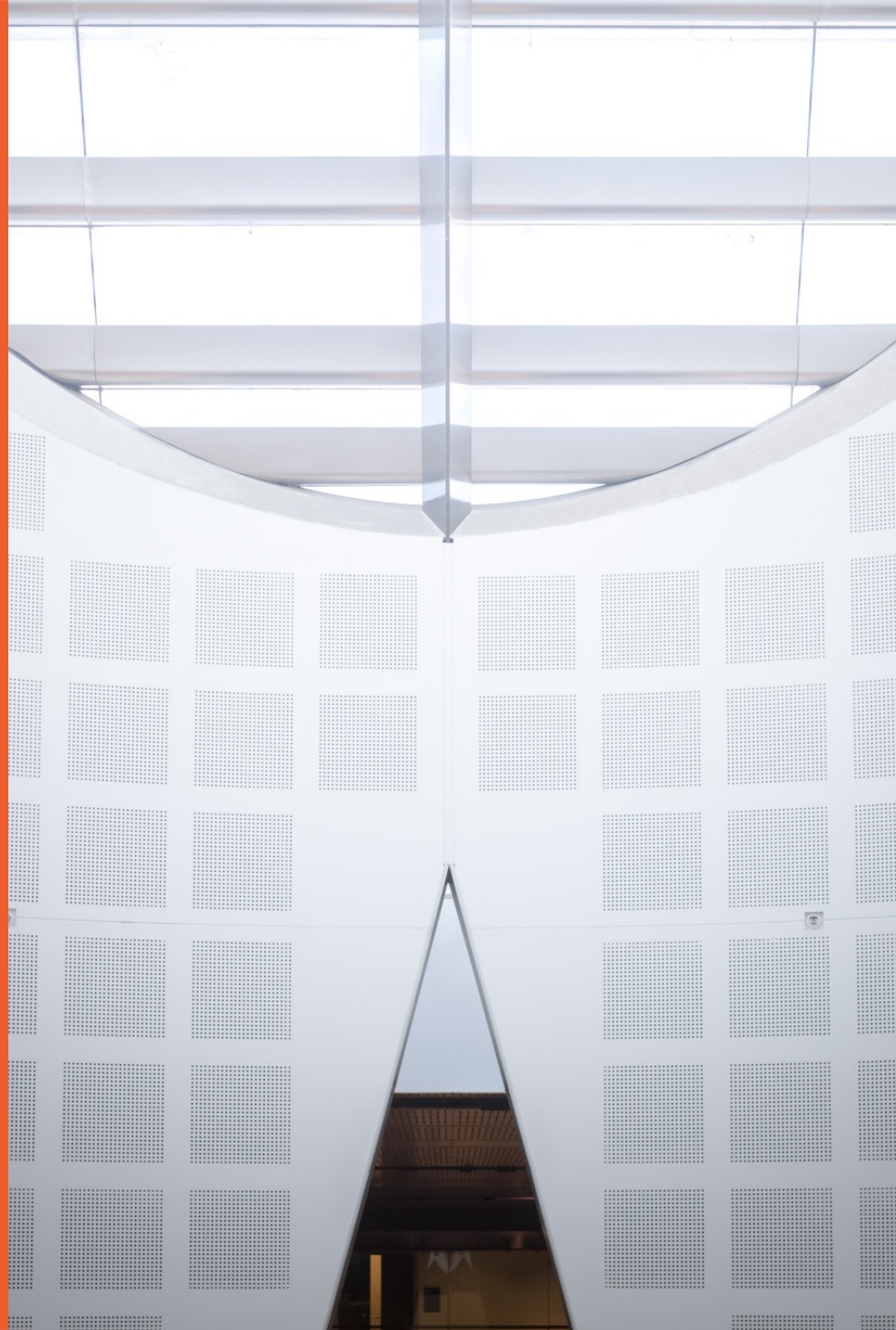
Clinical Prediction Models

Dr Chang Xu

School of Computer Science

Reference: Healthcare Data Analytics, Chapter 10



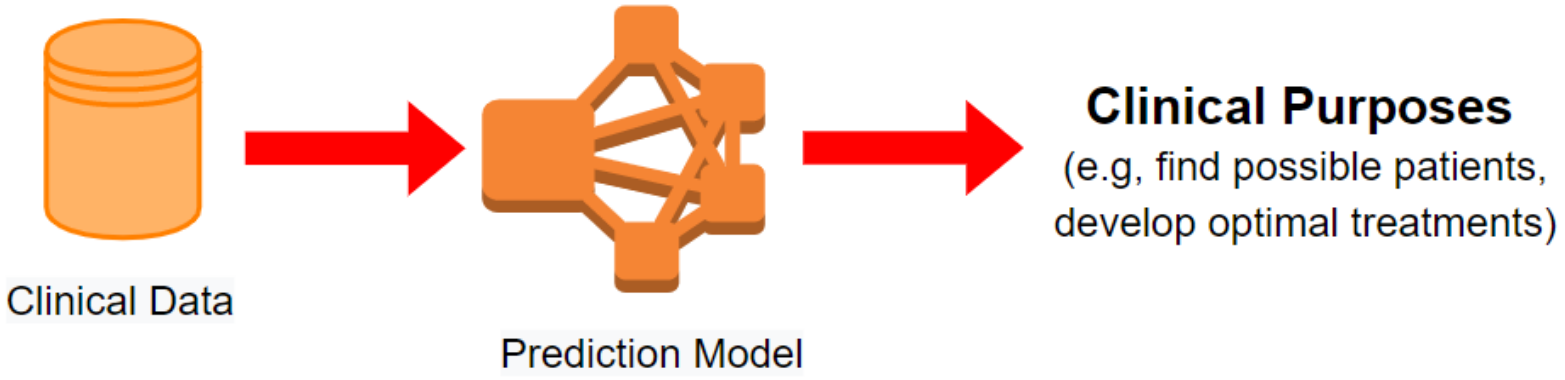
THE UNIVERSITY OF SYDNEY

Introduction

Clinical prediction is one of the most important branches of healthcare data analytics.

For possible patient, predicting which person will develop a disease or a complication will allow the healthcare team to try and prevent those events.

For treatment, clinical prediction models focus on predicting which treatments will work best for a patient or cause the least harm, which helps the healthcare team to develop an optimal treatment plan.



Introduction

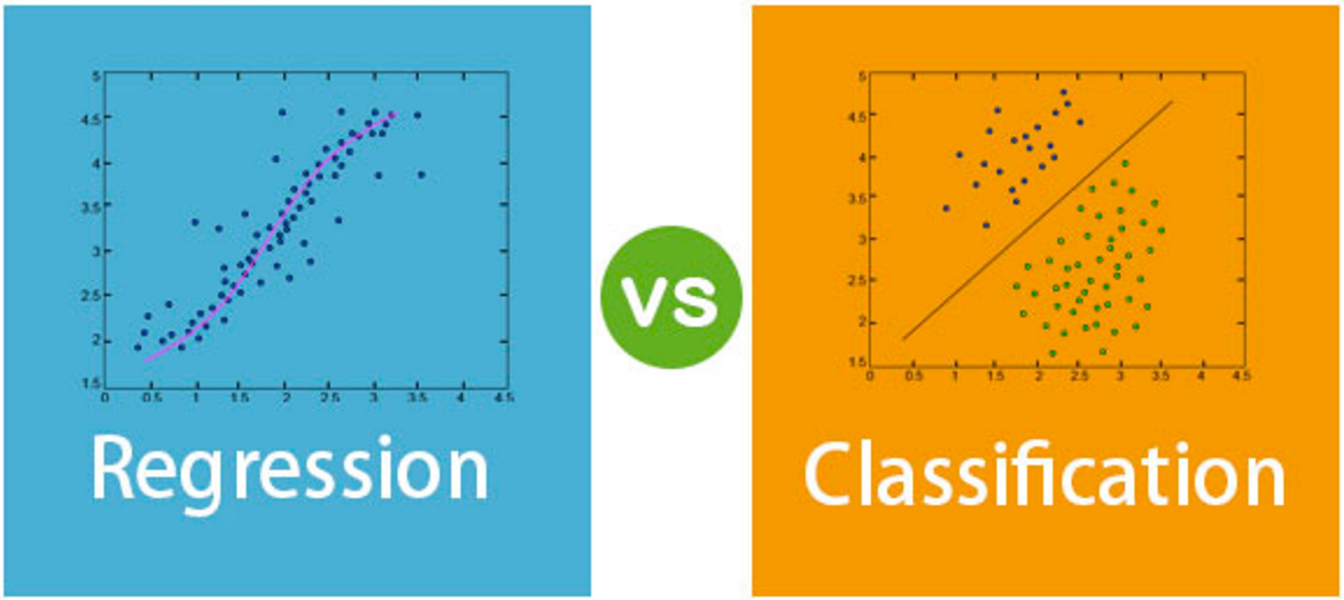
In this lecture, we will provide a relatively comprehensive review of the **supervised learning methods** that have been employed successfully for clinical prediction tasks.

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples [1].

Generally, supervised learning methods can be broadly classified into two categories: **classification** and **regression**. Both of these two classes of techniques focus on **discovering the underlying relationship between covariate variables**, which are also known as attributes and features, and a dependent variable (outcome).

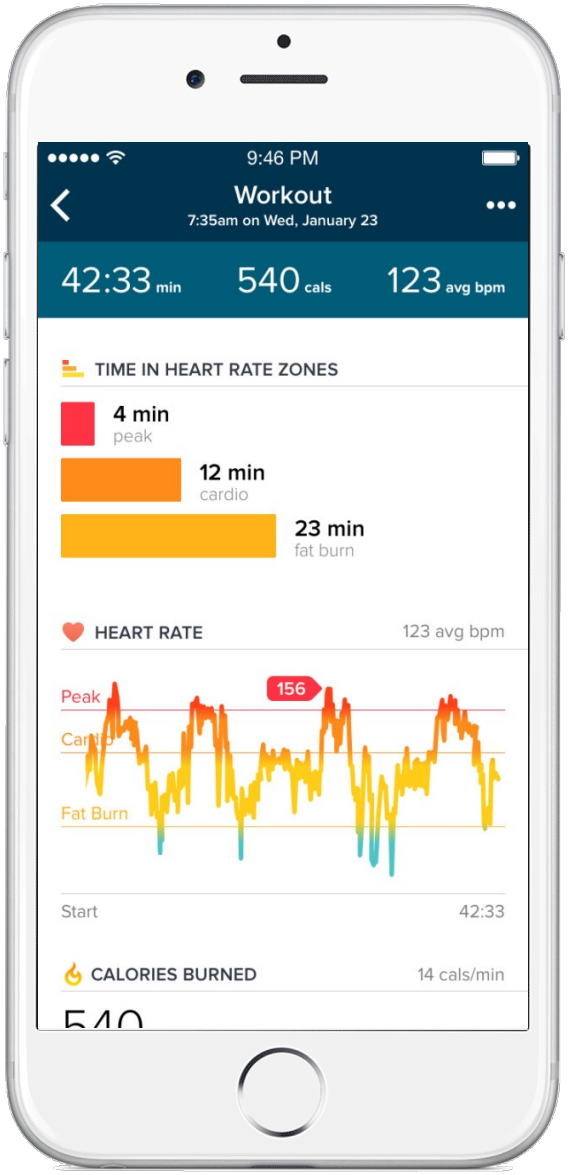
Introduction

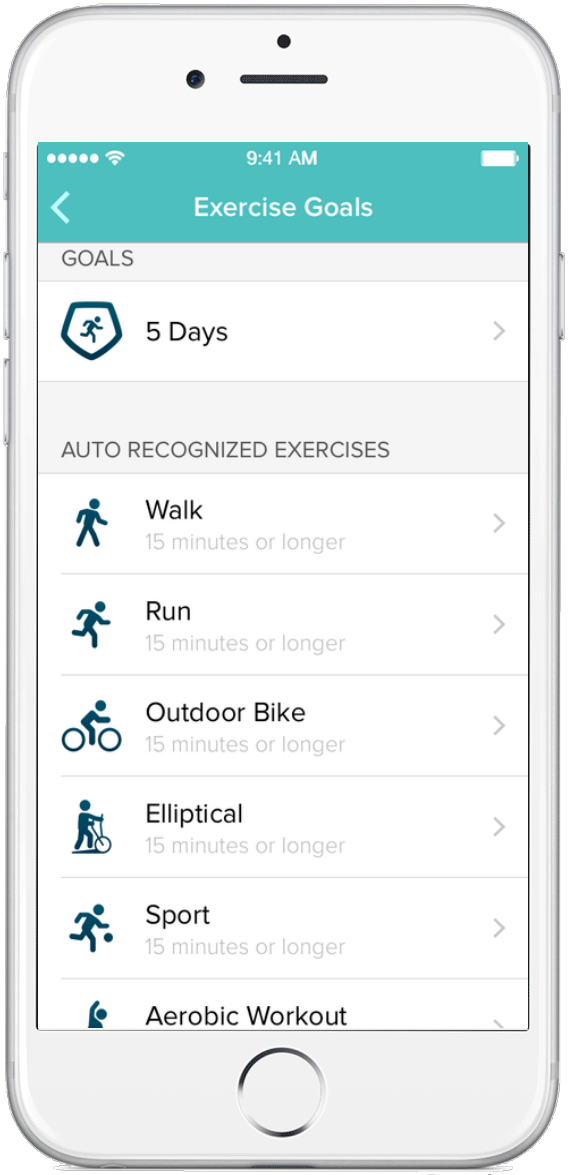
The main difference between these two approaches is that **a classification model generates class labels,** for example, to classify COVID-Positive or COVID-Negative, while **a regression model predicts continuous numerical outcomes**: to get a socre representing the risk of developing coronary heart disease.



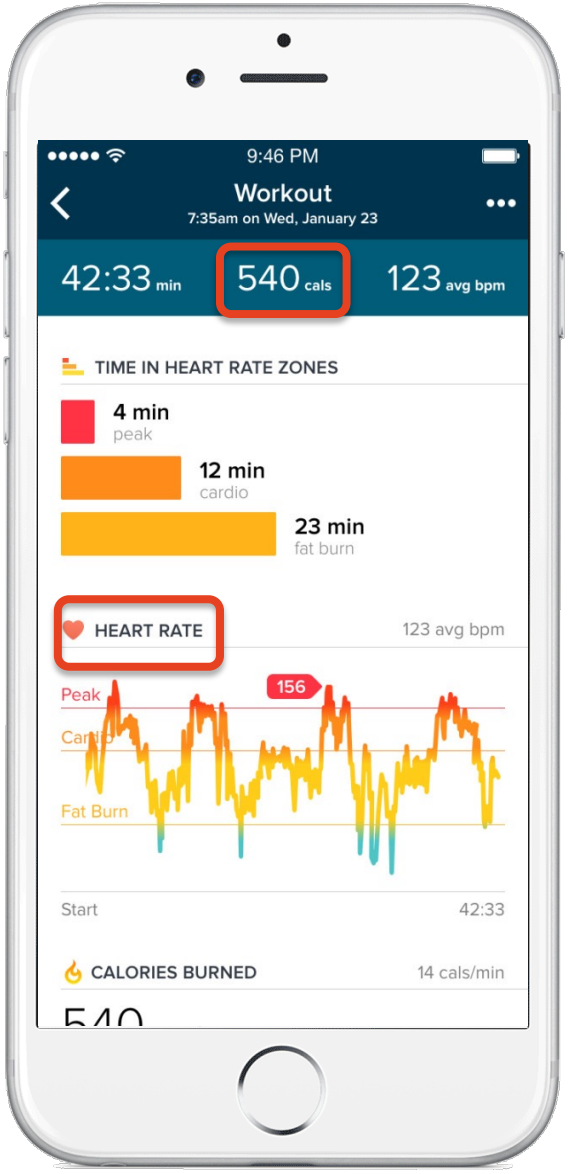
Regression

Health and fitness guidance - fitbit





Estimate how many calories do you burn?



