LINEAR CLASSIFIERS

J. Elder

Classification: Problem Statement

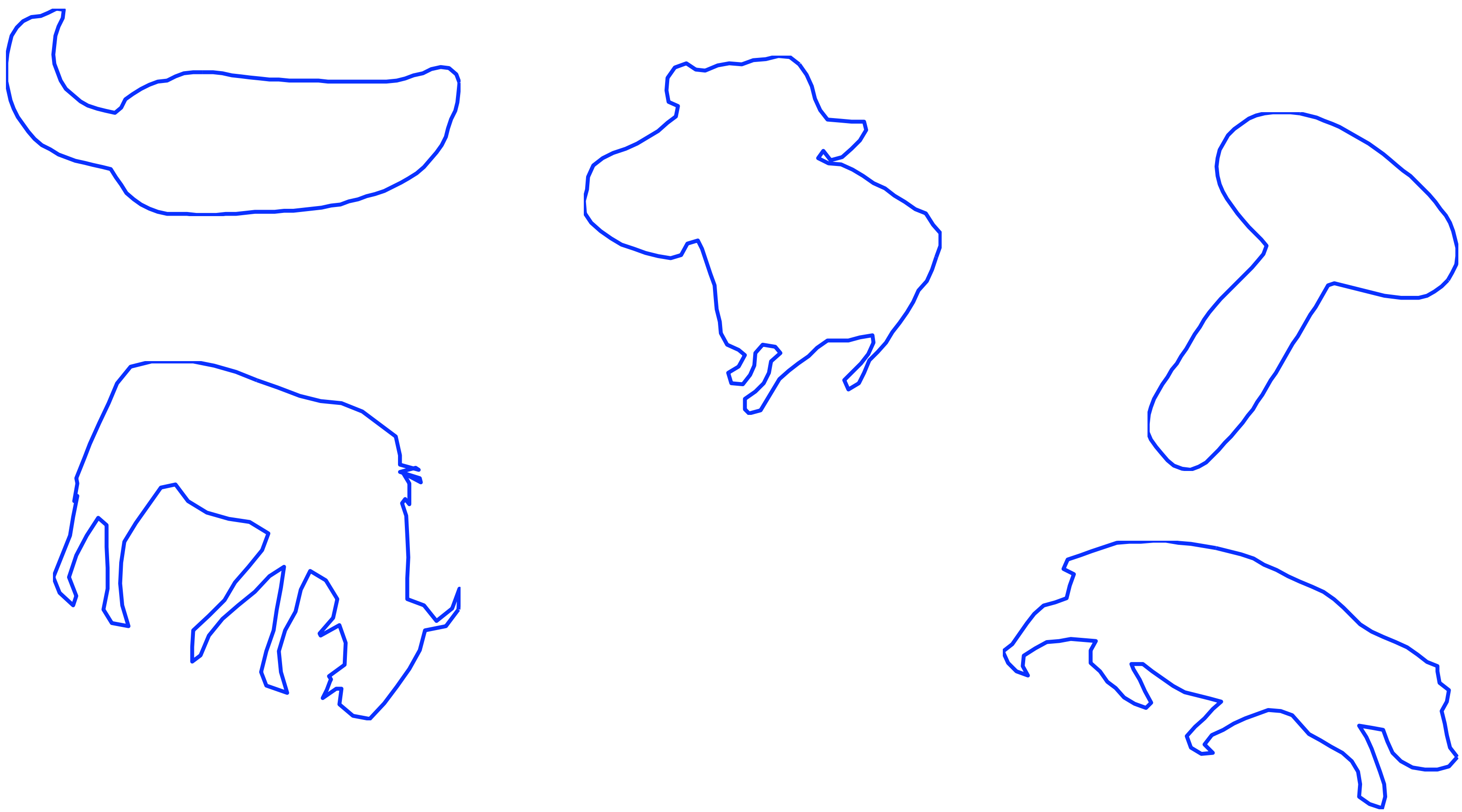
- In regression, we are modeling the relationship between a continuous input variable **x** and a continuous target variable *t*.
- In classification, the input variable **x** may still be continuous, but the target variable is discrete.
- In the simplest case, *t* can have only 2 values.

e.g., Let $t = +1$ assigned to C_1

$t = \ \ 1$ assigned to C_2

Example Problem

- Animal or Vegetable?

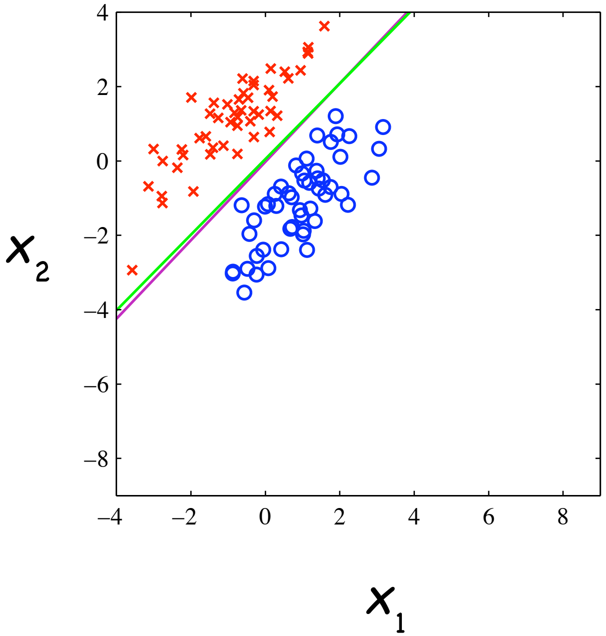


Linear Models for Classification

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

2D Input vector **x**

Two discrete classes C_1 and C_2



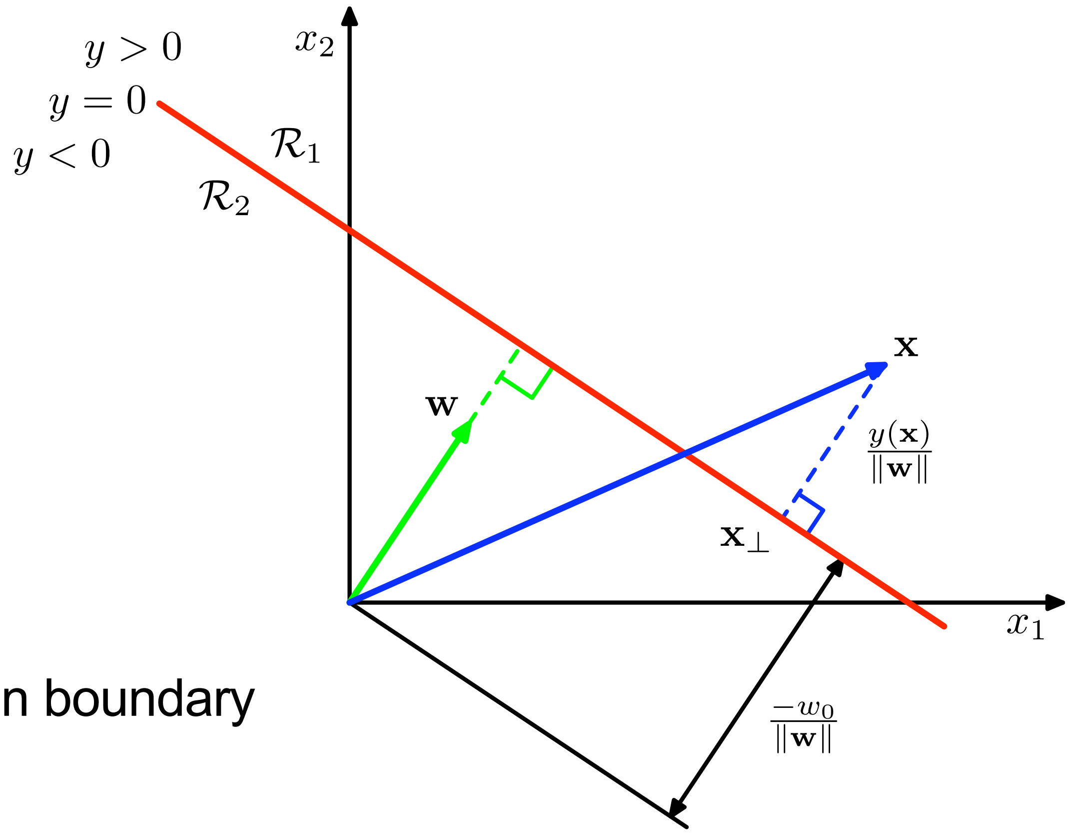
Two Class Discriminant Function

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

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

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

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

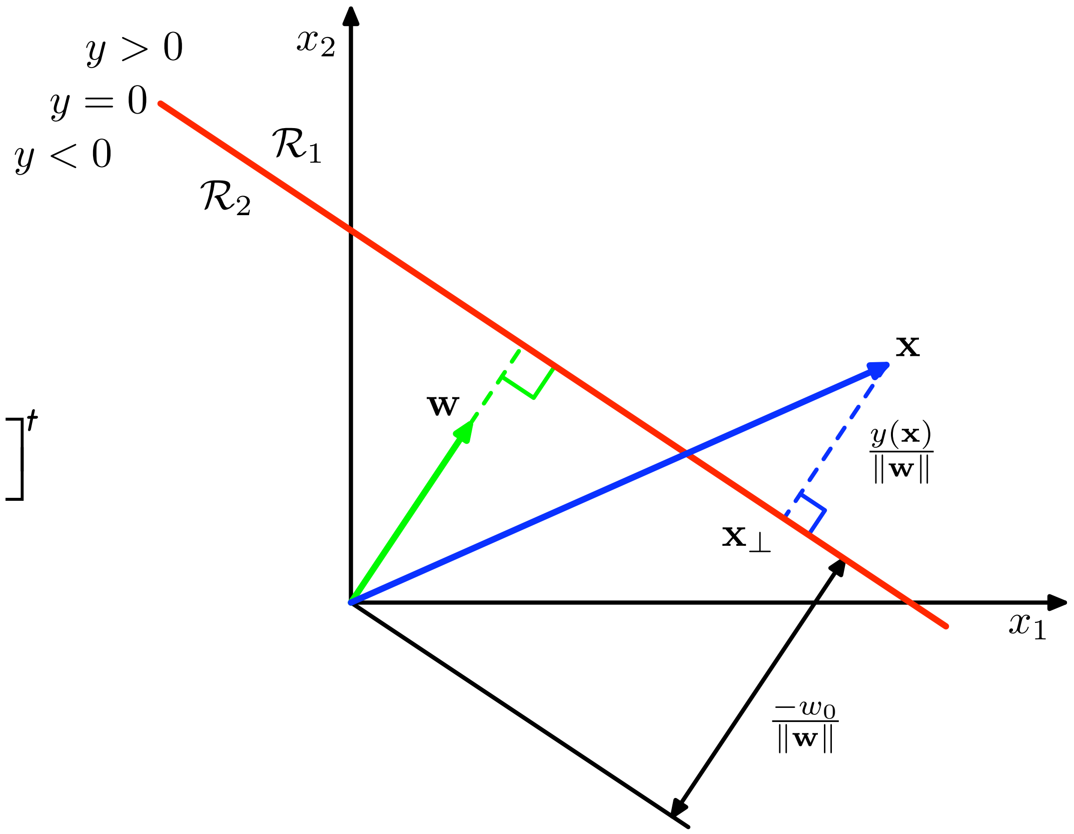


Two-Class Discriminant Function

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

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

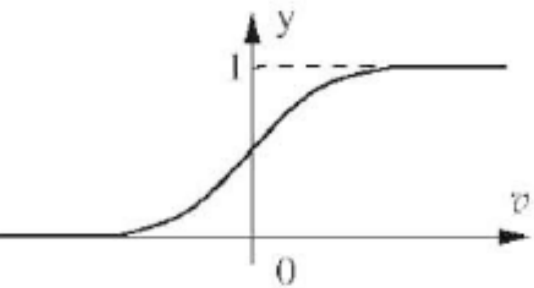
$$\mathbf{x} = \left[x_1 \ldots x_M\right]^t \Rightarrow \left[1 \;\; x_1 \ldots x_M\right]^t$$

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

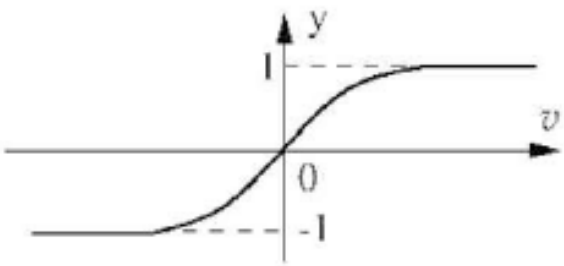
Generalized Linear Models

- 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.

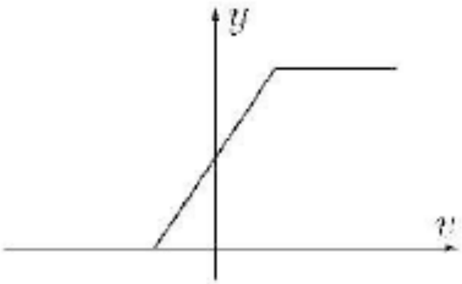
$$y(\mathbf{x}) = f\left(\mathbf{w}^t\mathbf{x} + w_0\right)$$



Log-sigmoid function



Tan-sigmoid function

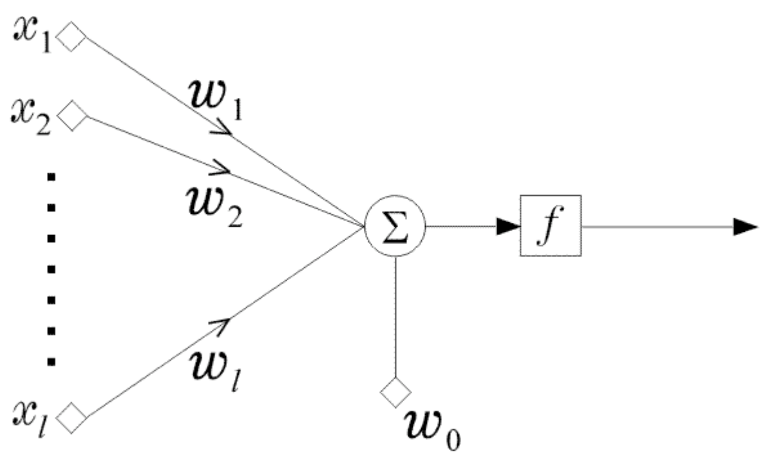


Linear function

The Perceptron

$$y(\mathbf{x}) = f\left(\mathbf{w}^t\mathbf{x} + w_0\right) \qquad \begin{array}{l} y(\mathbf{x}) \geq 0 \rightarrow \mathbf{x} \text{ assigned to } C_1 \\ y(\mathbf{x}) < 0 \rightarrow \mathbf{x} \text{ assigned to } C_2 \end{array}$$

- 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.



