Machine Lerning

Lecture 4

Dr. Arif Mahmood
Professor ITU

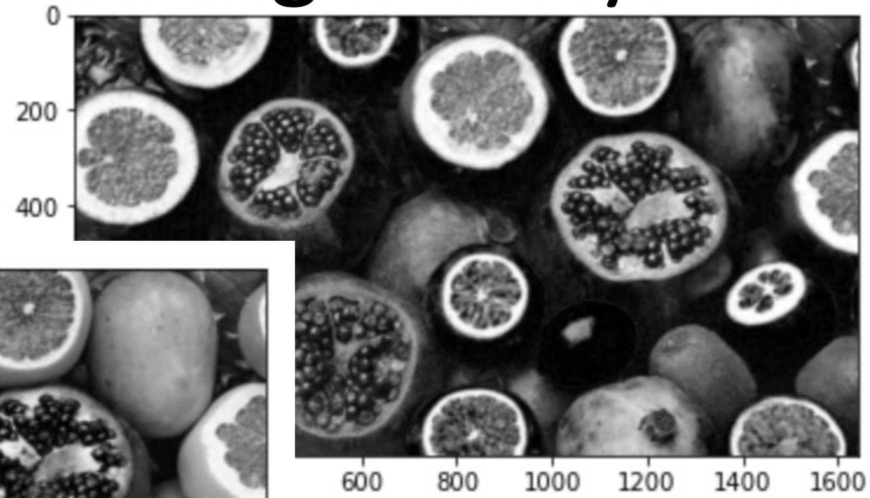
E-mail: arif.mahmood@itu.edu.pk

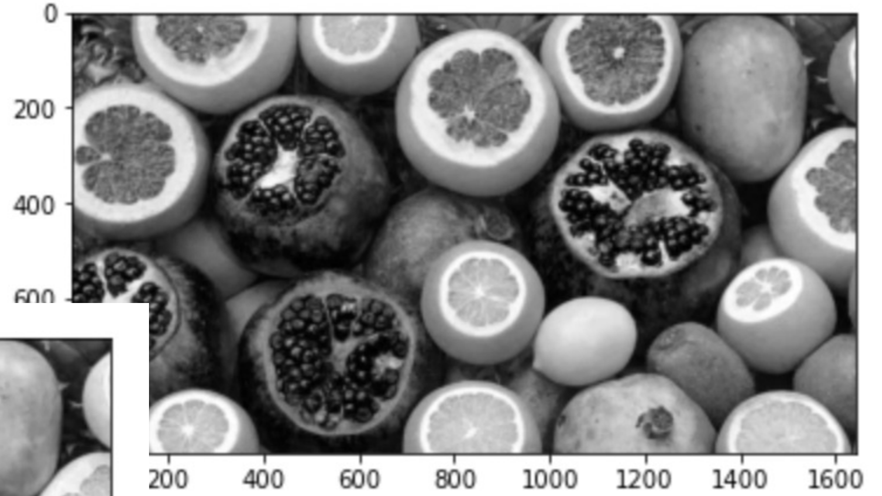
Website Google Scholar

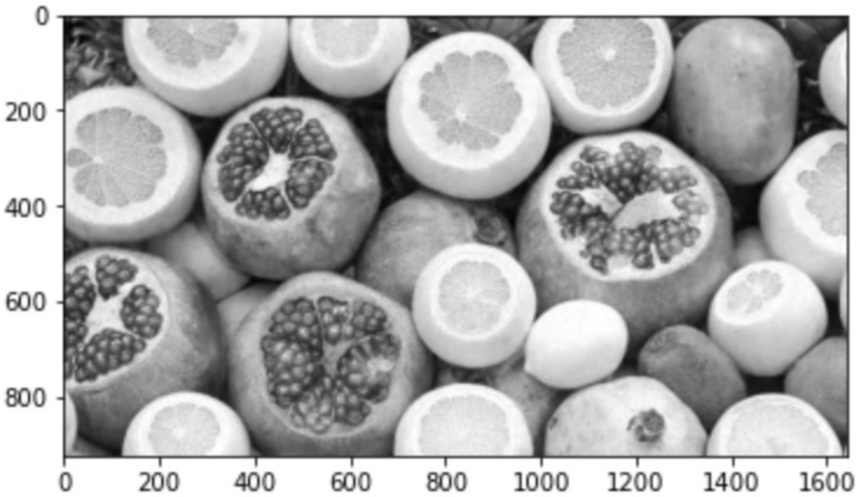
Office: 6-th Floor CS Department, ITU

Reading and Displaying Images using NumPy

- im = imread('Citrus-Fruits.jpeg’)
- imshow(im)
- im.shape
- r = im[:, :, 0]
- g = im[:, :, 1]
- b = im[:, :, 2]
- imshow(r, cmap=cm.gray)







Reading data from CSV Files

```python
import pandas as pd
from numpy import *
from numpy.linalg import norm

dat = pd.read_csv("galaxy.data")
x1 = dat.loc[:,"east.west"].values
x2 = dat.loc[:, "north.south"].values
y = dat.loc[:, "velocity"].values
```

galaxy.data

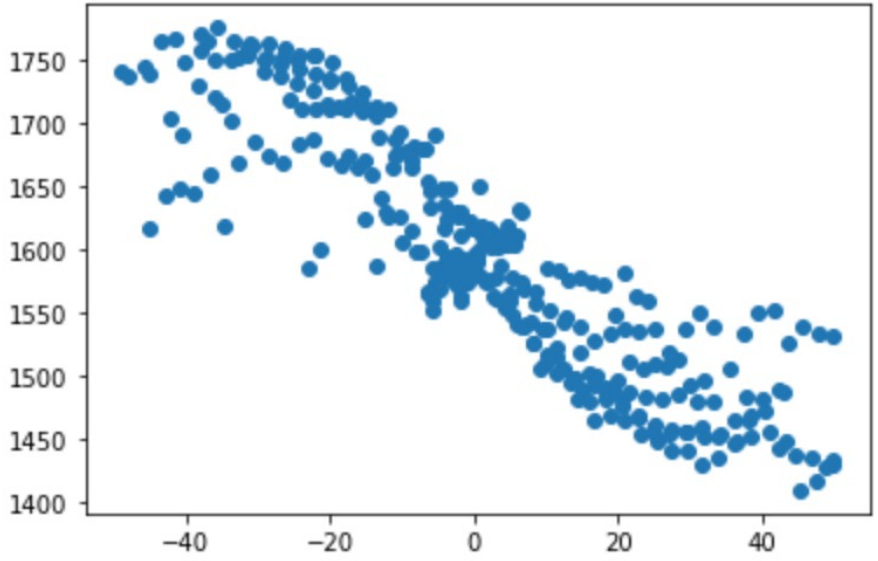
```
row.names,east.west,north.south,angle,radial.position,velocity
3,8.46278858184814,-38.1731719970703,102.5,39.0999980933372,1769
4,7.96497774124146,-35.9276924133301,102.5,36.7999993555072,1749
5,7.46716690063477,-33.6822128295898,102.5,34.5000006176771,1749
6,6.96935558319092,-31.436731338501,102.5,32.1999999145038,1758
7,6.47154426574707,-29.1912498474121,102.5,29.8999992113304,1750
8,5.97373342514038,-26.9457702636719,102.5,27.6000004735004,1745
9,5.47592210769653,-24.700288772583,102.5,25.2999997703271,1750
10,4.97811126708984,-22.4548072814941,102.5,22.9999991703602,1753
11,4.480299949646,-20.2093276977539,102.5,20.7000003293237,1734
12,3.98248887062073,-17.963846206665,102.5,18.3999996777536,1710
13,3.46303391456604,-15.6207361221313,102.5,16.0000000278342,1711
14,2.96522259712219,-13.3752555847168,102.5,13.7000002557293,1709
15,2.46741151809692,-11.1297740936279,102.5,11.3999996041592,1674
16,1.96960055828094,-8.8842945098877,102.5,9.1000008405275,1665
17,1.47178936004639,-6.63881301879883,102.5,6.80000016315577,1680
18,0.973978281021118,-4.39333200454712,102.5,4.49999997711987,1648
19,0.47616717219527,-2.14785122871399,102.5,2.20000001740068,1626
20,-0.0216439608484507,0.0976295992732048,102.5,-0.0999999984772791,1581
21,-0.519455075263977,2.34311056137085,102.5,-2.40000014125523,1602
22,-1.01726615428925,4.5885910987854,102.5,-4.69999986175692,1558
23,-1.5150773525238,6.83407211303711,102.5,-7.00000007359443,1538
24,-2.01288843154907,9.07955265045166,102.5,-9.29999979409612,1506
25,-2.51069951057434,11.3250341415405,102.5,-11.6000004456662,1521
26,-3.00851058959961,13.5705137252808,102.5,-13.8999992350995,1498
27,-3.5279655456543,15.9136238098145,102.5,-16.2999988850189,1501
28,-4.02577686309814,18.1591053009033,102.5,-18.5999995881922,1491
29,-4.52358770370483,20.4045867919922,102.5,-20.9000001881591,1481
30,-5.02139902114868,22.6500682830811,102.5,-23.2000008913325,1468
31,-5.51921033859253,24.8955478668213,102.5,-25.499999732369,1455
```

