Last updated: Oct 22, 2012

LINEAR CLASSIFIERS

J. Elder

CSE 4404/5327 Introduction to Machine Learning and Pattern Recognition

Problems

- Please do Problem 8.3 in the textbook. We will discuss this in class.

Classification: Problem Statement

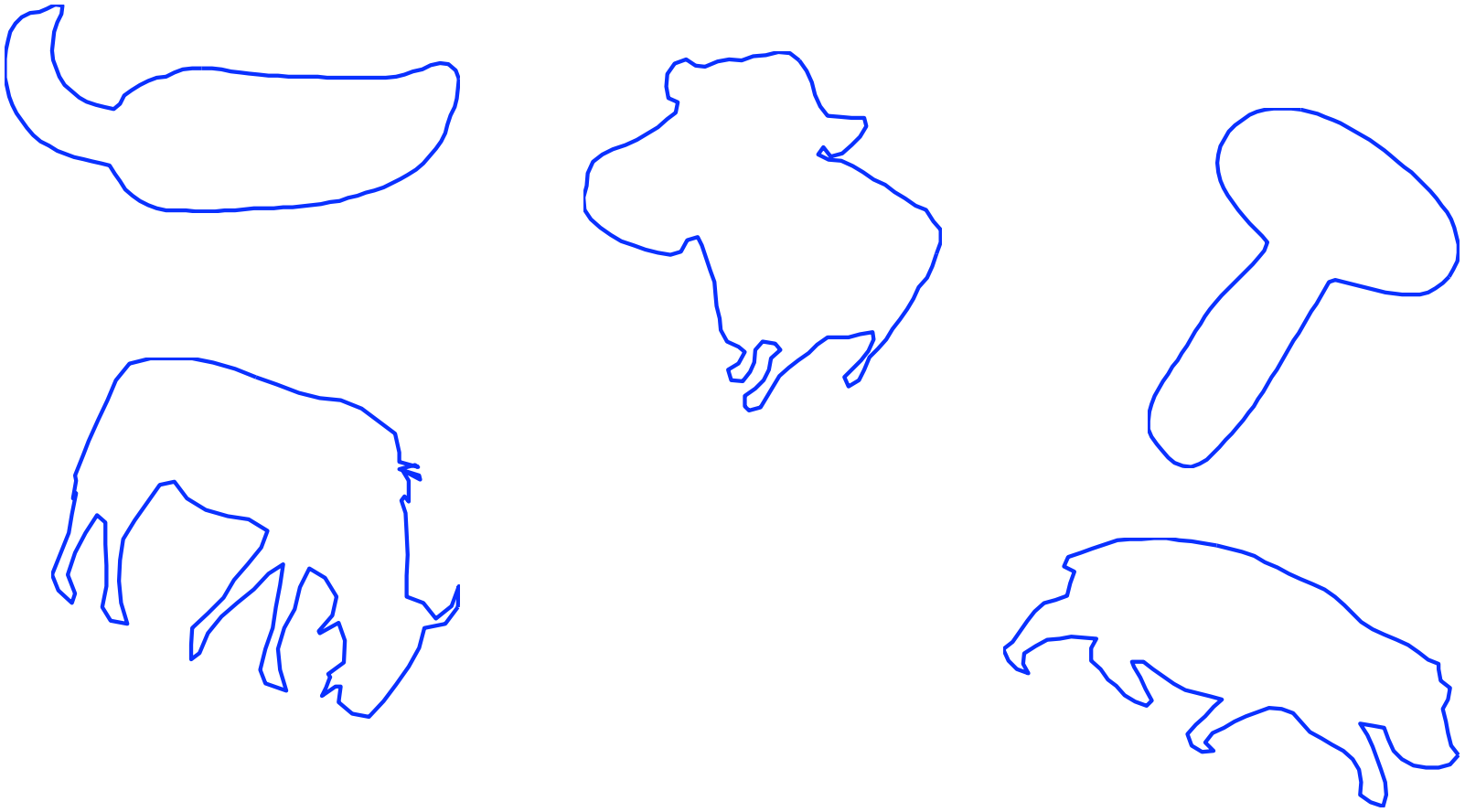
- In regression, we are modeling the relationship between a continuous input variable **x** and a continuous target variable *t*.
- In classification, the input variable **x** may still be continuous, but the target variable is discrete.
- In the simplest case, *t* can have only 2 values.

e.g., Let $t = +1 \leftrightarrow \mathbf{x}$ assigned to C_1

$t = -1 \leftrightarrow \mathbf{x}$ assigned to C_2

Example Problem

- Animal or Vegetable?



Discriminative Classifiers

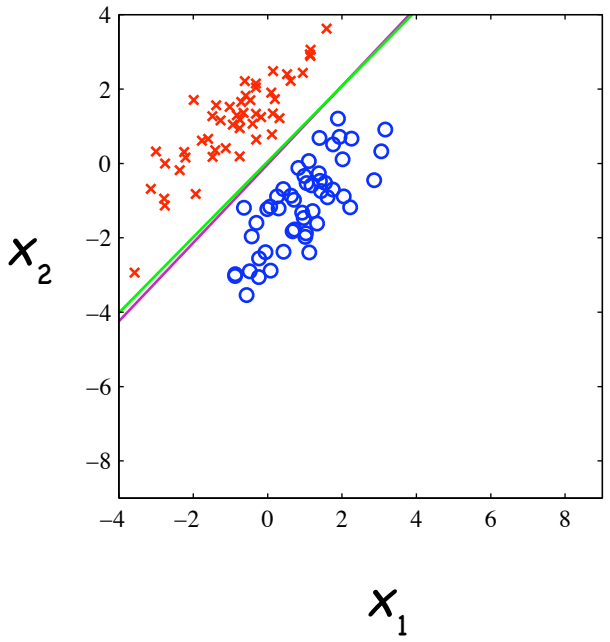
- If the conditional distributions are normal, the best thing to do is to estimate the parameters of these distributions and use Bayesian decision theory to classify input vectors. Decision boundaries are generally quadratic.
- However if the conditional distributions are not exactly normal, this **generative** approach will yield sub-optimal results.
- Another alternative is to build a **discriminative** classifier, that focuses on modeling the decision boundary directly, rather than the conditional distributions themselves.

Linear Models for Classification

- Linear models for classification separate input vectors into classes using linear (hyperplane) *decision boundaries*.
  - Example:

