CS 4602

Introduction to Machine Learning

Linear Classifiers

Instructor: Po-Chih Kuo

Roadmap

- Introduction and Basic Concepts
- Regression
- Decision Trees
- Bayesian Classifiers
- Linear Classifier
- Neural Networks
- Deep learning
- Convolutional Neural Networks
- RNN/Transformer
- Reinforcement Learning
- Model Selection and Evaluation
- Clustering
- Data Exploration & Dimensionality reduction

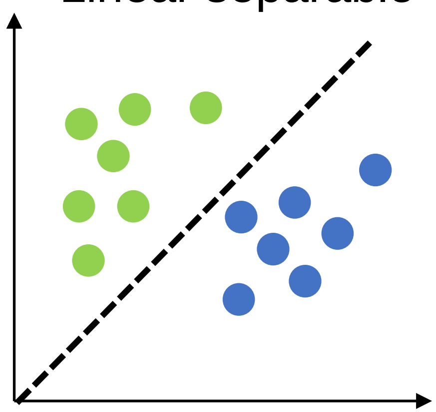
  - Perceptron Algorithm
  - Least-Squares Classifiers
  - Fisher's Linear Discriminant
  - Support Vector Machines

Recall

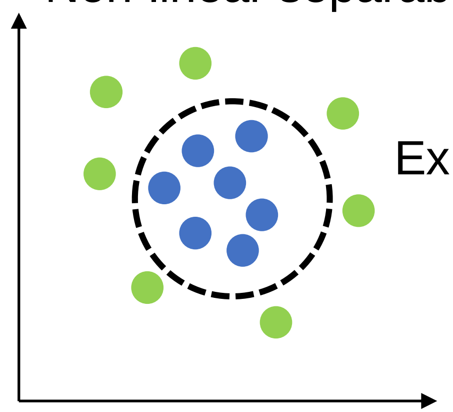
- In classification, the input variable **x** may still be continuous, but the target variable is discrete (blue and green).
  - Linear or Non-linear classifier?

Linear separable

Ex. Decision tree,
Linear classifier



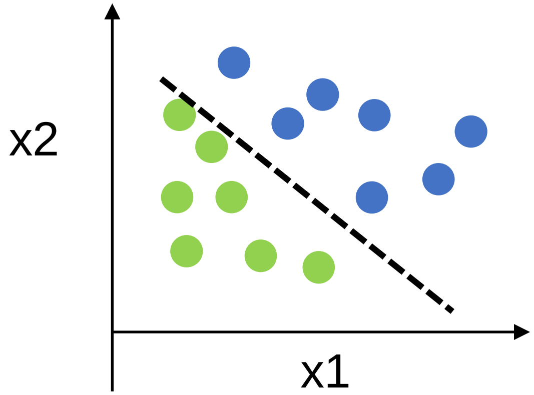
Non-linear separable



Ex. KNN, SVM, DL…

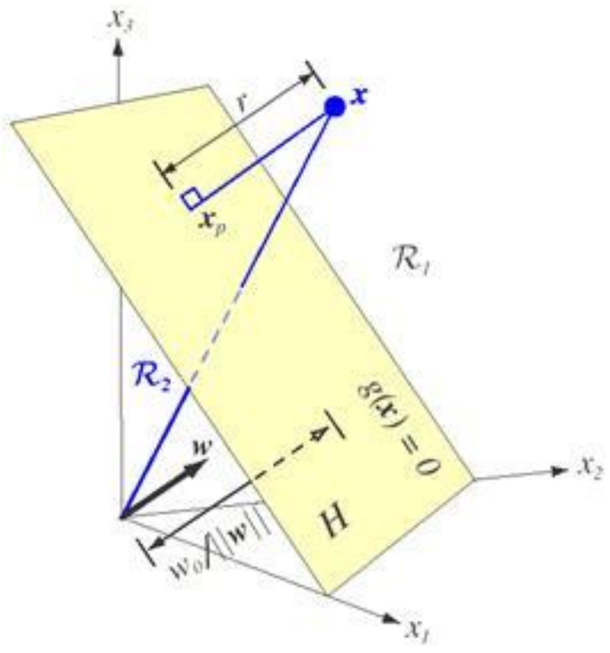
Linear Models for Classification

- Linear models for classification separate input vectors into classes using linear (hyperplane) decision boundaries.
- Example:
  - 2D Input vector x = (x1, x2)
  - Two discrete classes C1 (blue) and C2 (green)



Hyperplane

- Decision boundary g(x) is a hyperplane
- A hyperplane is
  - a point in 1D
  - a line in 2D
  - a plane in 3D



Two-Class Discriminant Function

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

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

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

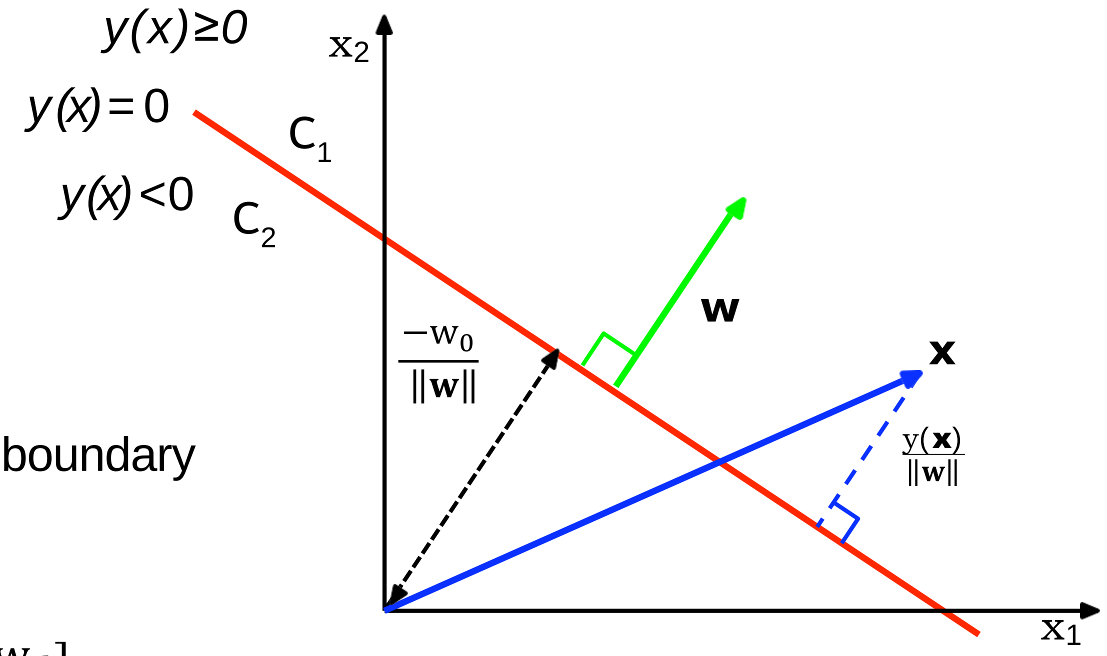
$y(\mathbf{x}) = 0$ defines the decision boundary

Let

$$\mathbf{w} = [w_0 \quad w_1 \quad \dots \quad w_d]$$

$$\mathbf{x} = [1 \quad x_1 \quad \dots \quad x_d]$$

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

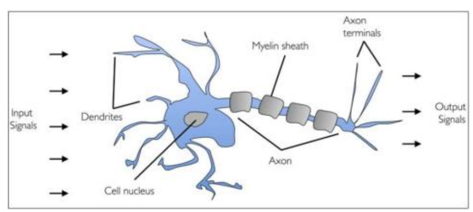


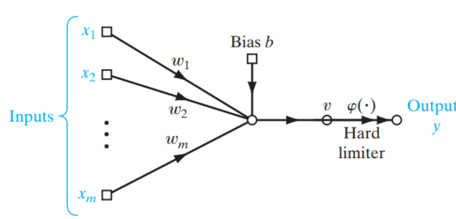
Check list

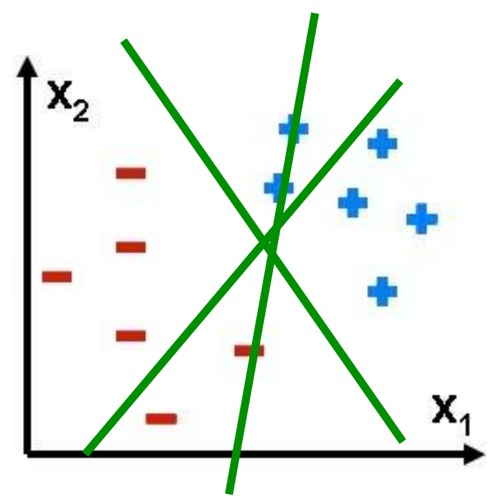
- Perceptron Algorithm
- Least-Squares Classifiers
- Fisher’s Linear Discriminant
- Support Vector Machines