Plotting Scatter Plots:

```python
from matplotlib.pyplot import *

scatter(x2, y)
```

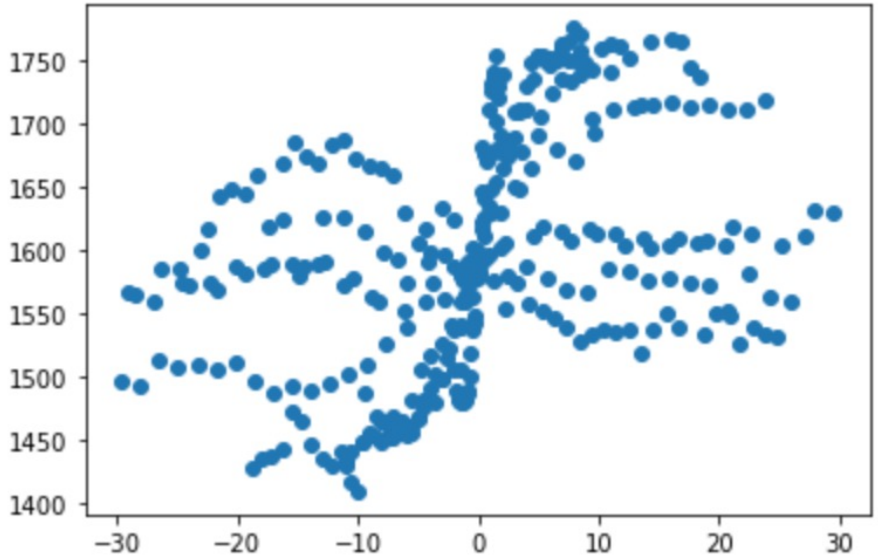
<matplotlib.collections.PathCollection at 0x11850(




```python
from matplotlib.pyplot import *

scatter(x1, y)
```

<matplotlib.collections.PathCollection at 0x1186001f0>



Loss Function and its Derivatives

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \quad X = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

```
def f(x, y, theta):
    # theta = np.array([t1, t2, t3])

    x = vstack( (ones((1, x.shape[1])), x))
    return sum( (y - dot(theta.T,x)) ** 2)

def df(x, y, theta):
    x = vstack( (ones((1, x.shape[1])), x))
    return -2*sum((y-dot(theta.T, x))*x, 1)
```

$$\mathrm{Y} = \theta^T X$$

$$\text{df}=\begin{bmatrix} \frac{\partial f}{\partial \theta_0} \\ \frac{\partial f}{\partial \theta_1} \\ \frac{\partial f}{\partial \theta_2} \end{bmatrix}$$

$$f = \sum (\mathrm{Y} - \theta^T X)^2$$

Comparison of Derivatives:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

$$\frac{\partial f(\theta_0, \theta_1, \theta_2)}{\partial \theta_0} = \frac{f(\theta_0 + h, \theta_1, \theta_2) - f(\theta_0 - h, \theta_1, \theta_2)}{2h}$$

```
x = vstack((x1, x2))
theta = array([-3, 2, 1])

h = 0.000001
print ((f(x, y, theta+array([h, 0, 0])) - f(x, y, theta-array([h, 0, 0])))/(2*h))
print (df(x, y, theta))
```

Gradient Descent Function

```
def grad_descent(f, df, x, y, init_t, alpha):
    EPS = 1e-5    #EPS = 10**(-5)
    prev_t = init_t-10*EPS
    t = init_t.copy()
    max_iter = 30000
    iter  = 0
    while norm(t - prev_t) >  EPS and iter < max_iter:
        prev_t = t.copy()
        t -= alpha*df(x, y, t)
        if iter % 500 == 0:
            print ("Iter", iter)
            print ("theta = (%.2f, %.2f, %.2f), f(theta) = %.2f" % (t[0], t[1], t[2], f(x, y, t)) )
            print ("Gradient: ", df(x, y, t), "\n")
        iter += 1
    return t
```

Solving Linear Regression Using GD:

```
x = vstack((x1, x2))
theta0 = array([0., 0., 0.])
theta = grad_descent(f, df, x, y, theta0, 0.0000010)
```

```
theta = (1599.71, 2.32, -3.54), f(theta) = 324483.39
Gradient:  [-48.18042752  -0.08059095   0.13488855]

Iter 16000
theta = (1599.73, 2.32, -3.54), f(theta) = 324482.54
Gradient:  [-34.932044    -0.0584305    0.09779765]

Iter 16500
theta = (1599.74, 2.32, -3.54), f(theta) = 324482.09
Gradient:  [-25.32662661  -0.04236361   0.0709058 ]

Iter 17000
theta = (1599.75, 2.32, -3.54), f(theta) = 324481.85
Gradient:  [-18.36245299  -0.0307147    0.05140852]

Iter 17500
theta = (1599.76, 2.32, -3.54), f(theta) = 324481.72
Gradient:  [-13.31324873  -0.02226895   0.0372725 ]
```

Solving Linear Regression Using Pseudo-Inverse:

$$\mathrm{Y} = \theta^T X$$

$$\mathrm{Y} X^T = \theta^T X X^T$$

$$\mathrm{Y}\left(X^T (X X^T)^{-1}\right) = \theta^T \qquad \theta = \left((X X^T)^{-1} X\right) \mathrm{Y}^T$$

```
x = vstack((x1, x2))
x = vstack( (ones((1, x.shape[1])), x))
dot(dot(linalg.inv(dot(x, x.T)),x), y)
```

```
array([1599.7805884 ,    2.32128786,   -3.53935822])
```

Classification with two classes

- If there are only two classes,

  orange => 1
  blue => 0
  to turn the classification problem into a regression problem

