CSC2515 Fall 2007
Introduction to Machine Learning

Lecture 3: Linear Classification Methods

All lecture slides will be available as .ppt, .ps, & .htm at
www.cs.toronto.edu/~hinton

Many of the figures are provided by Chris Bishop
from his textbook: "Pattern Recognition and Machine Learning"

What is "linear" classification?

- Classification is intrinsically non-linear
  - It puts non-identical things in the same class, so a difference in the 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 also be preceded by a fixed non-linearity (e.g. nonlinear basis functions).

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

adaptive linear function

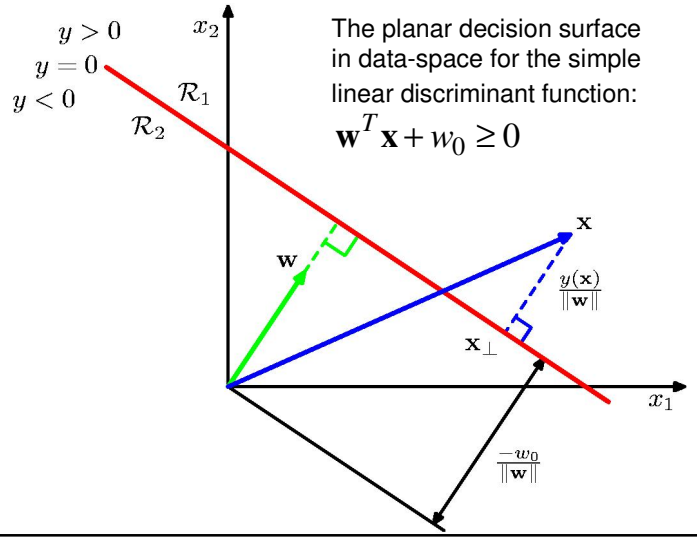
fixed non-linear function

Representing the target values for classification

- If there are only two classes, we typically use a single real valued output that has target values of 1 for the "positive" class and 0 (or sometimes -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.
- If there are 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 threshholding) 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?)



The planar decision surface in data-space for the simple linear discriminant function:

$$\mathbf{w}^T\mathbf{x} + w_0 \geq 0$$

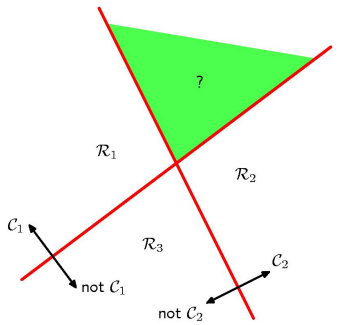
Reminder: Three different spaces that are easy to confuse

- Weight-space
  - Each axis corresponds to a weight
  - A point is a weight vector
  - Dimensionality = #weights +1 extra dimension for the loss
- Data-space
  - Each axis corresponds to an input value
  - A point is a data vector. A decision surface is a plane.
  - Dimensionality = dimensionality of a data vector
- "Case-space" (used in Bishop figure 3.2)
  - Each axis corresponds to a training case
  - A point assigns a scalar value to every training case
    - So it can represent the 1-D targets or it can represent the value of one input component over all the training data.
  - Dimensionality = #training cases

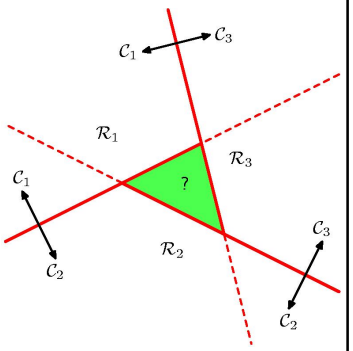
Discriminant functions for N>2 classes

- 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



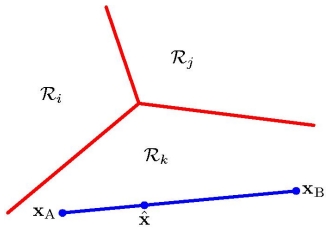
More than one good answer



Two-way preferences need not be transitive!

A simple solution

- Use N discriminant functions, y_i, y_j, y_k ... and pick the max.
  - This is guaranteed to give consistent and convex decision regions if y is linear.



