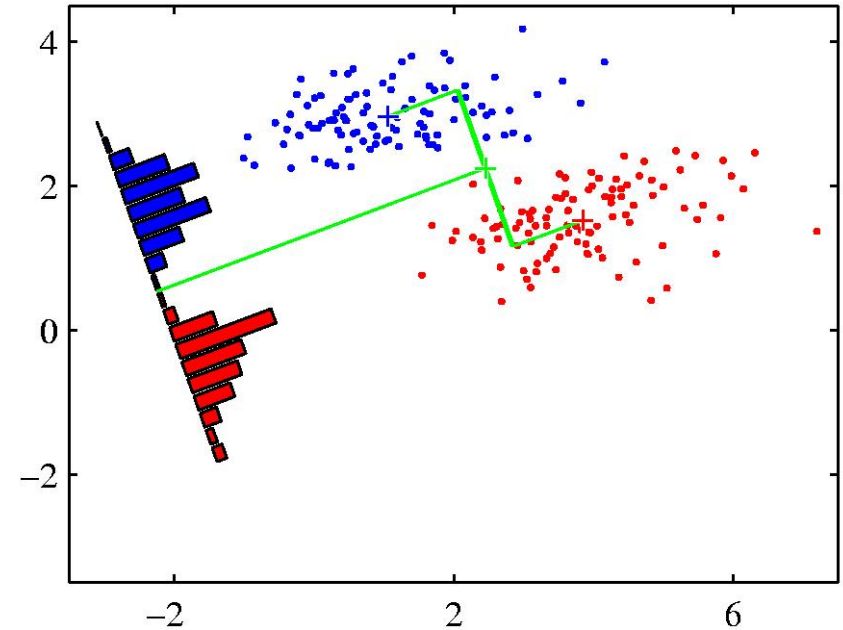
PCA don't work well



picture showing the advantage of Fisher's linear discriminant



When projected onto the line joining the class means, the classes are not well separated.



Fisher chooses a direction that makes the projected classes much tighter, even though their projected means are less far apart.

Math of Fisher's linear discriminants

- What linear transformation is best for discrimination?
- The projection onto the vector separating the class means seems sensible:

$$y = \mathbf{w}^T \mathbf{x}$$

$$\mathbf{w} \propto \mathbf{m}_2 - \mathbf{m}_1$$

- But we also want small variance within each class:

$$s_1^2 = \sum_{n \in C_1} (y_n - m_1)^2$$

$$s_2^2 = \sum_{n \in C_2} (y_n - m_2)^2$$

- Fisher's objective function is:

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

← between
← within

More math of Fisher's linear discriminants

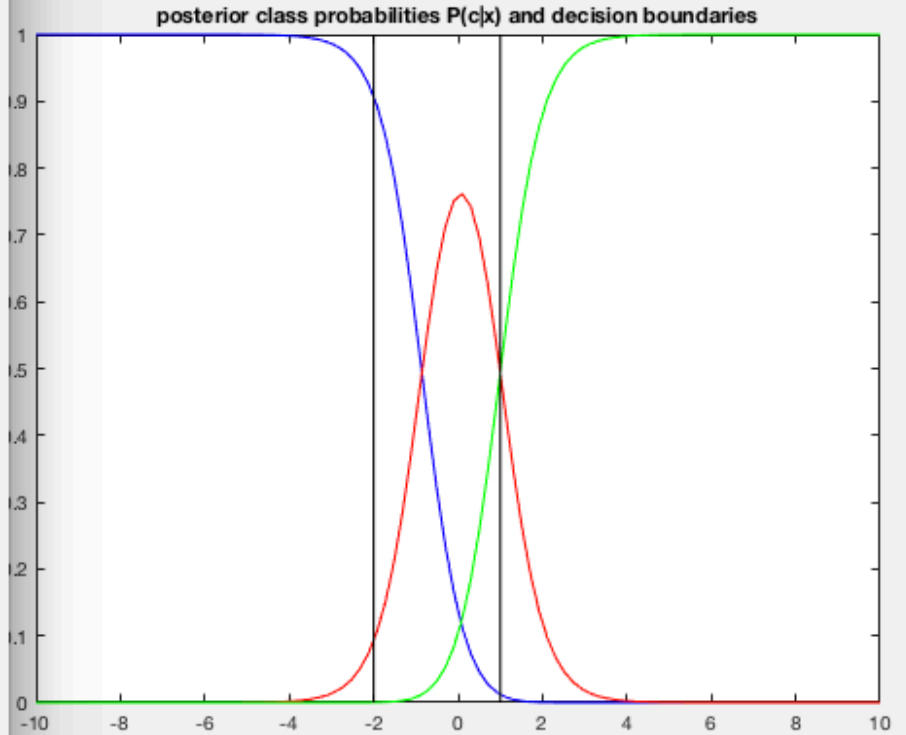
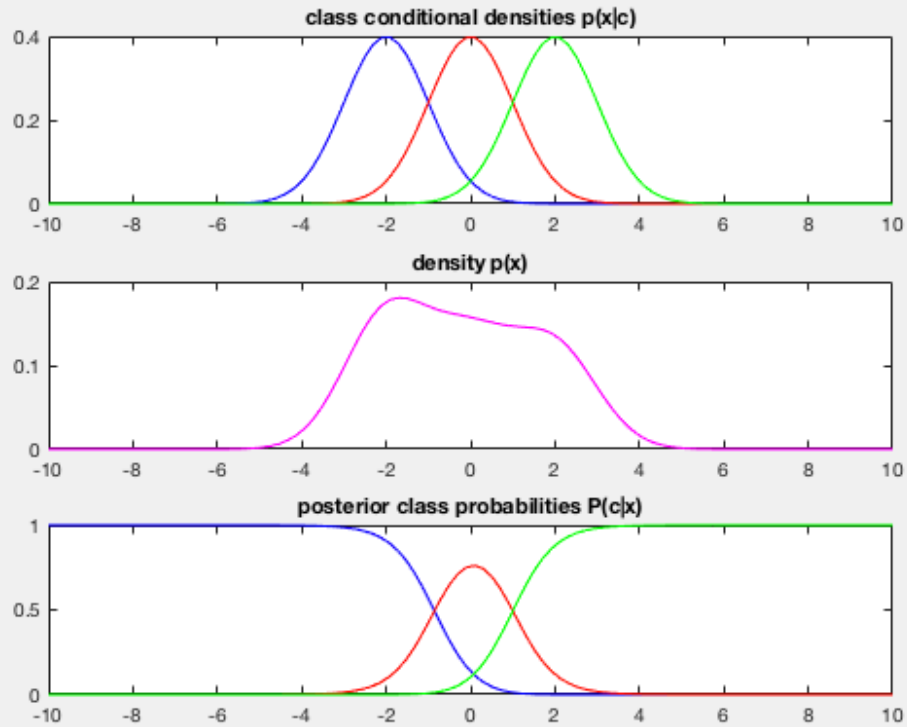
$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1) (\mathbf{m}_2 - \mathbf{m}_1)^T$$

$$\mathbf{S}_W = \sum_{n \in C_1} (\mathbf{x}_n - \mathbf{m}_1) (\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in C_2} (\mathbf{x}_n - \mathbf{m}_2) (\mathbf{x}_n - \mathbf{m}_2)^T$$

Optimal solution: $\mathbf{w} \propto \mathbf{S}_W^{-1} (\mathbf{m}_2 - \mathbf{m}_1)$

We have already done classification!