Parameter Learning

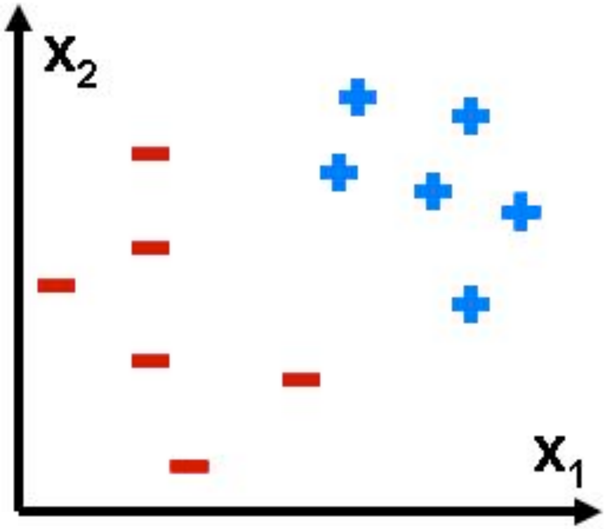
- How do we learn the parameters of a perceptron?

Outline

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

Case 1. Linearly Separable Inputs

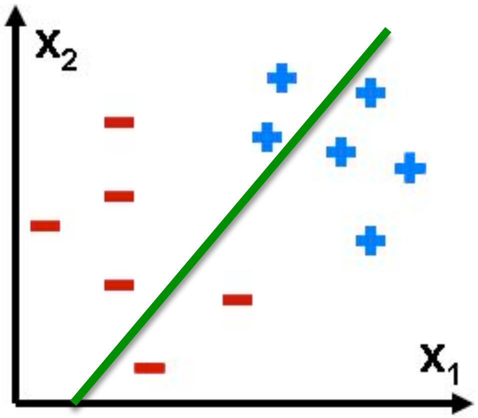
- For starters, let's assume that the training data is in fact perfectly linearly separable.
- In other words, there exists at least one hyperplane (one set of weights) that yields 0 classification error.
- We seek an algorithm that can automatically find such a hyperplane.



The Perceptron Algorithm

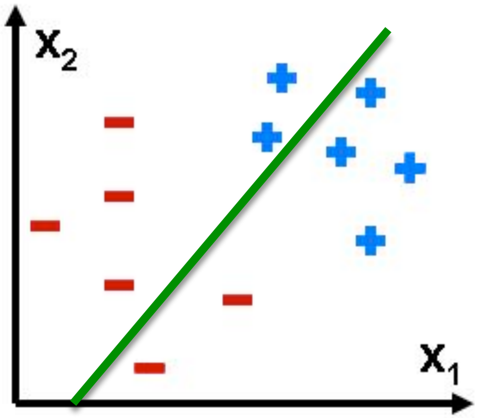
- The perceptron algorithm was invented by Frank Rosenblatt (1962).
- The algorithm is iterative.
- The strategy is to start with a random **guess** at the weights **w**, and to then iteratively change the weights to move the hyperplane in a direction that lowers the classification error.

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

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

- In other words, we would like

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

- Thus we seek to minimize

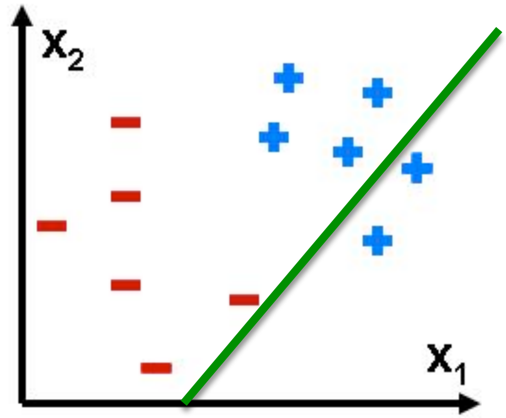
$$E_P(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^t\mathbf{x}_n t_n$$

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

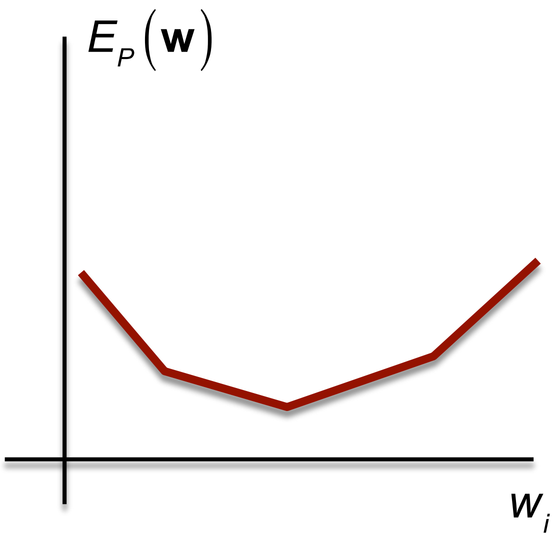
The Perceptron Criterion

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

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

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

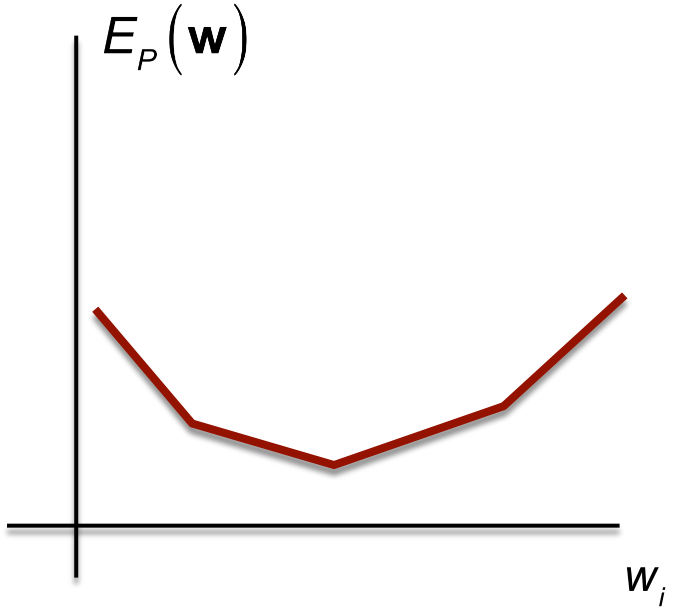
where $\mathcal{M}$ is the set of misclassified inputs.

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

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:

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

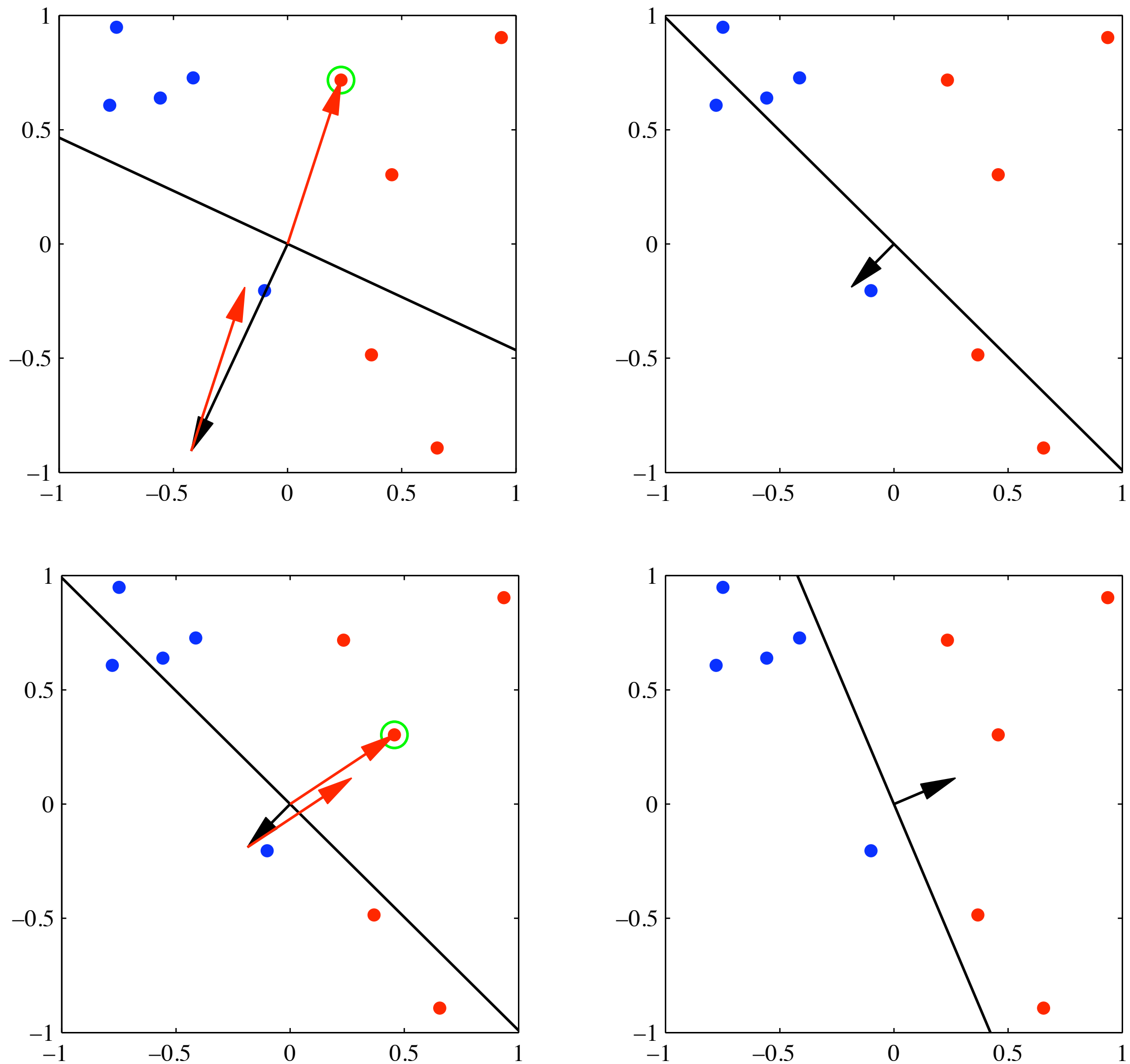
Not Monotonic

- 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

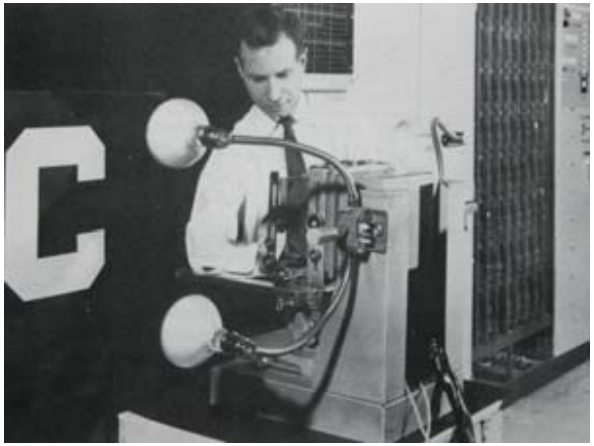
- Despite this non-monotonicity, **if in fact the data are linearly separable, then the algorithm is guaranteed to find an exact solution in a finite number of steps** (Rosenblatt, 1962).