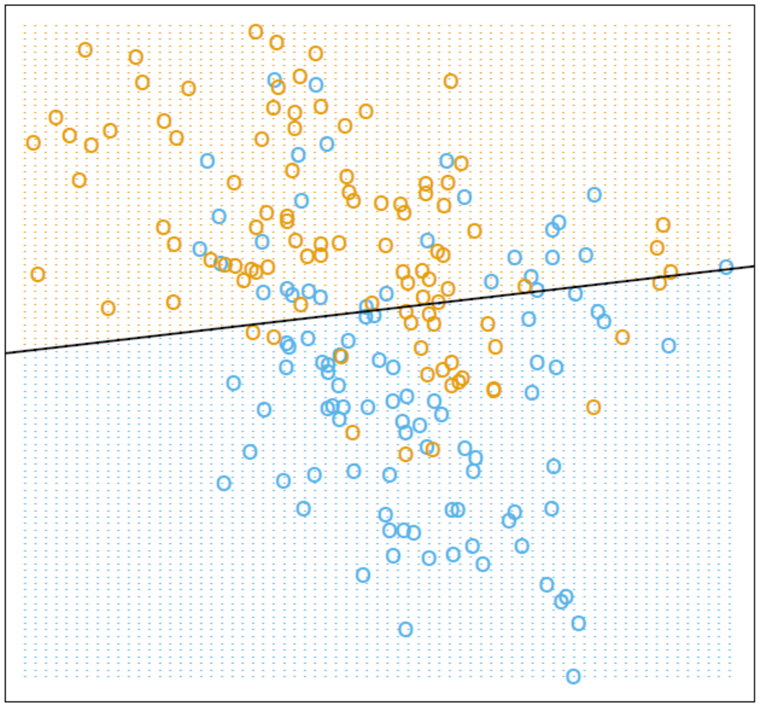
- Find the best

$$h_\theta(x) = \theta^T x$$

- Predict:

$$\begin{cases} 1, h_\theta(x) > 0.5 \\ 0, otherwise \end{cases}$$

Linear Regression of 0/1 Response



$\theta_1 x_1 + \theta_2 x_2$ (can add in θ_0)

What is the equation of the decision boundary?

But what about the loss function?

(Loss function = cost function)

What is the equation of the decision boundary?

Attempt #1:

- Quadratic loss, as in Linear Regression.

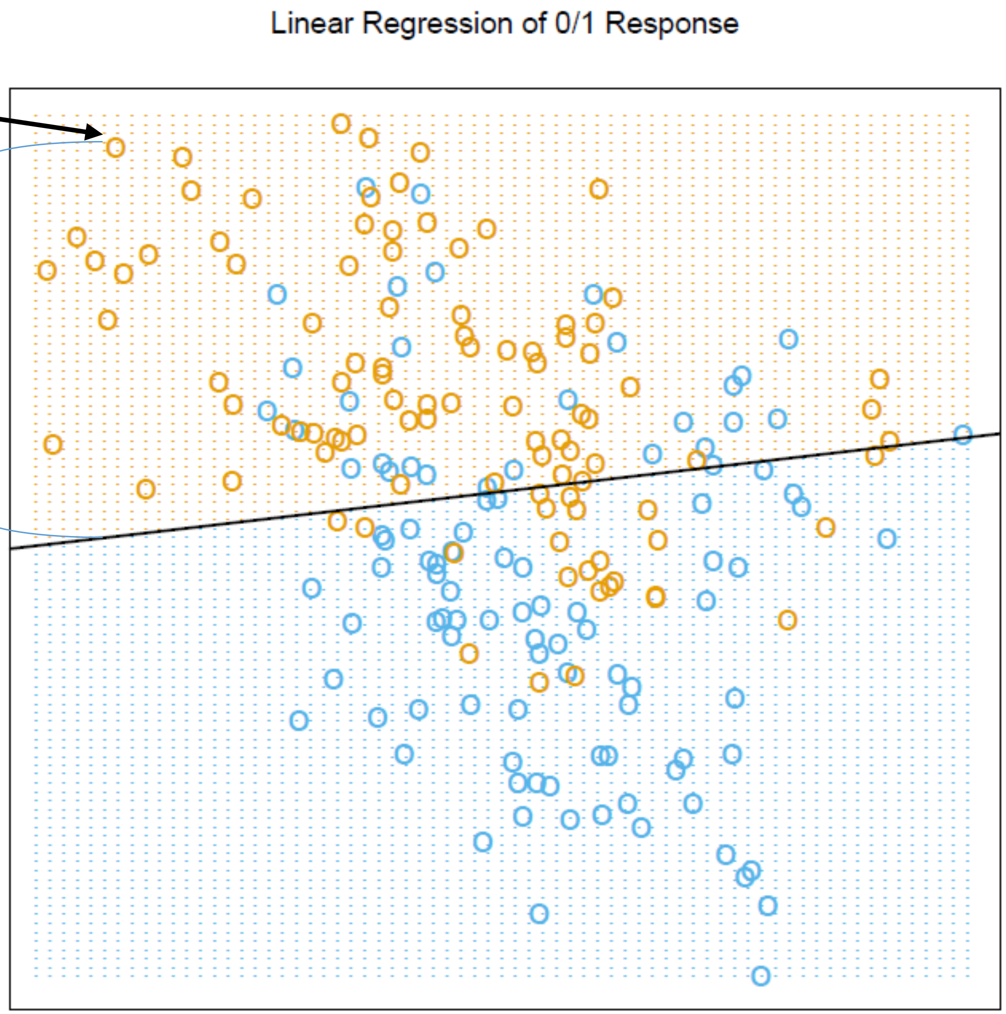
$$\sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

- What is the problem with this loss function?

Attempt #1:

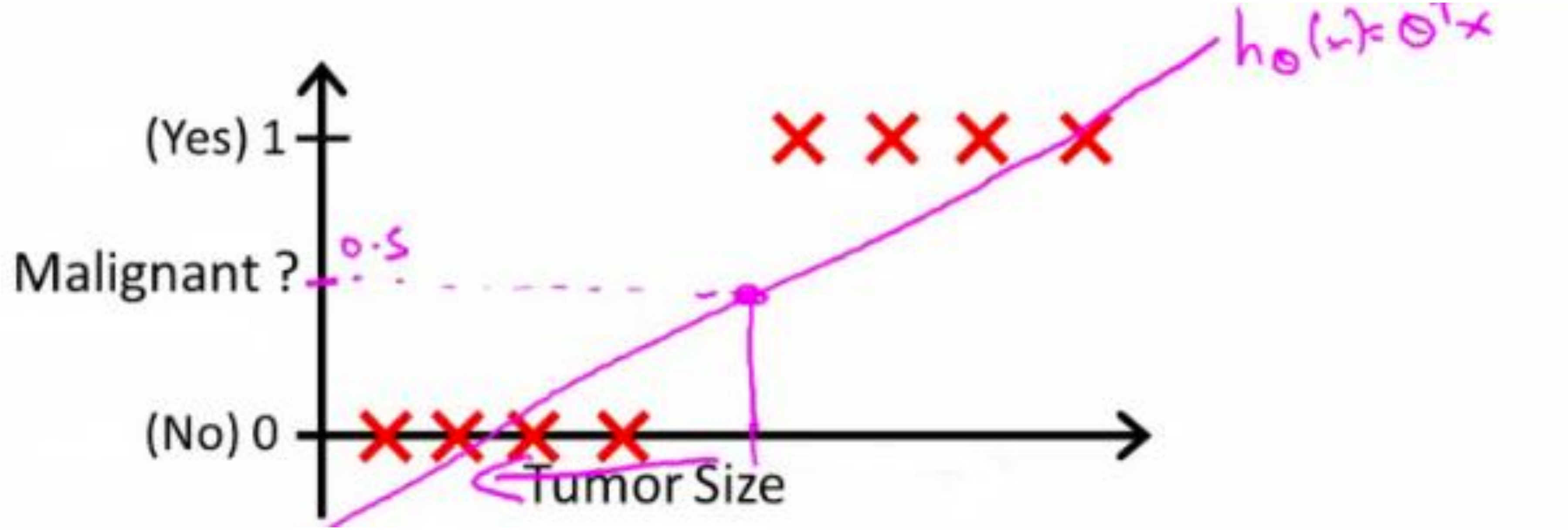
How much does this training data contribute to the loss?