The Perceptron Algorithm

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







Frank Rosenblatt (1928 – 1971)

The Perceptron Algorithm

- we seek w such that

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

- In other words, we would like

$$\mathbf{w}^T\mathbf{x}_n y_n \geq 0, \forall n$$

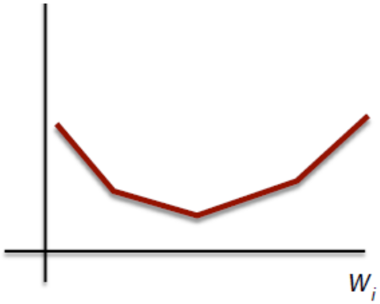
- Thus we seek to minimize

$$E(\mathbf{w}) = -\sum_{n \in M} \mathbf{w}^T\mathbf{x}_n y_n$$

M is the set of misclassified inputs

The Perceptron Algorithm

- E(**w**) is always non-negative.
- E(**w**) is continuing and piecewise linear, and thus easier to minimize.

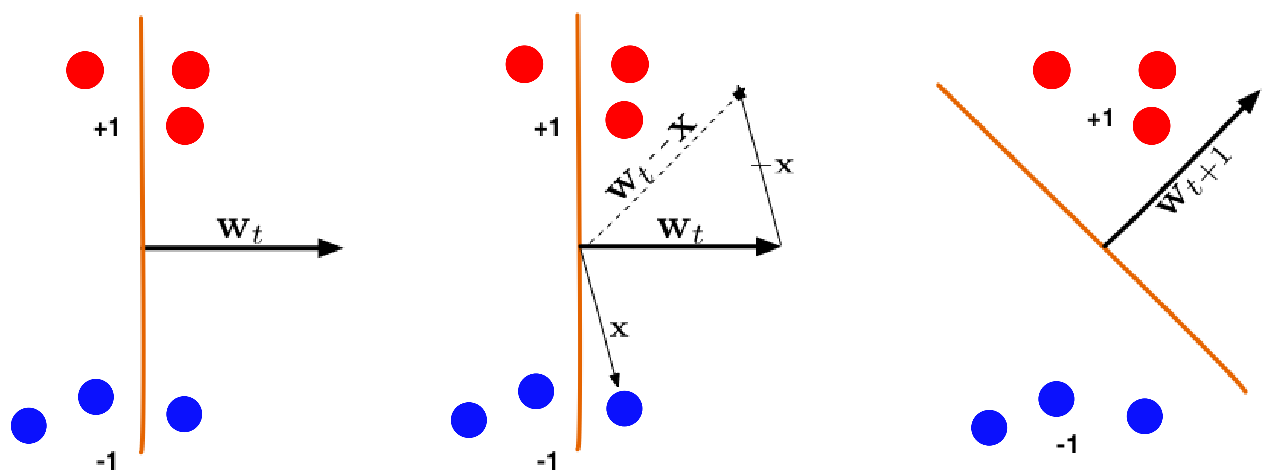


- Again, we can use gradient descent!

$$\mathbf{w}^{\theta+1} = \mathbf{w}^{\theta} - \eta \nabla E(\mathbf{w}) = \mathbf{w}^{\theta} + \eta \sum_{n \in M} \mathbf{x}_n y_n$$

If an input from $C_1(y = +1)$ is misclassified, we need to make its projection on **w** more positive.

Initialize $\vec{w} = \vec{0}$ // Initialize $\vec{w}$. $\vec{w} = \vec{0}$ misclassifies everything.
while TRUE **do** // Keep looping
  $m = 0$ // Count the number of misclassifications, m
  for $(x_i, y_i) \in D$ **do** // Loop over each (data, label) pair in the dataset, D
    if $y_i(\vec{w}^T \cdot \vec{x_i}) \leq 0$ **then** // If the pair $(\vec{x_i}, y_i)$ is misclassified
      $\boxed{\vec{w} \leftarrow \vec{w} + y\vec{x}}$ Gradient descent // Update the weight vector $\vec{w}$
      $m \leftarrow m + 1$ // Counter the number of misclassification
    end if
  end for
  if $m = 0$ **then** // If the most recent $\vec{w}$ gave 0 misclassifications
    break // Break out of the while-loop
  end if
end while // Otherwise, keep looping!

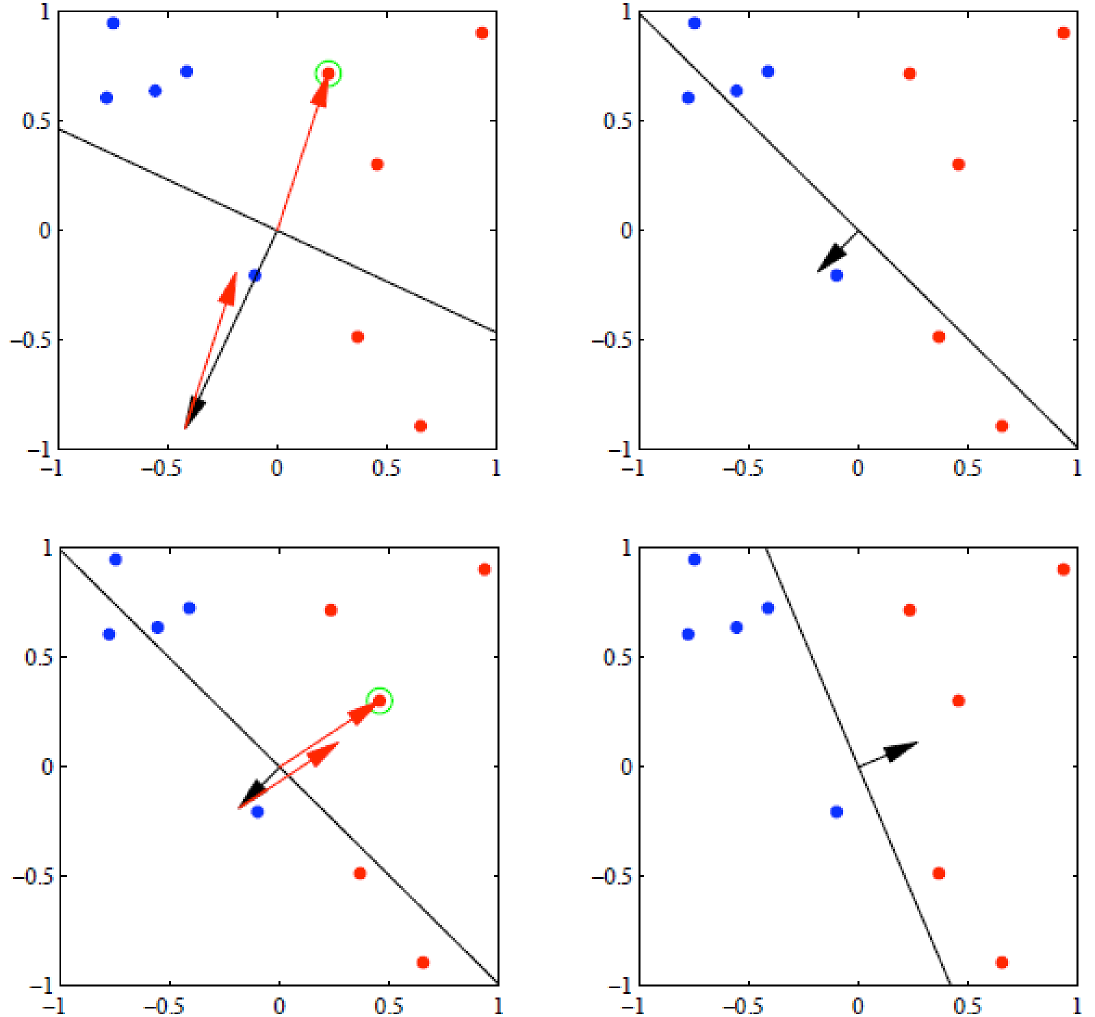


https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote03.html

Example

● +1

● -1



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.
- The perceptron algorithm is not guaranteed to reduce the total error monotonically at each stage.

