# ML Framework: From Lines to Perceptrons

**Dr. Fayyaz Minhas**

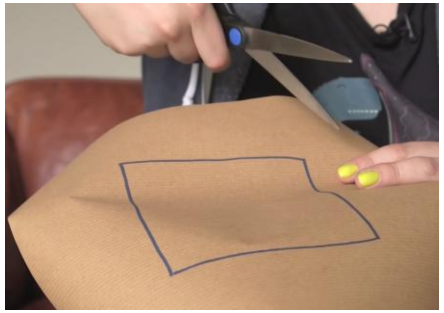
Department of Computer Science
University of Warwick
https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs909/

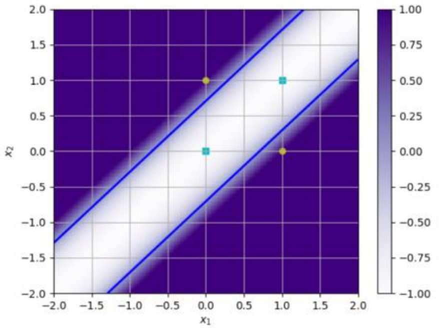
# Intuition of Linear Discriminants

- Paper Cutting

Fold and Cut Theorem Video:
https://www.youtube.com/watch?v=ZREp1mAPKTM

- How can we solve non-linear classification?
  - By folding the space on which examples lie and then making a single straight cut
    - Notice how folding changes the distance between points
  - How to achieve such folding?
    - One way is to transform the data






https://www.youtube.com/watch?v=ZREp1mAPKTM

The Fold-and-Cut Theorem implies that any pattern can be achieved with a single straight cut if the paper (or space) is folded appropriately.

***Thus, it is theoretically possible to partition any space into regions containing positive and negative training examples no matter how complex such a boundary is by simply folding the feature space appropriately and using a linear classifier (single straight cut).***

https://en.wikipedia.org/wiki/Fold-and-cut_theorem

# Preliminaries and Intuition

# Preliminaries

- Equations of lines and their properties

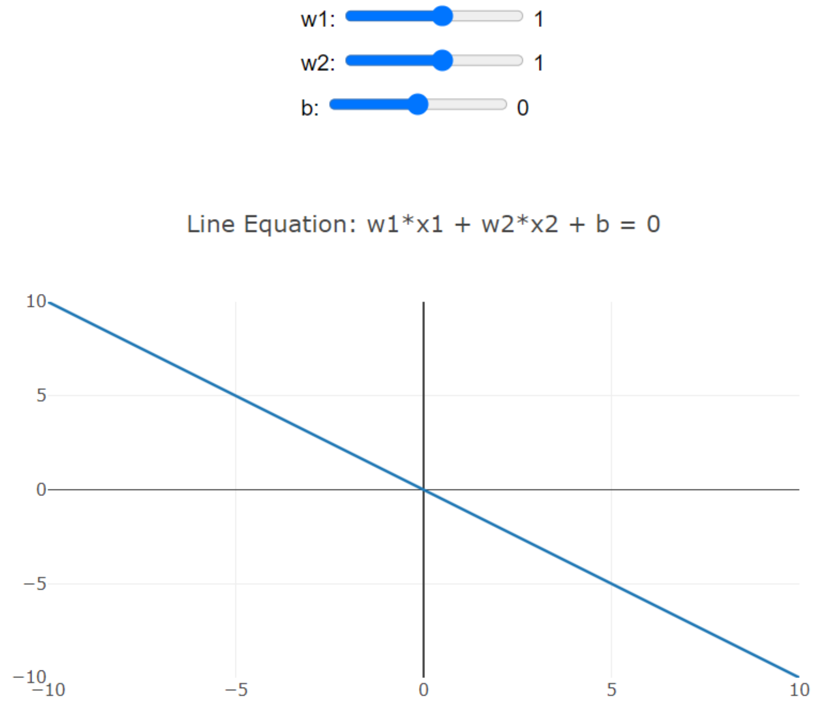
$$w_1 x_1 + w_2 x_2 + b = 0$$

$$x_2 = \frac{-w_1}{w_2} x_1 + \frac{-b}{w_2}$$

$$x_2 = m x_1 + c$$

$$f(\boldsymbol{x}; \boldsymbol{w}) = w_1 x_1 + w_2 x_2 + b = 0$$

$$f(\boldsymbol{x}; \boldsymbol{w}) = [w_0 \quad w_1 \quad w_2] \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} = \boldsymbol{w}^T \boldsymbol{x} = 0$$




If a point $x = (x_1, x_2)$ is on the line then $f(x; \boldsymbol{w}) = w_1 x_1 + w_2 x_2 + b = 0$
If it is above the line then $f(x; \boldsymbol{w}) = w_1 x_1 + w_2 x_2 + b > 0$
If it is below the line then $f(x; \boldsymbol{w}) = w_1 x_1 + w_2 x_2 + b < 0$

https://foxtrotmike.github.io/CS909/lines.html

# Preliminaries

- Distance function

$$d(\boldsymbol{a},\boldsymbol{b}) = \|\boldsymbol{a}-\boldsymbol{b}\| = \sqrt{(\boldsymbol{a_1}-\boldsymbol{b_1})^2 + (\boldsymbol{a_2}-\boldsymbol{b_2})^2}$$

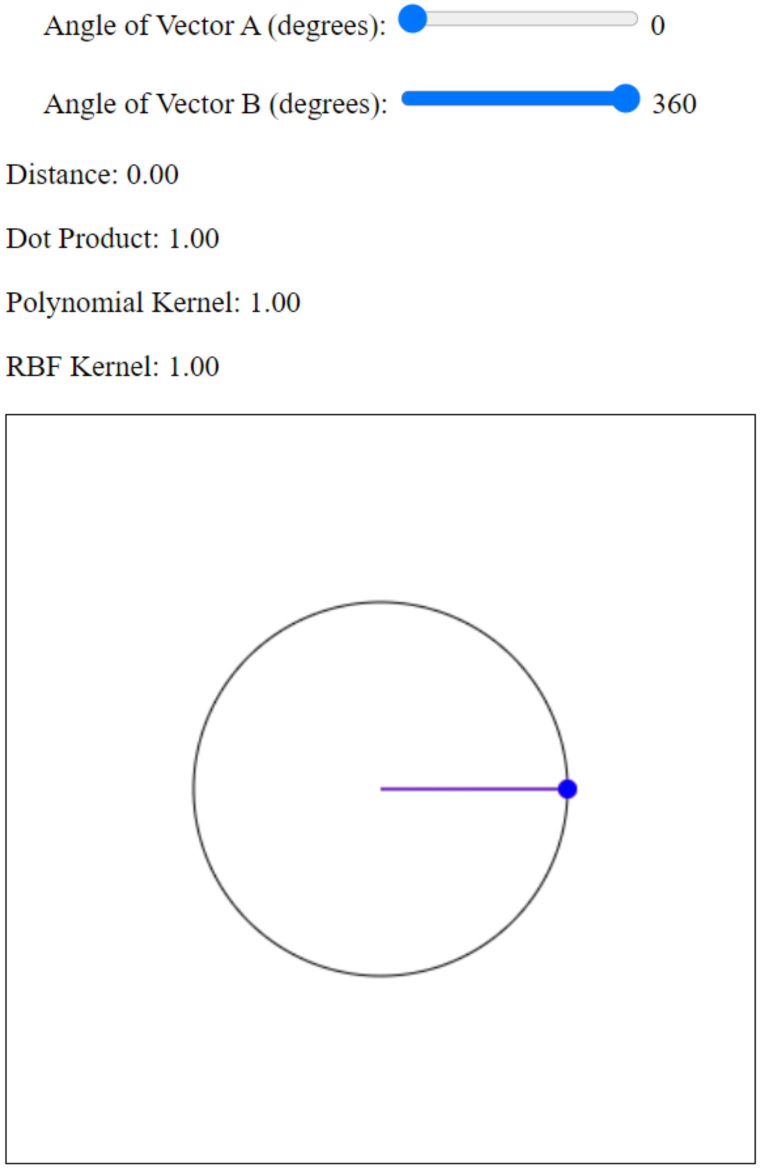
- Relation with Dot Product

$$d^2(\boldsymbol{a},\boldsymbol{b}) = \|\boldsymbol{a}-\boldsymbol{b}\|^2 = (\boldsymbol{a_1}-\boldsymbol{b_1})^2 + (\boldsymbol{a_2}-\boldsymbol{b_2})^2 = \boldsymbol{a}^T\boldsymbol{a} + \boldsymbol{b}^T\boldsymbol{b} - 2\boldsymbol{a}^T\boldsymbol{b}$$

- If $\boldsymbol{a}, \boldsymbol{b}$ are unit vectors (i.e., $\boldsymbol{a}^T\boldsymbol{a} = \|\boldsymbol{a}\|^2 = \boldsymbol{b}^T\boldsymbol{b} = \|\boldsymbol{b}\|^2 = 1$) then: $d^2(\boldsymbol{a},\boldsymbol{b}) = 2 - 2(\boldsymbol{a}^T\boldsymbol{b})$

- **Or the farther away or different two points are, the lower their dot product and vice-versa**

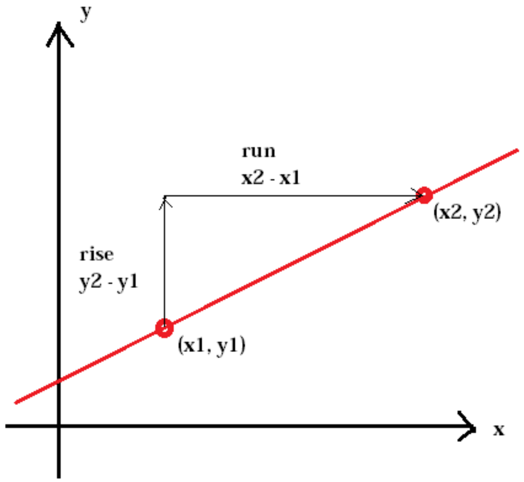
- **We can also have more generalized dot products called kernel functions that can measure similarity between two objects in a different way**
  - Linear kernel: $k(\boldsymbol{a},\boldsymbol{b}) = \boldsymbol{a}^T\boldsymbol{b}$
  - Polynomial kernel: $k(\boldsymbol{a},\boldsymbol{b}) = (\boldsymbol{a}^T\boldsymbol{b})^2$
  - Gaussian or Radial Basis Function (RBF) Kernel: $k(\boldsymbol{a},\boldsymbol{b}) = \exp(-\lambda\|\boldsymbol{a}-\boldsymbol{b}\|^2)$
  - Mahalanobis Kernel: $k(\boldsymbol{a},\boldsymbol{b};\boldsymbol{M}) = \exp\left(-(\boldsymbol{a}-\boldsymbol{b})^T\boldsymbol{M}(\boldsymbol{a}-\boldsymbol{b})\right)$
  - Exponential kernel: $\mathrm{k}(\boldsymbol{a},\boldsymbol{b};\boldsymbol{W_a},\boldsymbol{W_b}) = exp\left(\frac{1}{\sqrt{d}}\langle \boldsymbol{aW_a}, \boldsymbol{bW_b}\rangle\right)$
    - Asymmetric, non-mercer kernels

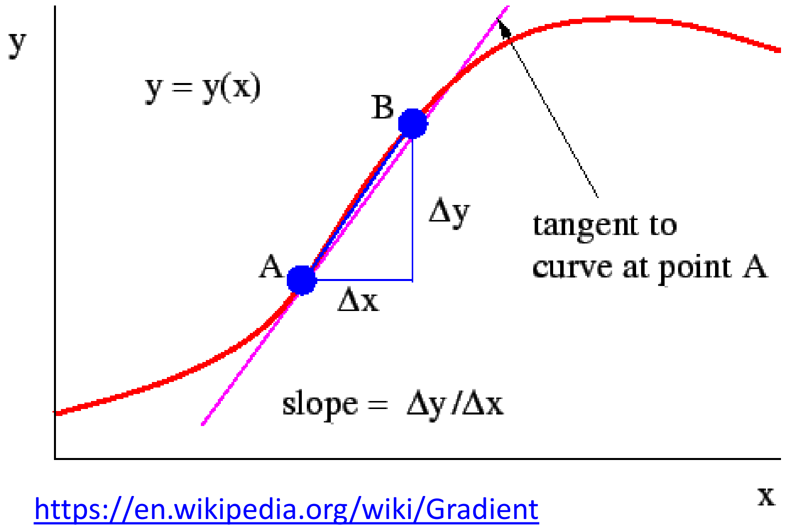



https://foxtrotmike.github.io/CS909/distance_dot.html

# Preliminaries

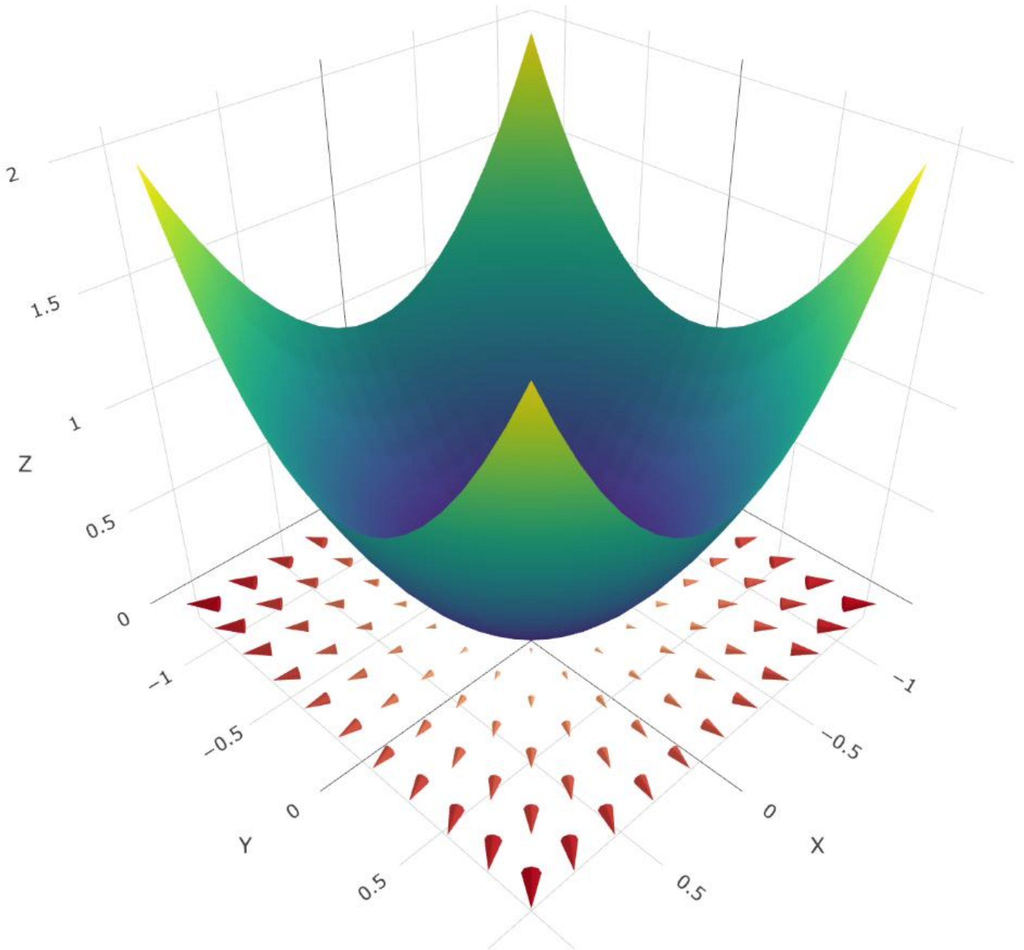
- Gradients




https://en.wikipedia.org/wiki/Gradient

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x^{(1)}} \\ \frac{\partial f(\boldsymbol{x})}{\partial x^{(2)}} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{\partial x^{(d)}} \end{bmatrix}$$