Example

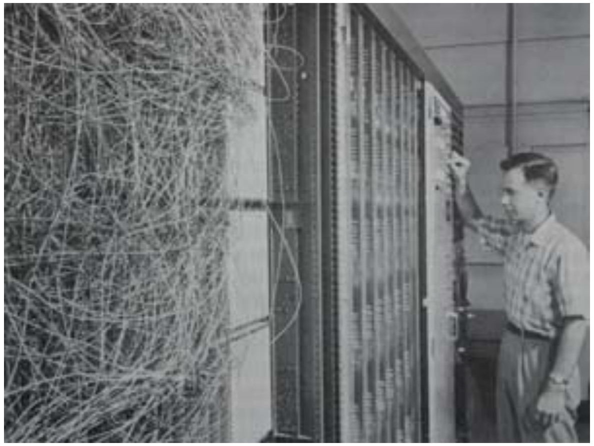

1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1
1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1
1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1
1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1

The First Learning Machine

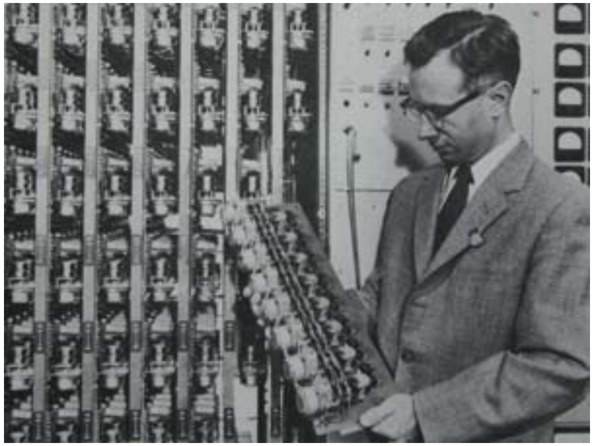
- Mark 1 Perceptron Hardware (c. 1960)



Visual Inputs



Patch board allowing configuration of inputs ϕ



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

Practical Limitations

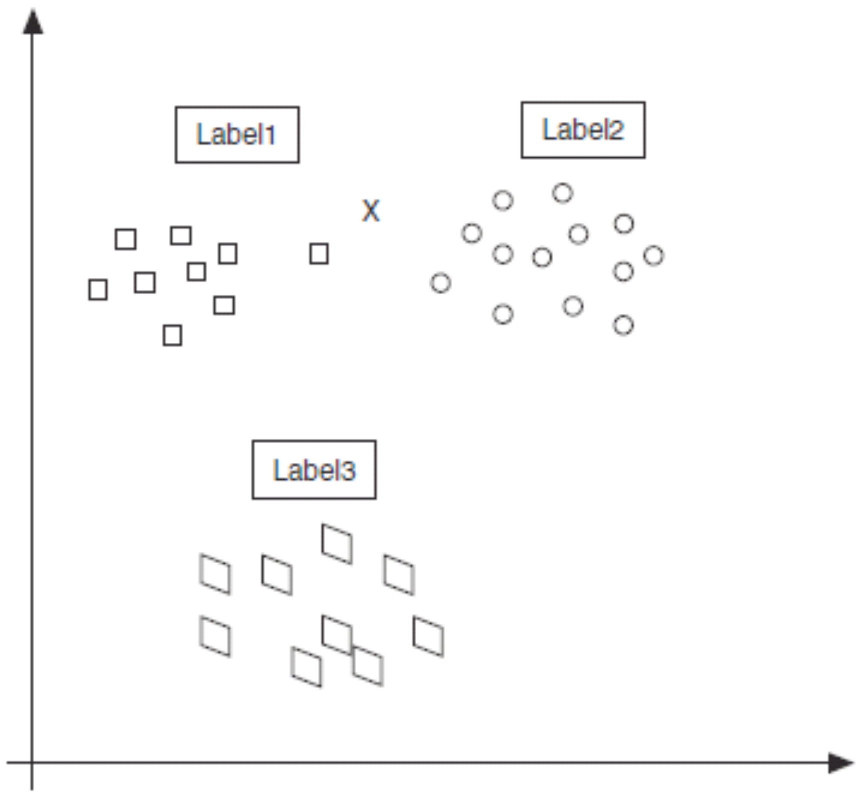
- The Perceptron Convergence Theorem is an important result. However, there are practical limitations:
  - Convergence may be slow
  - If the data are not separable, the algorithm will not converge.
  - We will only know that the data are separable once the algorithm converges.
  - The solution is in 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.

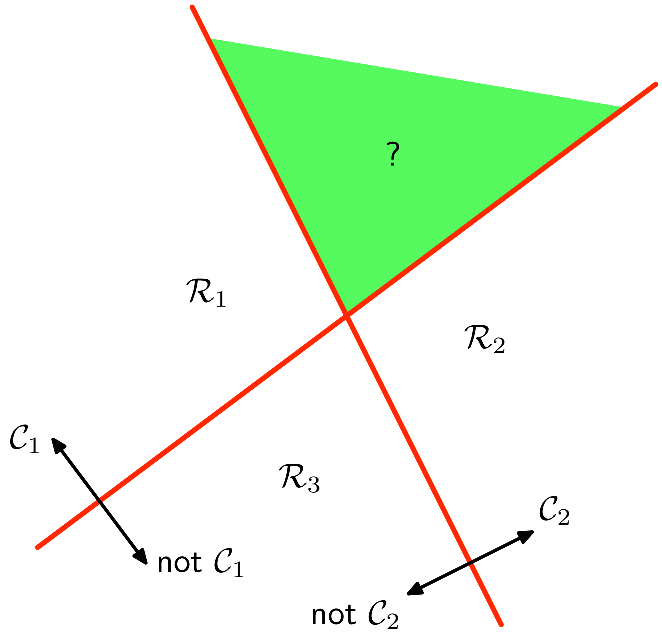
Generalization to Multiclass Problems

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



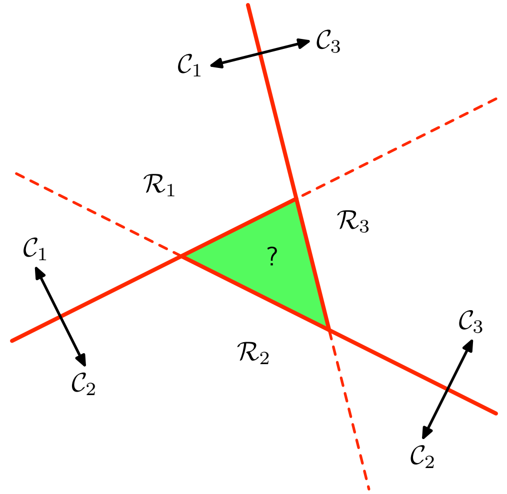
K>2 Classes

- 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_j, C_k from each other. (One-versus-one classifier.)
- Each point classified by majority vote.
- Problem: Ambiguous regions



K>2 Classes

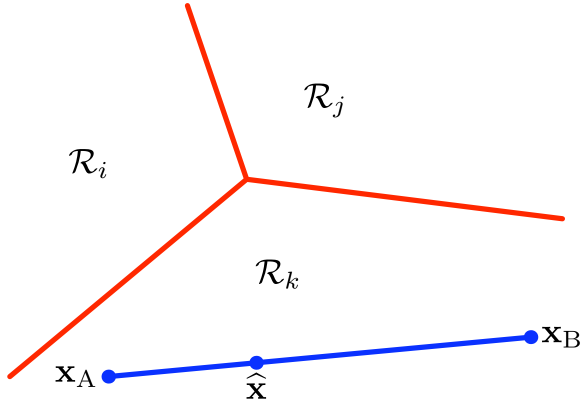
- Idea #3: Use K discriminant functions $y_k(x)$
- Use the **magnitude** of $y_k(x)$, not just the sign.

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

$\mathbf{x}$ 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

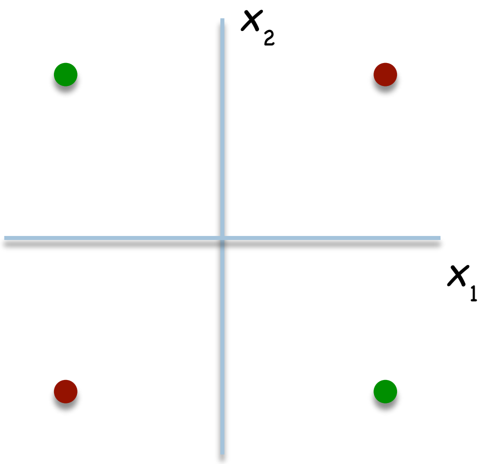
- When there are K>2 classes, target variables can be coded using the 1-of-K coding scheme:

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

↑ Element i

Computational Limitations of Perceptrons

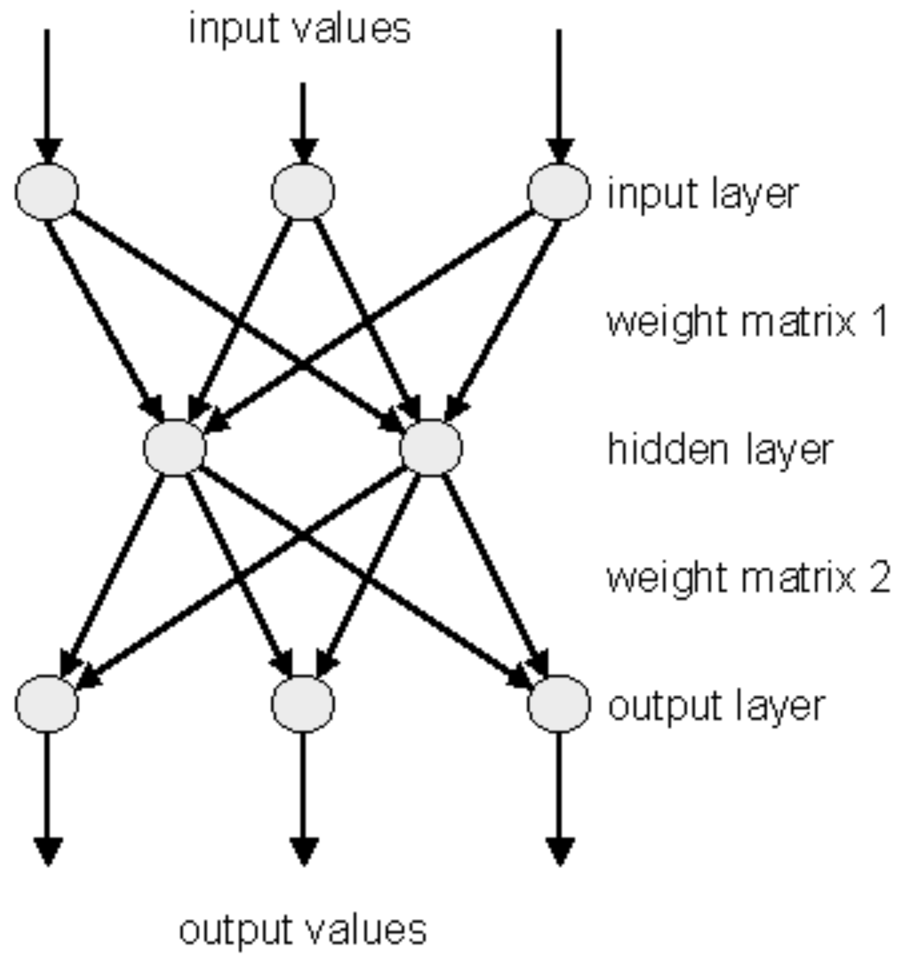
- Initially, the perceptron was thought to be a potentially powerful learning machine that could model human neural processing.
- However, Minsky & Papert (1969) showed that the single-layer perceptron could not learn a simple XOR function.
- This is just one example of a non-linearly separable pattern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



End of Lecture 11

Outline

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

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:

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

Outline

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

Fisher's Linear Discriminant

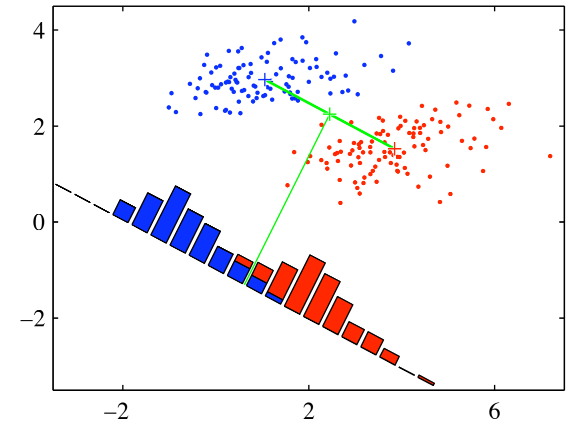
- Another way to view linear discriminants: find the 1D subspace that maximizes the separation between the two classes.

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

For example, might choose $\mathbf{w}$ to maximize $\mathbf{w}^t\left(\mathbf{m}_2 - \mathbf{m}_1\right)$, 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.

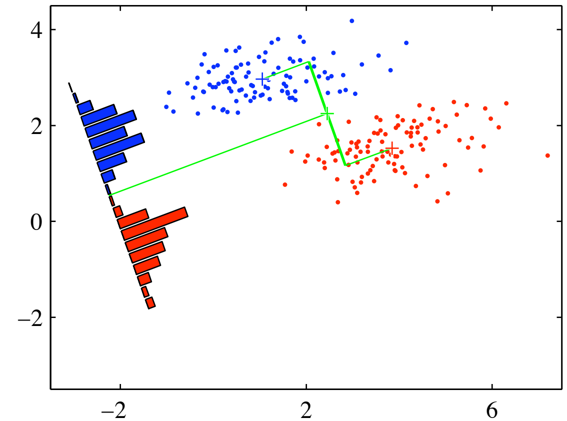


Fisher's Linear Discriminant

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

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



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

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

and

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

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

Connection between Least-Squares and FLD

Change coding scheme used in least-squares method to

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

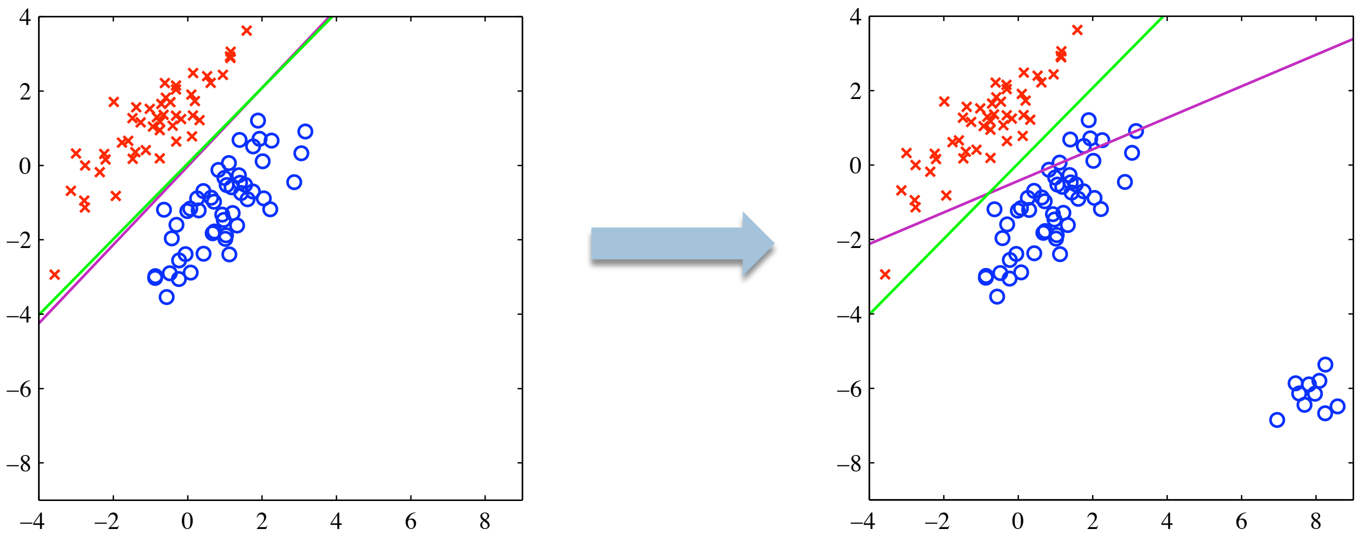
$$t_n = -\frac{N}{N_2} \text{ for } C_2$$