2*D* Input vector **x**

Two discrete classes C_1 and C_2



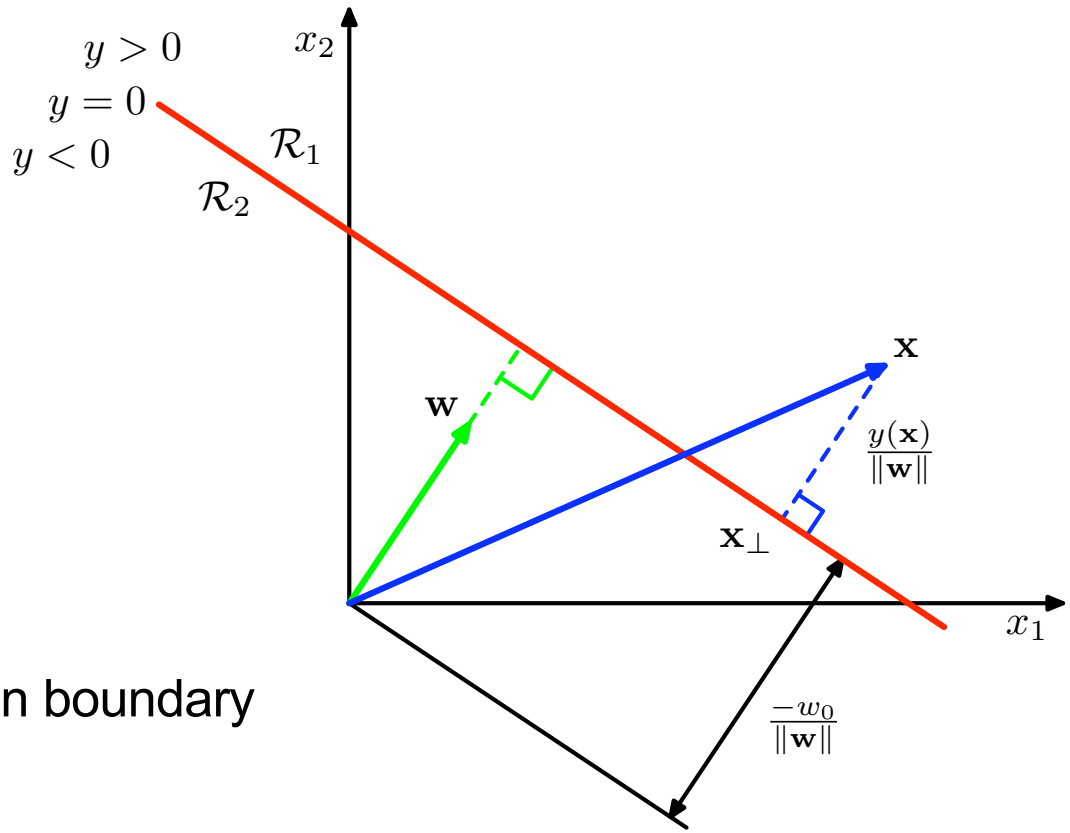
Two Class Discriminant Function

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

$y(\mathbf{x}) \geq 0 \rightarrow \mathbf{x}$ assigned to C_1

$y(\mathbf{x}) < 0 \rightarrow \mathbf{x}$ assigned to C_2

Thus $y(\mathbf{x}) = 0$ defines the decision boundary

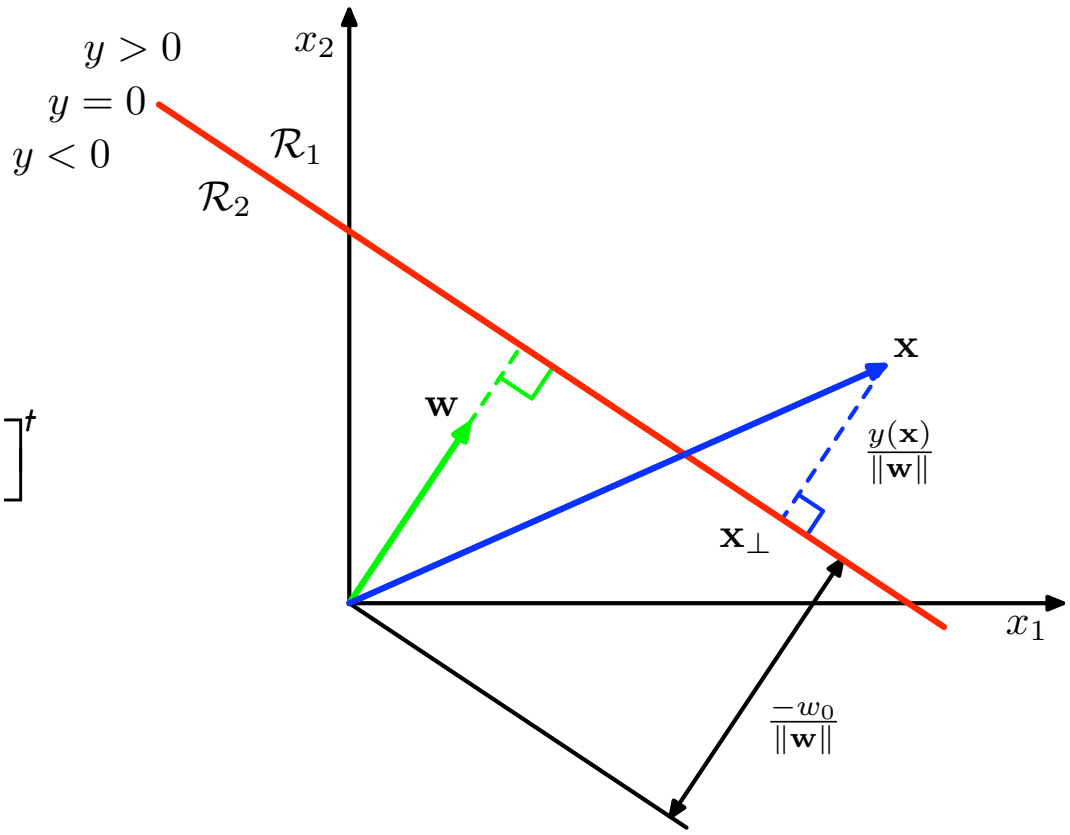


Two-Class Discriminant Function

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

$y(\mathbf{x}) \geq 0 \rightarrow \mathbf{x}$ assigned to C_1

$y(\mathbf{x}) < 0 \rightarrow \mathbf{x}$ assigned to C_2



For convenience, let

$$\mathbf{w} = \left[w_1 \ldots w_M\right]^t \Rightarrow \left[w_0 \;\; w_1 \ldots w_M\right]^t$$

and

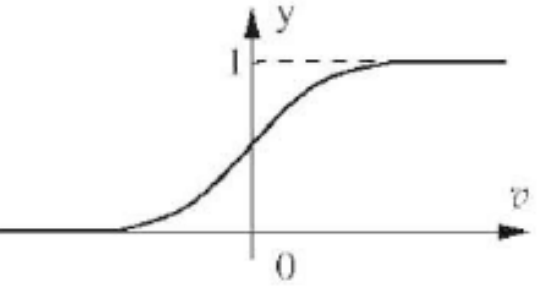
$$\mathbf{x} = \left[x_1 \ldots x_M\right]^t \Rightarrow \left[1 \;\; x_1 \ldots x_M\right]^t$$

So we can express $y(\mathbf{x}) = \mathbf{w}^t\mathbf{x}$

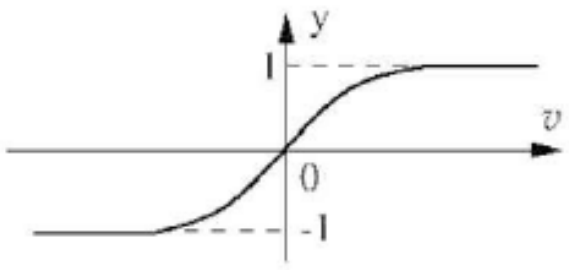
Generalized Linear Models

- For classification problems, we want *y* to be a predictor of *t*. In other words, we wish to map the input vector into one of a number of discrete classes, or to posterior probabilities that lie between 0 and 1.
- For this purpose, it is useful to elaborate the linear model by introducing a nonlinear activation function *f*, which typically will constrain *y* to lie between -1 and 1 or between 0 and 1.

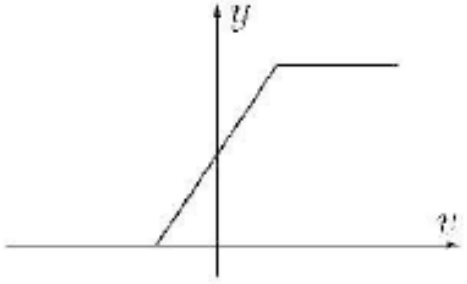
$$y(\mathbf{x}) = f\left(\mathbf{w}^t\mathbf{x} + w_0\right)$$



Log-sigmoid function



Tan-sigmoid function



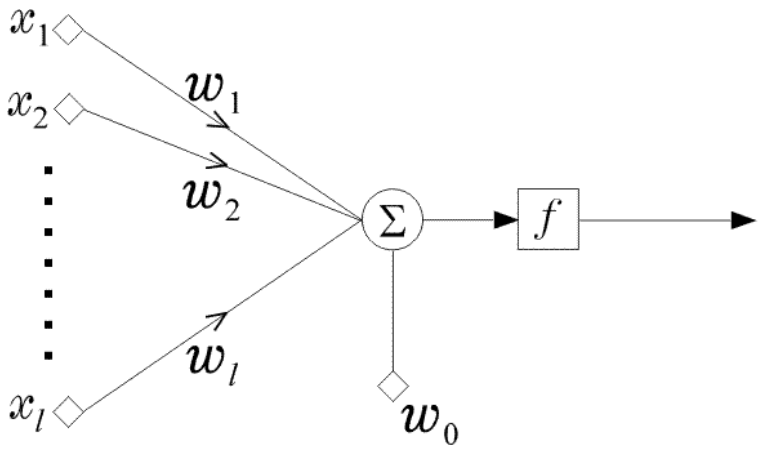
Linear function

The Perceptron

$$y(\mathbf{x}) = f\left(\mathbf{w}^t\mathbf{x} + w_0\right)$$

$$y(\mathbf{x}) \geq 0 \rightarrow \mathbf{x} \text{ assigned to } C_1$$
$$y(\mathbf{x}) < 0 \rightarrow \mathbf{x} \text{ assigned to } C_2$$

- A classifier based upon this simple generalized linear model is called a (single layer) **perceptron**.
- It can also be identified with an abstracted model of a neuron called the McCulloch Pitts model.



End of Lecture

Oct 15, 2012

Parameter Learning

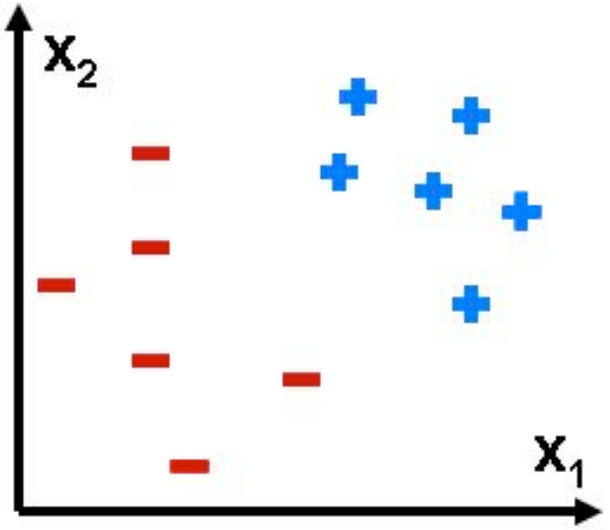
- How do we learn the parameters of a perceptron?