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

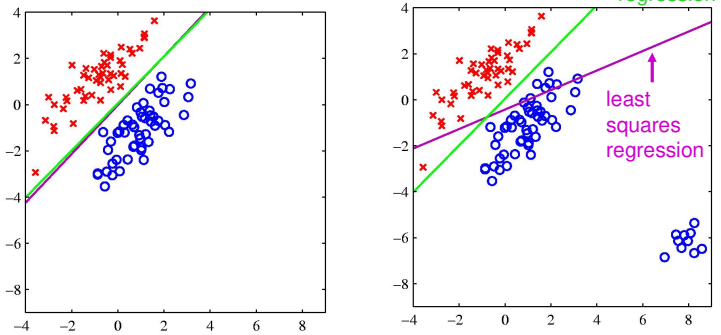
$$implies \; (for \;\; positive \; \alpha) \;\; that$$

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

Using "least squares" for classification

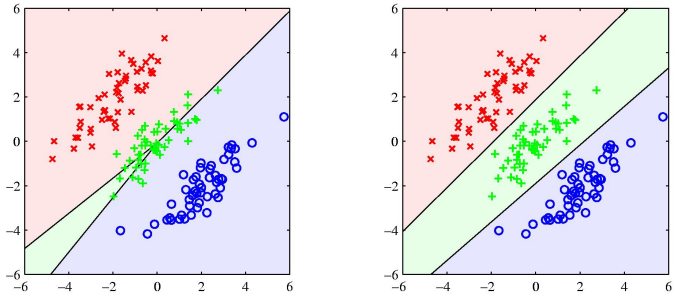
- This is not the right thing to do and it doesn't work as well as better methods, but it is easy:
  - It reduces classification to least squares regression.
  - We already know how to do regression. We can just solve for the optimal weights with some matrix algebra (see lecture 2).
- We use targets 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"

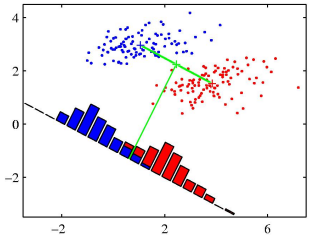
Another example where least squares regression gives poor decision surfaces



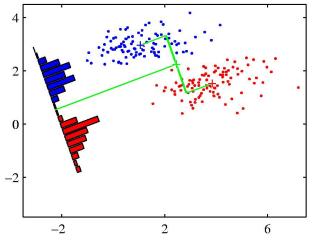
Fisher's linear discriminant

- A simple linear discriminant function is a projection of the data down to 1-D.
  - So choose the projection that gives the best separation of the classes. What do we mean by "best separation"?
- An obvious direction to choose is the direction of the line joining the class means.
  - But if the main direction of variance in each class is not orthogonal to this line, this will not give good separation (see the next figure).
- Fisher's method chooses the direction that maximizes the ratio of between class variance to within class variance.
  - This is the direction in which the projected points contain the most information about class membership (under Gaussian assumptions)

A 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?

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

- The projection onto the vector separating the class means seems sensible:

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

- But we also want small variance within each class:

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

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

- 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

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

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

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

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

Perceptrons

- "Perceptrons" describes a whole family of learning machines, but the standard type consisted of a layer of fixed non-linear basis functions followed by a simple linear discriminant function.
  - They were introduced in the late 1950's and they had a simple online learning procedure.
  - Grand claims were made about their abilities. This led to lots of controversy.
  - Researchers in symbolic AI emphasized their limitations (as part of an ideological campaign against real numbers, probabilities, and learning)
- Support Vector Machines are just perceptrons with a clever way of choosing the non-adaptive, non-linear basis functions and a better learning procedure.
  - They have all the same limitations as perceptrons in what types of function they can learn.
    - But people seem to have forgotten this.

The perceptron convergence procedure

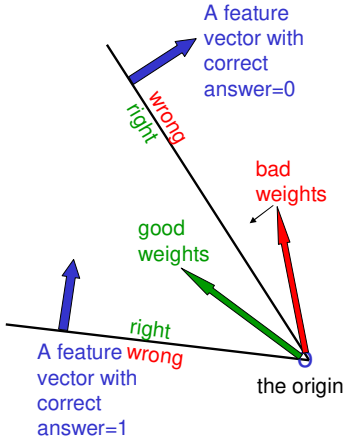
- Add an extra component with value 1 to each feature vector. The "bias" weight on this component is minus the threshold. Now we can forget the threshold.
- Pick training cases using any policy that ensures that every training case will keep getting picked
  - If the output is correct, leave its weights alone.
  - If the output is 0 but should be 1, add the feature vector to the weight vector.
  - If the output is 1 but should be 0, subtract the feature vector from the weight vector
- This is guaranteed to find a set of weights that gets the right answer on the whole training set if any such set exists
- There is no need to choose a learning rate.

A natural way to try to prove convergence

- The obvious approach is to write down an error function and try to show that each step of the learning procedure reduces the error.
  - For stochastic online learning we would like to show that each step reduces the expected error, where the expectation is across the choice of training cases.
  - It cannot be a squared error because the size of the update does not depend on the size of the mistake.
- The textbook tries to use the sum of the distances on the wrong side of the decision surface as an error measure.
  - Its conclusion is that the perceptron convergence procedure is not guaranteed to reduce the total error at each step.
    - This is true for that error function even if there is a set of weights that gets the right answer for every training case.

