Introduction to Artificial Intelligence CSCI 3202: The Perceptron Algorithm

Greg Grudic

Questions?

Binary Classification

- A binary classifier is a mapping from a set of *d* inputs to a single output which can take on one of TWO values
- In the most general setting

$$\text{inputs: } \mathbf{x} \in \Re^d$$

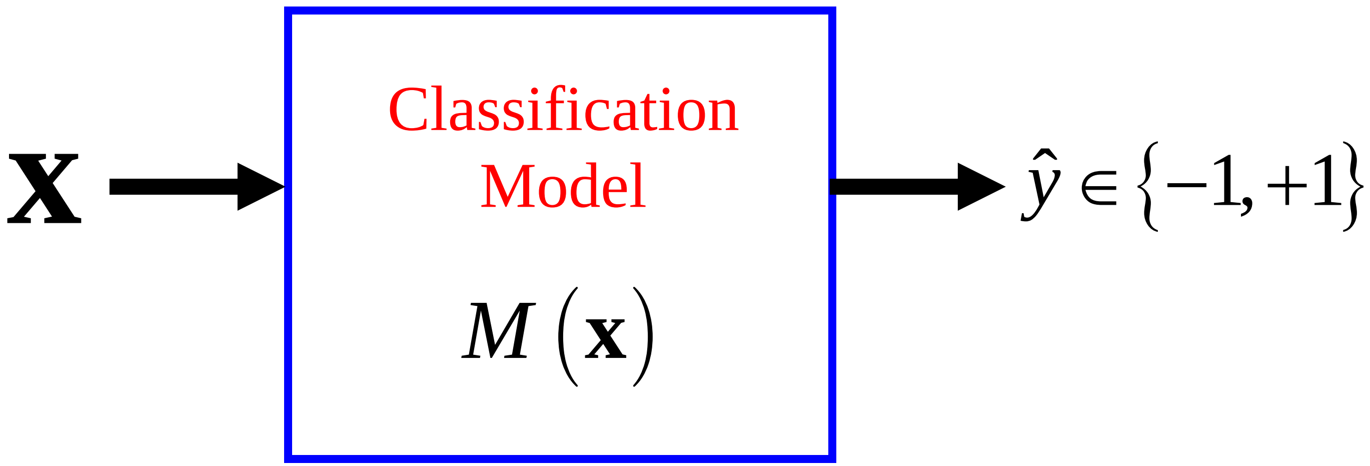
$$\text{output: } y \in \{-1, +1\}$$

- Specifying the output classes as -1 and +1 is arbitrary!
  - Often done as a mathematical convenience