Outline

- **The Perceptron Algorithm**
- Least-Squares Classifiers
- Fisher's Linear Discriminant
- Logistic Classifiers

Case 1. Linearly Separable Inputs

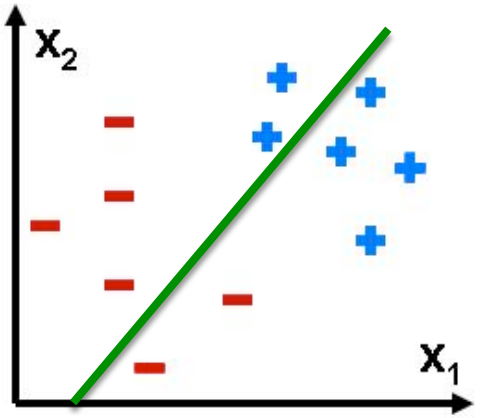
- For starters, let's assume that the training data is in fact perfectly linearly separable.
- In other words, there exists at least one hyperplane (one set of weights) that yields 0 classification error.
- We seek an algorithm that can automatically find such a hyperplane.



The Perceptron Algorithm

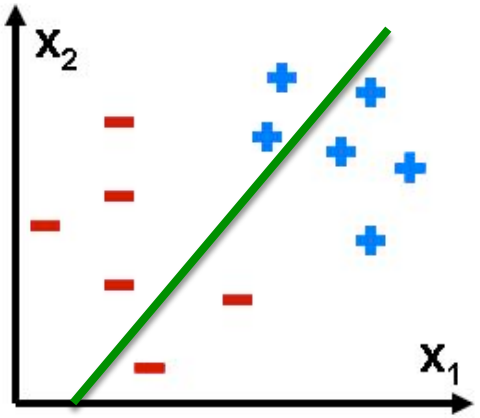
- The perceptron algorithm was invented by Frank Rosenblatt (1962).
- The algorithm is iterative.
- The strategy is to start with a random **guess** at the weights **w**, and to then iteratively change the weights to move the hyperplane in a direction that lowers the classification error.

Frank Rosenblatt (1928 – 1971)



The Perceptron Algorithm

- Note that as we change the weights continuously, the classification error changes in discontinuous, piecewise constant fashion.
- Thus we cannot use the classification error per se as our objective function to minimize.
- What would be a better objective function?



The Perceptron Criterion

- Note that we seek **w** such that

$\mathbf{w}^t\mathbf{x} \geq 0$ when $t = +1$

$\mathbf{w}^t\mathbf{x} < 0$ when $t = -1$

- In other words, we would like

$\mathbf{w}^t\mathbf{x}_n t_n \geq 0 \ \forall n$

- Thus we seek to minimize

$$E_P(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^t\mathbf{x}_n t_n$$

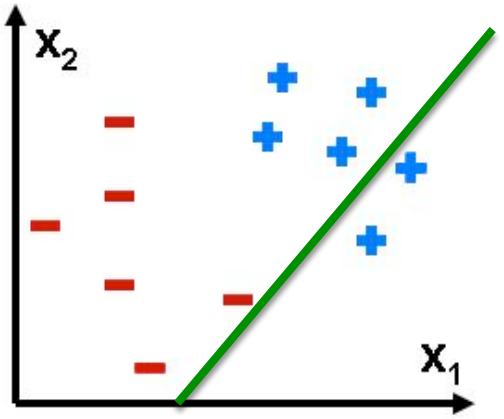
where $\mathcal{M}$ is the set of misclassified inputs.

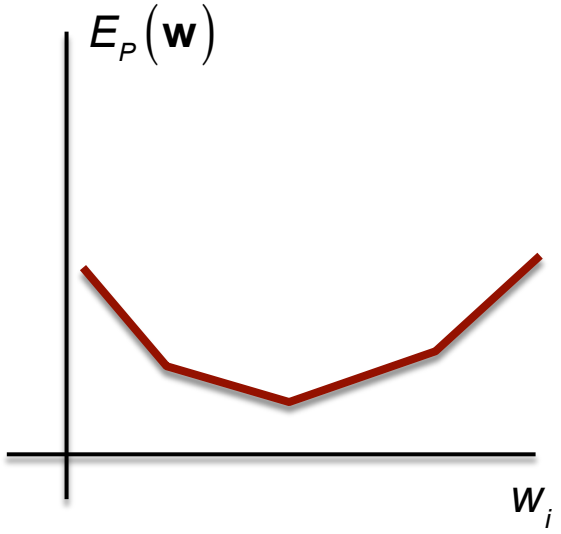
The Perceptron Criterion

$$E_P(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^t \mathbf{x}_n t_n$$

where $\mathcal{M}$ is the set of misclassified inputs.

- Observations:
  - $E_P(\mathbf{w})$ is always non-negative.
  - $E_P(\mathbf{w})$ is continuous and piecewise linear, and thus easier to minimize.





The Perceptron Algorithm

$$E_P(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^t \mathbf{x}_n t_n$$

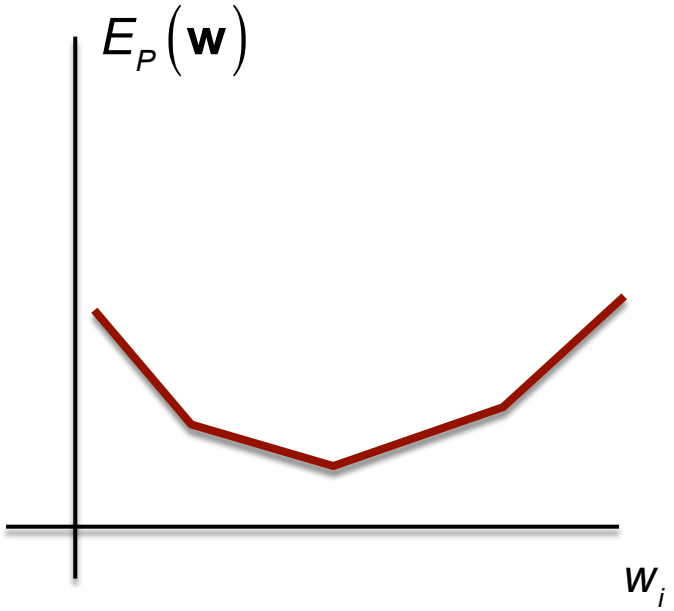
where $\mathcal{M}$ is the set of misclassified inputs.

$$\frac{dE_P(\mathbf{w})}{d\mathbf{w}} = -\sum_{n \in \mathcal{M}} \mathbf{x}_n t_n$$

where the derivative exists.

- Gradient descent:

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \nabla E_P(\mathbf{w}) = \mathbf{w}^{\tau} + \eta \sum_{n \in \mathcal{M}} \mathbf{x}_n t_n$$



The Perceptron Algorithm

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \nabla E_P(\mathbf{w}) = \mathbf{w}^{t} + \eta \sum_{n \in \mathcal{M}} \mathbf{x}_n t_n$$

- Why does this make sense?
  - If an input from $C_1 (t = +1)$ is misclassified, we need to make its projection on **w** more positive.
  - If an input from C_2 $(t = -1)$ is misclassified, we need to make its projection on **w** more negative.