Logistic regression (jump to page 205)

When there are only two classes we can model the conditional probability of the positive class as

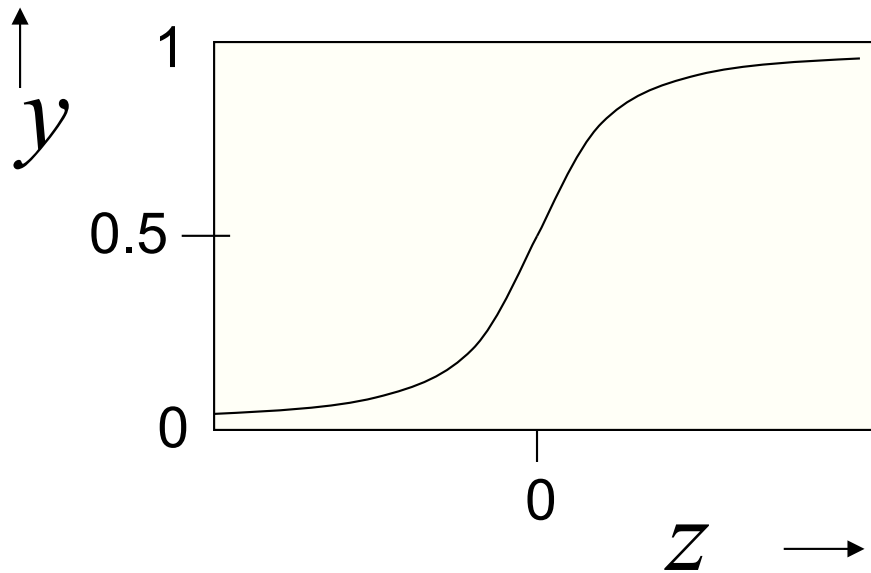
$$p(C_1 | \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + w_0) \quad \text{where} \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

If we use the right error function, something nice happens: The gradient of the logistic and the gradient of the error function cancel each other:

$$E(\mathbf{w}) = -\ln p(\mathbf{t} | \mathbf{w}), \quad \nabla E(\mathbf{w}) = \sum_{n=1}^N (y_n - t_n) \mathbf{x}_n$$

The logistic function

The output is a smooth function of the inputs and the weights.



$$z = \mathbf{w}^T \mathbf{x} + w_0$$

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{\partial z}{\partial w_i} = x_i \quad \frac{\partial z}{\partial x_i} = w_i$$

$$\frac{dy}{dz} = y(1 - y)$$





Its odd to express it
in terms of y.


The natural error function for the logistic

To fit a logistic model using maximum likelihood, we need to minimize the negative log probability of the correct answer summed over the training set.

$$\begin{aligned} E &= -\sum_{n=1}^N \ln p(t_n | y_n) \\ &= -\sum_{n=1}^N t_n \ln y_n + (1-t_n) \ln(1-y_n) \end{aligned}$$

 if t = 1  if t = 0

error derivative on
training case n


$$\begin{aligned} \frac{\partial E_n}{\partial y_n} &= -\frac{t_n}{y_n} + \frac{1-t_n}{1-y_n} \\ &= \frac{y_n - t_n}{y_n(1-y_n)} \end{aligned}$$

Using the chain rule to get the error derivatives

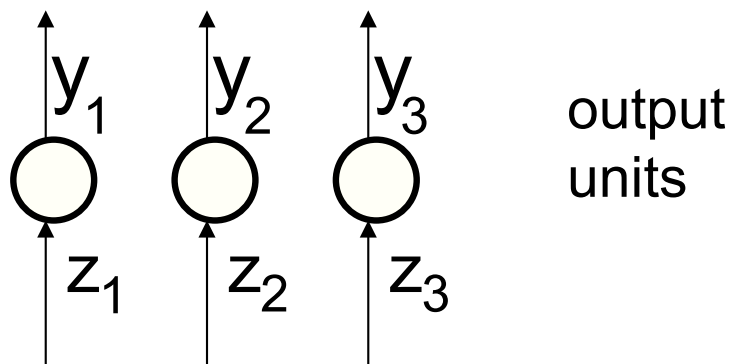
$$z_n = \mathbf{w}^T \mathbf{x}_n + w_0, \quad \frac{\partial z_n}{\partial \mathbf{w}} = \mathbf{x}_n$$

$$\frac{\partial E_n}{\partial y_n} = \frac{y_n - t_n}{y_n(1 - y_n)}, \quad \frac{dy_n}{dz_n} = y_n(1 - y_n)$$

$$\frac{\partial E_n}{\partial \mathbf{w}} = \frac{\partial E_n}{\partial y_n} \frac{dy_n}{dz_n} \frac{\partial z_n}{\partial \mathbf{w}} = (y_n - t_n) \mathbf{x}_n$$