- This is a regression problem.

- Input data $x \in \mathbb{R}^4$:
  - Duration of exercise
  - Average heart rate
  - Your weight in kilograms
  - Your age

- Target output y:
  - Number of calories burned

Machine Learning is so simple ……

Step 0: collect a dataset

$$\{\underbrace{(\boldsymbol{x}^{\boldsymbol{1}}, y^{1}), (\boldsymbol{x}^{\boldsymbol{2}}, y^{2}), \ldots, (\boldsymbol{x}^{\boldsymbol{N}}, y^{N})}_{\text{Training}}, \underbrace{(\boldsymbol{x}^{\boldsymbol{N+1}}, y^{N+1}), (\boldsymbol{x}^{\boldsymbol{N+2}}, y^{N+2}), \ldots, (\boldsymbol{x}^{\boldsymbol{N+M}}, y^{N+M})}_{\text{Test}}\}$$

Step 1: function with unknown

Step 2: define loss from training data

Step 3: optimization

Step 4: Test (Evaluation)

Basic Models: Linear Regression

Linear regression is a linear approach for modelling the relationship between a scalar response and one or more variables.

A linear prediction model to predict the calories burned:

$$\hat{y} = b + \boldsymbol{w}^T \boldsymbol{x}$$

where $\hat{y}$ is the estimated calories burned, x is the feature vector, and $\boldsymbol{w}$ and b are respectively weight vector and bias to be optimized, respectively.

- Loss: how good a set of values is.

$$L = \frac{1}{N}\sum_{n} e_n$$

$e = |y - \hat{y}|$ L is mean absolute error (MAE)

$e = (y - \hat{y})^2$ L is mean square error (MSE)

Basic Models: Linear Regression

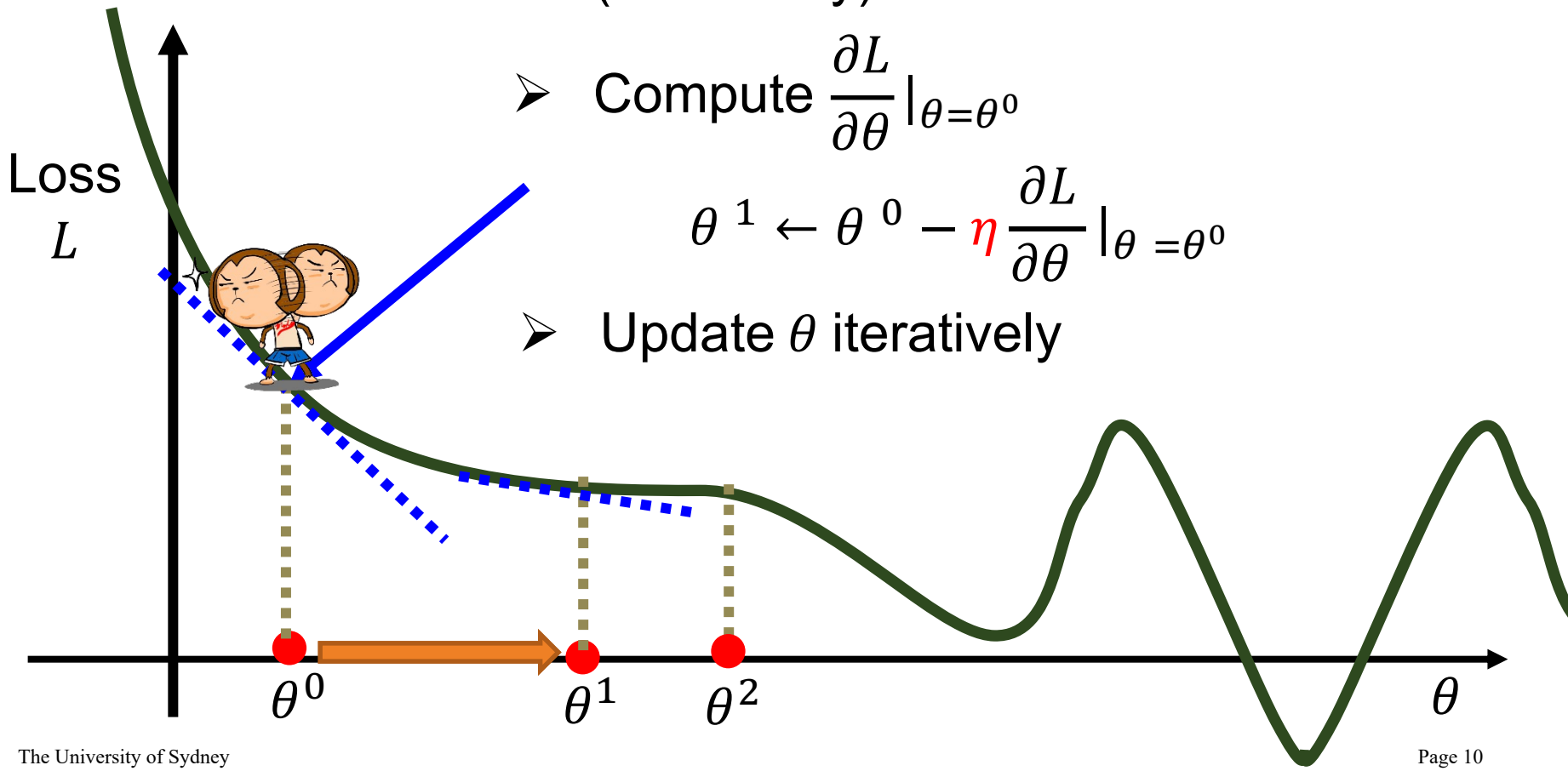
$$\theta^* = arg\min_{\theta} L$$

Gradient Descent

- (Randomly) Pick an initial value θ^0
- Compute $\frac{\partial L}{\partial \theta}|_{\theta=\theta^0}$

$$\theta^1 \leftarrow \theta^0 - \eta \frac{\partial L}{\partial \theta}|_{\theta=\theta^0}$$

