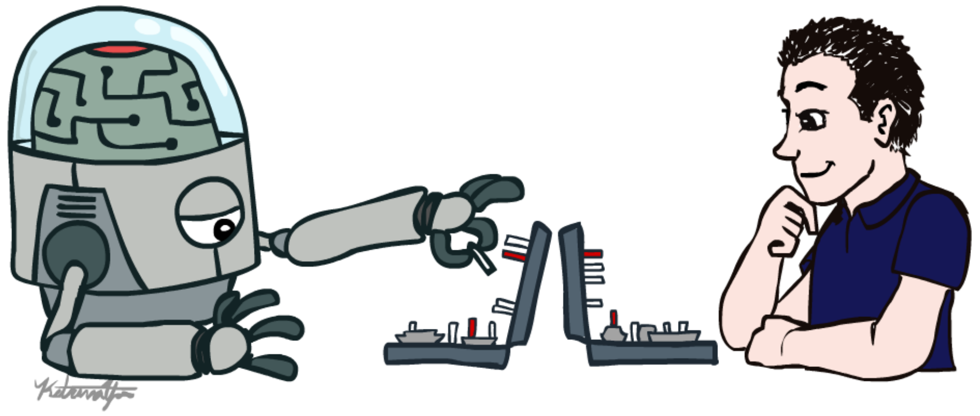
Lecture 12

Ashis Kumar Chanda
chanda@rowan.edu

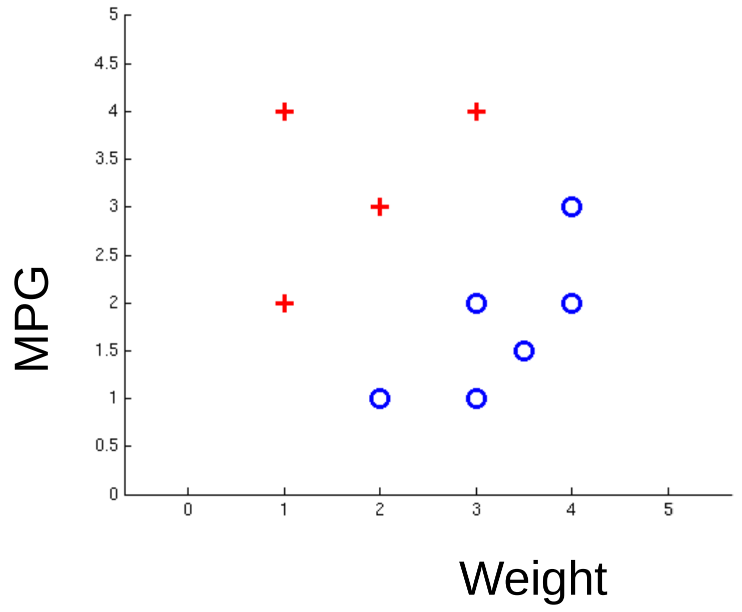


KNN + Logistic regression

K-NN

- K-NN: K-Nearest Neighbor method
- Learning from neighbors.

- Ex:
  - blue dot presents a car of the USA.
  - red dot presents a car outside of the USA.



KNN

- It can be used to solve both **classification** and regression problems.

- In k-NN **classification**, the output is a class membership.