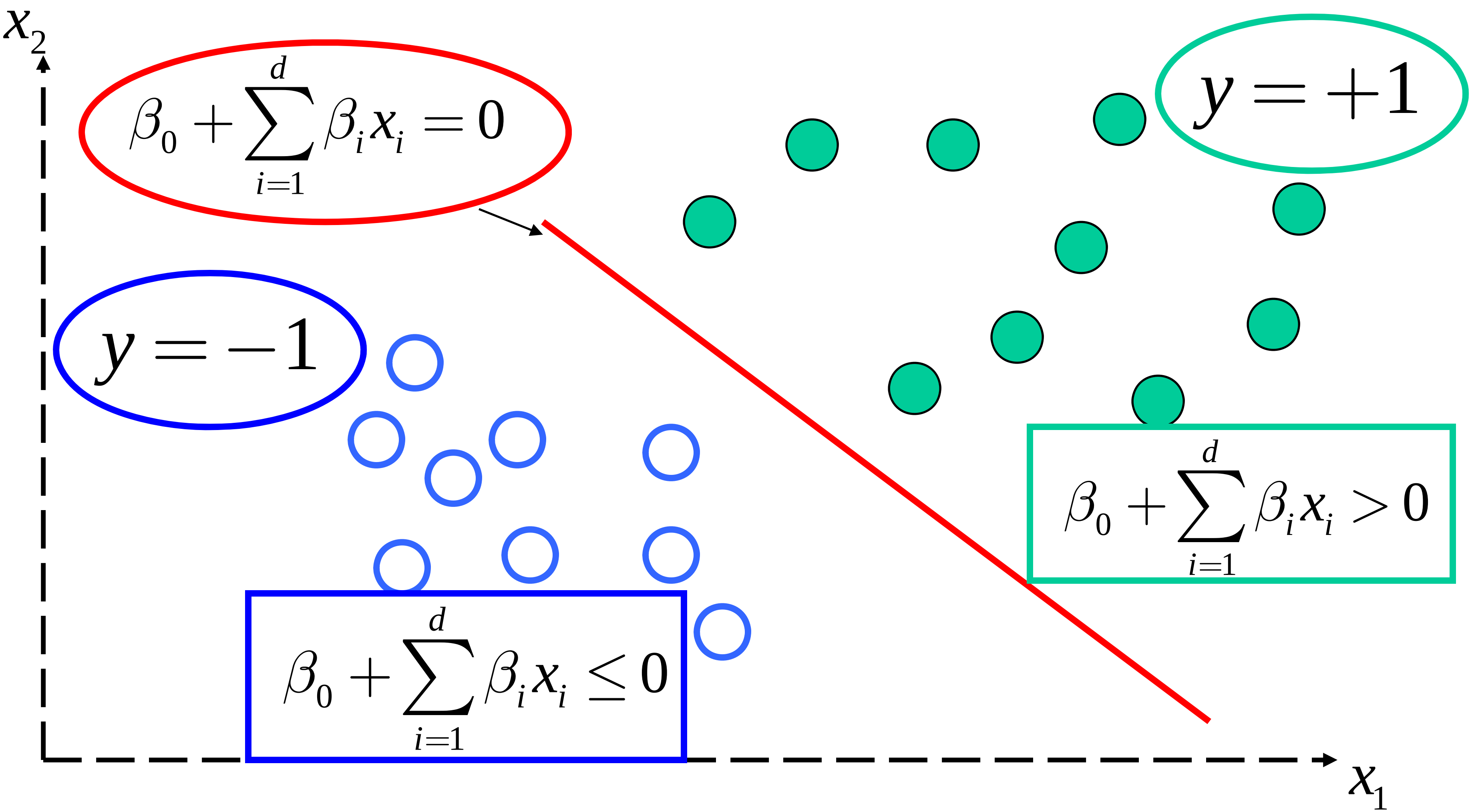
A Binary Classifier

Given learning data: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

A model is constructed:



Linear Separating Hyper-Planes



Linear Separating Hyper-Planes

- The Model:

$$\hat{y} = M(\mathbf{x}) = \operatorname{sgn}\left[\hat{\beta}_0 + \left(\hat{\beta}_1, ..., \hat{\beta}_d\right)\mathbf{x}^T\right]$$

- Where:

$$\operatorname{sgn}[A] = \begin{cases} 1 & \text{if } A > 0 \\ -1 & \text{otherwise} \end{cases}$$

- The decision boundary:

$$\hat{\beta}_0 + \left(\hat{\beta}_1, ..., \hat{\beta}_d\right)\mathbf{x}^T = 0$$

Linear Separating Hyper-Planes

- The model parameters are:

$$\left(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d\right)$$

- The *hat* on the betas means that they are estimated from the data
- Many different learning algorithms have been proposed for determining $\left(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d\right)$

Rosenblatt's Preceptron Learning Algorithm

- Dates back to the 1950's and is the motivation behind Neural Networks
- The algorithm:
  - Start with a random hyperplane $\left(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d\right)$
  - Incrementally modify the hyperplane such that points that are misclassified move closer to the correct side of the boundary
  - Stop when all learning examples are correctly classified

Rosenblatt's Preceptron Learning Algorithm

- The algorithm is based on the following property:
  - Signed distance of any point $\mathbf{x}$ to the boundary is:

$$d = \frac{\hat{\beta}_0 + \left(\hat{\beta}_1, \ldots, \hat{\beta}_d\right)\mathbf{x}^T}{\sqrt{\left(\sum_{i=1}^{d} \hat{\beta}_i^2\right)}} \propto \hat{\beta}_0 + \left(\hat{\beta}_1, \ldots, \hat{\beta}_d\right)\mathbf{x}^T$$