- Update θ iteratively



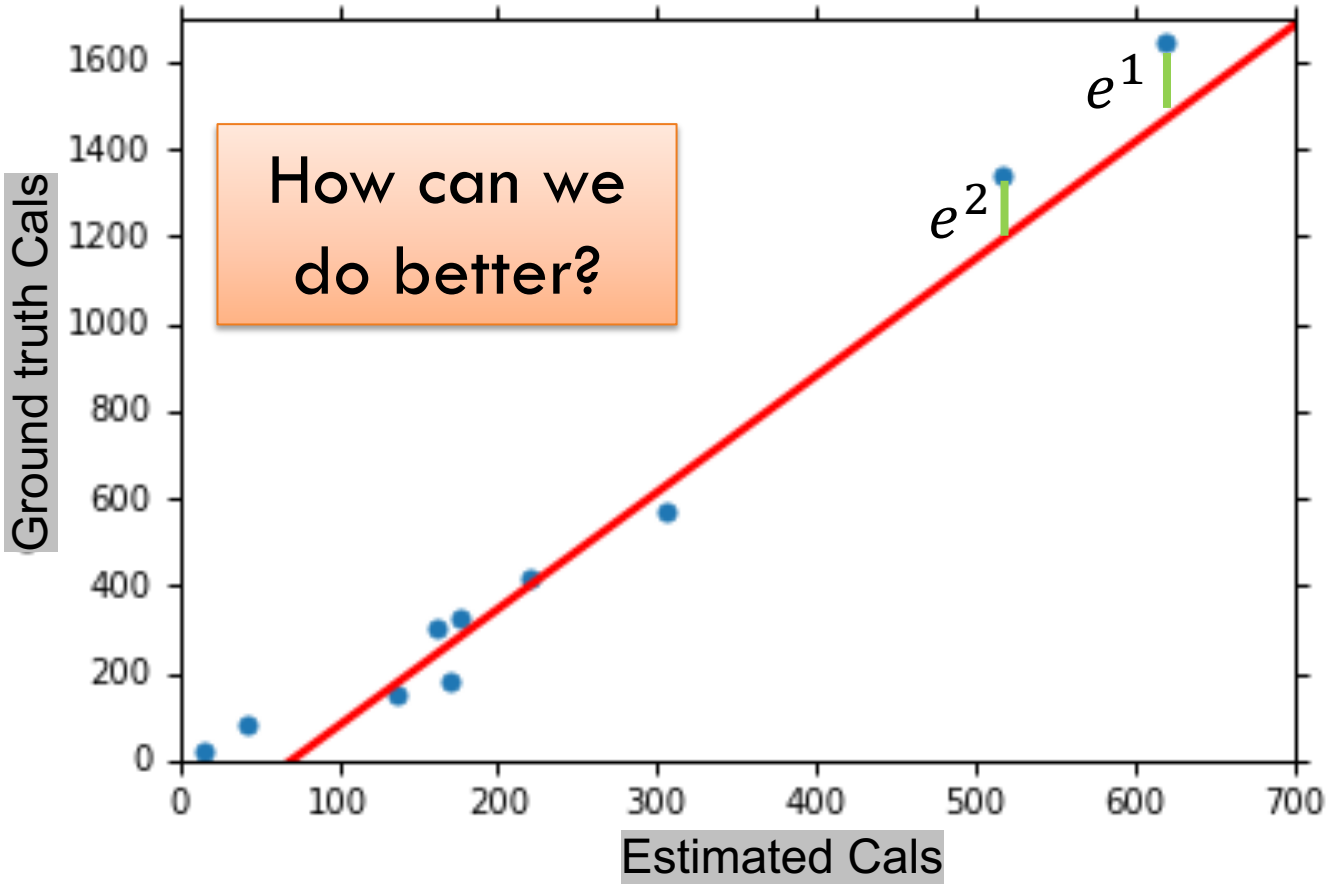
How's the results? - Generalization

$$\hat{y} = b + \boldsymbol{w}^T \boldsymbol{x}$$

Average Error on Training Data

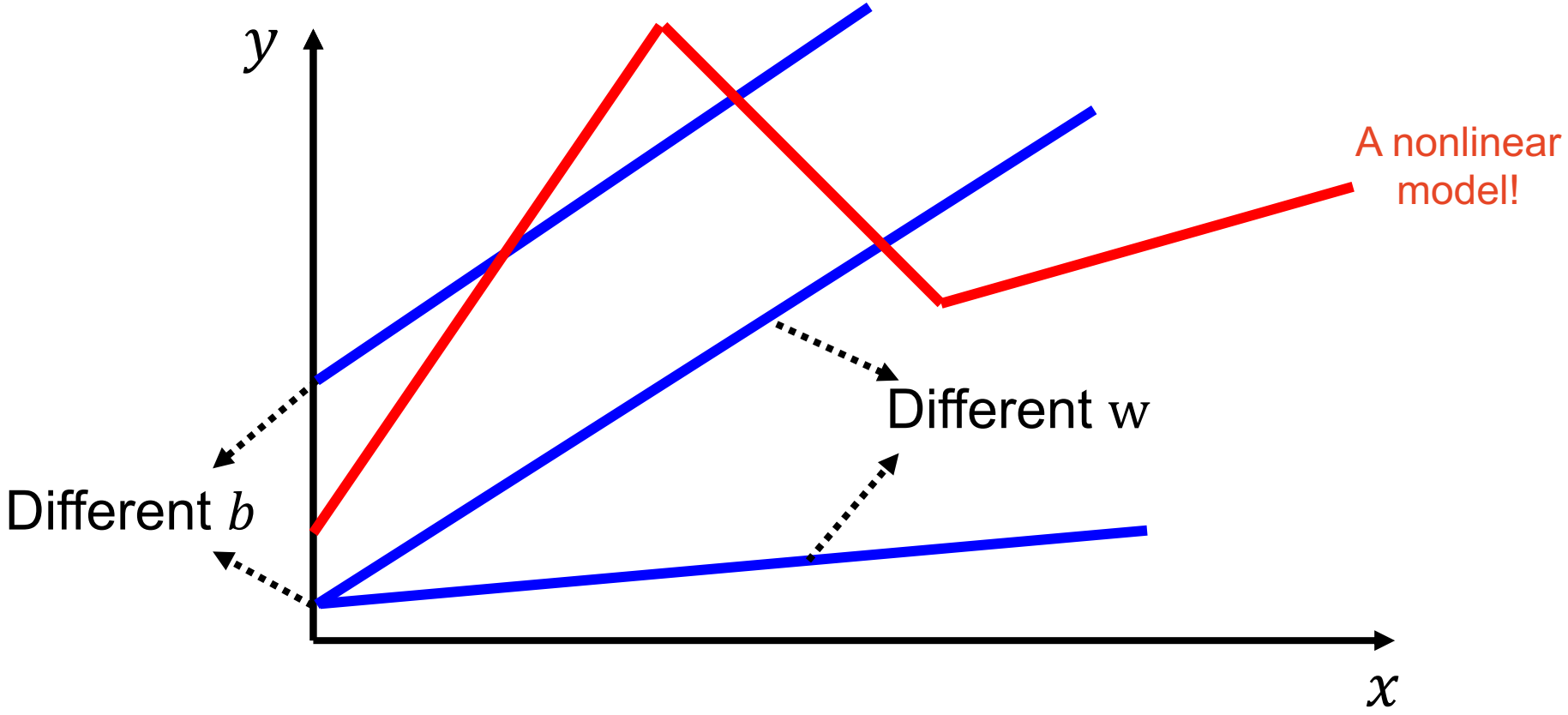
$$= \frac{1}{N} \sum_{i=1}^{N} e^i$$

Another M testing examples



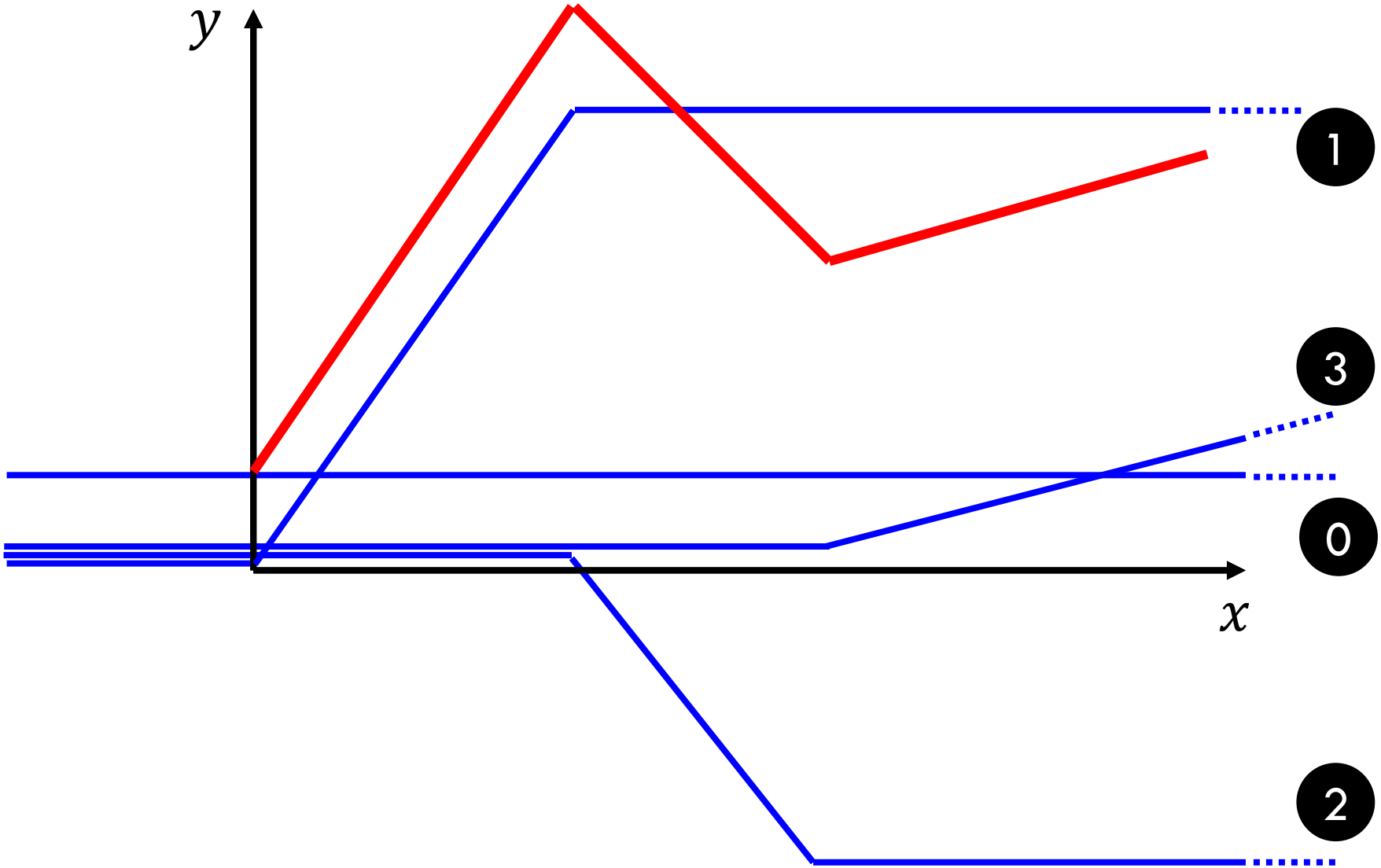
Linear models have severe limitation.

$$\hat{y} = b + \boldsymbol{w}^T \boldsymbol{x}$$



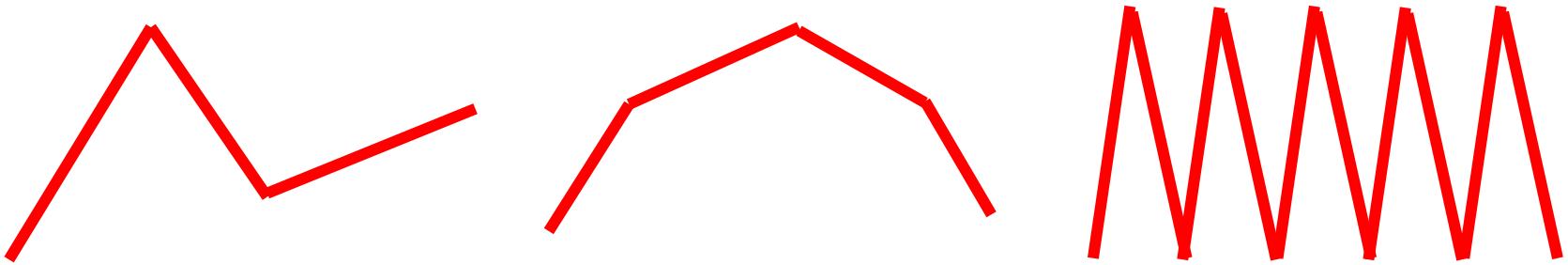
We need a more flexible model!

red curve = constant + sum of a set of (~ step function)



All Piecewise Linear Curves

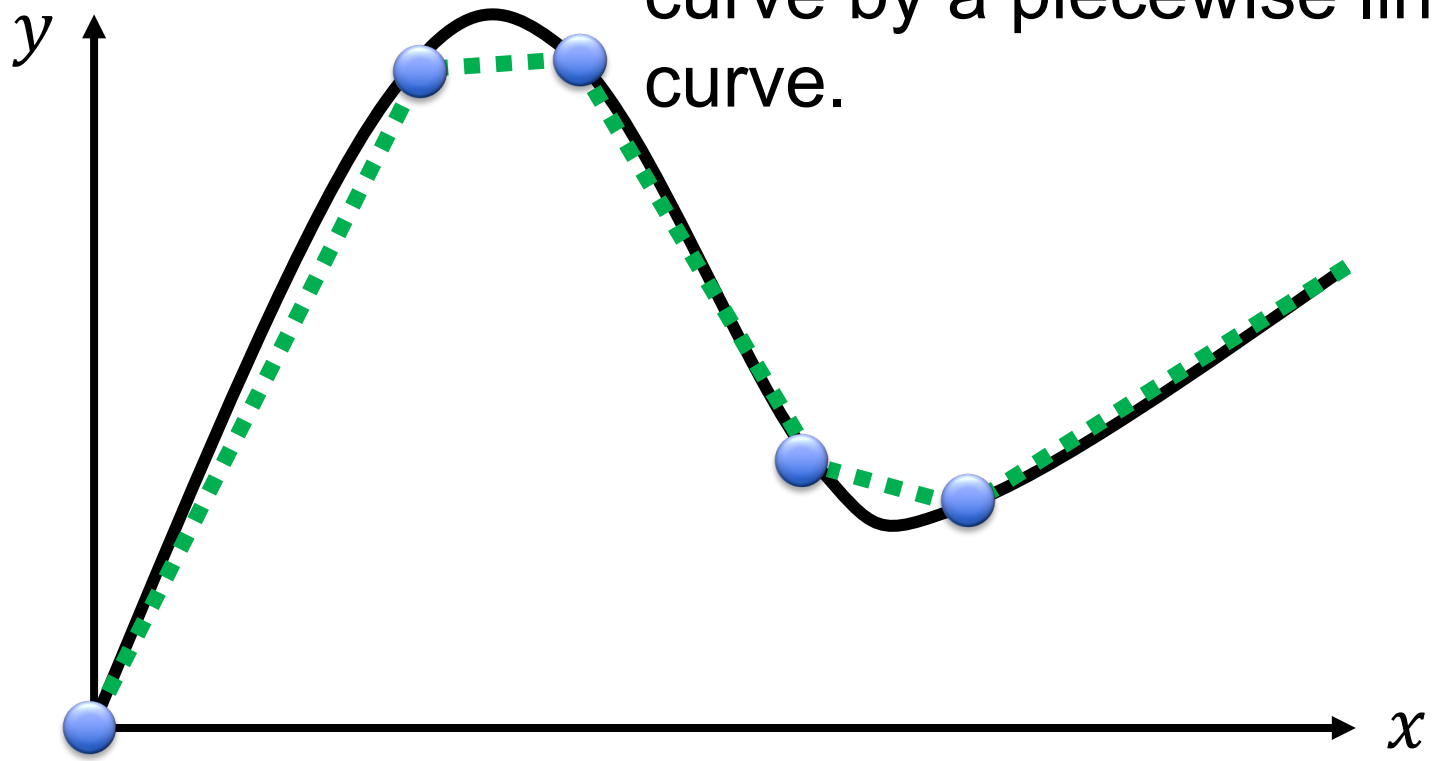
= constant + sum of a set of 



More pieces require more 

Beyond Piecewise Linear?

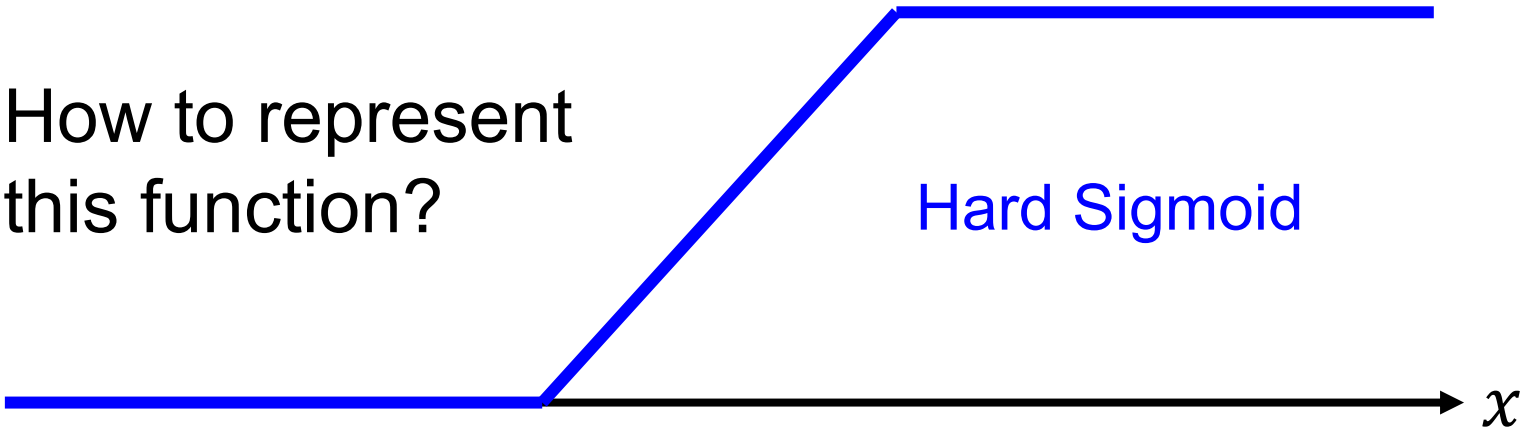
Approximate continuous curve by a piecewise linear curve.

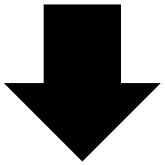


To have good approximation, we need sufficient piece

red curve = constant + sum of a set of 

How to represent this function?