A lot!



Quadratic loss penalizes data well within the decision boundary.

Example in 1D



Even with perfect classification,
Loss is still nonzero (and can be high!)

Attempt #2:

- Classification error or 0-1 loss.

$$\sum_{i=1}^{m} \mathrm{I}\left[y^{(i)}, t^{(i)}\right]$$

$$t^{(i)} = \begin{cases} 1, h_\theta\left(x^{(i)}\right) > 0.5 \\ 0, otherwise \end{cases}$$

- Where I is the indicator function:

$$\mathrm{I}[y, t] = \begin{cases} 1, y \neq t \\ 0, otherwise \end{cases}$$

- What is the problem with this loss function?

Not continuous.
Hard to optimize.
Cannot use gradient descent (Why?

Attempt #3:

$y = mx + c$

- Problem with linear regression (quadratic loss): **Predictions are allowed to take arbitrary real values!**
- Problem with linear regression (0-1 loss): **Hard to optimize!**

- Apply a nonlinearity or activation function: **sigmoid function:**

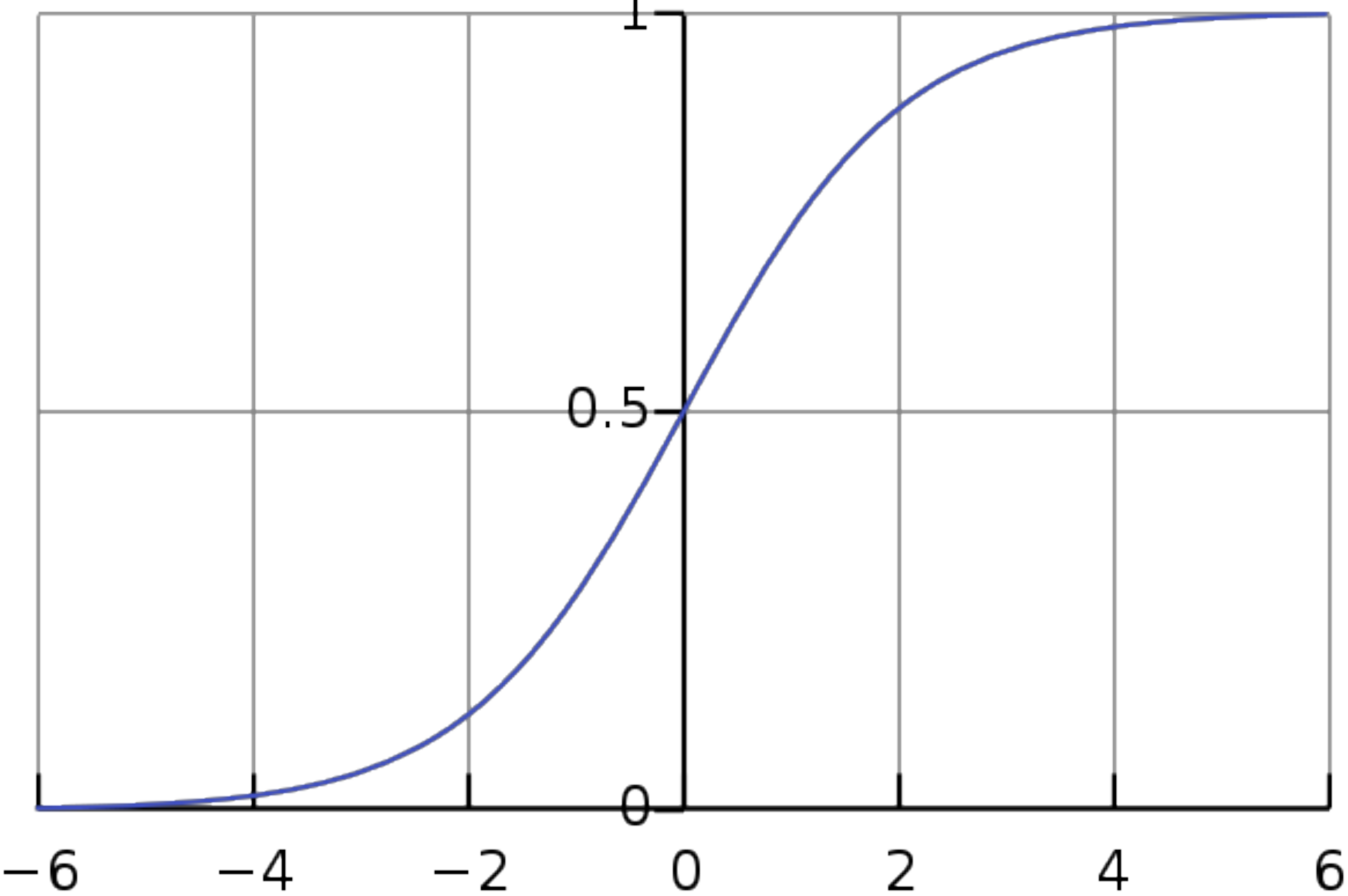
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$z = mx + c$

$\theta^T x = [m \;\; c]\begin{pmatrix} x \\ 1 \end{pmatrix}$

$\theta = \begin{bmatrix} m \\ c \end{bmatrix}$

Sigmoid function



Revised Setup

- If there are only two classes, transform, e.g.,
  orange => 1
  blue => 0
  to turn the classification problem into a regression problem

- Model:

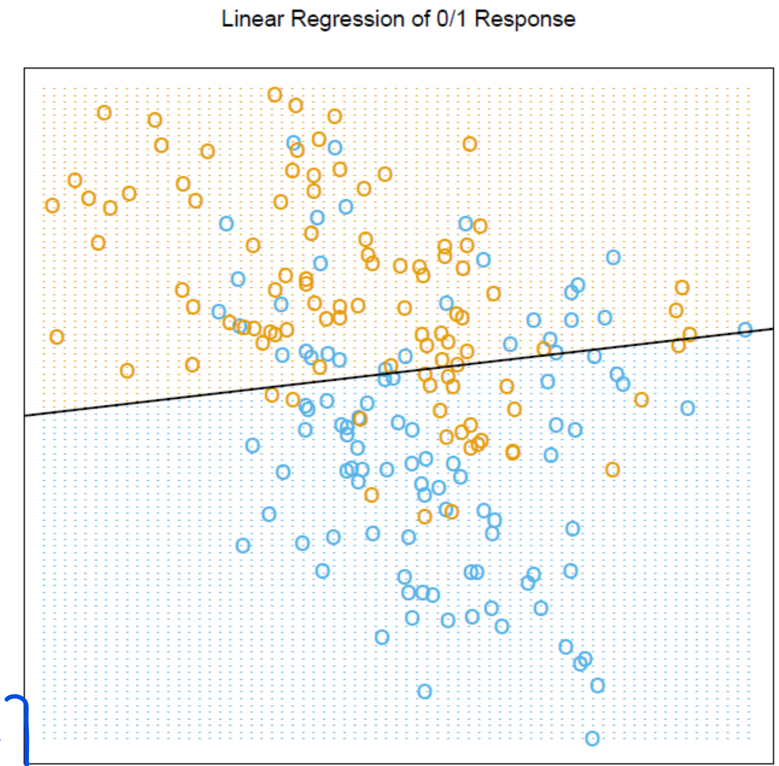
$$h_\theta(x) = \sigma(\theta^T x)$$

$$\theta = \begin{bmatrix} m \\ c \end{bmatrix}$$

$$x = \begin{bmatrix} x \\ 1 \end{bmatrix}$$

- Where

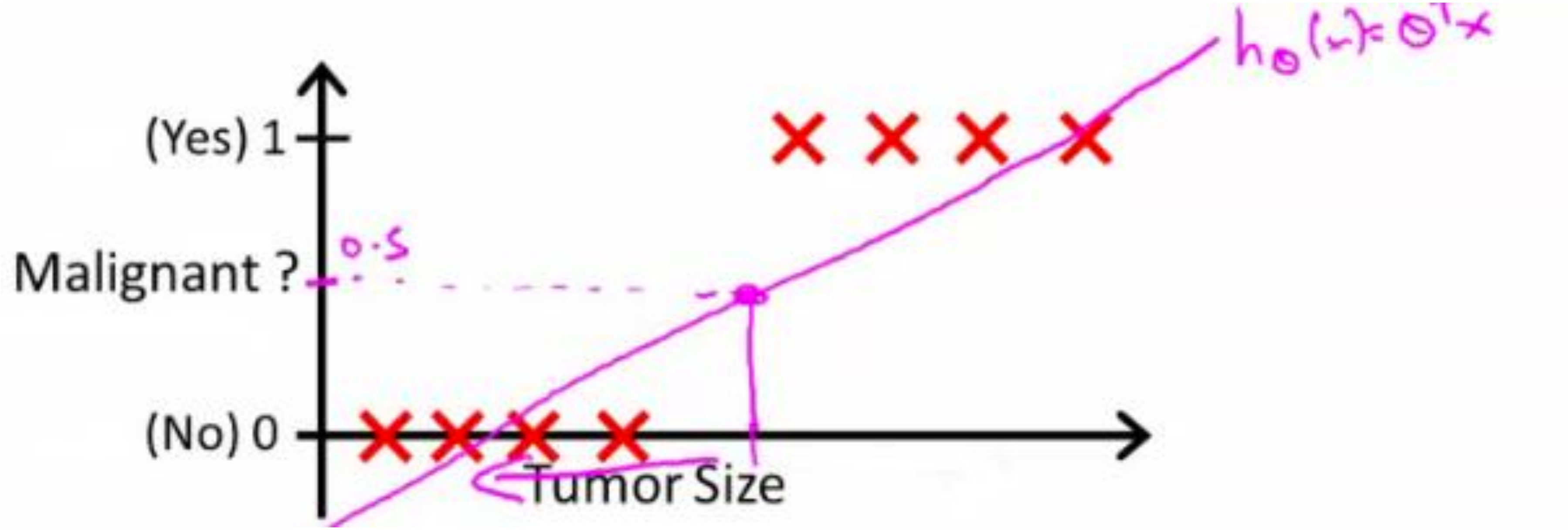
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



What is the equation of the decision boundary?

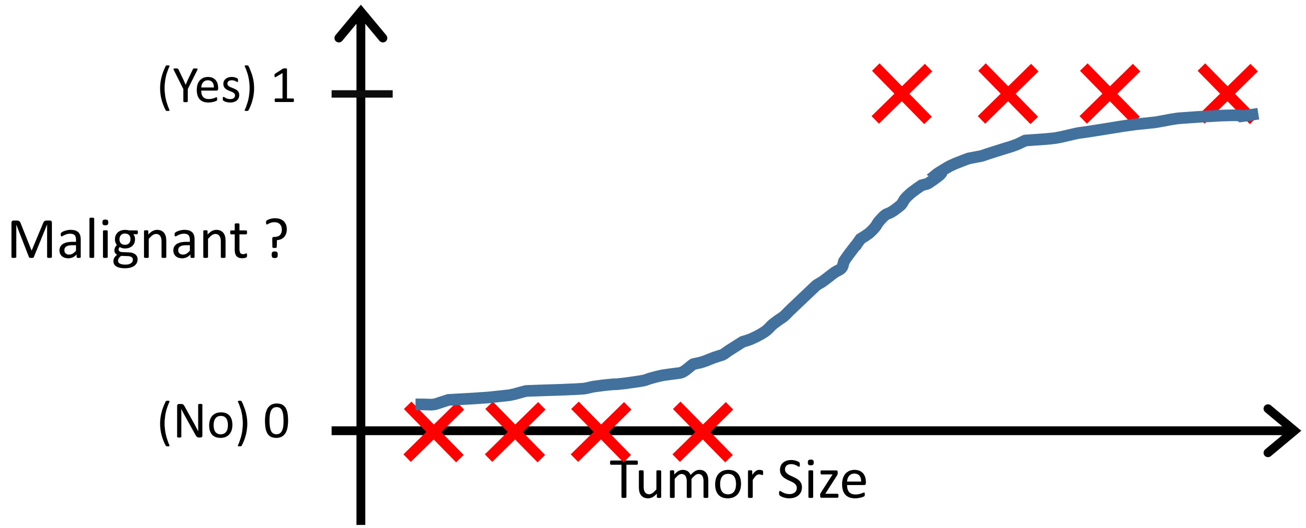
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2, \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

Reminder: linear prediction in 1D



Even with perfect classification,
Loss is still nonzero (and can be high!)

Example in 1D: applying the sigmoid



$y^i = \sigma(\theta_0 + \theta_1 x_1^i + \theta_2 x_2^i + \theta_3 x_3^i)$

Supervised Learning

What about the loss?

$y^i - \sigma(\theta^T x) = 0$

Features → x_1^i, x_2^i x_3^i y^i ↙ Label $\{0,1\}$

x_1^2 x_2^2 x_3^2 y^2

- Square Loss?

Label

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)} - \sigma\left(\theta^T \boldsymbol{x}^{(i)}\right)\right)^2$$

Non Linear Function

- On the board:
  - If $h_\theta(\boldsymbol{x}) = \sigma\left(\theta^T \boldsymbol{x}\right)$ is very close to 0 or 1, then the gradient of the loss is close to zero!
  - Why is that a problem?
  - Summary:

$$\nabla J(\theta) = \frac{1}{\mathrm{m}}\sum_{i=1}^{m}\left(y - \sigma\left(\theta^T x^{(i)}\right)\right)\sigma\left(\theta^T x^{(i)}\right)\left(1 - \sigma\left(\theta^T x^{(i)}\right)\right)x^{(i)}$$

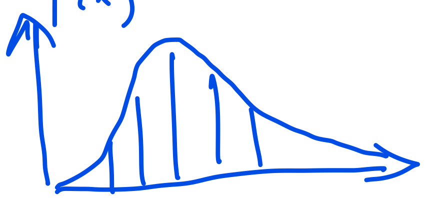
What loss should we use?