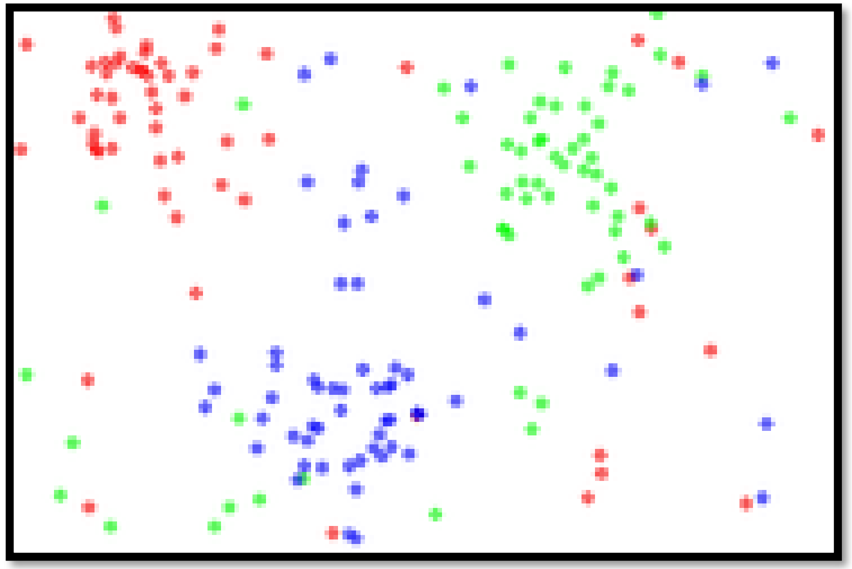
An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors.
Ex: a customer is taking loan, or not.

- In k-NN **regression**, the output is the property value for the object.

This value is the average of the values of k nearest neighbors.
Ex: The loan amount of a customer.

KNN

- **Method**:
- Compute the Euclidean or Manhattan distance from the **query** example to the labeled examples.
- Order the labeled examples by increasing distance.
- Find a heuristically optimal number k of nearest neighbors
- Take vote of k examples to get the class of query example.



KNN

- KNN is an offline method.
- If data has high dimensions, then it has worst result. [curse of dimensionality]
    - Solution: dimension reduction (i.e., PCA (principal component analysis))

- Need to normalize/scale feature values.
- Deciding the size of K is challenging.
    - K >> could be underfitting.
    - K << could be overfitting.

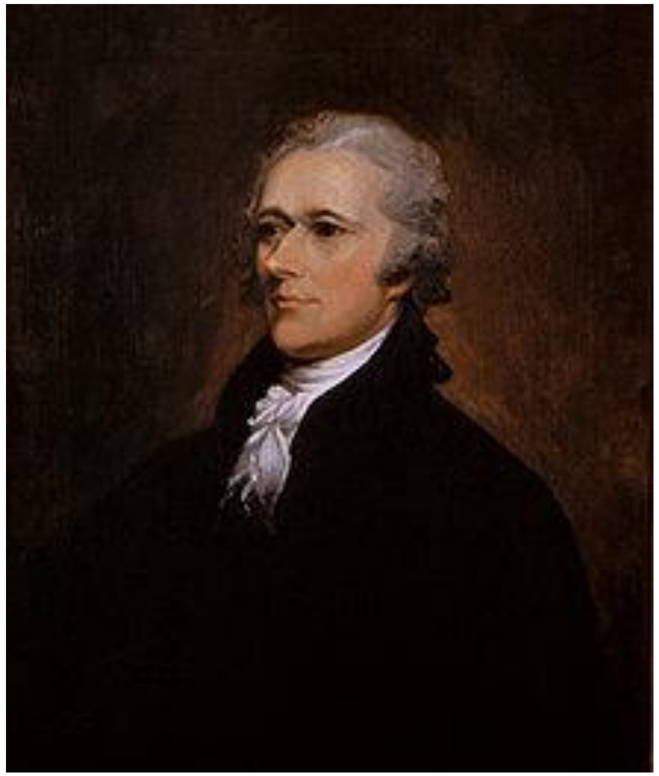
- Alternative solution: giving more weights to close points.

[A common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor]

Logistic regression

Classification Reminder

- Positive/negative sentiment
- Spam/not spam
- Authorship attribution (Hamilton or Madison?)