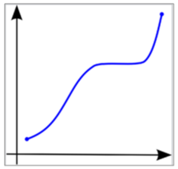


Figure 1 - A monotonically increasing function

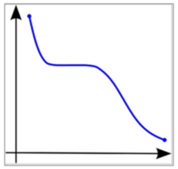


Figure 2 - A monotonically decreasing function

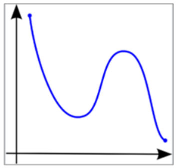


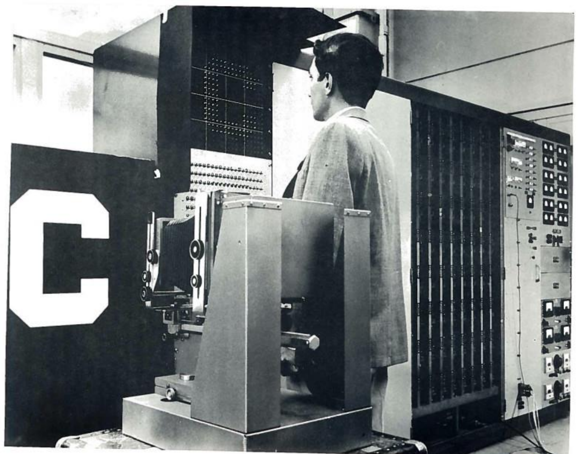
Figure 3 - A function that is not monotonic

Source: https://en.wikipedia.org/wiki/Monotonic_function


Rosenblatt

if the data are linearly separable, then the algorithm is guaranteed to find an exact solution in a finite number of steps

Mark 1 Perceptron Hardware (1960)



THE MARK I PERCEPTRON

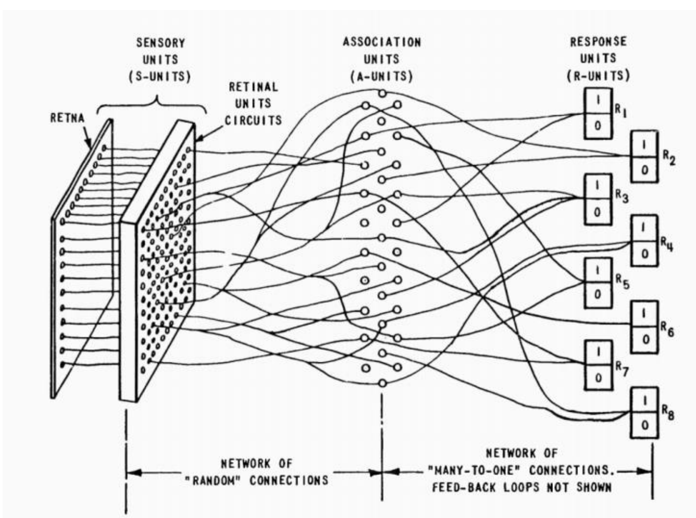
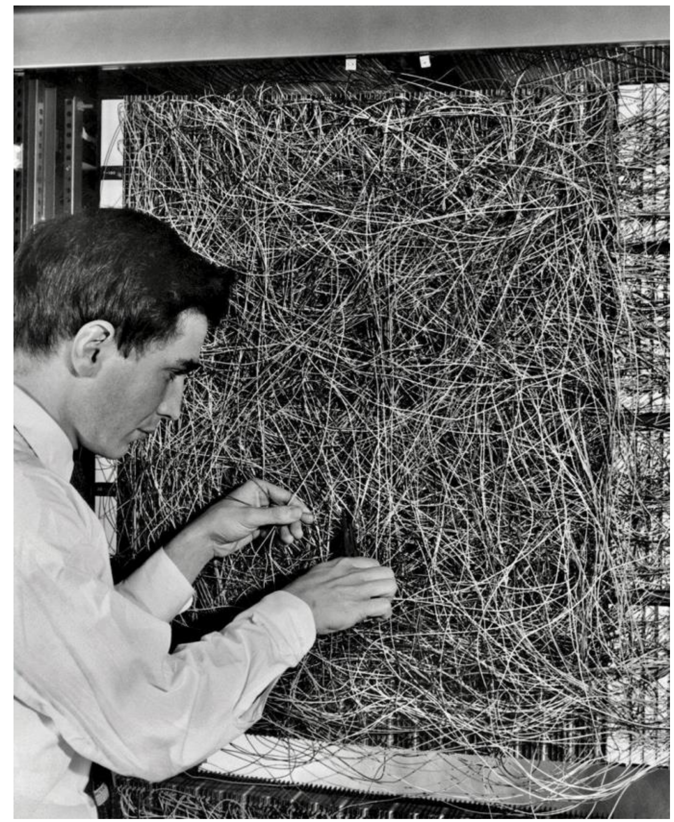


Figure I ORGANIZATION OF THE MARK I PERCEPTRON



Limitation

- Convergence may be slow.
- If the data are not separable, the algorithm will not converge.
- We will know that the data are separable once the algorithm converges.
- The solution will depend upon initialization, scheduling of input vectors, and the learning rate.

Check list

- ~~Perceptron Algorithm~~
- Least-Squares Classifiers
- Fisher’s Linear Discriminant
- Support Vector Machines

Dealing with Non-Linearly Separable Inputs

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

Learning **w** with least-squares

$y(\mathbf{X}) = \mathbf{W}^{\mathrm{T}}\mathbf{X}$

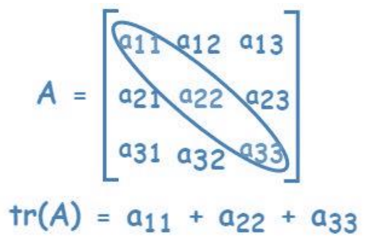
\# of classes

Training dataset $(\mathbf{x}_n, \mathbf{t}_n)$, $n = 1, \ldots, N$, where we use the 1-of-K coding scheme for $\mathbf{t}_n$

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

Let **X** be the $N \times D$ matrix whose n^{th} row is data $\mathbf{x}_n$

Let R(**W**)=**XW**-**T**



Define the error as E(**W**)=$\frac{1}{2}\sum_{i,j} R_{i,j}^2 = \frac{1}{2}\mathrm{Tr}\{\mathrm{R}(\mathbf{W})^{\mathrm{T}}\mathrm{R}(\mathbf{W})\}$