The Perceptron Algorithm

- The algorithm can be implemented sequentially:
  - Repeat until convergence:
    - For each input ($\mathbf{x}_n$, t_n):
      - If it is correctly classified, do nothing
      - If it is misclassified, update the weight vector to be

        $$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \eta \mathbf{x}_n t_n$$

      - Note that this will lower the contribution of input n to the objective function:

        $$-\left(\mathbf{w}^{(\tau)}\right)^t \mathbf{x}_n t_n \rightarrow -\left(\mathbf{w}^{(\tau+1)}\right)^t \mathbf{x}_n t_n = -\left(\mathbf{w}^{(\tau)}\right)^t \mathbf{x}_n t_n - \eta \left(\mathbf{x}_n t_n\right)^t \mathbf{x}_n t_n < -\left(\mathbf{w}^{(\tau)}\right)^t \mathbf{x}_n t_n.$$

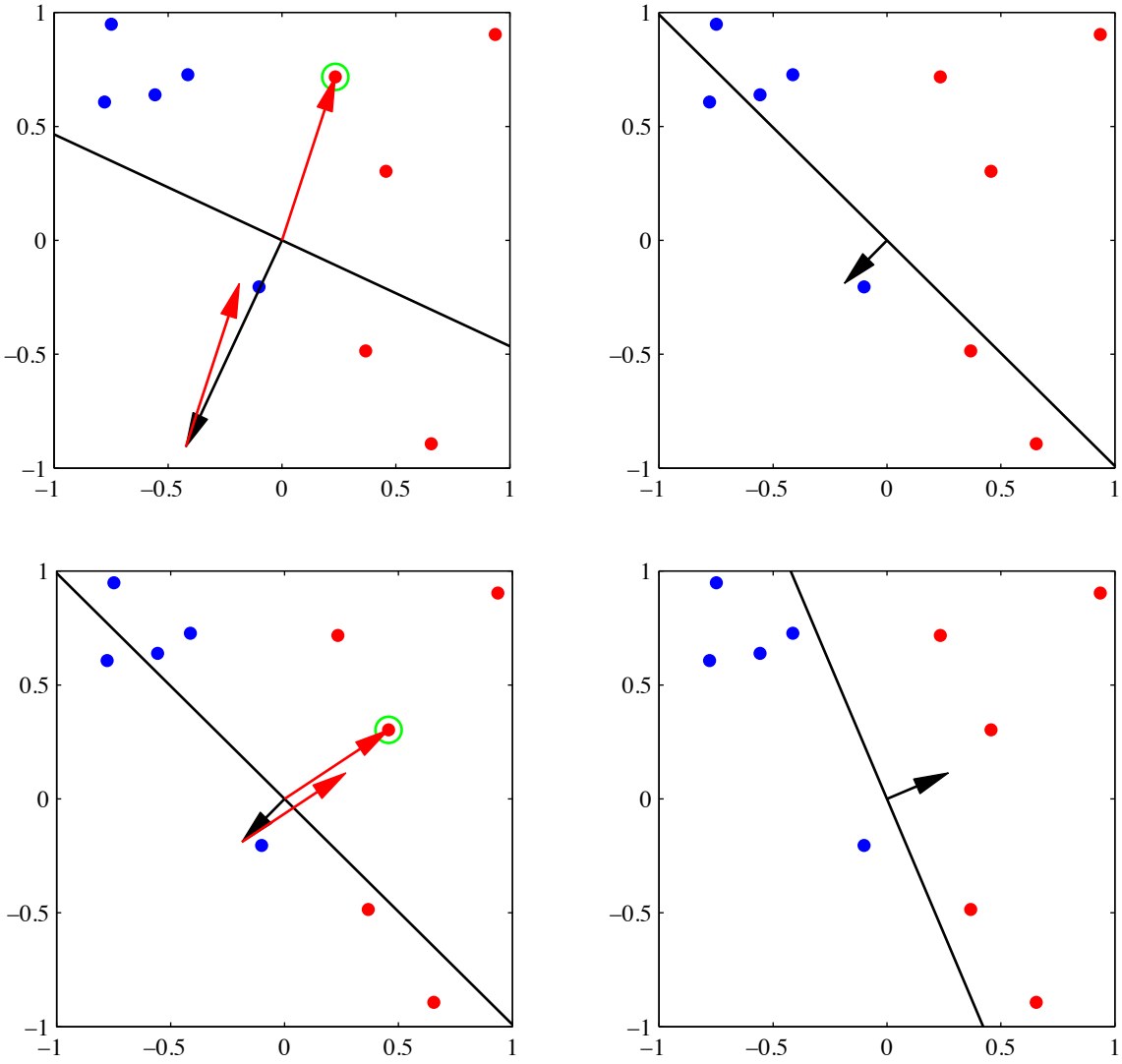
Not Monotonic

- While updating with respect to a misclassified input n will lower the error for that input, the error for other misclassified inputs may increase.
- Also, new inputs that had been classified correctly may now be misclassified.
- The result is that the perceptron algorithm is not guaranteed to reduce the total error monotonically at each stage.

The Perceptron Convergence Theorem

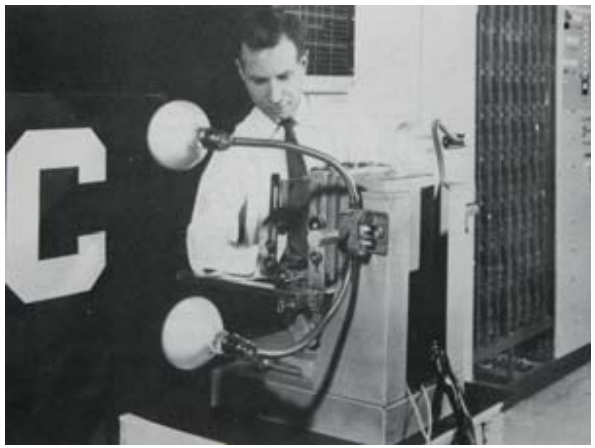
- Despite this non-monotonicity, **if in fact the data are linearly separable, then the algorithm is guaranteed to find an exact solution in a finite number of steps** (Rosenblatt, 1962).

Example

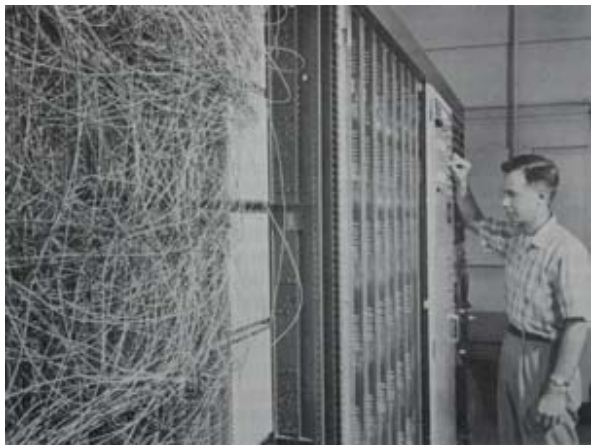


The First Learning Machine

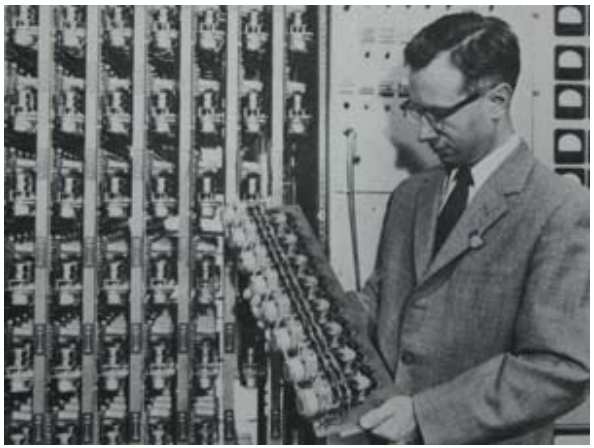
- Mark 1 Perceptron Hardware (c. 1960)



Visual Inputs



Patch board allowing configuration of inputs ϕ



Rack of adaptive weights **w** (motor-driven potentiometers)

Practical Limitations

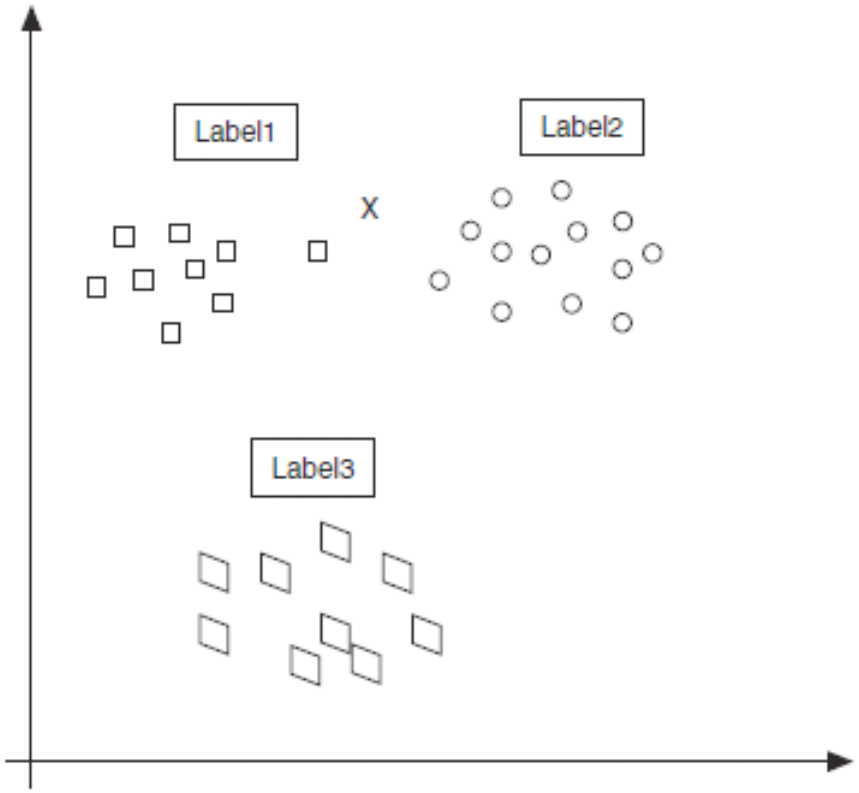
- The Perceptron Convergence Theorem is an important result. However, there are practical limitations:
  - Convergence may be slow
  - If the data are not separable, the algorithm will not converge.
  - We will only know that the data are separable once the algorithm converges.
  - The solution is in general not unique, and will depend upon initialization, scheduling of input vectors, and the learning rate η.