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

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

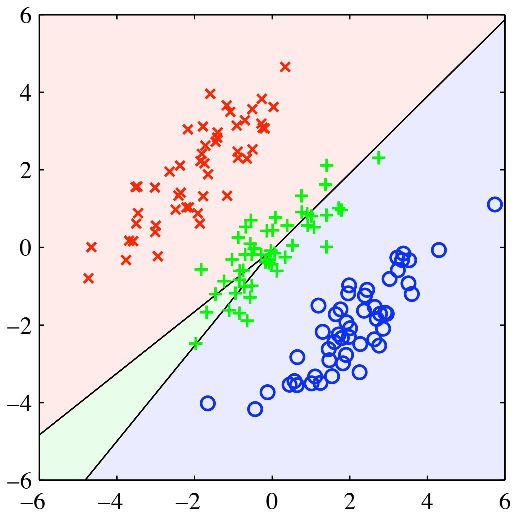
Least Squares Classifier

- Problem #1: Sensitivity to outliers



Least Squares Classifier

- 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**
- Support Vector Machines

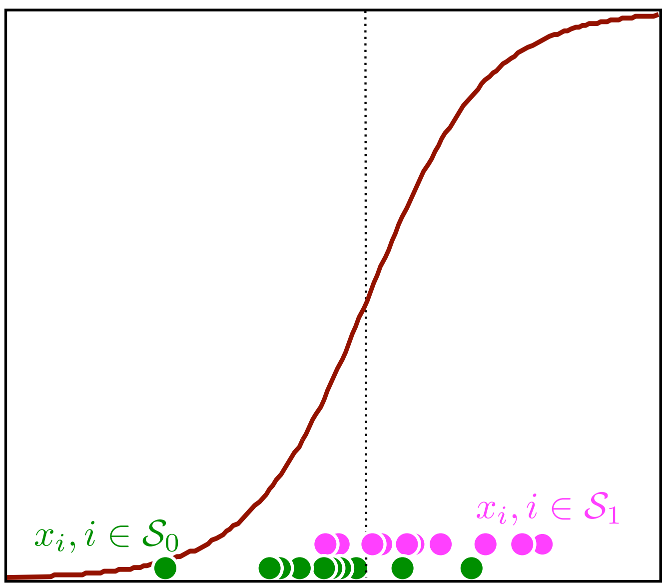
Logistic Regression ($K = 2$)

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

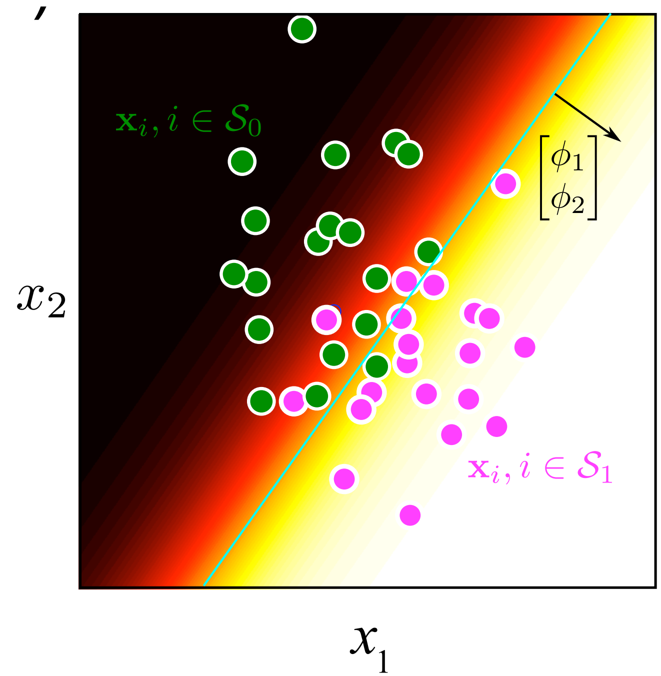
$$p(C_2 | \phi) = 1 - p(C_1 | \phi)$$

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

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



$\mathbf{w}^t \phi$



Logistic Regression

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

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**.

End of Lecture 12

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

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 $R_{nn} = y_n(1 - y_n)$

(Note that, since $\mathbf{R}_{nn} \geq 0$, $\mathbf{R}$ is positive semi-definite, and hence $\mathbf{H}$ is positive semi-definite

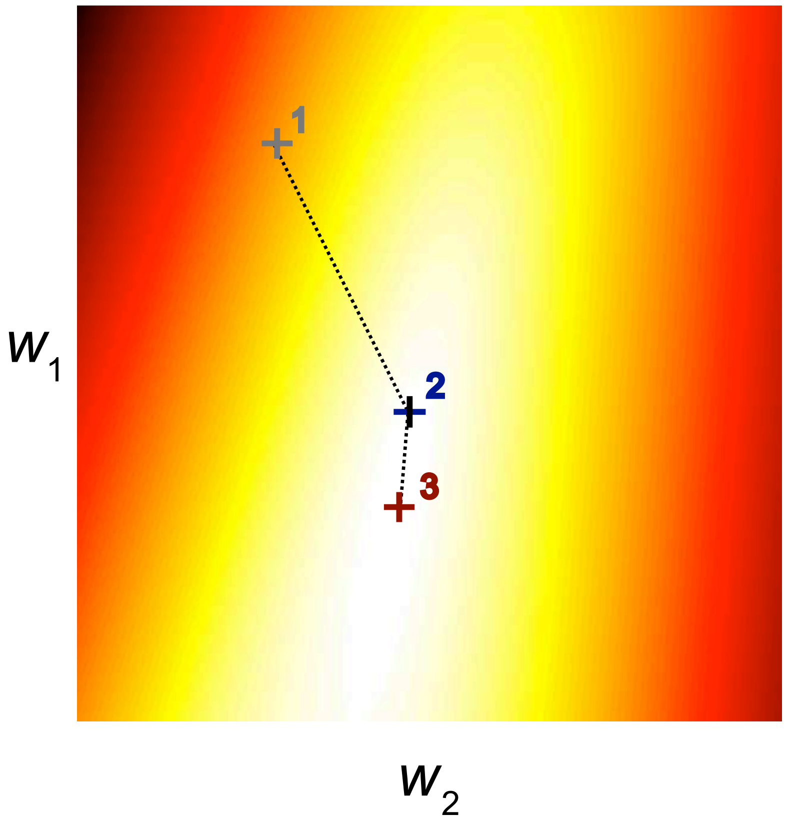
Thus $E(\mathbf{w})$ is convex.)

Thus

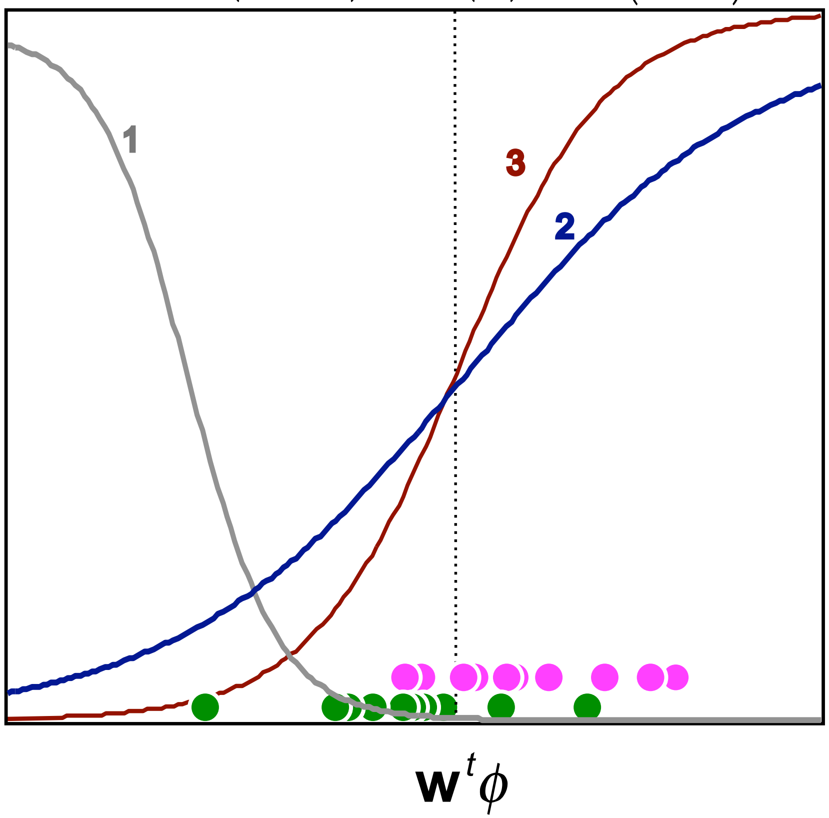
$$\mathbf{w}^{new} = \mathbf{w}^{(old)} - \left(\Phi^t \mathbf{R} \Phi\right)^{-1} \Phi^t \left(\mathbf{y} - \mathbf{t}\right)$$

ML for Logistic Regression

- Iterative Reweighted Least Squares



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



Logistic Regression

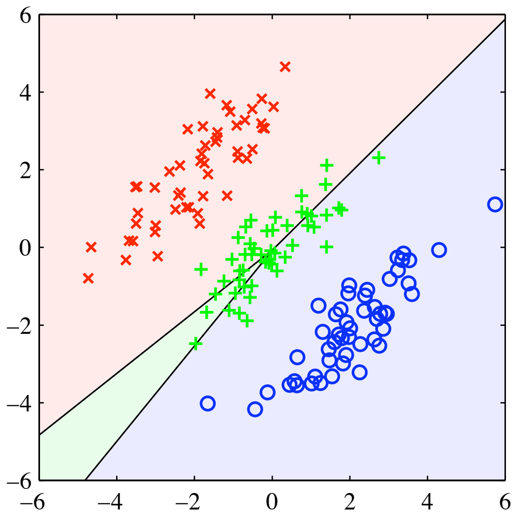
- For K>2, we can generalize the activation function by modeling the posterior probabilities as

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

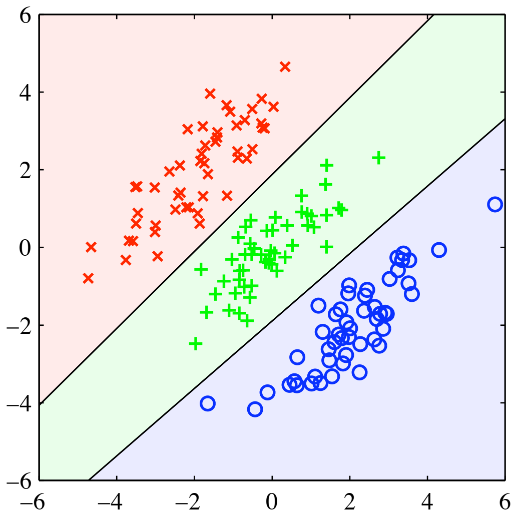
where the activations a_k are given by

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

Example



Least-Squares



Logistic

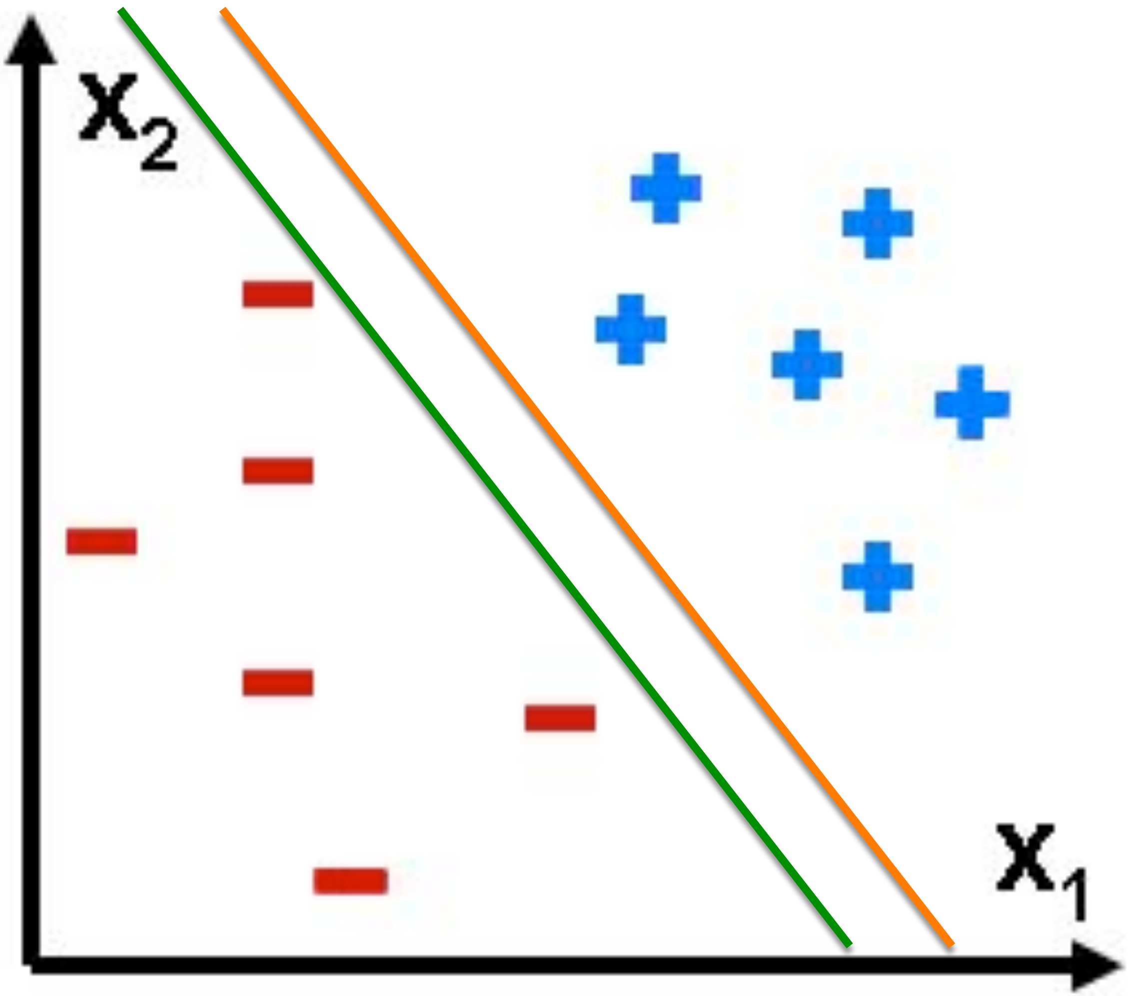
Outline

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

Motivation

- The perceptron algorithm is guaranteed to provide a linear decision surface that separates the training data, if one exists.
- However, if the data are linearly separable, there are in general an infinite number of solutions, and the solution returned by the perceptron algorithm depends in a complex way on the initial conditions, the learning rate and the order in which training data are processed.
- While all solutions achieve a perfect score on the training data, they won't all necessarily generalize as well to new inputs.