$$f(x,y) = x^2 + y^2 \text{ and } \nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

https://foxtrotmike.github.io/CS909/gradients.html

# Preliminaries: Finding minima and maxima of functions

- Given a function f(w)
- Take the derivative
- Substitute the derivative to zero
- Solve for $x$ when $\frac{df}{dw} = 0$

$$f(w) = (w - 0.5)^2$$
$$\frac{df}{dw} = 2(w - 0.5) = 0$$
$$w^* = 0.5$$

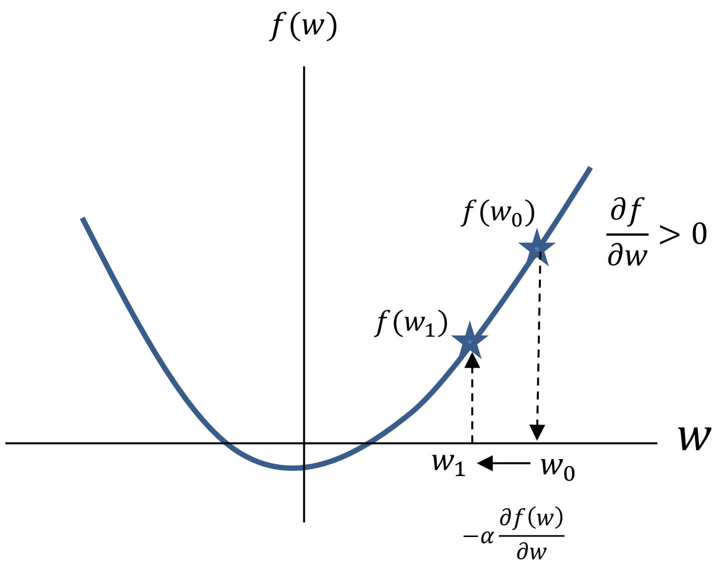
- Works when we can solve for w

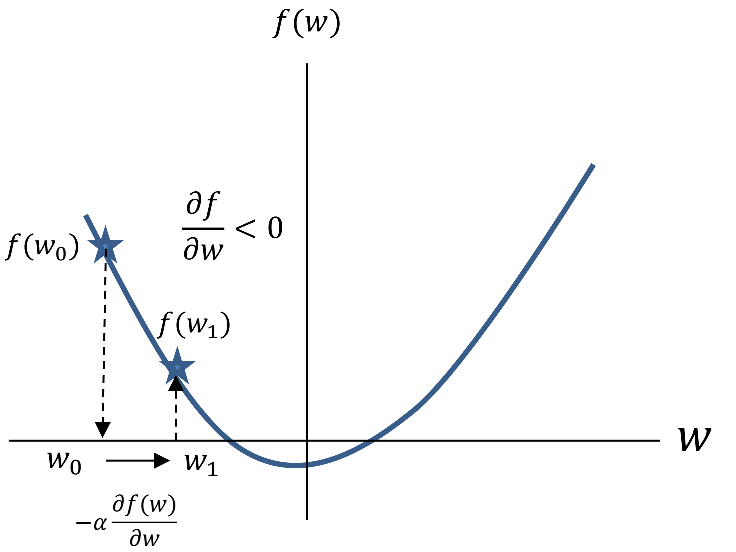
$$f(w) = (w - 0.5)^2 + sin(4w)$$
$$\frac{df}{dw} = 2(w - 0.5) + 4\cos(4w) = 0$$
$$w^* = ?$$

# Preliminaries: Gradient Descent

- In order to find the minima of a function, keep taking steps along a direction opposite to the gradient of the function

$$\boldsymbol{w}^{(\boldsymbol{k}+\mathbf{1})} = \boldsymbol{w}^{(\boldsymbol{k})} - \alpha \nabla f(\boldsymbol{w}^{(\boldsymbol{k})})$$



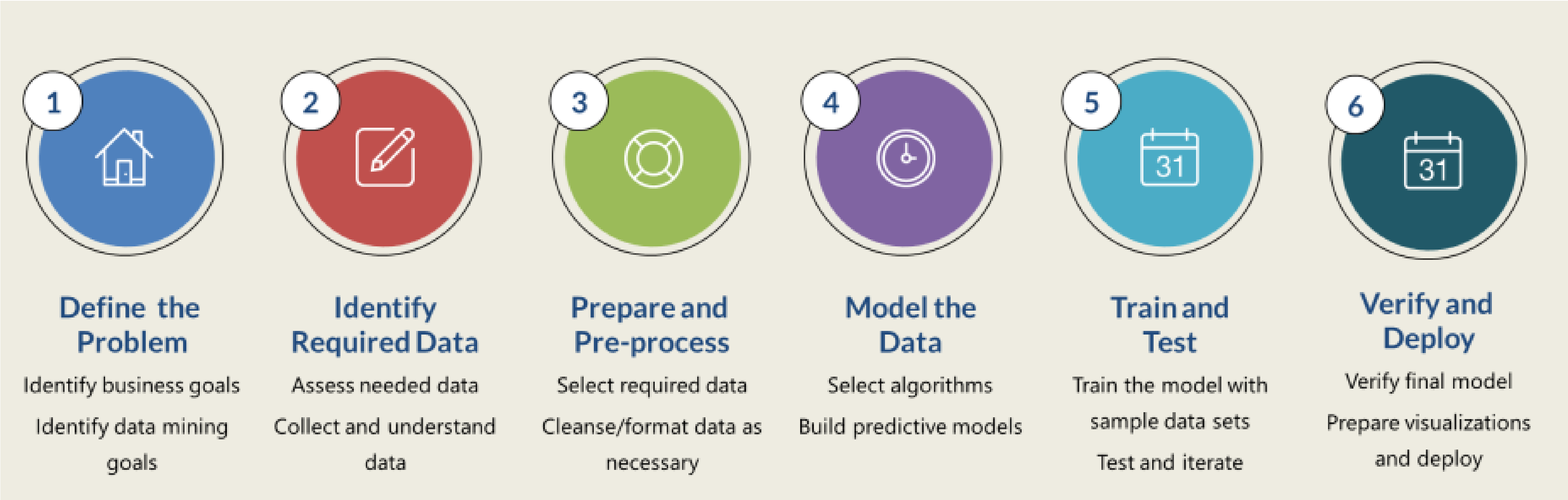
# GD Implementation

```
import numpy as np

def gd(fxn,dfxn,w0=0.0,lr = 0.01,eps=1e-4,nmax=1000, history = True):
    """
    Implementation of a gradient descent solver.
        fxn: function returns value of the target function for a given w
        dfxn: gradient function returns the gradient of fxn at w
        w0: initial position [Default 0.0]
        lr: learning rate [0.001]
        eps: min step size threshold [1e-4]
        nmax: maximum number of iters [1000]
        history: whether to store history of x or not [True]
    Returns:
        w: argmin_x f(w)
        converged: True if the final step size is less than eps else false
        H: history
    """
    H = []
    w = w0
    if history:
        H = [[w,fxn(w)]]
    for i in range(nmax):
        dw = -lr*dfxn(w) #gradient step
        if np.linalg.norm(dw)<eps: # we have converged
            break
        if history:
            H.append([w+dw,fxn(w+dw)])
        w = w+dw #gradient update
    converged = np.linalg.norm(dw)<eps
    return w,converged,np.array(H)
```

```
if __name__=='__main__':
    import matplotlib.pyplot as plt
    def myfunction(w):
        z = (w-0.5)**2#+np.sin(4*w)
        return z
    def mygradient(w):
        dz = 2*(w-0.5)#+4*np.cos(4*w)
        return dz


    wrange = np.linspace(-3,3,100)
    #select random initial point in the range
    w0 = np.min(wrange)+(np.max(wrange)-np.min(wrange))*np.random.rand()

    w,c,H = gd(myfunction,mygradient,w0=w0,lr = 0.01,eps=1e-4,nmax=1000, history = True)

    plt.plot(wrange,myfunction(wrange)); plt.plot(wrange,mygradient(wrange));
    plt.legend(['f(w)','df(w)'])
    plt.xlabel('w');plt.ylabel('value')
    s = 'Convergence in '+str(len(H))+' steps'
    if not c:
        s = 'No '+s
    plt.title(s)
    plt.plot(H[0,0],H[0,1],'ko',markersize=10)
    plt.plot(H[:,0],H[:,1],'r.-')
    plt.plot(H[-1,0],H[-1,1],'k*',markersize=10)
    plt.grid(); plt.show()
```

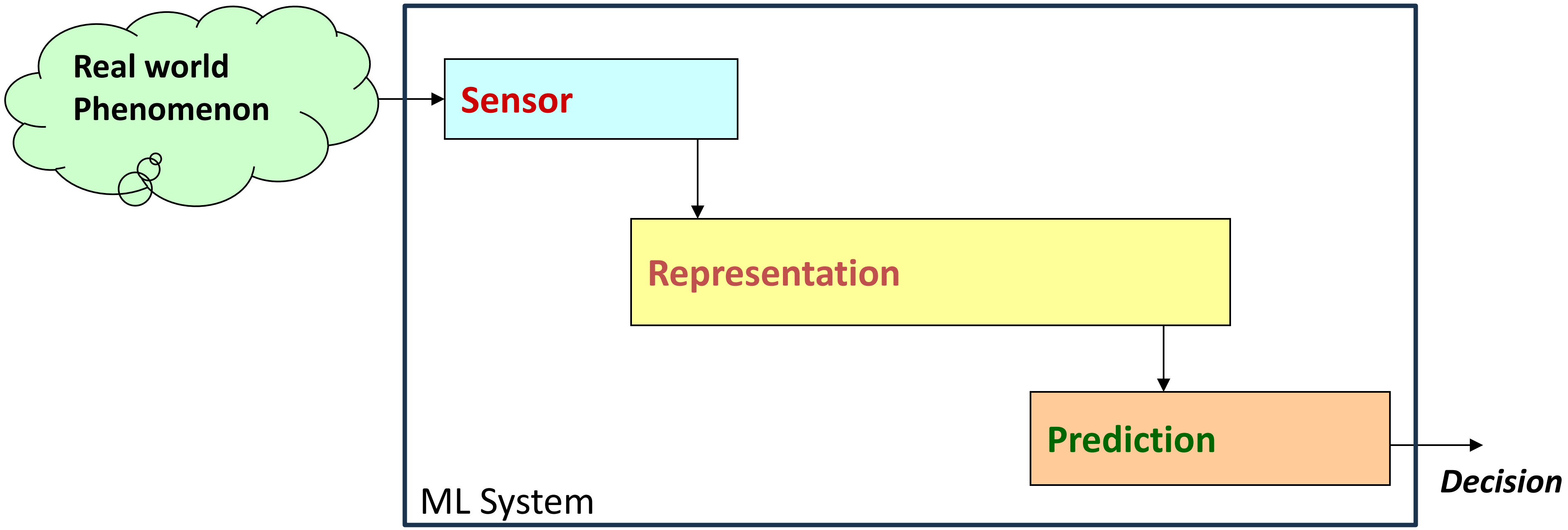
https://github.com/foxtrotmike/CS909/blob/master/gd.py
https://github.com/foxtrotmike/CS909/blob/master/dm_lab_2_fm.ipynb

# Steps in the development of a data science model



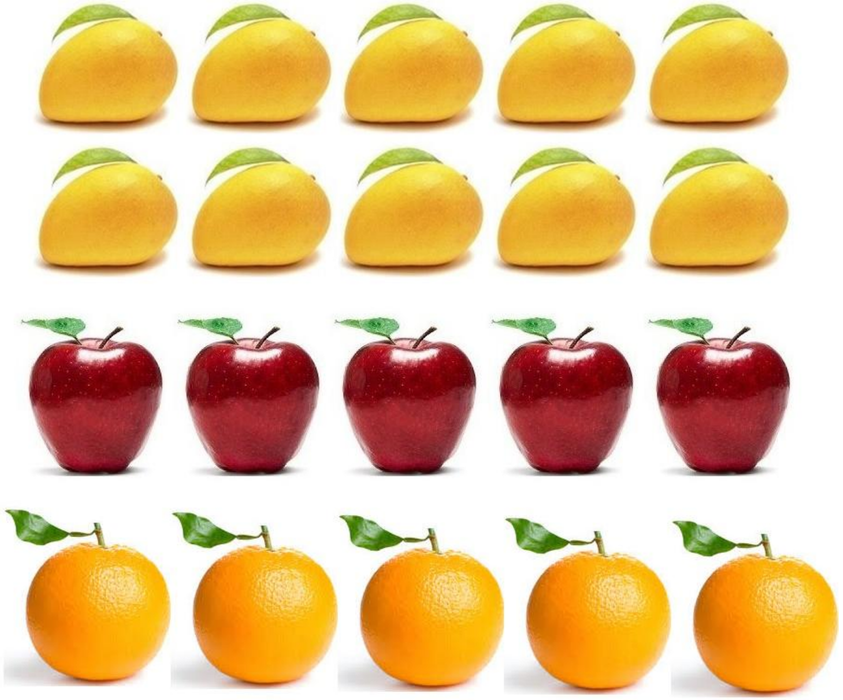
# Constructs of a Data Mining System for Prediction

- Identify the objective
  – Identify the unit of classification (example)
    - Image block, protein sequence, ….