Generalization to inputs that are not linearly separable.

- The single-layer perceptron can be generalized to yield good linear solutions to problems that are not linearly separable.
- Example: The Pocket Algorithm (Gal 1990)
  - Idea:
    - Run the perceptron algorithm
    - Keep track of the weight vector $\mathbf{w}^*$ that has produced the best classification error achieved so far.
    - It can be shown that $\mathbf{w}^*$ will converge to an optimal solution with probability 1.

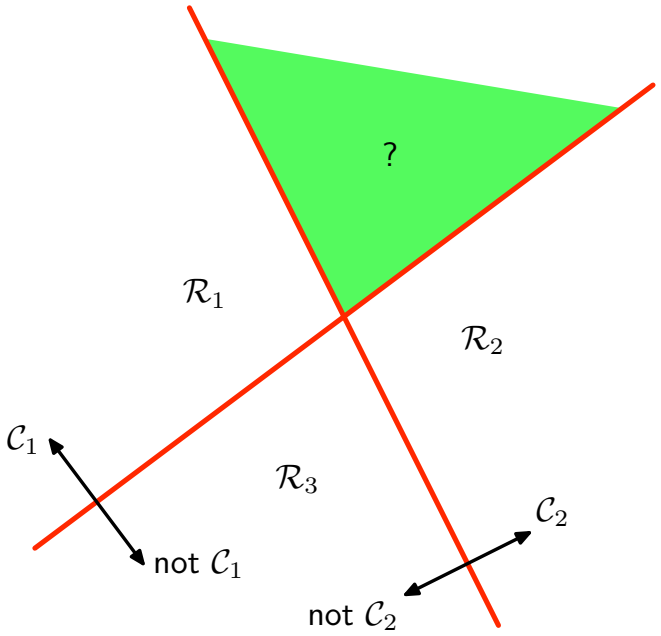
Generalization to Multiclass Problems

- How can we use perceptrons, or linear classifiers in general, to classify inputs when there are $K > 2$ classes?



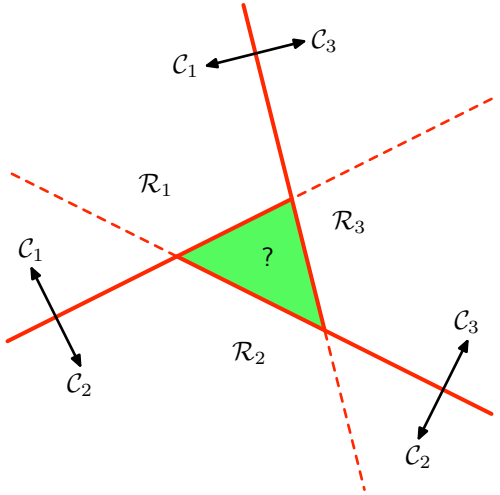
K>2 Classes

- Idea #1: Just use *K-1* discriminant functions, each of which separates one class C_k from the rest. (One-versus-the-rest classifier.)
- Problem: Ambiguous regions



K>2 Classes

- Idea #2: Use K(*K-1*)/*2* discriminant functions, each of which separates two classes C_i, C_k from each other. (One-versus-one classifier.)
- Each point classified by majority vote.
- Problem: Ambiguous regions



K>2 Classes

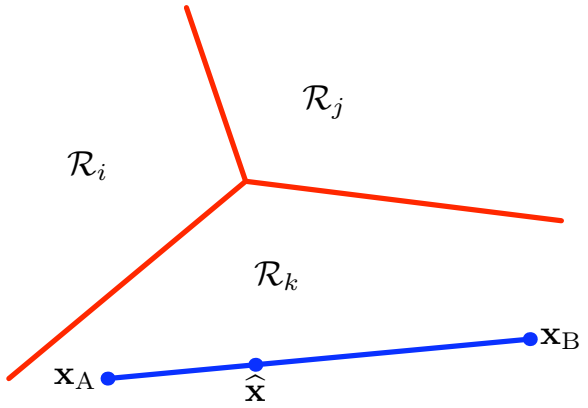
- Idea #3: Use K discriminant functions $y_k(x)$
- Use the **magnitude** of $y_k(x)$, not just the sign.

$$y_k(\mathbf{x}) = \mathbf{w}_k^t \mathbf{x}$$

$\mathbf{x}$ assigned to C_k if $y_k(\mathbf{x}) > y_j(\mathbf{x}) \forall j \neq k$

Decision boundary $y_k(\mathbf{x}) = y_j(\mathbf{x}) \rightarrow \left(w_k - w_j\right)^t x + \left(w_{k0} - w_{j0}\right) = 0$

Results in decision regions that are simply-connected and convex.



Example: Kesler's Construction

- The perceptron algorithm can be generalized to *K*-class classification problems.
- Example:
  - Kesler's Construction:
    - Allows use of the perceptron algorithm to simultaneously learn *K* separate weight vectors $\mathbf{w}_i$.
    - Inputs are then classified in Class *i* if and only if

      $$\mathbf{w}_i^t \mathbf{x} > \mathbf{w}_j^t \mathbf{x} \quad \forall j \neq i$$

    - The algorithm will converge to an optimal solution if a solution exists, i.e., if all training vectors can be correctly classified according to this rule.

1-of-K Coding Scheme

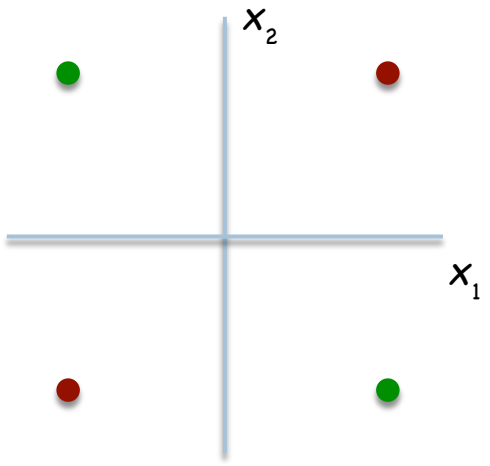
- When there are K>2 classes, target variables can be coded using the 1-of-K coding scheme:

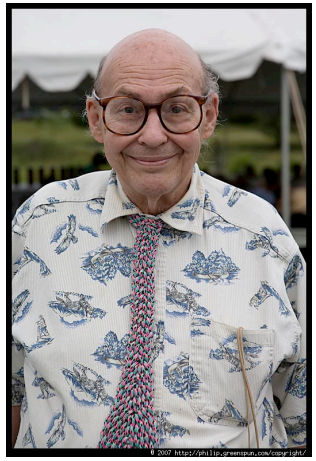
$$\text{Input from Class } C_i \Leftrightarrow t = [0\ 0\ \ldots 1 \ldots 0\ 0]^t$$

Element *i*

Computational Limitations of Perceptrons

- Initially, the perceptron was thought to be a potentially powerful learning machine that could model human neural processing.
- However, Minsky & Papert (1969) showed that the single-layer perceptron could not learn a simple XOR function.
- This is just one example of a non-linearly separable pattern that cannot be learned by a single-layer perceptron.