Alexander Hamilton

Text Classification: definition

- *Input*:
  - a document x
  - a fixed set of classes $C = \{c_1, c_2, \dots, c_J\}$

- *Output*: a predicted class $\hat{y} \in C$

Binary Classification in Logistic Regression

- Given a series of input/output pairs:
  - $(x^{(i)}, y^{(i)})$
- For each observation $x^{(i)}$
  - We represent $x^{(i)}$ by a **feature vector** $[x_1, x_2, \ldots, x_n]$
  - We compute an output: a predicted class $\hat{y}^{(i)} \in \{0,1\}$

Features in logistic regression

- For feature x_i, weight w_i tells is how important is x_i
  - x_i ="review contains '`awesome`'": w_i = +10
  - x_j ="review contains '`abysmal`'": w_j = −10
  - x_k ="review contains '`mediocre`'": w_k = −2

Logistic Regression for one observation x

- Input observation: vector $x = [x_1, x_2, \ldots, x_n]$
- Weights: one per feature: $W = [w_1, w_2, \ldots, w_n]$
  - Sometimes we call the weights $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$
- Output: a predicted class $\hat{y} \in \{0,1\}$

How to do classification (linear classification)

- For each feature x_i, weight w_i tells us importance of x_i
  - (Plus we'll have a bias b)
- We'll sum up all the weighted features and the bias

$$Z = \sum_{i=1}^{k} Wi\, Xi + b$$

- If this sum is high, we say y=1; if low, then y=0

$$z = w \cdot x + b$$

$$Z = \sum_{i=0}^{k} Wi\, Xi$$

But we want a probabilistic classifier

- We need to formalize “sum is high”.
- We’d like a principled classifier that gives us a probability, just like Naive Bayes did
- We want a model that can tell us:

  $p(y=1|x; \theta)$

  $p(y=0|x; \theta)$

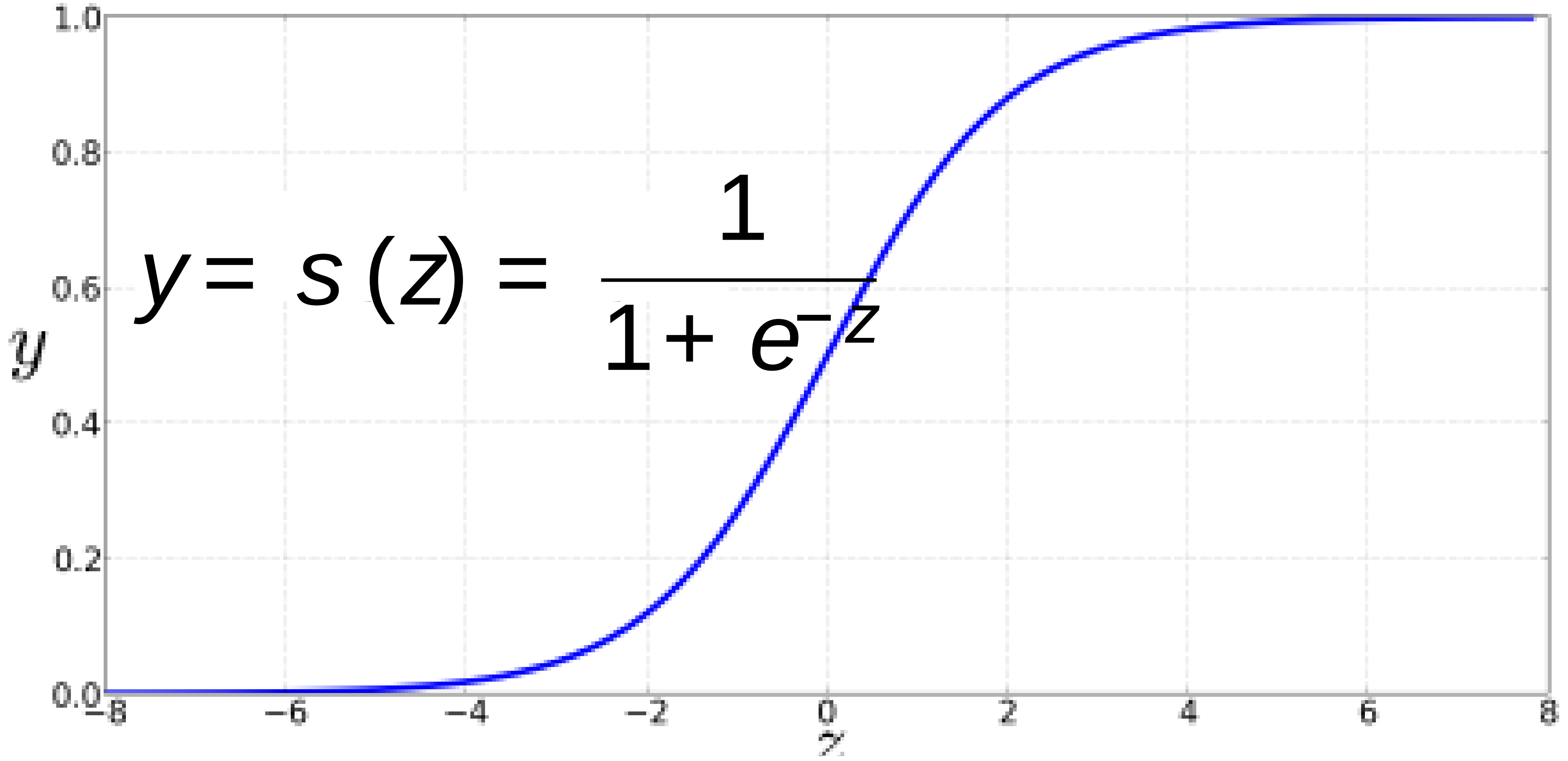
The problem: z isn't a probability, it's just a number!