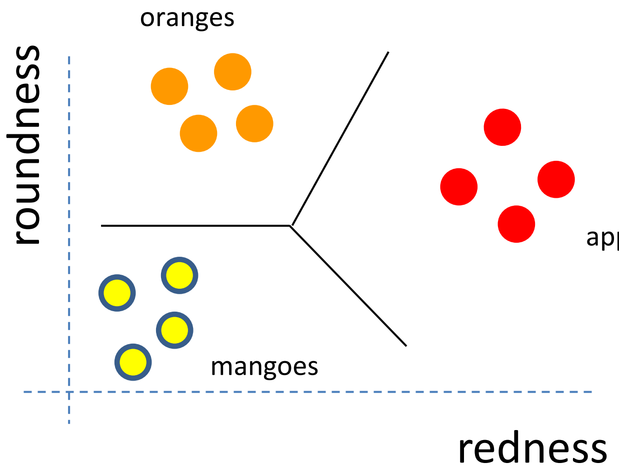


# Classification

- Given a set of data points (also called examples)
  - In solving a classification problem, the first step is to identify the unit of classification or "what is an example"
- Such that each example is represented by a feature vector
  - Representation of the example in terms of feature vector
- Assign a class label to each example
  - Such as apple, orange or mango



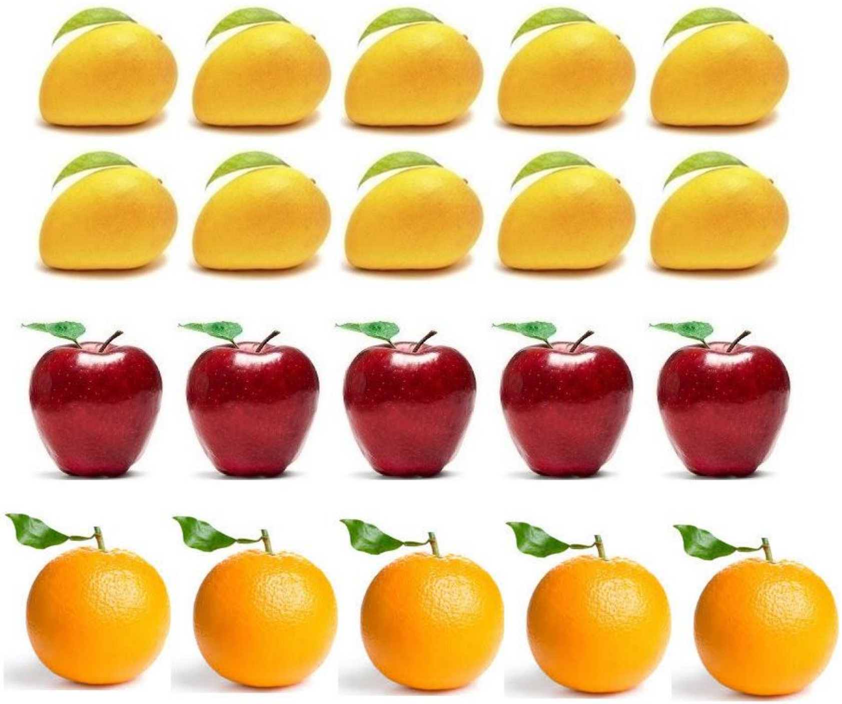
- Training Data
  - Set of examples for which both feature vectors and labels are available for "tuning"
    - Finding a mathematical function (called a classifier) that can be used to assign these labels
- Validation Data
  - Set of examples (with known labels) that are used to evaluate how well the trained classifier is expected to generalize to novel cases
- Test Data
  - Data for which the labels are not known and the ML model is used to find their labels



# Classification

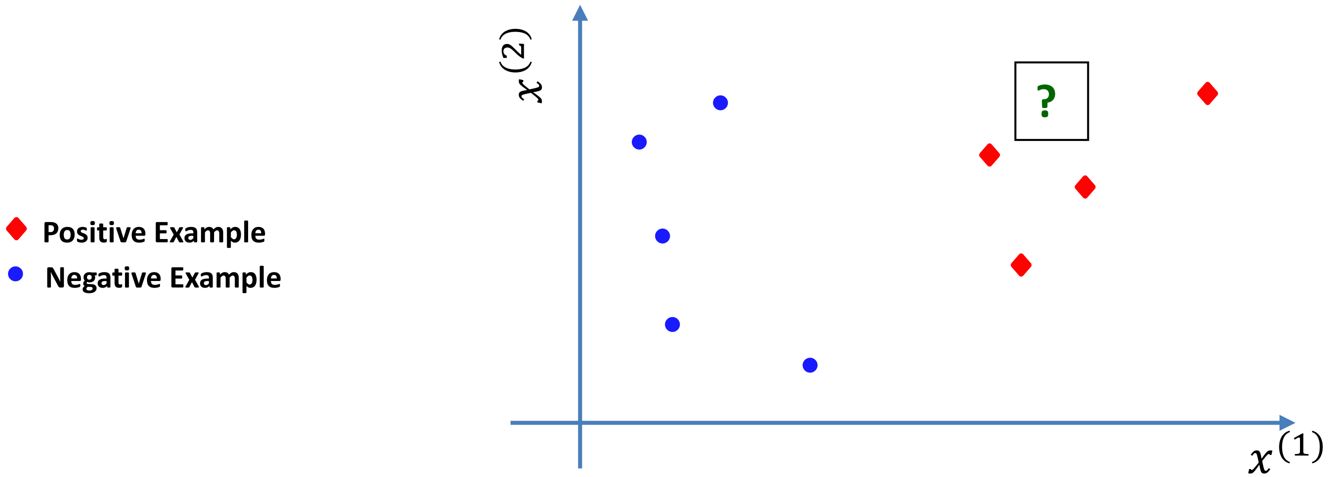
- The Objective of Classification is to assign class labels $y \in \{c_1, c_2, \ldots, c_M\}$
- to a given feature vector $\boldsymbol{x} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(d)} \end{bmatrix}$
- The classifier may use previously known and available training data
  - Good generalization, Good memorization
- The training data comprises of classified data points: $\mathrm{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}, \boldsymbol{x}_{i_{(d\times 1)}} = \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ \vdots \\ x_i^{(d)} \end{bmatrix}$

  $\mathrm{Y} = \{y_1, y_2, \ldots, y_N\}, y_i \in \{c_1, c_2, \ldots, c_M\}$



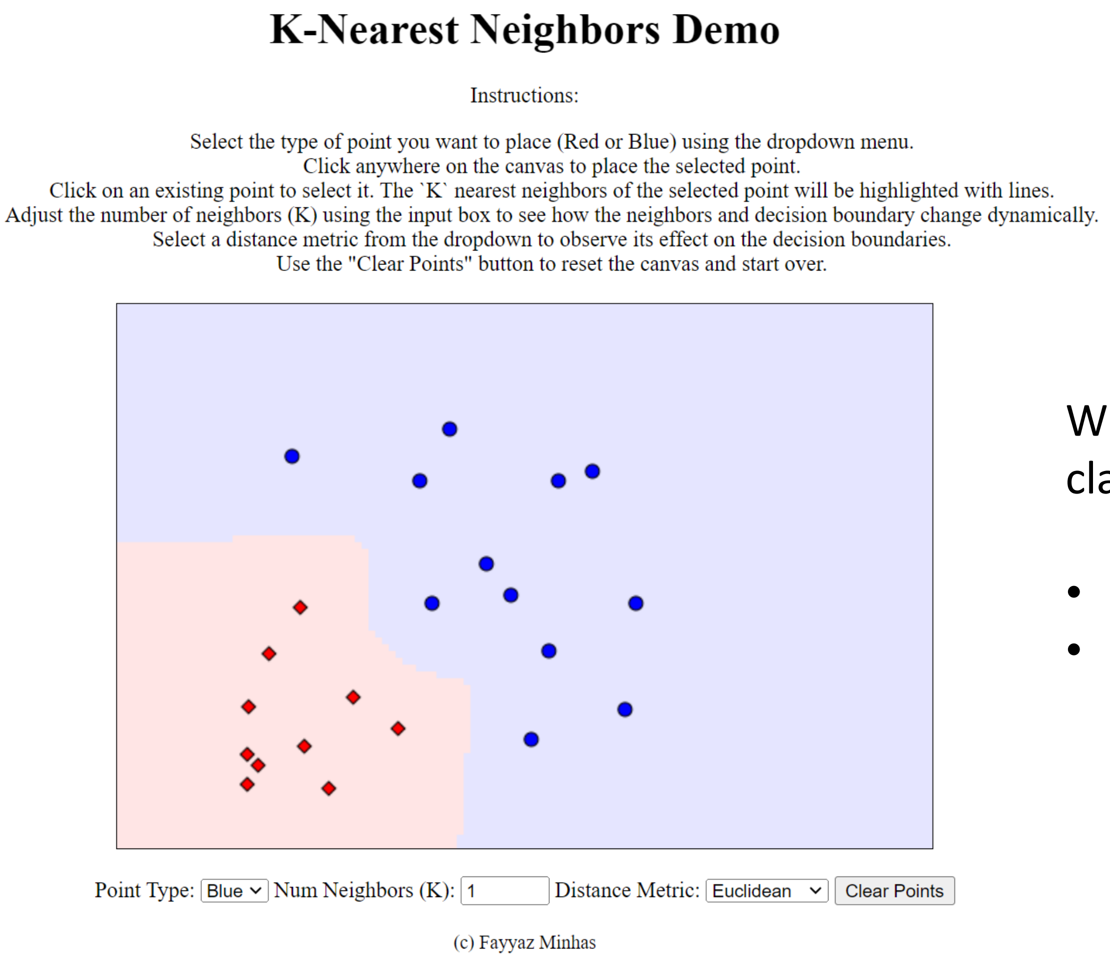
# Classification Approaches: Nearest Neighbor and kNN

$$D(\boldsymbol{x_a}, \boldsymbol{x_b}) = \sqrt{\left(x_a^{(1)} - x_b^{(1)}\right)^2 + \left(x_a^{(2)} - x_b^{(2)}\right)^2}$$




- Python Warm-up Lab Exercise
- https://github.com/foxtrotmike/CS909/blob/master/DM_1_kNN.ipynb

# Example (k=1)-Nearest Neighbor Classification



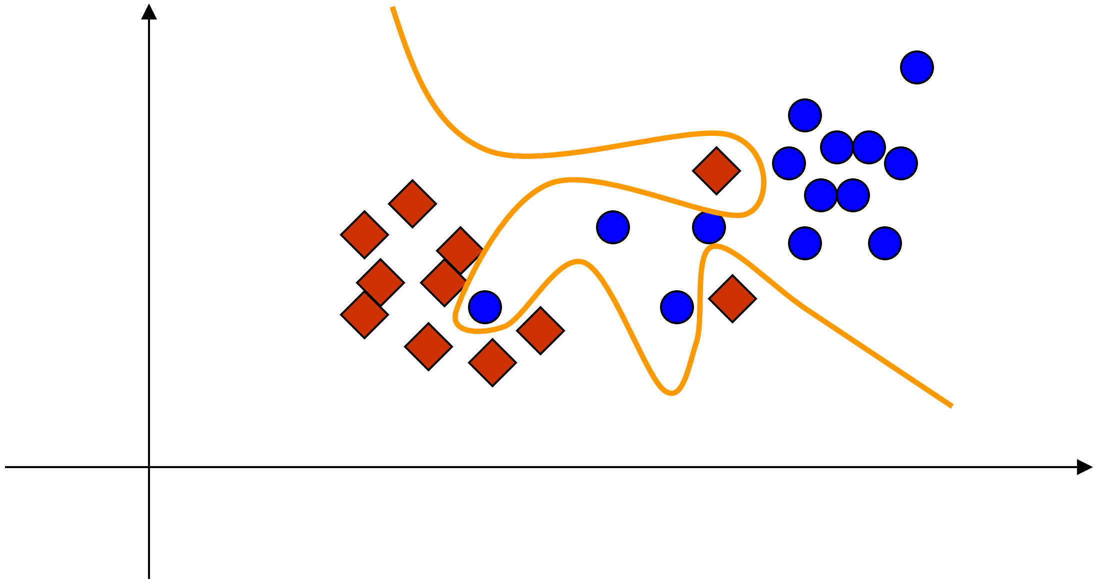

What are some issues with the k-NN classifier?

- Class imbalance.
- In higher-dimensions, there is essentially no notion of distance.

Demo: https://foxtrotmike.github.io/CS909/knn.html

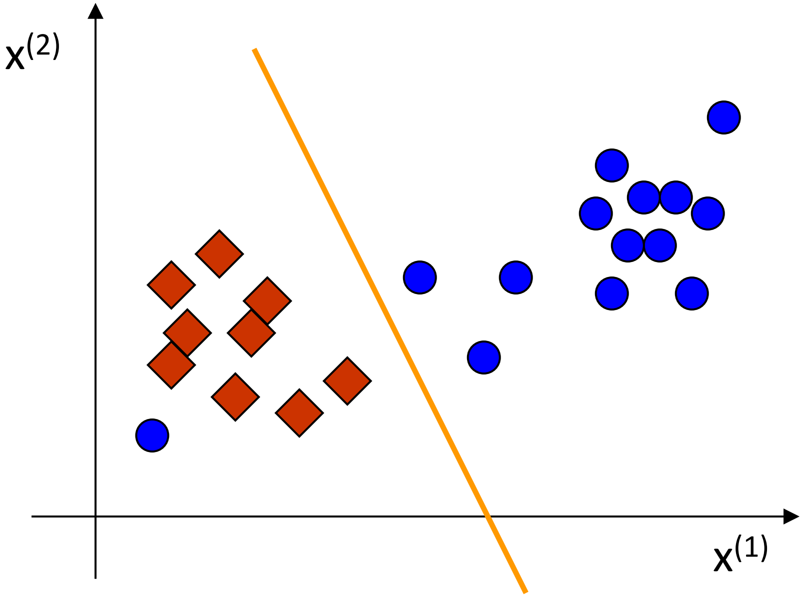
# Classification Approaches: Supervised…

- Nonlinear Classification boundary



# Linear Separability

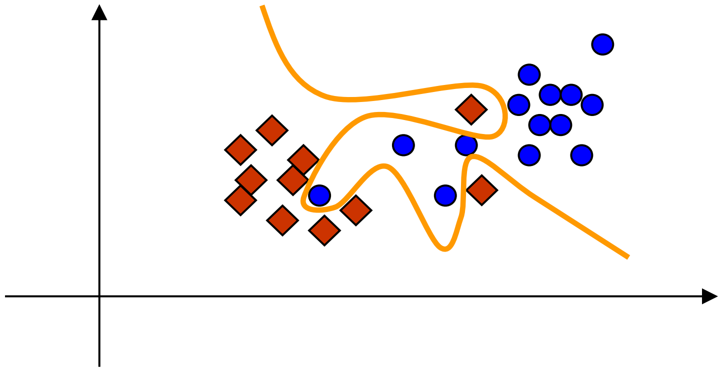
- If data points can be separated by a linear discriminant then that dataset/classification problem is called “linearly separable”



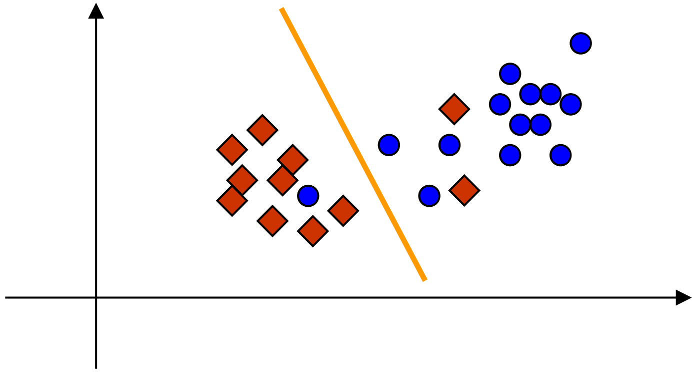

- Mathematically,
  - If there exists a linear function $f(\boldsymbol{x};\boldsymbol{w}) = w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_d x^{(d)} + b = 0$ such that
    - If $y_i = +1$, then $f(\boldsymbol{x_i};\boldsymbol{w}) > 0$
    - And If $y_i = -1$, then $f(\boldsymbol{x_i};\boldsymbol{w}) < 0$

# Classification Approaches: Supervised…

- Generalization and Memorization
  - Remembering everything is not learning
  - The true test of learning is handling similar but unseen cases, i.e., Generalization



Has great memorization but may generalize poorly (under certain assumptions)



Has lesser memorization but may generalize better (under certain assumptions)