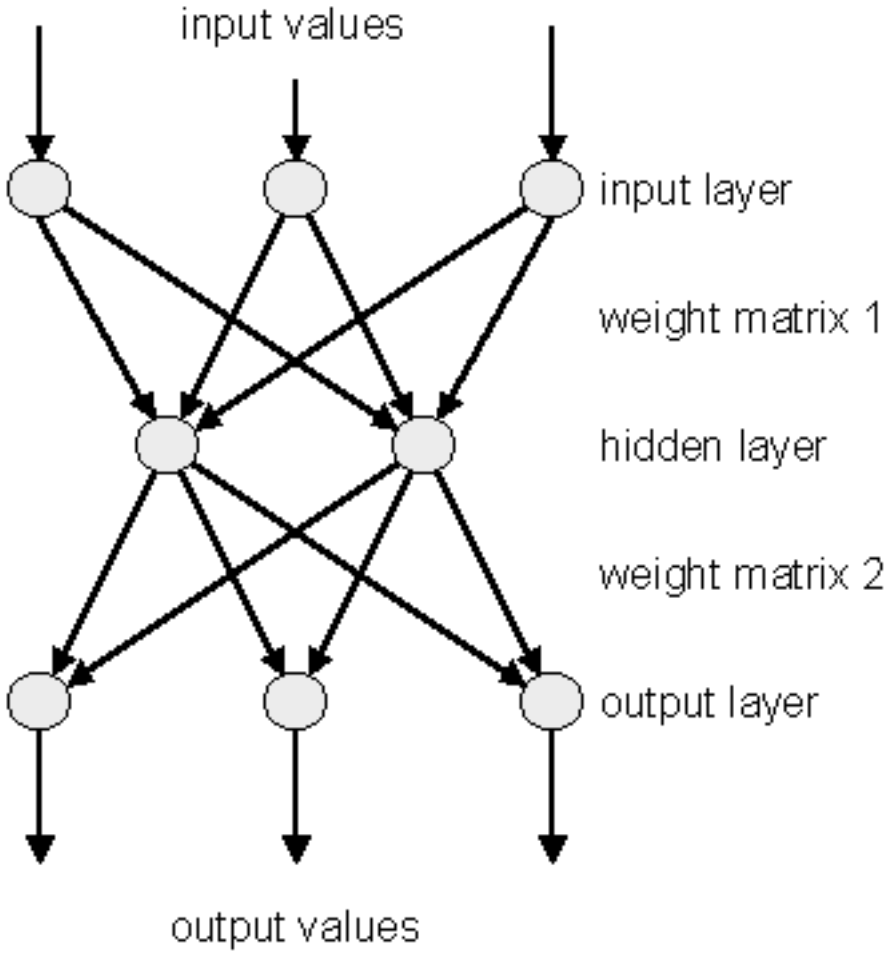




Marvin Minsky (1927 -)

Multi-Layer Perceptrons

- Minsky & Papert's book was widely misinterpreted as showing that artificial neural networks were inherently limited.
- This contributed to a decline in the reputation of neural network research through the 70s and 80s.
- However, their findings apply only to single-layer perceptrons. Multi-layer perceptrons are capable of learning highly nonlinear functions, and are used in many practical applications.



Outline

- The Perceptron Algorithm
- **Least-Squares Classifiers**
- Fisher's Linear Discriminant
- Logistic Classifiers

Dealing with Non-Linearly Separable Inputs

- The perceptron algorithm fails when the training data are not perfectly linearly separable.
- Let's now turn to methods for learning the parameter vector **w** of a perceptron (linear classifier) even when the training data are not linearly separable.

The Least Squares Method

- In the least squares method, we simply fit the $(\mathbf{x}, t)$ observations with a hyperplane $y(\mathbf{x})$.
- Note that this is kind of a weird idea, since the t values are binary (when $K=2$), e.g., 0 or 1.
- However it can work pretty well.

Least Squares: Learning the Parameters

Assume D – dimensional input vectors $\mathbf{x}$.

For each class $k \in 1 \ldots K$:

$$y_k(\mathbf{x}) = \mathbf{w}_k^t \mathbf{x} + w_{k0}$$

$$\rightarrow \mathbf{y}(\mathbf{x}) = \tilde{\mathbf{W}}^t \tilde{\mathbf{x}}$$

where

$$\tilde{\mathbf{x}} = (1, \mathbf{x}^t)^t$$

$\tilde{\mathbf{W}}$ is a $(D+1) \times K$ matrix whose kth column is $\tilde{\mathbf{w}}_k = \left(w_0, \mathbf{w}_k^t\right)^t$

Learning the Parameters

Method #2: Least Squares

$$\mathbf{y}(\mathbf{x}) = \tilde{\mathbf{W}}^t \tilde{\mathbf{x}}$$

Training dataset $(\mathbf{x}_n, \mathbf{t}_n), \quad n = 1, \ldots, N$

where we use the 1-of-K coding scheme for $\mathbf{t}_n$

Let $\mathbf{T}$ be the $N \times K$ matrix whose n^{th} row is $\mathbf{t}_n^t$

Let $\tilde{\mathbf{X}}$ be the $N \times (D+1)$ matrix whose n^{th} row is $\tilde{\mathbf{x}}_n^t$

Let $\mathrm{R}_D(\tilde{\mathbf{W}}) = \tilde{\mathbf{X}}\tilde{\mathbf{W}} - \mathbf{T}$

Then we define the error as $\mathrm{E}_D(\tilde{\mathbf{W}}) = \frac{1}{2}\sum_{i,j} R_{ij}^2 = \frac{1}{2}\mathrm{Tr}\left\{R_D(\tilde{\mathbf{W}})^t R_D(\tilde{\mathbf{W}})\right\}$

Setting derivative wrt $\tilde{\mathbf{W}}$ to 0 yields:

$$\tilde{\mathbf{W}} = \left(\tilde{\mathbf{X}}^t\tilde{\mathbf{X}}\right)^{-1}\tilde{\mathbf{X}}^t\mathbf{T} = \tilde{\mathbf{X}}^\dagger\mathbf{T}$$

Recall: $\frac{\partial}{\partial A}\mathrm{Tr}(AB) = B^t.$

Outline

- The Perceptron Algorithm
- Least-Squares Classifiers
- **Fisher's Linear Discriminant**
- Logistic Classifiers

Fisher's Linear Discriminant

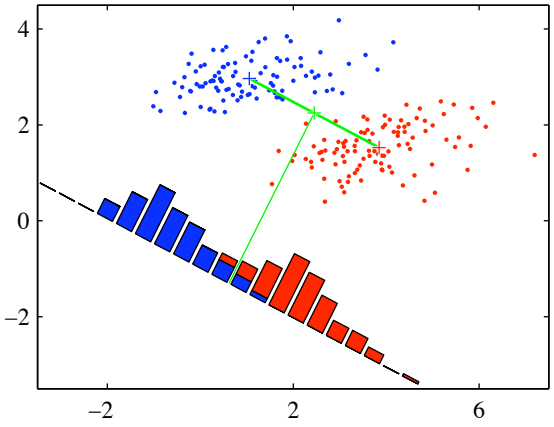
- Another way to view linear discriminants: find the 1D subspace that maximizes the separation between the two classes.

Let $\mathbf{m}_1 = \frac{1}{N_1}\sum_{n\in C_1}\mathbf{x}_n, \quad \mathbf{m}_2 = \frac{1}{N_2}\sum_{n\in C_2}\mathbf{x}_n$

For example, might choose $\mathbf{w}$ to maximize $\mathbf{w}^t(\mathbf{m}_2 - \mathbf{m}_1)$, subject to $\|\mathbf{w}\| = 1$

This leads to $\mathbf{w} \propto \mathbf{m}_2 - \mathbf{m}_1$

However, if conditional distributions are not isotropic, this is typically not optimal.

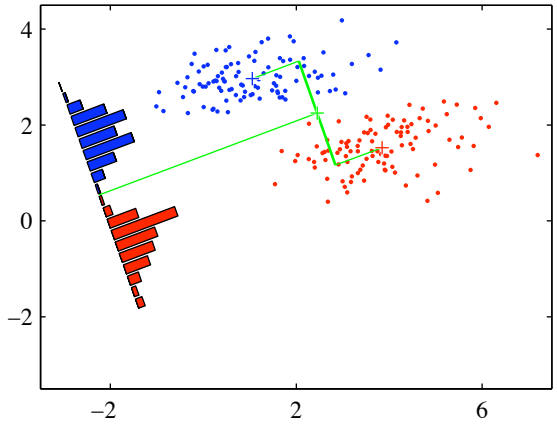


Fisher's Linear Discriminant

Let $m_1 = \mathbf{w}^t\mathbf{m}_1$, $m_2 = \mathbf{w}^t\mathbf{m}_2$ be the conditional means on the 1D subspace.

Let $s_k^2 = \sum_{n \in C_k} (y_n - m_k)^2$ be the within-class variance on the subspace for class C_k

The Fisher criterion is then $J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$



This can be rewritten as

$$J(\mathbf{w}) = \frac{\mathbf{w}^t\mathbf{S}_B\mathbf{w}}{\mathbf{w}^t\mathbf{S}_W\mathbf{w}}$$

where

$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^t$ is the between-class variance

and

$\mathbf{S}_W = \sum_{n \in C_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^t + \sum_{n \in C_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^t$ is the within-class variance

$J(\mathbf{w})$ is maximized for $\mathbf{w} \propto \mathbf{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$

Connection to MVN Maximum Likelihood

$J(\mathbf{w})$ is maximized for $\mathbf{w} \propto \mathbf{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$

- Recall that if the two distributions are normal with the same covariance Σ, the maximum likelihood classifier is linear, with

$$\mathbf{w} \propto \Sigma^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

- Further, note that S_W is proportional to the maximum likelihood estimator for Σ.
- Thus FLD is equivalent to assuming MVN distributions with common covariance.