- Therefore, if M is the set of misclassified learning examples, we can push them closer to the boundary by minimizing the following

$$D\left(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d\right) = -\sum_{i \in M} y_i \left(\hat{\beta}_0 + \left(\hat{\beta}_1, \ldots, \hat{\beta}_d\right)\mathbf{x}_i{}^T\right)$$

Rosenblatt's Minimization Function

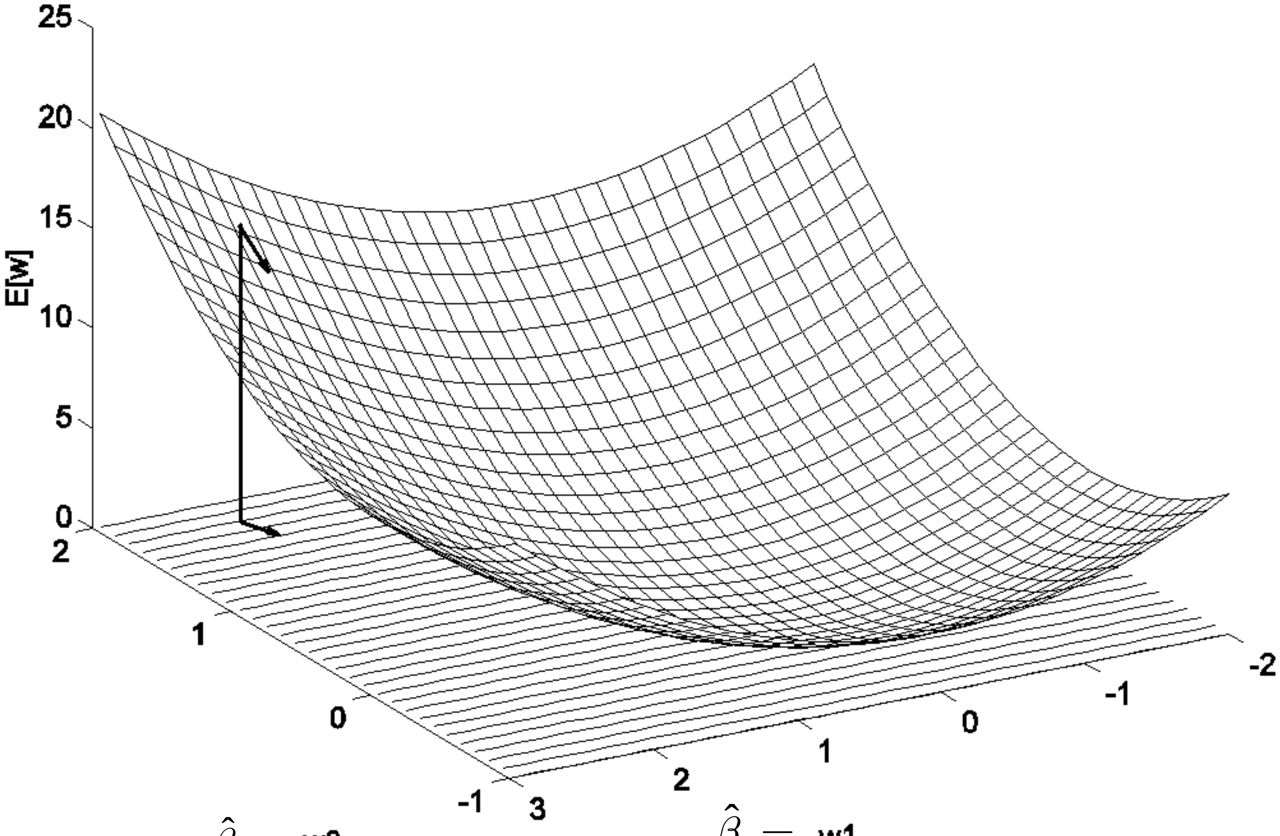
- This is classic Machine Learning!
- First define a cost function in model parameter space

$$D\left(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d\right) = -\sum_{i \in M} y_i \left(\hat{\beta}_0 + \sum_{k=1}^{d} \hat{\beta}_k x_{ik}\right)$$