# Discriminant based classification

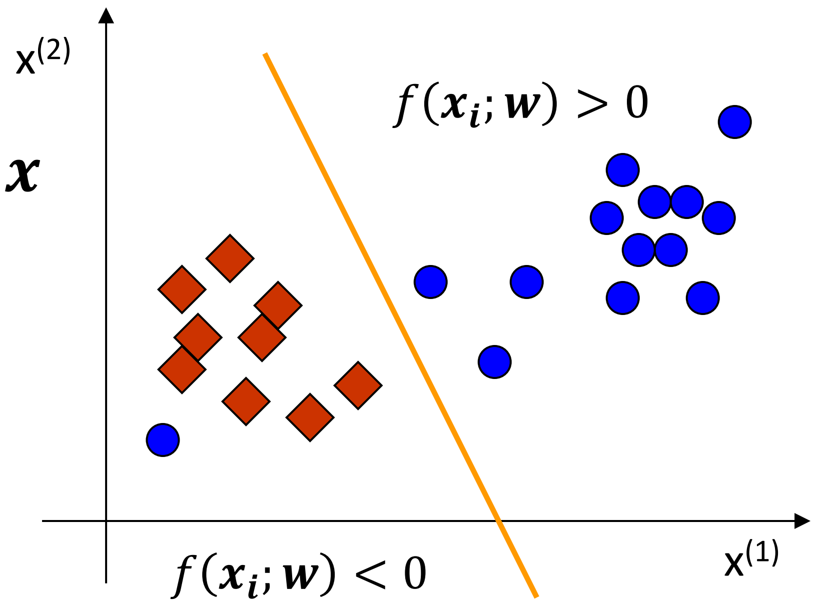
- In this type of classification, the objective is to learn a function or, in the case of more than 2 classes, a set of functions from training data which can generate decisions for test data such that the classes in the data can be separated
  - Class label assignment: $c(\boldsymbol{x}) = argmax_{k=1,\ldots,M}\, f_k(\boldsymbol{x})$
    - $f_k(\boldsymbol{x})$ tells you the '*k-classiness*' of an example $\boldsymbol{x}$
  - If M = 2
    - Choose class-1 if $f_1(\boldsymbol{x}) \geq f_2(\boldsymbol{x})$, i.e., $f_1(\boldsymbol{x}) - f_2(\boldsymbol{x}) \geq 0$
    - Otherwise assign it to class-2, i.e., $f_1(\boldsymbol{x}) - f_2(\boldsymbol{x}) < 0$
    - We can thus replace the two functions with a single function

$$f(\boldsymbol{x}) = f_1(\boldsymbol{x}) - f_2(\boldsymbol{x})$$

Assign to positive class if $f(\boldsymbol{x}) \geq 0$, otherwise negative

$f(\boldsymbol{x}) = 0$ separates the two classes and is called the discriminant

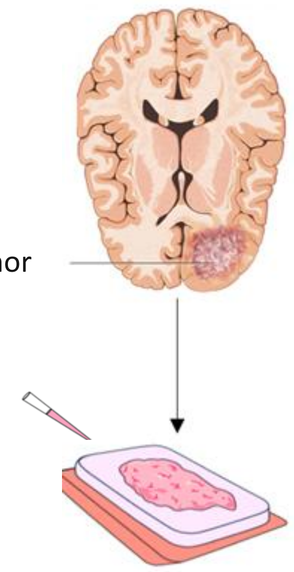
If the function(s) are linear, the classifier is called a linear discriminant



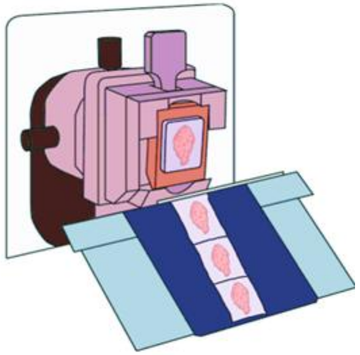
# Application: Computational Pathology

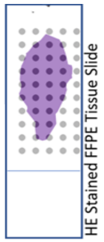
Tissue Acquisition → Slide Preparation → Whole Slide Scanning → **AI Assisted Clinical Decision Making**

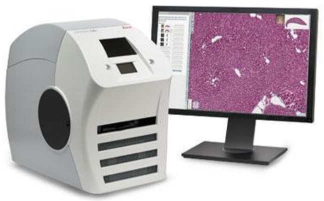
Tumor



Block preparation
Addition of dyes



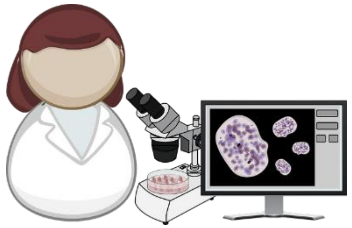


100,000 px

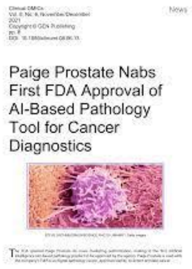
140,000 px

0.25 microns per pixel



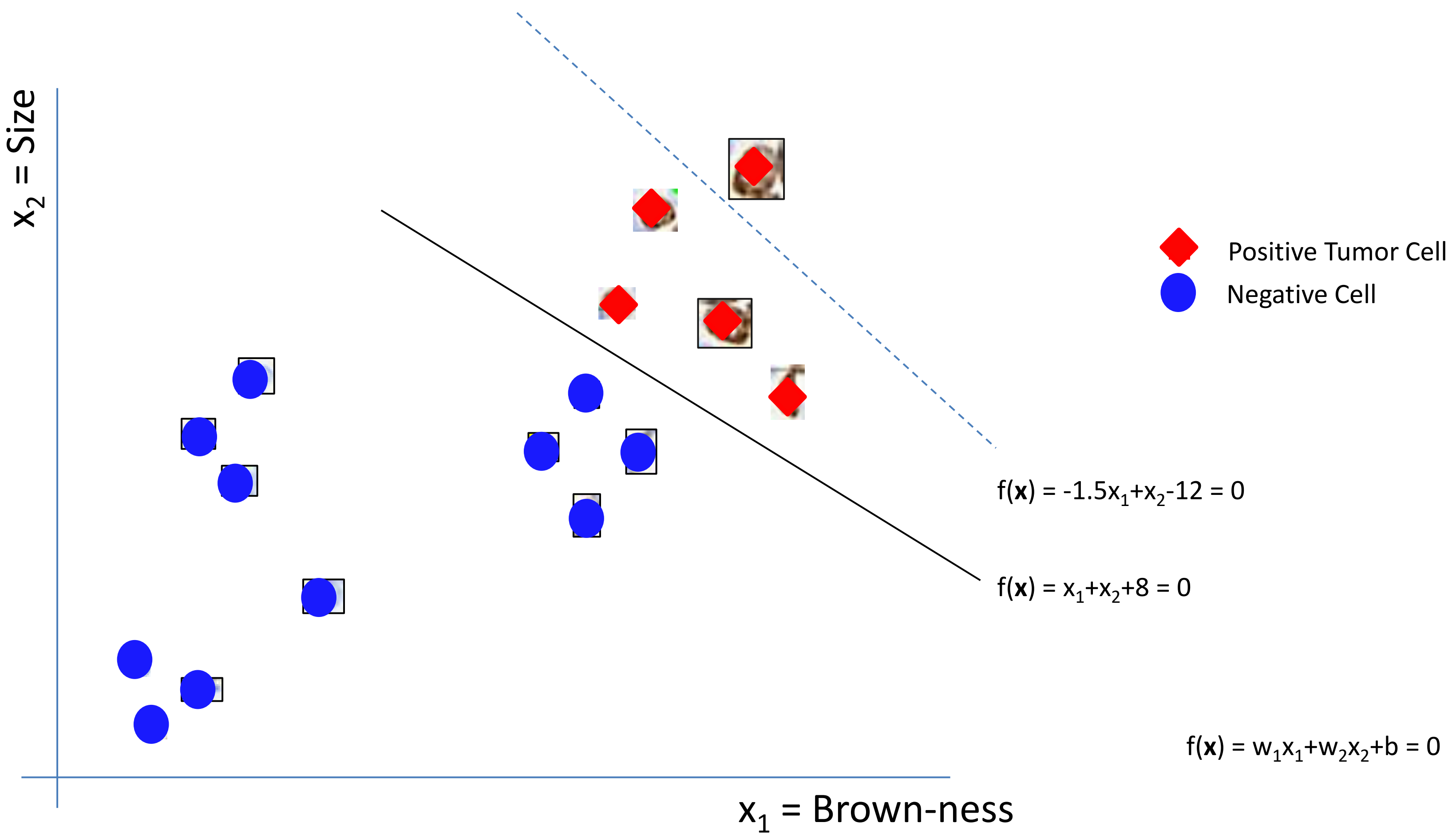
https://info.paige.ai/prostate






**Example Independent validation study of PAIGE Prostate:** Kanan, Christopher, Jillian Sue, Leo Grady, Thomas J. Fuchs, Sarat Chandarlapaty, Jorge S. Reis-Filho, Paulo G O Salles, Leonard Medeiros da Silva, Carlos Gil Ferreira, and Emilio Marcelo Pereira. “Independent Validation of Paige Prostate: Assessing Clinical Benefit of an Artificial Intelligence Tool within a Digital Diagnostic Pathology Laboratory Workflow.” Journal of Clinical Oncology 38, no. 15_suppl (May 20, 2020): e14076–e14076. https://doi.org/10.1200/JCO.2020.38.15_suppl.e14076.

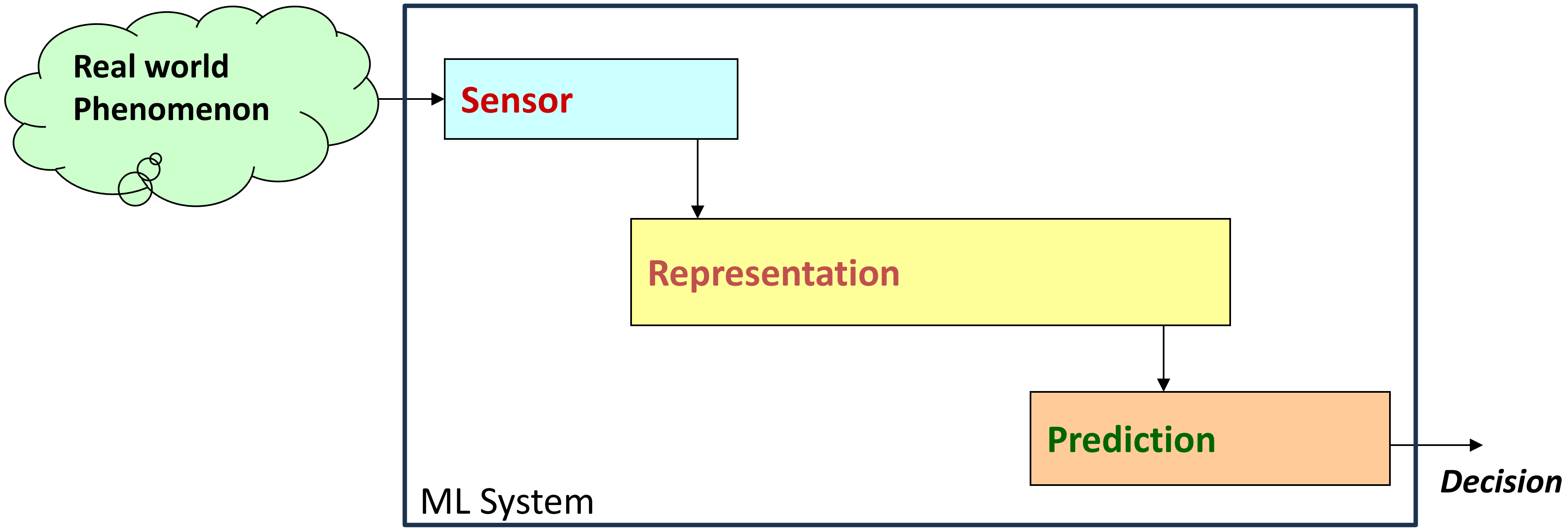
# Machine Learning in Cpath with simple lines



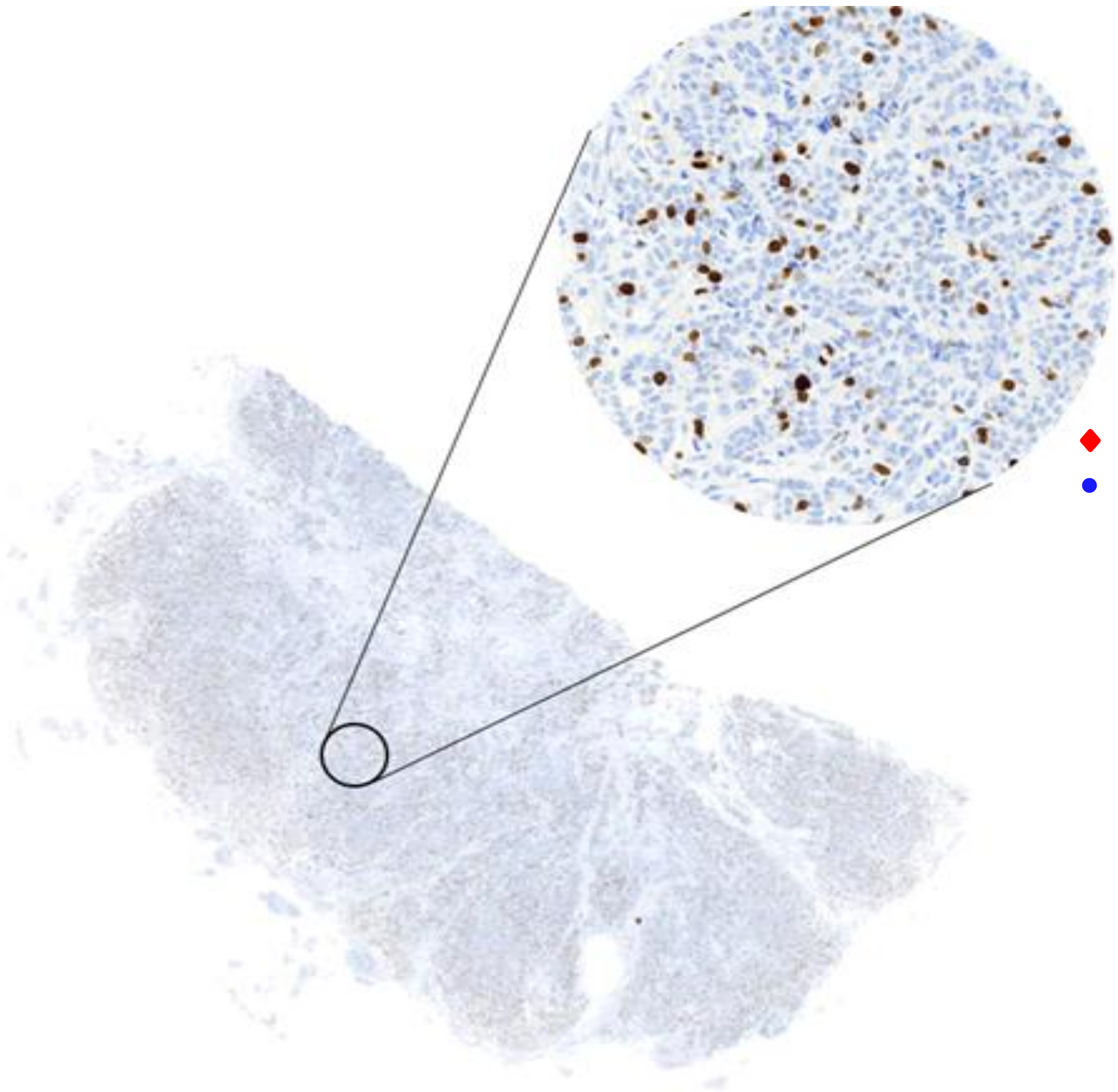

Goal is to be able to generalize to unseen data
With Deep Learning – we don't even need to define features!

# Constructs of a Data Mining System for Prediction

- Identify the objective
  - Identify the unit of classification (example)
    - Image block, protein sequence, ….