Connection to Least-Squares

Change coding scheme used in least-squares method to

$$t_n = \frac{N}{N_1} \text{ for } C_1$$

$$t_n = -\frac{N}{N_2} \text{ for } C_2$$

Then one can show that the ML $\mathbf{w}$ satisfies

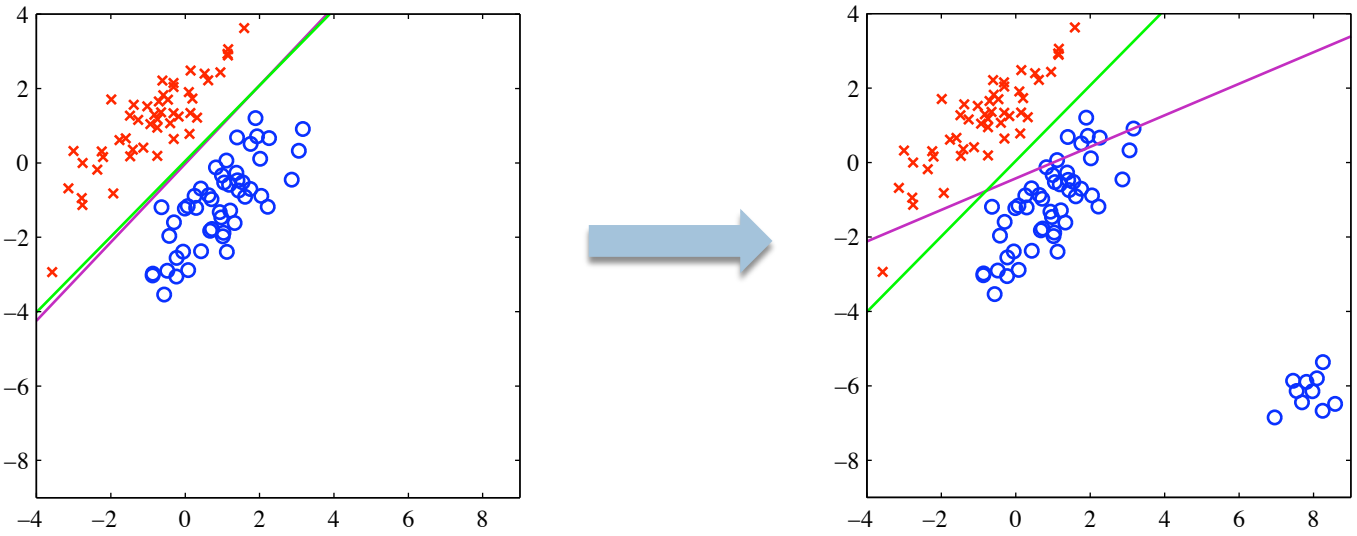
$$\mathbf{w} \propto \mathbf{S}_W^{-1}\left(\mathbf{m}_2 - \mathbf{m}_1\right)$$

End of Lecture

October 17, 2012

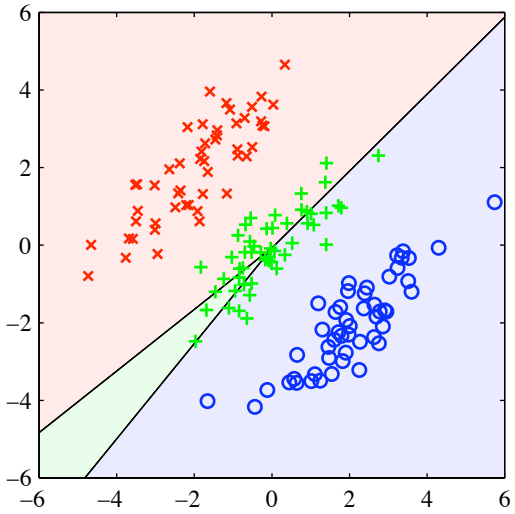
Problems with Least Squares

- Problem #1: Sensitivity to outliers



Problems with Least Squares

- Problem #2: Linear activation function is not a good fit to binary data. This can lead to problems.



Outline

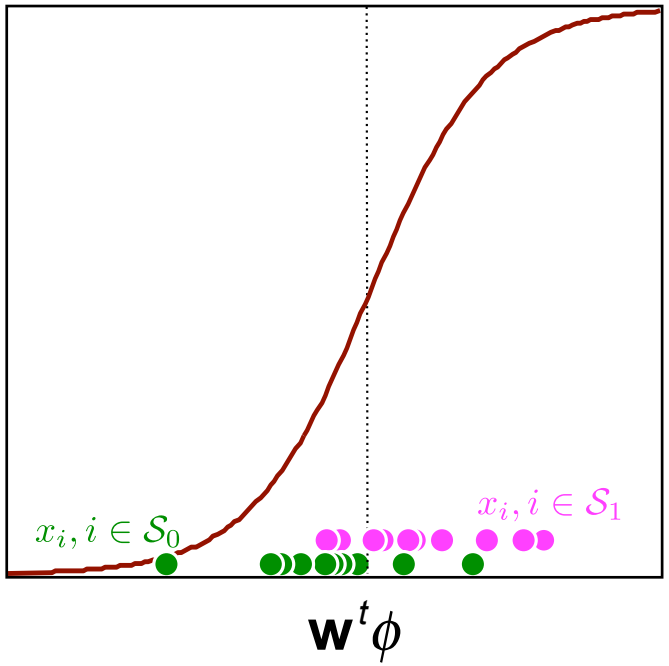
- The Perceptron Algorithm
- Least-Squares Classifiers
- Fisher's Linear Discriminant
- **Logistic Classifiers**

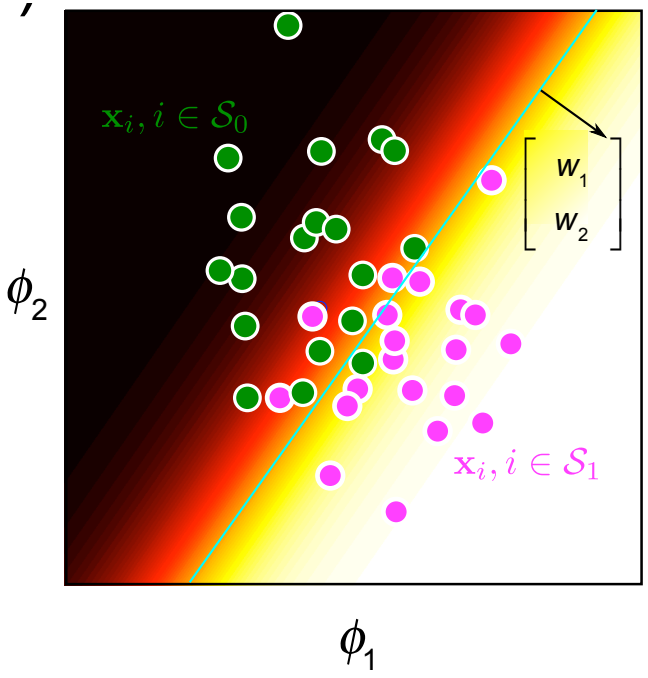
Logistic Regression ($K = 2$)

$$p(C_1|\phi) = y(\phi) = \sigma(\mathbf{w}^t\phi)$$
$$p(C_2|\phi) = 1 - p(C_1|\phi)$$

$$\text{where } \sigma(a) = \frac{1}{1+\exp(-a)}$$

$$p(C_1|\phi) = y(\phi) = \sigma(\mathbf{w}^t\phi)$$





Logistic Regression

$$p(C_1 \mid \phi) = y(\phi) = \sigma(\mathbf{w}^t \phi)$$
$$p(C_2 \mid \phi) = 1 - p(C_1 \mid \phi)$$

where

$$\sigma(a) = \frac{1}{1+\exp(-a)}$$

- Number of parameters
  - Logistic regression: M
  - Gaussian model: $2M + 2M(M+1)/2 + 1 = M^2+3M+1$

ML for Logistic Regression

$$p(\mathbf{t}\,|\,\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n}\left\{1-y_n\right\}^{1-t_n} \quad \text{where } \mathbf{t} = \left(t_1,\ldots,t_N\right)^t \text{ and } y_n = p\left(C_1\,|\,\phi_n\right)$$

We define the error function to be $E(\mathbf{w}) = -\log p\left(\mathbf{t}\,|\,\mathbf{w}\right)$

Given $y_n = \sigma\left(a_n\right)$ and $a_n = \mathbf{w}^t\phi_n$, one can show that

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}\left(y_n - t_n\right)\phi_n$$

Unfortunately, there is no closed form solution for **w**.