- We will use binary **Cross Entropy Loss**

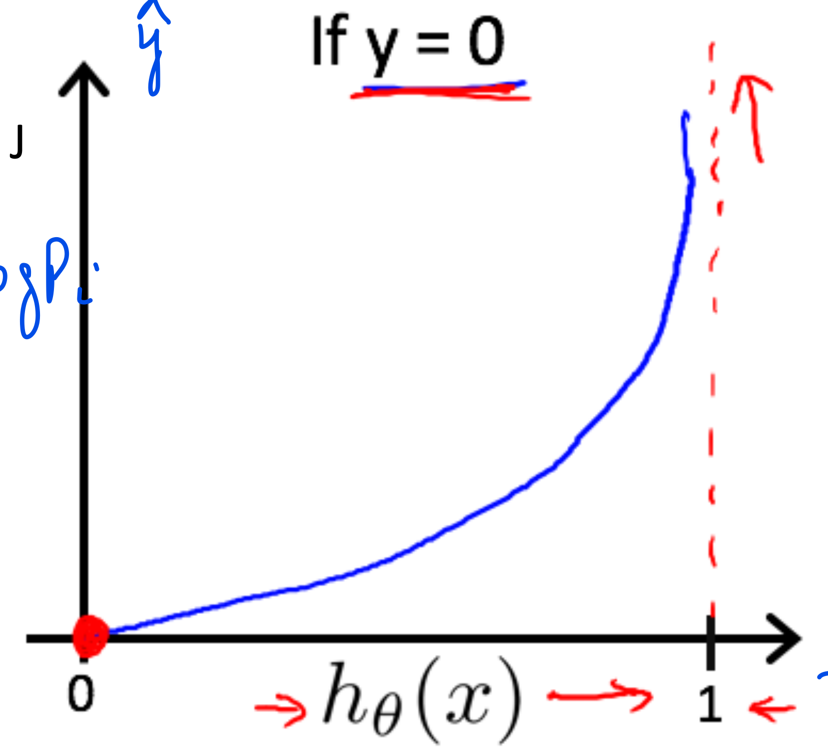
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m} -\underbrace{y^{(i)}}_{q}\log \underbrace{\sigma\left(\theta^T \boldsymbol{x}^{(i)}\right)}_{\hat{y}} \quad + \overbrace{(1-y^{(i)})}^{P(Lab=0)}\overbrace{\left(-\log\left(1-\sigma\left(\theta^T \boldsymbol{x}^{(i)}\right)\right)\right)}^{P(lab=0)}$$

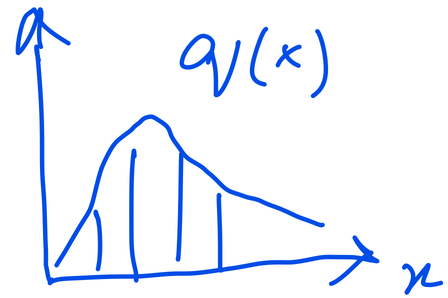
$y' = 1$

Entropy $= \sum_{i=1}^{m} -P_i \log P_i$



Prob Distribution



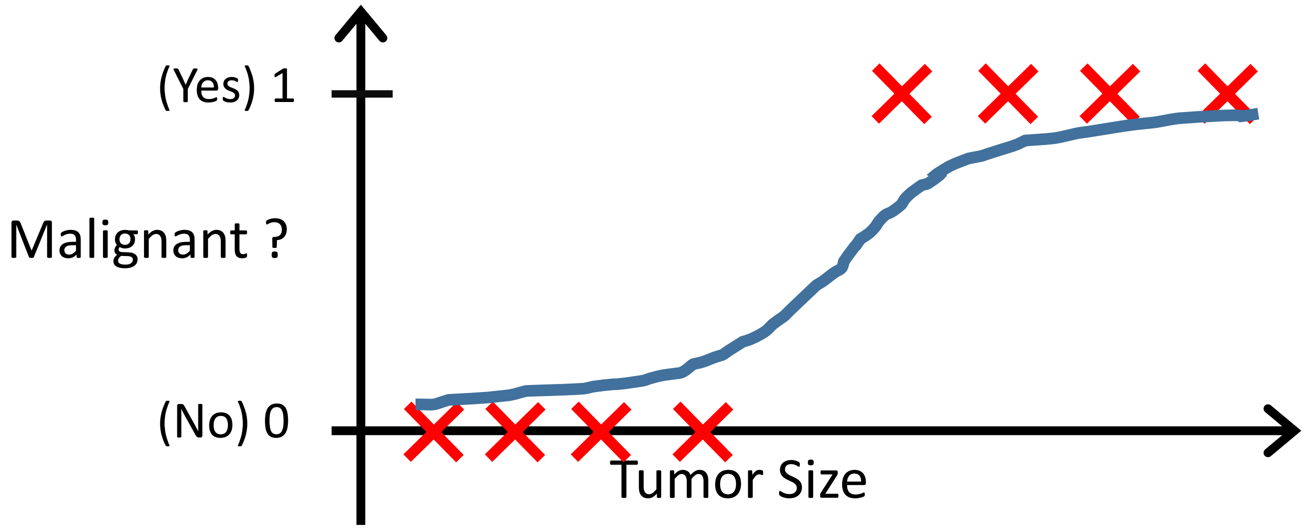


$-\sum_i P_i \log q_i$

Why Cross Entropy?

- Where does this come from?
- As in Linear Regression, we will use a probabilistic interpretation

Predictions look like probabilities



Logistic Regression

Logistic Regression

- Assume the data is generated according to

$$y^{(i)} = 1 \text{ with probability } \frac{1}{1+\exp(-\theta^T x^{(i)})}$$

$$y^{(i)} = 0 \text{ with probability } \frac{\exp(-\theta^T x^{(i)})}{1+\exp(-\theta^T x^{(i)})}$$

- This can be written concisely as:

odds

$$\log\left(\frac{P(y^{(i)}=1|x^{(i)},\theta)}{P(y^{(i)}=0|x^{(i)},\theta)}\right) = \log\left(\exp(\theta^T x^{(i)})\right) = \theta^T x^{(i)} > 0 \quad \text{Lab } 1$$

$$< 0 \quad \text{Lab } 0$$

(exercise)

Logistic Regression: Likelihood

- $P\left(y^{(i)} = 1 \middle| x^{(i)}, \theta\right) = \left(\frac{1}{1+\exp\left(-\theta^T x^{(i)}\right)}\right)^{y^{(i)}} \left(\frac{\exp\left(-\theta^T x^{(i)}\right)}{1+\exp\left(-\theta^T x^{(i)}\right)}\right)^{1-y^{(i)}}$

(just a trick that works because $y^{(i)}$ is either 1 or 0)

$\Sigma_1 \quad \Sigma_2 \quad \Sigma_3$

$P_1 \quad P_2 \quad P_3$

multiplication

- $P(y|x,\theta) = \Pi_{i=1}^{m} \left(\frac{1}{1+\exp\left(-\theta^T x^{(i)}\right)}\right)^{y^{(i)}} \left(\frac{\exp\left(-\theta^T x^{(i)}\right)}{1+\exp\left(-\theta^T x^{(i)}\right)}\right)^{1-y^{(i)}}$
- $\log P(y|x,\theta) =$
$\Sigma_{i=1}^{m} y^{(i)} \log\left(\frac{1}{1+\exp\left(-\theta^T x^{(i)}\right)}\right) + (1 - y^{(i)}) \log\left(\frac{\exp\left(-\theta^T x^{(i)}\right)}{1+\exp\left(-\theta^T x^{(i)}\right)}\right)$

Sum

$\sum y^i \log \sigma(\theta^T x) + (1 - y^i) \log(1 - \sigma(\theta^T x))$

Logistic Regression: Learning and Testing

- Learning: find the θ that maximizes the log-likelihood:

$$\sum_{i=1}^{m} y^{(i)} \log\left(\frac{1}{1+\exp(-\theta^T x^{(i)})}\right) + (1-y^{(i)}) \log\left(\frac{\exp(-\theta^T x^{(i)})}{1+\exp(-\theta^T x^{(i)})}\right)$$