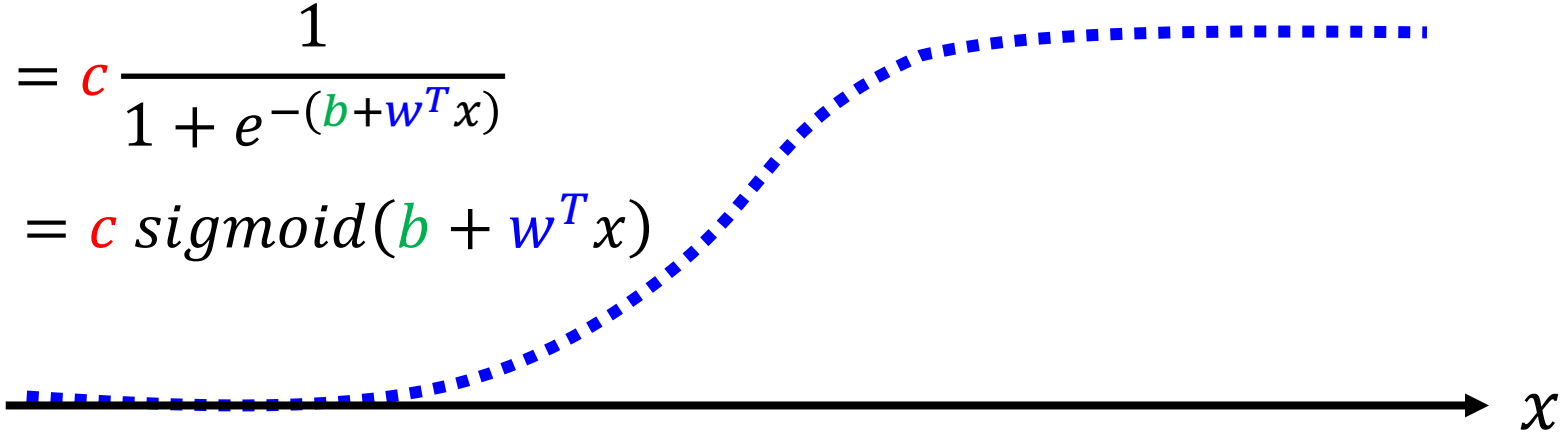




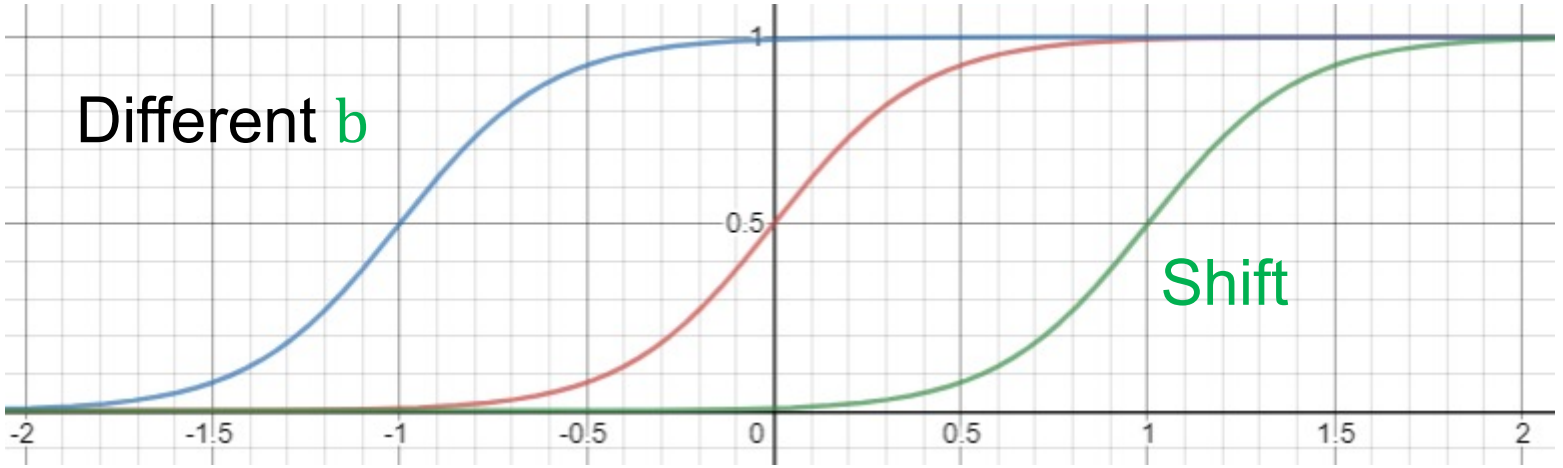
Sigmoid Function

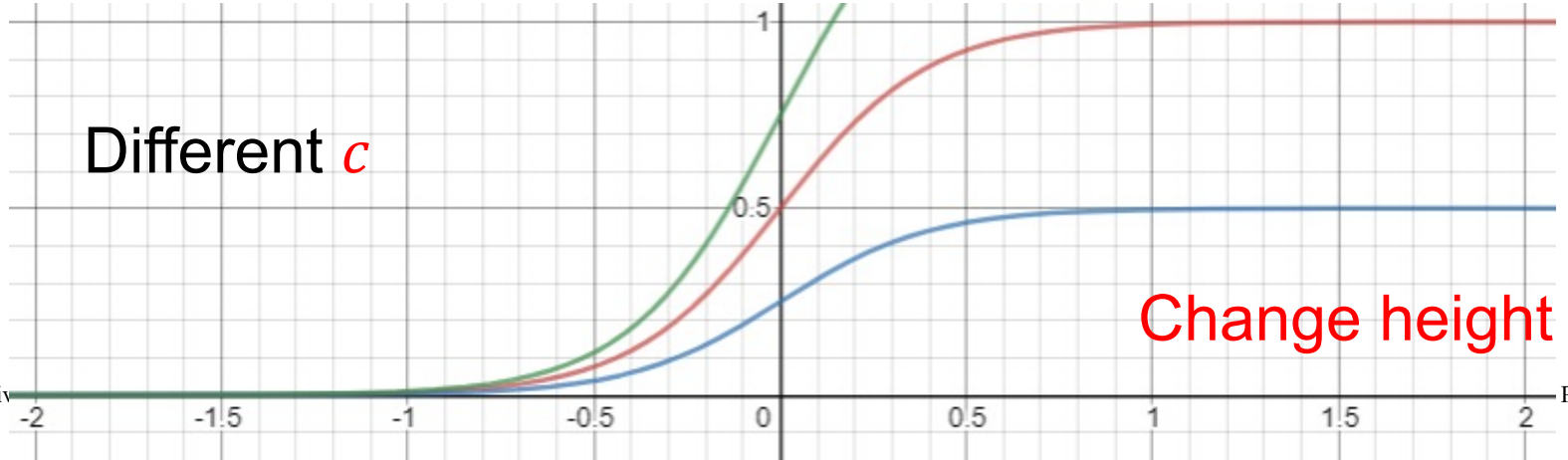
$$y = c\frac{1}{1+e^{-(b+w^T x)}}$$

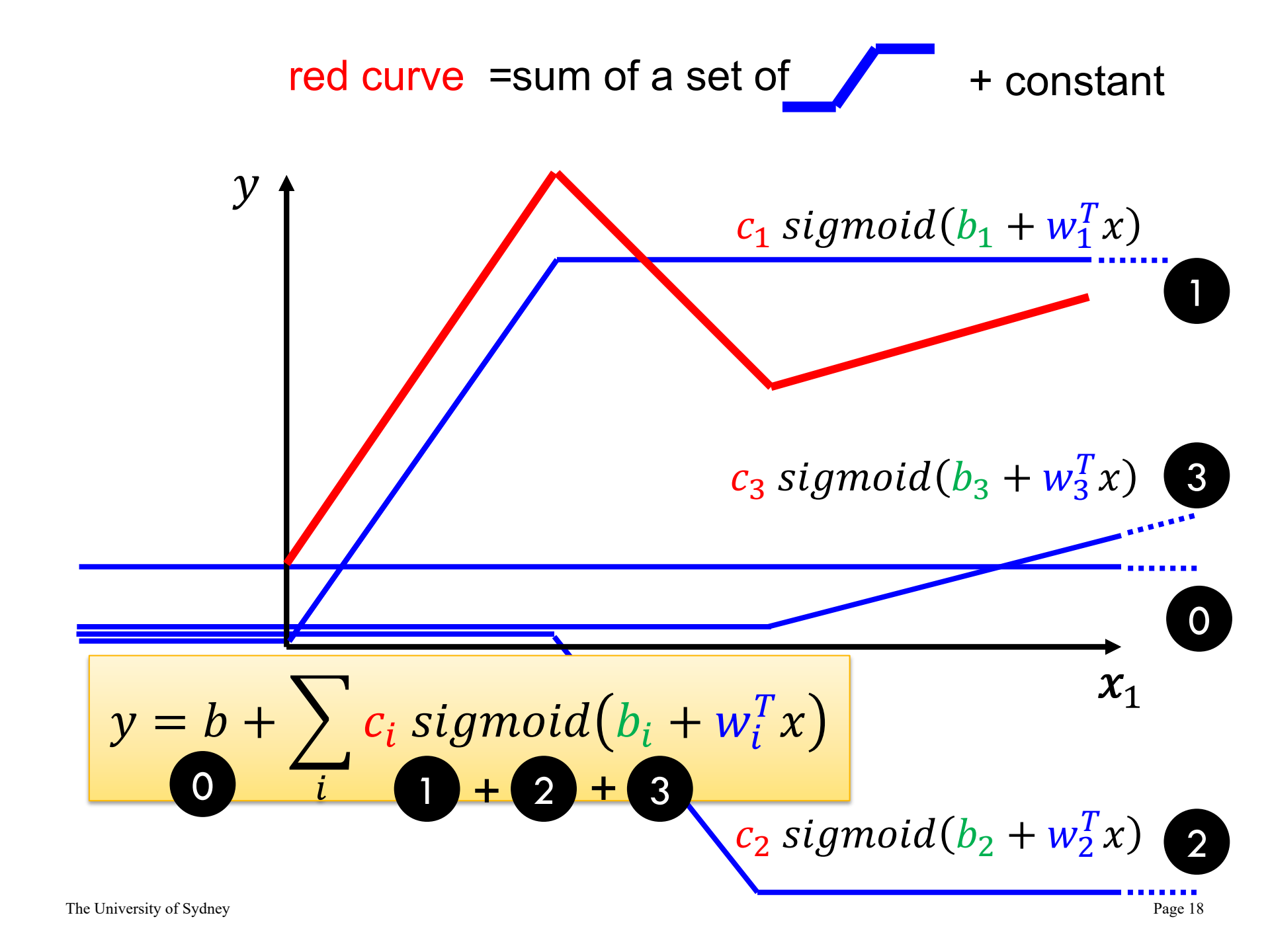
$$= c\ sigmoid(b + w^T x)$$

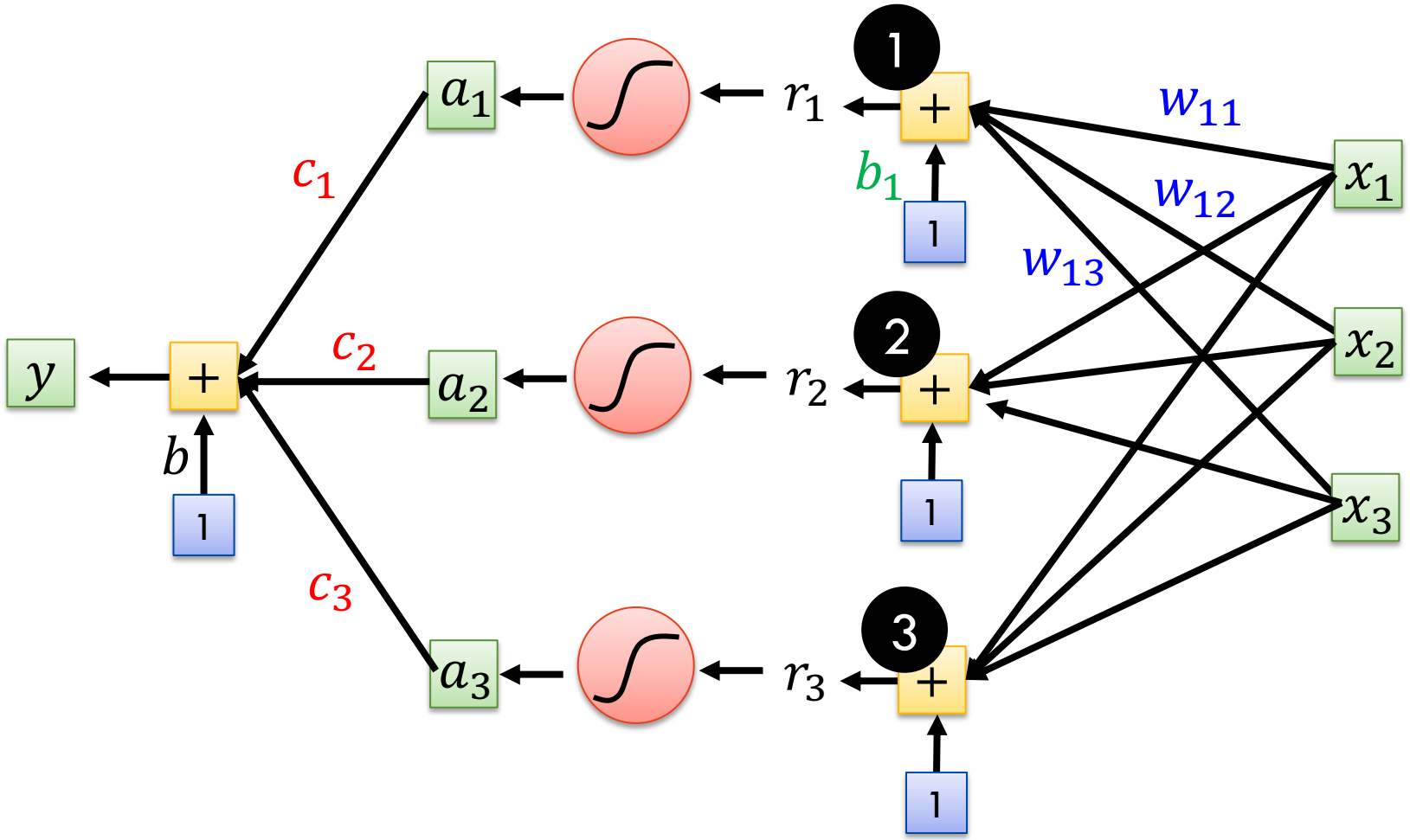



Different w
Change slopes
1
0.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


Different b
Shift
1
0.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


Different c
Change height
1
0.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


red curve =sum of a set of
+ constant
y
$c_1\ sigmoid(b_1 + w_1^T x)$
1
$c_3\ sigmoid(b_3 + w_3^T x)$
3
0
x_1
$y = b + \sum_i c_i\ sigmoid(b_i + w_i^T x)$
0
1 + 2 + 3
$c_2\ sigmoid(b_2 + w_2^T x)$
2