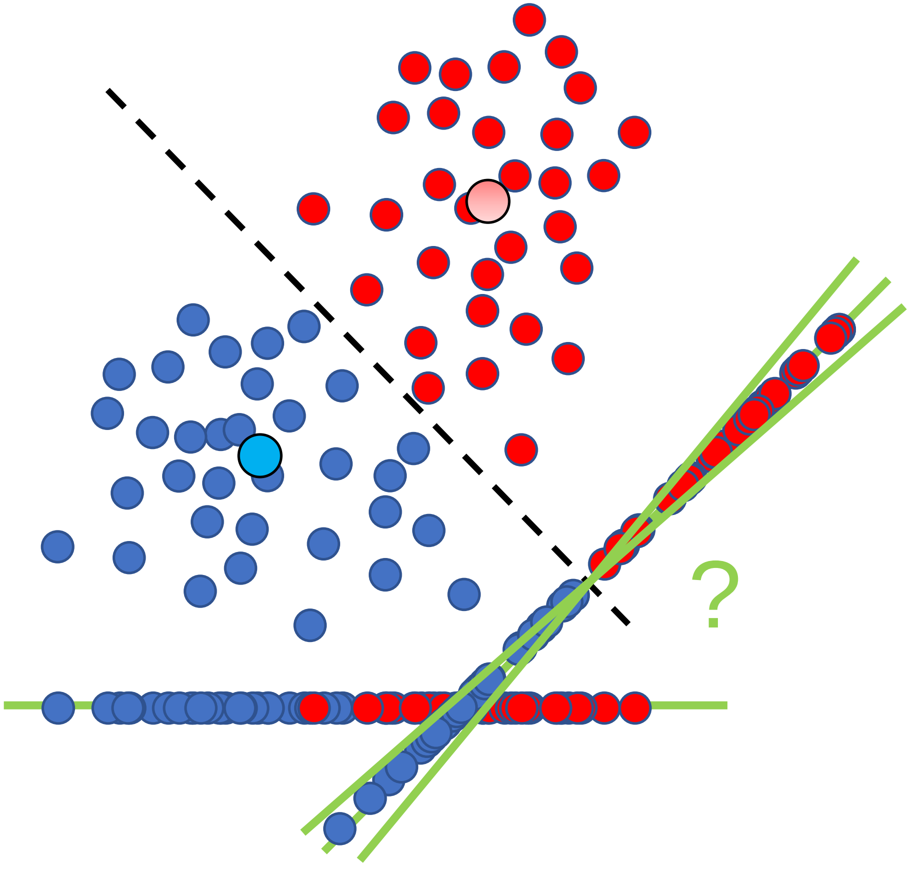
Setting derivative wrt **W** to 0 yields:

$$\mathbf{W} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{T}$$

Check list

- ~~Perceptron Algorithm~~
- ~~Least-Squares Classifiers~~
- Fisher’s Linear Discriminant
- Support Vector Machines

Best separation?

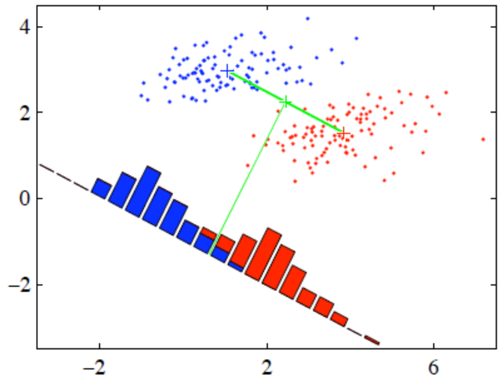


Fisher's Linear Discriminant

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

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

- We might choose **w** to maximize $\mathbf{w}^T(\mathbf{m}_2 - \mathbf{m}_1)$, subject to $||\mathbf{w}||=1$



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

Not optimal if conditional distributions are not isotropic!

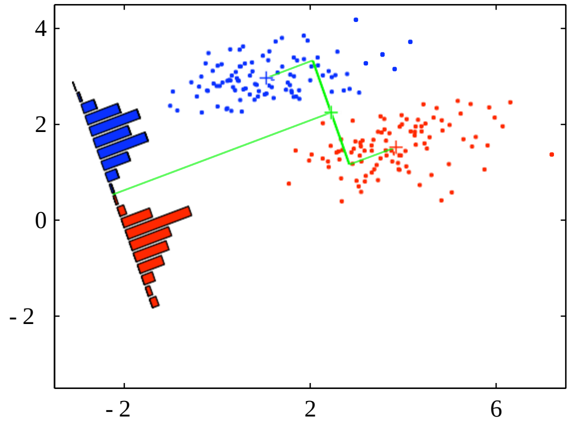
Fisher's Linear Discriminant

Let $m_1 = \mathbf{w}^{\mathrm{T}}\mathbf{m}_1$, $m_2 = \mathbf{w}^{\mathrm{T}}\mathbf{m}_2$ 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 $J(\mathbf{w}) = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{S}_B\mathbf{w}}{\mathbf{w}^{\mathrm{T}}\mathbf{S}_W\mathbf{w}}$



where

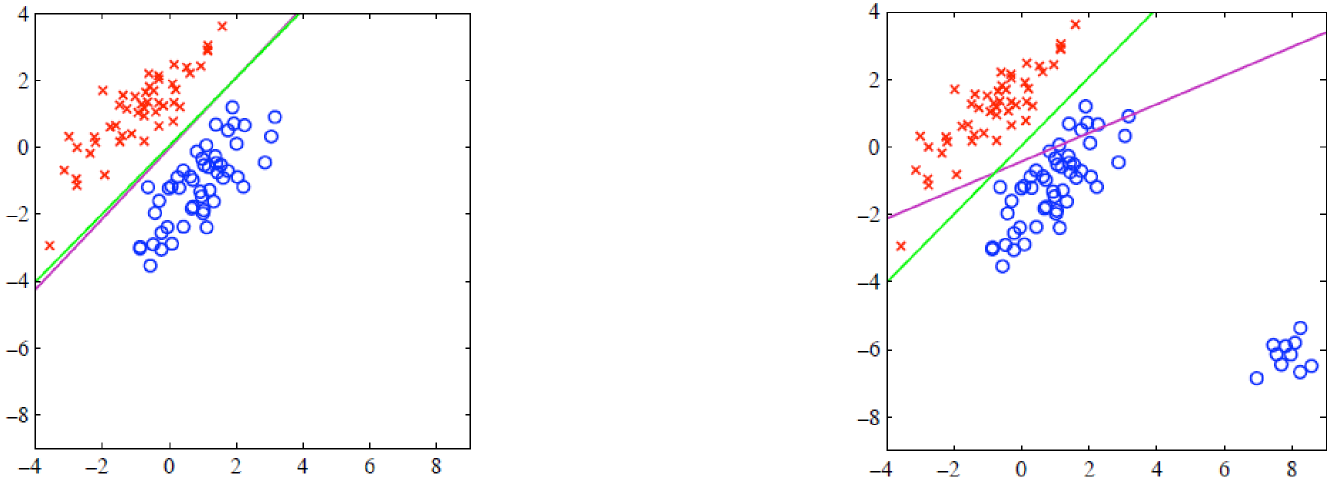
$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^{\mathrm{T}}$ is the between-class variance

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

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

Limitation

- Sensitivity to outliers



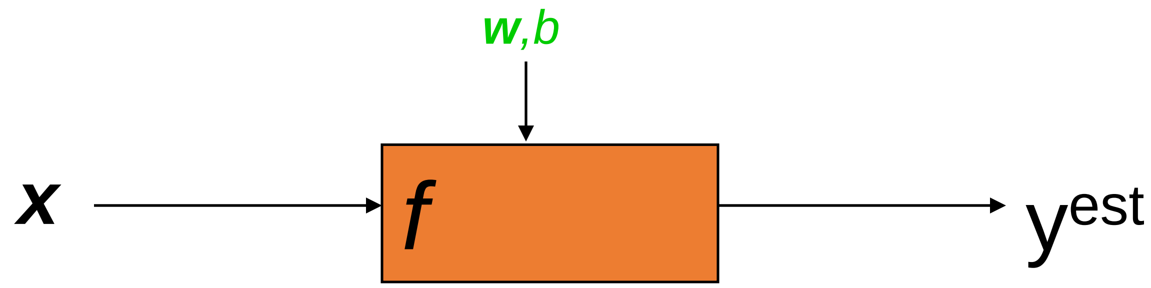
Check list

- ~~Perceptron Algorithm~~
- ~~Least-Squares Classifiers~~
- ~~Fisher’s Linear Discriminant~~
- Support Vector Machines