Weight and data space

- Imagine a space in which each axis corresponds to a feature value or to the weight on that feature
  - A point in this space is a weight vector. Feature vectors are shown in blue translated away from the origin to reduce clutter.
- Each training case defines a plane.
  - On one side of the plane the output is wrong.
- To get all training cases right we need to find a point on the right side of all the planes.
  - This feasible region (if it exists) is a cone with its tip at the origin

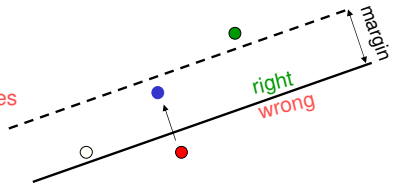


A better way to prove the convergence
(using the convexity of the solutions in weight-space)

- The obvious type of error function measures the discrepancy between the targets and the model's outputs.
- A different type of cost function is to use the squared distance between the current weights and a feasible set of weights.
  - Using this cost function we can show that every step of the procedure reduces the error.
    - Provided a set of feasible weights exists.
- Using this type of cost function, the procedure can easily be generalized to more than two classes using the MAX decision rule.

Why the learning procedure works

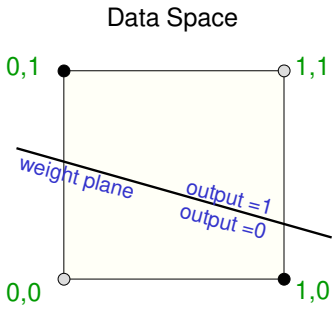
- Consider the squared distance between any satisfactory weight vector and the current weight vector.
  - Every time the perceptron makes a mistake, the learning algorithm reduces the squared distance between the current weight vector and any satisfactory weight vector (unless it crosses the decision plane).
- So consider "generously satisfactory" weight vectors that lie within the feasible region by a margin at least as great as the largest update.
  - Every time the perceptron makes a mistake, the squared distance to all of these weight vectors is always decreased by at least the squared length of the smallest update vector.



What perceptrons cannot learn

- The adaptive part of a perceptron cannot even tell if two single bit features have the same value!
  - Same: (1,1) → 1; (0,0) → 1
  - Different: (1,0) → 0; (0,1) → 0
- The four feature-output pairs give four inequalities that are impossible to satisfy:

$$w_1 + w_2 \geq \theta, \quad 0 \geq \theta$$
$$w_1 < \theta, \qquad w_2 < \theta$$



The positive and negative cases cannot be separated by a plane

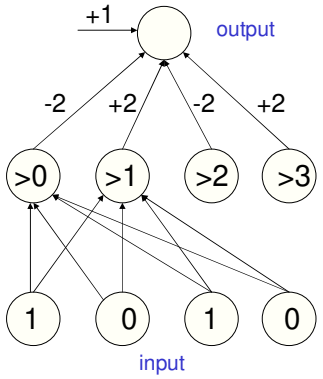
What can perceptrons do?

- They can only solve tasks if the hand-coded features convert the original task into a linearly separable one. How difficult is this?
- The N-bit parity task :
  - Requires N features of the form: Are at least m bits on?
  - Each feature must look at all the components of the input.
- The 2-D connectedness task
  - requires an exponential number of features!

The N-bit even parity task

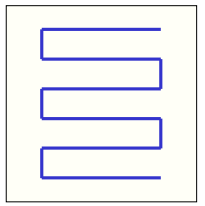
- There is a simple solution that requires N hidden units.
  - Each hidden unit computes whether more than M of the inputs are on.
  - This is a linearly separable problem.
- There are many variants of this solution.
  - It can be learned.
  - It generalizes well if:

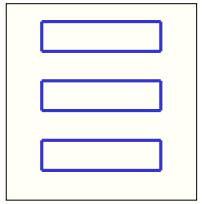
$$2^N >> N^2$$



Why connectedness is hard to compute

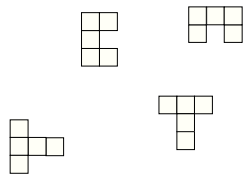
- Even for simple line drawings, there are exponentially many cases.
- Removing one segment can break connectedness
  - But this depends on the precise arrangement of the other pieces.
  - Unlike parity, there are no simple summaries of the other pieces that tell us what will happen.
- Connectedness is easy to compute with an iterative algorithm.
  - Start anywhere in the ink
  - Propagate a marker
  - See if all the ink gets marked.





Distinguishing T from C in any orientation and position

- What kind of features are required to distinguish two different patterns of 5 pixels independent of position and orientation?
  - Do we need to replicate T and C templates across all positions and orientations?
  - Looking at pairs of pixels will not work
  - Looking at triples will work if we assume that each input image only contains one object.



Replicate the following two feature detectors in all positions

If any of these equal their threshold of 2, it's a C. If not, it's a T.