- Then find an algorithm that modifies $\left(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d\right)$ such that this cost function is minimized
- One such algorithm is **Gradient Descent**

Gradient Descent

$$D\left(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d\right) =$$



$\hat{\beta}_0 =$ w0

$\hat{\beta}_1 =$ w1

The Gradient Descent Algorithm

$$\hat{\beta}_i \leftarrow \hat{\beta}_i - \rho \frac{\partial D\left(\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_d\right)}{\partial \hat{\beta}_i}$$

Where the learning rate is defined by: $\rho > 0$

The Gradient Descent Algorithm for the Perceptron

$$\frac{\partial D\left(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d\right)}{\partial \hat{\beta}_0} = -\sum_{i \in M} y_i \qquad \frac{\partial D\left(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d\right)}{\partial \hat{\beta}_j} = -\sum_{i \in M} y_i x_{ij}, \qquad j = 1, \ldots, d$$

Two Versions of the Perceptron Algorithm:

$$\begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_d \end{pmatrix} \leftarrow \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_d \end{pmatrix} - \rho \begin{pmatrix} y_i \\ y_i x_{i1} \\ \vdots \\ y_i x_{id} \end{pmatrix}$$

Update One misclassified point at a time (online)

$$\begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_d \end{pmatrix} \leftarrow \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_d \end{pmatrix} - \rho \begin{pmatrix} -\sum_{i \in M} y_i \\ -\sum_{i \in M} y_i x_{i1} \\ \vdots \\ -\sum_{i \in M} y_i x_{id} \end{pmatrix}$$

Update all misclassified points at once (batch)

The Learning Data

Training Data: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

- Matrix Representation of N learning examples of d dimensional inputs

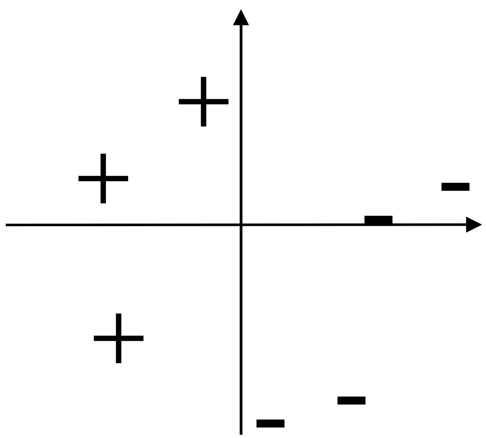
$$X = \begin{pmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{Nd} \end{pmatrix}, \quad Y = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

The Good Theoretical Properties of the Perceptron Algorithm

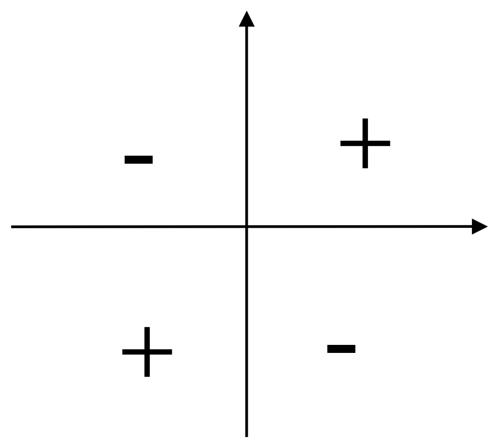
- If a solution exists the algorithm will always converge in a finite number of steps!
- Question: Does a solution always exist?

Linearly Separable Data

- Which of these datasets are separable by a linear boundary?