Which solution would you choose?



The Large Margin Classifier

- Unlike the Perceptron Algorithm, Support Vector Machines solve a problem that has a unique solution: they return the linear classifier with the maximum margin, that is, the hyperplane that separates the data and is farthest from any of the training vectors.
- Why is this good?

Support Vector Machines

SVMs are based on the linear model $y(\mathbf{x}) = \mathbf{w}^t \phi(\mathbf{x}) + b$

Assume training data $\mathbf{x}_1, \ldots, \mathbf{x}_N$ with corresponding target values $t_1, \ldots, t_N$, $\mathrm{t}_n \in \{-1, 1\}$.

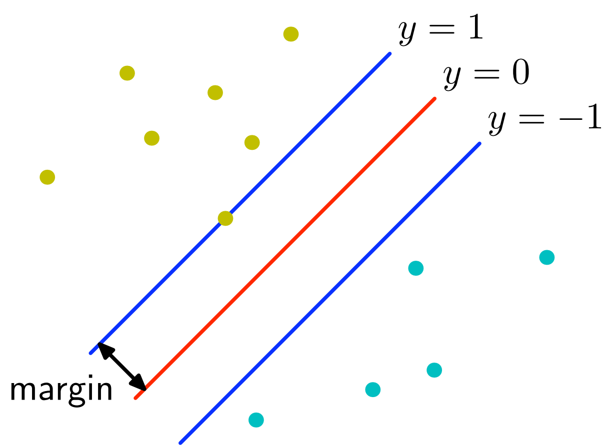
$\mathbf{x}$ classified according to sign of $y(\mathbf{x})$.

Assume for the moment that the training data are linearly separable in feature space.

Then $\exists \mathbf{w}, b : t_n y\left(\mathbf{x}_n\right) > 0 \;\; \forall n \in [1, \ldots N]$

Maximum Margin Classifiers

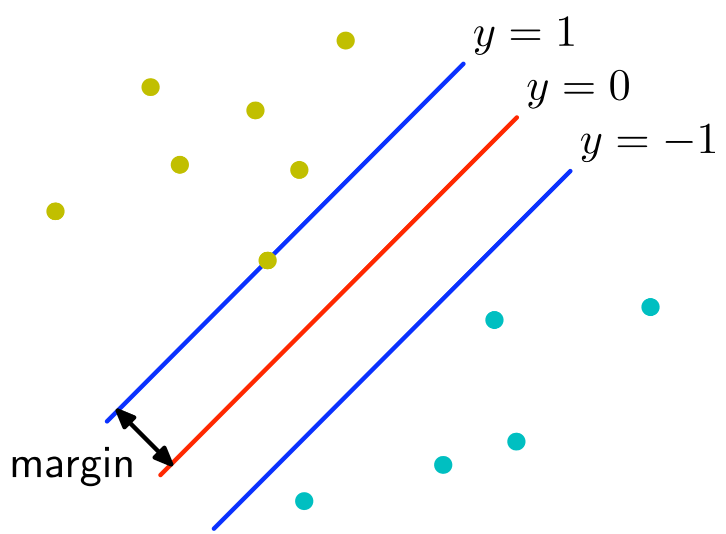
- When the training data are linearly separable, there are generally an infinite number of solutions for (**w**, *b*) that separate the classes exactly.
- The **margin** of such a classifier is defined as the orthogonal distance in feature space between the decision boundary and the closest training vector.
- SVMs are an example of a **maximum margin classifer**, which finds the linear classifier that maximizes the margin.



Probabilistic Motivation

- The maximum margin classifier has a probabilistic motivation.

If we model the class-conditional densities with a KDE using Gaussian kernels with variance σ^2, then in the limit as $\sigma \to 0$, the optimal linear decision boundary $\to$ maximum margin linear classifier.



Two Class Discriminant Function

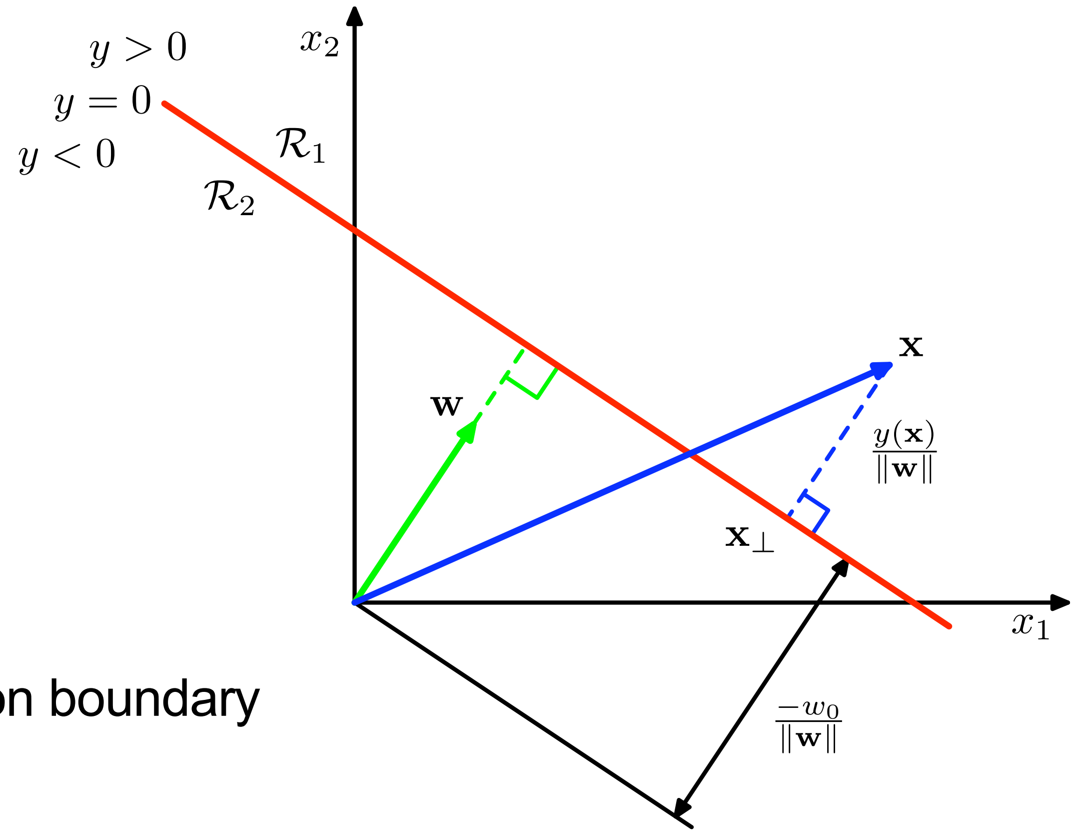
Recall:

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

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

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

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



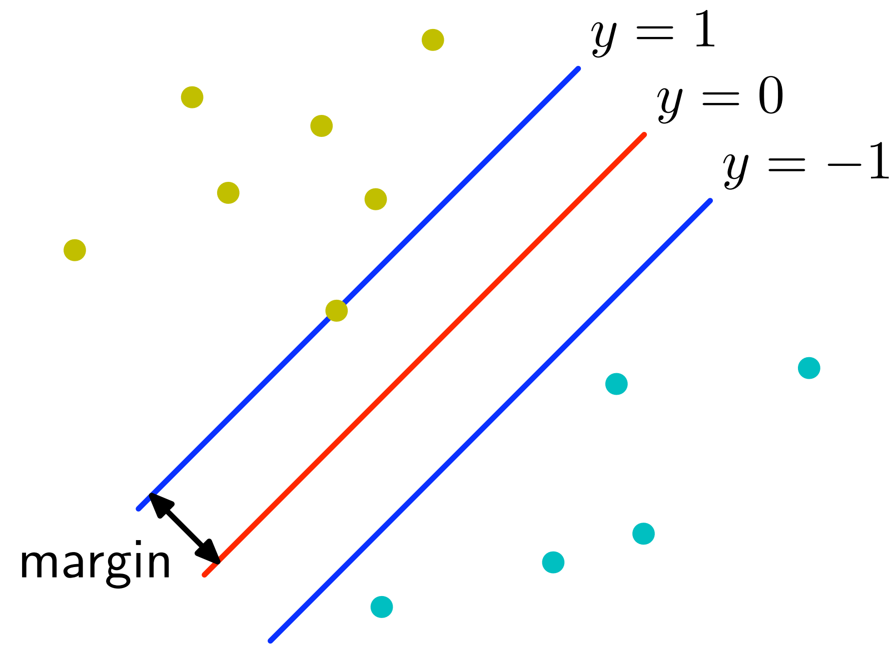
Maximum Margin Classifiers

Distance of point $\mathbf{x}_n$ from decision surface is given by:

$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n\left(\mathbf{w}^t\phi(\mathbf{x}_n)+b\right)}{\|\mathbf{w}\|}$$

Thus we seek:

$$\underset{\mathbf{w},b}{\arg\max}\left\{\frac{1}{\|\mathbf{w}\|}\min_n\left[t_n\left(\mathbf{w}^t\phi(\mathbf{x}_n)+b\right)\right]\right\}$$



Maximum Margin Classifiers

Distance of point $\mathbf{x}_n$ from decision surface is given by:

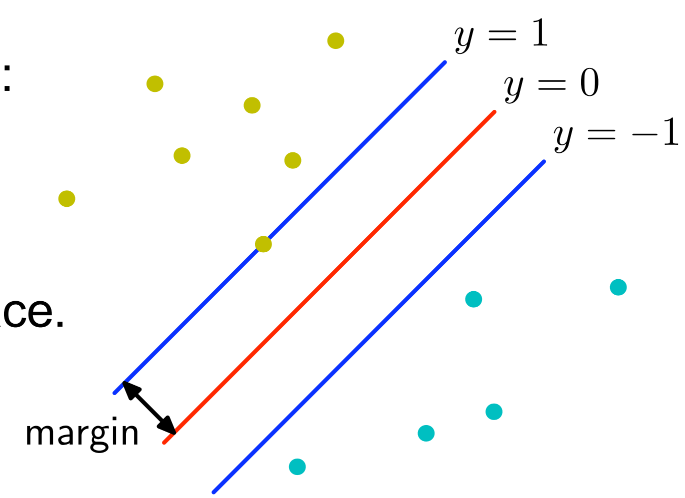
$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n \left(\mathbf{w}^t \phi(\mathbf{x}_n) + b\right)}{\|\mathbf{w}\|}$$

Note that rescaling $\mathbf{w}$ and b by the same factor leaves the distance to the decision surface unchanged.

Thus, wlog, we consider only solutions that satisfy:

$$t_n \left(\mathbf{w}^t \phi(\mathbf{x}_n) + b\right) = 1.$$

for the point $\mathbf{x}_n$ that is closest to the decision surface.



Quadratic Programming Problem

Then all points $\mathbf{x}_n$ satisfy $t_n\left(\mathbf{w}^t\phi(\mathbf{x}_n)+b\right)\geq 1$

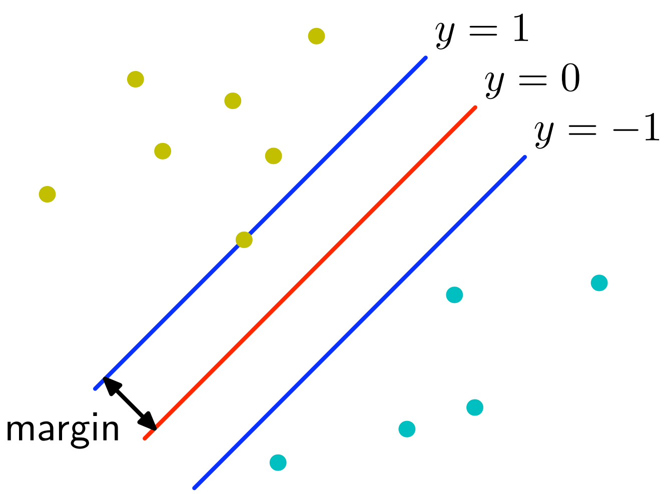
Points for which equality holds are said to be **active**.

All other points are **inactive**.

Now $\underset{\mathbf{w},b}{\arg\max}\left\{\frac{1}{\|\mathbf{w}\|}\min_n\left[t_n\left(\mathbf{w}^t\phi(\mathbf{x}_n)+b\right)\right]\right\}$

$\leftrightarrow \frac{1}{2}\underset{\mathbf{w}}{\arg\min}\|\mathbf{w}\|^2$

Subject to $t_n\left(\mathbf{w}^t\phi(\mathbf{x}_n)+b\right)\geq 1\ \forall \mathbf{x}_n$



This is a **quadratic programming** problem.