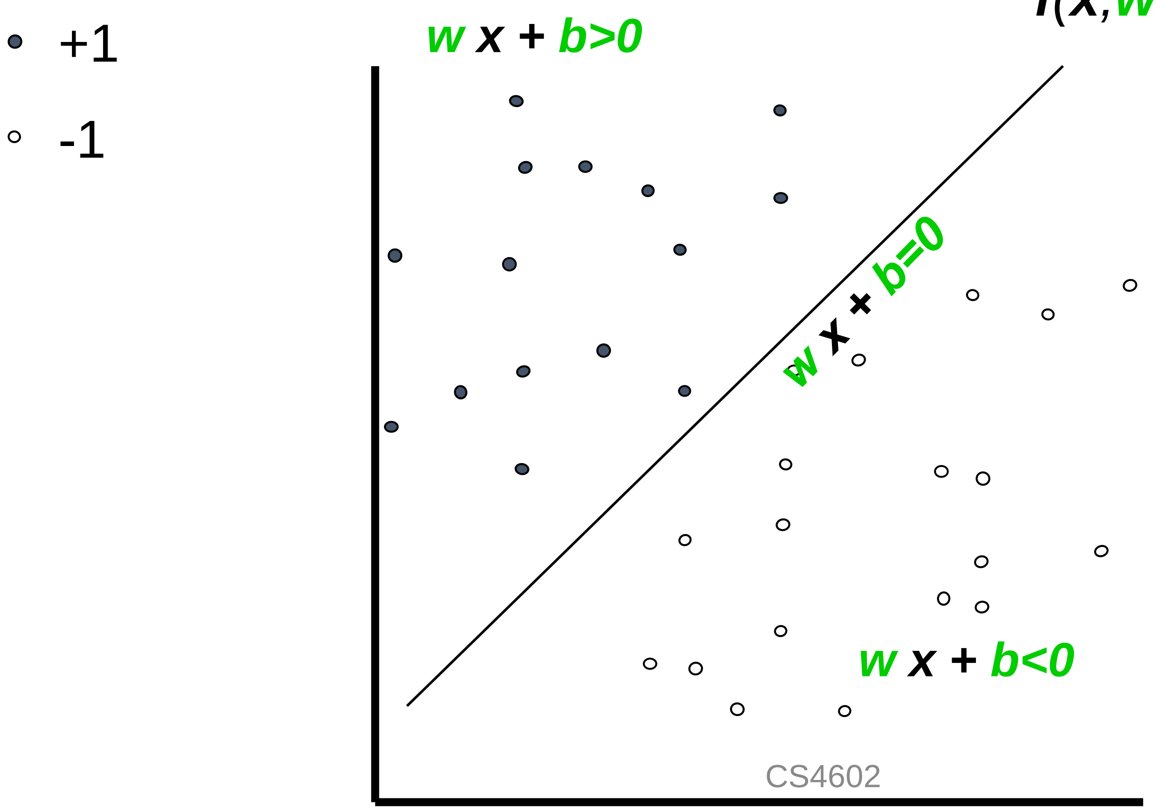
SVM: Motivation

- The perceptron algorithm is guaranteed to provide a linear decision surface that separates the training data, if one exists
- What if it doesn't exist? (back to this later)
- There are an infinite number of solutions, and the solution returned by the perceptron algorithm depends on the <u>initial conditions</u>, the <u>learning rate</u> and the order in which training data are processed.
- While all solutions achieve a <u>perfect</u> score on the training data, they won't all necessarily generalize as well to new inputs.

Linear Classifiers

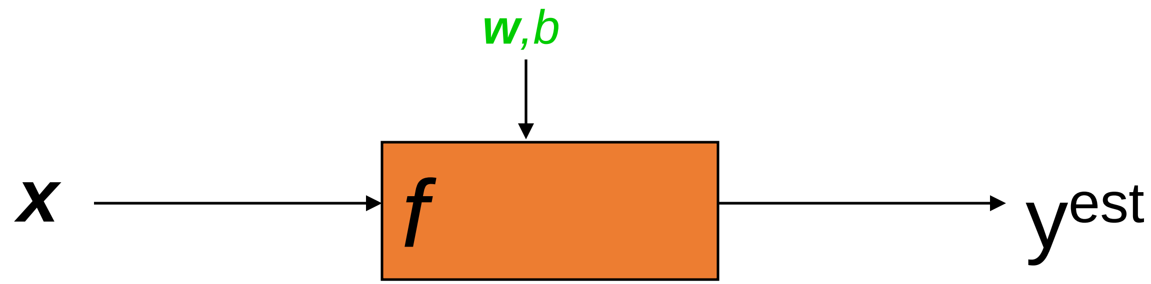


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$

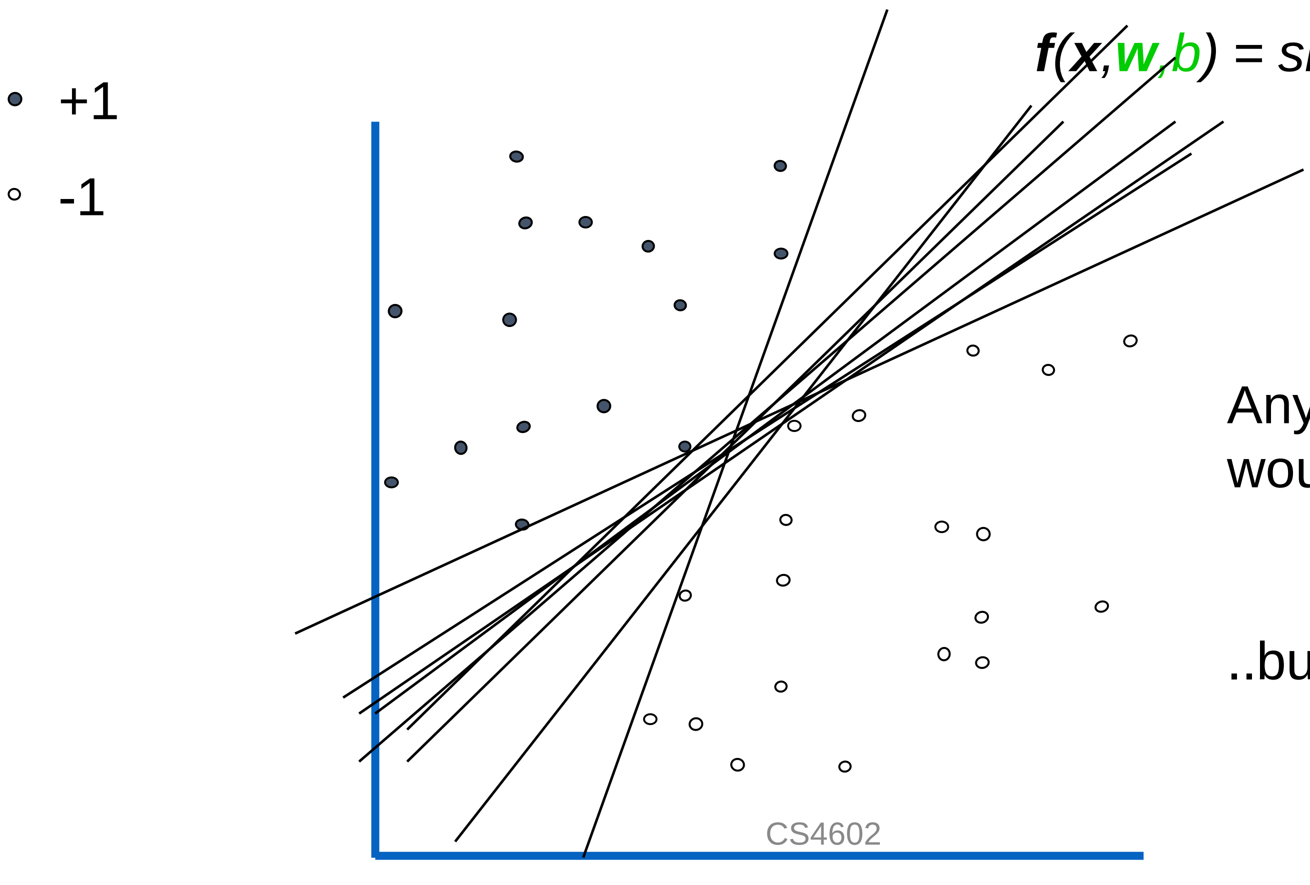


How would you classify this data?