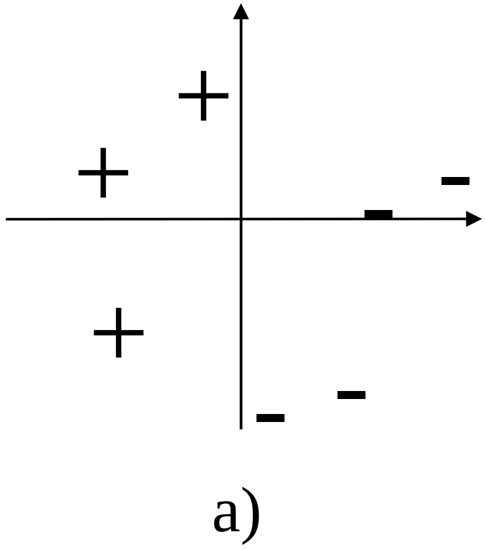
a)

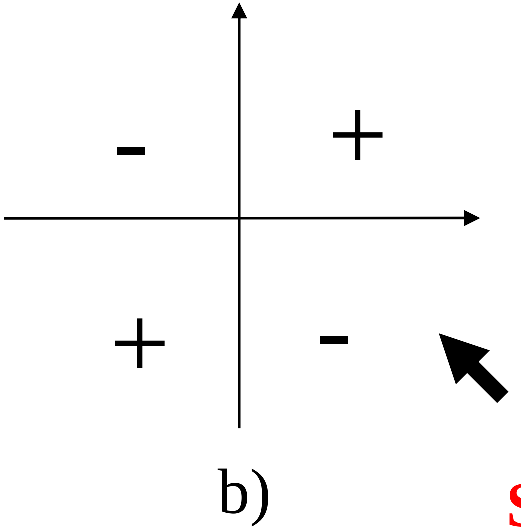


b)

Linearly Separable Data

- Which of these datasets are separable by a linear boundary?





Not Linearly Separable!

Bad Theoretical Properties of the Perceptron Algorithm

- If the data is not linearly separable, algorithm cycles forever!
  - Cannot converge!
  - This property "stopped" active research in this area between 1968 and 1984…
    - *Perceptrons*, Minsky and Pappert, 1969
- Even when the data is separable, there are infinitely many solutions
  - Which solution is best?
- When data is linearly separable, the number of steps to converge can be very large (depends on size of gap between classes)

What about Nonlinear Data?

- Data that is not linearly separable is called nonlinear data

- Nonlinear data can often be mapped into a nonlinear space where it is linearly separable

Nonlinear Models

- The Linear Model:

$$\hat{y} = M(\mathbf{x}) = \operatorname{sgn}\left[\hat{\beta}_0 + \sum_{i=1}^{d} \hat{\beta}_i x_i\right]$$