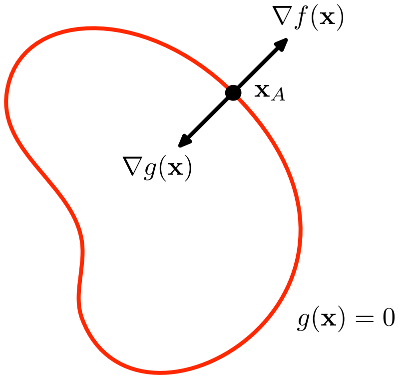
Solving this problem will involve **Lagrange multipliers**.

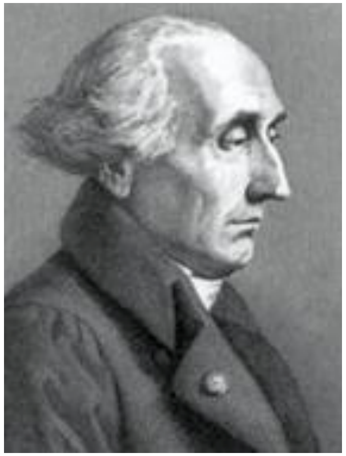
Lagrange Multipliers

Lagrange Multipliers (Appendix C.4 in textbook)

- Used to find stationary points of a function subject to one or more constraints.
- Example (equality constraint):

  Maximize $f(\mathbf{x})$ subject to $g(\mathbf{x}) = 0$.




Joseph-Louis Lagrange
1736-1813

- Observations:

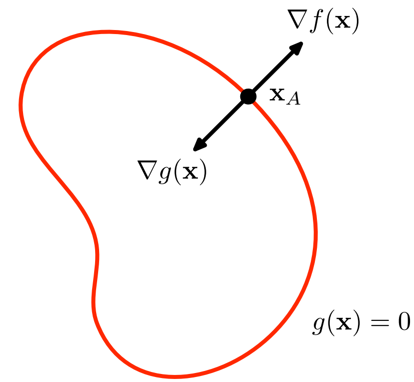
1. At any point on the constraint surface, $\nabla g(\mathbf{x})$ must be orthogonal to the surface.

2. Let $\mathbf{x}^*$ be a point on the constraint surface where $f(\mathbf{x})$ is maximized.
   Then $\nabla f(\mathbf{x})$ is also orthogonal to the constraint surface.

3. $\rightarrow \exists \lambda \neq 0$ such that $\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = 0$ at $\mathbf{x}^*$.
   λ is called a **Lagrange multiplier**.

Lagrange Multipliers (Appendix C.4 in textbook)

$\exists\lambda \neq 0$ such that $\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = 0$ at $\mathbf{x}^*$.



- Defining the **Lagrangian** function as:

  $$L(\mathbf{x},\lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

  we then have

  $$\nabla_{\mathbf{x}} L(\mathbf{x},\lambda) = 0.$$

  and

  $$\frac{\partial L(\mathbf{x},\lambda)}{\partial \lambda} = 0.$$

Example

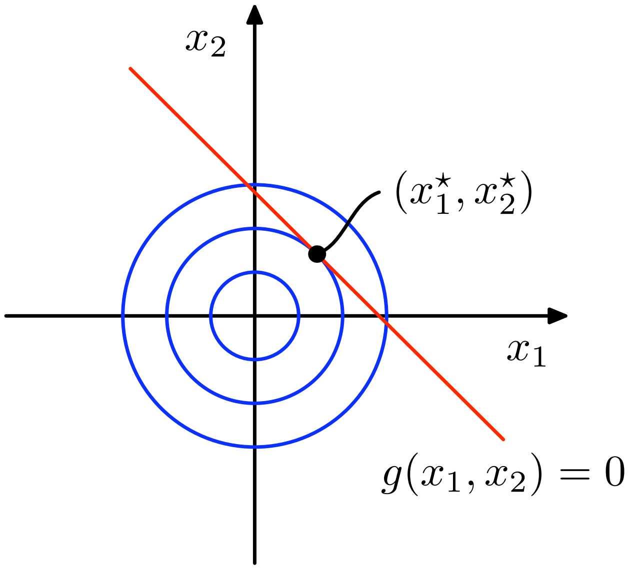
$$L(\mathbf{x},\lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

- Find the stationary point of

$$f(x_1,x_2) = 1 - x_1^2 - x_2^2$$

subject to

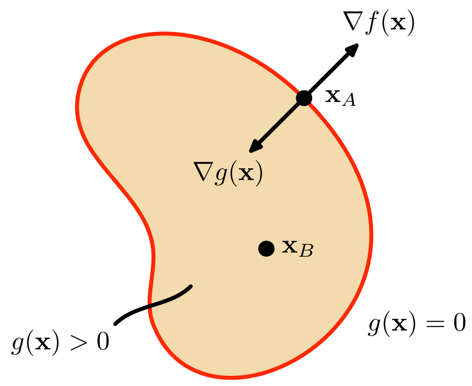
$$g(x_1,x_2) = x_1 + x_2 - 1 = 0$$



End of Lecture 13

Inequality Constraints

Maximize $f(\mathbf{x})$ subject to $g(\mathbf{x}) \geq 0$.



$$L(\mathbf{x},\lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

- There are 2 cases:
  1. $\mathbf{x}^*$ on the interior (e.g., $\mathbf{x}_B$)
     - Here g(x) > 0 and the stationary condition is simply $\nabla f(\mathbf{x}) = 0.$
     - This corresponds to a stationary point of the Lagrangian where $\lambda = 0$.
  2. $\mathbf{x}^*$ on the boundary (e.g., $\mathbf{x}_A$)
     - Here g(x) = 0 and the stationary condition is $\nabla f(\mathbf{x}) = -\lambda \nabla g(x),\ \lambda > 0.$
     - This corresponds to a stationary point of the Lagrangian where $\lambda > 0$.
- Thus the general problem can be expressed as maximizing the Lagrangian subject to
  1. $g(x) \geq 0$
  2. $\lambda \geq 0$
  3. $\lambda g(x) = 0$

  Karush-Kuhn-Tucker (KKT) conditions

Minimizing vs Maximizing

- If we want to **minimize** $f(x)$ subject to $g(x) \geq 0$, then the Lagrangian becomes

  $$L(\mathbf{x},\lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x})$$

  with $\lambda \geq 0$.

Extension to Multiple Constraints

- Suppose we wish to maximize $f(\mathbf{x})$ subject to

$$g_j(\mathbf{x}) = 0 \text{ for } j = 1,\ldots,J$$
$$h_k(\mathbf{x}) \geq 0 \text{ for k} = 1,\ldots,K$$

- We then find the stationary points of

$$L(\mathbf{x},\lambda) = f(\mathbf{x}) + \sum_{j=1}^{J} \lambda_j g_j(\mathbf{x}) + \sum_{k=1}^{K} \mu_k h_k(\mathbf{x})$$

subject to

$$h_k(\mathbf{x}) \geq 0$$
$$\mu_k \geq 0$$
$$\mu_k h_k(\mathbf{x}) = 0$$

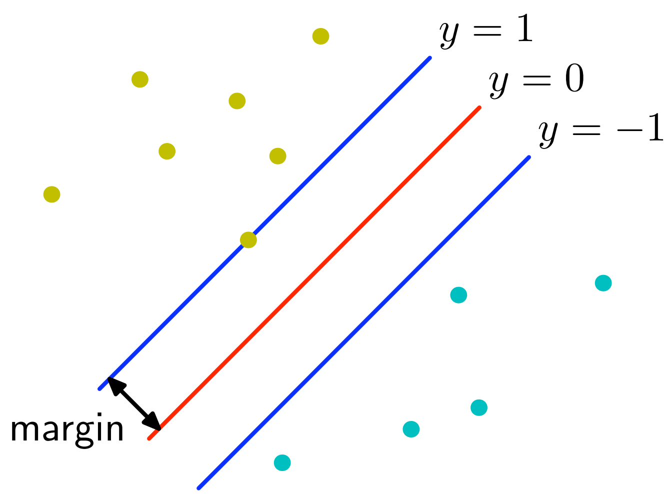
Back to our quadratic programming problem…

Quadratic Programming Problem

$$\frac{1}{2}\underset{\mathbf{w}}{\arg\min}\|\mathbf{w}\|^2, \text{ subject to } t_n\left(\mathbf{w}^t\phi\left(\mathbf{x}_n\right)+b\right)\geq 1 \ \forall \mathbf{x}_n$$

Solve using Lagrange multipliers a_n :

$$L(\mathbf{w},b,\mathbf{a})=\frac{1}{2}\|\mathbf{w}\|^2-\sum_{n=1}^{N}a_n\left\{t_n\left(\mathbf{w}^t\phi\left(\mathbf{x}_n\right)+b\right)-1\right\}$$



Dual Representation

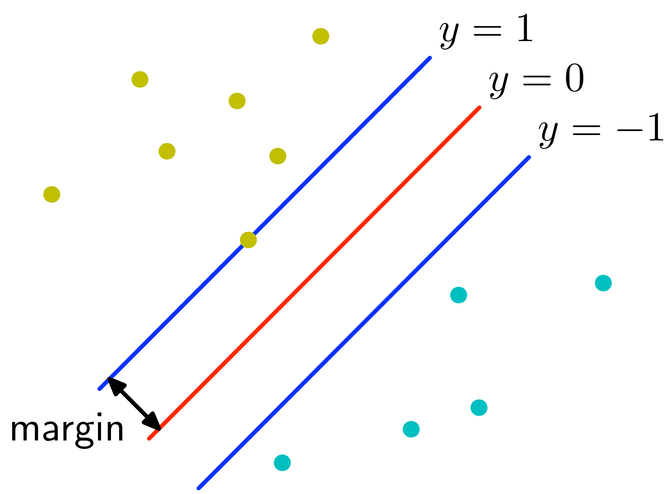
Solve using Lagrange multipliers a_n :

$$L(\mathbf{w},b,\mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n \left(\mathbf{w}^t \phi(\mathbf{x}_n) + b \right) - 1 \right\}$$

Setting derivatives with respect to **w** and b to 0, we get:

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)$$

$$\sum_{n=1}^{N} a_n t_n = 0$$



Dual Representation

$$L(\mathbf{w},b,\mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n \left(\mathbf{w}^t \phi(\mathbf{x}_n) + b \right) - 1 \right\}$$

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

Substituting leads to the dual representation of the maximum margin problem, in which we maximize:

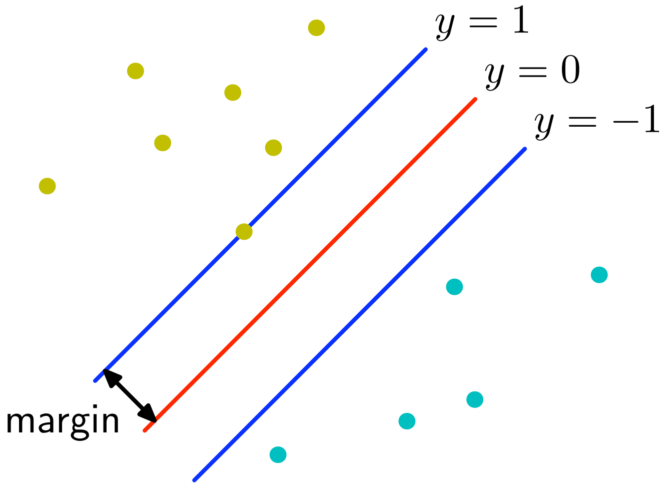
$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

with respect to **a**, subject to:

$$a_n \geq 0 \ \forall n$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

and where $k(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^t \phi(\mathbf{x}')$



Dual Representation

Using $\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)$, a new point x is classified by computing

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$

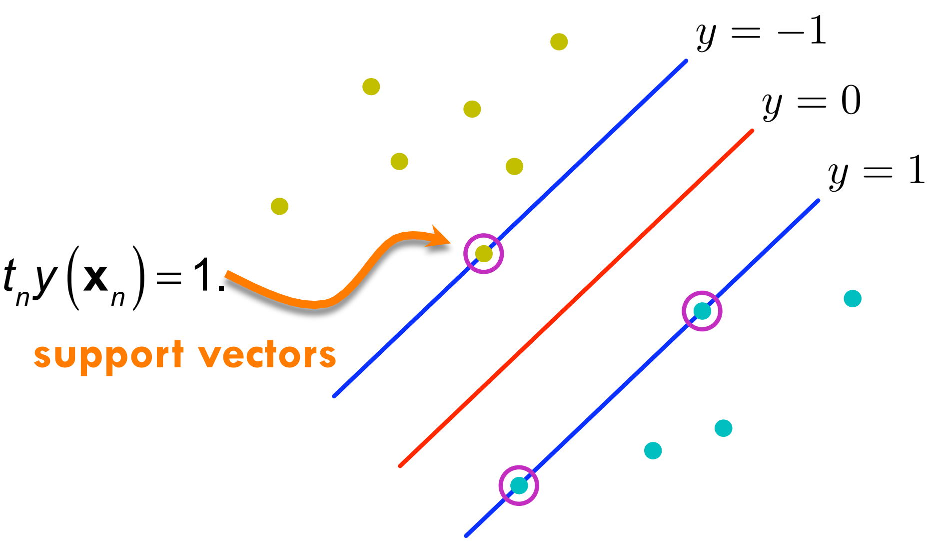
The Karush-Kuhn-Tucker (KKT) conditions for this constrained optimization problem are:

$$a_n \geq 0$$

$$t_n y\left(\mathbf{x}_n\right) - 1 \geq 0$$

$$a_n \left\{ t_n y\left(\mathbf{x}_n\right) - 1 \right\} = 0$$

Thus for every data point, either $a_n = 0$ or $t_n y\left(\mathbf{x}_n\right) = 1$.



Solving for the Bias

Once the optimal **a** is determined, the bias b can be computed by noting that any support vector $\mathbf{x}_n$ satisfies $t_n y(\mathbf{x}_n) = 1$.