- For x in the test set, compute

$$P(y=1|x,\theta) = \frac{1}{1+\exp(-\theta^T x)}$$

- Predict that $y = 1$if $P(y=1|x,\theta) > .5$

Logistic Regression: Decision Surface

- Predict $y = 1$ if $\frac{1}{1+\exp(-\theta^T x)} > 0.5$

$$\Leftrightarrow \qquad -\theta^T x < 0 \rightarrow \text{Lab} = 0$$

$$\Leftrightarrow \qquad \boxed{\theta^T x > 0} \rightarrow \text{Lab} = 1$$

- The decision surface is $\theta^T x = 0$, a hyperplane

$$\frac{1}{1+e^{-0.01}} > 0.5$$

$$\frac{1}{1+e^{0.01}} < 0.5$$

$$\frac{1}{1+e^{0}} = \frac{1}{2}$$

Logistic Regression

- Outputs the probability of the datapoint's belonging to a certain class:

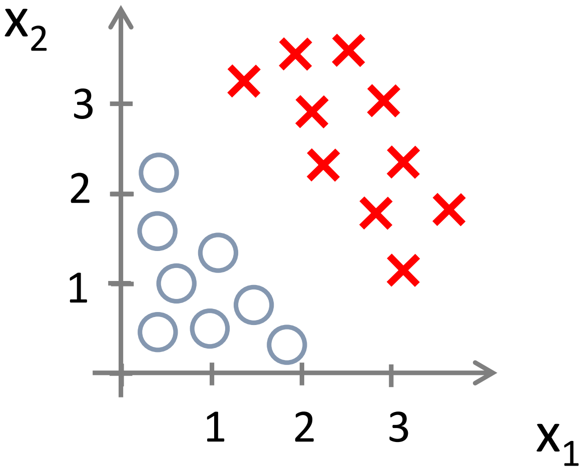
$$y^{(i)} = 1 \text{ with probability } \frac{1}{1+\exp(-\theta^T x^{(i)})}$$

$$y^{(i)} = 0 \text{ with probability } \frac{\exp(-\theta^T x^{(i)})}{1+\exp(-\theta^T x^{(i)})}$$

(compare with linear regression)

- Linear decision surface
- Probably the first thing you would try in a real-world setting for a classification task

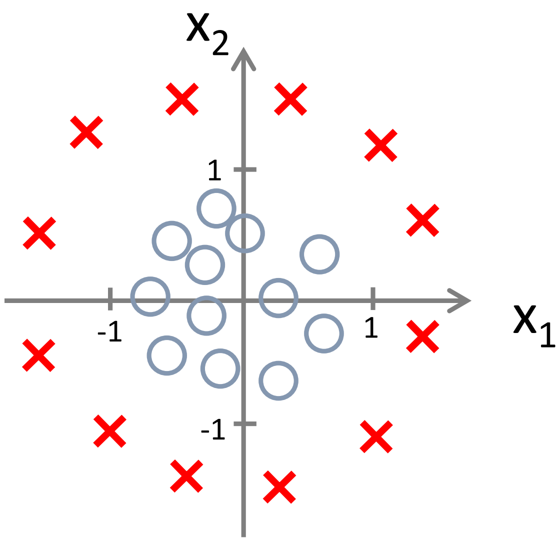
Decision boundary shapes



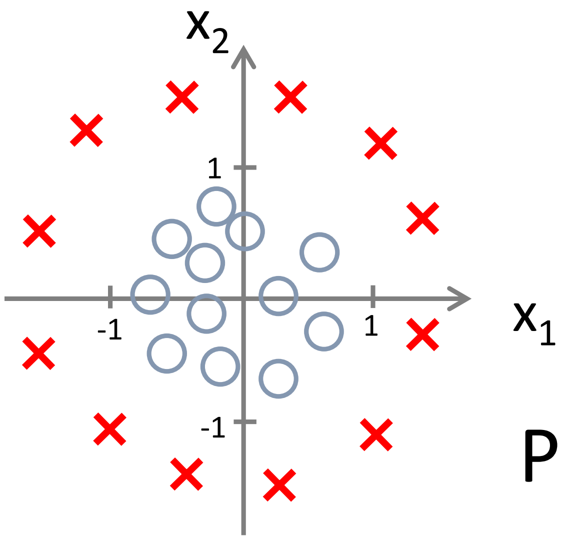
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Predict $y = 1$ if $-3 + x_1 + x_2 \geq 0$

Decision boundary shapes



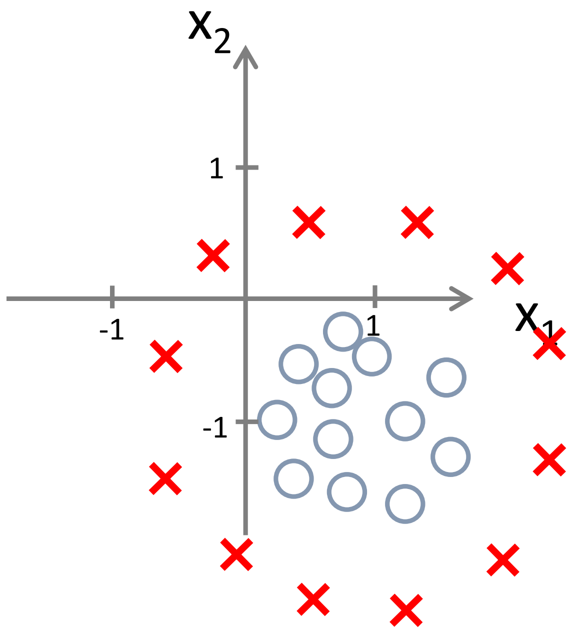
Decision boundary shapes



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Predict $y = 1$ if $-1 + x_1^2 + x_2^2 \geq 0$

What is the equation for a good decision boundary?