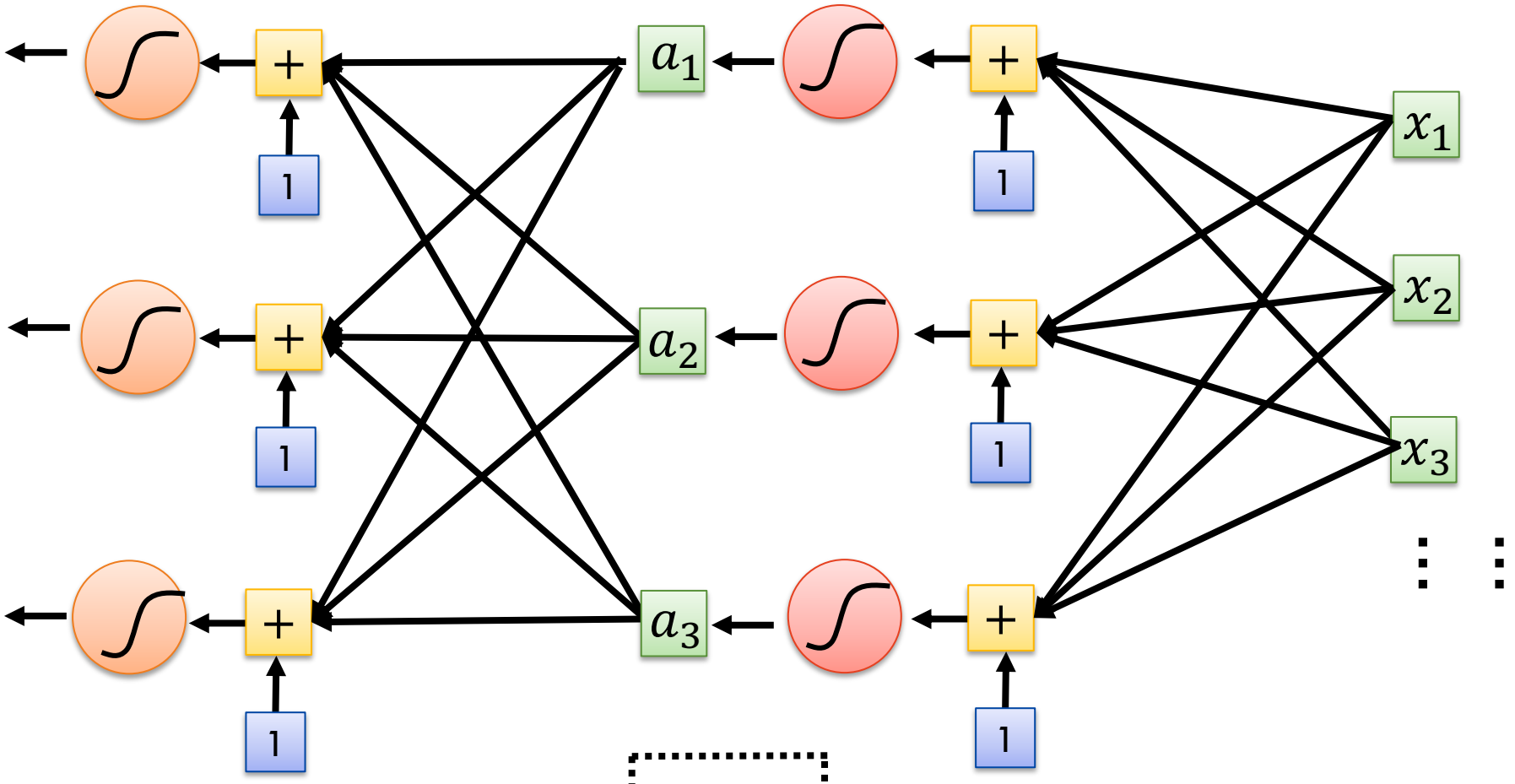


$$y = b + \boldsymbol{c}^T \sigma(\boldsymbol{b} + W\boldsymbol{x})$$

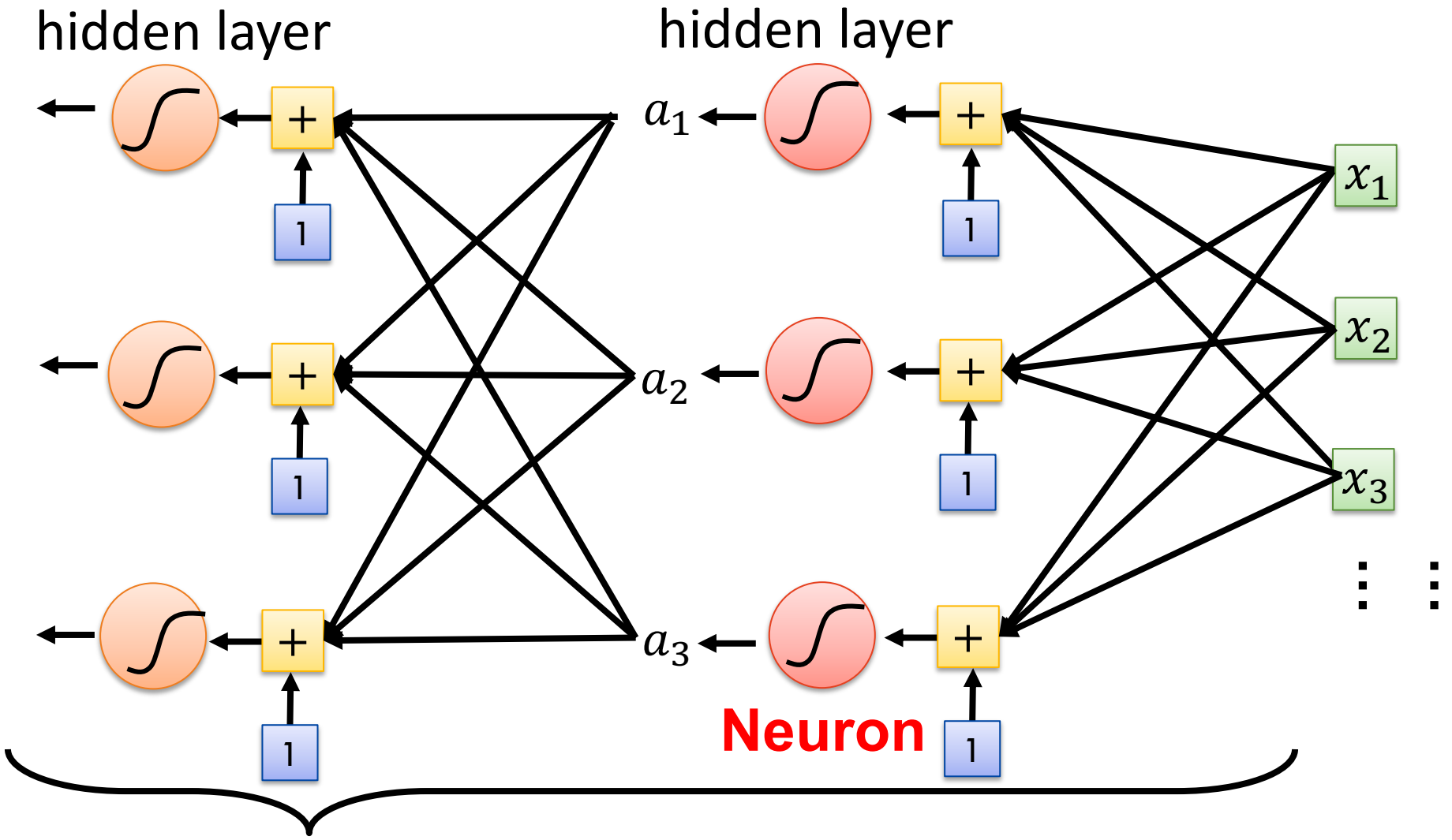
Loss e

$\hat{y}$

Multi-layer Perceptron



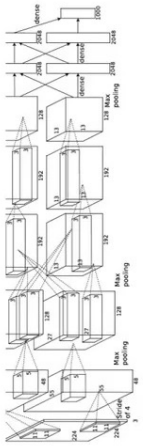
$$\boldsymbol{a}' = \sigma(\boldsymbol{b}' + W'\boldsymbol{a}) \qquad \boldsymbol{a} = \sigma(\boldsymbol{b} + W\boldsymbol{x})$$



Many layers means **Deep** ➡ Deep Learning

Deep = Many hidden layers

8 layers



16.4%

AlexNet (2012)

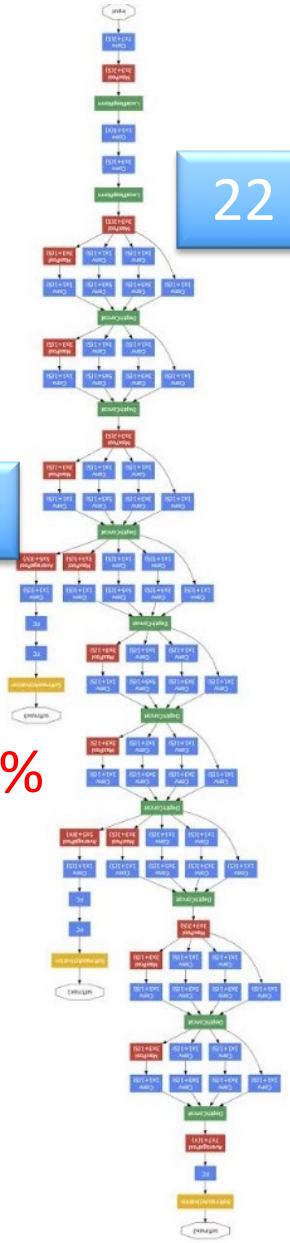
19 layers

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

7.3%

VGG (2014)

22 layers



6.7%

GoogleNet (2014)

Classification

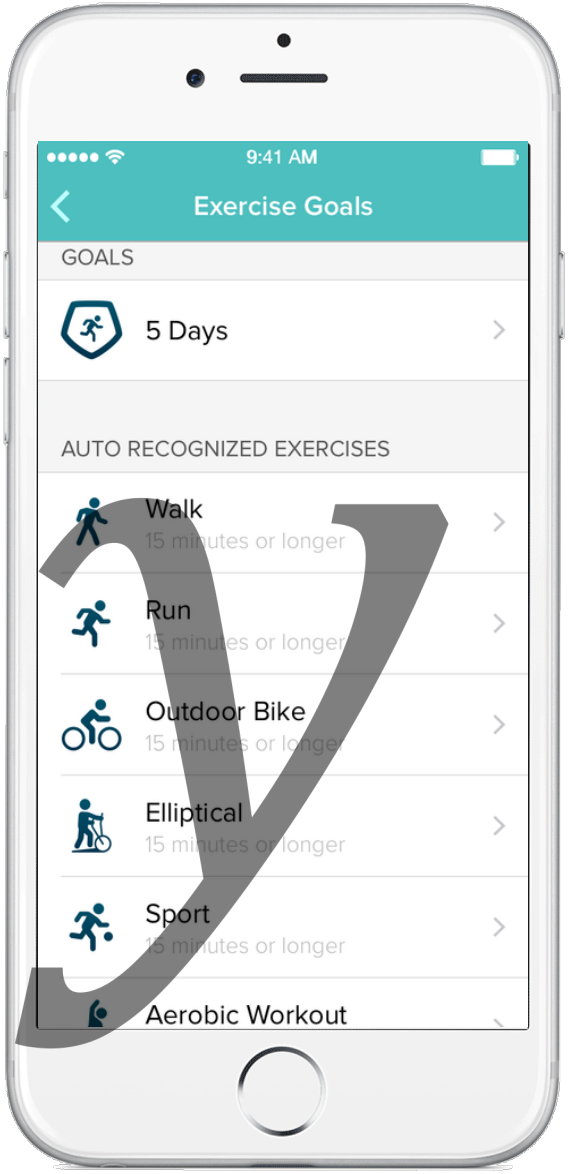
Support Vector Machine
K-Nearest Neighbor

Estimate your exercises types?

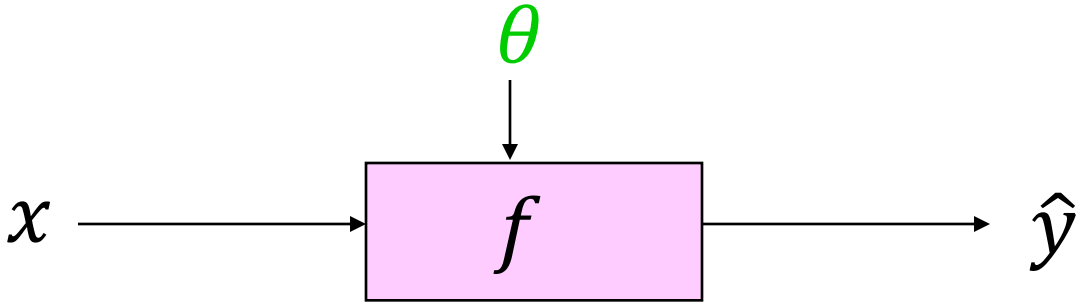
This is a classification problem.

Fitbit contains the following sensors:

- 3-axis accelerometer, which tracks motion patterns
- Gyroscope
- Altimeter, which tracks altitude changes
- Built-in GPS receiver + GLONASS, which tracks your location during a workout
- Multi-path optical heart rate tracker
- Multipurpose electrical sensors compatible with the ECG app and EDA Scan app
- On-wrist skin temperature sensor

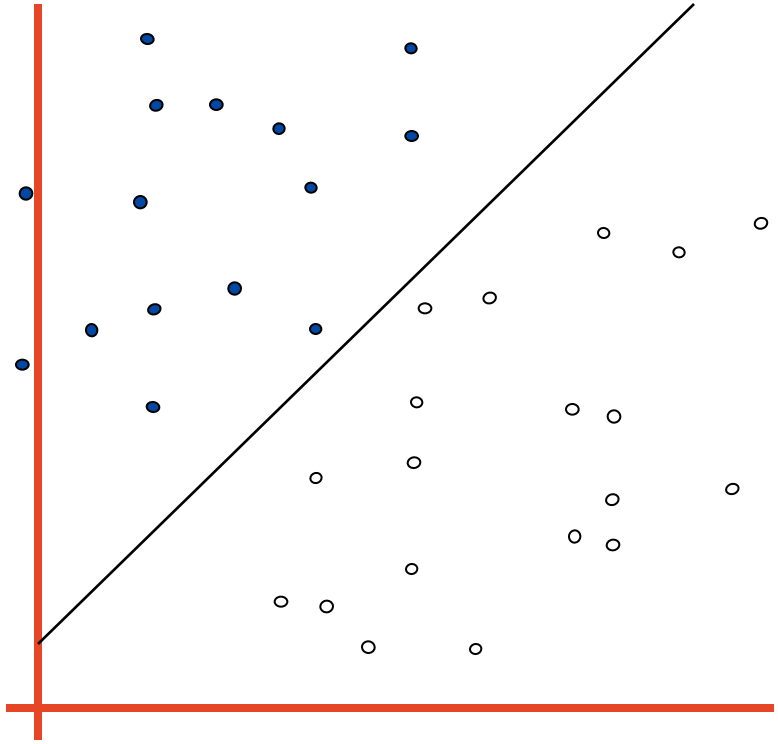


Linear Classifiers



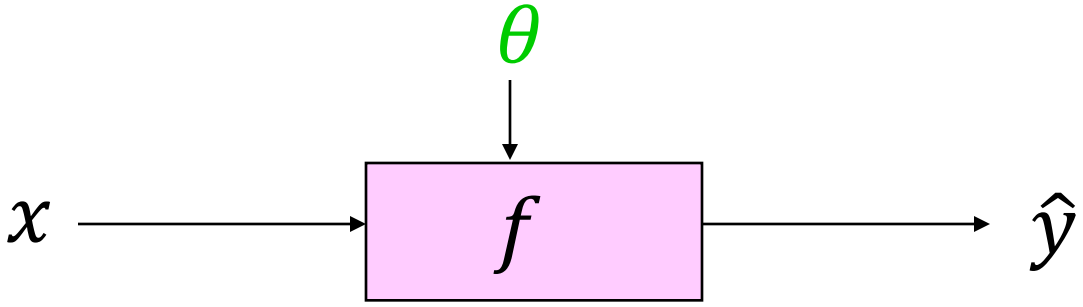
$$f(x, w, b) = sign(w^T x + b)$$

- denotes +1
- denotes -1



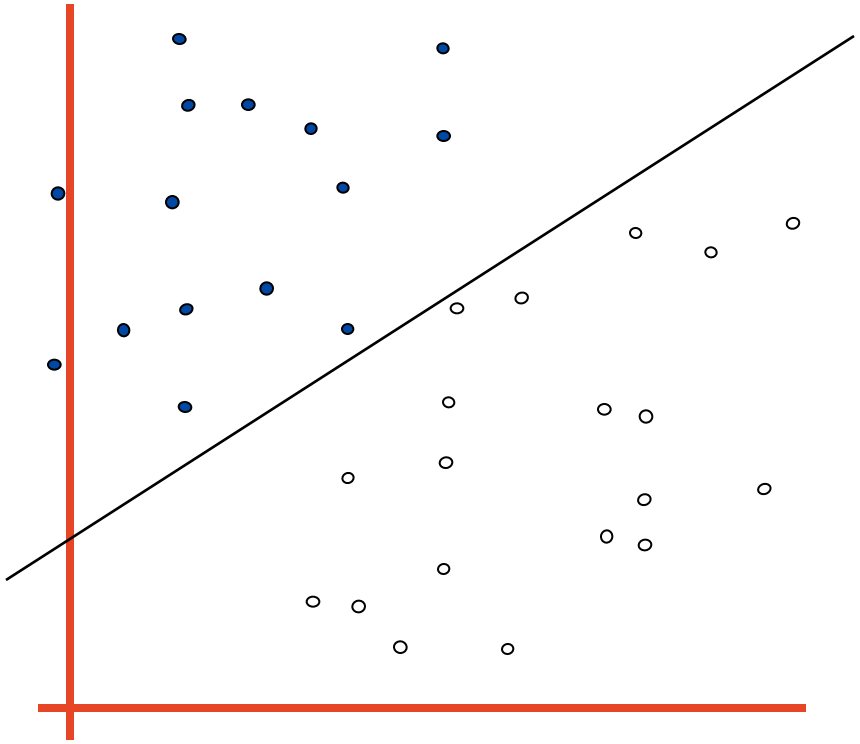
How would you classify this data?