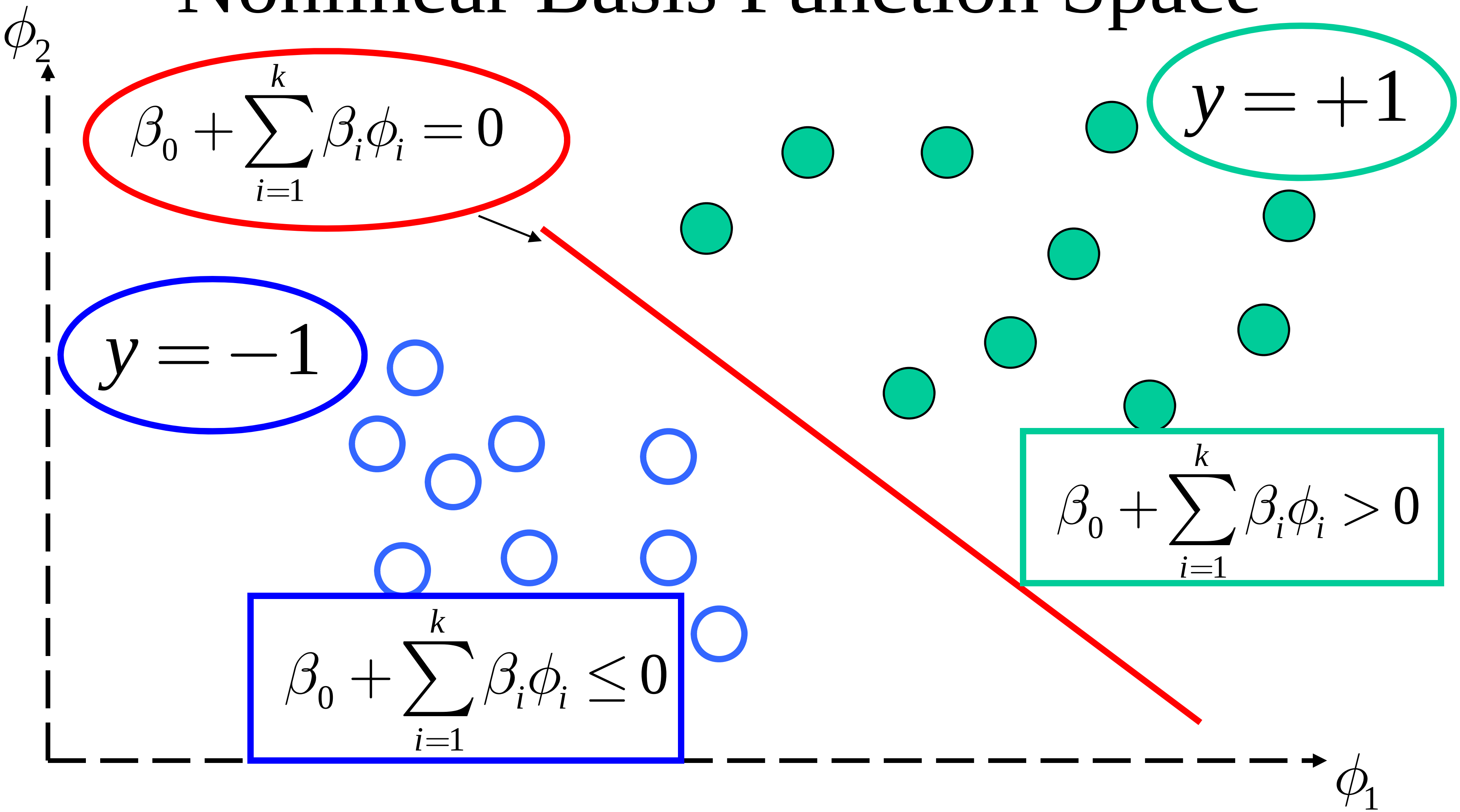
- The Nonlinear (basis function) Model:

$$\hat{y} = M(\mathbf{x}) = \operatorname{sgn}\left[\hat{\beta}_0 + \sum_{i=1}^{k} \hat{\beta}_i \phi_i(\mathbf{x})\right]$$