Using $y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$

we have $t_n \left(\sum_{m=1}^{N} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right) = 1$

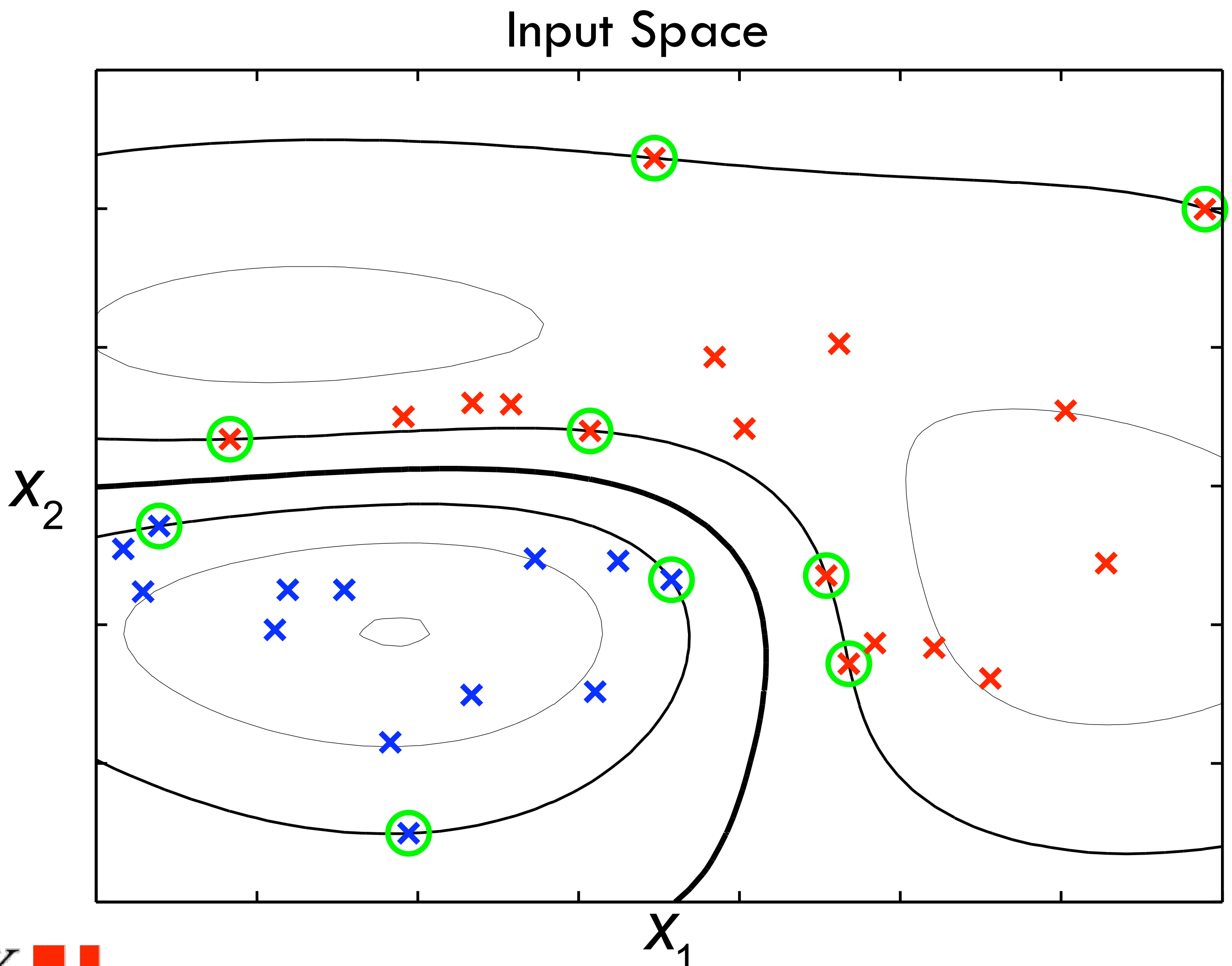
and so $b = t_n - \sum_{m=1}^{N} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m)$

A more numerically accurate solution can be obtained by averaging over all support vectors:

$$b = \frac{1}{N_S} \sum_{n \in S} \left(t_n - \sum_{m \in S} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right)$$

where S is the index set of support vectors and N_S is the number of support vectors.

Example (Gaussian Kernel)



Overlapping Class Distributions

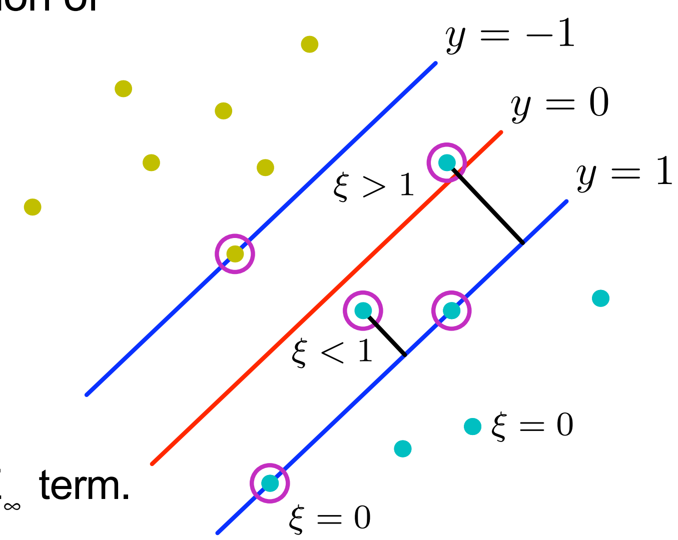
- The SVM for non-overlapping class distributions is determined by solving

$$\frac{1}{2}\arg\min_{\mathbf{w}} \|\mathbf{w}\|^2, \text{ subject to } t_n\left(\mathbf{w}^t\phi(\mathbf{x}_n)+b\right) \geq 1 \ \forall \mathbf{x}_n$$

Alternatively, this can be expressed as the minimization of

$$\sum_{n=1}^{N} E_\infty\left(y(\mathbf{x}_n)t_n - 1\right) + \lambda\|\mathbf{w}\|^2$$

where $E_\infty(z)$ is 0 if $z \geq 0$, and ∞ otherwise.



This forces all points to lie on or outside the margins, on the correct side for their class.

To allow for misclassified points, we have to relax this E_∞ term.

Slack Variables

To this end, we introduce N **slack variables** $\xi_n \geq 0,\ n = 1,\ldots N$.

$\xi_n = 0$ for points on or on the correct side of the margin boundary for their class

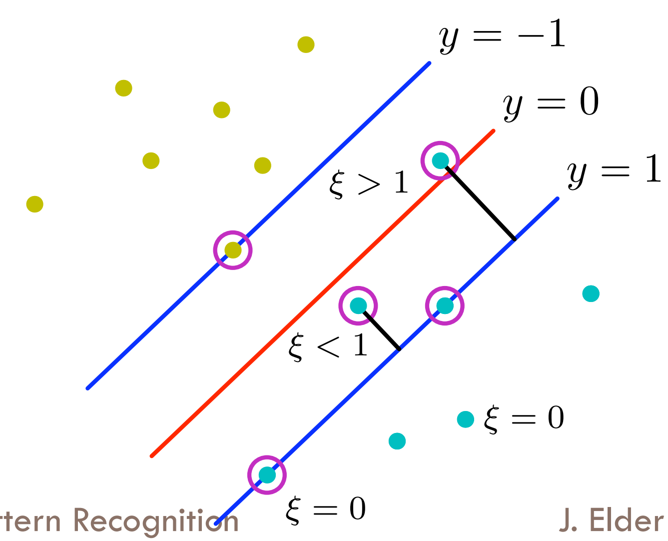
$\xi_n = \left| t_n - y\left(\mathbf{x}_n\right) \right|$ for all other points.

Thus $\xi_n < 1$ for points that are correctly classified

$\xi_n > 1$ for points that are incorrectly classified

We now minimize $C\sum_{n=1}^{N} \xi_n + \frac{1}{2}\|\mathbf{w}\|^2$, where $C > 0$.

subject to $t_n y\left(\mathbf{x}_n\right) \geq 1 - \xi_n$, and $\xi_n \geq 0,\ n = 1,\ldots N$



Dual Representation

This leads to a dual representation, where we maximize

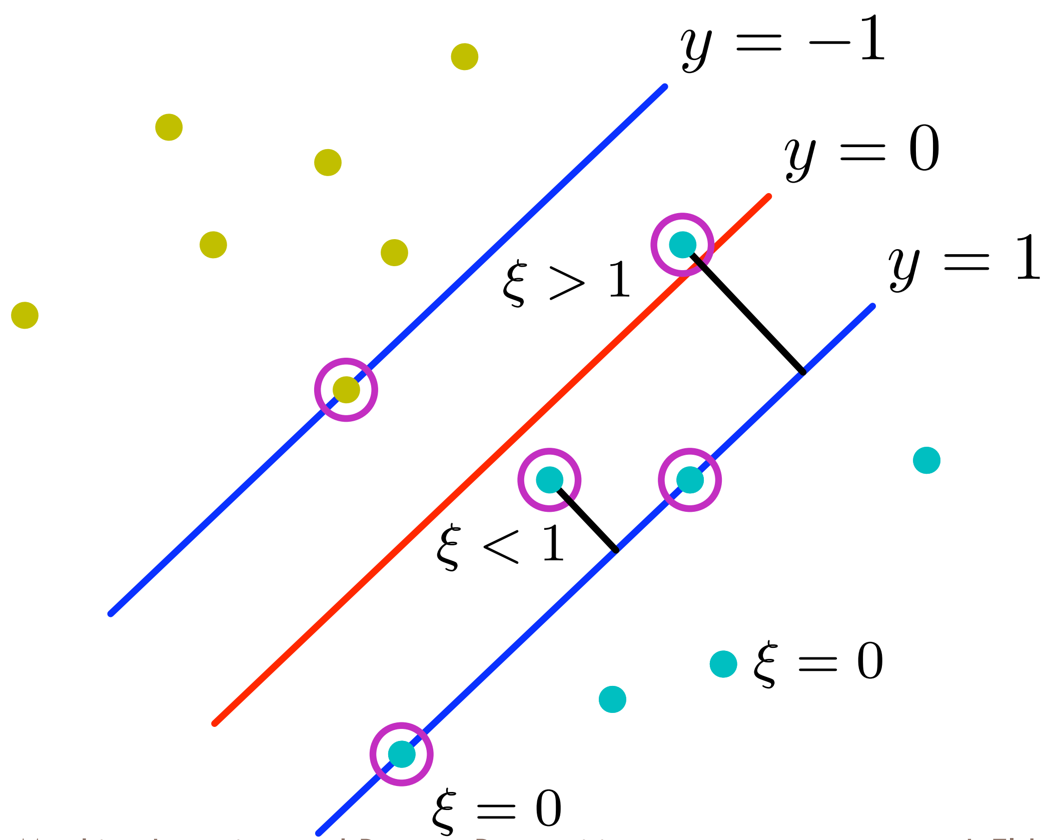
$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k\left(\mathbf{x}_n, \mathbf{x}_n\right)$$

with constraints

$$0 \le a_n \le C$$

and

$$\sum_{n=1}^{N} a_n t_n = 0$$



Support Vectors

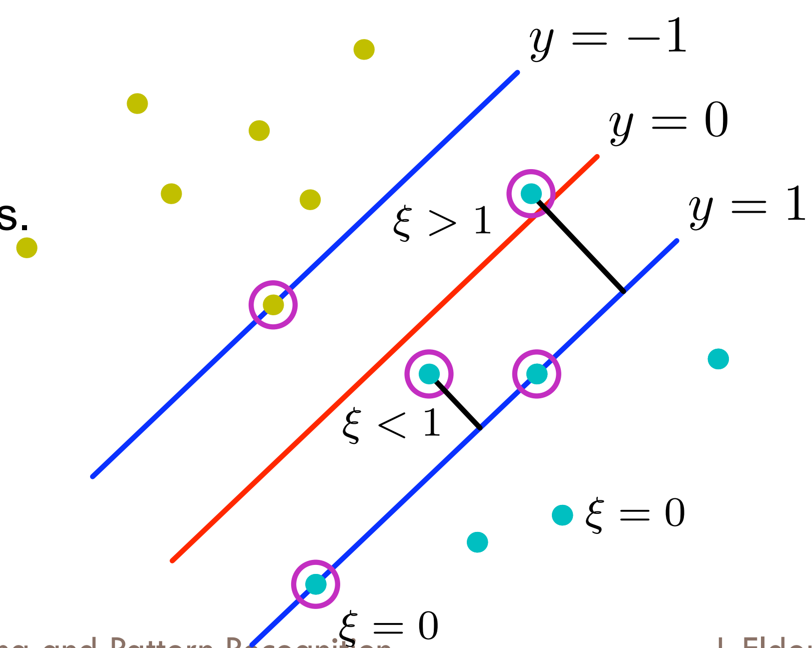
Again, a new point **x** is classified by computing

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$

For points that are on the correct side of the margin, $a_n = 0$.

Thus support vectors consist of points between their margin and the decision boundary, as well as misclassified points.

In other words, all points that are not on the right side of their margin are support vectors.