Linear Classifiers



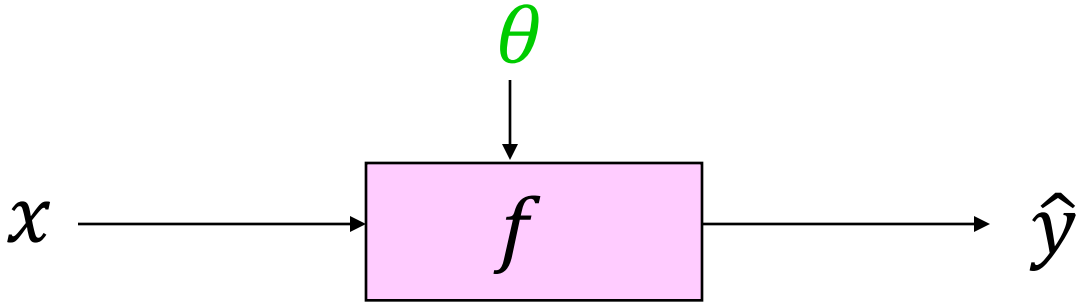
$$f(x, w, b) = sign(w^T x + b)$$

- denotes +1
- denotes -1

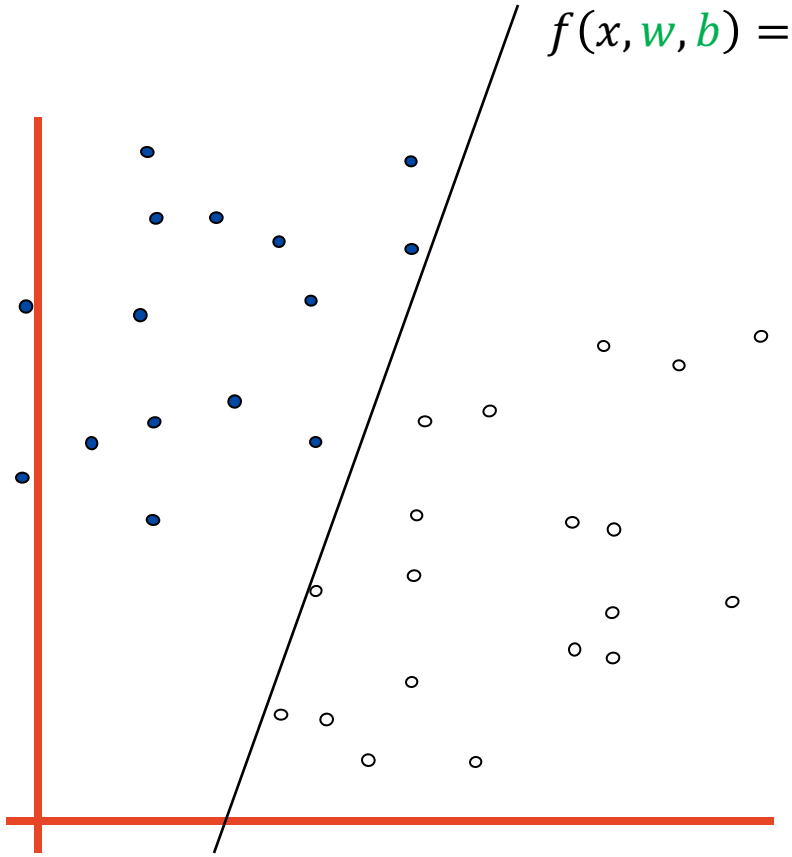


How would you classify this data?

Linear Classifiers

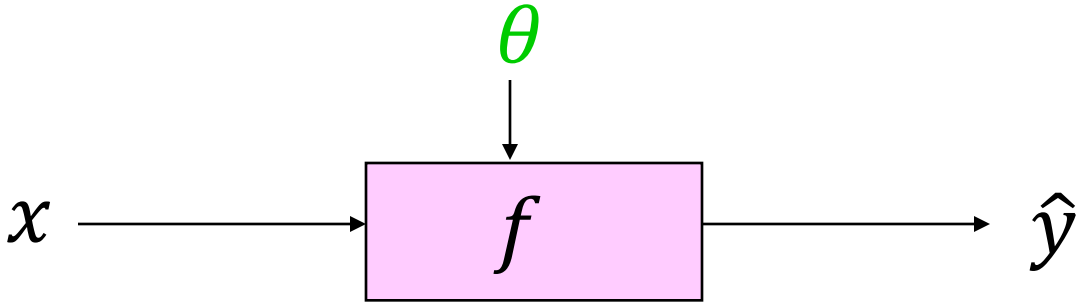


- denotes +1
- denotes -1



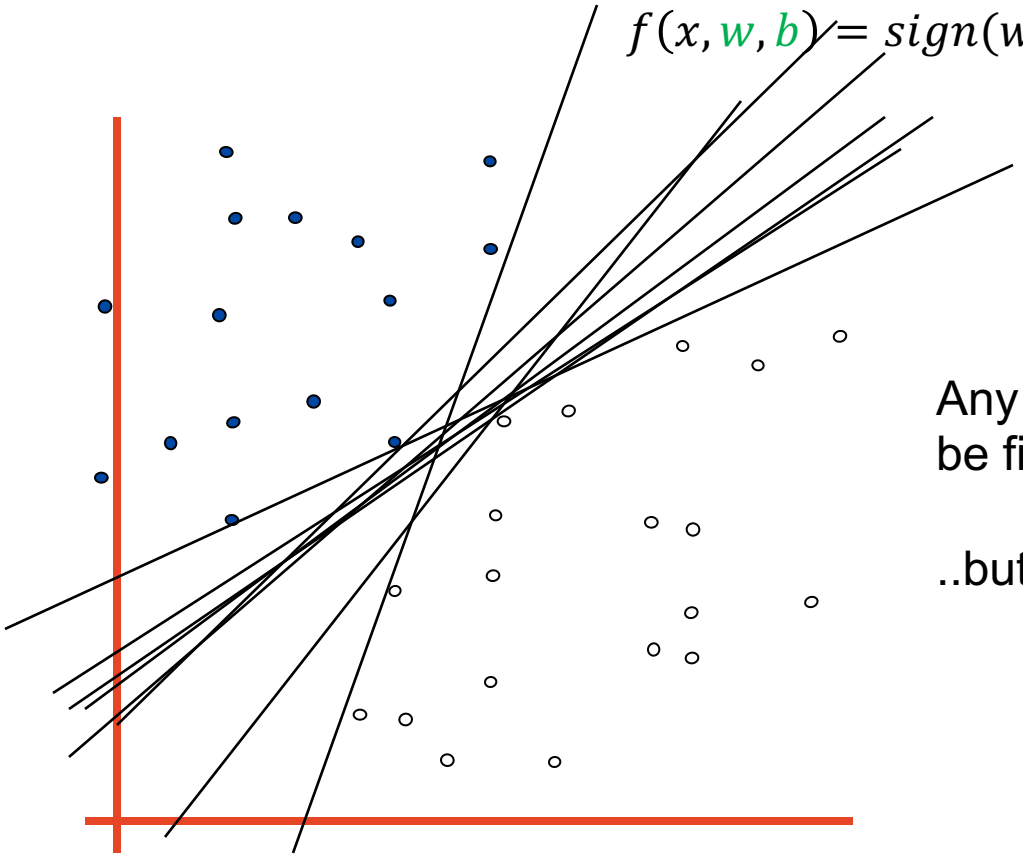
How would you classify this data?

Linear Classifiers



- denotes +1
- denotes -1

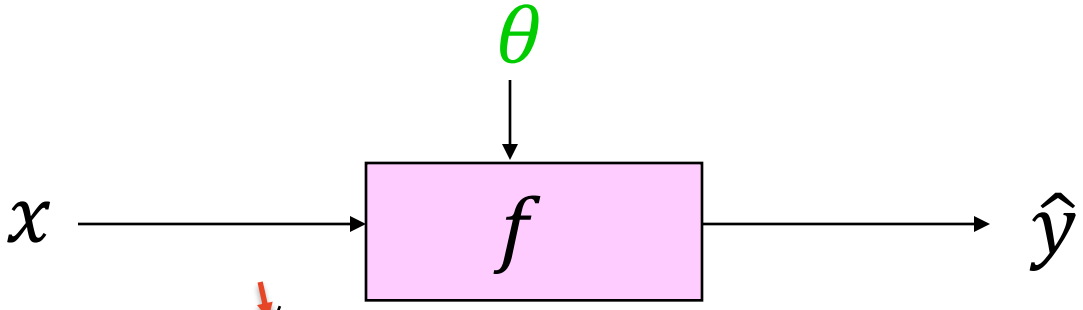
$$f(x, w, b) = sign(w^T x + b)$$



Any of these would be fine..

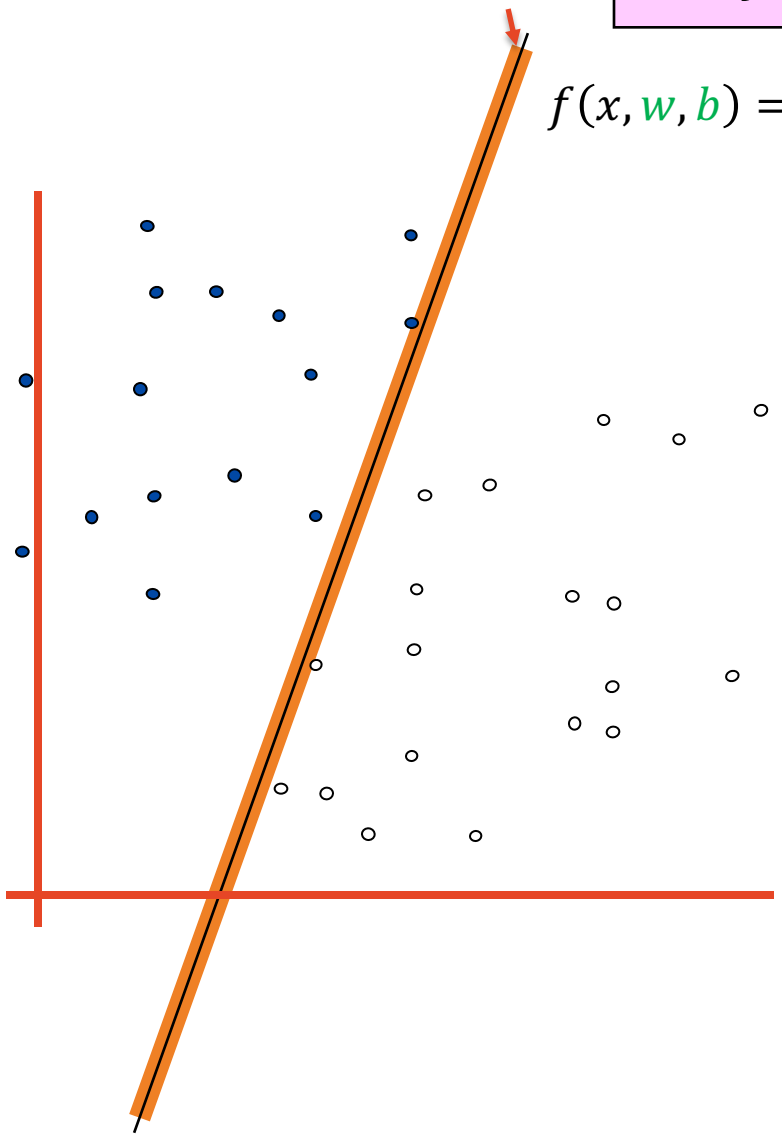
..but which is best?

Classifier Margin



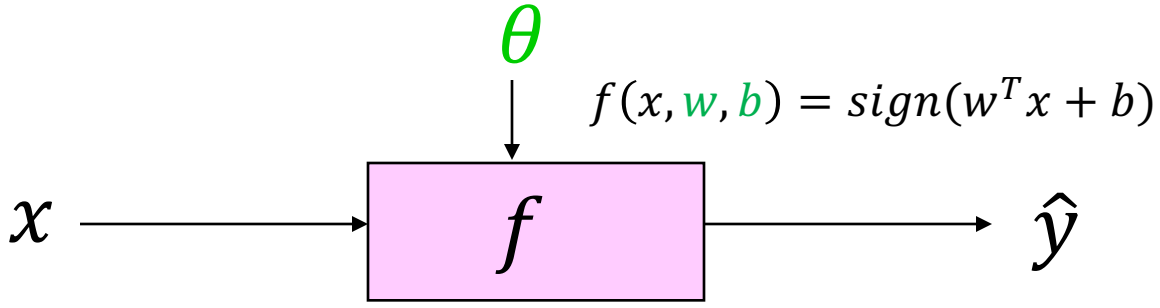
$f(x, w, b) = sign(w^T x + b)$

- denotes +1
- denotes -1



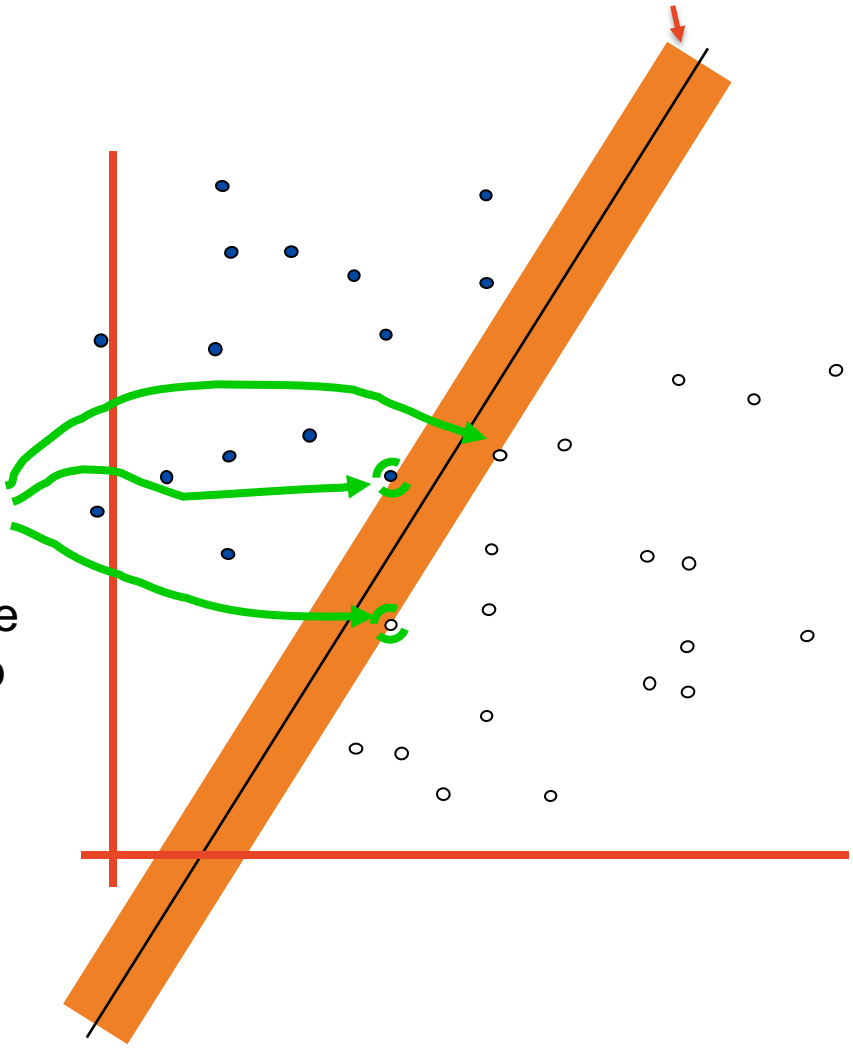
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