Cross-entropy or “softmax” error function for multi-class classification

The output units use a non-local non-linearity:



The natural cost function is the negative log prob of the right answer

The steepness of E exactly balances the flatness of the softmax.

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$$\frac{\partial y_i}{\partial z_i} = y_i (1 - y_i)$$

target value

$$E = - \sum_j \downarrow t_j \ln y_j$$

$$\frac{\partial E}{\partial z_i} = \sum_j \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_i} = y_i - t_i$$

Probabilistic Generative Models for Discrimination (p 196)

Use a generative model of the input vectors for each class,
and see which model makes a test input vector most probable.

The posterior probability of class 1 is:

$$p(C_1 | \mathbf{x}) = \frac{p(C_1)p(\mathbf{x} | C_1)}{p(C_1)p(\mathbf{x} | C_1) + p(C_0)p(\mathbf{x} | C_0)} = \frac{1}{1 + e^{-z}}$$

$$\text{where } z = \ln \frac{p(C_1)p(\mathbf{x} | C_1)}{p(C_0)p(\mathbf{x} | C_0)} = \boxed{\ln \frac{p(C_1 | \mathbf{x})}{1 - p(C_1 | \mathbf{x})}}$$



z is called the **logit** and is
given by the **log odds**

A simple example for continuous inputs

Assume input vectors for each class are Gaussian, and all classes have the same covariance matrix.

$$p(\mathbf{x} | C_k) = \underset{\substack{\text{normalizing} \\ \text{constant}}}{a} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \underset{\substack{\text{inverse} \\ \text{covariance matrix}}}{\boldsymbol{\Sigma}^{-1}} (\mathbf{x} - \boldsymbol{\mu}_k) \right\}$$