$$z = w \cdot x + b$$

- Solution: use a function of z that goes from 0 to 1

$$y = s(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + \exp(-z)}$$

The very useful sigmoid or logistic function



$$y = s(z) = \frac{1}{1 + e^{-z}}$$

Idea of logistic regression

- We’ll compute $w \cdot x + b$
- And then we’ll pass it through the sigmoid function:
- $\sigma(w \cdot x + b)$
- And we'll just treat it as a probability

Making probabilities with sigmoids

$$\begin{aligned} P(y=1) &= \sigma(w \cdot x + b) \\ &= \frac{1}{1+\exp(-(w \cdot x + b))} \end{aligned}$$

$$\begin{aligned} P(y=0) &= 1 - \sigma(w \cdot x + b) \\ &= 1 - \frac{1}{1+\exp(-(w \cdot x + b))} \\ &= \frac{\exp(-(w \cdot x + b))}{1+\exp(-(w \cdot x + b))} \end{aligned}$$

By the way:

$$\begin{aligned} P(y=0) &= 1-\sigma(w\cdot x+b) \\ &= 1-\frac{1}{1+\exp\left(-(w\cdot x+b)\right)} \\ &= \frac{\exp\left(-(w\cdot x+b)\right)}{1+\exp\left(-(w\cdot x+b)\right)} \end{aligned}$$