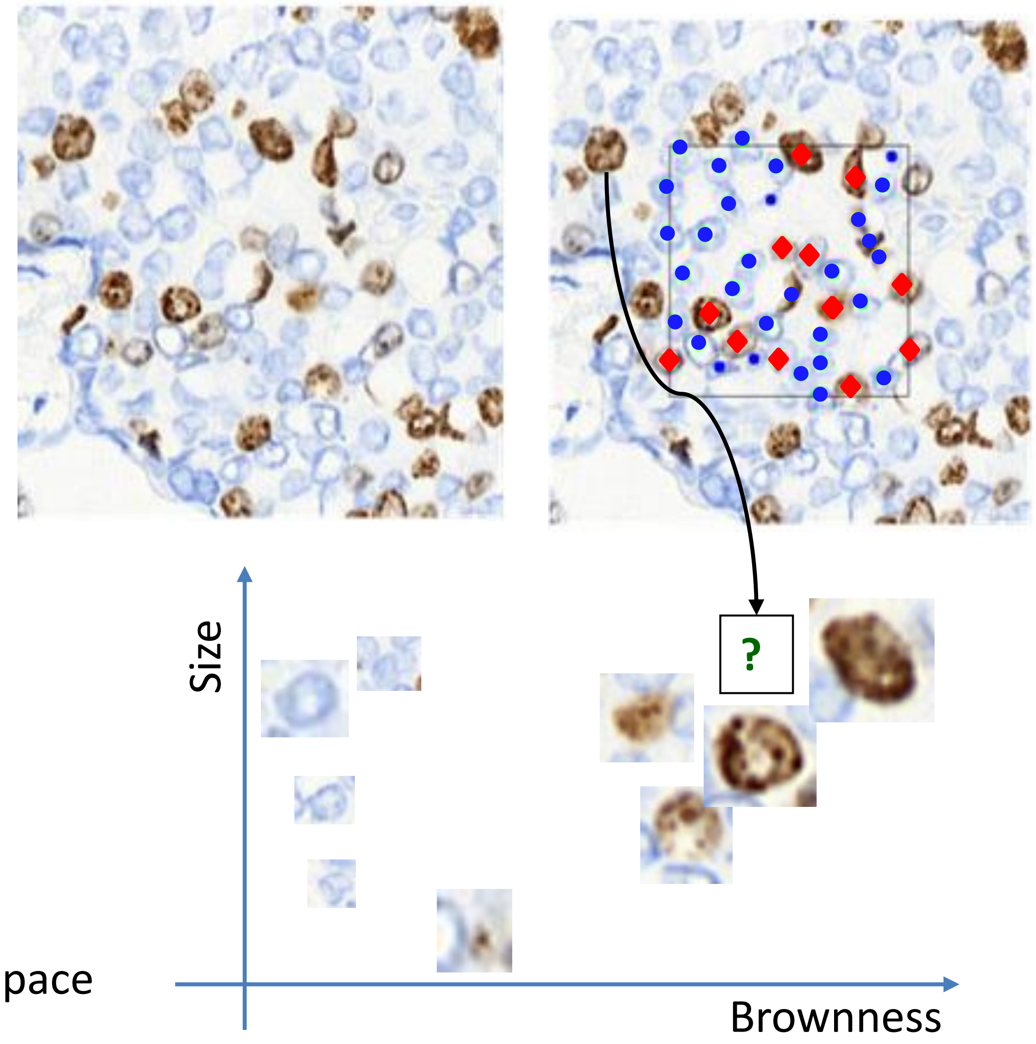


# How does ML work?



◆ Positive Tumor Cell

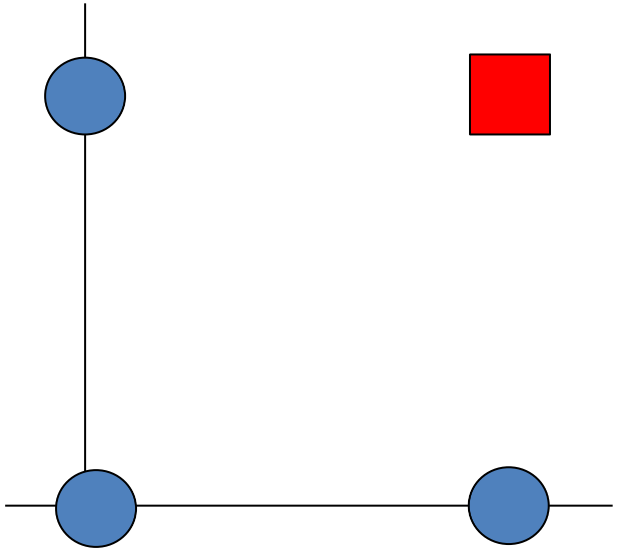
● Other Cell




Representation or Feature Space

Wenqi Lu, Islam M Miligy, Fayyaz Minhas, Young Saeng Park, David R J Snead, Emad A Rakha, Clare Verrill, Nasir Rajpoot "Lessons from a Breast Cell Annotation Competition Series for School Pupils." Scientific Reports, 2022. https://ora.ox.ac.uk/objects/uuid:9e34d4e6-c677-4380-9403-759808b349aa.

# Exercise



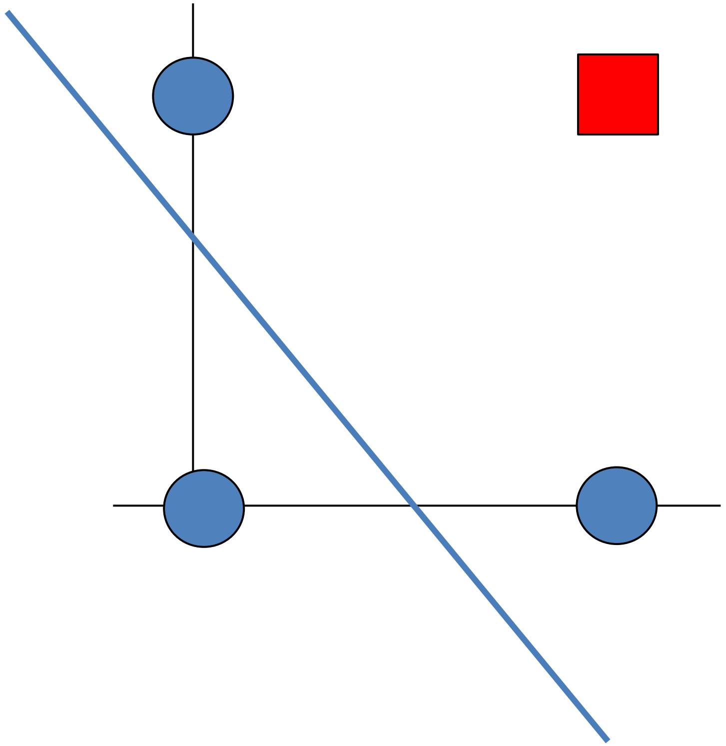
$$f(\boldsymbol{x};\boldsymbol{w}) = w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_d x^{(d)} + b = 0$$

such that

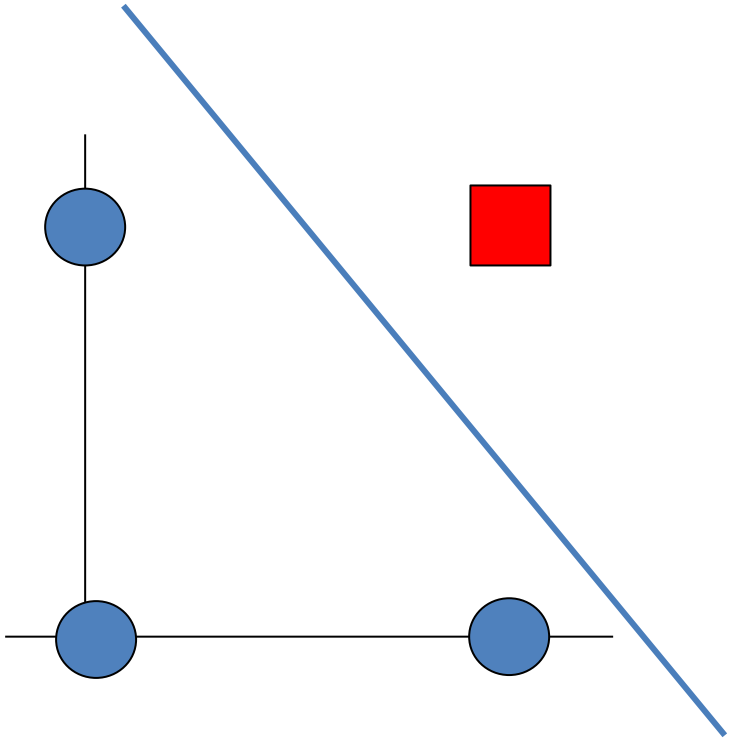
If $y_i = +1$, then $f(\boldsymbol{x_i};\boldsymbol{w}) > 0$