- denotes +1
- denotes -1

Support Vectors are those datapoints that the margin pushes up against

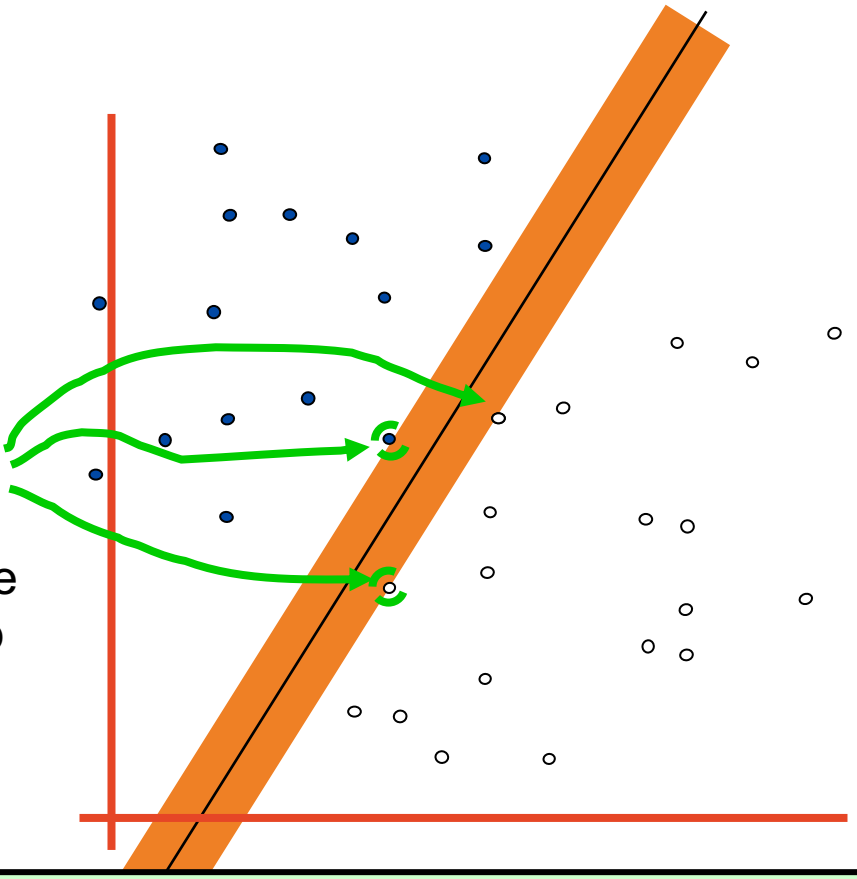


The maximum margin linear classifier is the linear classifier with the, um, maximum margin. This is the simplest kind of SVM – linear SVM.

Maximum Margin

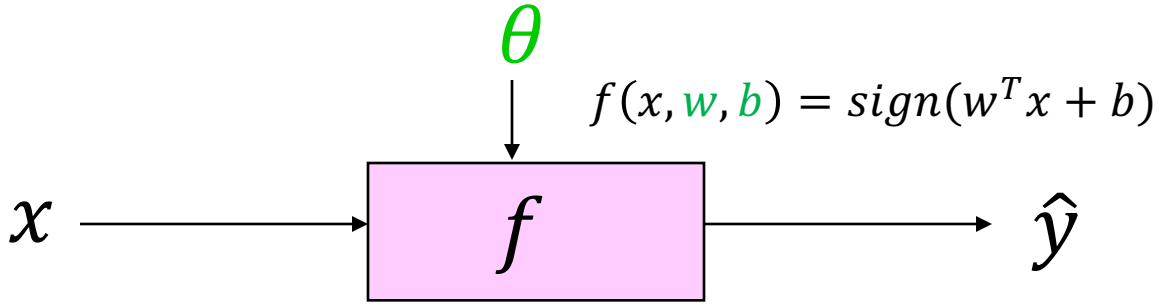
- denotes +1
- denotes -1

Support Vectors are those datapoints that the margin pushes up against

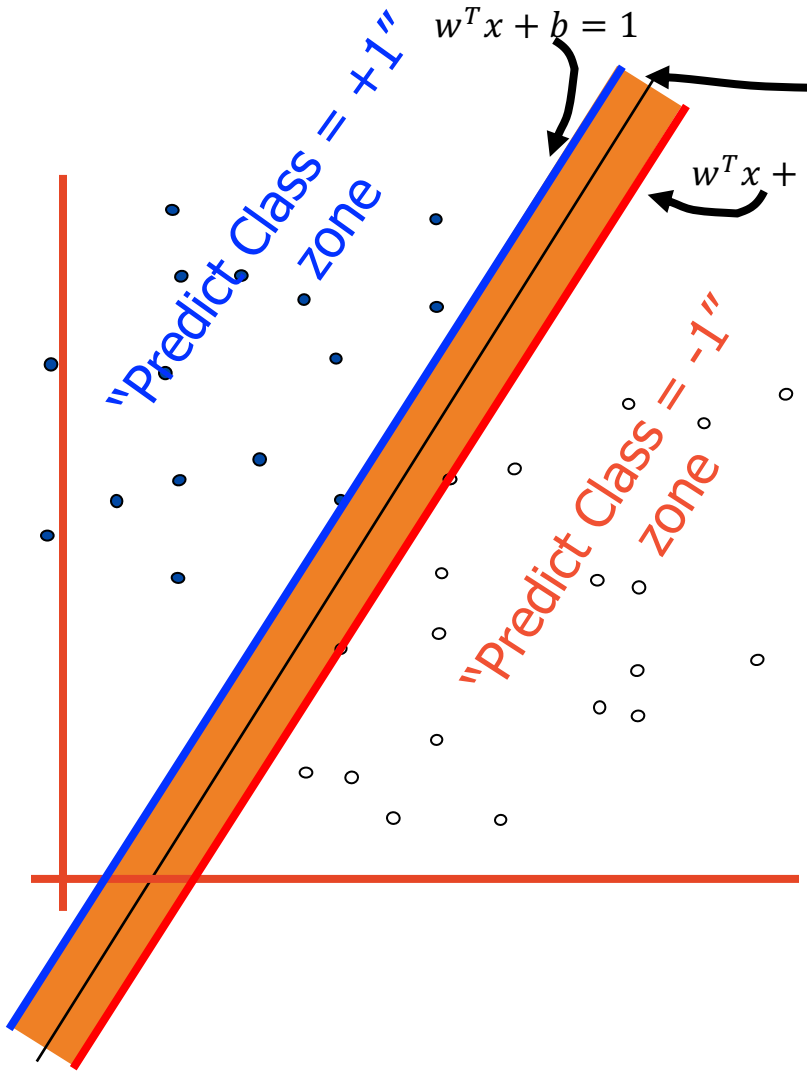


1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of causing a misclassification.
3. LOOCV is easy since the model is immune to removal of any non-support-vector datapoints.
4. There's some theory (using VC dimension) that is related to (but not the same as) the proposition that this is a good thing.
5. Empirically it works very very well.

Maximum Margin



- denotes +1
- denotes -1



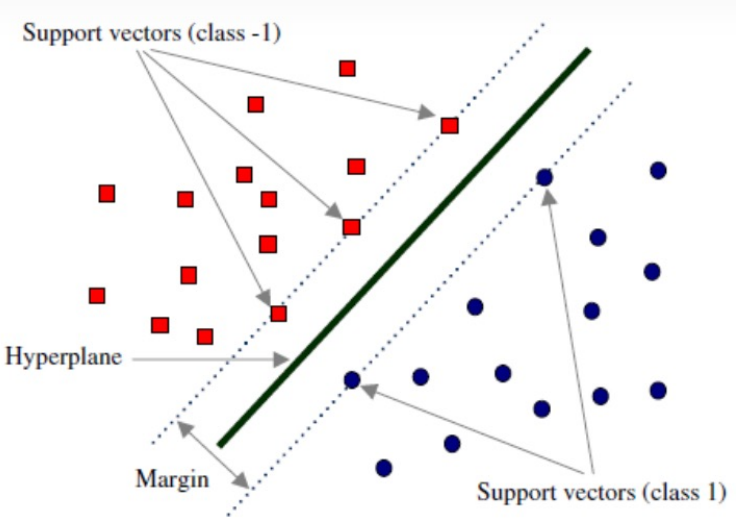
$$y_i(w^T x_i + b) \geq 1$$

	$w^T x_i + b = 2$	$w^T x_i + b = -2$
$y_i = +1$	≥ 1	-2
$y_i = -1$	-2	≥ 1

Support Vector Machine

The main idea of SVM is to find a hyperplane that separate the samples from two classes. To find the best hyperplane, we want to make the margin of samples from different classes as large as possible.

We define the hyperplane as $w^T x + b = 0$.



The margin is

$$margin = \frac{2}{||w||}$$

Maximizing it means to minimize $\frac{1}{2}||w||^2$.

Thus, the objective is

$$\min_{W,b} \quad \frac{1}{2}||w||^2$$

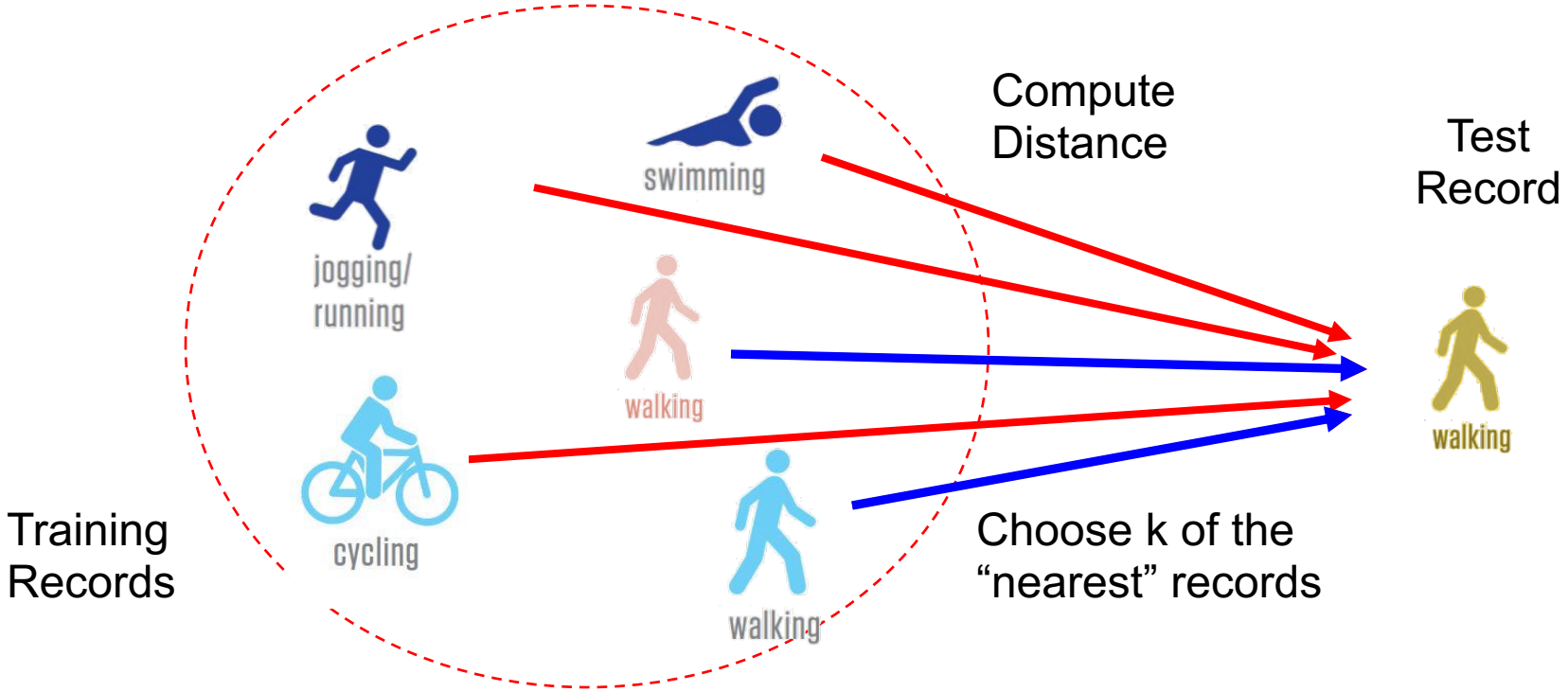
$$\text{s.t.} \quad y_i(w^T x_i + b) \geq 1, i = 1,2,\cdots,n.$$

We can use Lagrange multiplier method to solve W and b of the above optimization question.