Linear Classifiers



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$



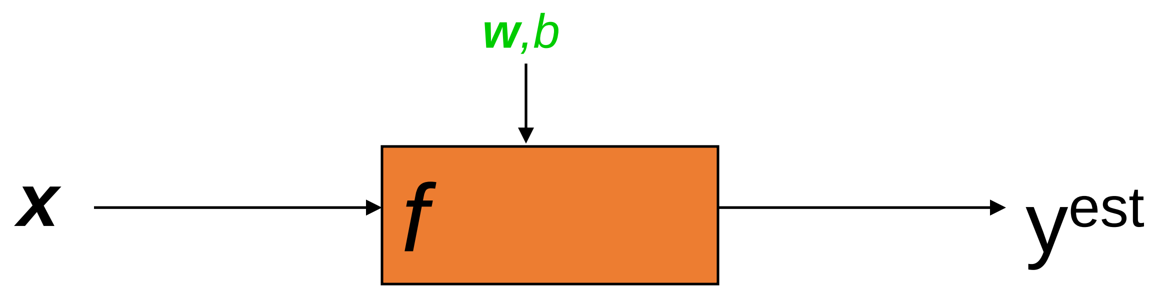
Any of these would be fine..

..but which is best?

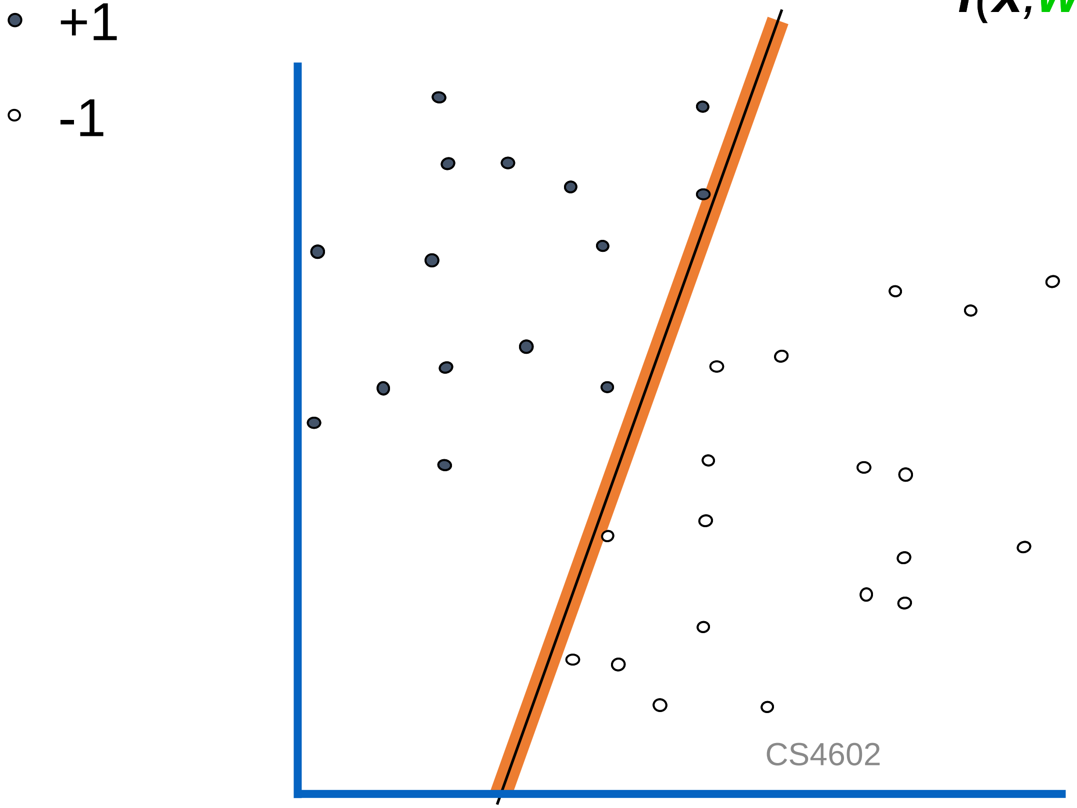
Support Vector Machine

- Which hyperplane to “best” separate data?
- What if it is impossible to separate the data by a hyperplane?

Classifier Margin

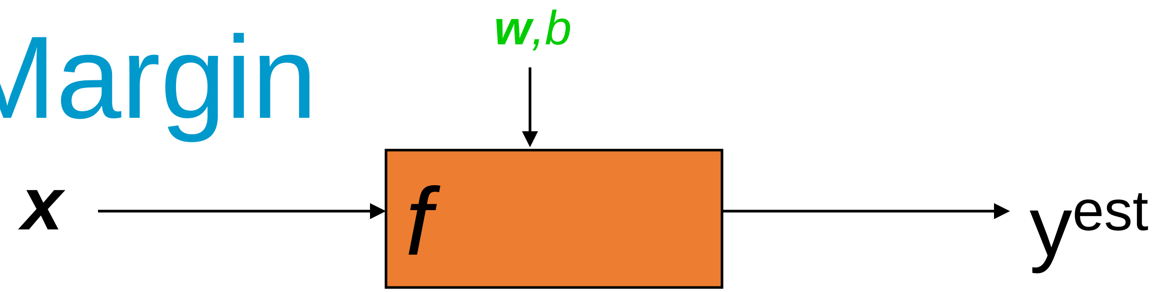


$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$$



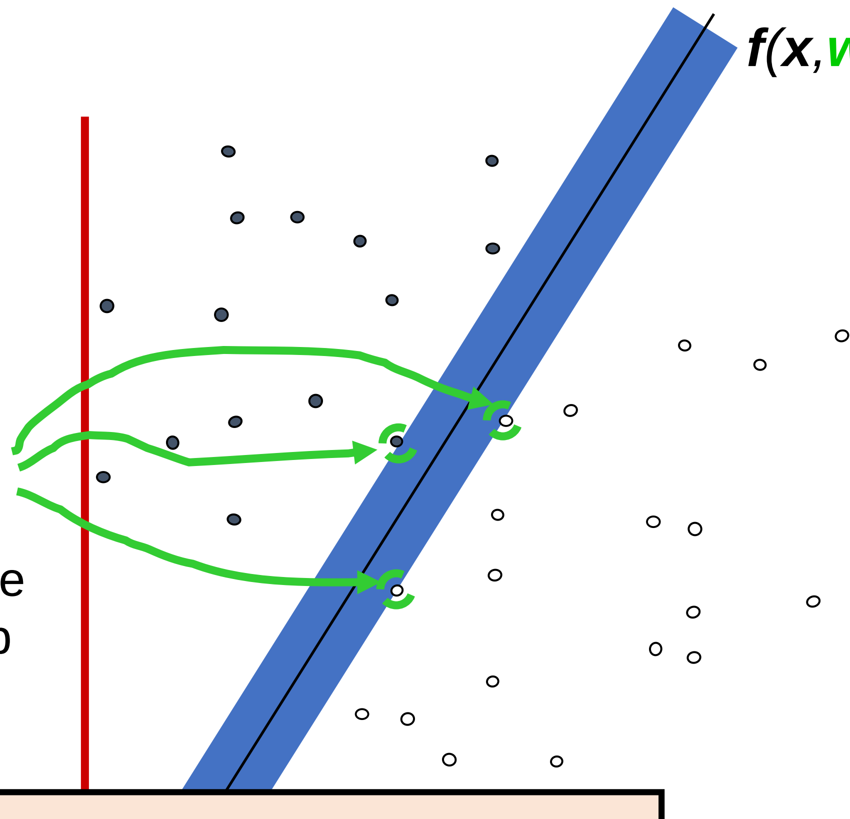
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



• +1

◦ -1



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$

The maximum margin linear classifier is the linear classifier with the maximum margin.

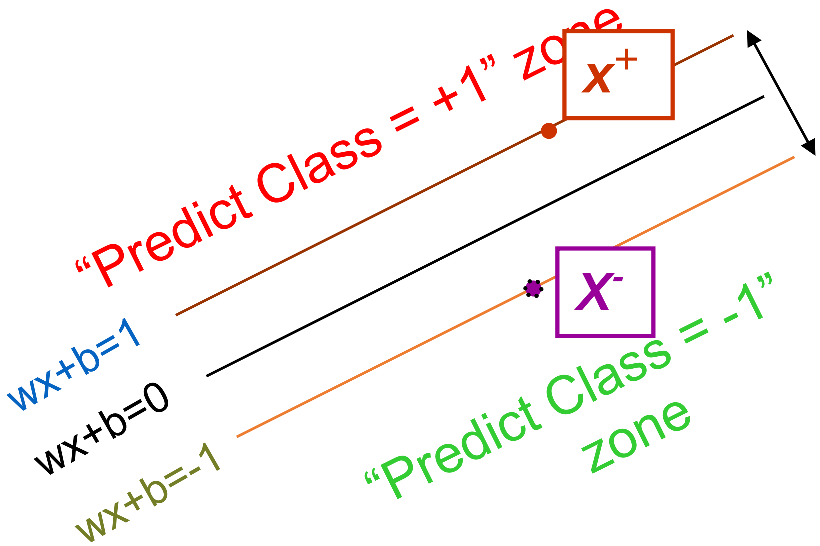
This is the simplest kind of SVM (Called an LSVM)

Linear

Support Vectors are those datapoints that the margin pushes up against

Implies that only support vectors are important; other training examples are ignorable.