= $\sigma(-(w\cdot x+b))$

Because

$$1-\sigma(x)=\sigma(-x)$$

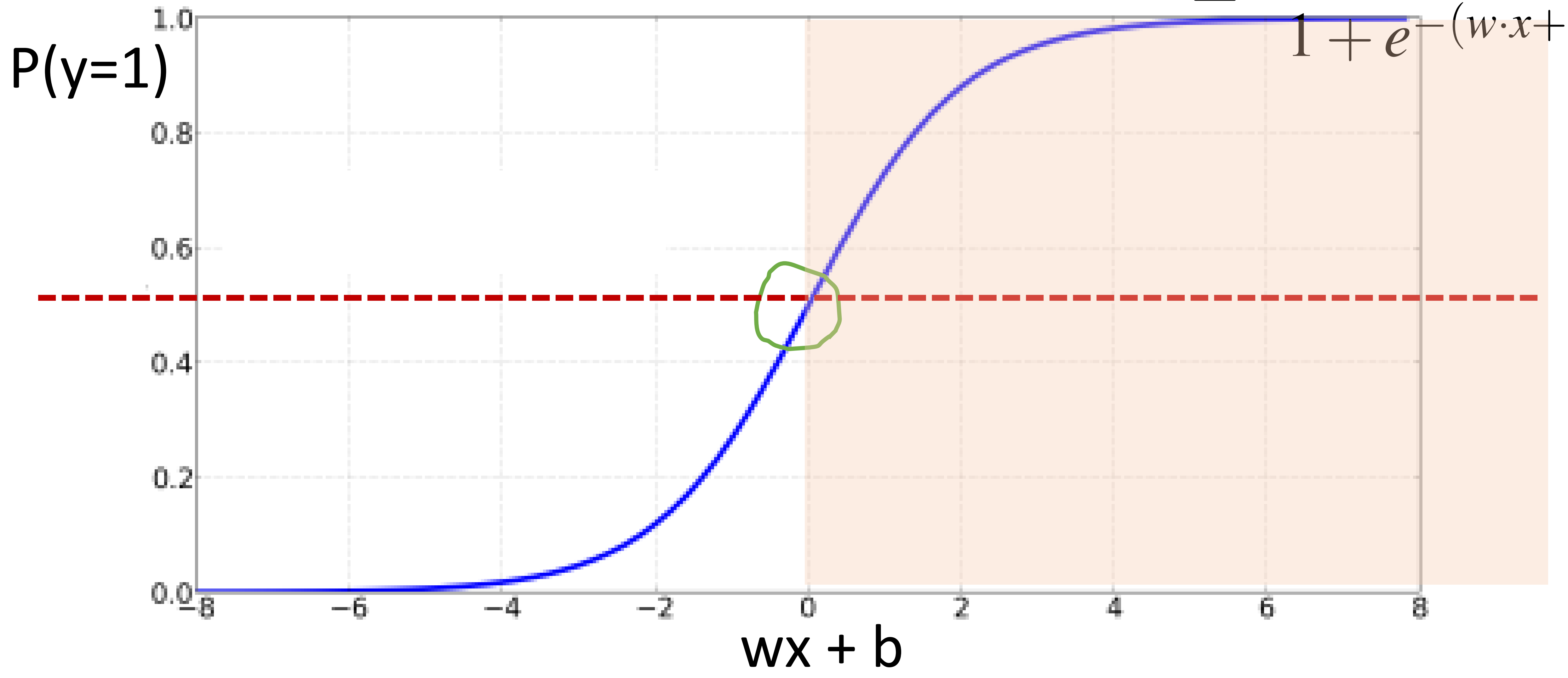
Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

0.5 here is called the **decision boundary**

The probabilistic classifier

$$P(y=1) = \sigma(w \cdot x + b)$$
$$= \frac{1}{1+e^{-(w \cdot x+b)}}$$



Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} &\text{if } \mathrm{w}\cdot\mathrm{x}+\mathrm{b} > 0 \\ &\text{if } \mathrm{w}\cdot\mathrm{x}+\mathrm{b} \leq 0 \end{aligned}$$

Wait, where did the W's come from?

- Supervised classification:
  - We know the correct label y (either 0 or 1) for each *x*.
  - But what the system produces is an estimate, $\hat{y}$
- We want to set *w* and *b* to minimize the **distance** between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$.
  - We need a distance estimator: a **loss function** or a **cost function**
  - We need an optimization algorithm to update *w* and *b* to minimize the loss.