And If $y_i = -1$, then $f(\boldsymbol{x_i};\boldsymbol{w}) < 0$

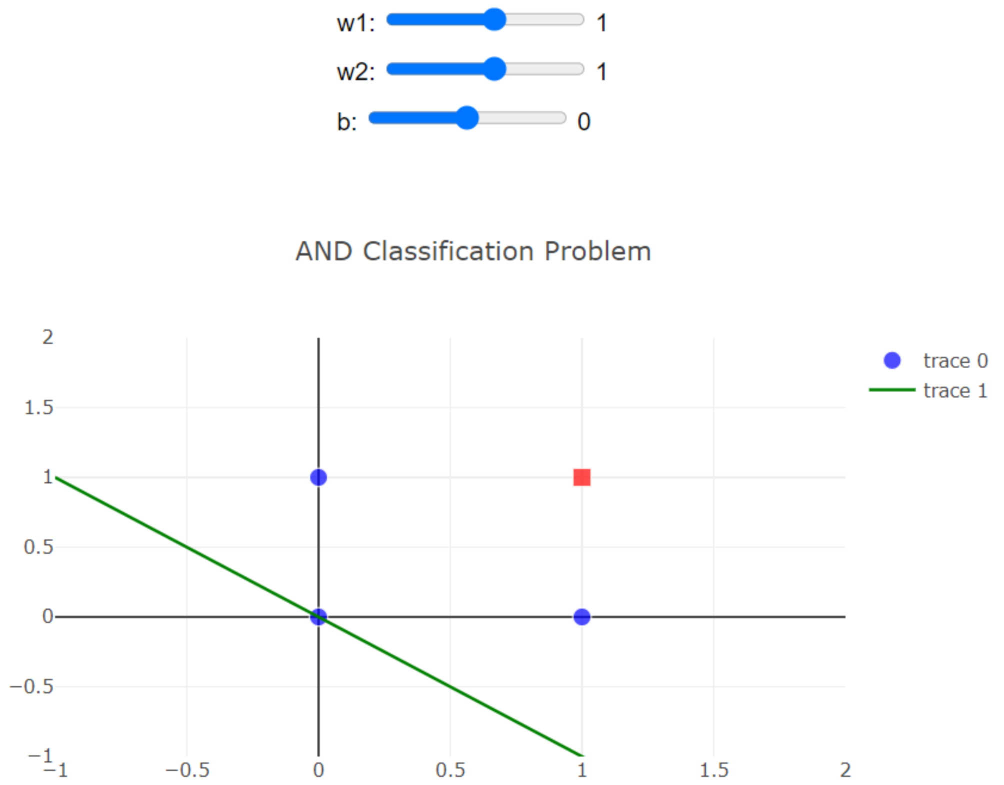
# Exercise



# Exercise

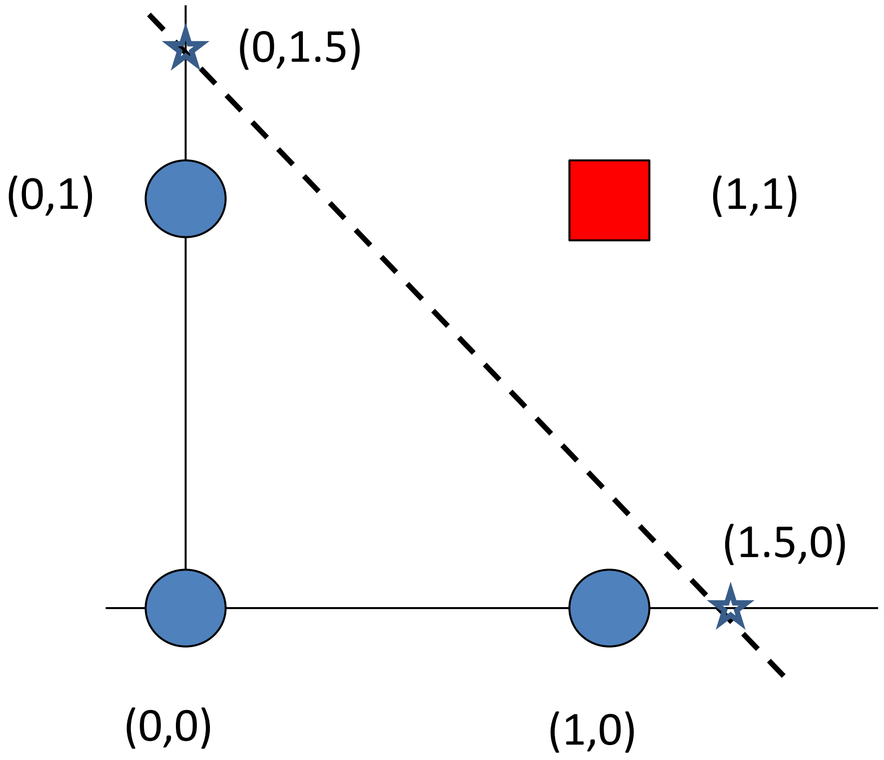


# Doing it interactively




https://foxtrotmike.github.io/CS909/AND-NEURON.html

# Example (Graphical Approach to finding the discriminant function)




$$f(\mathbf{x};\mathbf{w}) = w_1x^{(1)}+w_2x^{(2)}+b = 0$$

$$w_1(1.5)+w_2(0)+b = 0$$

$$b = -1.5w_1$$

$$w_1(0)+w_2(1.5)+b = 0$$

$$b = -1.5w_2$$

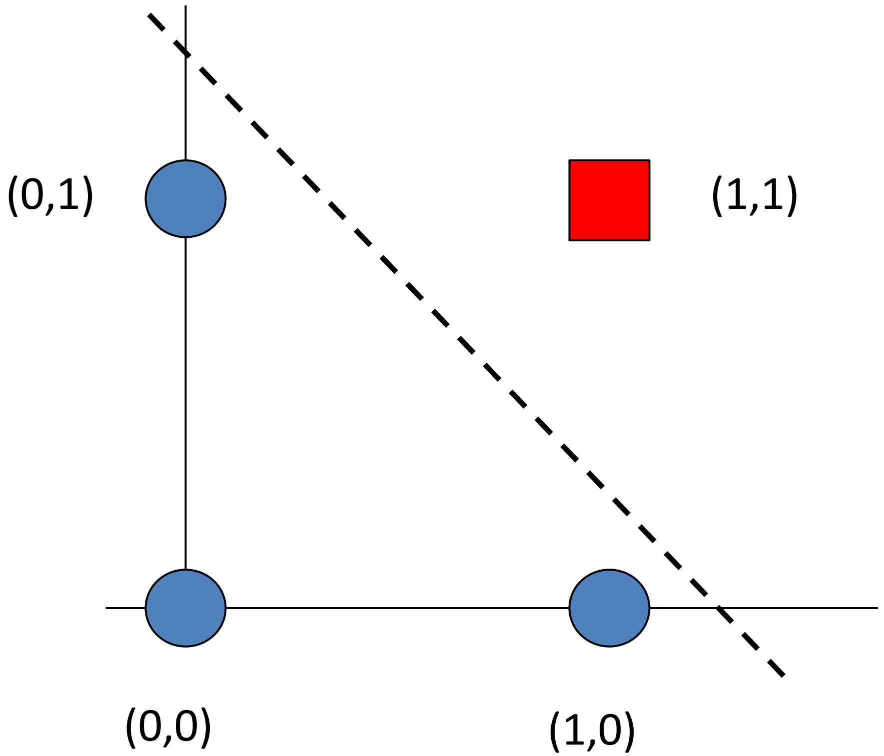
If I set $w_1 = 1.0$

$b = -1.5$
$w_2 = 1.0$

$$f(\mathbf{x};\boldsymbol{\theta}) = x^{(1)}+x^{(2)} -1.5$$

(1,1)→0.5
(1,0) →-0.5, (0,1) →-0.5, (0,0) →-1.5

# Example: Another Way (Algebraic Constraint Satisfaction to find equation)




$f(\mathbf{x};\mathbf{w}) = w_1x^{(1)}+w_2x^{(2)}+b = 0$

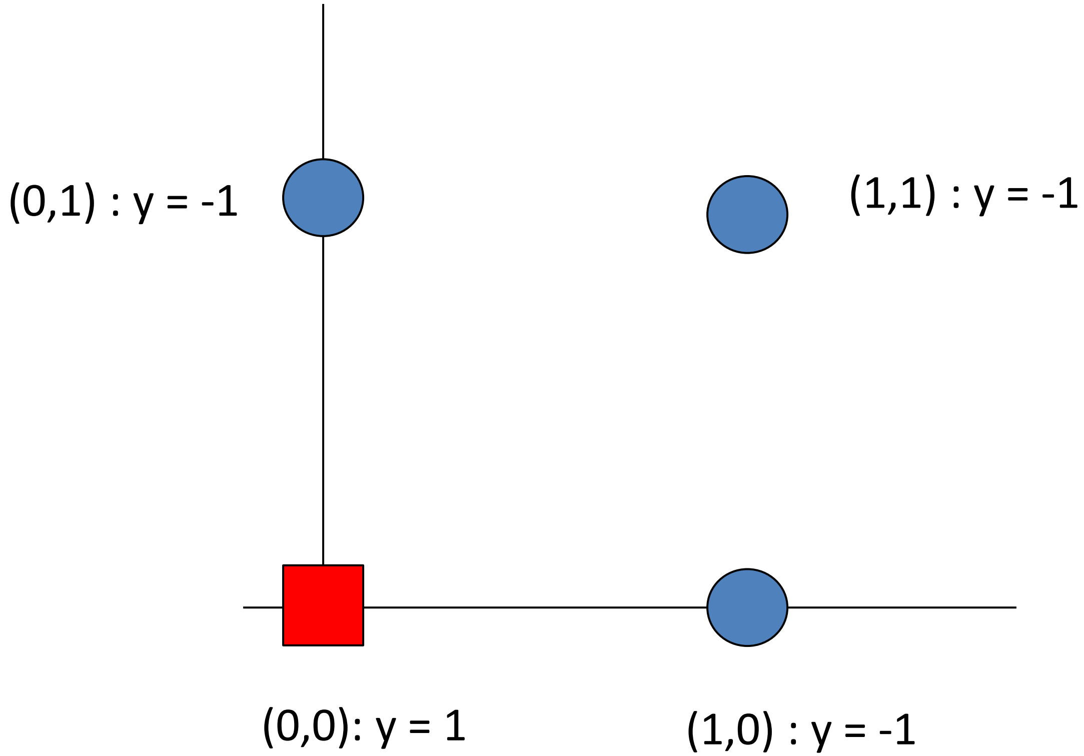
(1,1): $w_1(1.0)+w_2(1.0)+b > 0$
(1,0): $w_1(1.0)+w_2(0.0)+b < 0$
(0,1): $w_1(0.0)+w_2(1.0)+b < 0$
(0,0): $w_1(0.0)+w_2(0.0)+b < 0$

$w_1+w_2+b > 0$
$w_1+b < 0$
$w_2+b < 0$
$b < 0$

$b = -1.5$
$w_1= 1.0$
$w_2= 1.0$

# Exercise

- Is this problem linearly separable?

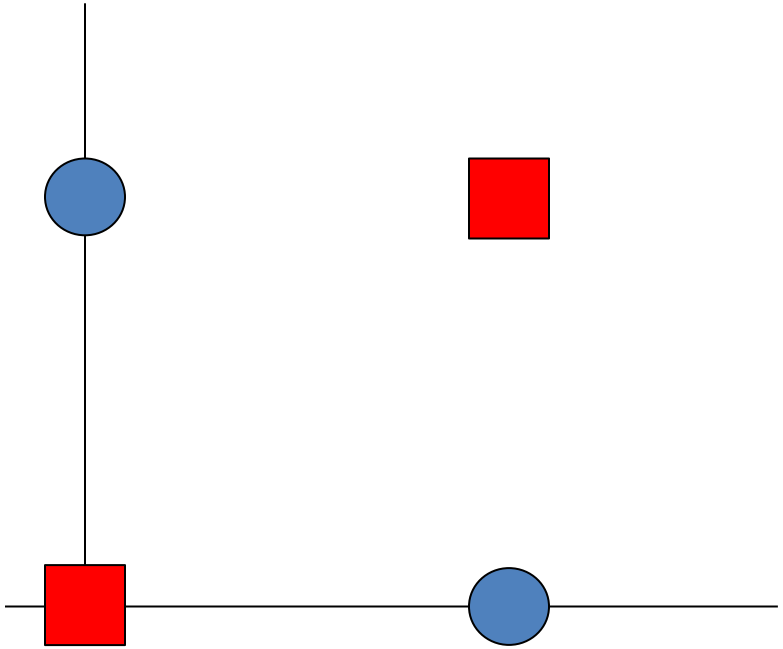


# What about this one?

- (0,0,0): -1
- (1,0,0): +1
- (0,1,0): -1
- (0,0,1):+1
- (1,0,1): +1
- (1,1,0): +1
- (0,1,1): +1
- (1,1,1): +1

$f(\mathbf{x};\mathbf{w}) = w_1x^{(1)}+w_2x^{(2)}+w_3x^{(3)}+b = 0$

- What about this one?



(1,1): $w_1(1.0)+w_2(1.0)+b > 0$

(1,0): $w_1(1.0)+w_2(0.0)+b < 0$

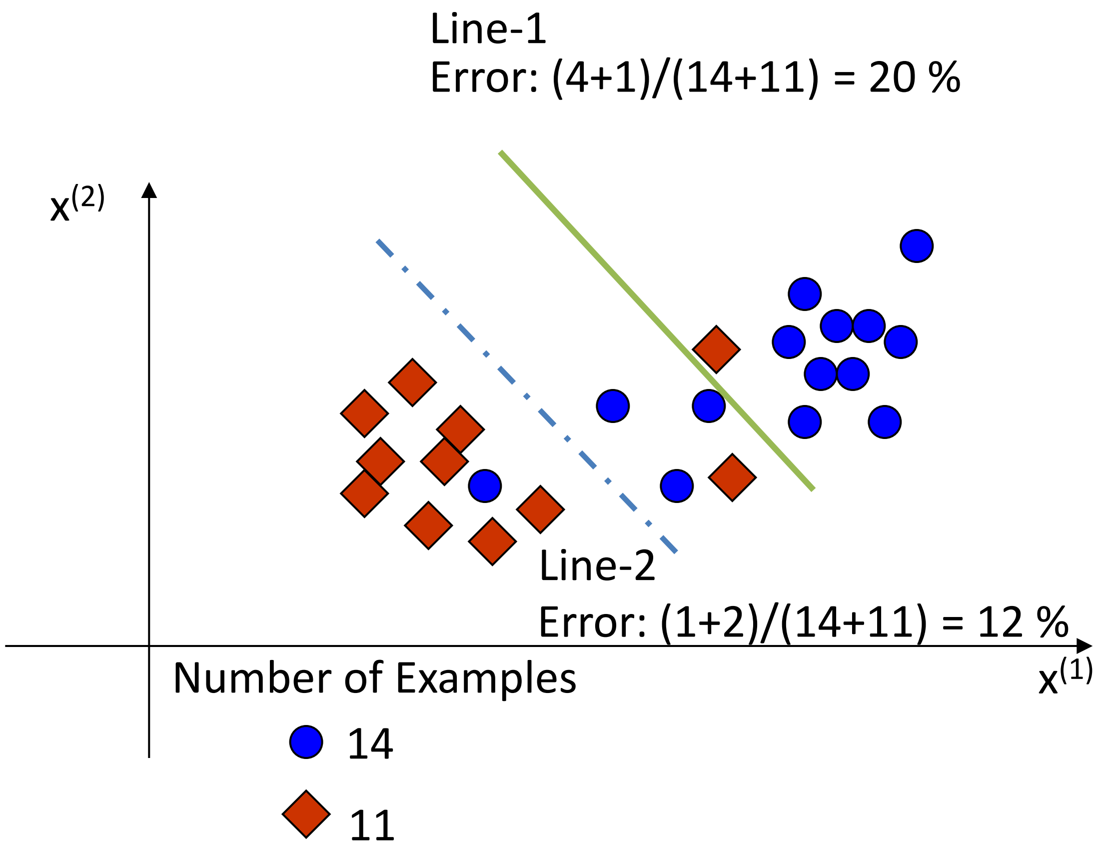
(0,1): $w_1(0.0)+w_2(1.0)+b < 0$

(0,0): $w_1(0.0)+w_2(0.0)+b > 0$

Machine learning and deep learning involve discovering meaningful representations of input data and then using these representations to partition data for various tasks

# Another way of looking at Classification

- We would like to minimize the number of errors a discriminant function $f(\boldsymbol{x})$ makes
- **Representation**: Assume we look at only linear functions
  $f(\boldsymbol{x};\boldsymbol{w}) = w_0 + w_1x^{(1)} + w_2x^{(2)} + \cdots + w_dx^{(d)} = 0$
- **Evaluation**: We need to define error that a particular $f(x;w)$ makes
- **Optimization**: We need to minimize the error by tuning $\boldsymbol{w}$

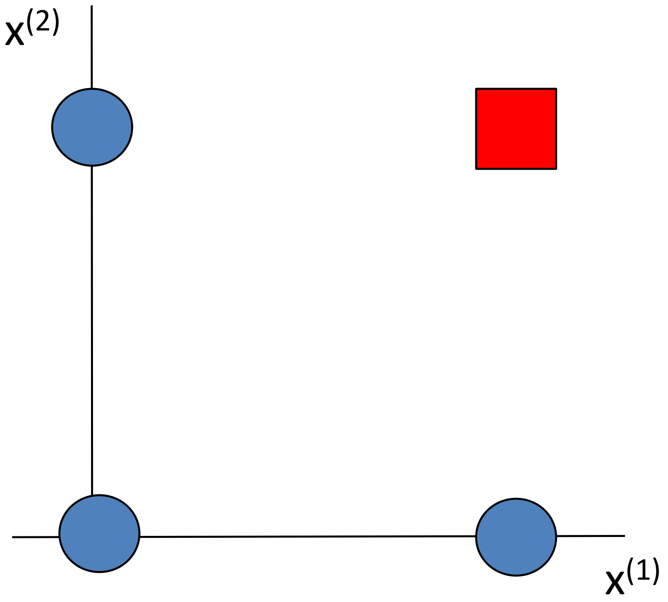


# Building Linear Discriminants

- Representation
  - Features
  - Linear Function

$$f(\mathbf{x};\boldsymbol{w}) = w_1x^{(1)} + w_2x^{(2)} + \cdots + w_dx^{(d)} + b = 0$$

- Evaluation
  - Misclassification Error
- Optimization
  - Find a line that minimizes misclassifications
  - How done: Visual reckoning / Constraint Satisfaction
- Why Study Linear Models?




(1,1): $w_1(1.0)+w_2(1.0)+b > 0$

(1,0): $w_1(1.0)+w_2(0.0)+b < 0$

(0,1): $w_1(0.0)+w_2(1.0)+b < 0$

(0,0): $w_1(0.0)+w_2(0.0)+b < 0$

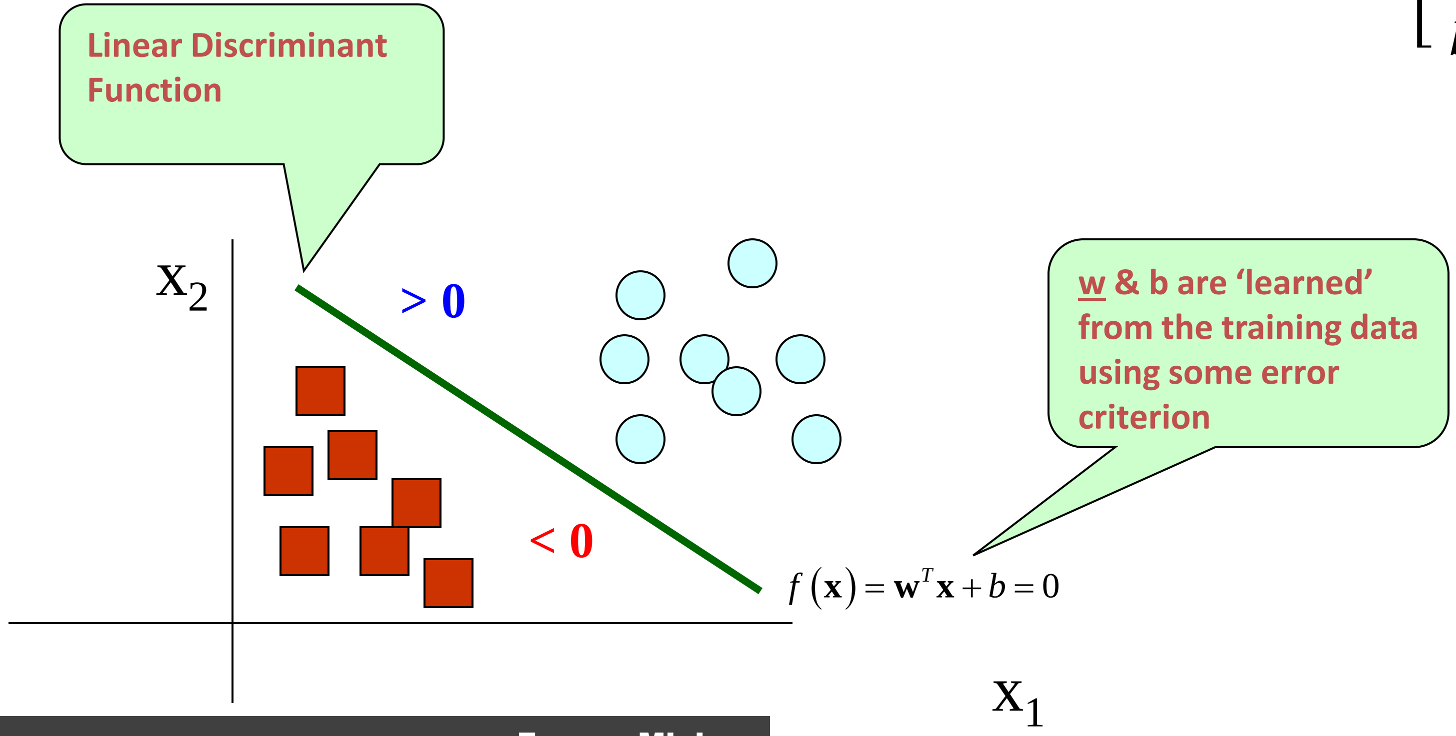
# A more mathematical look

- Linear Discriminants
- The linear discriminant function is given by

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(d)} \end{bmatrix}$$

$$f(\mathbf{x};\boldsymbol{w}) = w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_d x^{(d)} + b = \mathbf{w}^T\mathbf{x} + b$$
$$f(\mathbf{x}';\boldsymbol{w}') = w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_d x^{(d)} + b = \mathbf{w}'^T\mathbf{x}'$$

$$\mathbf{w}' = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{bmatrix} \quad \boldsymbol{x}' = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(d)} \\ 1 \end{bmatrix}$$