ML for Logistic Regression:

- Iterative Reweighted Least Squares
  - Although there is no closed form solution for the ML estimate of **w**, fortunately, the error function is convex.
  - Thus an appropriate iterative method is guaranteed to find the exact solution.
  - A good method is to use a local quadratic approximation to the log likelihood function (Newton-Raphson update):

$$\mathbf{w}^{(new)} = \mathbf{w}^{(old)} - \mathbf{H}^{-1}\nabla E(\mathbf{w})$$

where **H** is the Hessian matrix of $E(\mathbf{w})$

The Hessian Matrix H

$$H = \nabla_{\mathbf{w}} \nabla_{\mathbf{w}} E(\mathbf{w}), \text{ i.e., } H_{ij} = \frac{\partial E(\mathbf{w})}{\partial w_i \partial w_j}.$$

- H_{ij} describes how the i^{th} component of the gradient varies as we move in the w_j direction.
- Let **u** be any unit vector. Then
  - **Hu** describes the variation in the gradient as we move in the direction **u**.
  - $\mathbf{u}^t\mathbf{H}\mathbf{u}$ describes the projection of this variation onto u.
  - Thus $\mathbf{u}^t\mathbf{H}\mathbf{u}$ measures how much the gradient is changing in the **u** direction as we move in the **u** direction.

ML for Logistic Regression

$$\mathbf{w}^{(new)} = \mathbf{w}^{(old)} - \mathbf{H}^{-1}\nabla E(\mathbf{w})$$

where $\mathbf{H}$ is the Hessian matrix of $E(\mathbf{w})$:

$$\mathbf{H} = \Phi^t \mathbf{R} \Phi$$

where $\mathbf{R}$ is the $N \times N$ diagonal weight matrix with $R_{nn} = y_n\left(1 - y_n\right)$

and Φ is the $N \times M$ design matrix whose n^{th} row is given by ϕ_n^t.

(Note that, since $\mathbf{R}_{nn} \geq 0$, $\mathbf{R}$ is positive semi-definite, and hence $\mathbf{H}$ is positive semi-definite Thus $E(\mathbf{w})$ is convex.)

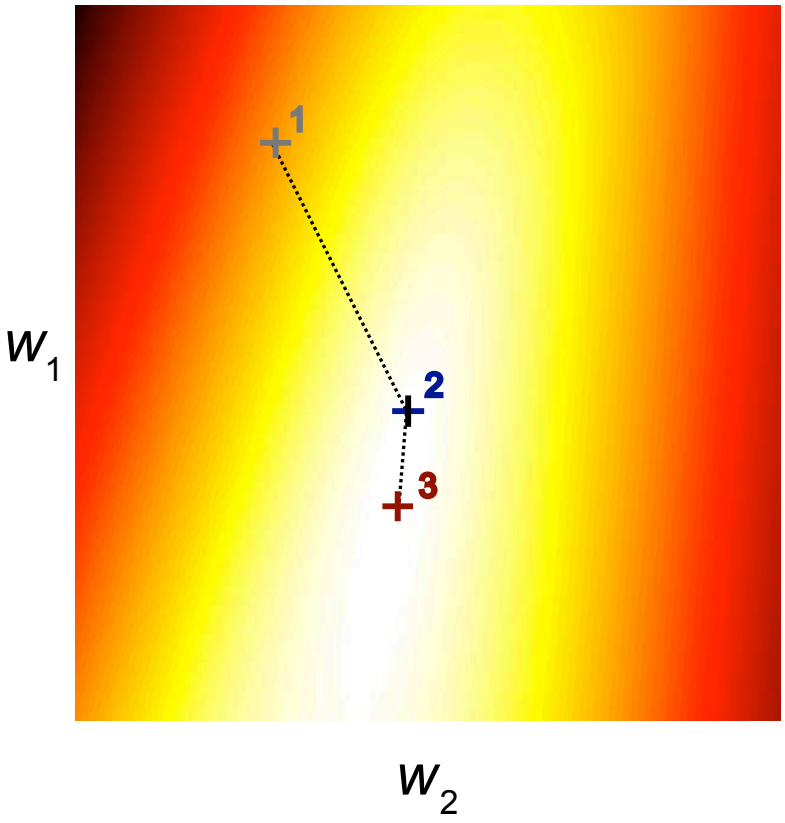
Thus

$$\mathbf{w}^{new} = \mathbf{w}^{(old)} - \left(\Phi^t \mathbf{R} \Phi\right)^{-1} \Phi^t \left(\mathbf{y} - \mathbf{t}\right)$$

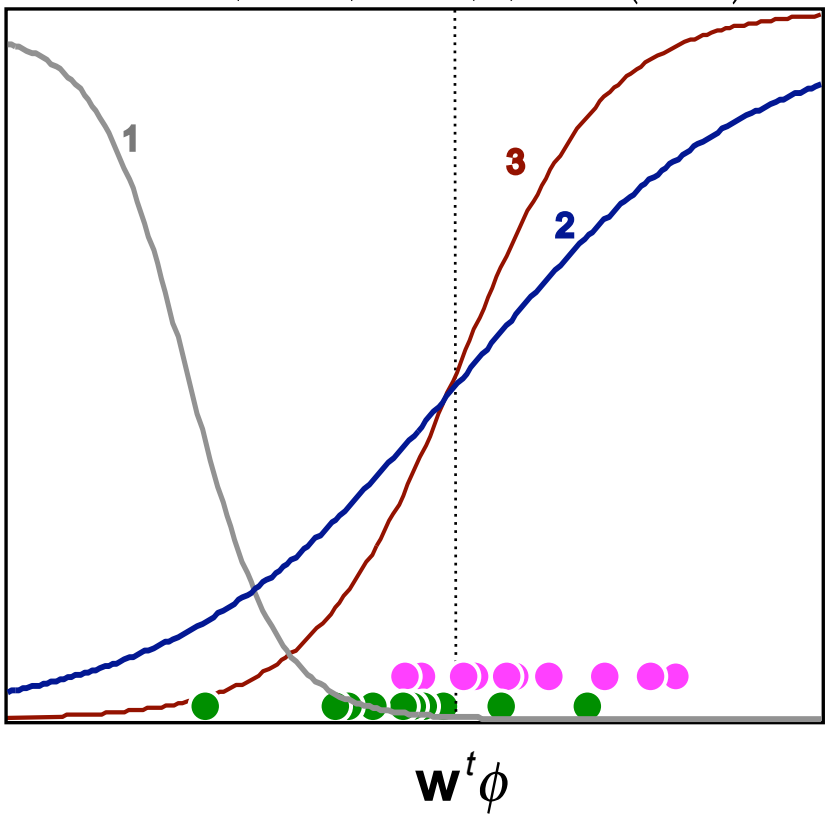
See Problem 8.3 in the text!

ML for Logistic Regression

- Iterative Reweighted Least Squares



$$p(C_1 \mid \phi) = y(\phi) = \sigma(\mathbf{w}^t \phi)$$



Logistic Regression

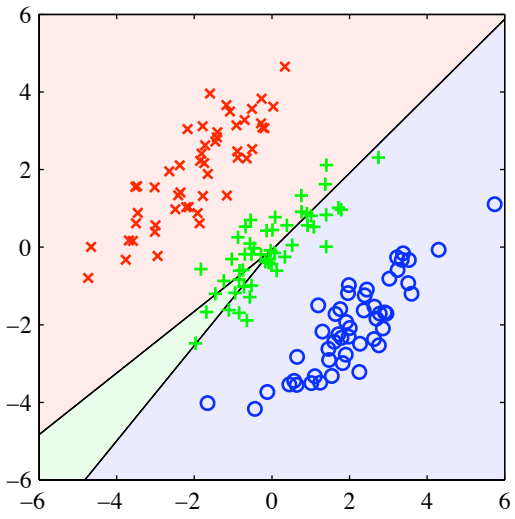
- For K>2, we can generalize the activation function by modeling the posterior probabilities as

$$p(C_k \mid \phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

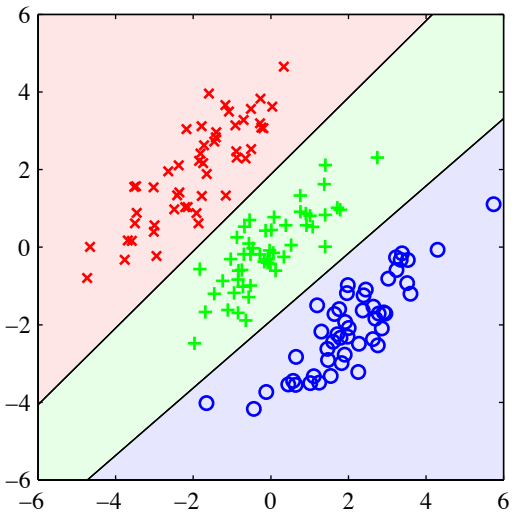
where the activations a_k are given by

$$a_k = \mathbf{w}_k^t \phi$$

Example



Least-Squares



Logistic

Outline

- The Perceptron Algorithm
- Least-Squares Classifiers
- Fisher's Linear Discriminant
- Logistic Classifiers