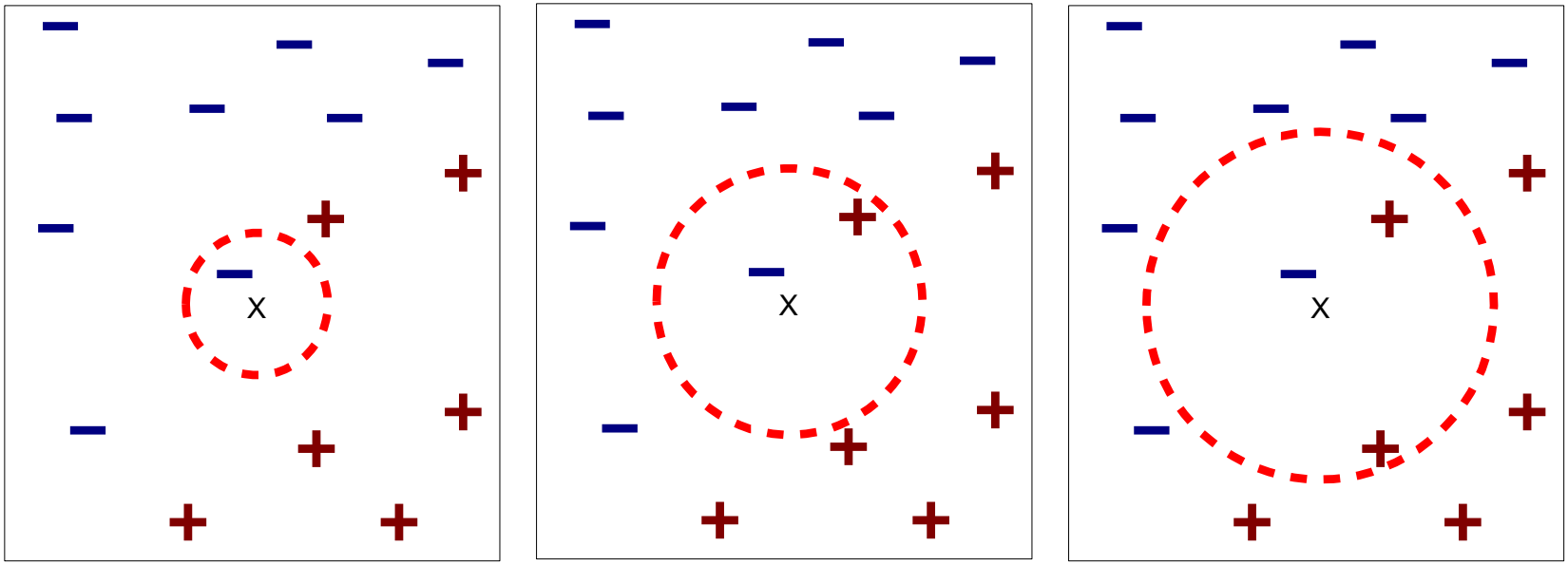
K-Nearest Neighbor Classifiers

Nearest Neighbor Classifiers

- k-NN classification rule is to assign to a test sample the majority category label of its k nearest training samples
- In practice, k is usually chosen to be odd, so as to avoid ties
- The $k = 1$ rule is generally called the nearest-neighbor classification rule



Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

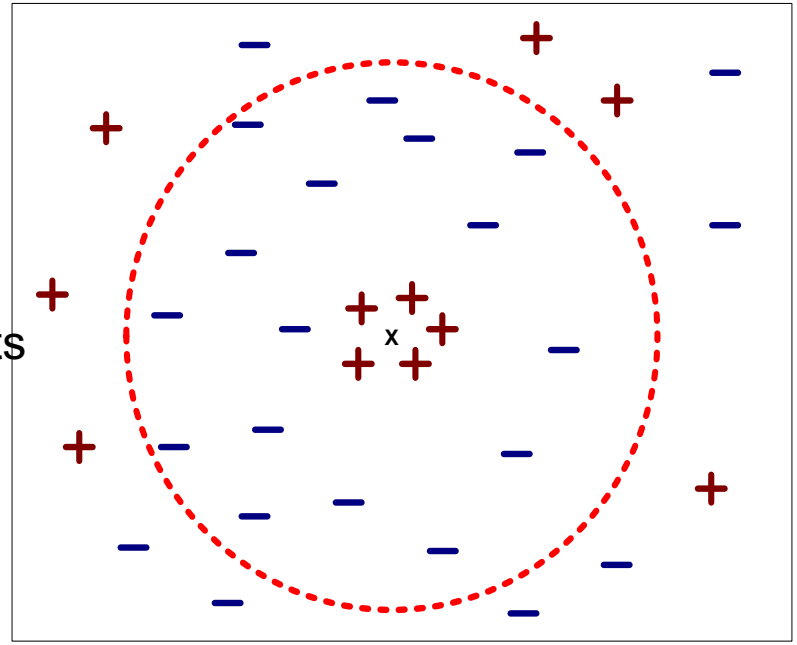
Nearest-Neighbor Classifiers: Issues

- The value of *k*, the number of nearest neighbors to retrieve
- Choice of Distance Metric to compute distance between records
- Computational complexity
  - Size of training set
  - Dimension of data

Value of K

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes

Rule of thumb:
K = sqrt(N)
N: number of training points

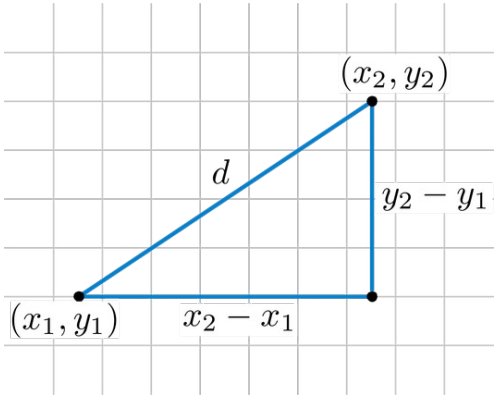


Distance Metric

–Euclidean Distance

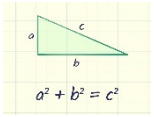
–*The Euclidean distance function measures the 'straight-line' distance.*

–*The distance between a point* (x_1, y_1) *and a point* (x_2, y_2) *can be calculated as:*



The data points (x_1, y_1) ***and*** (x_2, y_2) ***are in 2-dimensional space, so it will be:***

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



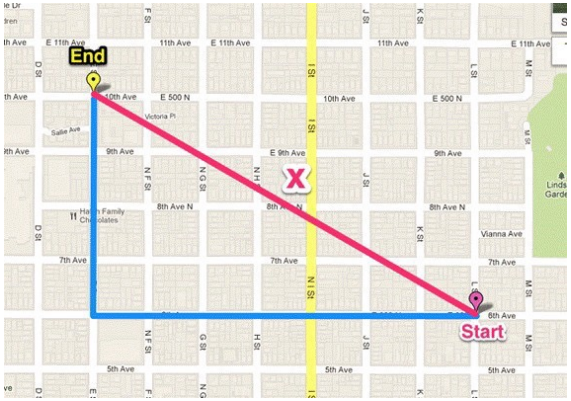
In 3-dimensional space, the Euclidean distance between the data points (x_1, y_1, z_1) *and* (x_2, y_2, z_2) *is:*

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

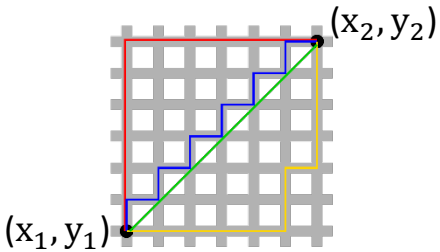
Distance Metric

–Manhattan Distance

–The Manhattan distance function computes the distance that would be traveled to get from one data point to the other if a grid-like path is followed. The Manhattan distance between two items is the sum of the differences of their corresponding components.



–The distance between a point (x_1, y_1) *and a point* (x_2, y_2) *can be calculated as:*

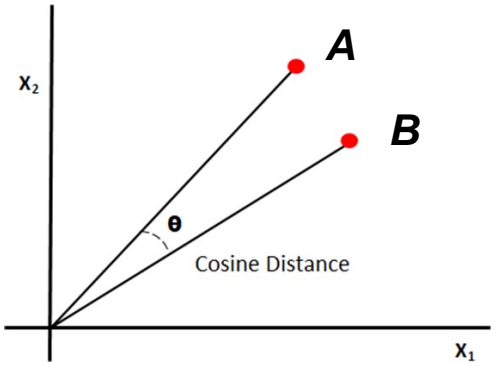


$$d = |x_2 - x_1| + |y_2 - y_1|$$

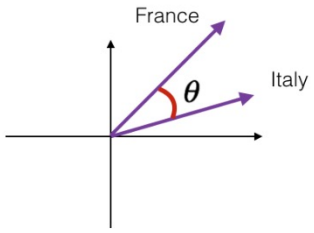
Distance Metric

–Cosine Distance

–It measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is useful because even if the two similar things are far apart by the Euclidean distance (due to its size), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.



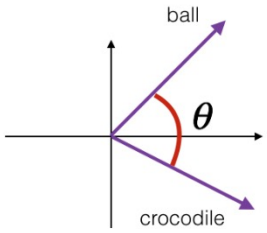
$$\frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}=\frac{\sum_{i=1}^{n}A_iB_i}{\sqrt{\sum_{i=1}^{n}A_i^2}\sqrt{\sum_{i=1}^{n}B_i^2}},$$



France and Italy are quite similar

θ is close to 0°

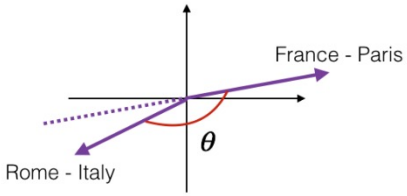
$\cos(\theta)\approx 1$



ball and crocodile are not similar

θ is close to 90°

$\cos(\theta)\approx 0$



the two vectors are similar but opposite
the first one encodes (city - country)
while the second one encodes (country - city)

θ is close to 180°

$\cos(\theta)\approx -1$

Distance Metric

–Other Distance/Similarity Measure

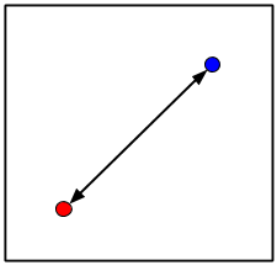
There are several types of distance measure algorithms in the world..

The most common and simplest measures are the Euclidean distance and Cosine distance

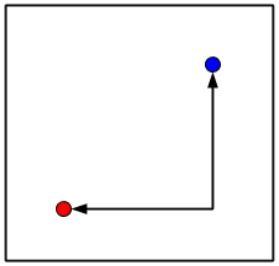
$$\sum_{i=1}^{n} |x_i - y_i|$$

Euclidean

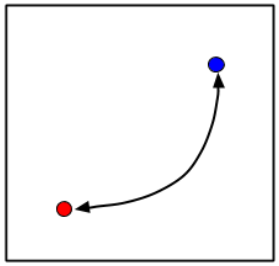
$$\left(\sum_{i=1}^{n} |x_i - y_i|^2\right)^{1/2}$$



Manhattan



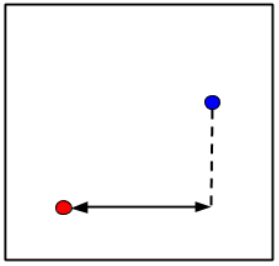
Minkowski



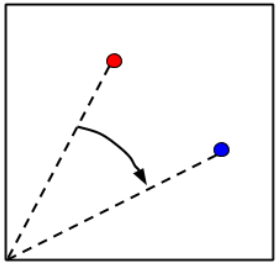
$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

Chebychev

$$\lim_{p\to\infty} \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p} = \max |x_i - y_i|$$



Cosine Similarity



Hamming

XOR

The number of bit positions in which the two bits are different.

Nearest Neighbour : Computational Complexity

- Expensive
  - To determine the nearest neighbour of a query point q, must compute the distance to all N training examples
    + Pre-sort training examples into fast data structures (kd-trees)
    + Compute only an approximate distance (LSH)
    + Remove redundant data (condensing)
- Storage Requirements
  - Must store all training data **P**
    + Remove redundant data (condensing)
    - Pre-sorting often increases the storage requirements
- High Dimensional Data
  - "Curse of Dimensionality"
    - Required amount of training data increases exponentially with dimension
    - Computational cost also increases dramatically
    - Partitioning techniques degrade to linear search in high dimension

Evaluation Metrics

Evaluation Metrics

Confusion Matrix Based Evaluation Metrics

	Predict positive	Predict negative
Actual positive	TP	FN
Actual negative	FP	TN

- **True positive (TP)** is the number of positive individuals correctly predicted as positive.
- **False positive (FP)** is the number of negative individuals incorrectly predicted as positive.
- **False negative (FN)** is the number of positive individuals incorrectly predicted as negative.
- **True negative (TN)** is the number of negative individuals correctly predicted as negative.

Evaluation Metrics

	Predict positive	Predict negative
Actual positive	TP	FN
Actual negative	FP	TN

The **accuracy** of measurement is defined as the closeness of agreement between a quantity value obtained by measurement and the true value of the measurand:

$$\textbf{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

Recall measures the ratio of actual positives that are correctly identified:

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

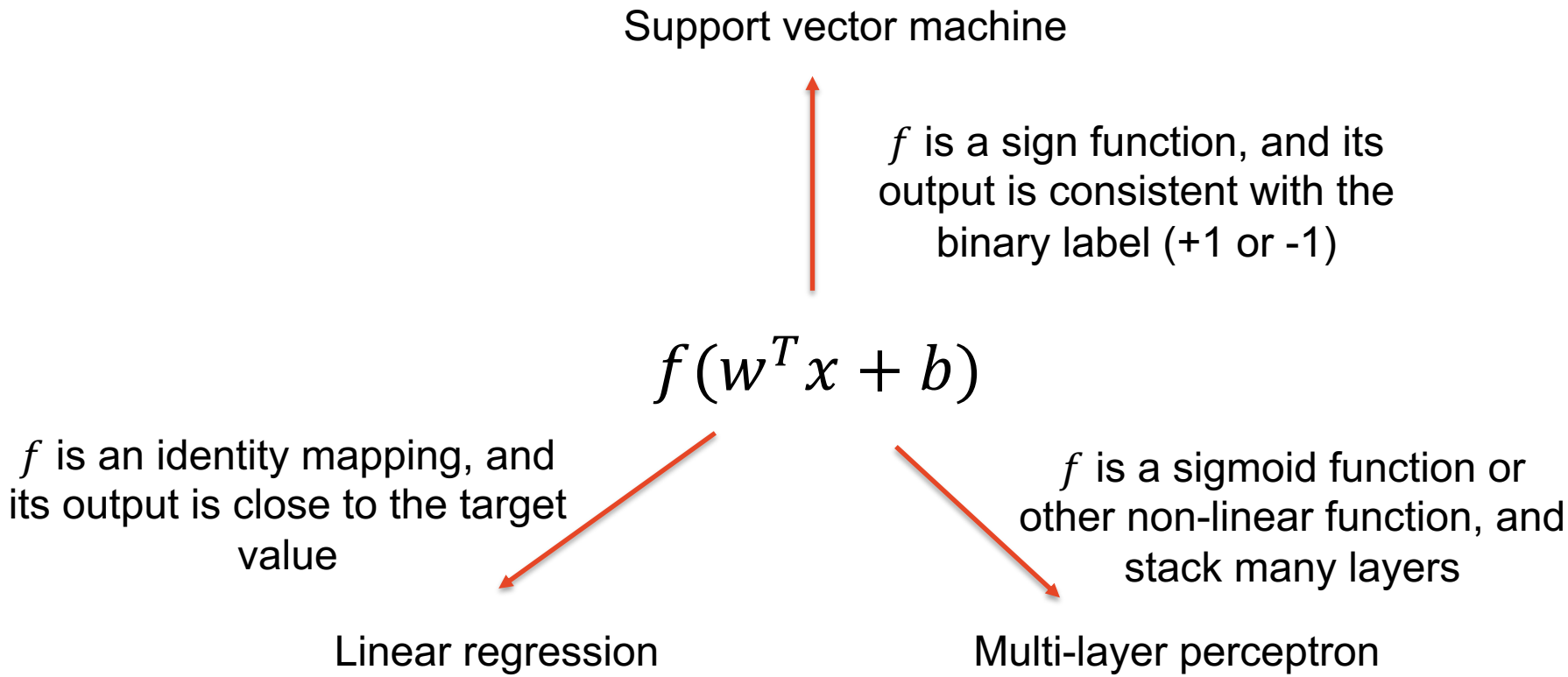
Precision measures the ratio of true positives to predicted positives:

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

F-measure is the harmonic mean of recall and precision:

$$\textbf{F-measure} = \frac{2 \times \textbf{Precision} \times \textbf{Recall}}{\textbf{Precision} + \textbf{Recall}}$$

Summary



The key in a KNN classifier is to calculate distance between examples.

Thank you!