Linear SVM Mathematically



M=Margin Width

What we know:

- **w x**$^+$ + b = +1
- **w x**$^-$ + b = -1
- **w** (**x**$^+$ - **x**$^-$)= 2

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

Linear SVM Mathematically

- Goal: **1) Correctly classify all training data**

$$wx_i + b \geq 1 \qquad \text{if } y_i = +1$$

$$wx_i + b \leq -1 \qquad \text{if } y_i = -1$$

$$y_i(wx_i + b) \geq 1 \qquad \text{for all i}$$

2) Maximize the Margin $M = \frac{2}{|w|}$

same as minimize $\frac{1}{2} w^T w$

- **We can formulate a Quadratic Optimization Problem and solve for w and b**

- Minimize $\Phi(w) = \frac{1}{2} w^T w$

  subject to $y_i(wx_i + b) \geq 1 \quad \forall i$

The Dual Problem

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- This is a quadratic programming (QP) problem
  - A global maximum of α_i can always be found
- **w** can be recovered by $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$

Non Support Vector: α_i =0

Making decision

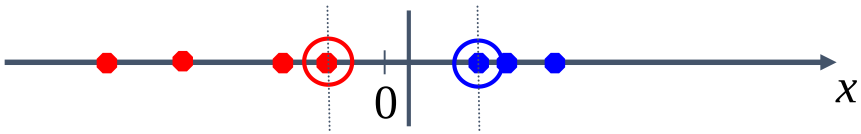
- The decision boundary:
  - $wx_i + b = \sum_{i \in SV} \alpha_i y_i (\mathbf{x}_i{}^T \mathbf{x}) + b$
- The decision:
  - y=sign($\sum_{i \in SV} \alpha_i y_i (\mathbf{x}_i{}^T \mathbf{x}) + b$)

Support Vector Machine

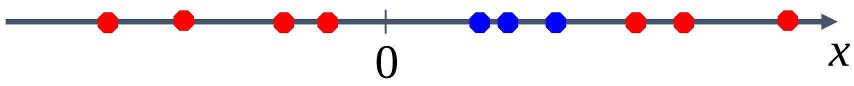
- Which hyperplane to separate data?
- What if it is impossible to separate the data by a hyperplane?

Linear SVM Mathematically

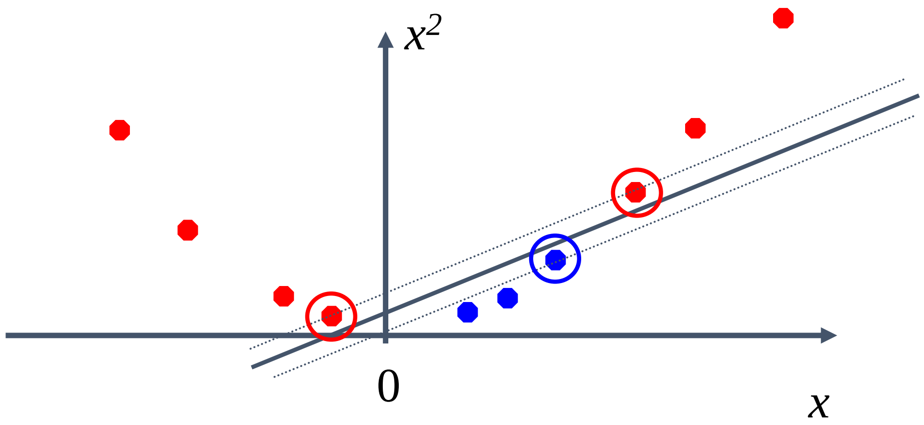
- Datasets that are linearly separable work out great:



- But what are we going to do if the dataset is just too hard?

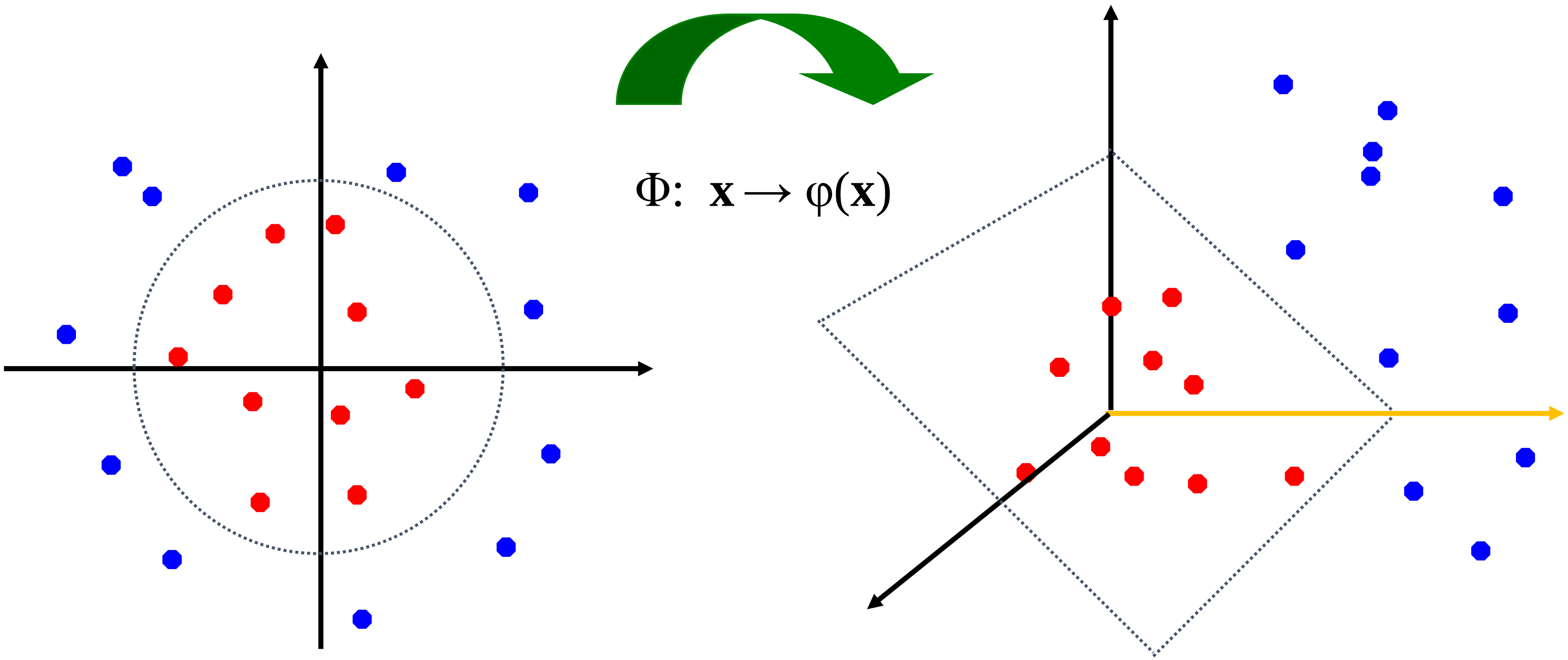


- How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



What if we add a new dimension that represents the distance to the origin?

The “Kernel Trick”

Avoids the **explicit mapping** to learn a nonlinear function or decision boundary!