Multiclass Classification

Email foldering/tagging : Work, Friends, Family, Hobby

$y = 1$ $y = 2$ $y = 3$ $y = 4$

Features: x_1: 1 if "extension" is in the email, 0 otherwise

x_2: 1 if "dog" is in the email, 0 otherwise

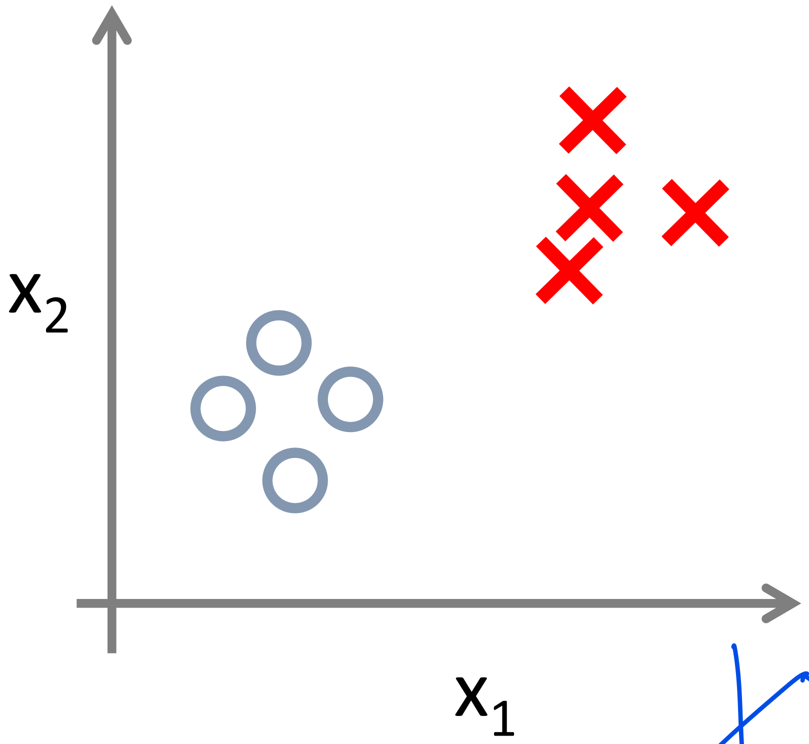
…

Medical diagrams: Not ill, Cold, Flu

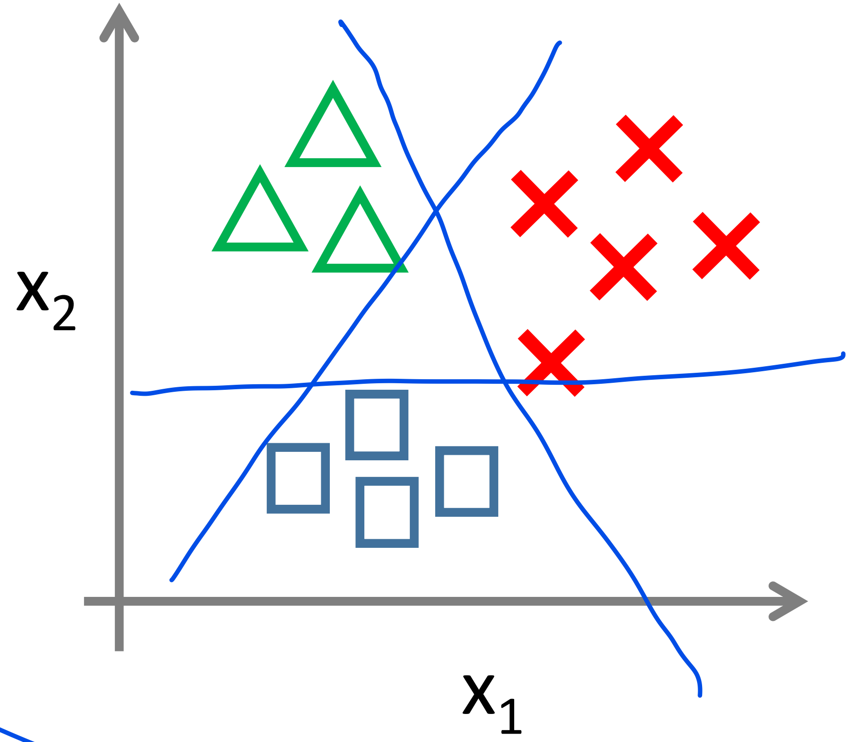
$y = 1$ $y = 2$ $y = 3$

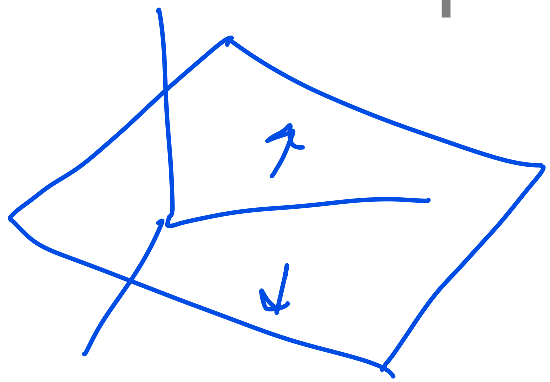
Features: temperature, cough presence, …

Binary classification:

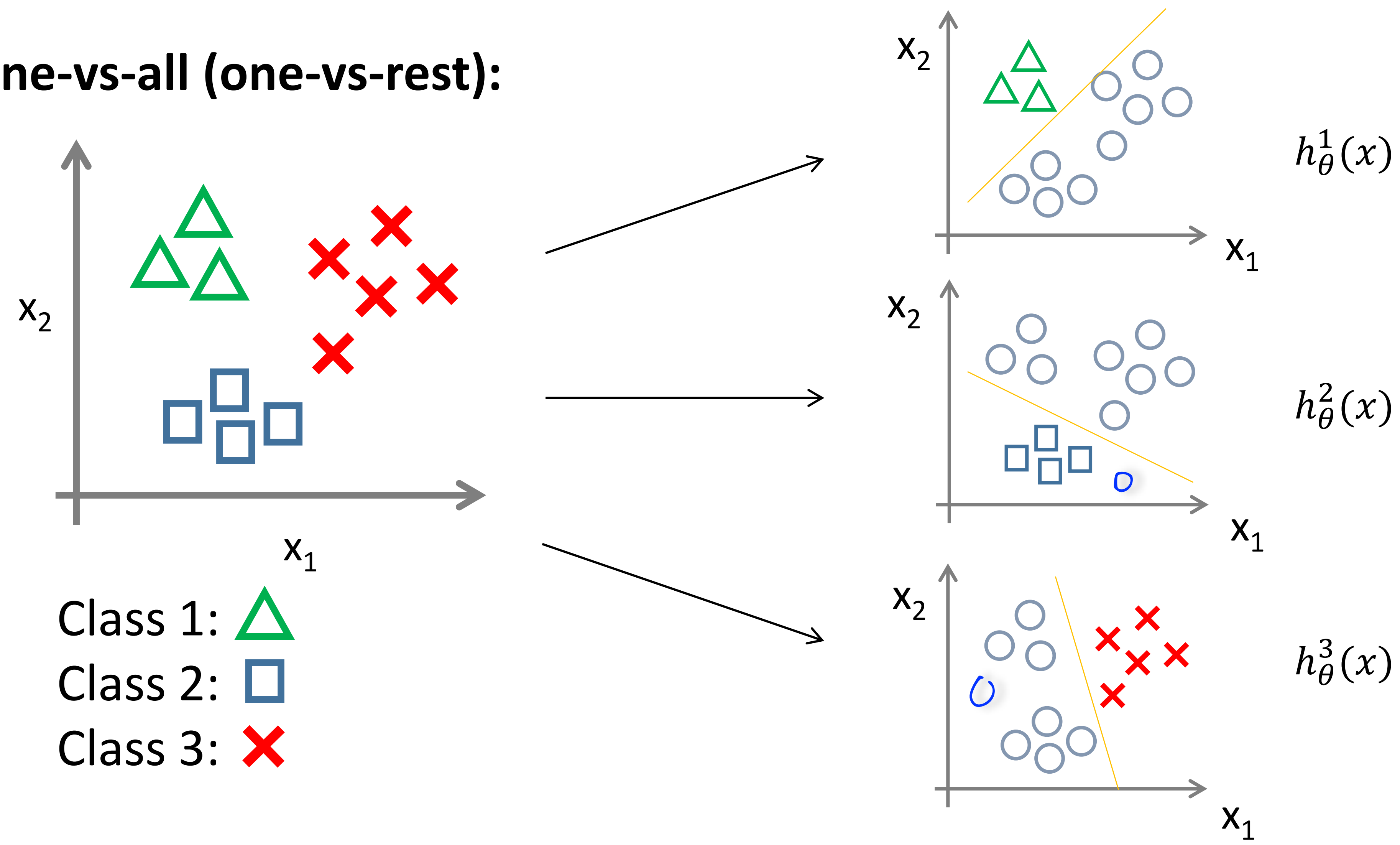


Multi-class classification:





One-vs-all (one-vs-rest):



Output the i such that $h_\theta^i(x)$ is the largest
(Idea: a large $h_\theta^i(x)$ means that the classifier is "sure")