# Classification Loss Function

- A misclassification is an error
  - If a training example has a label of $y = +1$,then its discriminant function score $f(\boldsymbol{x})$ should be _____
  - If a training example has a label of $y = -1$,then its discriminant function score $f(\boldsymbol{x})$ should be _____
  - Thus, we have an error whenever: ___________

# Classification Loss Function

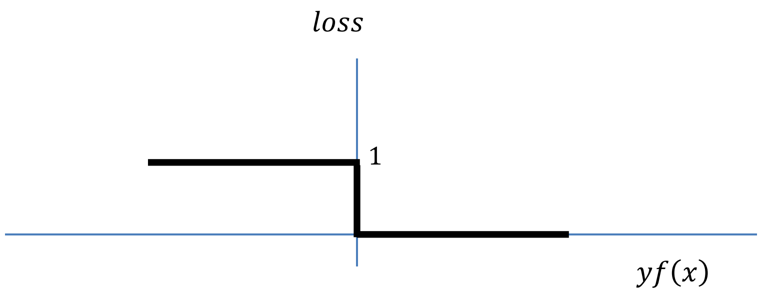
- A misclassification is an error
  - If a training example has a label of $y = +1$,then its discriminant function score $f(\boldsymbol{x})$ should be $> 0$

  - If a training example has a label of $y = -1$,then its discriminant function score $f(\boldsymbol{x})$ should be $< 0$

  - Thus, we have an error whenever: $yf(\boldsymbol{x}) < 0$

# 0-1 Loss/Error

- Consider a single example:
  - Our error function is: $l(f(x), y) = \begin{cases} 0 & yf(x) > 0 \\ 1 & yf(x) \leq 0 \end{cases}$

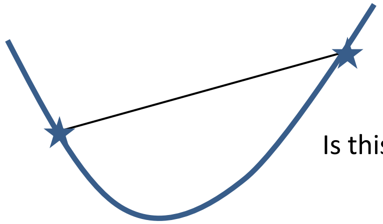



Formally called the zero-one loss function

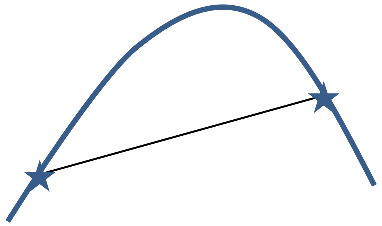
# 0-1 Misclassification Error

- We want to find the parameters of the discriminant that minimize the loss for all examples in training
- Issues with 0-1 loss
  - Non Differentiable
  - Leads to poor optimization
- We need a “surrogate” or approximation of the loss
  - Should be continuous
  - Should be an over-approximation of the 0-1 loss
    - Generates at least as much error as the 0-1 loss would
  - Should be convex
    - Convex loss function leads to convex optimization problems which are easier to solve as they have a single minima

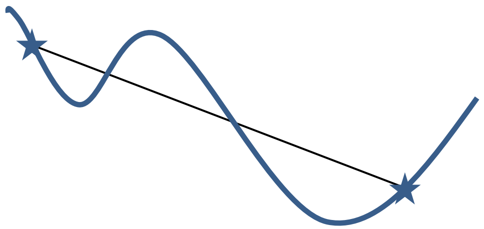
Convexity: If a line connecting two points on a curve lies on or above the curves at all times



Is this convex?



Is this convex?



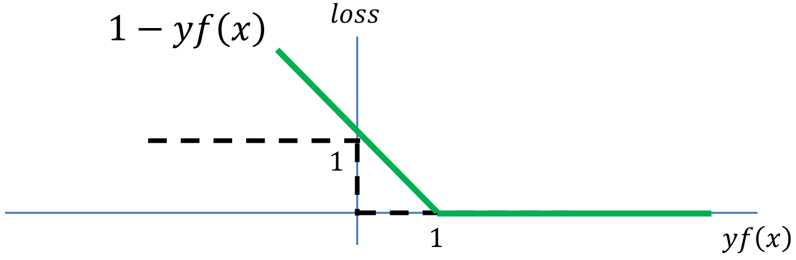
Is this convex?

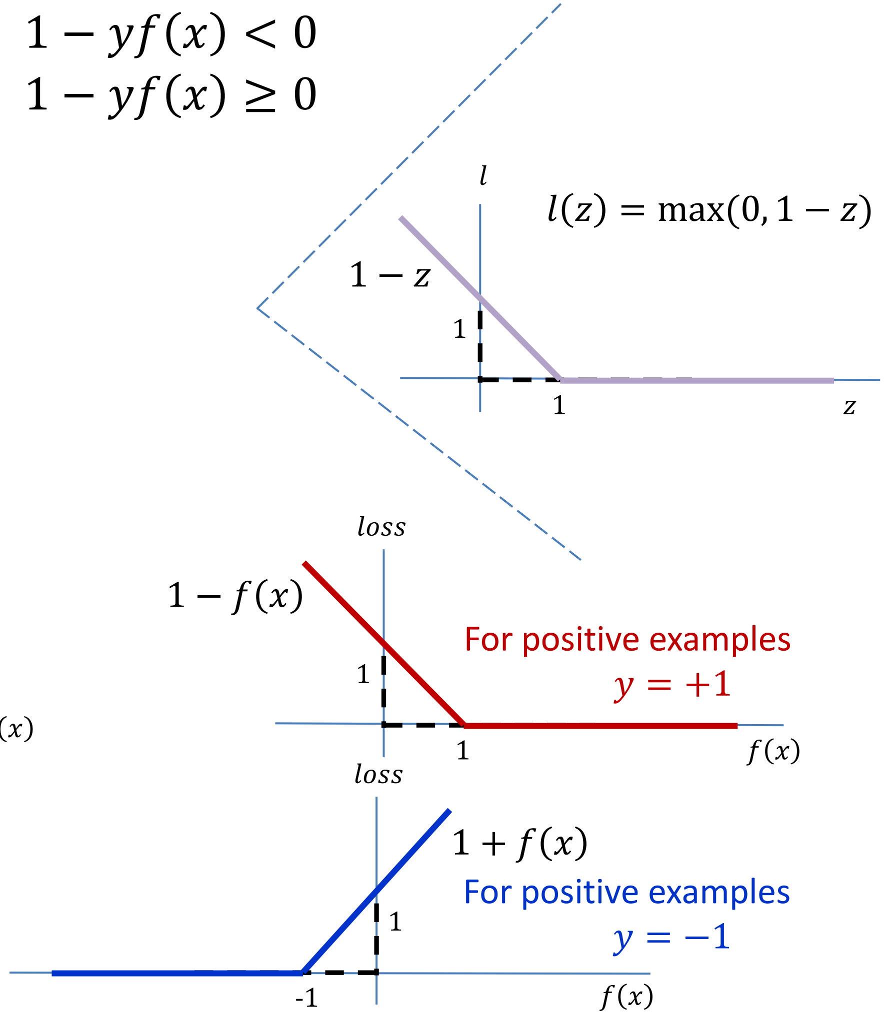
# Surrogate Classification Loss

$$l(f(\boldsymbol{x}), y) = \begin{cases} 0 & yf(x) > 1 \\ 1 - yf(x) & yf(x) \le 1 \end{cases} = \begin{cases} 0 & 1 - yf(x) < 0 \\ 1 - yf(x) & 1 - yf(x) \ge 0 \end{cases}$$

$OR$

$$l(f(x), y) = \max(0, 1 - yf(x))$$




- **Hinge Loss Function**
  - A convex over-approximation of the 0-1 loss
  - Adds some "margin" of error to the classification
    - Prediction label can be +1 or -1 depending upon whether $f(x) > 0$ or $f(x) < 0$
    - However, we incur a loss if for positive training examples $f(x) < 1$ or for negative examples $f(x) > -1$

# Optimization

$$\min_w L(\boldsymbol{X},\boldsymbol{Y};\boldsymbol{w}) = \sum_{i=1}^{N} \max\{0,1-y_i f(\boldsymbol{x}_i;\mathbf{w})\}$$

- How can we solve it?
  - Take the derivative and substitute to zero
  - How else can we solve it?
    - Use gradient descent

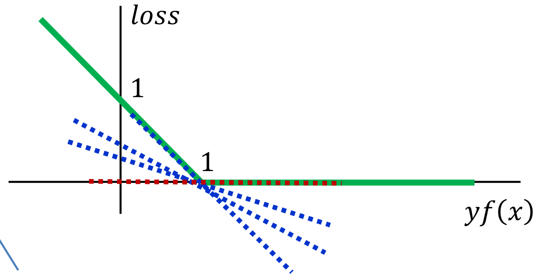
# Optimization

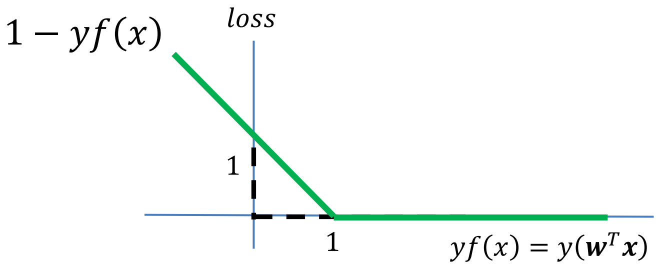
$$\min_w L(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{w}) = \sum_{i=1}^{N} l(f(\boldsymbol{x}_i;\mathbf{w})), y_i) = \sum_{i=1}^{N} \max\{0, 1 - y_i f(\boldsymbol{x}_i;\mathbf{w})\}$$

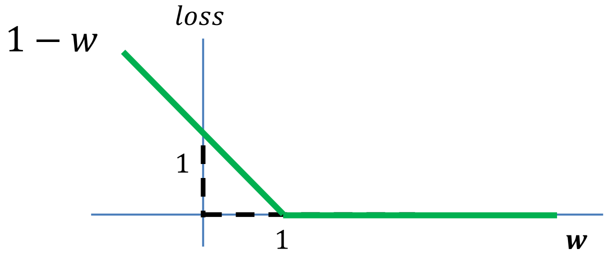
$$\frac{\partial L}{\partial \boldsymbol{w}} = \sum_{i=1}^{N} \frac{\partial l(f(\boldsymbol{x}_i;\mathbf{w})), y_i)}{\partial \boldsymbol{w}}$$

$$\frac{\partial}{\partial \boldsymbol{w}} \max\{0, 1 - y(\boldsymbol{w}^T \boldsymbol{x})\} = \begin{cases} 0 & 1 - yf(\boldsymbol{x};\mathbf{w}) < 0 \\ -y\boldsymbol{x} & else \end{cases} = \begin{cases} -y\boldsymbol{x} & l(f(\boldsymbol{x};\mathbf{w})), y) > 0 \\ 0 & else \end{cases}$$

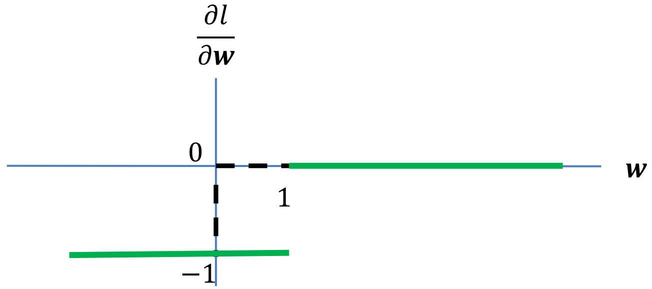
***What happens when $yf(x) = 1$ where the function has a "kink"?***
There, we can choose to define the "sub"-gradient to be the slope of any line that lies below or on the loss function itself (see dotted lines below). Consequently defining $\frac{\partial L}{\partial w}\Big|_{yf(x)=1} = \mathbf{0}$ should work (slope of red line).




For a simple example in which $x = 1, y = 1$




For a simple example in which $x = 1, y = 1$

# Algorithm

- Given:
  - Training Examples: $\{(\boldsymbol{x}_i, y_i) | i = 1 \dots N\}, \mathrm{y_i} \in \{-1, +1\}$
  - Learning rate (step size): $\alpha$
- Initialize $w^{(0)}$ at random
- Until Convergence ($k = 1 \dots K$ epochs)
  - For $i = 1 \dots N$
    - Pick example $\boldsymbol{x}_i$ with label $y_i$
    - Compute $f(\boldsymbol{x}_i) = \boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})^{\boldsymbol{T}}} \boldsymbol{x}_{\boldsymbol{i}}$
    - If $y_i f(\boldsymbol{x}_i) < 1$ then update weight vector using gradient descent

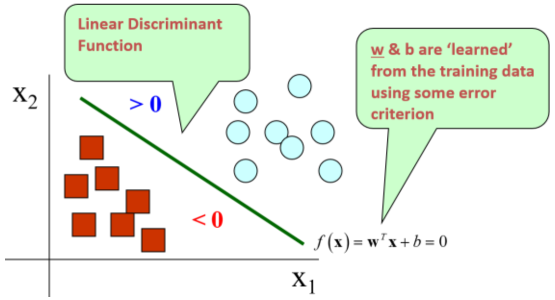
$$\boldsymbol{w}^{(\boldsymbol{k})} = \boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})} - \alpha \nabla \boldsymbol{l}\left(\boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})}\right) = \boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})} - \alpha(-y_i \boldsymbol{x}_i) = \boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})} + \alpha y_i \boldsymbol{x}_i$$

    - Check for convergence to stop

$$\nabla_{\boldsymbol{w}} \max\{0, 1 - y(\boldsymbol{w}^T \boldsymbol{x})\} = \begin{cases} 0 & 1 - y f(\boldsymbol{x}; \mathbf{w}) < 0 \\ -y\boldsymbol{x} & else \end{cases}$$

# **REO** For Perceptron

- **Representation**
  - Features
  - Discriminant
    - Linear




$$f(\mathbf{x}) = w_1\boldsymbol{x}_1 + w_2\boldsymbol{x}_2 + \cdots + w_d\boldsymbol{x}_d + b = \mathbf{w}^T\mathbf{x}$$

- **Evaluation**
  - 0/1 (Step) Loss $\quad l(f(x),y) = \begin{cases} 0 & yf(x) > 0 \\ 1 & yf(x) \leq 0 \end{cases}$
  - Hinge Loss $\quad l(f(x),y) = \begin{cases} 0 & yf(\boldsymbol{x}) > 1 \\ 1 - yf(\boldsymbol{x}) & yf(\boldsymbol{x}) \leq 1 \end{cases} = \max\{0, 1 - y(\boldsymbol{w}^T\boldsymbol{x})\}$
  - Overall Loss $\quad L = \frac{1}{N}\sum_{i=1}^{N} l(f(\boldsymbol{x}_i;\mathbf{w})), y_i) \xrightarrow[iid]{} E[l(f(\boldsymbol{x};\mathbf{w})), y)]$

- **Optimization**
  - Using Gradient Descent $\quad \nabla\boldsymbol{L} = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{w}} l(f(\boldsymbol{x}_i;\mathbf{w})), y_i)$