- The linear classifier relies on dot product between vectors $K(x_i,x_j)=x_i^T x_j$
- If every data point is mapped into high-dimensional space via some transformation Φ: $x \rightarrow \varphi(x)$, the dot product becomes:

$$K(x_i,x_j)= \varphi(x_i)^T \varphi(x_j)$$

- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.
- Example:

2-dimensional vectors $x=[x_1 \;\; x_2]$; let $K(x_i,x_j)=(1 + x_i^T x_j)^2$,

Need to show that $K(x_i,x_j)= \varphi(x_i)^T \varphi(x_j)$:

$$K(x_i,x_j)=(1 + x_i^T x_j)^2,$$

$$= 1+ x_{i1}^2 x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$$

$$= [1 \;\; x_{i1}^2 \;\; \sqrt{2}\, x_{i1}x_{i2} \;\; x_{i2}^2 \;\; \sqrt{2}\, x_{i1} \;\; \sqrt{2}\, x_{i2}]^T [1 \;\; x_{j1}^2 \;\; \sqrt{2}\, x_{j1}x_{j2} \;\; x_{j2}^2 \;\; \sqrt{2}\, x_{j1} \;\; \sqrt{2}x_{j2}]$$

$$= \varphi(x_i)^T \varphi(x_j), \quad \text{where } \varphi(x) = [1 \;\; x_1^2 \;\; \sqrt{2}\, x_1x_2 \;\; x_2^2 \;\; \sqrt{2}\, x_1 \;\; \sqrt{2}\, x_2]$$

Making decision with Kernel

- $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$
- The decision boundary:
  - $wx_i + b = \sum_{i \in SV} \alpha_i y_i ({\mathbf{x}_i}^T \mathbf{x}) + b$
- The decision:
  - y=sign($\sum_{i \in SV} \alpha_i y_i ({\mathbf{x}_i}^T \mathbf{x}) + b$)
- The decision when mapping to feature space:
  - y=sign($\sum_{i \in SV} \alpha_i y_i (\underbrace{\varphi(\mathbf{x}_i)^T \varphi(\mathbf{x})}_{K(\mathbf{x}_i,\mathbf{x}_j)}) + b$)

Examples of Kernel Functions

- Linear: $K(\mathbf{x_i},\mathbf{x_j}) = \mathbf{x_i}^T\mathbf{x_j}$

- Polynomial of power *p*: $K(\mathbf{x_i},\mathbf{x_j}) = (1+ \mathbf{x_i}^T\mathbf{x_j})^p$

- Gaussian (radial-basis function network):

$$K(\mathbf{x_i},\mathbf{x_j}) = \exp\left(-\frac{\left\|\mathbf{x_i}-\mathbf{x_j}\right\|^2}{2\sigma^2}\right)$$

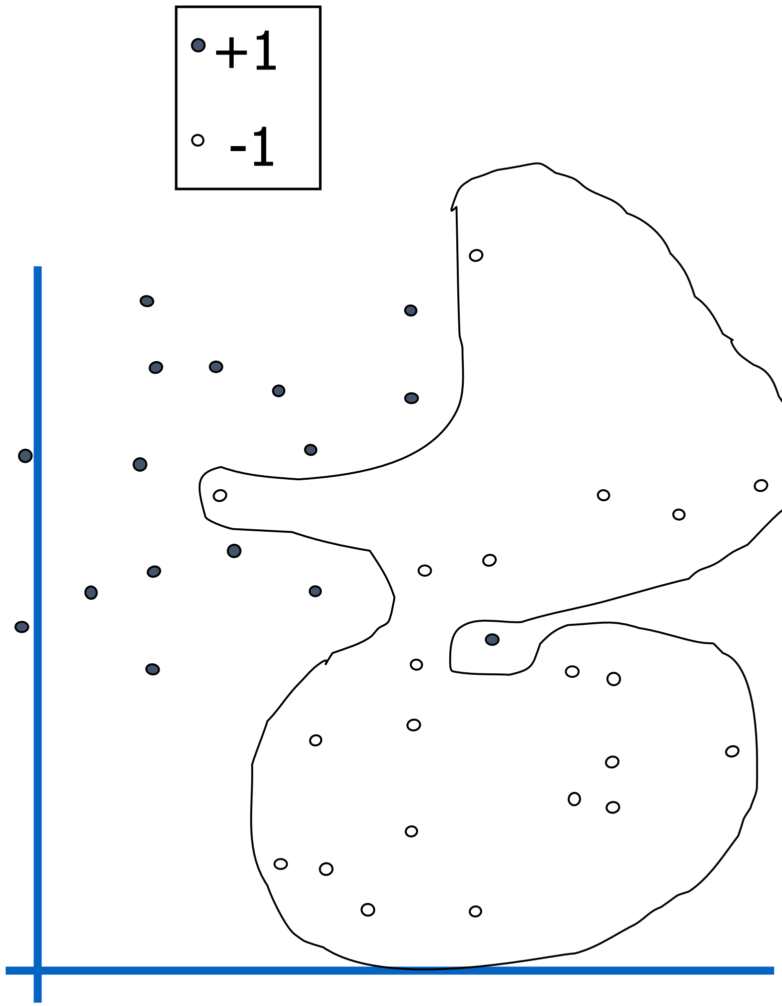
- Sigmoid: $K(\mathbf{x_i},\mathbf{x_j}) = \tanh(\beta_0\mathbf{x_i}^T\mathbf{x_j} + \beta_1)$

Nonlinear!

Nonlinear SVM

- SVM locates a separating hyperplane in the feature space and classify points in that space.
- It does not need to represent the space explicitly, simply by defining a kernel function.
- The kernel function plays the role of the dot product in the feature space.

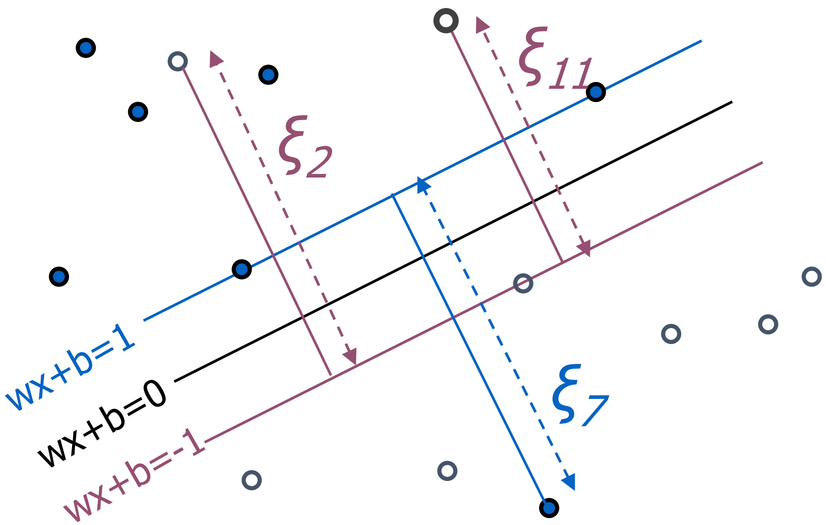
Data noise



- **Hard Margin:**
  - We require all data points be classified correctly
  - No training error
- **What if the training set is noisy?**
  - use very powerful kernels

Soft Margin Classification

Slack variables ξi can be added to allow misclassification of difficult or noisy examples.



What should our quadratic optimization criterion be?

Minimize

$$\frac{1}{2}w^T w + C\sum_{k=1}^{M}\xi_k$$

- ξ_i = *0 for data points that are correctly classified*
- $\xi_i = |t_i - y(x_i)|$ *for other points:*
  - ξ_i = *1 for data points on the decision boundary* $y(x_i) = 0$
  - ξ_i *> 1 for data points that are misclassified*

- The old formulation:

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi(w)} = ½\, \mathbf{w}^T\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
$y_i\, (\mathbf{w^T x_i} + b) \geq 1$

- The new formulation incorporating slack variables:

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi(w)} = ½\, \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
$y_i\, (\mathbf{w^T x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

- Parameter *C* can be viewed as a way to control overfitting.

Some Issues

- Choice of kernel
  - Gaussian or polynomial kernel is default
- Choice of kernel parameters
  - e.g. σ in Gaussian kernel
  - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.
- Optimization criterion – Hard margin v.s. Soft margin
  - a lengthy series of experiments in which various parameters are tested

Questions?

https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.07-Support-Vector-Machines.ipynb#scrollTo=pYH2boXfHt-O