Learning components

- A loss function:
  - **cross-entropy loss**
- An optimization algorithm:
  - **stochastic gradient descent**

The distance between $\hat{y}$ and y

- We want to know how far is the classifier output:

  $\hat{y}$ = σ(w·x+b)

- from the true output:

  y [= either 0 or 1]

- We'll call this difference:

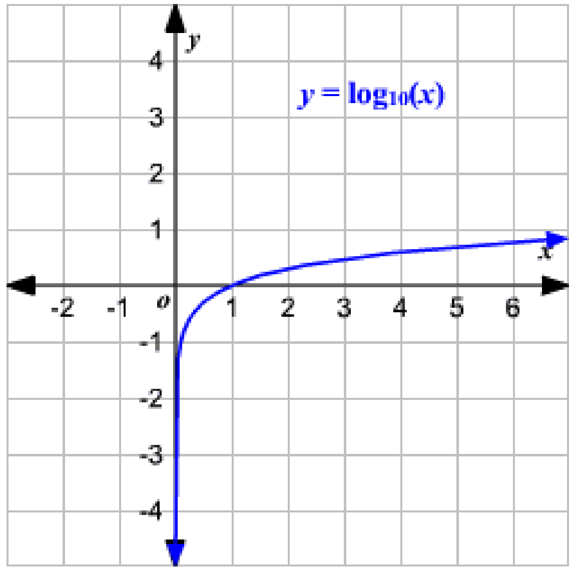
  $L(\hat{y}, y)$ = how much $\hat{y}$ differs from the true y

Intuition of negative log likelihood loss = cross-entropy loss

- A case of conditional maximum likelihood estimation
- We choose the parameters *w*,*b* that maximize
- the log probability
- of the true *y* labels in the training data
- given the observations *x*
-

Intuition of negative log likelihood loss = cross-entropy loss

- **Goal**: maximize probability of the correct label
- Since there are only 2 discrete outcomes (0 or 1) we can express the negative log likelihood loss as
- $-\log(\hat{y})$ if $y = 1$
- $-\log(1 - \hat{y})$ if $y = 0$



When x is 1, y is 0. It means no loss.
When x is 0.5, y is negative.
When x is 0, y is infinite. (i.e., high loss)
[Taking negative log loss, to take positive value]

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

Maximize:
$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

- Now take the log of both sides (mathematically handy)

Maximize:
$$\begin{aligned} \log p(y|x) &= \log \left[\hat{y}^y (1-\hat{y})^{1-y}\right] \\ &= y \log \hat{y} + (1-y) \log(1-\hat{y}) \end{aligned}$$

- Whatever values maximize log p(y|x) will also maximize p(y|x)

-

Deriving cross-entropy loss for a single observation x

- **Cross-entropy loss**

$$L_{\mathrm{CE}}(\hat{y}, y) = -\log p(y|x) \;=\; -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

Or, plugging in definition of $\hat{y}$:

Minimize:

$$L_{\mathrm{CE}}(\hat{y}, y) \;=\; -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