Logistic regression (jump to page 205)

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

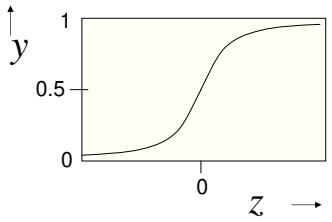
$$p(C_1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + w_0) \qquad where \quad \sigma(z) = \frac{1}{1+\exp(-z)}$$

- 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:

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

The logistic function

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



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

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

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

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

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.

$$E = -\sum_{n=1}^{N} \ln p(t_n \mid y_n)$$

$$= -\sum_{n=1}^{N} t_n \ln y_n + (1-t_n)\ln(1-y_n)$$

if t =1 if t =0

error derivative on training case n

$$\frac{\partial E_n}{\partial y_n} = -\frac{t_n}{y_n} + \frac{1-t_n}{1-y_n}$$

$$= \frac{y_n - t_n}{y_n(1-y_n)}$$

Using the chain rule to get the error derivatives

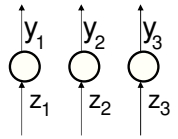
$$z_n = \mathbf{w}^T \mathbf{x}_n + w_0, \qquad \frac{\partial z_n}{\partial \mathbf{w}} = \mathbf{x}_n$$

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

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

The cross-entropy or "softmax" error function for multi-class classification

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



output units

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

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

target value

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

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

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

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

A special case of softmax for two classes

$$y_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_0}} = \frac{1}{1 + e^{-(z_1 - z_0)}}$$

- So the logistic is just a special case that avoids using redundant parameters:
  - Adding the same constant to both z1 and z0 has no effect.
  - The over-parameterization of the softmax is because the probabilities must add to 1.

Probabilistic Generative Models for Discrimination

- Use a separate 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 given by:

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

$$where \quad z = \ln \frac{p(C_1)p(\mathbf{x} \mid C_1)}{p(C_0)p(\mathbf{x} \mid C_0)} = \ln \frac{p(C_1 \mid \mathbf{x})}{1 - p(C_1 \mid \mathbf{x})}$$

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

A simple example for continuous inputs

- Assume that the input vectors for each class are from a Gaussian distribution, and all classes have the same covariance matrix.

normalizing constant; inverse covariance matrix

$$p(\mathbf{x} \mid C_k) = a \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right\}$$

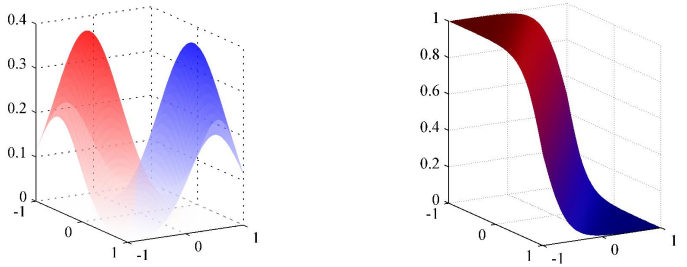
- For two classes, C1 and C0, the posterior is a logistic:

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

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

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

A picture of the 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 we don't 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 just the difference between the transformed means.

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

gives the same value

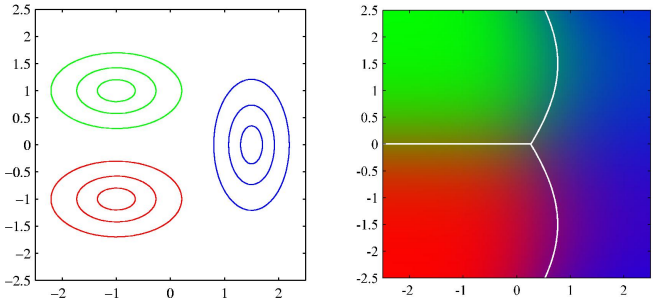
for $\mathbf{w}^T\mathbf{x}$ *as*:

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

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

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

The posterior when the covariance matrices are different for different classes.

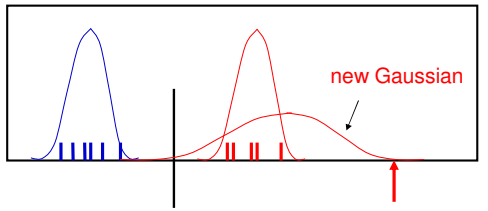


The decision surface is planar when the covariance matrices are the same and quadratic when they are not.

Two ways to train a set of class-specific generative models

- Generative approach Train each model separately to fit the input vectors of that class.
  - Different models can be trained on different cores.
  - It is easy to add a new class without retraining all the other classes
- These are significant advantages when the models are harder to train than the simple linear models considered here.
- Discriminative approach Train all of the parameters of both models to maximize the probability of getting the labels right.

An example where the two types of training behave very differently



For generative fitting, the red mean moves rightwards but the decision boundary moves leftwards! If you really believe its Gaussian data this is sensible.