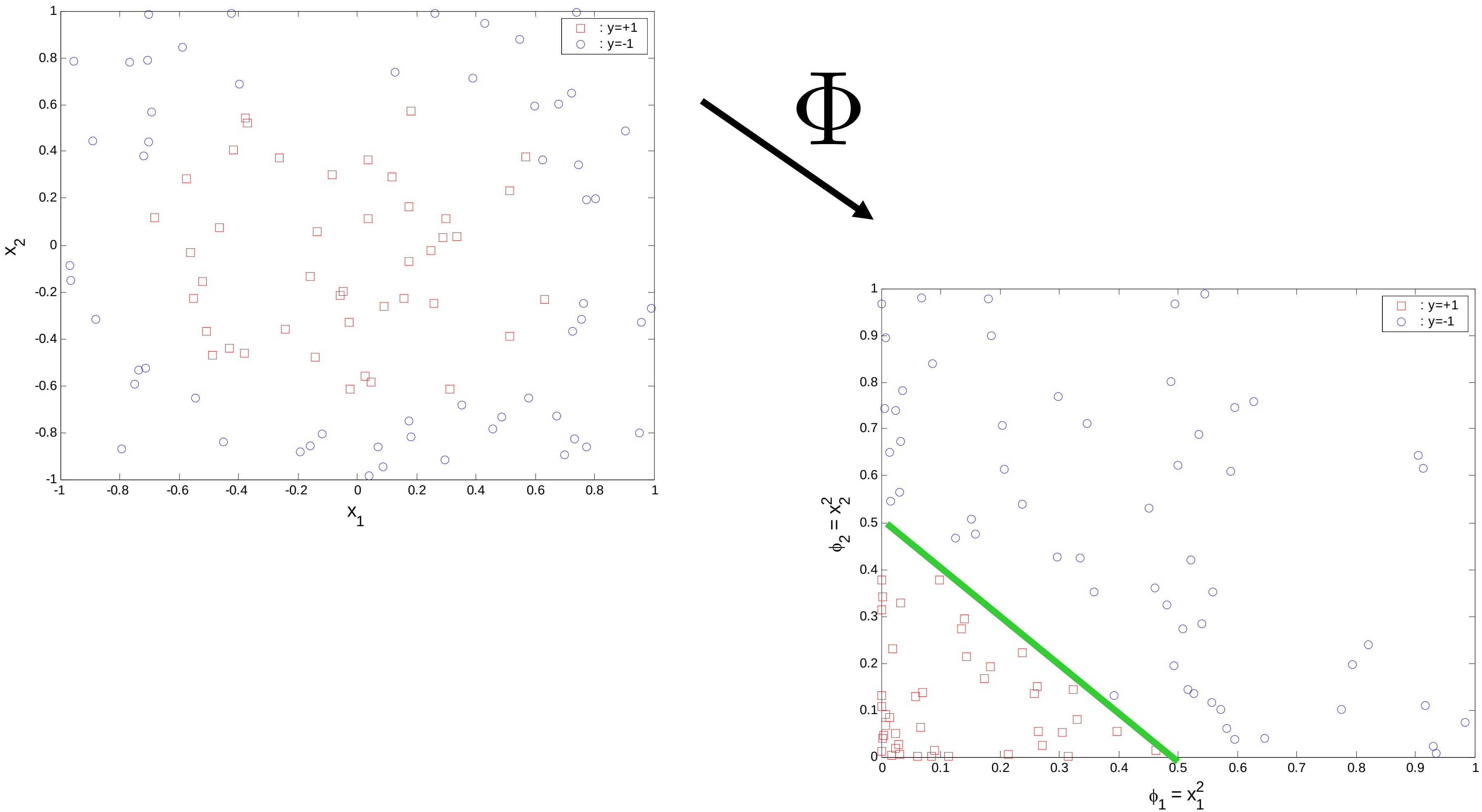
- Examples of Nonlinear Basis Functions:

$$\phi_1(\mathbf{x}) = x_1^2 \quad \phi_2(\mathbf{x}) = x_2^2 \quad \phi_3(\mathbf{x}) = x_1 x_2 \quad \phi_4(\mathbf{x}) = \sin(x_{55})$$

Linear Separating Hyper-Planes In Nonlinear Basis Function Space



An Example



Kernels as Nonlinear Transformations

- Polynomial

$$K(\mathbf{x}_i, \mathbf{x}_j) = \left(\left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle + q\right)^k$$

- Sigmoid

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(\kappa \left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle + \theta\right)$$

- Gaussian or Radial Basis Function (RBF)

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2\right)$$

The Kernel Model

Training Data: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

$$\hat{y} = M(\mathbf{x}) = \operatorname{sgn}\left[\hat{\beta}_0 + \sum_{i=1}^{N} \hat{\beta}_i K(\mathbf{x}_i, \mathbf{x})\right]$$

The number of basis functions equals the number of training examples!

- Unless some of the beta's get set to zero…

Gram (Kernel) Matrix

Training Data: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

$$K = \begin{pmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \cdots & K(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ K(\mathbf{x}_N, \mathbf{x}_1) & \cdots & K(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

Properties:

- Positive Definite Matrix
  - Symmetric
  - Positive on diagonal
- N by N

Picking a Model Structure?

- How do you pick the Kernels?
  - Kernel parameters
- These are called **learning parameters** or **hyperparamters**
  - Two approaches choosing learning paramters
    - Bayesian
      - Learning parameters must maximize probability of correct classification on future data based on prior biases
    - Frequentist
      - Use the training data to learn the model parameters $\left(\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_d\right)$
      - Use validation data to pick the best hyperparameters.
- More on learning parameter selection later

Perceptron Algorithm Convergence

- Two problems:
  - No convergence when data is not separable in basis function space
  - Gives infinitely many solutions when data is separable
- Can we modify the algorithm to fix these problems?