Bias

Again, a new point **x** is classified by computing

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$

Once the optimal **a** is determined, the bias *b* can be computed from

$$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(t_n - \sum_{m \in S} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right)$$

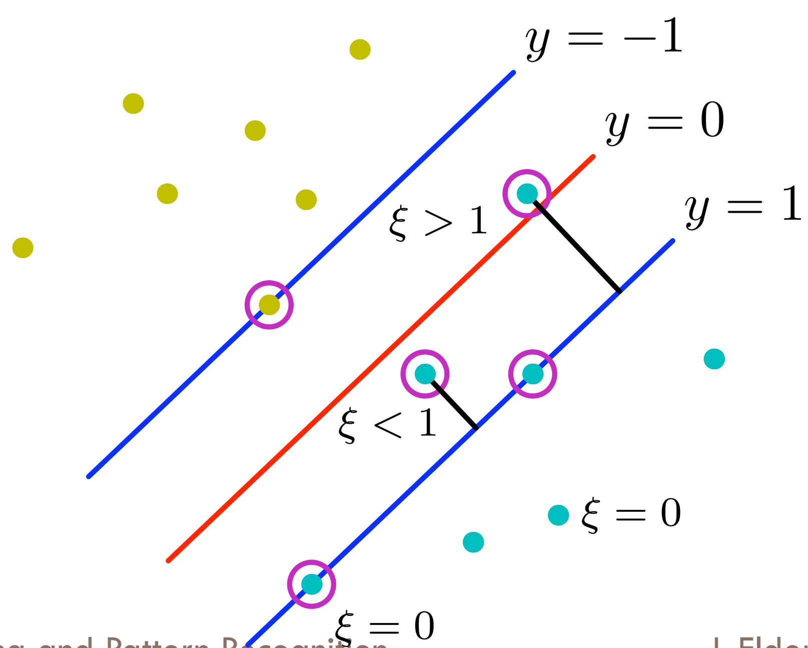
where

S is the index set of support vectors

N_S is the number of support vectors

$\mathcal{M}$ is the index set of points on the margins

$N_{\mathcal{M}}$ is the number of points on the margins



Solving the Quadratic Programming Problem

$$\text{Maximize } \tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k\left(\mathbf{x}_n, \mathbf{x}_m\right)$$

$$\text{subject to } 0 \le a_n \le C \text{ and } \sum_{n=1}^{N} a_n t_n = 0$$

- Problem is convex.
- Standard solutions are generally $O(N^3)$.
- Traditional quadratic programming techniques often infeasible due to computation and memory requirements.
- Instead, methods such as **sequential minimal optimization** can be used, that in practice are found to scale as $O(N)$ - $O(N^2)$.

Chunking

$$\text{Maximize } \tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$\text{subject to } 0 \le a_n \le C \text{ and } \sum_{n=1}^{N} a_n t_n = 0$$

- Conventional quadratic programming solution requires that matrices with N^2 elements be maintained in memory.

$$K \sim O(N^2), \text{ where } K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m)$$

$$T \sim O(N^2), \text{ where } T_{nm} = t_n t_m$$

$$A \sim O(N^2), \text{ where } A_{nm} = a_n a_m$$

- This becomes infeasible when N exceeds ~10,000.

Chunking

$$\text{Maximize } \tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k\left(\mathbf{x}_n, \mathbf{x}_m\right)$$

$$\text{subject to } 0 \le a_n \le C \text{ and } \sum_{n=1}^{N} a_n t_n = 0$$

- Chunking (Vapnik, 1982) exploits the fact that the value of the Lagrangian is unchanged if we remove the rows and columns of the kernel matrix where $a_n = 0$ or $a_m = 0$.

Chunking

Minimize $C\sum_{n=1}^{N}\xi_n + \frac{1}{2}\|\mathbf{w}\|^2$, where $C > 0$.

$\xi_n = 0$ for points on or on the correct side of the margin boundary for their class

$\xi_n = |t_n - y(\mathbf{x}_n)|$ for all other points.

- Chunking (Vapnik, 1982)
  1. Select a small number (a 'chunk') of training vectors
  2. Solve the QP problem for this subset
  3. Retain only the support vectors
  4. Consider another chunk of the training data
  5. Ignore the subset of vectors in all chunks considered so far that lie on the correct side of the margin, since these do not contribute to the cost function
  6. Add the remainder to the current set of support vectors and solve the new QP problem
  7. Return to Step 4
  8. Repeat until the set of support vectors does not change.

This method reduces memory requirements to $O(N_S^2)$, where N_S is the number of support vectors.

This may still be big!

Decomposition Methods

- It can be shown that the global QP problem is solved when, for all training vectors, satisfy the following optimality conditions:

$$a_i = 0 \Leftrightarrow t_i y(\mathbf{x}_i) \geq 1.$$

$$0 < a_i < C \Leftrightarrow t_i y(\mathbf{x}_i) = 1.$$

$$a_i = C \Leftrightarrow t_i y(\mathbf{x}_i) \leq 1.$$

- Decomposition methods decompose this large QP problem into a series of smaller subproblems.
- Decomposition (Osuna et al, 1997)
  - Partition the training data into a small working subset B and a fixed subset N.
  - Minimize the global objective function by adjusting the coefficients in B
  - Swap 1 or more vectors in B for an equal number in N that fail to satisfy the optimality conditions
  - Re-solve the global QP problem for B
- Each step is $O(B)^2$ in memory.
- Osuna et al (1997) proved that the objective function decreases on each step and will converge in a finite number of iterations.

Sequential Minimal Optimization

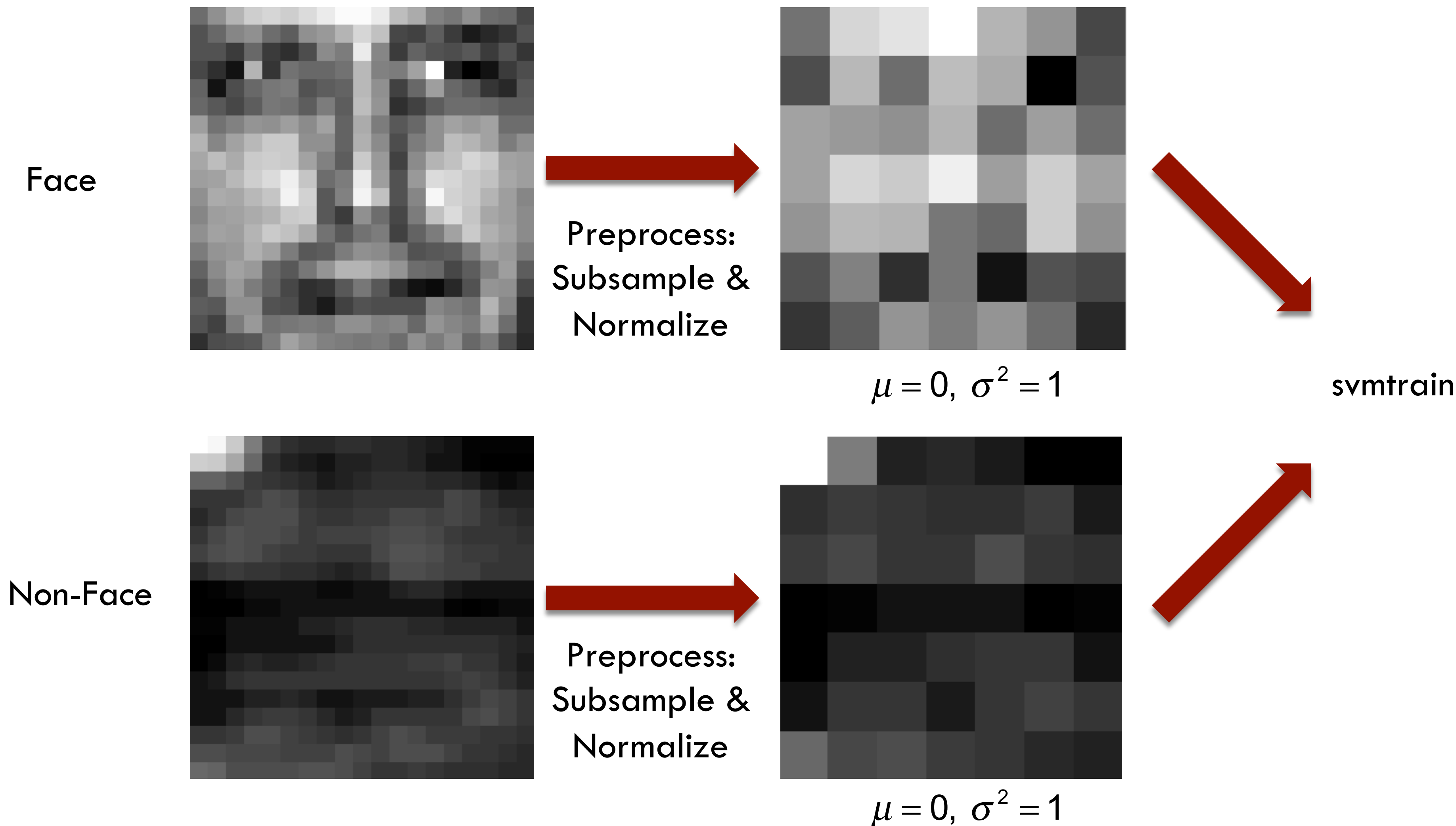
- Sequential Minimal Optimization (Platt 1998) takes decomposition to the limit.
- On each iteration, the working set consists of just two vectors.
- The advantage is that in this case, the QP problem can be solved analytically.
- Memory requirement are $O(N)$.
- Compute time is typically $O(N) - O(N^2)$.

LIBSVM

- LIBSVM is a widely used library for SVMs developed by Chang & Lin (2001).
  - Can be downloaded from www.csie.ntu.edu.tw/~cjlin/libsvm
  - MATLAB interface
  - Uses SMO
  - Will use for Assignment 2.

End of Lecture 14

LIBSVM Example: Face Detection



LIBSVM Example: MATLAB Interface

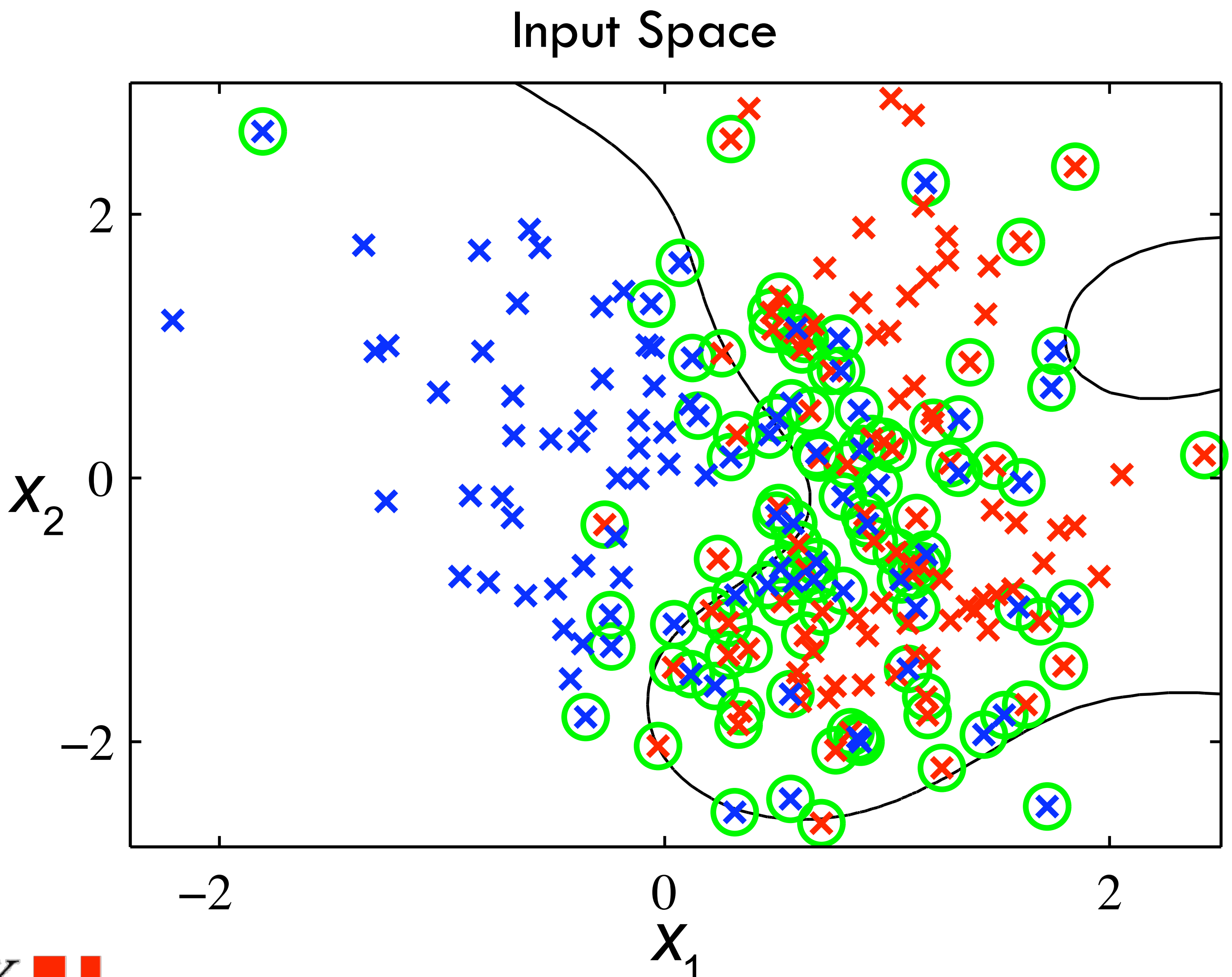
Selects linear SVM

```
model=svmtrain(traint, trainx, '-t 0');

[predicted_label, accuracy, decision_values] = svmpredict(testt, testx, model);
```

Accuracy = 70.0212% (661/944) (classification)

Example



Relation to Logistic Regression

The objective function for the soft-margin SVM can be written as:

$$\sum_{n=1}^{N} E_{SV}\left(y_n t_n\right) + \lambda \left\|\mathbf{w}\right\|^2$$

where $E_{SV}(z) = \left[1 - z\right]_+$ is the **hinge error function**,

and $\left[z\right]_+ = z \text{ if } z \geq 0$

$= 0$ otherwise.

For $t \in \{-1, 1\}$, the objective function for a regularized version of logistic regression can be written as:

$$\sum_{n=1}^{N} E_{LR}\left(y_n t_n\right) + \lambda \left\|\mathbf{w}\right\|^2$$

where $E_{LR}(z) = \log\left(1 + \exp(-z)\right)$.