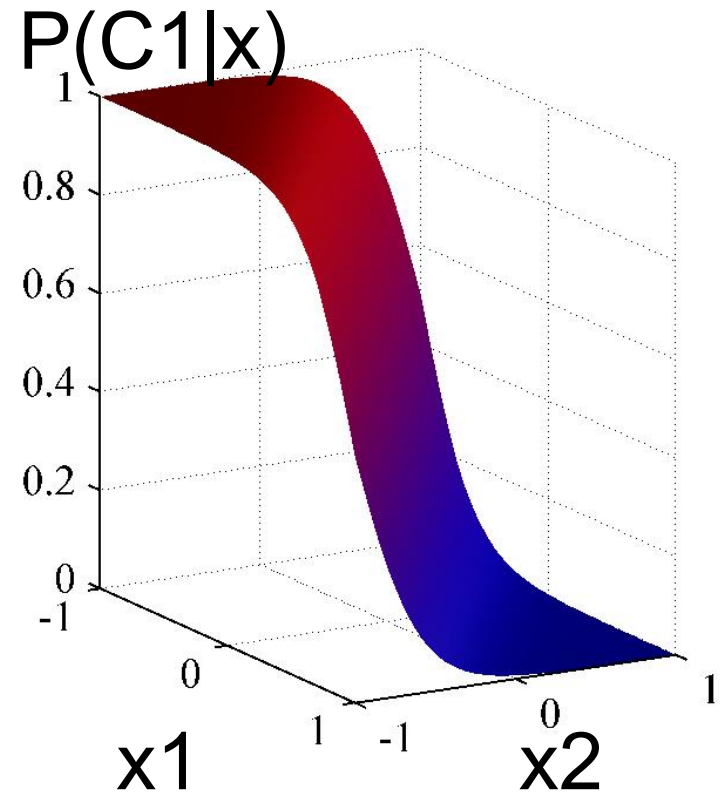
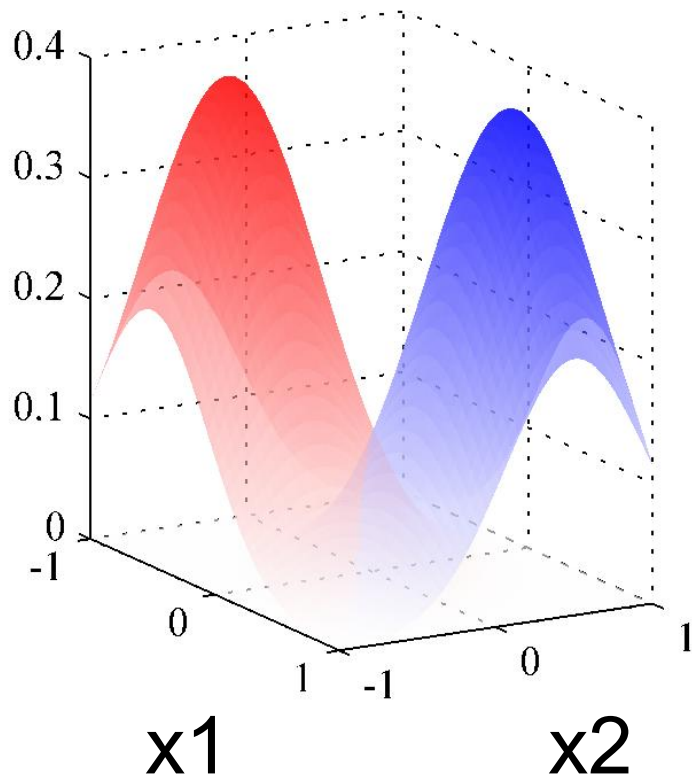
For two classes, C_1 and C_0 , the posterior is a logistic:

$$p(C_1 | \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + w_0)$$

$$\mathbf{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$$

$$w_0 = -\frac{1}{2} \boldsymbol{\mu}_1^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1 + \frac{1}{2} \boldsymbol{\mu}_0^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_0 + \ln \frac{p(C_1)}{p(C_0)}$$

Two Gaussian models and the resulting posterior for the red class



A way of thinking about the role of the inverse covariance matrix

- If the Gaussian is spherical no need to worry about the covariance matrix.
- So we could start by transforming the data space to make the Gaussian spherical
 - This is called “whitening” the data.
 - It pre-multiplies by the matrix square root of the inverse covariance matrix.
- In the transformed space, the weight vector is the difference between the transformed means.