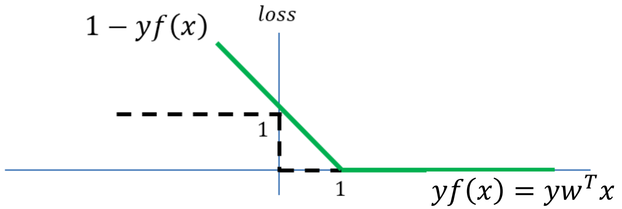
$$\nabla_{\boldsymbol{w}}\max\{0,1-y(\boldsymbol{w}^T\boldsymbol{x})\} = \begin{cases} 0 & 1-yf(\boldsymbol{x};\mathbf{w}) < 0 \\ -y\boldsymbol{x} & else \end{cases} = \begin{cases} -y\boldsymbol{x} & l(f(\boldsymbol{x};\mathbf{w})), y) > 0 \\ 0 & else \end{cases} = \mathrm{I}(l(f(\boldsymbol{x};\mathbf{w})), y))(-y\boldsymbol{x})$$

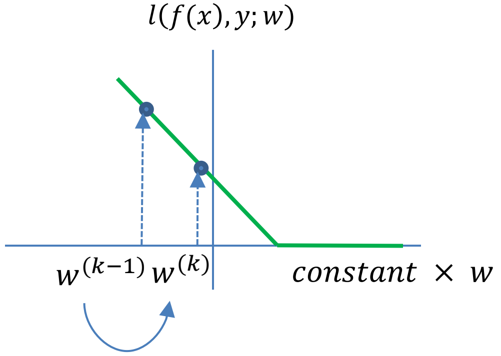
- Given:
  - Training Examples: $\{(\boldsymbol{x}_i, y_i) | i = 1 \ldots N\}, y_i \in \{-1, +1\}$
- Initialize $w^{(0)}$ at random
- Until Convergence (k=1…K)
  - For i = 1…N
    - Pick example $\boldsymbol{x}_i$ with label $y_i$
    - Compute $f(\boldsymbol{x}_i) = \boldsymbol{w}^{(\boldsymbol{k}-1)^{\boldsymbol{T}}}\boldsymbol{x}_{\boldsymbol{i}}$
    - If $y_i f(\boldsymbol{x}_i) < 1$ then update your weight vector using gradient descent

$$\nabla_{\boldsymbol{w}}\max\{0,1-y(\boldsymbol{w}^T\boldsymbol{x})\} = \begin{cases} 0 & 1-yf(\boldsymbol{x};\mathbf{w}) < 0 \\ -y\boldsymbol{x} & else \end{cases}$$

$$\boldsymbol{w}^{(\boldsymbol{k})} = \boldsymbol{w}^{(\boldsymbol{k}-1)} - \alpha\nabla\boldsymbol{l}\left(\boldsymbol{w}^{(\boldsymbol{k}-1)}\right) = \boldsymbol{w}^{(\boldsymbol{k}-1)} - \alpha(-y_i\boldsymbol{x}_i) = \boldsymbol{w}^{(\boldsymbol{k}-1)} + \alpha y_i\boldsymbol{x}_i$$

    - Check for convergence to stop



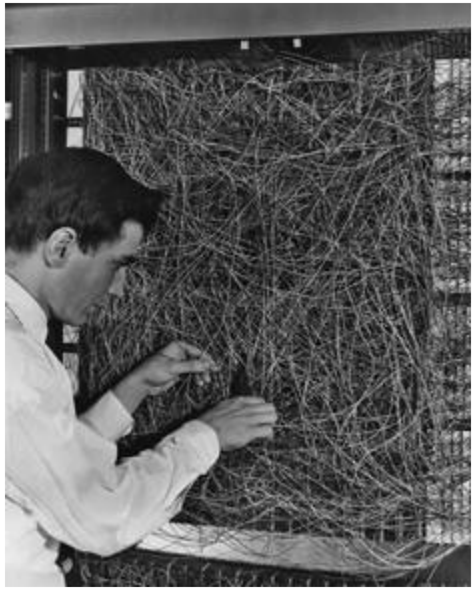




$$\boldsymbol{w}^{(\boldsymbol{k})} \leftarrow \boldsymbol{w}^{(\boldsymbol{k}-1)} - \alpha\nabla\boldsymbol{l}\left(\boldsymbol{w}^{(\boldsymbol{k}-1)}\right)$$

$$\boldsymbol{w}^{(\boldsymbol{k})} \leftarrow \boldsymbol{w}^{(\boldsymbol{k}-1)} - \boldsymbol{\alpha}\mathrm{I}(l(f(\boldsymbol{x};\mathbf{w})), y))(-y\boldsymbol{x}) = \boldsymbol{w}^{(\boldsymbol{k}-1)} + \alpha\mathrm{I}(l(f(\boldsymbol{x};\mathbf{w})), y))(y\boldsymbol{x})$$

# Perceptron

- A simpler version of this algorithm is called: Perceptron
  - It updated weights whenever an example was misclassified $(y_i f(\boldsymbol{x}_i) < 0)$ instead of when $y_i f(\boldsymbol{x}_i) < 1$
  - Rosenblatt (1962)
  - Minsky and Papert (1969, 1988)
  - This algorithm provides theoretical guarantees of convergence to a correct separating boundary
    - If the data is linearly separatable and you allow the pereceptron algorithm to run long enough, you will find the separating line!
    - **Perceptron Learning Rule Convergence Theorem**

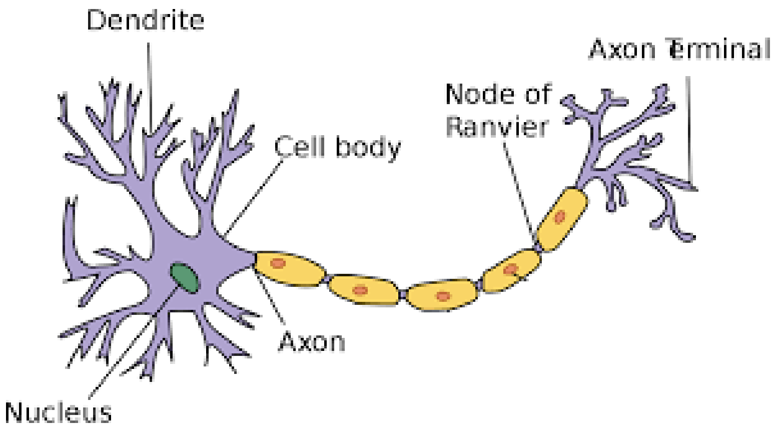


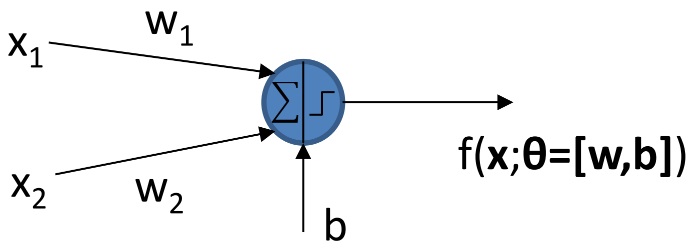
Frank Rosenblatt
July 11, 1928 – July 11, 1971

Marvin Minsky
Aug. 9, 1927 – Jan. 24, 2016

# Perceptron

- One of the first "artificial" neural networks




$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(d)} \end{bmatrix}$$

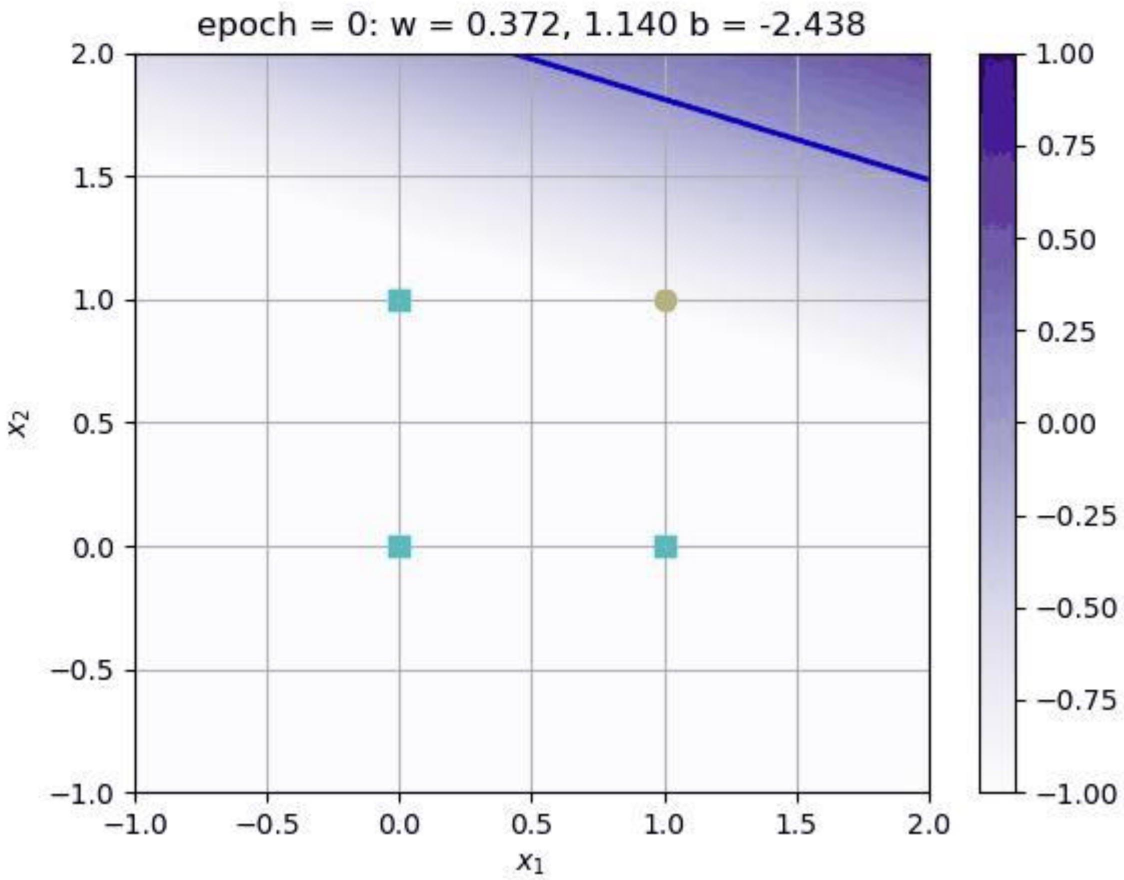
$$f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b$$

# Coding Exercise

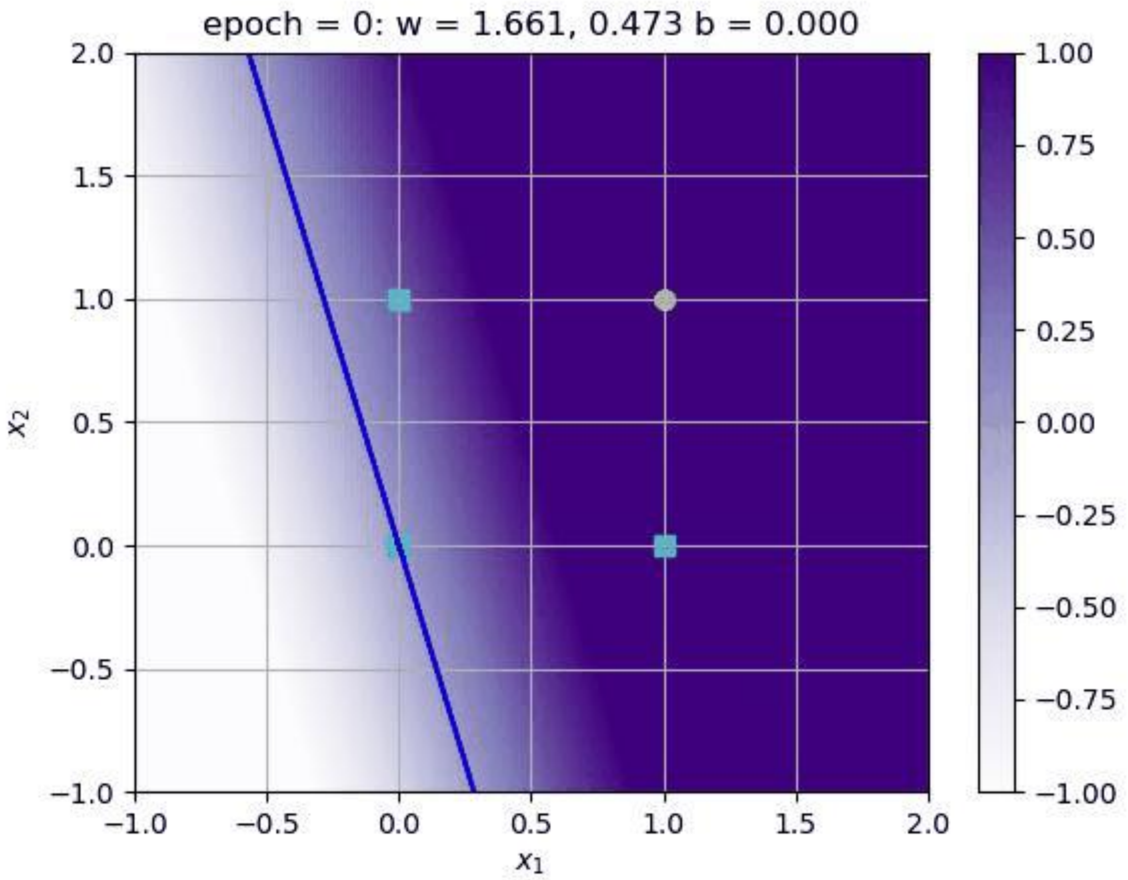
```
import numpy as np
import matplotlib.pyplot as plt
import itertools

class Perceptron:

    def __init__(self,alpha = 0.1, epochs = 200):
        self.alpha = alpha
        self.epochs = epochs
        self.W = np.array([0])
        self.bias = np.random.randn()
        self.Lambda = 0.5
    def fit(self,Xtr,Ytr):
        d = Xtr.shape[1]
        self.W = np.random.randn(d)
        for e in range(self.epochs):
            finished = True
            for i,x in enumerate(Xtr):
                if Ytr[i]*self.predict(np.atleast_2d(x))<1:
                    finished = False
                    self.W += self.alpha*Ytr[i]*x
                    self.bias += self.alpha*Ytr[i]
            if finished: break

    def score(self,x):
        return np.dot(x,self.W) + self.bias

    def predict(self,x):
        return np.sign(self.score(x))
```

```
if __name__=='__main__':
    from plotit import plotit
    Xtr = np.array([[-1,0],[0,1],[4,4],[2,3]])
    ytr = np.array([-1,-1,+1,+1])
    clf = Perceptron()
    clf.fit(Xtr,ytr)
    z = clf.score(Xtr)
    print("Prediction Scores:",z)
    y = clf.predict(Xtr)
    print("Prediction Labels:",y)
    plotit(Xtr,ytr,clf=clf.score,conts=[0],
           extent = [-5,+5,-5,+5])
```

https://github.com/foxtrotmike/CS909/blob/master/dm_lab_2_fm.ipynb




https://github.com/foxtrotmike/CS909/blob/master/perceptron_video.py


epoch = 0: w = 1.661, 0.473 b = 0.000
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
$x_2$
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
$x_1$
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

# End of Lecture

We want to make a machine that will be proud of us.

- Danny Hillis