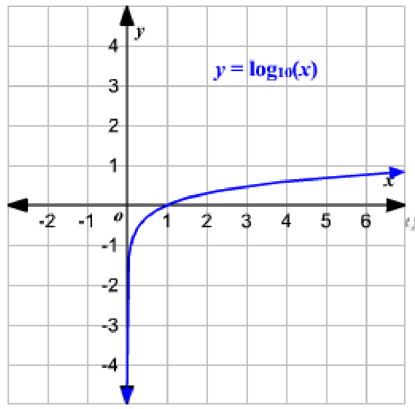
Our goal: minimize the loss

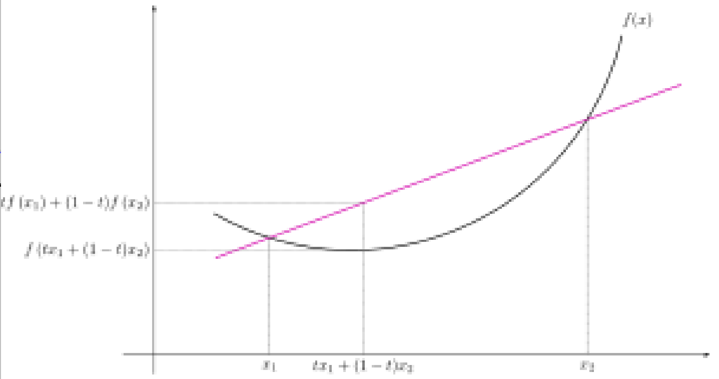
- Let's make explicit that the loss function is parameterized by weights θ=(w,b)
- And we'll represent $\hat{y}$ as $f\,(x;\,\theta\,)$ to make the dependence on θ more obvious.
- We want the weights that minimize the loss, averaged over all examples:

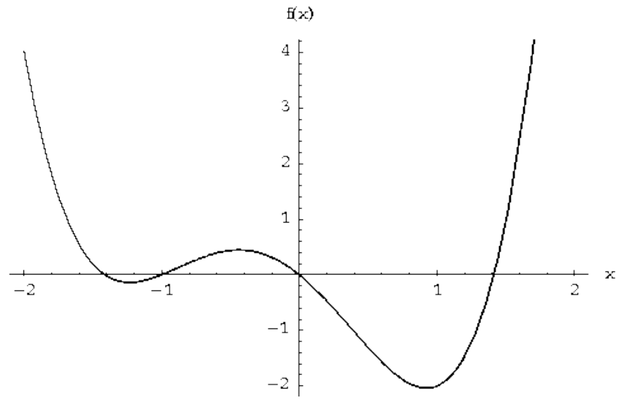
$$\hat{\theta} \;=\; \underset{\theta}{\operatorname{argmin}} \frac{1}{m}\sum_{i=1}^{m} L_{\mathrm{CE}}(f(x^{(i)};\theta),y^{(i)})$$

Our goal: minimize the loss

- For logistic regression, loss function is **convex**
- A convex function has just one minimum
- **Gradient descent** starting from any point is guaranteed to find the minimum.
- A real-valued function is called **convex** if the line segment between any two points on the graph of the function does not lie below the graph between the two points.







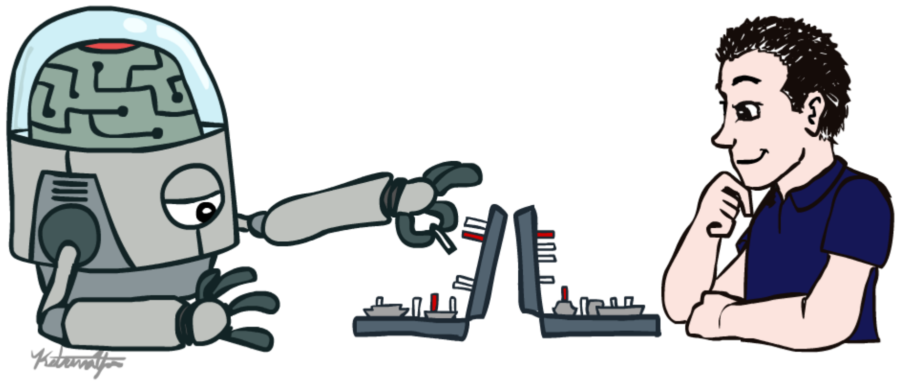
How much do we move in that direction ?

- The value of the gradient (slope in our example) $\frac{d}{dw}L(f(x;w),y)$ weighted by a **learning rate** η
- Higher learning rate means move *w* faster

$$w^{t+1} = w^{t} - h\frac{d}{dw}L(f(x;w),y)$$

Hyperparameters

- The learning rate η is a **hyperparameter**
  - too high: the learner will take big steps and overshoot
  - too low: the learner will take too long
- Hyperparameters:
- Briefly, a special kind of parameter for an ML model
- Instead of being learned by algorithm from supervision (like regular parameters), they are chosen by algorithm designer.



Thanks!