$$\mathbf{w} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$$

gives the same value

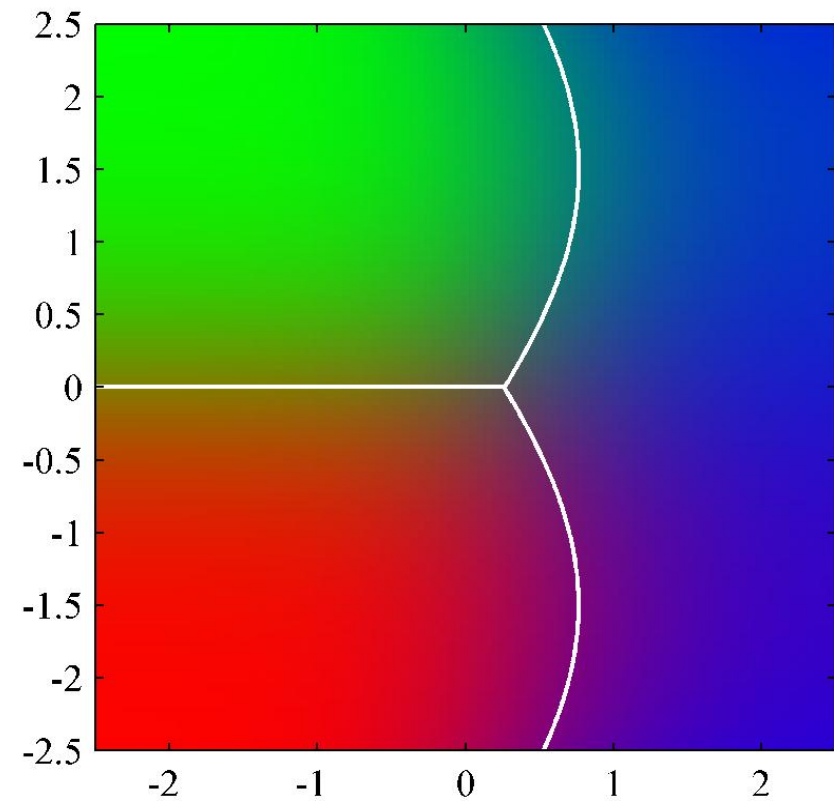
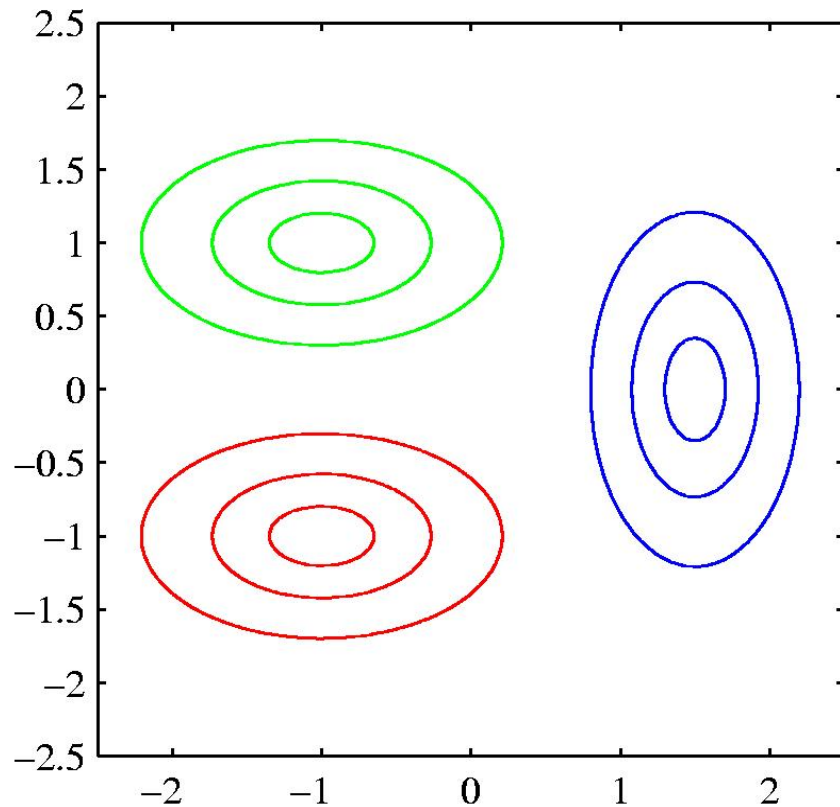
for $\mathbf{w}^T \mathbf{x}$ as :

$$\mathbf{w}_{aff} = \Sigma^{-\frac{1}{2}} \boldsymbol{\mu}_1 - \Sigma^{-\frac{1}{2}} \boldsymbol{\mu}_0$$

$$\text{and } \mathbf{x}_{aff} = \Sigma^{-\frac{1}{2}} \mathbf{x}$$

gives for $\mathbf{w}_{aff}^T \mathbf{x}_{aff}$

The posterior when the covariance matrices are different for different classes



The decision surface is planar when the covariance matrices are the same and quadratic when not.